

User's Guide for ContactCenters Simulation Library

SSJ Extensions API Specification

Version: March 4, 2014

ERIC BUIST

This is the API specification for the SSJ Extensions for contact center simulation. These extensions enhance statistical collecting with new high-level facilities and provide a framework to perform simulations based on batch means or independent replications. They also include a system to convert XML configuration files into Java parameter objects.

Contents

Overview	2
Package umontreal.iro.lecuyer.stat	3
TallyWithTimes	4
TallyWithMovingWindow	5
AccumulateWithTimes	7
Package umontreal.iro.lecuyer.randvar	9
RandomVariateGenWithShift	10
RandomVariateGenIntWithShift	11
ExpKernelDensityGen	12
Package umontreal.iro.lecuyer.xmlbind	13
JAXBParamsConverter	14
ArrayConverter	23
ParamReadHelper	27
DistributionCreationException	33
GeneratorCreationException	35
RemappingContentHandler	37
NamedInfo	38
SourceArray2D	40
SourceSubset2D	41
CSVSourceArray2D	42
ExcelSourceArray2D	43
DBSourceArray2D	44

Package umontreal.iro.lecuyer.xmlconfig	45
ParamReader	49
ParamReadException	59
Param	61
StorableParam	62
AbstractParam	63
TimeParam	68
SourceArray2D	72
CSVSourceArray2D	74
ExcelSourceArray2D	75
DBSourceArray2D	76
SourceSubset2D	78
ParamWithSourceArray	79
ArrayParam	82
ArrayParam2D	86
DBConnectionParam	90
PropertyParam	93
RandomVariateGenParam	95
DistributionCreationException	103
GeneratorCreationException	105
DOMUtils	107
 Package umontreal.iro.lecuyer.util	 113
ClassFinderWithBase	114
StringConvert	115
UnsupportedConversionException	123
DoubleFormatter	125
DoubleFormatterWithError	126
DefaultDoubleFormatter	127
DefaultDoubleFormatterWithError	128
LowMemoryNotifier	129
LowMemoryListener	132
ArrayUtil	133

FileUtil	143
ExceptionUtil	144
Pair	145
LineBreaker	147
LaTeXFormatter	149
LaTeXArrayFormatter	153
LaTeXObjectMatrixFormatter	158
ModifiableWorkbook	161
IntArray	163
Package umontreal.iro.lecuyer.collections	165
TransformingCollection	166
TransformingList	168
TransformingSet	169
TransformingMap	170
MergedCollection	172
MergedList	173
MergedSet	175
MergedMap	176
FilteredIterator	177
FilteredListIterator	179
ObjectTypeIterator	181
ObjectTypeListIterator	182
Matrix	183
AbstractMatrix	188
DenseMatrix	189
DescendingOrderComparator	190

Overview

The ContactCenters simulation library uses and extends the Stochastic Simulation in Java (SSJ) library [4] in addition to the Collections Tuned (Colt) library [2]. The package `umontreal.iro.lecuyer.stat` enhances the SSJ's basic statistics functionalities to allow more compact and easier statistical collecting of many performance measures. The package `umontreal.iro.lecuyer.simevents` extends the discrete-event simulation to support batch means simulation more conveniently. The package `umontreal.iro.lecuyer.xmlconfig` provides some general facilities for converting XML configuration files to complex parameter objects.

Package umontreal.iro.lecuyer.stat

Special tallies in line with the stat package of SSJ.

TallyWithTimes

```
package umontreal.iro.lecuyer.stat;
```

```
public class TallyWithTimes extends Tally
```

Nested class

```
    public static class Observation
```

TallyWithMovingWindow

Represents a tally with a moving window of fixed length. Usually, a tally counts the observations from the time it is initialized to the current time. This tally can be used to collect observations during a moving time window divided in P successive intervals of fixed duration d . This tally defines P internal tallies collecting period-specific observations. When an observation is given to this tally, it is added into the current tally which is changed by a simulation event happening every d time units. After the P th period, the first tally becomes the current tally again, and its observations are lost.

This tally can be used as a usual tally, except `start()` must be called at the simulation time corresponding to the beginning of the first collecting period.

```
package umontreal.iro.lecuyer.stat;
```

```
public class TallyWithMovingWindow extends Tally
```

Constructors

```
public TallyWithMovingWindow (boolean keepObs, int numCollectingPeriods,
                               double collectingPeriodDuration, double
                               endingTime)
```

Constructs a new tally with moving window using `numCollectingPeriods` of duration `collectingPeriodDuration`, and with simulation ending at `endingTime`.

Parameters

`keepObs` determines if the internal tallies can keep observations.

`numCollectingPeriods` the number of collecting periods.

`collectingPeriodDuration` the duration of a collecting period.

`endingTime` the ending time of the simulation.

```
public TallyWithMovingWindow (String name, boolean keepObs, int
                               numCollectingPeriods, double
                               collectingPeriodDuration, double endingTime)
```

Constructs a new tally with moving window with name `name`, using `numCollectingPeriods` of duration `collectingPeriodDuration`, and with simulation ending at `endingTime`.

Parameters

`name` the name of the tally.

`keepObs` determines if the internal tallies can keep observations.

`numCollectingPeriods` the number of collecting periods.

`collectingPeriodDuration` the duration of a collecting period.

`endingTime` the ending time of the simulation.

Methods

`public Tally getTally (int i)`

Returns the *i*th internal tally.

Parameter

i the index of the internal tally.

Returns the internal tally.

`public int getNumCollectingPeriods()`

Returns the number of collecting periods.

Returns the number of collecting periods.

`public double getCollectingPeriodDuration()`

Returns the duration of the collecting periods.

Returns the duration of the collecting periods.

`public void init()`

Initializes this tally as well as all internal tallies.

`public void start()`

Starts this tally with moving window by scheduling the first period-changing event. This method should be called at the simulation time corresponding to the first period.

`public void stop()`

Stops this tally with moving average by cancelling the currently scheduled period-changing event.

`public TallyWithMovingWindow clone()`

Clones this tally as well as the internal tallies.

`public double variance()`

Computes and returns the sample variance of the observations in the moving window. This method can be used only if the internal tallies can store observations.

AccumulateWithTimes

A subclass of `StatProbe`, for collecting statistics on a variable that evolves in time, with a piecewise-constant trajectory. Each time the variable changes its value, the method `update` must be called to inform the probe of the new value. The probe can be reinitialized by `init`. This class is similar to `Accumulate`, but it uses a user-specified times rather than simulation times.

```
package umontreal.iro.lecuyer.stat;

public class AccumulateWithTimes extends StatProbe
    implements Cloneable
```

Constructors

```
public AccumulateWithTimes()
```

Constructs a new `Accumulate` statistical probe and initializes it by invoking `init()`.

```
public AccumulateWithTimes (String name)
```

Construct and initializes a new `Accumulate` statistical probe with name `name` and initial time 0.

Methods

```
public void init()
```

Initializes the statistical collector and puts the current value of the corresponding variable to 0 at time 0. A call to `init` should normally be followed immediately by a call to `update` to give the value of the variable at the initialization time.

```
public void init (double time)
```

Similar to `init()`, but the initial time is given by `time`.

```
public void init (double time, double x)
```

Same as `init` followed by `update(time, x)`.

Parameter

`x` initial value of the probe

```
public void update (double time)
```

Updates the accumulator using the last value passed to `update`.

```
public void update (double time, double x)
```

Gives a new observation `(time, x)` to the statistical collector. If broadcasting to observers is activated for this object, this method will also transmit the new information to the registered observers by invoking the methods `notifyListeners`.

Parameters

time the time of the observation

x new observation given to the probe

public double average()

Returns the time-average since the last initialization to the last call to **update** (**double**).

public double getInitTime()

Returns the initialization time for this object. This is the simulation time when **init** was called for the last time.

Returns the initialization time for this object

public double getLastTime()

Returns the last update time for this object. This is the simulation time of the last call to **update** or the initialization time if **update** was never called after **init**.

Returns the last update time of this object

public double getLastValue()

Returns the value passed to this probe by the last call to its **update** method (or the initial value if **update(double,double)** was never called after **init**).

Returns the last update value for this object

public AccumulateWithTimes clone()

Clone this object.

Package `umontreal.iro.lecuyer.randvar`

Extends the random variate generation facilities of SSJ.

RandomVariateGenWithShift

Random variate generator applying a shift to the generated values. This generator uses another random variate generator to generate variates. For each variate v , the `nextDouble()` method of this generator returns $v - \ell$, where $\ell \in \mathbb{R}$ is a constant, user-defined shift.

```
packageumontreal.iro.lecuyer.randvar;  
  
public class RandomVariateGenWithShift extends RandomVariateGen
```

Constructor

```
public RandomVariateGenWithShift (RandomVariateGen gen, double shift)  
    Constructs a new random variate generator with underlying generator gen, and shift shift.
```

Parameters

`gen` the generator being used.

`shift` the shift ℓ .

Methods

```
public double getShift()  
    Returns the current value of the shift  $\ell$ .
```

Returns the current value of the shift.

```
public void setShift (double shift)  
    Sets the current value of the shift to shift.
```

Parameter

`shift` the new value of the shift.

```
public RandomVariateGen getRandomVariateGenerator()  
    Returns the random variate generator being used by this object.
```

Returns the associated random variate generator.

RandomVariateGenIntWithShift

Random variate generator applying a shift to the generated values. This generator uses another random variate generator to generate variates. For each variate v , the `nextInt()` method of this generator returns $v - \ell$, where $\ell \in \mathbb{N}$ is a constant, user-defined shift.

```
package umontreal.iro.lecuyer.randvar;
```

```
public class RandomVariateGenIntWithShift extends RandomVariateGenInt
```

Constructor

```
public RandomVariateGenIntWithShift (RandomVariateGenInt gen, int shift)
```

Constructs a new random variate generator with underlying generator `gen`, and shift `shift`.

Parameters

`gen` the generator being used.

`shift` the shift ℓ .

Methods

```
public int getShift()
```

Returns the current value of the shift ℓ .

Returns the current value of the shift.

```
public void setShift (int shift)
```

Sets the current value of the shift to `shift`.

Parameter

`shift` the new value of the shift.

```
public RandomVariateGenInt getRandomVariateGenerator()
```

Returns the random variate generator being used by this object.

Returns the associated random variate generator.

ExpKernelDensityGen

Exponential kernel density random variate generator. This random variate generator uses the empirical distribution of log-service times to generate service times. It uses a gaussian kernel with positive reflection and applies the exponential function on every generated variate.

```
package umontreal.iro.lecuyer.randvar;  
  
public class ExpKernelDensityGen extends KernelDensityGen
```

Constructor

```
public ExpKernelDensityGen (RandomStream stream, EmpiricalDist dist)
```

Constructs a new exponential kernel density generator from the empirical distribution `dist` and the random stream `stream`. This constructor calls the `KernelDensityGen.setPositiveReflection (boolean)` public method.

Parameters

`stream` the random number stream to generate the uniforms.

`dist` the empirical distribution for the log-service times.

Package `umontreal.iro.lecuyer.xmlbind`

Extends the JAXB framework with new classes to simplify the process of marshalling and unmarshalling objects with validation using XML Schema. The JAXB framework, implemented in package `javax.xml.bind`, can be used to map elements of an XML document to Java objects. The binding compiler `xjc` is used to derive Java classes from a XML Schema, and the JAXB runtime to unmarshal XML contents into Java objects, or marshal objects back to XML.

JAXB-derived classes do not contain any behavior; these are simple data classes. As a result, work is often required to transform objects from these derived classes into other more useful objects. For example, an helper method is necessary to turn objects containing parameters for a random variate generator into a true generator with an associated implementation generating numbers. This package defines classes containing such helper methods. The class `ParamReadHelper` provides methods for maps of properties, random variate generators, and database connections. The class `ArrayConverter` provides helper methods for 2D arrays of various common types. It also defines an abstract converter class, `JAXBParamsConverter`, which encapsulates a JAXB context and mechanisms to read and write XML files with validation using a Schema. See the documentation of package `javax.xml.bind`, in the Java API specification, for more information about JAXB.

JAXBParamsConverter

Convenience base class to marshal and unmarshal objects of a specific class using JAXB. When using JAXB directly, one must create a context, use that context to get an unmarshaller or a marshaller, convert the unmarshalled JAXB element into a value object, or wrap a value object into a JAXB element for marshalling. This class can help in performing these tasks. It also manages the association of a schema to unmarshallers, and marshallers in order to perform validation. It also implements a mechanism to replace JAXB-generated namespace prefixes by user-defined prefixes while marshalling. This does not affect the validity of marshalled XML, but it can increase the readability of the output files.

The methods `unmarshal (File)`, and `marshal (T, File)` can be used to marshal and unmarshal objects. Alternatively, simple console applications may use `unmarshalOrExit (File)`, and `marshalOrExit (T, File)` which prints a detailed message and exits in case of an error while the regular methods throw exceptions.

This class must be extended with a specific type of parameter object to be used. This type can be any class derived by the JAXB-provided `xjc` compiler from a XML Schema. It must be passed as a type parameter when creating this class. More specifically, any concrete subclass must implement the `getContext()` method used to create the JAXB context as well as the `getSchema()` method to get the schema object used for validation.

For example, suppose that we have a Schema describing parameters of call centers, with a root element of type `CallCenterParams`. The JAXB compiler produces a class named `CallCenterParams`. The user-defined converter class then extends the base class `JAXBParamsConverter<CallCenterParams>`, and provides an implementation for the two mandatory methods.

Type parameter

T the type of objects processed by the converter.

```
package umontreal.iro.lecuyer.xmlbind;  
  
public abstract class JAXBParamsConverter<T>
```

Constructor

```
public JAXBParamsConverter (Class<T> objClass)
```

Constructs a new converter manipulating objects of class `objClass`.

Parameter

`objClass` the class of objects.

Methods

`public abstract JAXBContext getContext() throws JAXBException`

Constructs and returns the JAXB context used to read parameters. This method should create and return a static instance of the JAXB context since the creation of the context is costly.

Any concrete subclass should define a static field of type `JAXBContext`. If the field is non-null, the method returns its value. Otherwise, it initializes the field using `JAXBContext.newInstance (String)`, and returns the resulting context. The arguments given to `JAXB` depends on the JAXB-derived class associated with the concrete subclass.

Returns the JAXB context to be used.

Throws

`JAXBException` if an error occurs while creating the JAXB context.

`public abstract Schema getSchema() throws SAXException`

Constructs and returns a schema for the document type represented by a concrete subclass. If no schema is used, this method should return `null`. If an error occurs when reading or parsing the schema, this method should throw a SAX exception.

It is recommended to use `SchemaFactory` to create the `Schema` object, and to store it in a static variable for future use, because loading and parsing the schema might be costly. If the schema is stored at the same location as class files, `Class.getResourceAsStream (String)` can be used to obtain a stream for the schema.

Returns the schema object, or `null`.

Throws

`SAXException` if an error occurred during reading or parsing.

`public abstract Map<String, String> getNamespacePrefixes
()`

Returns a map associating prefixes with namespace URI. Each key of the returned map corresponds to a prefix while each value denotes a URI. This map is used while marshalling in order to assign meaningful prefixes to namespace URI rather than the default prefixes. An empty map can be used to disable namespace prefix mapping.

When using namespaces, each XML element and attribute can be qualified with a namespace URI which is referred to, in the XML document, using a prefix. These prefixes, which are not unique in contrast with URIs, can be chosen arbitrarily, but they should be human-readable for clearer documents. However, by default, JAXB generates its own prefix each time it finds a new namespace URI during marshalling; there is no standard way to impose prefixes. This map can be used to bind user-defined prefixes to the URIs used by the XML document. When this method returns a non-empty map, the marshalling mechanism of this class uses the `RemappingContentHandler` to perform the namespace prefix mapping in a way independent from the JAXB implementation.

Returns the map associating namespace prefixes to URIs.

```
public Schema readSchema()
```

Tries to use `getSchema()` to obtain the schema. If this method throws a SAX exception, the exception is logged, validation is disabled, and this method returns `null`. Otherwise, the schema object is returned. This can be used to work around bugs in some Java implementations preventing valid schemas to be read; parsing will then continue, without validation.

Returns the schema object, or `null`.

```
public void initUnmarshaller (Unmarshaller um) throws JAXBException
```

Initializes the unmarshaller before it is used by this object. By default, this method installs the event handler returned by `getEventHandler()`. It can be overridden to perform custom initialization such as setting properties.

Parameter

`um` the unmarshaller being initialized.

```
public void initMarshaller (Marshaller m) throws JAXBException
```

Initializes the marshaller before it is used by this object. By default, this method associates the event handler returned by `getEventHandler()` to the marshaller, and activates the `Marshaller.JAXB_FORMATTED_OUTPUT` property. It can be overridden to perform custom initialization such as setting properties.

Parameter

`m` the marshaller being initialized.

```
public boolean isValidating()
```

Determines if this converter is validating unmarshalled and marshalled instances. The default value is `true`, which activates validation. However, validation is disabled if no schema is specified, or if the schema cannot be initialized.

Returns the status of the validating indicator.

```
public void setValidating (boolean validating)
```

Sets the status of the validating indicator to `validating`.

Parameter

`validating` the new value of the indicator.

```
public ValidationEventHandler getEventHandler()
```

Returns the validation event handler associated with marshallers and unmarshallers used by this converter. This returns `null` if no handler was set (the default).

Returns the validation event handler, or `null`.

```
public void setEventHandler (ValidationEventHandler handler)
```

Sets the validation event handler to `handler`.

Parameter

`handler` the new validation event handler.

`public T getValue (Object jaxbElement) throws JAXBException`

Converts the JAXB element returned by an unmarshaller into an instance of class `T`. This uses the `JAXBIntrospector.getValue (Object)` to obtain the object, and casts it to an instance of class `T`.

Parameter

`jaxbElement` the JAXB element.

Returns the cast object.

Throws

`JAXBException` if an error occurs during conversion.

`public Object getJAXBObject (Object value) throws JAXBException`

Constructs and returns a JAXB element with `value` as a value. The returned object can be given to a JAXB `Marshaller` in order to be serialized to XML.

This method first uses `JAXBIntrospector.isElement (Object)` to test if the given value is an element, and returns the value if the test succeeds. Otherwise, this method looks for an `ObjectFactory` class in the package of the class of the given value. If such a factory is found, the method searches a factory method in this factory taking an object corresponding to the type of the given value, and calls this method to get a JAXB element encapsulating the value.

Parameter

`value` the value to encapsulate.

Returns the JAXB element corresponding to the value.

Throws

`JAXBException` if an error occurs during the process.

`public T unmarshal (File file) throws JAXBException`

Unmarshals the given input file to an object.

This method first calls `getContext()` to obtain a JAXB context. It then uses the obtained JAXB context to create an unmarshaller, sets its schema to the value returned by `readSchema()`, and initializes it by calling `initUnmarshaller (Unmarshaller)`. It then uses the unmarshaller to process the given file, and calls `getValue (Object)` to obtain the value of the resulting JAXB element.

Parameter

`file` the input file.

Returns the unmarshalled object.

Throws

JAXBException if an error occurs during unmarshalling.

`public T unmarshal (URL url) throws JAXBException`

Unmarshals the given URL into an object, and returns the constructed object. This is similar to `unmarshal (File)`, for a URL.

Parameter

`url` the URL of the input.

Returns the output object.

Throws

JAXBException if an error occurs during unmarshalling.

`public T unmarshalGZipped (File file) throws JAXBException`

Reads the given input file as a GZipped file, using `GZIPInputStream`, and unmarshals the uncompressed XML data. If the given file is not in GZIP format, this method calls `unmarshal (File)` to unmarshal a plain text file.

Parameter

`file` the file to be read.

Returns the unmarshalled object.

Throws

JAXBException if an error occurs during file reading.

`public T unmarshalGZipped (URL url) throws JAXBException`

Similar to `unmarshalGZipped (File)`, for a URL instead of a file.

Parameter

`url` the URL pointing to the XML data.

Returns the unmarshalled object.

Throws

JAXBException if an error occurs during the unmarshalling process.

`public T unmarshal (Node node) throws JAXBException`

Unmarshals the given node into a JAXB object, and returns the constructed object. This is similar to `unmarshal (File)`, for a DOM node.

Parameter

`node` the DOM node to be unmarshalled.

Returns the resulting object.

Throws

`JAXBException` if an error occurs during unmarshalling.

```
public T unmarshal (Source source) throws JAXBException
```

Unmarshals the given source into an object, and returns the constructed object. This is similar to `unmarshal (File)`, for a source.

Parameter

`source` the source to be unmarshalled.

Returns the constructed object.

Throws

`JAXBException` if an error occurs during unmarshalling.

```
public T unmarshalOrExit (File file)
```

Unmarshals the given file using `unmarshalGZipped (File)`, but if an error occurs, messages are printed on the standard error output, and the method exits the VM using `System.exit (int)`. This method is intended to be used in `main` methods of command-line programs.

Parameter

`file` the input file.

Returns the parameter object.

```
public void marshal (T object, ContentHandler handler1) throws  
                    JAXBException
```

Marshals the given object by generating SAX events and sending them to the given content handler.

This method calls `getJAXBObject (Object)` on the given object to convert it into a JAXB object. It then creates a marshaller using the context returned by `getContext()`, sets its schema to the value returned by `readSchema()`, initializes it with `initMarshaller (Marshaller)`, and gives it the JAXB object obtained from the given value. If `getNamespacePrefixes()` returns a non-empty map, the supplied handler is wrapped around a `RemappingContentHandler` in order to take namespace prefixes into account, and wrapped handler is passed to the marshaller. Otherwise, the content handler is passed directly to the marshaller.

Parameters

`object` the object to marshal.

`handler1` the content handler.

Throws

`JAXBException` if an exception occurs during marshalling.

```
public void marshal (T object, Result res) throws JAXBException
```

Marshals the given object `object` to the target set by `res`.

This method calls `getJAXBObject (Object)` to turn the given object into a JAXB object. If `getNamespacePrefixes()` returns an empty map, this method creates a marshaller the same way as `marshal (T, ContentHandler)`, and uses it directly to marshal the JAXB object. Otherwise, it creates a `TransformerHandler`, and gives it to `marshal (T, ContentHandler)`.

Parameters

`object` the object to marshal.

`res` the result object.

Throws

`JAXBException` if an error occurs during marshalling.

```
public void marshal (T object, File res) throws JAXBException
```

Marshals the given value object into the given output file. This creates a new `StreamResult` with the given file object, and calls `marshal (T, Result)` to performing marshalling.

Parameters

`object` the value to marshal.

`res` the output file.

Throws

`JAXBException` if an error occurs during marshalling.

```
public void marshalAndGZip (T object, File res) throws JAXBException
```

Marshals the given object to the given file, and gzips the marshalled contents.

Parameters

`object` the object to be marshalled.

`res` the target file.

Throws

`JAXBException` if an error occurs during marshalling.

```
public void marshal (T object, Node node) throws JAXBException
```

Similar to `marshal (T, File)`, for a DOM node.

Parameters

`object` the value object to marshal.

`node` the output node.

Throws

`JAXBException` if an error occurs during marshalling.

```
public void marshalOrExit (T object, File file)
```

Marshals the object `object` to the file `file`, using the `marshal (T, File)` method, but if an error occurs, this method prints messages on the standard error output, and exits the VM using `System.exit (int)`. This method is intended to be used in `main` methods of command-line programs.

Parameters

`object` the object to marshal.

`file` the output file.

```
public void marshalAndGZipOrExit (T object, File file)
```

Similar to `marshalOrExit (T, File)`, except that the method `marshalAndGZip (T, File)` is called instead of `marshal (T, File)`.

Parameters

`object` the object to be marshalled.

`file` the output file.

```
public static boolean hasWarnings (ValidationEvent... evs)
```

Determines if the given list of validation events contains at least one event representing a warning. This method tests each event of the given list for their severity returned by `ValidationEvent.getSeverity()`, and returns `true` as soon as one event with severity `ValidationEvent.WARNING` is found. This returns `false` if the list is empty, or if it contains only events representing errors or fatal errors.

Parameter

`evs` the tested list.

Returns `true` if and only if the given list contains at least one warning event.

```
public static boolean hasErrors (ValidationEvent... evs)
```

Similar to `hasWarnings (ValidationEvent...)` for errors instead of warnings.

Parameter

`evs` the tested list.

Returns `true` if and only if the given list contains at least one error event.

```
public static boolean hasFatalErrors (ValidationEvent... evs)
```

Similar to `hasWarnings (ValidationEvent...)` for fatal errors instead of warnings.

Parameter

`evs` the tested list.

Returns `true` if and only if the given list contains at least one fatal error event.

```
public static String validationEventsToString (ValidationEvent... evs)
```

Formats and returns a string containing a description for each validation event in the given list. For each event, this method formats a string using `validationEventToString (ValidationEvent)`, and separates each event with two newlines.

Parameter

`evs` the list of validation events.

Returns the formatted string.

```
public static String validationEventToString (ValidationEvent ev)
```

Constructs and returns a string representing the validation event `ev`. This string contains the severity (warning, error, or fatal error) of the event, and its descriptive message. It also contains the result of `locatorToString (ValidationEventLocator)` which converts the location information into a string.

Parameter

`ev` the validation event being formatted.

Returns the formatted string.

```
public static String locatorToString (ValidationEventLocator locator)
```

Formats the given locator `locator` into a string. If the locator specifies a URL, this method returns the URL as well as the line and column. If it contains an object, it returns the result of its `toString` method. If it specifies a DOM node, the node is formatted to an XPath-like expression. Otherwise, `Unknown location` is returned.

Parameter

`locator`

Returns the string representation of the locator.

ArrayConverter

Provides helper methods to convert 2D arrays read by JAXB to the Java's more natural representation of 2D arrays, namely arrays of arrays. For example, `DoubleArray` can be used to read a 2D array in an XML file. It defines a `DoubleArray.getRows()` method returning a list of `DoubleArray.Row` instances representing rows. Each row is defined by a list of values, and a repeat count. This class defines the `unmarshalArray (DoubleArray)` method converting such an array object to a more natural `double[] []` 2D array. It also provides the `marshalArray (double[] [])` which does the inverse operation. Other similar methods are provided for arrays of integers, and arrays of durations.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class ArrayConverter
```

Methods

```
protected static <T> T[] [] unmarshalArray (Class<T> componentClass, List<?
                                         > rows)
```

Converts the list of arrays `rows` into a Java 2D array. JAXB objects representing 2D arrays provide a method returning a list of elements, each element being an object of an element-dependent class with no common base class. This method accepts a list of such objects, and uses Reflection as follows to get a more natural 2D array. For each element of the given list, it looks for `getRepeat` and `getValue` methods to obtain the number of times the inner array must be repeated in the 2D array, and the values composing the inner array, respectively. This method verifies that `getRepeat` returns an `int` or an `Integer`, and that `getValue` returns a `List`. It is assumed that the list returned by `getValue` contains instances of the class represented by `componentClass`. This method is intended to be called by front-end methods public such as `unmarshalArray (DoubleArray)`.

Type parameter

T the type of components in the 2D array.

Parameters

componentClass the component class of the 2D array.

rows the list of row elements.

Returns the resulting 2D array.

```
public static double[] [] unmarshalArray (DoubleArray array)
```

Unmarshals a 2D array JAXB object into a Java 2D array of double-precision values.

Parameter

array the 2D array represented as a JAXB object.

Returns the Java 2D array.

```
public static boolean[] [] unmarshalArray (BooleanArray array)
```

Unmarshals a 2D array JAXB object into a Java 2D array of boolean values.

Parameter

array the 2D array represented as a JAXB object.

Returns the Java 2D array.

```
public static int[] [] unmarshalArray (IntArray array)
```

Unmarshals a 2D array JAXB object into a Java 2D array of integers.

Parameter

array the 2D array represented as a JAXB object.

Returns the Java 2D array.

```
public static Duration[] [] unmarshalArray (DurationArray array)
```

Unmarshals a 2D array JAXB object into a Java 2D array of time durations. Note that time durations can be converted to times in milliseconds using `Duration.getTimeInMillis (Date)`.

Parameter

array the 2D array represented as a JAXB object.

Returns the Java 2D array.

```
public static Duration[] [] unmarshalArray (NonNegativeDurationArray array)
```

Similar to `unmarshalArray (DurationArray)` for non-negative durations.

Parameter

array the 2D array represented as a JAXB object.

Returns the Java 2D array.

```
protected static <T> List<?> marshalArray (RowFactory<T> factory, T[] []  
                                             array)
```

Uses the given row factory to convert the specified Java 2D array into a list intended to be associated with the JAXB representation of a 2D array. For each array in **array**, this method creates a list and passes it to the row factory **factory**. The row objects corresponding to the inner arrays are then regrouped into a list which is returned. This method is intended to be used by `marshalArray (double[] [])`.

Type parameter

T the class of the components in the array.

Parameters

factory the row factory.

array the Java 2D array.

Returns the list to be used in the JAXB representation of the array.

```
public static BooleanArray marshalArray (boolean[] [] array)
```

Marshals a Java 2D array of boolean values into an object that can be serialized to XML by JAXB.

Parameter

array the input Java 2D array.

Returns the output object for JAXB.

```
public static DoubleArray marshalArray (double[] [] array)
```

Marshals a Java 2D array of double-precision values into an object that can be serialized to XML by JAXB.

Parameter

array the input Java 2D array.

Returns the output object for JAXB.

```
public static IntArray marshalArray (int[] [] array)
```

Marshals a Java 2D array of integers into an object that can be serialized to XML by JAXB.

Parameter

array the input Java 2D array.

Returns the output object for JAXB.

```
public static DurationArray marshalArray (Duration[] [] array)
```

Marshals a Java 2D array of durations into an object that can be serialized to XML by JAXB. Duration objects can be created using `DatatypeFactory`.

Parameter

array the input Java 2D array.

Returns the output object for JAXB.

```
public static NonNegativeDurationArray marshalArrayNonNegative  
(Duration[] [] array)
```

Similar to `marshalArray (Duration[] [])`, for non-negative durations only.

Parameter

array the input Java 2D array.

Returns the output object for JAXB.

```
public static double[] unmarshalArray (List<Double> list)
```

Converts a list containing double-precision values wrapped into objects of class `Double` to an array of double-precision values.

Parameter

`list` the list of wrapped values.

Returns the array of values.

```
public static List<Double> marshalArray (double[] array)
```

Converts an array of double-precision values to a list containing values wrapped into objects of class `Double`.

Parameter

`array` the array of values.

Returns the list of values.

Nested class

```
protected static interface RowFactory<T>
```

Represents a factory object for creating rows when marshalling a 2D array to JAXB object.

Type parameter

`T` the type of components in the 2D array.

Method

```
public Object createRow (List<T> values)
```

Constructs and returns a new row from the list of values.

Parameter

`values` the list of values.

Returns the new row object.

ParamReadHelper

Defines methods that can be used to construct Java objects from some parameter objects whose classes are derived by the JAXB binding compiler. Objects from JAXB-derived classes have behavior; they contain data only. This class provides methods to convert some complex parameter objects representing maps of properties, database connections, and probability distributions to more useful Java and SSJ objects. It complements the `ArrayConverter` class which is specialized for 2D arrays.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class ParamReadHelper
```

Methods

```
public static Map<String, Object> unmarshalProperties (PropertiesParams  
                                                    prop)
```

Constructs and returns a map containing the properties stored into the given `PropertiesParams` object `prop` which can be considered as a list of properties. For each `AbstractProperty` stored in `prop`, this method adds an entry to the returned map. The name of the new entry is the name of the property while the value of the entry is the value of the property. The `null` value is used for properties with no values. The type of the value is determined by the concrete subclass of `AbstractProperty`. The supported types are `Boolean`, `String`, `Integer`, `Double`, `Duration`, `Date`, and arrays of these preceding types except `Integer[]`, `Double[]`, and `Date[]`. Arrays of numbers are converted to `int[]`, and `double[]` instead.

This method returns an empty map if `prop` is `null`.

Parameter

`prop` the properties in the XML file.

Returns the Java Properties object.

Throws

`IllegalArgumentException` if more than one property with the same name was found.

```
public static Properties getProperties (Map<String, Object> prop)
```

Converts the given map of properties to a Java `Properties` object. This method accepts a map that is usually created by `unmarshalProperties (PropertiesParams)`, and turns it into a `Properties` object by converting each of its non-`null` values to strings, with the `Object.toString()` method. A `null` reference as a property value results in a `null` reference in the returned map.

This method returns an empty property set if its argument is `null`.

Parameter

`prop` the properties represented as a map.

Returns the properties represented as a `Properties` object.

```
public static PropertiesParams marshalProperties (Map<String, Object> prop)
```

Marshals the given map into a `PropertiesParams` object. The result of this method can be associated directly with a JAXB element of type `PropertiesParams`. Each entry of the given map is converted into an object whose class extends `AbstractProperty`, the exact subclass depending on the class of the value. This method supports the same property types as `unmarshalProperties (PropertiesParams)`. If a property in `prop` has a value with an unsupported non-array type, the value is turned into a string using its `toString` method. Unsupported array types are converted to strings using a mechanism similar to `Arrays.deepToString (Object[])` but working for objects and primitive types.

This method returns `null` if its argument is `null`.

Parameter

`prop` the Java Properties.

Returns the Properties parameters.

```
public static Connection createConnection (DBConnectionParams dbParams)
                                     throws SQLException
```

Creates a database connection from the parameters stored in the given `dbParams` object. This method first constructs a `Properties` object from the properties in `dbParams` with the help of the `unmarshalProperties (PropertiesParams)`, and `getProperties (Map)` methods. It then checks attributes in `dbParams` to decide the way the connection is established. More specifically, if a JNDI name is specified, i.e., `DBConnectionParams.isSetJndiDataSourceName()` returns `true`, the properties are used as an environment to create an `InitialContext` for JNDI, the constructed context is used to look for a `JDBC DataSource`, and the connection is obtained. Otherwise, the driver is loaded if its class is not `null`, and the connection is established using the URI and properties.

Parameter

`dbParams` the parameters of the connection.

Returns the reference to an object representing the established database connection.

Throws

`SQLException` if a connection error occurred.

```
public static Distribution createBasicDistribution (RandomVariateGenParams
                                                  rvgp) throws
                                                  DistributionCreationException
```

Constructs and returns an object representing the distribution based on the parameters in `rvgp`. This method first resolves the class name given by `RandomVariateGenParams.getDistributionClass()`. The creation of the distribution object then depends on whether data or parameters are specified in `rvgp`, which is determined using `RandomVariateGenParams.isEstimateParameters()`. The constructed distribution does not take account of

the truncation bounds `RandomVariateGenParams.getLowerBound()`, and `RandomVariateGenParams.getUpperBound()`; this is considered in `createTruncatedDist (Distribution, RandomVariateGenParams)`. Moreover, the shift returned by `RandomVariateGenParams.getShift()` is only used for parameter estimation; it is considered by `createGenerator (RandomVariateGenParams, RandomStream)`.

If `RandomVariateGenParams.isEstimateParameters()` returns `true`, the result of `RandomVariateGenParams.getParams()` is considered as an array of observations, and parameter estimation is performed. For this, the array is copied into an array of double-precision values. If a shift is specified using `RandomVariateGenParams.getShift()`, it is added to each observation in the intermediate array. If the distribution class corresponds to `EmpiricalDist` or `PiecewiseLinearEmpiricalDist`, the observations are sorted, and used to construct the distribution directly. Otherwise, parameter estimation is performed by using `DistributionFactory.getDistributionMLE (Class, double[], int)`. In the case of discrete distributions over the integers, each double-precision observation is rounded to the nearest integer, and used for parameter estimation by the method `DistributionFactory.getDistributionMLE (Class, int[], int)`.

If `RandomVariateGenParams.isEstimateParameters()` returns `false`, the array returned by `RandomVariateGenParams.getParams()` represents parameters. This method then looks for a constructor, in the selected distribution class, taking an array of double-precision values, and calls that constructor if it exists. If such a constructor cannot be found, it searches for a constructor taking n numerical parameters, where n is the number of parameters given. A distribution-creation exception is thrown if no suitable constructor can be found.

Parameter

`rvgp` the parameters of the random variate generator.

Returns a reference to the object representing the distribution.

Throws

`DistributionCreationException` if an exception occurred during the creation of the distribution.

```
public static boolean estimateParameters (RandomVariateGenParams rvgp)
```

```
throws
```

```
DistributionCreationException
```

Replaces the array of observations returned by `RandomVariateGenParams.getParams()` with an array obtained by parameter estimation. Parameters are estimated a way similar to `createBasicDistribution (RandomVariateGenParams)`, and copied into the array returned by `RandomVariateGenParams.getParams()`. However, instead of using `DistributionFactory`, this method calls the `getMaximumLikelihoodEstimate` or `getMLE` static methods directly to get the array of parameters. After this method returns, the array of estimated parameters can be obtained using `RandomVariateGenParams.getParams()` while the array of observations is lost, and `RandomVariateGenParams.isEstimateParameters()` returns `false`. This method does nothing if `RandomVariateGenParams.isEstimateParameters()` returns `false` for the given `rvgp` object.

Parameter

`rvgp` the random variate generator parameters.

Returns `true` if and only if the data is replaced by estimated parameters.

Throws

`DistributionCreationException` if an exception occurred when getting the distribution class.

```
public static double getMean (RandomVariateGenParams rvgp) throws  
                             DistributionCreationException
```

Determines the mean of the distribution corresponding to the parameters given by `rvgp`. This method creates the distribution, and uses `Distribution.getMean()` to obtain the mean. It also applies the shift given by `RandomVariateGenParams.getShift()`.

Parameter

`rvgp` the parameters of the random variate generator.

Returns the mean.

Throws

`DistributionCreationException` if a problem occurred while creating the distribution.

```
public static double getVariance (RandomVariateGenParams rvgp) throws  
                                 DistributionCreationException
```

Determines the variance of the distribution corresponding to the parameters given by `rvgp`. This method creates the distribution, and uses `Distribution.getVariance()` to obtain the variance.

Parameter

`rvgp` the parameters of the random variate generator.

Returns the variance.

Throws

`DistributionCreationException` if a problem occurred while creating the distribution.

```
public static TruncatedDist createTruncatedDist (Distribution dist,  
                                                  RandomVariateGenParams  
                                                  rvgp) throws  
                                                  DistributionCreationException
```

Constructs and returns a truncated distribution object from distribution `dist`, and parameters `tp`. This method throws a distribution-creation exception if the given distribution is not continuous, or if parameters are invalid.

Parameters

`dist` the distribution to truncate.

`rvgp` the parameters of the random variate generator.

Returns the truncated distribution.

Throws

`DistributionCreationException` if an error occurs during the construction of the distribution.

```
public static RandomVariateGen createGenerator (RandomVariateGenParams
                                              rvgp, RandomStream stream)
                                              throws
                                              DistributionCreationException,
                                              GeneratorCreationException
```

Constructs and returns a random variate generator from the parameters given by `rvgp`, and the random stream `stream`. This method uses `createBasicDistribution (RandomVariateGenParams)` to create the basic distribution, and `createGenerator (RandomVariateGenParams, RandomStream, Distribution)` to create the random variate generator.

Parameters

`rvgp` the parameters of the generator.

`stream` the random stream.

Returns the random variate generator.

Throws

`DistributionCreationException` if an error occurred during the creation of the distribution.

`GeneratorCreationException` if an error occurred during the creation of the generator.

```
public static RandomVariateGen createGenerator (RandomVariateGenParams
                                              rvgp, RandomStream stream,
                                              Distribution dist) throws
                                              DistributionCreationException,
                                              GeneratorCreationException
```

Constructs and returns a random variate generator from the parameters given in `rvgp`, the random stream `stream`, and the probability distribution `dist`. First, if truncation parameters are specified, this method uses `createTruncatedDist (Distribution, RandomVariateGenParams)` to construct the truncated distribution, and creates a random variate generator using inversion. Otherwise, the method creates a generator from the class given by `RandomVariateGenParams.getGeneratorClass()`. If no generator class is specified, a `RandomVariateGen` is used for any distribution, except discrete distributions over the integers for which the method uses `RandomVariateGenInt`. This results in using inversion if no random variate generator was specified. The constructed generator is finally wrapped around a `RandomVariateGenWithShift` instance if a shift was defined in `rvgp`.

Parameters

`rvgp` the random variate generator parameters.

`stream` the random stream.

`dist` the probability distribution.

Returns the random variate generator.

Throws

`DistributionCreationException` if an exception occurred during construction of the truncated distribution.

`GeneratorCreationException` if an exception occurred when constructing the generator.

```
public static boolean sameGenerators (RandomVariateGenParams rvgp1,  
                                     RandomVariateGenParams rvgp2, double  
                                     tol)
```

Determines if `rvgp1` and `rvgp2` describe two equivalent random variate generators. That is, if they have the same distribution, generation method, and same parameters within tolerance `tol`. Numerical parameters are compared as follows by this method: parameters a and b are equal if and only if $|b - a| < \text{tol}$.

Parameters

`rvgp1` the first random variate generator.

`rvgp2` the second random variate generator.

`tol` the tolerance for comparing numbers.

Returns `true` if and only if the generators are considered to be equivalent.

DistributionCreationException

This exception is thrown when a problem occurs during the construction of a distribution object by `ParamReadHelper.createBasicDistribution (RandomVariateGenParams)` or `ParamReadHelper.createTruncatedDist (Distribution, RandomVariateGenParams)`.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class DistributionCreationException extends Exception
```

Constructors

```
public DistributionCreationException()
```

Constructs a new distribution creation exception with no distribution information or message.

```
public DistributionCreationException (String message)
```

Constructs a new distribution creation exception with no distribution information and message `message`.

Parameter

`message` the message describing the exception.

```
public DistributionCreationException (Class<? extends Distribution>
                                     distClass, double[] distParams)
```

Constructs a new distribution creation exception with distribution class `distClass`, distribution parameters `distParams`, and no message.

Parameters

`distClass` the class of the distribution which cannot be created.

`distParams` the parameters given to the constructor of the distribution class.

```
public DistributionCreationException (Class<? extends Distribution>
                                     distClass, double[] distParams,
                                     String message)
```

Constructs a new distribution creation exception with distribution class `distClass`, distribution parameters `distParams`, and message `message`.

Parameters

`distClass` the class of the distribution which cannot be created.

`distParams` the parameters given to the constructor of the distribution class.

`message` the message describing the exception.

Methods

```
public Class<? extends Distribution> getDistributionClass()  
( )
```

Returns the distribution class which caused the exception.

Returns the distribution class having caused the exception.

```
public double[] getDistributionParameters()
```

Returns the distribution parameters for which there is no corresponding constructor in the distribution class, or an exception occurred during the call to a constructor.

Returns the distribution parameters having caused the exception.

```
public String toString()
```

Returns a short description of this exception. If no distribution class and parameters are associated with this exception, this method returns the result of the superclass's `toString` method. Otherwise, it returns a string with the following contents.

- The name of this class
- “: ”
- If a message is given, the text of the message.
- “, for ”
- The name of the distribution class if available
- If the parameters are available, “(”, parameters, “)”

Returns the short string describing the exception.

GeneratorCreationException

This exception is thrown when a problem occurs during the construction of a random variate generator by `ParamReadHelper.createGenerator (RandomVariateGenParams, RandomStream)`.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class GeneratorCreationException extends Exception
```

Constructors

```
public GeneratorCreationException()
```

Constructs a new generator creation exception with no generator information or message.

```
public GeneratorCreationException (String message)
```

Constructs a new generator creation exception with no generator information and message `message`.

Parameter

`message` the message describing the exception.

```
public GeneratorCreationException (Distribution dist, Class<? extends  
                                RandomVariateGen> genClass)
```

Constructs a new generator creation exception with distribution `dist`, generator class `genClass`, and no message.

Parameters

`dist` the distribution associated with the generator.. = *

`genClass` the class of the random variate generator that cannot be created.

```
public GeneratorCreationException (Distribution dist, Class<? extends  
                                RandomVariateGen> genClass, String  
                                message)
```

Constructs a new generator creation exception with distribution `dist`, generator class `genClass`, and message `message`.

Parameters

`dist` the distribution associated with the generator..

`genClass` the class of the random variate generator that cannot be created.

`message` the message describing the exception.

Methods

```
public Distribution getDistribution()
```

Returns the distribution which caused the exception.

Returns the distribution having caused the exception.

```
public Class<? extends RandomVariateGen> getGeneratorClass  
( )
```

Returns the generator class which caused the exception.

Returns the generator class having caused the exception.

```
public String toString()
```

Returns a short description of this exception. If no distribution and generator class are associated with this exception, this method returns the result of the superclass's `toString` method. Otherwise, it returns a string with the following contents.

- The name of this class
- “: ”
- If a message is given, the text of the message.
- “, for ”
- If the distribution is available
 - “distribution ”
 - The result of `Object.toString()`
 - If the generator class is available “, ”
- If the generator class is given, “generator ” followed by the generator class name

Returns the short string describing the exception.

RemappingContentHandler

SAX content handler remapping namespace prefixes. Some tools produces XML without giving control over the prefixes associated with namespace URIs. For example, the JAXB marshaller can output XML, but it generates prefixes such as `ns2`, `ns3`, etc. This content handler can be used to filter SAX events generated by such processors to map namespace URIs to user-defined prefixes. One simply provides a map associating prefixes with URIs along with a content handler filtered events are sent to. This content handler is used by `JAXBParamsConverter` to implement namespace prefix mapping independently of the JAXB provider.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class RemappingContentHandler implements ContentHandler
```

Constructor

```
public RemappingContentHandler (Map<String, String> prefixToUri,  
                                ContentHandler targetHandler)
```

Constructs a new remapping content handler sending events to the given target handler, and using the supplied prefix-to-URI map.

Parameters

`targetHandler` the target handler to send events to.

`prefixToUri` the prefix-to-URI mapping.

Throws

`NullPointerException` if the content handler is `null`.

`IllegalArgumentException` if one URI is mapped to multiple prefixes.

Methods

```
public ContentHandler getTargetHandler()
```

Returns the target content handler.

Returns the target content handler.

```
public void setTargetHandler (ContentHandler targetHandler)
```

Sets the target content handler.

Parameter

`targetHandler` the new target handler.

NamedInfo

Represents the information about an entity with a name and possibly properties. This object is constructed from a `Named` instance which is obtained by unmarshalling some XML elements using JAXB. It allows the user to access the properties using a Java map rather than a list with an object for each property. This class is often extended to represent specific entities, for example the call types of a call center.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class NamedInfo
```

Constructors

```
public NamedInfo (Named named)
```

Constructs a new named entity from the parameter object `named`.

Parameter

`named` the parameter object representing the named entity.

```
public NamedInfo (String name)
```

Constructs a named entity with name `name`, and no property.

Parameter

`name` the name of the new entity.

```
public NamedInfo (String name, Map<String, ? extends Object> properties)
```

Constructs a new named entity with name `name`, and properties stored in the map `properties`. Each key of the given map represents the name of a property while the corresponding value in the map is the value of the property.

Parameters

`name` the name of the entity.

`properties` the properties of the entity.

Methods

```
public String getName()
```

Returns the name associated with this named entity. This returns `null` if no name is associated.

Returns the associated name.

```
public Map<String, Object> getProperties()
```

Returns the properties associated with the entity represented by this object. Each key of the returned map represents the name of a property while the corresponding value in the map is the value of the property.

Returns the associated properties.

```
public Map<String, String> getStringProperties()
```

Returns a map constructed by converting each value of properties in map returned by `getProperties()` to a string. If a property has the `null` value, it is converted to the “null” string.

Returns the properties, with their values converted to strings.

SourceArray2D

Represents a 2D array obtained from a data source such a text file, or a spreadsheet. Such a source array can be used to create 1D or 2D arrays. The `rows()`, `columns (int)`, and `get (Class, int, int)` methods can then be used to inspect the array.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public interface SourceArray2D extends Closeable
```

Methods

```
public int rows()
```

Returns the number of rows in the source array.

Returns the number of rows in the array.

```
public int columns (int row)
```

Returns the number of columns in row `row` of the source array.

Parameter

`row` the row to test.

Returns the number of columns in the row.

Throws

`IllegalArgumentException` if the row index is out of bounds.

```
public <T> T get (Class<T> pcls, int row, int column)
```

Returns the element at row `row` and column `column` of the source array, converted to class `pcls`.

Type parameter

`T` the target class.

Parameters

`pcls` the target class.

`row` the row index.

`column` the column index.

Returns the element.

Throws

`IllegalArgumentException` if the row or column indices are out of bounds.

`ClassCastException` if the element cannot be converted to the target class.

```
public void close()
```

Clears the data in the source array.

SourceSubset2D

Represents a source subset obtained from a source array. Such a 2D array is obtained by taking a subset of the rows and the columns of another source array.

```
package umontreal.iro.lecuyer.xmlbind;  
  
public class SourceSubset2D implements SourceArray2D
```

Constructor

```
public SourceSubset2D (SourceArray2D sourceArray, int fromRow, int  
                      fromColumn, int numRows, int numColumns, boolean  
                      transposed)
```

Constructs a new subset from the array `sourceArray`.

Parameters

`sourceArray` the original source array.

`fromRow` the starting row in the original array.

`fromColumn` the starting column in the original array.

`numRows` the number of rows in the subset.

`numColumns` the number of columns in the subset.

`transposed` determines if the subset needs to be transposed.

Throws

`IllegalArgumentException` if one or more arguments are negative or out of bounds.

`IllegalStateException` if `sourceArray` is not initialized.

CSVSourceArray2D

Represents a source array whose contents is read from a CSV-formatted text file. Each line of the text file pointed to by a URL becomes a row of the source array, with elements of the row separated using commas. Text is read using `TextDataReader.readCSVData (URL, char, char)` with `,` as the column delimiter and `"` as the string delimiter, while `StringConvert.fromString (URI, ClassFinder, Class, String)` is used to convert strings to target objects.

In the XML file, the `URL` attribute of an element representing a CSV source array must be used to indicate the URL of the CSV data file.

```
package umontreal.iro.lecuyer.xmlbind;  
  
public class CSVSourceArray2D implements SourceArray2D
```

Method

```
public URL getURL()
```

Returns the URL of the text file containing the values of the array, in CSV format.

Returns the URL of the CSV data.

ExcelSourceArray2D

Represents a source array whose contents is read from a Microsoft Excel workbook. Each row of the sheet with the given name, and contained in the workbook pointed to by a URL becomes a row of the source array, with elements of the row separated using commas. Data is read using JExcel API.

In the XML file, the `URL` attribute of an element representing a Excel source array must be used to indicate the URL of the data file. The `sheetName` attribute can also be used to indicate the name of a sheet. By default, the first sheet is read.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class ExcelSourceArray2D
```

Methods

```
public URL getURL()
```

Returns the URL of the text file containing the values of the array, in CSV format.

Returns the URL of the CSV data.

```
public void close()
```

Closes the workbook associated with this object.

```
public SourceArray2D getSheet (int sheetIndex)
```

Constructs and returns a source array corresponding to the sheet with index `sheetIndex` in the workbook.

Parameter

`sheetIndex` the sheet index.

Returns the source array corresponding to the sheet.

```
public SourceArray2D getSheet (String sheetName)
```

Constructs and returns a source array corresponding to the sheet with name `sheetName` in the workbook.

Parameter

`sheetName` the sheet name.

Returns the source array corresponding to the sheet.

DBSourceArray2D

Represents a source array whose data is extracted from a database using JDBC. The elements of the array are obtained by performing a query on a database. Each row of the resulting result set is a row in the source array, while each column corresponding to a field of the result set becomes a column in the array. The JDBC connection is initialized using `DBConnectionParam`, and the result set is converted into an array of objects using `JDBCManager.readObjectData2D (Connection, String)`. Any numeric object (instances of `Number`) is converted to the target class while other objects not corresponding to the target class are converted to string before they are passed to `StringConvert.fromString (URI, ClassFinder, Class, String)`.

In a XML file, the `dataQuery` attribute of an element representing a database-based source array is used to specify the query on the database. The `database` nested element is then used to describe the connection to the database.

```
package umontreal.iro.lecuyer.xmlbind;
```

```
public class DBSourceArray2D
```

Methods

```
public DBConnectionParams getDatabase()
```

Returns the parameters of the database connection used to obtain data for this source array.

Returns the database connection parameters.

```
public Connection getConnection()
```

Returns the connection to the database.

Returns the current connection.

Package `umontreal.iro.lecuyer.xmlconfig`

Provides some facilities to extract and verify parameters read from data files in XML format. Parameter reading is often tedious and clutters the application code with repetitive statements. Read parameters must be converted from string to useful types, checked for validity and processed in many ways before they can be used by an application program. By using XML [5], one can take advantage of a standardized file format syntax and already implemented and robust parsers providing error checking. However, even when a validating parser, which is able to ensure that an XML file satisfy predefined structural constraints, is used, the parameters must be extracted from a parse tree and converted from string.

This package provides facilities to extract and convert values from a Document Object Model (DOM) document constructed using the Java API for XML (JAXP) or any other API implementing DOM Level 2 [3] or above. It uses a design-patterns oriented technique strongly inspired from Apache Ant [1] task building but extended to be used in a more general context.

Format of parameter files

This section briefly introduces the XML format as used by the parameter files. See [5] for the full XML specification. The first line of an XML file is optional and contains an header specifying the version of the format and the encoding. This line is like the following one:

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

If this line is not given, the UTF-8 character encoding is assumed. Specifying the encoding can be useful to allow accented characters to appear in the input when the used editor does not support UTF-8.

In an XML file, an *element* is a container with a name and possibly some attributes. It is represented by a starting marker and an ending marker. The text between these markers is called the *contents* of the element. For example, `<element>contents</element>` declares an element with name `element` and contents `contents`. An element with no contents can be formatted as `<element/>`. This is called a *self-closing element*. The whole XML document is contained into a special element called the *root element*.

An *attribute* is a key-value pair associated with an element. An element can have zero or more attributes. For example, `<element attribute="value"/>` declares a self-closing element `element` with attribute `attribute` having value `value`. The order of an element's attributes is not important in any XML document.

The nested contents can be simple or complex. *Simple contents* is composed of only *character data*, i.e., text with no XML markers. If such characters are required for some reasons, they must be escaped by using *entities*. Entities are sequences of characters automatically substituted with character data by an XML parser. For the user, they act similarly to macros. Table 1 shows the entities used to escape reserved characters.

Table 1: XML entities used to escape reserved characters

Entity	Escaped character
<code>&lt;</code>	<code><</code>
<code>&gt;</code>	<code>></code>
<code>&quot;</code>	<code>"</code>
<code>&amp;</code>	<code>&</code>

Complex contents is composed of character data and other elements. Some XML document types specify an order in which the elements need to be presented. For the parameter reader, the order of elements is not important.

At any point in the XML file, *comments* of the form `<!-- comment -->` can be added. These comments are ignored by the parameter reader and can be used to document the parameter files.

Processing instructions can be used to communicate with specific XML processors. The parameter reading facility implemented in this package supports the `import` processing instruction which can be used to import packages when referring to class names in parameter files. The `import` processing instruction works the same way as the Java `import` statement. For example, `<?import java.util.*?>` imports all classes in the `java.util` package.

Extracting parameters

For an XML document to be converted into a parameter object, an instance of `ParamReader` is needed. The root element name must be bound to a parameter object class, by modifying the `elements` map. For example, the following code associates the `myparams` root element with the class `MyParams`.

```
ParamReader reader = new ParamReader();
reader.elements.put ("myparams", MyParams.class);
```

After the binding is done, the reading method can be invoked. The `read` method accepts a `Document` implementation or an XML file name and turns it into a parameter object. In the preceding example, the general format of the XML file would be

```
<myparams>
  ...
</myparams>
```

The following call would read the XML file and creates a `MyParams` instance.

```
MyParams par = (MyParams)reader.read ("file.xml");
```

Converting back to XML

To be converted back to XML, a parameter object needs to provide a write method specified in the `StorableParam` interface. This method turns the parameter object into a DOM document. The DOM document can then be converted to an XML file. The `AbstractParam` class provides a `write` method capable of converting the storable parameter object to an XML file.

For example, if `MyParams` implements `StorableParam`, the following code can write the parameter object `par` to an XML file.

```
AbstractParam.write (new ClassFinder(), "file.xml",  
                    par, "myparams", 3);
```

The class finder is used to convert `Class` objects to simple class names, taking the import declarations into account. The name of the root element, `myparams`, must be given, as well as the number of spaces for each indentation level.

Creating new parameter object classes

A parameter object can come from any class implementing the `Param` interface, providing a no-argument constructor, and supplying setter, adder, or creator methods that will be mapped to XML attributes and nested elements. *Setter* methods are used to notify the parameter object about the value of attributes whereas *adder* and *creator* methods are used to notify nested elements.

For attribute *attr*, the setter method `setAttr` is called and the string contents of the attribute is converted to the target class by `StringConvert`. The *id* attribute has a special meaning for the parameter reader; it assigns a name to an element. When the *xref* attribute is associated with an element, the element's contents is expected to be empty and the only allowed attribute is *xref*. The parameter reader replaces such a reference element's attributes and contents with the element having the matching *id* attribute.

Nested XML elements are recursively turned into parameter objects by adder and creator methods. An adder method has the form `addElement` whereas a creator method has the form `createElement`. For simple contents, the adder method can accept a class that can be converted by `StringConvert`. For complex contents to be represented, another parameter object must be used.

For a parameter object to become storable, it must implement the `StorableParam` sub-interface of `Param`. This interface specifies a writing method responsible for the conversion. The `DOMUtils` class contains helper methods that can be used during the conversion of any parameter object to a DOM tree.

Example of a parameter file. The following example shows how the parameters from a sample file are handled and converted.

```
<?xml version="1.0"?>

<myparams id="test">
  <numtypes>3</numtypes>
  <numgroups>2</numgroups>
  <arrivalrates>2, 4, 2</arrivalrates>
</myparams>
```

We define a class `MyParams` implementing `Param` and providing a no-argument constructor and the following methods. The methods do not need to be public.

```
class MyParams implements Param {
    void setId (String id) { ... }
    void addNumtypes (int n) { ... }
    void addNumgroups (int n) { ... }
    ArrayParam createArrivalrates() {
        return new ArrayParam (double.class);
    }
    addArrivalrates (ArrayParam rates) { r = rates.getDoubleValues(); }
}
```

After the parameter object is constructed, the parameter reader calls `setId` with the string `test`. All other attributes are mapped to a corresponding setter method.

The parameter reader then calls `addNumtypes` with integer 3. The same process happens for the `numgroups` nested element.

When the `arrivalrates` element is found, the `createArrivalrates` method is called to create a parameter object. This creator method is necessary because the `ArrayParam` class does not provide a no-argument constructor since the user needs to specify the component class for the array. The parameter reader constructs the array parameter object and pass it back to `addArrivalrates` when the configuration is done. The `getDoubleValues` method can be used to get an array of double-precision elements when the array component class is numeric. Note that there is a similar class for 2D arrays called `ArrayParam2D`.

ParamReader

Constructs a parameter object from an XML document parsed using a DOM parser. For this parameter reader to be used, values must be added to the map `elements` in order to map root elements to class names. The method `read (String)` can be used to construct a new parameter object from a DOM document or an XML file.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class ParamReader
```

Fields

```
public Map<String, Class<? extends Param>> elements
```

Provides mappings for root elements to Java classes. The keys of the `elements` map are strings representing XML element names whereas the values are `Class` objects corresponding to parameter objects. When an element belongs to a namespace, its corresponding key in the map is given by *namespaceURI/tagname*, where *tagname* is the tag name of the element, without the namespace prefix. If the element is not in a namespace, its key name is its tag name. For example, if the root element was given by

```
<pr:parameters xmlns:pr="http://www.test.uri">
```

The key name would be `http://www.test.uri/parameters`. If no namespace is used, the name is `parameters`.

```
public File[] searchPath
```

Contains the search path for the `searchFile (String)` method. See the documentation of `searchFile (String)` for more information. The default search path contains a single file referring to `.`, the current directory.

```
public URI baseURI
```

Contains the base URI used by the `readURL (String)` method. The default base URL corresponds to the current working directory.

Constructor

```
public ParamReader()
```

Constructs a new parameter reader.

Methods

```
public static URI getDefaultBaseURI()
```

Returns the default base URI, which corresponds to the location of the current directory.

Returns the default base URL.

```
public boolean isUsingSetAccessible()
```

Determines if the parameter reader can use the `AccessibleObject.setAccessible (boolean)` when accessing members using Reflection. This reader allows setter, adder, and dispatcher methods in parameter objects to be protected or private. In this case, it uses the `setAccessible` method to bypass Java access control. However, this can cause problems when using applets or Java Web Start programs. As a result, one can prevent this parameter reader from calling the `setAccessible` method by calling `setUsingSetAccessible (boolean)` with `false`. However, disabling the set-accessible usage flag may result in `ParamReadExceptions` caused by illegal accesses. By default, this returns `true`.

Returns the status of the set-accessible usage flag, default being `true`.

```
public void setUsingSetAccessible (boolean useSetAccessible)
```

Sets the set-accessible usage flag to `useSetAccessible`.

Parameter

`useSetAccessible` the new value of the flag.

See also `isUsingSetAccessible()`

```
public ClassFinder getClassFinder()
```

Returns the class finder associated with this parameter reader. The import declarations associated with this class finder are obtained from the XML parameter file through `import` processing instructions. For each parsed element, `ClassFinder.saveImports()` is used at the beginning of processing and `ClassFinder.restoreImports()` is used at the end. For each processing instruction `<?import name=?>`, `name` is added to the list returned by `ClassFinder.getImports()`.

Returns the class finder associated to this parameter reader.

```
public Element getElementById (String id)
```

Returns the element, in the currently read document, having an `id` attribute with value `id`. If no such element exists, `null` is returned.

Parameter

`id` the identifier of the element.

Returns the corresponding element.

```
public Param readFile (String fileName) throws IOException,  
                      ParserConfigurationException, SAXException
```

Reads the parameter object given by the file `fileName`. Uses `searchFile (String)` to find an existing file with the name `fileName` on the current search path, and passes this file to the `read (File)` method. With the default search path, this looks in the current directory only.

Parameter

`fileName` the file name to be parsed.

Returns the constructed parameter object.

Throws

`ParserConfigurationException` if the parser could not be configured properly.

`IOException` if an I/O error occurs.

`SAXException` if a parse error occurs.

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

```
public Param readURL (String url) throws IOException,  
                     ParserConfigurationException, SAXException
```

Reads the parameter object given by the URL `url`. Uses `URI.resolve (URI)` to resolve `url` against `baseURI`, converts the resulting URI into a URL, and give the resulting URL to `read (URL)`.

Parameter

`url` the relative URL.

Returns the parameter object.

Throws

`IOException` if an I/O exception occurs during parameter reading.

`SAXException` if a parse error occurs.

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

```
@Deprecated public Param read (String fileName) throws IOException,  
                               ParserConfigurationException, SAXException
```

Uses `searchFile (String)` to find an existing file with the name `fileName` on the current search path, and passes this file to the `read (File)` method. With the default search path, this looks in the current directory only.

Parameter

`fileName` the file name

Returns the constructed parameter object.

Throws

`ParserConfigurationException` if the parser could not be configured properly.

`IOException` if an I/O error occurs.

`SAXException` if a parse error occurs.

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

Deprecated Use `readFile (String)` instead.

```
public Param read (File fileName) throws IOException,  
                  ParserConfigurationException, SAXException
```

Reads a parameter object from the XML file `fileName`. If the given file is found, the method adds its parent directory to the search path, and sets it as the base URI. It then creates an XML parser using JAXP, parses the XML file and passes the created DOM document to (`Documnet`). The `DocumentBuilder` instance used to parse the XML files is created only once for each instance of parameter reader, by the `getDocumentBuilder()` method. After this process, the method restores the original search path.

Parameter

`fileName` the file name to be parsed.

Returns the constructed parameter object.

Throws

`ParserConfigurationException` if the parser could not be configured properly.

`IOException` if an I/O error occurs.

`SAXException` if a parse error occurs.

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

```
public File searchFile (String fileName) throws FileNotFoundException
```

Returns a file object corresponding to an existing file with name `fileName` by looking on the current search path. If the given file name is an absolute path, or the search path stored in the `searchPath` field is `null` or has length 0, this method creates a file object from the given file name. If that file exists, i.e., if `fileName` is a valid relative or absolute path pointing to an existing file, this method returns the file object. Otherwise, for each non-`null` element `p` of the `searchPath` array, this method makes a file object with parent `p` and name `fileName`. It returns the first file object referring to an existing file. If no file with the given name can be found on the search path, this method throws a `FileNotFoundException`.

Parameter

`fileName` the name of the file to search for.

Returns the found file.

Throws

`FileNotFoundException` if the file cannot be found.

```
public Param read (URL url) throws IOException,  
                  ParserConfigurationException, SAXException
```

Reads a parameter object from the XML file located at URL `url`. The method sets the given URL as the base URI. If the given URL corresponds to a file, the method also adds its parent directory to the search path. It then creates an XML parser using JAXP, parses the XML file and passes the created DOM document to (`Documnet`). The `DocumentBuilder` instance used to parse the XML files is created only once for each instance of parameter reader, by the `getDocumentBuilder()` method. After this process, the method restores the original search path.

Parameter

`url` the URL pointing to the file to be parsed.

Returns the constructed parameter object.

Throws

`ParserConfigurationException` if the parser could not be configured properly.

`IOException` if an I/O error occurs.

`SAXException` if a parse error occurs.

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

```
public Param read (InputStream stream) throws IOException,  
                  ParserConfigurationException, SAXException
```

This is similar to `read (String)`, but it reads the XML document from the stream `stream` instead of from a file.

Parameter

`stream` the stream to read the XML document from.

Returns the constructed parameter object.

Throws

`ParserConfigurationException` if the parser could not be configured properly.

`IOException` if an I/O error occurs.

`SAXException` if a parse error occurs.

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

```
public DocumentBuilder getDocumentBuilder() throws
                        ParserConfigurationException
```

Returns the document builder instance associated with this instance of parameter reader. If no document builder is associated yet, one is created. This coalescent document builder is configured to expand entities, ignore white spaces and is not validating. See `DocumentBuilderFactory` for more information.

Returns the associated document builder.

Throws

`ParserConfigurationException` if the parser could not be configured properly.

```
public Param read (Document doc)
```

Reads a DOM document `doc` and constructs a parameter object based on its contents. If some problems occur during the extraction of parameters, a `ParamReadException` is thrown.

When given an XML document, the method maps the root element name to a Java `Class` object by using `elements`.

Before the parameter object is created, the given document is scanned for elements with non-empty `id` attributes. These elements are put into an internal map that is cleared before this method ends. Each time an element with a `xref` attribute is found, it is automatically replaced with an element having the corresponding `id` attribute. The parameter reader keeps a stack of currently-processed elements to avoid infinite loops during this reference resolution process.

The `createParameterObject (Class, Element)` method is then used to construct the parameter object from the XML element.

Parameter

`doc` the DOM document being read.

Returns the constructed parameter object.

Throws

`ParamReadException` if an extraction error occurs.

`ClassCastException` if `elements` contains a value which is not of class `Class`.

```
public <T extends Param> T createParameterObject (Class<T> c, Element e1)
```

Constructs a new parameter object of class `c` from the DOM element `e1`. The method first tries to construct the parameter object by calling one of the four type of constructors, in that order: `(ParamReader, Element)`, `(ParamReader)`, `(Element)`, and `()`. The constructors receiving the processed element allow a class to override the automatic parameter extraction. Otherwise, the method uses `processElement (Element, Param)` to perform the processing if the call to the constructor succeeds. If the construction of the object fails, a `ParamReadException` is thrown.

Parameters

`c` the target class of the parameter object.

`e1` the element the parameters are extracted from.

Returns the constructed parameter object.

Throws

`ParamReadException` if an error occurs during parameter extraction.

```
public void processElement (Element e1, Param o)
```

Configures the parameter object `o` by processing the DOM element `e1`. To achieve this result with arbitrary objects, the method assumes the parameter objects respect some design patterns summarized in tables 2 and 3. These patterns specify methods one can implement to interact with the parameter reader.

These methods are not specified in the `Param` interface because their signatures are incomplete or they are optional. Some method names depend on the attribute or nested element name and the argument type directs the conversion from string.

Some of the possible methods can take an optional URI argument which corresponds to the namespace URI of the attribute or nested element being set, created or added, or `null` if no namespace is used. If a namespace URI is incorrect, a tester method can return `false` and other methods can throw a `ParamReadException`.

If the class of the parameter object defines several setter, adder or creator methods having the same name, the reader selects a single method to call. The methods of the parameter object are sorted using the comparator `MethodComparator` and the sorted array of methods is searched linearly, the first appropriate method being selected. This comparator implements an heuristic to place the most specialized methods first. In particular, the method with the greatest number of arguments and supported by the reader is taken. If more than one methods have the same greatest number of arguments, some argument types have priority over other types. `ParamReader`, `Param`, and `Node` have high priority whereas arrays have low priority.

For each attribute with local name `attr`, a suport test is performed. If the parameter object class defines a `isAttributeSupported` method returning a boolean value, this method is

Name	Role
uri	Namespace URI of attribute or nested element
lname	Local name of attribute or nested element
<i>class</i>	Name of a class or primitive type
<i>pclass</i>	Name of a class implementing Param
value	The value converted from string
el	The element being processed
cel	The nested (or child) element
attr	An attribute in el

Table 2: Variables used in parameter object methods

Method	Design pattern	Role
Tester	<code>boolean isAttributeSupported ([ParamReader r,][String uri,][String lname])</code>	Determines if an attribute is supported
Setter	<code>void setAttr ([ParamReader r,][String uri,][class value])</code>	Sets <i>attr</i> to value
Tester	<code>boolean isNestedElementSupported ([ParamReader r,][String uri,][String lname])</code>	Determines if a nested element is supported
Adder	<code>void addNested ([ParamReader r,][String uri,][class paramobj])</code>	Adds processed nested element <i>nested</i>
Creator	<code>pclass createNested ([ParamReader r,][String uri Element cel])</code>	Creates the nested element <i>nested</i>
Text	<code>void nestedText ([ParamReader r,][class value])</code>	Adds character data
Dispatcher	<code>void defaultSetAttribute ([ParamReader r,][Attr attr])</code>	Fallback method for attributes with no setter method
Dispatcher	<code>void defaultNestedElement ([ParamReader r,][Element cel])</code>	Fallback method for elements with no adder or creator methods
Ender	<code>void finishReading ([ParamReader r,][Element el])</code>	Terminates the reading

Table 3: Summary of design patterns for parameter objects

called to test **attr**. If it returns **false**, an exception is thrown. If it returns **true** or the method does not exist, the attribute processing continues. After the support test, the method tries to call an appropriate setter method in the parameter object. The **String-Convert.fromString (URI, ClassFinder, Class, String)** method is given the string value of the attribute, and the result is given to the setter method. If no setter method is provided for a given attribute, the parameter reader tries to call a dispatcher method

named `defaultSetAttribute`. If no setter or dispatcher method can be called to set a given attribute, the reader terminates with a `ParamReadException`.

After the attributes are set, nested elements can be constructed and added to the parameter object. For nested text, a `nestedText` method is looked for. The argument of `nestedText` can be of any type provided that `StringConvert.fromString (URI, ClassFinder, Class, String)` is capable of converting it from `String`.

A nested element with local name `nested` is processed using an adder or a creator method. As with attributes, a support test is performed using `isNestedElementSupported`, if it exists. Often, a nested element may contain nested text only. For such elements, if the `paramobj` argument does not correspond to a class implementing `Param`, the parameter reader uses `StringConvert.fromString (URI, ClassFinder, Class, String)` to convert the nested text into an object for the adder method. Otherwise, the class of `paramobj` must implement `Param`. The `createParameterObject (Class, Element)` method is called recursively to obtain a parameter object which is passed to the adder method. The class of the nested parameter object depends on the argument of the adder method.

If the nested element cannot be constructed by a single-argument or a no-argument constructor, the adder method can be replaced by a creator method of the form `createNested`. The method must construct and return the parameter object, possibly after configuring it. When the traversed element is received, the automatic extraction process is bypassed. If both an adder and creator methods are present with the same argument type, the creator method is called first and the adder method is called with the processed parameter object which was constructed using the creator method.

If no adder or creator methods are available to process a nested element, the method tries to call a dispatcher method with name `addNested`. If no adder, creator or dispatcher method can process a given element, a `ParamReadException` is thrown.

When all contents is processed, the method looks for a `finishReading` method in the parameter object. If such a method is found, it is called with `e1` as an argument. This method can be used to finalize the parameter reading and processing.

Parameters

- `e1` the element being processed.
- `o` the parameter object being defined.

Throws

`ParamReadException` if a problem occurs during the extraction of parameters.

Nested class

```
public static class MethodComparator implements Comparator<Method>
```

Comparator for sorting the methods returned by `Introspection.getMethods (Class)`. For more consistent searching of overloaded methods, the `ParamReader.processElement (Element, Param)` method sorts the methods returned by `Introspection.getMethods (Class)`. Methods with different names are sorted in alphabetical order. If two or more methods have the same name, they are sorted from the greatest to the smallest visibility: public, protected, package-private, and private. If two methods share the same name and visibility, they are sorted from the greatest to the smallest number of arguments.

When methods with the same name, visibility, and same number of arguments must be compared, the comparator applies a test to each argument, until the methods can be ordered. First, argument 0 is compared with other methods' argument 0. If the arguments are equal, or cannot be ordered, the test is performed with argument 1, 2, etc., until the arguments can be ordered. If all the arguments are equal or cannot be ordered, the methods cannot be ordered and are declared equal by the comparator. In this case, the final order of the methods depends on the Virtual Machine being used.

Class (including subclasses)	Score
ParamReader	1
Node	2
Param	3
TimeParam or primitive type	4
Number	5
File, URI, URL	6
String	7
Object	8
Other	10
Arrays	2xComponent

Table 4: Score assigned to classes when comparing arguments

Arguments are compared based on their types only. To compare data types, the comparator assigns a score to each one and orders the type with the smallest score first. For array types, the score of the component type is multiplied by two. If types have the same score, they are equal or cannot be ordered by this algorithm. Table 4 gives the score assigned to each class.

ParamReadException

This exception is thrown when a problem happens when converting a DOM document into a tree of parameter objects.

```
package umontreal.iro.lecuyer.xmlconfig;  
  
public class ParamReadException extends RuntimeException
```

Constructors

```
public ParamReadException()
```

Constructs a new parameter reading exception with no message and no node.

```
public ParamReadException (Node node)
```

Constructs a new parameter reading exception with no message and the node `node`.

Parameter

`node` the node the exception happened into.

```
public ParamReadException (String message)
```

Constructs a new parameter reading exception with the given `message`.

Parameter

`message` the error message describing the exception.

```
public ParamReadException (Node node, String message)
```

Constructs a new parameter reading exception with the given `message` and the node `node`.

Parameters

`node` the node the exception happened into.

`message` the error message describing the exception.

Methods

```
public Node getNode()
```

Returns the DOM node in which the exception happened. If no node was associated with this exception, this returns `null`.

Returns the node in which the exception happened, or `null`.

`public String toString()`

Returns a short description of this exception. If no DOM node is associated with the exception, this calls the base class's `toString` method. Otherwise, a string containing the following elements is constructed and returned.

- The name of the class of this object.
- " : In "
- Information about the concerned node
- If a description message was provided
 - ", "
 - The description message

The node information depends on its type. For an element, the formatted node name is given. For an attribute, the name, value and owner element formatted name are given. For a text node, the contents of the node is returned, as well as the formatted name of its parent, are given. For a generic node, the name and value are returned. All node names are formatted using `DOMUtils.formatNodeName (Node)` and values are formatted using `DOMUtils.formatNodeValue (Node, int)` with a maximal length of 100.

Returns the short string describing the exception.

Param

Marks an object as a parameter object.

```
package umontreal.iro.lecuyer.xmlconfig;  
  
public interface Param
```


StorableParam

Represents a parameter object providing the capability to be converted back to a DOM element.

```
package umontreal.iro.lecuyer.xmlconfig;  
  
public interface StorableParam extends Param
```

Method

```
public Element toElement (ClassFinder finder, Node parent, String  
                           elementName, int spc)
```

Converts this parameter object to a DOM element using the class finder **finder** for formatting class names, with parent node **parent**, element name **elementName**, and **spc** spaces for each indentation level. The method must create an **Element** instance with name **elementName** and add it to the node **parent** of the DOM tree. It is recommended to use **DOMUtils** helper methods for this. After the element is created, attributes can be set and nested contents can be added. The configured DOM element is then returned.

Parameters

finder the class finder used to format class names.

parent the parent of the new element.

elementName the name of the constructed element.

spc the number of spaces for each indentation level.

Returns the newly-constructed element.

AbstractParam

Provides common attributes for parameter objects. This class provides the `id` attribute which can be used to identify an element in the XML file. The `xref` attribute can be used to reference an identified element. Both attributes are of type string.

Examples `<element id="element-id"/>`

`<element xref="element-id"/>`

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public abstract class AbstractParam implements Param
```

Methods

```
public String getId()
```

Returns the identifier associated with this element. By default, this is the empty string.

Returns the identifier of the element.

```
public void setId (String id)
```

Sets the identifier of this parameter element to `id`.

Parameter

`id` the new identifier of the element.

Throws

`NullPointerException` if `id` is null.

```
public String getXref()
```

Returns the identifier of the referenced element. By default, this is `null`.

Returns the referenced element identifier.

```
public void setXref (String xref)
```

Sets the referenced identifier to `xref`.

Parameter

`xref` the new referenced identifier.

Throws

`NullPointerException` if `xref` is null.

```
public void check()
```

Verifies that every needed parameter was specified. Throws a `ParamReadException` in case of missing parameters.

Throws

`ParamReadException` if some parameters are missing or invalid.

```
public static void write (String fileName, StorableParam par, String
                        rootName, int spc) throws IOException,
                        ParserConfigurationException,
                        TransformerException
```

Equivalent to `write (new ClassFinder(), fileName, par, rootName, spc)`.

```
public static void write (ClassFinder finder, String fileName,
                        StorableParam par, String rootName, int spc)
                        throws IOException,
                        ParserConfigurationException,
                        TransformerException
```

Convenience method to write the parameters `par` into an XML file `file`, with a root element name `rootName` and `spc` spaces for each needed indentation level. The class finder `finder` is used to convert `Class` objects into simple names. This method uses an XML transformer to write the document obtained using the `createDocument (ClassFinder, StorableParam, String, int)` method to an XML file.

Parameters

`finder` the class finder used to format class names.

`fileName` the name of the output file.

`par` the parameter object to be stored.

`rootName` the name of the root element of the XML file.

`spc` the number of spaces per indentation level.

Throws

`IOException` if an I/O error occurs.

`ParserConfigurationException` if the XML document builder could not be created.

`TransformerException` if the XML transformer could not be created properly.

```
public static void write (File file, StorableParam par, String rootName,
                        int spc) throws IOException,
                        ParserConfigurationException,
                        TransformerException
```

Equivalent to `write (new ClassFinder(), file, par, rootName, spc)`.

```
public static void write (ClassFinder finder, File file, StorableParam par,
                        String rootName, int spc) throws IOException,
                        ParserConfigurationException,
                        TransformerException
```

Same as `write (ClassFinder, String, StorableParam, String, int)`, for a file object rather than a file name.

Parameters

`finder` the class finder used to format class names.

`file` the object representing the output file.

`par` the parameter object to be stored.

`rootName` the name of the root element of the XML file.

`spc` the number of spaces per indentation level.

Throws

`IOException` if an I/O error occurs.

`ParserConfigurationException` if the XML document builder could not be created.

`TransformerException` if the XML transformer could not be created properly.

```
public static void write (OutputStream out, StorableParam par, String
                        rootName, int spc) throws IOException,
                        ParserConfigurationException,
                        TransformerException
```

Equivalent to `write (new ClassFinder(), out, par, rootName, spc)`.

```
public static void write (ClassFinder finder, OutputStream out,
                        StorableParam par, String rootName, int spc)
                        throws IOException,
                        ParserConfigurationException,
                        TransformerException
```

Same as `write (ClassFinder, String, StorableParam, String, int)`, but writes the XML contents to the output stream `out`.

Parameters

`finder` the class finder used to format class names.

`out` the output stream for the XML contents.

`par` the parameter object to be stored.

`rootName` the name of the root element of the XML file.

`spc` the number of spaces per indentation level.

Throws

`IOException` if an I/O error occurs.

`ParserConfigurationException` if the XML document builder could not be created.

`TransformerException` if the XML transformer could not be created properly.

```
public static void write (Writer out, StorableParam par, String rootName,  
                        int spc) throws IOException,  
                        ParserConfigurationException,  
                        TransformerException
```

Equivalent to `write (new ClassFinder(), out, par, rootName, spc)`.

```
public static void write (ClassFinder finder, Writer out, StorableParam  
                        par, String rootName, int spc) throws  
                        IOException, ParserConfigurationException,  
                        TransformerException
```

Same as `write (ClassFinder, String, StorableParam, String, int)`, but writes the XML contents to the writer `out`.

Parameters

`finder` the class finder used to format class names.

`out` the writer for the XML contents.

`par` the parameter object to be stored.

`rootName` the name of the root element of the XML file.

`spc` the number of spaces per indentation level.

Throws

`IOException` if an I/O error occurs.

`ParserConfigurationException` if the XML document builder could not be created.

`TransformerException` if the XML transformer could not be created properly.

```
public static Document createDocument (ClassFinder finder, StorableParam  
                                     par, String rootName, int spc)  
                                     throws  
                                     ParserConfigurationException
```

Constructs a DOM document from the storable parameter object `par`, using the class finder `finder` to resolve simple class names, with root element with name `rootName`, and with `spc` spaces of indentation.

Parameters

`finder` the class finder used to resolve simple class names.

`par` the parameter object to write.

`rootName` the name of the root element.

`spc` the number of spaces of indentation.

Returns the constructed document.

Throws

`ParserConfigurationException`

TimeParam

Represents a time duration or a rate parameter expressed in a time unit. This parameter object can be constructed from a string, or a value with a `TimeUnit` instance representing its unit. The `get (TimeUnit)` method can be used to obtain the value of the parameter, converted into a user-specified time unit. Usually, this parameter is used to represent a duration. Rates are supported for distribution scale parameters.

The time parameter can also be used as an XML attribute type or as a nested element when using `ParamReader` to parse an XML document into a parameter object.

```
package umontreal.iro.lecuyer.xmlconfig;

public class TimeParam extends AbstractParam
    implements Cloneable
```

Constructors

```
public TimeParam()
```

Constructs a time parameter with value 0.

```
public TimeParam (double value)
```

Equivalent to `TimeParam (value, false, null)`.

Parameter

`value` the encoded value.

Throws

`IllegalArgumentException` if `value` is negative.

```
public TimeParam (double value, TimeUnit unit)
```

Equivalent to `TimeParam (value, false, unit)`.

Parameters

`value` the encoded value.

`unit` the time unit in which the value is assumed to be expressed.

Throws

`IllegalArgumentException` if `value` is negative.

```
public TimeParam (double value, boolean rate, TimeUnit unit)
```

Constructs a new time parameter with encoded value `value`, rate indicator `rate`, and time unit `unit`. If `rate` is `true`, `value` represents a rate that must be converted to a time by inversion before any time unit conversion is performed. `unit` represents the time unit in which `value` is assumed to be expressed. If `unit` is `null`, the `get (TimeUnit)` method performs no time conversion.

Parameters

value the encoded value.

rate the rate indicator.

unit the time unit in which the value is assumed to be expressed.

Throws

IllegalArgumentException if **value** is negative.

public TimeParam (String str)

Constructs a time parameter from the string **str**. A string representing a time parameter must contain a value, an optional rate indicator and an optional unit. If the value is followed by the rate indicator **/**, it is inverted before time unit conversion is performed by **get (TimeUnit)**. The time unit must be the short name of any instance of **TimeUnit**. For example, **12s** represents 12 seconds and **12/m** represents a rate of 12 units per minute.

Parameter

str the string representation of the time parameter.

Throws

IllegalArgumentException if the string is invalid.

Methods

public double getValue()

Returns the value of this time parameter, without conversion.

Returns the time parameter's value.

public void setValue (double value)

Sets the value of this time parameter to **value**.

Parameter

value the new time parameter's value.

Throws

IllegalArgumentException if **value** is negative or NaN.

public boolean getRateIndicator()

Returns the rate indicator of this time parameter.

Returns the time parameter's rate indicator.

public void setRateIndicator (boolean rate)

Sets the rate indicator of this time parameter to **rate**.

Parameter

`rate` the new time parameter's rate indicator.

```
public TimeUnit getTimeUnit()
```

Returns the time unit of this parameter.

Returns this parameter's time unit.

```
public void setTimeUnit (TimeUnit unit)
```

Sets the time unit of this parameter to `unit`.

Parameter

`unit` the new time unit of this parameter.

```
public double get (TimeUnit dstUnit)
```

Equivalent to `get (false, dstUnit)`.

Parameter

`dstUnit` the destination time unit.

Returns the computed duration.

```
public double get (boolean dstRate, TimeUnit dstUnit)
```

Returns the rate or time extracted from this parameter, expressed in `dstUnit`. If `dstRate` is `true`, the returned value corresponds to a rate. Otherwise, it corresponds to a time.

If `getTimeUnit()` returns `null`, the method returns `getValue()` unchanged. Otherwise, if `getRateIndicator()` returns `true`, `1/getValue()` is used instead of `getValue()`. The method uses `TimeUnit.convert (double, TimeUnit, TimeUnit)` to convert the value from source unit `getTimeUnit()` to destination unit `dstUnit`. If `dstRate` is `true`, one over the converted value is returned. Otherwise, the converted value is returned.

Parameters

`dstRate` determines if the returned value must be a rate.

`dstUnit` the destination time unit.

Returns the computed duration or rate.

```
public static TimeParam valueOf (String str)
```

Constructs a new time parameter from the string `str`, using `TimeParam (String)` and returns the constructed instance.

Parameter

`str` the string representation of the time parameter.

Returns the constructed time parameter.

Throws

`IllegalArgumentException` if the string is invalid.

`public String toString()`

Formats this time parameter as a string. The returned string can be used by `TimeParam (String)` or `valueOf (String)` to construct a time parameter.

Returns the time parameter, formatted as a string.

`public boolean isAttributeSupported (String a)`

For internal use only.

`public void nestedText (String str)`

For internal use only.

SourceArray2D

Represents a 2D array obtained from a data source such a text file, or a database. Such a source array can be used to create 1D or 2D arrays. Any implementation of this interface must be initialized through the `init()` method before elements can be extracted from the source array. The `rows()`, `columns (int)`, and `get (Class, int, int)` methods can then be used to inspect the array.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public interface SourceArray2D
```

Methods

```
public int rows()
```

Returns the number of rows in the source array.

Returns the number of rows in the array.

Throws

`IllegalStateException` if the array was not initialized.

```
public int columns (int row)
```

Returns the number of columns in row `row` of the source array.

Parameter

`row` the row to test.

Returns the number of columns in the row.

Throws

`IllegalStateException` if the source array was not initialized.

`IllegalArgumentException` if the row index is out of bounds.

```
public <T> T get (Class<T> pcls, int row, int column) throws  
                UnsupportedOperationException
```

Returns the element at row `row` and column `column` of the source array, converted to class `pcls`.

Type parameter

`T` the target class.

Parameters

`pcls` the target class.

`row` the row index.

`column` the column index.

Returns the element.

Throws

`IllegalArgumentException` if the row or column indices are out of bounds.

`IllegalStateException` if the array was not initialized.

`ClassCastException` if the element cannot be converted to the target class.

```
public void init()
```

Initializes the source array.

Throws

`IllegalStateException` if the source array cannot be initialized.

```
public void dispose()
```

Clears the data in the source array.

```
public String getElementName()
```

Returns the name of the XML element representing the type of source array implemented.

Returns the name of the XML representing the array type.

CSVSourceArray2D

Represents a source array whose contents is read from a CSV-formatted text file. Each line of the text file pointed to by a URL becomes a row of the source array, with elements of the row separated using commas. Text is read using `TextDataReader.readCSVData (URL, char, char)` with `,` as the column delimiter and `"` as the string delimiter, while `StringConvert.fromString (URI, ClassFinder, Class, String)` is used to convert strings to target objects.

In the XML file, the `URL` attribute of an element representing a CSV source array must be used to indicate the URL of the CSV data file.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class CSVSourceArray2D extends AbstractParam
    implements SourceArray2D, Cloneable, StorableParam
```

Methods

```
public URL getURL()
```

Returns the URL of the text file containing the values of the array, in CSV format.

Returns the URL of the CSV data.

```
public void setURL (URL url)
```

Sets the URL pointing to the CSV file containing the elements of this array to `url`.

Parameter

`url` the URL of the data.

ExcelSourceArray2D

Represents a source array whose contents is read from a Microsoft Excel workbook. Each row of the sheet with the given name, and contained in the workbook pointed to by a URL becomes a row of the source array, with elements of the row separated using commas. Data is read using JExcel API.

In the XML file, the `URL` attribute of an element representing a Excel source array must be used to indicate the URL of the data file. The `sheetName` attribute can also be used to indicate the name of a sheet. By default, the first sheet is read.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class ExcelSourceArray2D extends AbstractParam
    implements SourceArray2D, Cloneable, StorableParam
```

Methods

```
public URL getURL()
```

Returns the URL of the text file containing the values of the array, in CSV format.

Returns the URL of the CSV data.

```
public void setURL (URL url)
```

Sets the URL pointing to the CSV file containing the elements of this array to `url`.

Parameter

`url` the URL of the data.

```
public String getSheetName()
```

Returns the name of the sheet to extract data from.

Returns the name of the sheet.

```
public void setSheetName (String sheetName)
```

Sets the sheet name to `sheetName`.

Parameter

`sheetName` the sheet name.

DBSourceArray2D

Represents a source array whose data is extracted from a database using JDBC. The elements of the array are obtained by performing a query on a database. Each row of the resulting result set is a row in the source array, while each column corresponding to a field of the result set becomes a column in the array. The JDBC connection is initialized using `DBConnectionParam`, and the result set is converted into an array of objects using `JDBCManager.readObjectData2D (Connection, String)`. Any numeric object (instances of `Number`) is converted to the target class while other objects not corresponding to the target class are converted to string before they are passed to `StringConvert.fromString (URI, ClassFinder, Class, String)`.

In a XML file, the `dataQuery` attribute of an element representing a database-based source array is used to specify the query on the database. The `database` nested element is then used to describe the connection to the database.

```
package umontreal.iro.lecuyer.xmlconfig;

public class DBSourceArray2D extends AbstractParam
    implements SourceArray2D, Cloneable, StorableParam
```

Methods

```
public DBConnectionParam getDatabase()
```

Returns the parameters of the database connection used to obtain data for this source array.

Returns the database connection parameters.

```
public void setDatabase (DBConnectionParam dbParams)
```

Sets the parameters of the database connection to `dbParams`.

Parameter

`dbParams` the parameters for the database connection.

```
public String getDataQuery()
```

Returns the SQL query used to obtain data for the element in the source array. The query, e.g., `SELECT Column FROM Table`, is made on the database whose parameters are given by `getDatabase()`.

Returns the SQL query for data.

```
public void setDataQuery (String dataQuery)
```

Sets the SQL query for data to `dataQuery`.

Parameter

`dataQuery` the SQL query for data.

`public void init()`

Initializes the source array by performing the SQL query. This method establishes the connection to the database using JDBC, issues the SQL query, and copies the elements of the result set in an internal 2D array. The resulting internal array, which is the contents of the source array, is a snapshot of the result of the query; it is not updated automatically if the database changes.

`public void addDatabase (DBConnectionParam dbParams1)`

For internal use only.

SourceSubset2D

Represents a source subset obtained from a source array. Such a 2D array is obtained by taking a subset of the rows and the columns of another source array.

```
package umontreal.iro.lecuyer.xmlconfig;  
  
public class SourceSubset2D implements SourceArray2D
```

Constructor

```
public SourceSubset2D (SourceArray2D sourceArray, int fromRow, int  
                      fromColumn, int numRows, int numColumns, boolean  
                      transposed)
```

Constructs a new subset from the array `sourceArray`.

Parameters

`sourceArray` the original source array.

`fromRow` the starting row in the original array.

`fromColumn` the starting column in the original array.

`numRows` the number of rows in the subset.

`numColumns` the number of columns in the subset.

`transposed` determines if the subset needs to be transposed.

Throws

`IllegalArgumentException` if one or more arguments are negative or out of bounds.

`IllegalStateException` if `sourceArray` is not initialized.

ParamWithSourceArray

Represents a parameter object whose contents can be extracted from a source subset created from a source array. The source array is created by reading data from an external source such as a CSV file or a database. The source subset is then constructed by taking a possibly transposed portion of the source array.

In an XML element whose parameters are extracted from a source array, the source array is described using the `CSV` or `DB` subelements. The `CSV` element, represented by `CSVSourceArray2D`, takes a single `URL` attribute pointing to a CSV-formatted text file containing the data. The `DB` element, represented by `DBSourceArray2D`, requires a `dataQuery` attribute giving the query to perform on a database described by a `database` subelement.

A single source array can contain information for multiple destination arrays. For example, a spreadsheet containing one column of arrival rates for each call type might be exported to CSV, and used as a source array. Therefore, facilities are provided to subset the source array. For this, the `startingRow`, `startingColumn`, `numRows`, and `numColumns` attributes of `a` can be used to indicate the portion of the source array to consider. The first two attributes give the (zero-based) starting row and column of the subset. These are optional and defaults to 0. The last two attributes give the dimensions of the subset. If the number of rows [columns] is omitted, it defaults to the number of rows [columns] in the source array minus the starting row [column]. If the resulting subarray is rectangular, it can finally be transposed by using the `transposed` boolean attribute. This latter boolean attribute is also optional, and defaults to `false`, i.e., no transposition.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class ParamWithSourceArray extends AbstractParam
    implements StorableParam, Cloneable
```

Methods

```
public SourceArray2D getDataMatrix()
```

Returns the matrix from which the data is extracted.

Returns the matrix to extract data from.

```
public void setDataMatrix (SourceArray2D dataMatrix)
```

Sets the matrix to extract data from to `dataMatrix`.

Parameter

`dataMatrix` the new matrix to extract data from.

```
public boolean isTransposed()
```

Determines if the matrix of data must be transposed before data is extracted.

Returns the status of the transpose indicator.

```
public void setTransposed (boolean transposed)
```

Sets the matrix transposition indicator to `transposedDataMatrix`.

Parameter

`transposed` the new value of the indicator.

```
public int getStartingRow()
```

Determines the first row to read in the matrix.

Returns the starting row.

```
public void setStartingRow (int startingRow)
```

Sets the starting row to extract external data from to `startingRow`.

Parameter

`startingRow` the starting row.

```
public int getNumRows()
```

Returns the number of rows to be extracted when the array comes from an external source. If the number of rows is set to `Integer.MAX_VALUE`, all available rows, starting from `getStartingRow()` are extracted. The default number of rows is `Integer.MAX_VALUE`.

Returns the number of rows in the array.

```
public void setNumRows (int numRows)
```

Sets the number of rows to be extracted from an external source to construct this array to `numRows`

Parameter

`numRows`

```
public int getStartingColumn()
```

Determines the first column to read in the matrix.

Returns the starting column.

```
public void setStartingColumn (int startingColumn)
```

Sets the starting column to extract external data from to `startingColumn`.

Parameter

`startingColumn` the starting column.

```
public int getNumColumn()
```

Returns the number of columns to be extracted when the array comes from an external source. If the number of columns is set to `Integer.MAX_VALUE`, all available columns, starting from `getStartingColumn()` are extracted. The default number of rows is `Integer.MAX_VALUE`.

Returns the number of columns in the array.

```
public void setNumColumns (int numColumns)
```

Sets the number of columns to be extracted from an external source to construct this array to `numColumns`

Parameter

`numColumns` the number of columns.

```
public void initSourceArray()
```

Initializes the source array associated with this object.

```
public void disposeSourceArray()
```

Clears the data in the source array.

```
public SourceArray2D getSourceSubset()
```

Returns the source subset used to extract data.

Returns the source subset.

ArrayParam

Represents a parameter object containing an array of parameters. When a creator method constructs this parameter object, it needs to indicate the class of the components in the array. The component class can be any non-array primitive type or class supported by `StringConvert`. It is recommended to use this class as a temporary placeholder. A creator method constructs and returns an instance whereas an adder method receives the configured instance and extracts the array, without having to keep the parameter object.

In the XML file, two formats are supported for the array parameter. A list of comma-separated strings that will be converted to objects of the component class can be used as contents. Alternatively, the array can be encoded using a `row` nested element for each element. The `row` element supports the `repeat` attribute allowing the specification of the number of times an element must be repeated in the array.

For example,

```
<doublearray>2.3, 3.2, 3.2, 5.1</doublearray>
```

is equivalent to

```
<doublearray>
  <row>2.3</row>
  <row repeat="2">3.2</row>
  <row>5.1</row>
</doublearray>
```

and can be converted into an array containing 2.3, 3.2, 3.2, and 5.1.

Arrays can also be specified externally by using the CSV or DB sub-elements. See `ParamWithSourceArray` for more information. The constructed source subset is read row by row to obtain an array of objects.

This class provides methods to get the extracted array as well as methods for the primitive types.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class ArrayParam extends ParamWithSourceArray
    implements Cloneable, StorableParam
```

Constructor

```
public ArrayParam (Class<?> componentClass, String elementName)
    Constructs a new array parameter with components of class componentClass.
```

Parameter

`componentClass` the component class.

Throws

`NullPointerException` if `componentClass` is `null`.

`IllegalArgumentException` if `componentClass` is an array class.

Methods

```
public Class<?> getComponentClass()
```

Returns the class of the components in this array parameter.

Returns the component class.

```
public Object[] getValues()
```

Returns the associated array of objects. The returned array can safely be casted to an array of the component class.

Returns the associated array of objects.

```
public float[] getFloatValues()
```

Converts the objects returned by `getValues()` to `float` and returns the resulting array.

Returns the array of single-precision values.

Throws

`IllegalStateException` if the component class is incompatible.

```
public double[] getDoubleValues()
```

Converts the objects returned by `getValues()` to `double` and returns the resulting array.

Returns the array of double-precision values.

Throws

`IllegalStateException` if the component class is incompatible.

```
public int[] getIntValues()
```

Converts the objects returned by `getValues()` to `int` and returns the resulting array.

Returns the array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public short[] getShortValues()
```

Converts the objects returned by `getValues()` to `short` and returns the resulting array.

Returns the array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public long[] getLongValues()
```

Converts the objects returned by `getValues()` to `long` and returns the resulting array.

Returns the array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public byte[] getByteValues()
```

Converts the objects returned by `getValues()` to `byte` and returns the resulting array.

Returns the array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public char[] getCharValues()
```

Converts the objects returned by `getValues()` to `char` and returns the resulting array.

Returns the array of characters.

Throws

`IllegalStateException` if the component class is incompatible.

```
public boolean[] getBooleanValues()
```

Converts the objects returned by `getValues()` to `boolean` and returns the resulting array.

Returns the array of booleans.

Throws

`IllegalStateException` if the component class is incompatible.

```
public void setValues (Object[] v)
```

Sets the array elements to `v`. If the component type of `v` is not assignable to the component class, this method throws an `IllegalArgumentException`.

Parameter

`v` the values.

Throws

`NullPointerException` if `v` is `null`.

`IllegalArgumentException` if the component class is incompatible with `v`.

```
public boolean isAttributeSupported (String a)
```

For internal use only.

```
public void nestedText (ParamReader reader, String str)
```

For internal use only.

```
public void defaultNestedElement (ParamReader reader, Element cel)
```

For internal use only.

Nested class

```
public static final class RowParam implements Param, Cloneable
```

For internal use only.

ArrayParam2D

Represents a parameter object containing a 2D array or a matrix of parameters. When a creator method constructs such a parameter object, it needs to specify a component class for the elements in the 2D array. The component class can be any non-array primitive type or class supported by `StringConvert`. It is recommended to use this class as a temporary placeholder. A creator method constructs and returns an instance whereas an adder method receives the configured instance and extracts the array, without having to keep the parameter object.

In the XML file, two formats are allowed to represent this parameter. The 2D array can be specified as a list of arrays or one `row` element can be used for each row of the 2D array. As with `ArrayParam`, the `row` elements supports the `repeat` attribute.

For example,

```
<matrix>
  {1, 2,3},
  {4, 5},
  {4, 5},
  {4, 5},
  {4, 5},
  {9, 11,13}
</matrix>
```

is equivalent to

```
<matrix>
  <row>1,2,3</row>
  <row repeat="4">4, 5</row>
  <row> 9 , 11, 13</row>
</matrix>
```

Matrices can also be specified externally by using the CSV or DB sub-elements. See `ParamWithSourceArray` for more information.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class ArrayParam2D extends ParamWithSourceArray
    implements Cloneable, StorableParam
```

Constructor

```
public ArrayParam2D (Class<?> componentClass)
    Constructs a new 2D array parameter with components of class componentClass.
```

Parameter

`componentClass` the component class.

Throws

`NullPointerException` if `componentClass` is `null`.

`IllegalArgumentException` if `componentClass` is an array class.

Methods

```
public Class<?> getComponentClass()
```

Returns the class of the components in this array parameter.

Returns the component class.

```
public Object[][] getValues()
```

Returns the values in the 2D array represented by this parameter object. The returned 2D array can safely be casted to a 2D array of the component class.

Returns the represented 2D array.

Throws

`ParamReadException` if no 2D array was specified.

```
public float[][] getFloatValues()
```

Converts the objects returned by `getValues()` to `float` and returns the resulting 2D array.

Returns the 2D array of single-precision values.

Throws

`IllegalStateException` if the component class is incompatible.

```
public double[][] getDoubleValues()
```

Converts the objects returned by `getValues()` to `double` and returns the resulting 2D array.

Returns the 2D array of double-precision values.

Throws

`IllegalStateException` if the component class is incompatible.

```
public int[][] getIntValues()
```

Converts the objects returned by `getValues()` to `int` and returns the resulting 2D array.

Returns the 2D array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public short[] [] getShortValues()
```

Converts the objects returned by `getValues()` to `short` and returns the resulting 2D array.

Returns the 2D array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public long[] [] getLongValues()
```

Converts the objects returned by `getValues()` to `long` and returns the resulting 2D array.

Returns the 2D array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public byte[] [] getByteValues()
```

Converts the objects returned by `getValues()` to `byte` and returns the resulting 2D array.

Returns the 2D array of integers.

Throws

`IllegalStateException` if the component class is incompatible.

```
public char[] [] getCharValues()
```

Converts the objects returned by `getValues()` to `char` and returns the resulting 2D array.

Returns the 2D array of characters.

Throws

`IllegalStateException` if the component class is incompatible.

```
public boolean[] [] getBooleanValues()
```

Converts the objects returned by `getValues()` to `boolean` and returns the resulting 2D array.

Returns the 2D array of booleans.

Throws

`IllegalStateException` if the component class is incompatible.

```
public void setValues (Object[] [] m)
```

Sets the 2D array to `m`. If the component type of `m` is not assignable to the component class, this method throws an `IllegalArgumentException`.

Parameter

`m` the new 2D array.

Throws

`NullPointerException` if `m` is null.

`IllegalArgumentException` if the component class is incompatible with `m`.

```
public boolean isAttributeSupported (String a)
```

For internal use only.

```
public MatrixRowParam createRow()
```

For internal use only.

```
public void nestedText (ParamReader reader, String str)
```

For internal use only.

Nested class

```
public static final class MatrixRowParam extends ArrayParam
```

For internal use only.

DBConnectionParam

Represents the parameters for a database connection established using JDBC. A connection can be established using a JNDI name corresponding to a `DataSource`, or using a JDBC URI intended for the `DriverManager`. Connection information is given using attributes. For example, the following code gives parameters for a JDBC connection using MySQL.

```
...
<db jdbcDriverClass="com.mysql.jdbc.Driver"
    jdbcURI="jdbc:mysql://mysql.iro.umontreal.ca/database">
    <property name="user" value="foo"/>
    <property name="password" value="bar"/>
</db>
```

```
package umontreal.iro.lecuyer.xmlconfig;

public class DBConnectionParam extends AbstractParam
    implements StorableParam, Cloneable
```

Constructors

```
public DBConnectionParam()
```

Nullary constructor for the parameter reader.

```
public DBConnectionParam (String jndiDataSourceName)
```

Constructs parameters for a JDBC connection using a data source obtained via JNDI, with name `jndiDataSourceName`.

Parameter

`jndiDataSourceName` the name of the data source.

Throws

`NullPointerException` if `jndiDataSourceName` is null.

```
public DBConnectionParam (Class<? extends Driver> jdbcDriverClass, URI
    jdbcURI)
```

Constructs parameters for a JDBC connection using the driver manager, with URI `jdbcURI`, and driver class `jdbcDriverClass`.

Parameters

`jdbcDriverClass` the driver class, can be null.

`jdbcURI` the JDBC URI.

Throws

`NullPointerException` if `jdbcURI` is `null`.

Methods

`public Connection createConnection() throws SQLException`

Creates the database connection from the parameters stored in this object. This method first constructs a `Properties` object from the properties in this object. If a JNDI name is specified, these properties are used as an environment for the `InitialContext` constructor, the constructed context is used to look for a data source, and the connection is obtained. Otherwise, the driver is loaded if its class is not `null`, and the connection is established using the URI and properties.

Returns the established database connection.

Throws

`SQLException` if a connection error occurred.

`public Class<? extends Driver> getJdbcDriverClass()`

Returns the JDBC driver class.

Returns the JDBC driver class.

`public void setJdbcDriverClass (Class<? extends Driver> jdbcDriverClass)`

Sets the JDBC driver class to `jdbcDriverClass`.

Parameter

`jdbcDriverClass` the new JDBC driver class.

Throws

`IllegalArgumentException` if the driver class is non-null, and does not implement the `Driver` interface.

`public URI getJdbcURI()`

Returns the JDBC URI.

Returns the JDBC URI.

`public void setJdbcURI (URI jdbcURI)`

Sets the JDBC URI to `jdbcURI`.

Parameter

`jdbcURI` the new JDBC URI.

`public String getJndiDataSourceName()`

Returns the JNDI name of the data source that will be used to obtain the connection.

Returns the JNDI name of the data source.

```
public void setJndiDataSourceName (String jndiDataSourceName)
```

Sets the JNDI name of the data source to `jndiDataSourceName`.

Parameter

`jndiDataSourceName`

```
public Set<PropertyParam> getProperties()
```

Returns the properties used by this parameter object. The properties are used as an environment for configuring JNDI, or as parameters to the driver manager, depending on how the connection is established.

Returns the connection properties.

```
public void addProperty (PropertyParam p)
```

Adds a new connection property.

Parameter

`p` the new property.

PropertyParam

Represents a property, i.e., a name-value pair.

```
package umontreal.iro.lecuyer.xmlconfig;

public class PropertyParam extends AbstractParam
    implements StorableParam, Cloneable
```

Constructors

```
public PropertyParam()
```

Nullary constructor for the parameter reader.

```
public PropertyParam (String name, String value)
```

Constructs a new property with name `name`, and value `value`.

Parameters

`name` the name.

`value` the value.

Throws

`NullPointerException` if `name` is null.

Methods

```
public String getName()
```

Returns the name of this property.

Returns the name of the property.

```
public void setName (String name)
```

Sets the name of the property to `name`.

Parameter

`name` the new name of the property.

Throws

`NullPointerException` if `name` is null.

```
public String getValue()
```

Returns the value of this property.

Returns the value of this property.

```
public void setValue (String value)
```

Sets the value of this property to `value`.

Parameter

`value` the value of this property.

Throws

`NullPointerException` if `value` is `null`.

RandomVariateGenParam

Stores the parameters of a probability distribution to create the distribution object or a matching random variate generator at a later time. Three parameters need to be defined to use this object: the class of the probability distribution, a string defining the parameters as given to the distribution class' constructor, and the class of the random variate generator. Alternatively, the parameters of the distribution can be replaced by an array of values for parameters to be estimated by maximum likelihood. One can also specify a location parameter ℓ for the generated values. This parameter, which defaults to 0, is subtracted from any generated value and added to every value specified as data to estimate parameters. It does not affect explicitly-specified parameters.

The `createDistribution()` and `createDistributionInt()` methods can be used to construct a distribution object whereas the `createGenerator` (`RandomStream`) and `createGeneratorInt` (`RandomStream`) methods construct a generator.

In an XML element, the `distributionClass` attribute is required to correspond to the name of a class implementing the `Distribution` interface. The nested text of the element is used as an array of parameters the class tries to pass to a constructor inside the distribution class. To use MLE, one must specify the data in a `data` element or `dataURL` attribute. `data` is an array of double-precision values while `dataURL` points to a resource containing the data, one value per line. Note that the file is searched in the same directory as the XML parameter file. For discrete distributions on the integers, the given values are rounded to the nearest integers.

The `data` element or the `dataFile` attribute can also be used for empirical distributions. More specifically, when `distributionClass` is set to `EmpiricalDist` or `PiecewiseLinearEmpiricalDist`, the given data is sorted and used directly by these distributions. The shift ℓ is given via the `shift` attribute.

The class of the random variate generator can optionally be changed by providing a compatible class name in the `generatorClass` attribute. This allows the method for generating the variates to be selected from a parameter file.

For example, the element

```
<?import umontreal.iro.lecuyer.probdist.*?>
<?import umontreal.iro.lecuyer.randvar.*?>
...
  <rvg distributionClass="GammaDist"
    generatorClass="GammaAcceptanceRejectionGen">
    32.3, 25.2
  </rvg>
```

can be mapped to a gamma distribution with $\alpha = 32.3$ and $\lambda = 25.2$ and a gamma variate generator using acceptance-rejection.

```
package umontreal.iro.lecuyer.xmlconfig;

public class RandomVariateGenParam extends AbstractParam
    implements StorableParam, Cloneable
```

Constructors

```
public RandomVariateGenParam()
```

Default constructor for parameter reader.

```
public RandomVariateGenParam (Class<? extends Distribution> distClass,
                               String params)
```

Constructs a new distribution parameter object with distribution class `distClass` and parameter string `params`. The given distribution class must be a subclass of `Distribution` or `DiscreteDistributionInt` and the parameter string must correspond to the arguments given to the constructor of the distribution class, without the parentheses. Arrays can be given as arguments by surrounding them with braces.

Parameters

`distClass` the class of the represented distribution.

`params` the parameters passed to the constructor.

Throws

`NullPointerException` if `distClass` is null.

`IllegalArgumentException` if the given class is not a subclass of `Distribution` or `DiscreteDistributionInt`.

```
public RandomVariateGenParam (Distribution dist)
```

Constructs a new random variate generator parameter object for the distribution `dist`.

Parameter

`dist` the probability distribution.

Methods

```
public boolean isDiscreteDistributionInt()
```

Determines if the associated distribution class extends `DiscreteDistributionInt`. If this is the case, returns `true`. Otherwise, returns `false`. If the distribution class is not set, i.e., `getDistributionClass()` returns null, an `IllegalStateException` is thrown.

Returns `true` if and only if a discrete distribution of integer is associated with this parameter object.

Throws

`IllegalStateException` if the distribution class was not set up.

```
public Class<? extends Distribution> getDistributionClass  
( )
```

Returns the class of distribution object contained in this parameter. If the distribution was not set, this returns `null`.

Returns the class of distribution object.

```
public void setDistributionClass (Class<? extends Distribution> distClass)
```

Sets the class of distribution object to `distClass`. This method resets the random variate generator class to the default value as specified in `getGeneratorClass()`, and the distribution parameters to `null`.

Parameter

`distClass` the new distribution class.

Throws

`NullPointerException` if `distClass` is `null`.

`IllegalArgumentException` if the given class does not implement `Distribution` or extend `DiscreteDistributionInt`.

```
public double getShift()
```

Returns the shift ℓ applied to all generated values.

Returns the shift being applied.

```
public void setShift (double shift)
```

Sets the shift ℓ being applied to all generated values to `shift`.

Parameter

`shift` the new value of the shift.

```
public String getDistributionParameters()
```

Returns the parameters associated with the distribution. This corresponds to a comma-separated list of parameters as given to the constructor of the distribution class, without the parentheses.

Returns the parameters of the distribution.

```
public void setDistributionParameters (String params)
```

Sets the distribution parameters to `params`.

Parameter

`params` the distribution parameters.

```
public double[] getData()
```

Returns the data used to estimate the parameters of the selected distribution. Note that this data is ignored if parameters are specified directly (`getDistributionParameters()` returns a non-empty string).

Returns the data for estimating the parameters.

```
public void setData (double[] data)
```

Sets the data used for estimating the parameters of the selected distribution to `data`.

Parameter

`data` the data used for parameter estimation.

```
public Class<? extends RandomVariateGen> getGeneratorClass  
( )
```

Returns the class of the random variate generator associated with this object. The default class is `RandomVariateGen` for a `Distribution` and `RandomVariateGenInt` for a `DiscreteDistributionInt`. This has a non-null value only after the distribution class is set up, using a constructor, an XML element, or `setDistributionClass (Class)`.

Returns the class of random variate generator.

```
public void setGeneratorClass (Class<? extends RandomVariateGen> genClass)
```

Sets the class of random variate generator to `genClass`. This method cannot be called until a distribution class is set up.

Parameter

`genClass` the new random variate generator class.

Throws

`NullPointerException` if `genClass` is null.

`IllegalArgumentException` if the given class does not extend `RandomVariateGen`.

`IllegalStateException` if a distribution class was not set up.

```
public Distribution createDistribution()
```

Returns the new probability distribution extracted from the parameters. This method can be called only if `isDiscreteDistributionInt()` returns `false`.

Returns the extracted probability distribution.

Throws

IllegalStateException if the distribution class is not set or corresponds to a **DiscreteDistributionInt** subclass.

DistributionCreationException if the distribution cannot be created successfully.

```
public DiscreteDistributionInt createDistributionInt()
```

Returns the new discrete probability distribution extracted from the parameters. This method can be called only if **isDiscreteDistributionInt()** returns **true**.

Returns the extracted probability distribution.

Throws

IllegalStateException if the distribution class is not set or corresponds to a **Distribution** implementation.

DistributionCreationException if the distribution cannot be created successfully.

```
public void setDistribution (Distribution dist)
```

Sets the probability distribution to **dist**. This resets the generator class to **RandomVariateGen**.

Parameter

dist the new distribution.

```
public TimeUnit getUnit()
```

Returns the time unit in which the values produced by the probability distribution are expressed. If this is set to **null** (the default), no time conversion is performed.

Returns the distribution's time unit.

```
public void setUnit (TimeUnit unit)
```

Sets the distribution's time unit to **unit**.

Parameter

unit the new distribution's time unit.

See also **getUnit()**

```
public void setConstant (double c)
```

Sets the current distribution to a constant value **c**. This is a special case of a discrete distribution with a single observation **c** having probability 1.

Parameter

c the constant returned by the produced generator.

```
public double getMean()
```

Returns the mean value for the current distribution. This method calls **createDistribution().getMean**

Returns the mean.

```
public double getMean (TimeUnit targetUnit)
```

Returns the mean for the current distribution, expressed with the time unit `targetUnit`. This method returns the same value as `getMean()` if `getUnit()` returns `null` or `targetUnit` is `null`.

Parameter

`targetUnit` the target time unit.

Returns the mean, possibly converted to the target time unit.

```
public double getVariance()
```

Returns the variance for the current distribution. This method calls `createDistribution().getVariance()`.

Returns the mean.

```
public double getExpLambda()
```

Returns the λ parameter for the associated exponential distribution. If the distribution class is not set up or not exponential, this throws an `IllegalStateException`. Otherwise, the method tries to create the exponential distribution object and returns the corresponding λ parameter.

Returns the λ parameter.

Throws

`IllegalStateException` if the distribution is incompatible.

```
public void setExpLambda (double lambda)
```

Sets the distribution class to exponential and sets the λ parameter to `lambda`.

Parameter

`lambda` the λ parameter for the exponential distribution.

Throws

`IllegalArgumentException` if `lambda` is negative or 0.

```
public double getGammaAlpha()
```

Returns the α parameter for the associated gamma distribution. If the distribution class is not set up or not gamma, this throws an `IllegalStateException`. Otherwise, the method tries to create the gamma distribution object and returns the corresponding γ parameter.

Returns the α parameter.

Throws

`IllegalStateException` if the distribution is incompatible.

```
public double getGammaLambda()
```

Returns the λ parameter for the associated gamma distribution. If the distribution class is not set up or not gamma, this throws an `IllegalStateException`. Otherwise, the method tries to create the gamma distribution object and returns the corresponding λ parameter.

Returns the λ parameter.

Throws

`IllegalStateException` if the distribution is incompatible.

```
public void setGammaParams (double alpha, double lambda)
```

Sets the distribution class to gamma and sets the α and λ parameters to `alpha` and `lambda`, respectively.

Parameters

`alpha` the α parameter for the gamma distribution.

`lambda` the λ parameter for the gamma distribution.

Throws

`IllegalArgumentException` if `alpha` or `lambda` are negative or 0.

```
public RandomVariateGen createGenerator (RandomStream stream)
```

Constructs a new random variate generator from the distribution information associated with this object and the stream `stream`. This method constructs a random variate generator from the class given by `getGeneratorClass()` by selecting an appropriate constructor. The method only uses constructors taking a random stream and a probability distribution. Other constructors from a random variate generator are ignored. If the parameter object contains information about a discrete distribution of integers, a `RandomVariateGenInt` is constructed.

Parameter

`stream` the random stream used to generate uniforms.

Throws

`DistributionCreationException` if the probability distribution could not be created successfully.

`GeneratorCreationException` if the generator cannot be created.

`IllegalStateException` if some distribution or generator parameters are missing or invalid.

```
public RandomVariateGenInt createGeneratorInt (RandomStream stream)
```

Constructs a new integer random variate generator from the distribution information associated with this object and the stream `stream`. This is similar to `createGenerator (RandomStream)`, except it returns an integer random variate generator. An `IllegalStateException` is thrown if the parameter object contains information about a `Distribution` implementation.

Parameter

`stream` the random stream used to generate uniforms.

Throws

`DistributionCreationException` if the probability distribution could not be created successfully.

`GeneratorCreationException` if the generator cannot be created.

`IllegalStateException` if some distribution or generator parameters are missing or invalid, or if the associated parameters correspond to an incompatible distribution.

```
public boolean isAttributeSupported (String a)
```

For internal use only.

```
public void nestedText (ParamReader reader, String par)
```

For internal use only.

```
public void estimateParameters (boolean clearData)
```

Uses data obtained by `getData()` to estimate the parameters of the distribution with class `getDistributionClass()`, and stores the estimation in the distribution's parameters returned by `getDistributionParameters()`. This method does nothing if `getDistributionClass()` returns `null` or a class corresponding to an empirical distribution. Otherwise, it tries to call a static method `getMaximumLikelihoodEstimate (double[], int)` or `getMaximumLikelihoodEstimate (int[], int)` (for discrete distributions over the integers) with the values in the array returned by `getData()`. The resulting array is converted into a string assuming that an appropriate constructor exists in the distribution class. If `clearData` is `true`, the references to the data and the data file for this object are set to `null` after parameter estimation.

Parameter

`clearData` determines if the data bound to this object must be discarded.

```
public ArrayParam createData()
```

For internal use only.

```
public void addData (ArrayParam p)
```

For internal use only.

DistributionCreationException

This exception is thrown when a problem occurs during the construction of a distribution object by `RandomVariateGenParam.createDistribution()` or `RandomVariateGenParam.createDistributionInt()`, or during the parsing of the nested text containing a distribution XML element.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class DistributionCreationException extends RuntimeException
```

Constructors

```
public DistributionCreationException()
```

Constructs a new distribution creation exception with no distribution information or message.

```
public DistributionCreationException (String message)
```

Constructs a new distribution creation exception with no distribution information and message `message`.

Parameter

`message` the message describing the exception.

```
public DistributionCreationException (Class<? extends Distribution>  
                                     distClass, String distParams)
```

Constructs a new distribution creation exception with distribution class `distClass`, distribution parameters `distParams`, and no message.

Parameters

`distClass` the class of the distribution which cannot be created.

`distParams` the parameters given to the constructor of the distribution class.

```
public DistributionCreationException (Class<? extends Distribution>  
                                     distClass, String distParams, String  
                                     message)
```

Constructs a new distribution creation exception with distribution class `distClass`, distribution parameters `distParams`, and message `message`.

Parameters

`distClass` the class of the distribution which cannot be created.

`distParams` the parameters given to the constructor of the distribution class.

`message` the message describing the exception.

Methods

```
public Class<? extends Distribution> getDistributionClass()  
( )
```

Returns the distribution class which caused the exception.

Returns the distribution class having caused the exception.

```
public String getDistributionParameters()
```

Returns the distribution parameters for which there is no corresponding constructor in the distribution class, or an exception occurred during the call to a constructor.

Returns the distribution parameters having caused the exception.

```
public String toString()
```

Returns a short description of this exception. If no distribution class and parameters are associated with this exception, this method returns the result of the superclass's `toString` method. Otherwise, it returns a string with the following contents.

- The name of this class
- ": For "
- The name of the distribution class if available
- If the parameters are available, "(", parameters, ")"
- If a message is given, ", " followed by the message string.

Returns the short string describing the exception.

GeneratorCreationException

This exception is thrown when a problem occurs during the construction of a random variate generator by `RandomVariateGenParam.createGenerator (RandomStream)` or `RandomVariateGenParam.createGeneratorInt (RandomStream)`.

```
package umontreal.iro.lecuyer.xmlconfig;

public class GeneratorCreationException extends RuntimeException
```

Constructors

```
public GeneratorCreationException()
```

Constructs a new generator creation exception with no generator information or message.

```
public GeneratorCreationException (String message)
```

Constructs a new generator creation exception with no generator information and message `message`.

Parameter

`message` the message describing the exception.

```
public GeneratorCreationException (Class<? extends Distribution> distClass,
                                   String distParams, Class<? extends
                                   RandomVariateGen> genClass)
```

Constructs a new generator creation exception with distribution class `distClass`, distribution parameters `distParams`, generator class `genClass`, and no message.

Parameters

`distClass` the class of the distribution associated with the generator..

`distParams` the parameters given to the constructor of the distribution class.

`genClass` the class of the random variate generator that cannot be created.

```
public GeneratorCreationException (Class<? extends Distribution> distClass,
                                   String distParams, Class<? extends
                                   RandomVariateGen> genClass, String
                                   message)
```

Constructs a new generator creation exception with distribution class `distClass`, distribution parameters `distParams`, generator class `genClass`, and message `message`.

Parameters

distClass the class of the distribution associated with the generator..

distParams the parameters given to the constructor of the distribution class.

genClass the class of the random variate generator that cannot be created.

message the message describing the exception.

Methods

```
public Class<? extends Distribution> getDistributionClass  
( )
```

Returns the distribution class which caused the exception.

Returns the distribution class having caused the exception.

```
public String getDistributionParameters()
```

Returns the distribution parameters for which there is no corresponding constructor in the distribution class.

Returns the distribution parameters having caused the exception.

```
public Class<? extends RandomVariateGen> getGeneratorClass  
( )
```

Returns the generator class which caused the exception.

Returns the generator class having caused the exception.

```
public String toString()
```

Returns a short description of this exception. If no distribution class, distribution parameters and generator class are associated with this exception, this method returns the result of the superclass's `toString` method. Otherwise, it returns a string with the following contents.

- The name of this class
- ": For "
- If the distribution class or parameters are available
 - "distribution "
 - The name of the distribution class if available
 - If the parameters are available, "(", parameters, ")"
 - If the generator class is available ", "
- If the generator class is given, "generator " followed by the generator class name
- If a message is given, ", " followed by the message string.

Returns the short string describing the exception.

DOMUtils

Provides utility methods to build indented DOM documents intended to be converted to XML files. When using the DOM API to build a document, one must manually insert spaces and end-of-lines, otherwise a one-line XML file will be produced if the document is converted to a stream using a JAXP transformer. JAXP does not give access to a flexible automatic indentation facility when using its transformer factory; parameters for this feature are specific to the XML transformer implementation. This class provides helper methods capable of automatically adding some indentations and easing the constructing of the most common DOM elements.

```
package umontreal.iro.lecuyer.xmlconfig;
```

```
public class DOMUtils
```

Methods

```
public static String getIndent (Node node, int spc)
```

Returns a string of spaces to indent future nested elements of **node**, using **spc** spaces for each indentation level.

If **node** or its owner document are **null** or if the node is a document, the empty string is returned. Otherwise, a string of **spc** spaces is appended to the indentation that would be computed for the parent of **node**.

Parameters

node the node whose children must be indented.

spc the number of spaces for each indentation level.

Returns the string of spaces.

Throws

IllegalArgumentException if **spc** is negative.

```
public static Element addNestedTextElement (Node parent, String name,  
                                             String text, int spc)
```

Adds the nested element with tag name **name** and with nested text **text** to the parent node **parent**. In the parent node, a text node whose contents is generated using **getIndent (Node, int)** is added if **spc** is greater than 0. A new element is constructed and added to the parent. If **spc** is greater than 0, a text node containing a newline character is added to the parent. In the created element a text node containing **text** is added and the element is returned.

Parameters

parent the parent node.

name the tag name of the new element.

text the nested text.

spc the number of spaces for each indentation level.

Returns the newly-created element.

```
public static Element addNestedElement (Node parent, String name, boolean
                                         empty, int spc)
```

This method is similar to `addNestedTextElement (Node, String, String, int)` except that the created element with tag name **name** will not have nested text by default. If **empty** is **true**, no child node is added in the element. Otherwise, a newline is added if **spc** is greater than 0. This newline is added assuming that the new element will contain other elements.

Parameters

parent the parent node.

name the tag name of the new element.

empty if the element will be empty.

spc the number of spaces for each level of indentation.

Returns the created element.

```
public static Comment addNestedComment (Node parent, String text, int spc)
```

Creates a comment node in the element **parent**, with text **text**, using **spc** spaces per indentation level. Before the created and returned comment node, the method adds a text node containing the string returned by `getIndent (Node, int)` to the **parent**. After the comment node, it adds a text node containing a newline if **spc** is greater than 0. In the comment **text**, any spaces or newlines at the beginning and the end of the string is removed. After each newline, the method adds the string returned by `getIndent (Node, int)` folloed by **spc** spaces.

Parameters

parent the parent node.

text the text of the comment.

spc the number of spaces for each indentation level.

Returns the created comment node.

```
public static Element addNestedArrayElement (Node parent, String name,
                                             Object array, String[]
                                             rowNames, int spc)
```

Creates a new element with name `name`, with the contents of the array `array`, and adds it to node `parent`. It outputs an array intended to be read by `ArrayParam`.

The method uses `addNestedElement (Node, String, boolean, int)` to create the element and for each element in the array, it creates a `row` subelement containing the value. Before adding the `row i` using `addNestedTextElement (Node, String, String, int)`, if `rowNames[i]` is non-null, a comment containing `rowNames[i]` is added using `addNestedComment (Node, String, int)`. Elements are extracted from the array using `Array.get (Object, int)` and converted to `String` using `StringConvert.numberToString (Number)` for numeric types, and `Object.toString()` for other types. The given array must not contain `null` values. If an element in the array appears several times consecutively, as tested by `Object.equals (Object)`, a `repeat` attribute is added to the corresponding `row` subelement instead of repeating the value.

Parameters

`parent` the parent node.

`name` the name of the created element.

`array` the array being formatted.

`rowNames` an array containing a name for each array element, or `null`.

`spc` the number of spaces for each indentation level.

Returns the created array element.

Throws

`IllegalArgumentException` if `array` is not an instance of an array class.

```
public static Element addNestedArrayElement (Node parent, String name,
                                             Object array, int spc)
```

Equivalent to `addNestedArrayElement (parent, name, array, null, spc)`. This adds an array element with no comments describing rows.

Parameters

`parent` the parent node.

`name` the name of the created element.

`array` the array being formatted.

`spc` the number of spaces for each indentation level.

Returns the created array element.

Throws

`IllegalArgumentException` if `array` is not an instance of an array class.

```
public static Element addNestedArray2DElement (Node parent, String name,
                                                Object array2D, String[]
                                                rowNames, int spc)
```

Creates a new element with name `name`, with the contents of the 2D array `array2D`, and adds it to node `parent`. It outputs a 2D array intended to be read by `ArrayParam2D`.

The method uses `addNestedElement (Node, String, boolean, int)` to create the element and for each row in the 2D array, it creates a row subelement containing the array values. Before adding the row `i` using `addNestedTextElement (Node, String, String, int)`, if `rowNames[i]` is non-null, a comment containing `rowNames[i]` is added using `addNestedComment (Node, String, int)`. Each row of the 2D array corresponds to a 1D array. Each row element contains a comma-separated list of array values. Each value is formatted with `StringConvert.numberToString (Number)` for numeric types, or `Object.toString()` for other types. If consecutive rows contain the same number of columns and the same elements, as tested with `Object.equals (Object)`, only one row element is added to represent the repeated row, using the `repeat` attribute.

Parameters

`parent` the parent node.

`name` the name of the created element.

`array2D` the 2D array being formatted.

`rowNames` an array containing a name associated with each row, or `null`.

`spc` the number of spaces for each indentation level.

Returns the created matrix element.

Throws

`IllegalArgumentException` if `matrix` is not an instance of a 2D array class.

```
public static Element addNestedArray2DElement (Node parent, String name,
                                                Object array2D, int spc)
```

Equivalent to `addNestedArray2DElement (parent, name, array2D, null, spc)`. This adds a 2D array element with no comments describing rows.

Parameters

`parent` the parent node.

`name` the name of the created element.

`array2D` the 2D array being formatted.

`spc` the number of spaces for each indentation level.

Returns the created matrix element.

Throws

`IllegalArgumentException` if `matrix` is not an instance of a 2D array class.

```
public static void endElement (Element el, int spc)
```

Terminates the nested element `el`. This method uses `getIndent (Node, int)` to add a text node composed of spaces in the element. This allows the closing tag of the element to appear at the same indentation level as the opening tag.

Parameters

`el` the terminated element.

`spc` the number of spaces for each indentation level.

```
public static void unindent (Node node)
```

Suppresses any indenting text node from the DOM node `node`. This recursively removes any child text node containing only newlines and spaces. After the cleanup process, the node is normalized using `Node.normalize()`.

Parameter

`node` the node (or document) being unindented.

```
public static void reindent (Node node, int spc)
```

Adds newlines and whitespaces for the node `node` to be indented in an XML output file. For each child element of `node`, an indent string obtained with `getIndent (node, spc)` is prepended and a newline is appended.

Parameters

`node` the node being reindented.

`spc` the number of spaces for each indentation level.

```
public static String formatNodeName (Node node)
```

Formats a string representing the name of this node `node` in the XML document, in a XPath-like format. If the given node is an attribute, the string `[@name]`, where `name` is the attribute name, is appended to the return value of this method for its owner element. The format of the string for an element is given by `[parent/]tagname(index)`, where `parent` is the result of this method for the parent node, `tagname` is the name of the element and `index` is the return value of `getNodeIndex (Node)`. If the node has no sibling, as tested by `nodeHasSiblings (Node)`, the index as well as the parentheses are omitted.

Parameter

`node` the node name.

Returns the string representation.

```
public static boolean nodeHasSiblings (Node node)
```

Determines if the node `node` has at least one sibling.

Parameter

`node` the tested node.

Returns `true` if the node has at least a previous or a next sibling, `false` otherwise.

```
public static int getNodeIndex (Node node)
```

Returns the index of the child node `node` for its parent.

Parameter

`node` the queried node.

Returns the index of the node.

```
public static String formatNodeValue (Node node, int maxLength)
```

Formats the value of the node `node` with maximal string length `maxLength`. The method first gets the result of `node.getNodeValue()`. If this result is `null`, it returns `null`. Otherwise, if the length of the string is smaller than `maxLength`, the string is returned unchanged. Otherwise, it is truncated to `maxLength` and `...` is appended to the result.

Parameter

`node` the node to be processed.

Returns the formatted value.

Package `umontreal.iro.lecuyer.util`

Contains some utility classes.

ClassFinderWithBase

Extends the `ClassFinder` class to find classes extending a specified base class or implementing a given base interface. The overridden `findClass (String)` method ensures that the class found by `ClassFinder` is assignable to the base class.

Type parameter

`T` the type of the base class.

```
package umontreal.iro.lecuyer.util;
```

```
public class ClassFinderWithBase<T> extends ClassFinder
```

Constructor

```
public ClassFinderWithBase (Class<T> baseClass)
```

Constructs a new class finder with base class `baseClass`.

Parameter

`baseClass` the base class.

Throws

`NullPointerException` if `baseClass` is null.

Method

```
public Class<T> getBaseClass()
```

Returns the base class for any class returned by the `findClass (String)` method.

Returns the base class.

StringConvert

Provides utility methods to convert strings into Java objects. The Java class library already contains some facilities to convert strings to many object types, but the target class must be known at compile time. With these utility methods, a conversion can be performed with a target class known at run time.

```
package umontreal.iro.lecuyer.util;
```

```
public class StringConvert
```

Methods

```
public static <T> T fromString (URI baseURI, ClassFinder finder, Class<T>
                               cls, String val) throws
                               UnsupportedOperationException,
                               NameConflictException
```

Tries to convert the string `val` into an object of the class `cls` or one of its subclasses or implementations. `cls` can be a primitive type, a class type, or an array. When a primitive target type is given, the method returns an object from the corresponding wrapper class. The class finder `finder` is used to resolve class names and will be replaced by `Class.forName (String)` if `null`.

If `cls` is `String` or `Object`, `val` is returned unchanged. If it is `null`, a null pointer is returned. The method tries to use `numberFromString (Class, String)` for numeric types. If `cls` corresponds to the `Duration` class, this method calls `durationFromString (String)`. If `cls.isEnum()` returns `true`, it uses `Enum.valueOf (Class, String)` to convert the string to an object representing an enum constant. For the `char` type, the first character of the string is returned into the `Character` wrapper object. For boolean type, the method uses `booleanFromString (String)`. Objects of class `URI` are obtained using `uriFromString (URI, String)` while objects of class `URL` are obtained by `urlFromString (URI, String)`. If no previous class is compatible with the target class, the method tries to use `wrapperFromString (Class, String)` to perform the conversion.

If the target object is from class `Class`, the given string is interpreted as a class name and passed to `findClass` to get a `Class` object, using the class finder to resolve the class name. If `finder` is `null`, `Class.forName (String)` is used instead.

The method also processes arrays in arbitrary depths. For a one-dimensional array, the string is read as a list of tokens separated with commas and for each token, the method is called recursively. The `getArrayElements (String)` method is used to perform the tokenization. To encode multi-dimensional arrays, one must encode arrays into tokens by using the brackets. For example, if the target class is `int [] []`, `[1,2]`, `[3,4]` will be converted as a 2D array with 1,2 in the index 0 and 3,4 in the index 1.

If the target class is not an array and cannot be converted in any ways, the method forwards the call to `getStaticValue (URI, ClassFinder, Class, String)`.

Parameters

baseURI the base URI used by `uriFromString (URI, String)` and `urlFromString (URI, String)`.

finder the class finder being used to look for classes if necessary.

cls the target class of the returned object.

val the string to be converted.

Returns the converted object.

Throws

IllegalArgumentException if the string cannot be converted, although conversion to the target class is supported.

UnsupportedConversionException if the target class is not supported by the converter.

NameConflictException if a name conflict occurs when resolving a class name.

```
public static Object[] getArgumentsFromString (URI baseURI, ClassFinder
                                              finder, Class<?>[]
                                              paramTypes, String[] vals)
                                              throws
                                              UnsupportedConversionException,
                                              NameConflictException
```

Converts an array of strings **vals** to an array of arguments to be passed to a constructor or method with parameter types **paramTypes**. The method tries to convert **vals[p]** to the class given by **paramTypes[p]** using the `fromString (URI, ClassFinder, Class, String)` method and stores the result in the returned array.

Parameters

baseURI the base URI used by `fromString (URI, ClassFinder, Class, String)`

paramTypes the target type of each argument.

vals the string arguments.

Returns an array of objects representing each argument.

Throws

IllegalArgumentException if **vals** and **paramTypes** have different lengths or a conversion error occurred.

NullPointerException if **vals**, **paramTypes** or any elements of these arrays are **null**.

UnsupportedConversionException if the converter does not support one or more argument class.

```
public static <T> T getStaticValue (URI baseURI, ClassFinder finder, Class
                                   <T> cls, String val) throws
                                   UnsupportedOperationException,
                                   NameConflictException
```

Converts `val` to an instance of `cls` by considering the string as a call to a constructor, a static method or a static field. The string must be specified as a Java expression. The arguments of constructors and static methods are processed by `fromString (URI, ClassFinder, Class, String)`. The constructed class, the type of the static field, or the return value of the static method must be assignable to the target class. If the conversion fails, an `IllegalArgumentException` is thrown.

Parameters

`baseURI` the base URI used by `uriFromString (URI, String)` and `urlFromString (URI, String)`.

`finder` the class finder being used.

`cls` the target class for the conversion.

`val` the expression being converted.

Returns the constructed object.

Throws

`UnsupportedConversionException` if `fromString (URI, ClassFinder, Class, String)` fails processing the arguments because the class of the called constructor's arguments, the type of the referenced field, or the return value of the called method are not supported as target classes.

`NameConflictException` if a class name conflict arises.

`IllegalArgumentException` if the target class is supported but a conversion problem occurs.

```
public static String[] getArrayElements (String val)
```

Tokenizes the string `val` into an array of strings using whitespaces or commas as delimiters, and merging back parts surrounded by braces or parentheses. Parentheses always appear in resulting tokens while the surrounding braces are removed. Any consecutive substring of whitespace is replaced by a single whitespace character. This method is different from `String.split (String, int)` because it takes braces and parentheses into account.

For example, the string `1 2 3 4` given to this method would produce an array containing four elements: `1`, `2`, `3`, and `4`. The string `{1,2} {3 4}` will be separated in two strings by this method: `1 2` and `3 4`. The string `{{1 2} 3} a (4 5)` would become an array containing `{ 1 2 }`, `3`, and `a (4 5)`.

Parameter

`val` the value to be tokenized.

Returns an array of strings representing each token.

```
public static <T> T numberFromString (Class<T> cls, String val) throws
                                   UnsupportedOperationException
```

Converts the string `val` to an instance of `cls` being a subclass of `Number` or a primitive numeric type. In the `val` argument, the method accepts a number or the special strings `Infinity`, `INF`, `-Infinity`, and `-INF`. For an integer, a short, a long or a byte, infinity is represented using the greatest or smallest acceptable value for the built-in types. For single or double-precision floating points, the infinity can be represented directly. For floating-points, the `nan` string is accepted to denote a NaN.

Parameters

`cls` the target class.

`val` the string being converted.

Returns the constructed numeric wrapper object.

Throws

`NumberFormatException` if a conversion problem occurred.

`UnsupportedOperationException` if the class is not a supported subclass of `Number` or a primitive numeric type.

```
public static Boolean booleanFromString (String val)
```

Converts a string `val` to a boolean wrapper object. The method accepts 1, `true`, `yes`, and `on` as a true value and 0, `false`, `no`, and `off` as a false value.

Parameter

`val` the value being converted.

Returns the result of the conversion.

Throws

`IllegalArgumentException` if the value cannot be converted.

```
public static URI uriFromString (URI baseURI, String uri)
```

Constructs and returns a new URI from the string `url`, resolved against the base URI `baseURI`. If `baseURI` is `null`, this method creates a new URI using the `URI.URI (String)` constructor. Otherwise, it resolves the string `uri` against the base URI to obtain the URI object. Any exception is wrapped into an illegal-argument exception.

Parameters

`baseURI` the base URI.

`uri` the URI to resolve.

Returns the resolved URI object.

```
public static URL urlFromString (URI baseURI, String url)
```

This method calls `uriFromString (URI, String)` with the given base URI, and URL, and converts the resulting URI into a URL using `URI.toURL()`.

Parameters

`baseURI` the base URI.

`url` the URL.

Returns the resulting URL object.

```
public static Duration durationFromString (String str) throws
                                         UnsupportedOperationException
```

Constructs and returns a `Duration` object obtained from the string `str`. This method uses `DatatypeFactory.newDuration (String)` to perform the conversion.

Parameter

`str` the string to convert.

Returns the obtained duration.

Throws

`UnsupportedOperationException` if the data type factory could not be created.

```
public static <T> T wrapperFromString (Class<T> cls, String val) throws
                                         UnsupportedOperationException
```

Converts a string to a wrapper object of class `cls`. This method tries to find a constructor for class `cls` taking a `String` as its unique argument. If such a constructor can be found, it is used to instantiate a new object which is returned. If no such constructor can be found and invoked, the method searches for a `valueOf` public static method taking a `String` argument and returning an instance of `cls` or a subclass of `cls`. If the class does not contain an appropriate constructor or static method, an `UnsupportedOperationException` is thrown. If the constructor or method is found but cannot be called successfully, an `IllegalArgumentException` is thrown.

Parameters

`cls` the target class for the conversion.

`val` the value being converted.

Returns the converted value.

Tries to call a method named `name` on object `o` with arguments given by `vals`, with expected return value class `rcls`. The method searches for all public methods in `cls` taking `vals.length` arguments (0 argument if `vals` is null), and tries to convert the arguments using `getArgumentsFromString (URI, ClassFinder, Class[], String[])`. If `o` is null, the search will be restricted to static methods.

Parameters

`baseURI` the base URI used by `uriFromString (URI, String)` and `urlFromString (URI, String)`.

`finder` the class finder used by `getArgumentsFromString (URI, ClassFinder, Class[], String[])`.

`rcls` the target class of the return type.

`cls` the class defining the method to be invoked.

`o` the object on which the method is invoked, should be an instance of the class represented by `cls`.

`name` the name of the method.

`vals` the arguments for the method.

Returns the return value.

Throws

`UnsupportedConversionException` if some arguments are not compatible with `fromString (URI, ClassFinder, Class, String)`.

`NameConflictException` if a name conflict occurs during a class name resolution.

`IllegalArgumentException` if no method can be found or some conversions failed.

`NoSuchMethodException` if no appropriate method could be found.

`public static String intToString (int i)`

Converts the integer value `i` to a string by using `String.valueOf (int)`. If `i` is equal to the minimum or maximum value of an integer, `-Infinity` or `Infinity` are returned, respectively.

Parameter

`i` the integer being formatted.

Returns the formatted integer.

`public static String byteToString (byte b)`

Same as `intToString (int)` for a byte.

Parameter

`b` the byte being formatted.

Returns the formatted byte.

```
public static String shortToString (short s)
```

Same as `intToString (int)` for a short.

Parameter

`s` the short integer being formatted.

Returns the formatted value.

```
public static String longToString (long l)
```

Same as `intToString (int)` for a long.

Parameter

`l` the long integer being formatted.

Returns the formatted value.

```
public static String numberToString (Number n)
```

Formats the number `n` into a string. If `n` is an instance of wrapped primitive numeric type, the appropriate `convert` static method of this class is called to perform the conversion. Otherwise, `Object.toString()` is used. This method is used to return `Infinity`, and `NaN` as required, instead of large or small values.

Parameter

`n` the number being formatted.

Returns the formatted value.

UnsupportedConversionException

Exception occurring when a conversion is not supported by a method of `StringConvert`. This exception is thrown when a conversion method is given a target class it does not support. This differs from the case where the target class is supported while the conversion fails.

```
package umontreal.iro.lecuyer.util;
```

```
public class UnsupportedConversionException extends Exception
```

Constructors

```
public UnsupportedConversionException()
```

Constructs a new unsupported conversion exception with no target class or message.

```
public UnsupportedConversionException (String message)
```

Constructs a new unsupported conversion exception with no target class and message `message`.

Parameter

`message` the message describing the exception.

```
public UnsupportedConversionException (Class<?> cls, String val)
```

Constructs a new unsupported conversion exception with target class `cls`, converted value `val` and no message.

Parameters

`cls` the target class.

`val` the value being converted.

```
public UnsupportedConversionException (Class<?> cls, String val, String  
                                     message)
```

Constructs a new unsupported conversion exception with target class `cls`, converted value `val` and message `message`.

Parameters

`cls` the target class.

`val` the value being converted.

`message` the message describing the exception.

Methods

```
public Class<?> getTargetClass()
```

Returns the target class of the conversion causing the exception.

Returns the target class.

```
public String getValue()
```

Returns the value to be converted and causing the exception.

Returns the value to be converted.

```
public String toString()
```

Returns a short description of the exception. If no target class and value is associated to the object, this calls the base class's `toString` method. Otherwise, a string containing the following elements is constructed and returned.

- The name of the class of the object.
- `": Cannot convert value "`
- If `getValue()` returns a non-`null` value, the returned value followed by a space
- `"to target class"`
- If the target class is specified, a space followed by the target class name
- If the message is specified, the message preceded by `", "`.

Returns the short string describing the exception.

DoubleFormatter

Represents an object that can format a double-precision value into a string.

```
package umontreal.iro.lecuyer.util;  
  
public interface DoubleFormatter
```

Method

```
public String format (double x)
```

Formats the double `x` as a string, and returns the resulting string.

Parameter

`x` the value being formatted.

Returns the formatted value.

DoubleFormatterWithError

Represents an object that can format a double-precision value into a string, while formatting can possibly be affected by the error on the formatted value.

```
package umontreal.iro.lecuyer.util;
```

```
public interface DoubleFormatterWithError extends DoubleFormatter
```

Methods

```
public String format (double x, double error)
```

Formats the value `x` with error `error` into a string, and returns the formatted string. The error can be, e.g., the radius of a confidence interval, or a standard deviation. The given error must be used only to affect how `x` is formatted; it must be formatted into the returned string.

Parameters

`x` the value being formatted.

`error` the error on the formatted value.

Returns the formatted value.

```
public String format (double x)
```

This should be equivalent to `format (x, 0)`.

DefaultDoubleFormatter

Default formatter. Let x be a value to be formatted. Let d_1 be the number of needed significant digits for $|x| \leq 1$, and d_2 be the number of digits for $|x| > 1$. If $x \geq 10^{d_1}$, the formatter uses `PrintfFormat.f (double)` with 0 decimal digit of precision. If $x < 10^{-d_2}$, the formatter uses `PrintfFormat.e (double)` with d_2 digits of precision. Otherwise, the formatter uses `PrintfFormat.g (double)` with d_1 or d_2 significant digits, depending if $|x|$ is smaller than or equal to 1, or greater than 1.

```
package umontreal.iro.lecuyer.util;
```

```
public class DefaultDoubleFormatter implements DoubleFormatter
```

Constructors

```
public DefaultDoubleFormatter()
```

Constructs a formatter that uses 3 significant digits for all numbers.

```
public DefaultDoubleFormatter (int digitsSmall, int digitsLarge)
```

Constructs a formatter that uses `digitsSmall` significant digits for values smaller than 1, and `digitsLarge` significant digits for other values.

Parameters

`digitsSmall` the number of significant digits for values smaller than 1.

`digitsLarge` the number of significant digits for values greater than or equal to 1.

Methods

```
public int getDigitsSmall()
```

Returns the number of significant digits for values smaller than 1.

Returns the number of significant digits for values smaller than 1.

```
public int getDigitsLarge()
```

Returns the number of significant digits for values greater than or equal to 1.

Returns the number of significant digits for values greater than or equal to 1.

DefaultDoubleFormatterWithError

Default double formatter with error. Uses `PrintfFormat.formatWithError (int, int, int, double, double, String[])` to format values with 3 significant digits.

```
package umontreal.iro.lecuyer.util;

public class DefaultDoubleFormatterWithError implements
    DoubleFormatterWithError
```

Constructors

```
public DefaultDoubleFormatterWithError()
```

Constructs a formatter formatting values with 3 significant digits.

```
public DefaultDoubleFormatterWithError (int numDigits)
```

Constructs a formatter formatting values with `numDigits` significant digits.

Parameter

`numDigits` the number of significant digits for formatting.

Method

```
public int getNumDigits()
```

Returns the number of significant digits for formatting.

Returns the number of significant digits.

LowMemoryNotifier

Uses the Java 5 Management API to monitor memory usage, and notifies registered listeners when memory becomes low. Memory is partitionned into *memory pools* for the Management API. Each pool can have a usage threshold which triggers a notification when exceeded.

When constructing a low-memory notifier, one specifies a collection of memory pools to monitor. For each monitored pool supporting the usage threshold, the threshold is set to a fraction of the maximum memory available for that pool, the default fraction is 0.75. When memory usage exceeds this fraction, a notification is emitted by the memory pool, and this class notifies the registered low-memory listeners. If the maximum memory available for the pool has increased between the time the object was created and the low-memory notification, the usage fraction becomes smaller than the threshold usage fraction, and no listener is notified. The usage threshold of the concerned pool is then updated.

```
package umontreal.iro.lecuyer.util;

public class LowMemoryNotifier
```

Constructors

```
public LowMemoryNotifier()
```

Constructs a low-memory notifier with usage threshold set to 0.75 and monitoring every memory pool supporting usage threshold.

```
public LowMemoryNotifier (double usageThreshold)
```

Constructs a low-memory notifier with usage threshold set to `usageThreshold` and monitoring every memory pool supporting usage threshold.

Parameter

`usageThreshold` the usage threshold.

Throws

`IllegalArgumentException` if `usageThreshold` is smaller than 0 or greater than 1.

```
public LowMemoryNotifier (Collection<MemoryPoolMXBean> pools)
```

Constructs a low-memory notifier with usage threshold set to 0.75, and monitoring every memory pool in `pools` supporting usage threshold.

Parameter

`pools` the collection of memory pools.

Throws

NullPointerException if `pools` is null.

```
public LowMemoryNotifier (Collection<MemoryPoolMXBean> pools, double  
                           usageThreshold)
```

Constructs a low-memory notifier with usage threshold set to `usageThreshold`, and monitoring every memory pool in `pools` supporting usage threshold.

Parameters

`pools` the collection of memory pools.

`usageThreshold` the usage threshold.

Throws

NullPointerException if `pools` is null.

IllegalArgumentException if `usageThreshold` is smaller than 0 or greater than 1.

Methods

```
public void addLowMemoryListener (LowMemoryListener listener1)
```

Registers the low-memory listener `listener`.

Parameter

`listener1` the new low-memory listener to be registered.

Throws

NullPointerException if `listener` is null.

```
public void removeLowMemoryListener (LowMemoryListener listener1)
```

Unregisters the low-memory listener `listener`.

Parameter

`listener1` the low-memory listener to be unregistered.

```
public void removeLowMemoryListeners()
```

Removes all low-memory listeners registered with this object.

```
public List<LowMemoryListener> getLowMemoryListeners()
```

Returns the low-memory listeners currently registered with this object.

Returns the list of currently-registered low-memory listeners.

```
public double getUsageThreshold()
```

Returns the usage threshold used by this low-memory notifier.

Returns the current usage threshold.

```
public void setUsageThreshold (double usageThreshold)
```

Sets the usage threshold used by this low-memory notifier to `usageThreshold`

Parameter

`usageThreshold` the current usage threshold.

Throws

`IllegalArgumentException` if `usageThreshold` is smaller than 0 or greater than 1.

```
public Collection<MemoryPoolMXBean> getMemoryPools()
```

Returns the memory pools monitored by this object.

Returns the monitored memory pools.

LowMemoryListener

Represents a low-memory notification listener that can be registered with a low-memory notifier.

```
package umontreal.iro.lecuyer.util;  
  
public interface LowMemoryListener
```

Method

```
public void lowMemory (LowMemoryNotifier source, MemoryPoolMXBean pool,  
                      double fraction)
```

This method is called when a low-memory condition is detected by the low-memory notifier **source** for the memory pool **pool**, **fraction** indicating the number of used bytes over the maximum number of bytes for the pool.

Parameters

source the low-memory notifier.

pool the memory pool which has exceeded its memory usage threshold.

fraction the number of used bytes over the maximal number of bytes.

ArrayUtil

Provides static utility methods to resize and test arrays.

```
package umontreal.iro.lecuyer.util;
```

```
public class ArrayUtil
```

Methods

```
public static int[] [] resizeArray (int[] [] oldArray, int newRowLen, int  
                                   newColLen)
```

Resize array **oldArray** to **newRowLen** rows and **newColLen** columns. If the new lengths are smaller than the old lengths, the last elements of the arrays are lost. If the new lengths are greater than the old lengths, new elements having the default value **t** are appended to the array. Returns a reference to the new array.

Parameters

oldArray old array

newRowLen new number of rows

newColLen new number of columns

Returns resized array

```
public static int[] [] resizeRow (int[] [] array, int row, int len)
```

Resize row **row** of array **array** to have **len** elements. The other rows of **array** are unchanged. If the new length **len** is greater than the old length, new elements having the default value 0 are appended to row **row**. If **row** is greater or equal to the old number of rows, the array is extended to have **row + 1** rows. The new rows have length 0, except for row **row** which has length **len**.

Returns a reference to the resized array.

Parameters

array old array

row row index

len new length of row **row**

Returns resized array

```
public static double[] [] resizeArray (double[] [] oldArray, int newRowLen,  
                                       int newColLen, double x)
```

Resize array **oldArray** to **newRowLen** rows and **newColLen** columns. If the new lengths are smaller than the old lengths, the last elements of the arrays are lost. If the new lengths are greater than the old lengths, new elements having the default value **x** are appended to the array. Returns a reference to the new array.

Parameters

`oldArray` old array

`newRowLen` new number of rows

`newColLen` new number of columns

`x` default value of the new elements

Returns resized array

```
public static double[][] resizeRow (double[][] array, int row, int len)
```

Resize row `row` of array `array` to have `len` elements. The other rows of `array` are unchanged. If the new length `len` is greater than the old length, new elements having the default value 0 are appended to row `row`. If `row` is greater or equal to the old number of rows, the array is extended to have `row + 1` rows. The new rows have length 0, except for row `row` which has length `len`.

Returns a reference to the resized array.

Parameters

`array` old array

`row` row index

`len` new length of row `row`

Returns resized array

```
public static <T> T[] resizeArray (T[] oldArray, int newLength)
```

Resizes an array `oldArray` to the length `newLength`, and returns a reference to an array with the appropriate length. If the length of `oldArray` corresponds to `newLength`, the method returns the old array reference. Otherwise, a new array is constructed, and the elements are copied from the old array, using `System.arraycopy (Object, int, Object, int, int)`. If the new length is smaller than the old length, the last elements of the array are lost. If the new length is greater than the old length, new elements having the default value (`null`, 0, or `false`, depending on the type of array) are appended to the array.

Parameters

`oldArray` the old array to be resized.

`newLength` the required length of the returned array.

Returns the old array or a resized copy of the old array.

Throws

`NullPointerException` if `oldArray` is `null`.

`IllegalArgumentException` if `oldArray` does not correspond to an array.

`NegativeArraySizeException` if `newLength` is negative.

```
public static void checkRectangularMatrix (Object m)
```

Determines if the given object `m` is a rectangular matrix. To be a rectangular matrix, the object must be an array of arrays of any primitive or non-array reference type. Each of the arrays must be non-`null` and have the same length.

Parameter

`m` the object to be tested.

Throws

`NullPointerException` if `m` or one of its elements are `null`.

`IllegalArgumentException` if the object is not a rectangular matrix.

```
public static <T> T deepClone (T array, boolean cloneElements)
```

Constructs and returns a deep clone of the array `array`. If the given object corresponds to a one-dimensional array, the method clones the array. It also clones the elements in the array if `cloneElements` is `true`. If the given array is multi-dimensional, the method creates a new array of the same length, and recursively calls itself to clone nested arrays.

If `cloneElements` is `true`, the elements in the given array must be arrays, primitive elements, or objects implementing the `Cloneable` interface.

This method is equivalent to `array.clone()` if the given array is one-dimensional, and `cloneElements` is `false`.

Type parameter

`T` the type of the array.

Parameters

`array` the array to clone.

`cloneElements` determines if elements in the array are cloned.

Returns the cloned array.

Throws

`IllegalArgumentException` if the class of the given object is not an array.

```
public static <T> T deepClone (T array)
```

Equivalent to `deepClone (array, false)` .

Type parameter

`T` the type of the array.

Parameter

`array` the array to clone.

Returns the cloned array.

```
public static byte[][] getTranspose (byte[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static short[][] getTranspose (short[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static int[][] getTranspose (int[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static long[][] getTranspose (long[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static float[][] getTranspose (float[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static double[][] getTranspose (double[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static char[][] getTranspose (char[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static boolean[][] getTranspose (boolean[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static <T> T[][] getTranspose (T[][] m)
```

Returns the transpose of a matrix `m`. This method assumes that `m` is rectangular and has dimensions $a \times b$. It creates a matrix of dimensions $b \times a$, and stores element (i, j) of `m` in its element (j, i) .

Parameter

`m` the matrix to be transposed.

Returns the transposed matrix.

Throws

`NullPointerException` if `m` is null, or `m[i]` is null for at least one index `i`.

```
public static byte min (byte... a)
```

Returns the value of the smallest element in the array `a`.

Parameter

`a` the tested array.

Returns the minimum.

```
public static short min (short... a)
```

Returns the value of the smallest element in the array `a`.

Parameter

`a` the tested array.

Returns the minimum.

```
public static int min (int... a)
```

Returns the value of the smallest element in the array `a`.

Parameter

`a` the tested array.

Returns the minimum.

```
public static long min (long... a)
```

Returns the value of the smallest element in the array `a`.

Parameter

a the tested array.

Returns the minimum.

```
public static float min (float... a)
```

Returns the value of the smallest element in the array **a**.

Parameter

a the tested array.

Returns the minimum.

```
public static double min (double... a)
```

Returns the value of the smallest element in the array **a**.

Parameter

a the tested array.

Returns the minimum.

```
public static <T extends Comparable<T>> T min (T... a)
```

Returns the value of the smallest element in the array **a**.

Parameter

a the tested array.

Returns the minimum.

```
public static byte max (byte... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static short max (short... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static int max (int... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static long max (long... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static float max (float... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static double max (double... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static <T extends Comparable<T>> T max (T... a)
```

Returns the value of the greatest element in the array **a**.

Parameter

a the tested array.

Returns the maximum.

```
public static boolean allDifferent (byte... a)
```

Determines if the elements in the array **a** are all different, and returns the result of the test.

Parameter

a the tested array.

Returns **true** if and only if all elements are different.

```
public static boolean allDifferent (short... a)
```

Determines if the elements in the array **a** are all different, and returns the result of the test.

Parameter

a the tested array.

Returns `true` if and only if all elements are different.

```
public static boolean allDifferent (int... a)
```

Determines if the elements in the array **a** are all different, and returns the result of the test.

Parameter

a the tested array.

Returns `true` if and only if all elements are different.

```
public static boolean allDifferent (long... a)
```

Determines if the elements in the array **a** are all different, and returns the result of the test.

Parameter

a the tested array.

Returns `true` if and only if all elements are different.

```
public static boolean allDifferent (float... a)
```

Determines if the elements in the array **a** are all different, and returns the result of the test.

Parameter

a the tested array.

Returns `true` if and only if all elements are different.

```
public static boolean allDifferent (double... a)
```

Determines if the elements in the array **a** are all different, and returns the result of the test.

Parameter

a the tested array.

Returns `true` if and only if all elements are different.

```
public static float[] round (int maxDigits, float... a)
```

Roudns each number in **a** to **maxDigits** digits.

Parameters

maxDigits the maximal number of digits.

a the array.

Returns the resulting array.

```
public static double[] round (int maxDigits, double... a)
```

Roudns each number in **a** to **maxDigits** digits.

Parameters

`maxDigits` the maximal number of digits.

`a` the array.

Returns the resulting array.

```
public static int getMinDigits (float... a)
```

Returns the minimal number of digits after the decimal point required for the numbers in the array `a` to be rounded without different values becoming equal. For example, this method returns 2 when given the array { 1.26, 1.28, 1.26, 1.27 } but 1 for the array { 1.26, 1.42 }.

Parameter

`a` the tested array.

Returns the minimal number of digits.

```
public static int getMinDigits (double... a)
```

Returns the minimal number of digits after the decimal point required for the numbers in the array `a` to be rounded without different values becoming equal. For example, this method returns 2 when given the array { 1.26, 1.28, 1.26, 1.27 } but 1 for the array { 1.26, 1.42 }.

Parameter

`a` the tested array.

Returns the minimal number of digits.

```
public static int[] merge (int[]... arrays)
```

Merges the given arrays into a single array, and returned the constructed array.

Parameter

`arrays` the arrays to be merged together.

Returns the merged array.

```
public static double[][] copy (double[][] M)
```

Copies the array `M` and returns the copy.

Parameter

`M`

Returns copy of `M`

FileUtil

```
package umontreal.iro.lecuyer.util;
```

```
public class FileUtil
```

Methods

```
public static boolean delete (File f)
```

Deletes the file or empty directory `f`, and returns the success indicator of the operation. This method is similar to `File.delete()`, except that if the first call to `delete` fails, it attempts to call the method a second time after 10ms. This is a heuristic attempt to fix a possible race condition when deleting a file under Windows XP inspired from Ant's source code.

Parameter

`f` the file or empty being deleted.

Returns the success indicator of the operation.

```
public static void copyFile (File srcFile, File destFile) throws
```

`IOException`

Copies the file `srcFile` into the destination file `destFile`. If `destFile` represents a directory, a new file having the same name and contents as `srcFile` is created in the referred directory. If `destFile` represents an existing file, it is overwritten with the contents of `srcFile`. Otherwise, a new file is created.

Parameters

`srcFile` the source file.

`destFile` the destination file.

Throws

`IOException` if a problem occurs during the copy.

`NullPointerException` if any argument is null.

```
public static void moveFile (File srcFile, File destFile) throws
```

`IOException`

Moves the file `srcFile` to `destFile`. This method is similar to `copyFile (File, File)`, except it deletes the source file after the copy succeeds. Moreover, before making a full copy of the file, the method tries to use `File.renameTo (File)` which might rename the file without copying it on some platforms. The copy happens only if the call to `File.renameTo (File)` fails.

Parameters

`srcFile` the source file.

`destFile` the destination file.

Throws

`IOException` if a problem occurs during file moving.

ExceptionUtil

Provides helper methods to format exceptions in a more compact way than the default exception formatting. The default behavior of the Java Virtual Machine when an uncaught exception occurs is to call the `Throwable.printStackTrace()` method in order to display the full stack trace on the console. This produces a rather verbose output, especially if the exception has a cause, i.e., `Throwable.getCause()` returns `true`. However, for some exception types, e.g., `IOException`, it might be sufficient to display only the exception class name and message. This can be done by `Throwable.toString()`, but the cause of the exception is then lost. This class provides the `throwableToString (Throwable)` which can format and return a short version of the a throwable's cause chain. Methods are also provided to install a default exception handler which calls `throwableToString (Throwable)` instead of `Throwable.printStackTrace()`.

```
package umontreal.iro.lecuyer.util;
```

```
public class ExceptionUtil
```

Methods

```
public static String throwableToString (Throwable throwable)
```

Converts the throwable `throwable` to a string, and returns the constructed string. This method uses `Throwable.toString()` to get a string from the given throwable, and all its causes. The formatting is similar to the default way Java prints exceptions, but stack trace elements are omitted. However, if the `umontreal.iro.lecuyer.util.PrintStackTrace` property is set to `true`, the full stack trace is also formatted for each throwable.

Parameter

`throwable` the throwable to convert.

Returns the string corresponding to the throwable.

```
public static void replaceDefaultExceptionHandler()
```

Sets the uncaught exception handler to use `throwableToString (Throwable)` to print exceptions, i.e., instances of `Exception`. Other instances of `Throwable`, e.g., instances of `Error`, are printed using `Throwable.printStackTrace()`.

```
public static String formatMemoryStatus()
```

Constructs and returns a string giving memory status of the virtual machine. This string gives the free, available and maximal memory of the JVM.

Returns the string with memory status.

Pair

Represents a pair of values.

Type parameters

S the type of the first value.

T the type of the second value.

```
package umontreal.iro.lecuyer.util;
```

```
public class Pair<S, T> implements Cloneable, Serializable
```

Constructors

```
public Pair (S first, T second)
```

Constructs a new pair for values `first` and `second`.

Parameters

`first` the first value.

`second` the second value.

```
public Pair (Pair<? extends S, ? extends T> pair)
```

Constructs a new pair from the pair `pair`.

Parameter

`pair` the pair to get values from.

Throws

`NullPointerException` if `pair` is null.

Methods

```
public S getFirst()
```

Returns the first value of this pair.

Returns the first value.

```
public void setFirst (S first)
```

Sets the first value of this pair to `first`.

Parameter

`first` the new first value of this pair.

```
public T getSecond()
```

Returns the second value of this pair.

Returns the second value of this pair.

```
public void setSecond (T second)
```

Sets the second value of this pair to `second`.

Parameter

`second` the second value of this pair.

```
public Pair<S, T> clone()
```

Clones this pair. This method does not clone the values in the pair.

LineBreaker

Provides facilities to write multiple lines into output streams or writers, with automatic conversion of line separators. Strings are often generated using a string builder. Strings are often generated by string builders before they are written into streams (e.g., files or console). When the generated strings contain multiple lines, the newline character (`\n`) is often used as a line separator, but this does not always correspond to the platform's end-of-line separator. This class can be used to convert the line separators to the appropriate- platform-dependent, line separator before writing strings into streams.

```
package umontreal.iro.lecuyer.util;
```

```
public class LineBreaker
```

Methods

```
public static void writeLines (PrintWriter out, String str)
```

Prints the string `out` using the correct platform-specific line separator. This method splits the given string `str` using newline characters (`\r`, `\n`, and `\r\n`) as delimiters, and calls `PrintWriter.println (String)` for each part.

Parameters

`out` the print writer.

`str` the string to write.

```
public static void writeLines (PrintStream out, String str)
```

Similar to `writeLines (PrintWriter, String)`, for a print stream.

Parameters

`out` the print stream.

`str` the string.

```
public static void writeLines (BufferedWriter out, String str) throws  
                             IOException
```

Similar to `writeLines (PrintWriter, String)`, for a buffered writer.

Parameters

`out` the buffered writer.

`str` the string.

Throws

if an I/O error occurs.

```
public static <T extends Appendable> T writeLines (T out, String str,  
                                                    String lineSeparator)  
                                                    throws IOException
```

Similar to `writeLines (PrintWriter, String)`, for a generic appendable object, and a user-defined line separator `lineSeparator`.

Parameters

`out` the buffered writer.

`str` the string.

`lineSeparator` the line separator.

Returns the appendable `out`.

Throws

if an I/O error occurs.

```
public static String getDefaultLineSeparator()
```

Returns the default, platform-dependent, line separator.

Returns the default line separator.

LaTeXFormatter

Provides methods for formatting times and number for typesetting with L^AT_EX.

```
package umontreal.iro.lecuyer.util;  
  
public class LaTeXFormatter implements Closeable, Flushable, Appendable
```

Fields

```
public static final String DEFAULTMATHENSURINGCMD
```

Default math-ensuring command, `ensuremath`.

```
public static final String DEFAULTNAN
```

Default string for representing NaN, `---`.

Methods

```
public LaTeXFormatter formatTime (double time)
```

Formats a string representing the time duration `time`, given in seconds. The returned string is of the form `1h2min3s`.

Parameter

`time` the time to be formatted.

Returns the latex formatter.

```
public LaTeXFormatter formatInteger (int precision, long x)
```

Formats the integer `x` for the locale `locale` with at least `precision` digits. If `grouping` is `true`, a locale-specific delimiter is used to separate groups, usually thousands. If the grouping corresponds to a whitespace, it is converted to `~`, the L^AT_EX unbreakable space.

Parameters

`precision` the minimal number of digits.

`x` the integer being formatted.

Returns the formatter object.

Throws

`IllegalArgumentException` if `precision` is negative.

```
public LaTeXFormatter formatFixed (int precision, double x)
```

Formats the given double-precision number `x` to a string with a fixed number `precision` of decimal digits, for the locale `locale`. If `grouping` is `true`, a locale-specific delimiter is used to separate groups, usually thousands. If the grouping corresponds to a whitespace, it is converted to `~`, the `LATEX` unbreakable space. If the formatted string has to be typeset in math mode, it is surrounded with the given math-ensuring command `mathEnsuringCmd`.

Parameters

`precision` the number of decimal digits.

`x` the number being formatted.

Returns this formatter object.

Throws

`IllegalArgumentException` if `precision` is negative.

```
public LaTeXFormatter formatScientific (int precision, double x)
```

Formats the double-precision value `x` into a string with the scientific notation for the locale `locale`, with `precision` decimal digits of precision. If the formatted string has to be typeset in math mode, it is surrounded with the given math-ensuring command `mathEnsuringCmd`.

Parameters

`precision` the number of decimal digits.

`x` the value being formatted.

Returns this formatter object.

Throws

`IllegalArgumentException` if `precision` is negative.

```
public LaTeXFormatter formatNumber (int precision, double x)
```

Formats the given number `x` for the locale `locale`, with `precision` significant digits. This method uses scientific notation when the exponent is smaller than `-4`, or greater than or equal to `precision`. Otherwise, it uses standard notation. If `grouping` is `true`, a locale-specific delimiter is used to separate groups, usually thousands. If the grouping corresponds to a whitespace, it is converted to `~`, the `LATEX` unbreakable space. If the formatted string has to be typeset in math mode, it is surrounded with the given math-ensuring command `mathEnsuringCmd`.

Parameters

`precision` the number of significant digits.

`x` the value being formatted.

Returns this formatter object.

Throws

`IllegalArgumentException` if `precision` is negative or 0.

```
public LaTeXFormatter formatNumber (int digitsSmall, int digitsLarge,
                                   double x)
```

Formats `x` with `digitsSmall` significant digits if it is smaller than 1, and `digitsLarge` significant digits if it is larger than 1. Let x be a value to be formatted. Let d_1 be the number of needed significant digits for $|x| \leq 1$, and d_2 be the number of digits for $|x| > 1$. If $x \geq 10^{d_1}$, the formatter uses `formatFixed (int, double)` with 0 decimal digit of precision. If $x < 10^{-d_2}$, the formatter uses `formatScientific (double)` with d_2 digits of precision. Otherwise, the formatter uses `formatNumber (int, int, double)` with d_1 or d_2 significant digits, depending if $|x|$ is smaller than or equal to 1, or greater than 1.

Parameters

`digitsSmall` the number of significant digits for small numbers.

`digitsLarge` the number of significant digits for large numbers.

`x` the value to be formatted.

Returns this formatter object.

```
public static String processInfiniteAndNaN (double x, String
                                           mathEnsuringCmd, String
                                           nanString)
```

Returns a string representing infinite or NaN for `x`, or `null` if `x` is finite. This method is used by `formatFixed (int, double)`, `formatScientific (int, double)`, and `formatNumber (int, int, double)`.

Parameters

`x` the number being processed.

`mathEnsuringCmd` the math-ensuring command.

`nanString` the string representing NaN.

Returns a string representing infinity or NaN, or `null`.

```
public static String fixGroupingAndDecimal (NumberFormat nf, String str)
```

Fixes groupings of the string `str` formatted by the number formatter `nf`, for \LaTeX compatibility. If the grouping used by the given number formatter is a whitespace, this method replaces it with a \LaTeX unbreakable space. Otherwise, the string is unchanged.

Parameters

`str` the string to fix.

`nf` the number formatter having formatted the string.

Returns the fixed string.

```
public static String fixScientific (String str, String mathEnsuringCmd)
```

Converts the string `str` using Java scientific notation to \LaTeX notation. For example, this converts `2e4` to `2*10^{4}`.

Parameters

`str` the string being fixed.

`mathEnsuringCmd` the math-ensuring \LaTeX command.

Returns the fixed string.

LaTeXArrayFormatter

Provides utility methods to format every element of arrays using methods of `LaTeXFormatter`. All methods in this class format each element of given arrays, separated by a comma and a space. For example, the method `formatIntegerArray (int, byte[])` may produce 2, 5, 1, 6.

```
package umontreal.iro.lecuyer.util;
```

```
public class LaTeXArrayFormatter implements Closeable, Flushable
```

Methods

```
public LaTeXArrayFormatter formatIntegerArray (int precision, byte[] array)
```

Formats each integer of the array `array` with `precision` digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatIntegerArray (int precision, short[]  
                                              array)
```

Formats each integer of the array `array` with `precision` digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatIntegerArray (int precision, int[] array)
```

Formats each integer of the array `array` with `precision` digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatIntegerArray (int precision, long[] array)
```

Formats each integer of the array `array` with `precision` digits.

Parameters

precision the minimal number of digits of precision.

array the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatFixedArray (int precision, float[] array)
```

Formats each number of the array **array** using a fixed **precision** number of decimal digits.

Parameters

precision the minimal number of digits of precision.

array the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatFixedArray (int precision, double[] array)
```

Formats each number of the array **array** using a fixed **precision** number of decimal digits.

Parameters

precision the minimal number of digits of precision.

array the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatScientificArray (int precision, float[]  
                                                  array)
```

Formats each number of the array **array** in scientific notation with **precision** decimal digits.

Parameters

precision the minimal number of digits of precision.

array the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatScientificArray (int precision, double[]  
                                                  array)
```

Formats each number of the array **array** in scientific notation with **precision** decimal digits.

Parameters

precision the minimal number of digits of precision.

array the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int precision, byte[] array)
```

Formats each number of the array `array` with `precision` significant digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int precision, short[] array)
```

Formats each number of the array `array` with `precision` significant digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int precision, int[] array)
```

Formats each number of the array `array` with `precision` significant digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int precision, long[] array)
```

Formats each number of the array `array` with `precision` significant digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int precision, float[] array)
```

Formats each number of the array `array` with `precision` significant digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int precision, double[]  
                                             array)
```

Formats each number of the array `array` with `precision` significant digits.

Parameters

`precision` the minimal number of digits of precision.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int digitsSmall, int  
                                             digitsLarge, byte[] array)
```

Formats each number of the array `array` using `LaTeXFormatter.formatNumber (int, int, double)`.

Parameters

`digitsSmall` the number of significant digits for numbers smaller than 1.

`digitsLarge` the number of significant digits for numbers greater than 1.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int digitsSmall, int  
                                             digitsLarge, short[] array)
```

Formats each number of the array `array` using `LaTeXFormatter.formatNumber (int, int, double)`.

Parameters

`digitsSmall` the number of significant digits for numbers smaller than 1.

`digitsLarge` the number of significant digits for numbers greater than 1.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int digitsSmall, int  
                                             digitsLarge, int[] array)
```

Formats each number of the array `array` using `LaTeXFormatter.formatNumber (int, int, double)`.

Parameters

`digitsSmall` the number of significant digits for numbers smaller than 1.

`digitsLarge` the number of significant digits for numbers greater than 1.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int digitsSmall, int
                                             digitsLarge, long[] array)
```

Formats each number of the array `array` using `LaTeXFormatter.formatNumber (int, int, double)`.

Parameters

`digitsSmall` the number of significant digits for numbers smaller than 1.

`digitsLarge` the number of significant digits for numbers greater than 1.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int digitsSmall, int
                                             digitsLarge, float[] array)
```

Formats each number of the array `array` using `LaTeXFormatter.formatNumber (int, int, double)`.

Parameters

`digitsSmall` the number of significant digits for numbers smaller than 1.

`digitsLarge` the number of significant digits for numbers greater than 1.

`array` the array to be formatted.

Returns this array formatter.

```
public LaTeXArrayFormatter formatNumberArray (int digitsSmall, int
                                             digitsLarge, double[] array)
```

Formats each number of the array `array` using `LaTeXFormatter.formatNumber (int, int, double)`.

Parameters

`digitsSmall` the number of significant digits for numbers smaller than 1.

`digitsLarge` the number of significant digits for numbers greater than 1.

`array` the array to be formatted.

Returns this array formatter.

LaTeXObjectMatrixFormatter

```
package umontreal.iro.lecuyer.util;
```

```
public class LaTeXObjectMatrixFormatter extends Formatter
```

Constructors

```
public LaTeXObjectMatrixFormatter()
```

Constructs and returns a matrix formatter with alignment `ttLEFTi/ttj`.

```
public LaTeXObjectMatrixFormatter (String alignment)
```

Constructs and returns a matrix formatter.

Parameter

`alignment` the given alignment used to align a column.

Methods

```
protected String form (AbstractMatrix1D matrix, int index, Former  
                        formatter)
```

Converts a given cell to a String; no alignment considered.

```
protected String form (ObjectMatrix1D matrix, int index, Former formatter)
```

Converts a given cell to a String; no alignment considered.

```
protected String[] [] format (AbstractMatrix2D matrix)
```

Returns a string representations of all cells; no alignment considered.

```
protected String[] [] format (ObjectMatrix2D matrix)
```

Returns a string representations of all cells; no alignment considered.

```
protected String toString (AbstractMatrix2D matrix)
```

Returns a string representation of the given matrix.

Parameter

`matrix` the matrix to convert.

```
public String toString (ObjectMatrix1D matrix)
```

Returns a string representation of the given matrix.

Parameter

`matrix` the matrix to convert.

```
public String toString (ObjectMatrix2D matrix)
```

Returns a string representation of the given matrix.

Parameter

`matrix` the matrix to convert.

```
public String toString (ObjectMatrix3D matrix)
```

Returns a string representation of the given matrix.

Parameter

`matrix` the matrix to convert.

```
public String toTitleString (ObjectMatrix2D matrix, String[] rowNames,
                             String[] columnNames, String rowAxisName,
                             String columnAxisName, String title)
```

Returns a string representation of the given matrix with axis as well as rows and columns labeled. Pass `!tt!null!|tt!` to one or more parameters to indicate that the corresponding decoration element shall not appear in the string converted matrix.

Parameters

`matrix` The matrix to format.

`rowNames` The headers of all rows (to be put to the left of the matrix).

`columnNames` The headers of all columns (to be put to above the matrix).

`rowAxisName` The label of the y-axis.

`columnAxisName` The label of the x-axis.

`title` The overall title of the matrix to be formatted.

Returns the matrix converted to a string.

```
public String toTitleString (ObjectMatrix3D matrix, String[] sliceNames,
                             String[] rowNames, String[] columnNames,
                             String sliceAxisName, String rowAxisName,
                             String columnAxisName, String title)
```

Returns a string representation of the given matrix with axis as well as rows and columns labeled. Pass `!tt!null!|tt!` to one or more parameters to indicate that the corresponding decoration element shall not appear in the string converted matrix.

Parameters

matrix The matrix to format.

sliceNames The headers of all slices (to be put above each slice).

rowNames The headers of all rows (to be put to the left of the matrix).

columnNames The headers of all columns (to be put to above the matrix).

sliceAxisName The label of the z-axis (to be put above each slice).

rowAxisName The label of the y-axis.

columnAxisName The label of the x-axis.

title The overall title of the matrix to be formatted.

Returns the matrix converted to a string.

ModifiableWorkbook

Encapsulates a workbook from JExcel API that can be created from an already existing file, modified, and written back to the input file. JExcel API can be used to read workbooks compatible with Microsoft's Excel, or create workbooks from scratch. However, modifying already existing workbooks requires the original workbook to be read, changed in a temporary file, and copied back to the original location. This class takes care of these steps. After an instance is constructed using a file object, the method `getWorkbook()` can be used to get the currently opened workbook. The `close()` method closes the workbook.

```
package umontreal.iro.lecuyer.util;
```

```
public class ModifiableWorkbook
```

Constructors

```
public ModifiableWorkbook (File outputFile) throws IOException,  
                           BiffException
```

Constructs a new modifiable workbook from the Excel file referred to by `outputFile`. If the given file does not exist, this method creates a new workbook. Otherwise, it reads the existing workbook, and creates a temporary file containing a new workbook starting with the existing one.

Parameter

`outputFile` the output file.

Throws

`IOException` if an error occurs when reading the output file.

`BiffException` if an error occurs while parsing an existing workbook.

```
public ModifiableWorkbook (String outputFile) throws IOException,  
                           BiffException
```

Similar to the constructor `ModifiableWorkbook (File)`, with a string instead of a file object.

Methods

```
public WritableWorkbook getWorkbook()
```

Returns a reference to the encapsulated workbook.

Throws

`IllegalStateException` if this object is closed.

```
public File getOutputFile()
```

Returns a reference to the file object representing the output file.

Throws

`IllegalStateException` if this object is closed.

```
public void close() throws WriteException, IOException
```

Closes the encapsulated workbook. This method calls `WritableWorkbook.write()` followed by `WritableWorkbook.close()` on the encapsulated workbook returned by `getWorkbook()`. If the workbook was created from another one, the temporary file containing the written workbook is copied back to the output file returned by `getOutputFile()`.

Throws

`WriteException` if an error occurs while formatting the encapsulated workbook.

`IOException` if an error occurs while writing the encapsulated workbook to disk.

```
public void discardChanges() throws WriteException, IOException
```

Discards all the changes made in the encapsulated workbook. This method closes the encapsulated workbook, and deletes the temporary files that contains it. If the workbook was created from an existing file, the existing file is unchanged.

Throws

`WriteException` if an error occurs while closing the workbook.

`IOException` if an error occurs while closing the workbook.

```
protected void finalize() throws Throwable
```

This method calls the `close()` method.

IntArray

Represents an immutable array of integers. This class is similar to the `Integer` class, but it wraps an array of integers rather than an integer. Instances of this class can be used as set elements or map keys, because this class implements the `hashCode()` and `equals (Object)` methods to compare the contents of the array.

```
package umontreal.iro.lecuyer.util;

public class IntArray implements Comparable<IntArray>
```

Field

```
public final int length
```

Gives the length of the wrapped array.

Constructor

```
public IntArray (int[] array)
```

Constructs a new array object from the given array of integers. A `null` value is considered as an array with length 0.

Parameter

`array` the array to be wrapped.

Methods

```
public int[] getArray()
```

Returns a copy of the wrapped array. Note that modifying the returned array does not affect the array wrapped by this object.

Returns a copy of the wrapped array.

```
public int getElement (int i)
```

Returns the element with index `i` of the wrapped array.

Parameter

`i` the queried index.

Returns the value of the element.

Throws

`ArrayIndexOutOfBoundsException` if `i` is negative or greater than or equal to `length`.

```
public int hashCode()
```

Returns the result of `Arrays.hashCode (int[])` on the wrapped array.

```
public boolean equals (Object o)
```

If `o` corresponds to an instance of `IntArray`, tests the equality of the wrapped arrays using `Arrays.equals (int[], int[])`, and returns the result of the test. Otherwise, returns `false`.

```
public String toString()
```

Returns the result of `Arrays.toString (int[])` called on the wrapped array.

Package `umontreal.iro.lecuyer.collections`

Provides collections dynamically transforming the elements of other collections or merging collections together. Classes of this package provide views of the transformed or merged collections, i.e., any modification to the original collection is reflected on the transformed collection.

TransformingCollection

Represents a collection that dynamically transforms the elements of another collection. This abstract class defines a collection containing an inner collection of elements of a certain type, and provides facilities to convert these inner elements to outer elements of another type. A concrete subclass simply needs to implement the `convertFromInnerType` (IE) and `convertToInnerType` (OE) methods for converting between the inner and the outer types.

It is strongly recommended that the mapping established by the conversion methods be one-to-one, i.e., an element in the inner collection corresponds to a single element in the outer collection. Otherwise, the size of the outer collection might be incorrect, and the iterator may unexpectedly give the same elements multiple times. Also, `null` should always correspond to `null`.

Type parameters

OE the outer type of the elements.

IE the inner type of the elements.

```
package umontreal.iro.lecuyer.collections;

public abstract class TransformingCollection<OE, IE> extends
    AbstractCollection<OE>
```

Constructor

```
public TransformingCollection (Collection<IE> innerCollection)
    Constructs a new transforming collection mapping the elements of the inner collection
    innerCollection.
```

Parameter

`innerCollection` the inner collection.

Throws

`NullPointerException` if `innerCollection` is `null`.

Methods

```
public Collection<IE> getInnerCollection()
    Returns the inner collection.
```

Returns the inner collection.

```
public abstract OE convertFromInnerType (IE e)
    Converts an element in the inner collection to an element of the outer type.
```

Parameter

e the inner element.

Returns the outer element.

```
public abstract IE convertToInnerType (OE e)
```

Converts an element of the outer type to an element of the inner collection.

Parameter

e the outer element.

Returns the inner element.

TransformingList

Represents a list that dynamically transforms the elements of another list. This class extends the transforming collection to implement the `List` interface.

Type parameters

`OE` the outer type of the elements.

`IE` the inner type of the elements.

```
package umontreal.iro.lecuyer.collections;
```

```
public abstract class TransformingList<OE, IE> extends TransformingCollection<OE, IE> implements List<OE>
```

Constructor

```
public TransformingList (List<IE> innerList)
```

Constructs a new transforming list mapping the elements of the inner list `innerList`.

Parameter

`innerList` the inner list.

Throws

`NullPointerException` if `innerList` is null.

Methods

```
public List<IE> getInnerCollection()
```

Returns the inner list.

Returns the inner list.

```
public TransformingList<OE, IE> tryToMakeRandomAccess()
```

Attempts to make this transforming list random-accessible, i.e., supporting fast random access. If the inner list implements the `RandomAccess` interface, this method returns a transforming list implementing `RandomAccess`. Otherwise, the method returns this reference.

Returns the transforming list, possibly random-accessible.

TransformingSet

Represents a set that dynamically transforms the elements of another set. This class extends the transforming collection to implement the `Set` interface.

Type parameters

`OE` the type of the outer elements

`IE` the type of the inner elements

```
package umontreal.iro.lecuyer.collections;
```

```
public abstract class TransformingSet<OE, IE> extends TransformingCollection<  
    OE, IE>  
    implements Set<OE>
```

Constructor

```
public TransformingSet (Set<IE> innerSet)
```

Constructs a new transforming set mapping the elements of the inner set `innerSet`.

Parameter

`innerSet` the inner set.

Throws

`NullPointerException` if `innerSet` is null.

Method

```
public Set<IE> getInnerCollection()
```

Returns the inner set.

Returns the inner set.

TransformingMap

Represents a map that dynamically transforms the elements of another map. This abstract class defines a map containing an inner map of elements with certain key and value types, and provides facilities to convert keys and values to outer other types. A concrete subclass simply needs to provide methods for converting keys and values between the inner and the outer types. The mapping established for the keys by these methods must be one-to-one. Otherwise, the size of the outer map might be incorrect, and the iterators may give the same entry multiple times. Also, a `null` key or value should always correspond to a `null` element.

Type parameters

OK the type of the outer keys

OV the type of the outer values

IK the type of the inner keys

IV the type of the inner values

```
package umontreal.iro.lecuyer.collections;
```

```
public abstract class TransformingMap<OK, OV, IK, IV> extends AbstractMap<OK,  
    OV>
```

Constructor

```
public TransformingMap (Map<IK, IV> innerMap)
```

Constructs a new transforming map converting keys and values of map `innerMap`.

Parameter

`innerMap` the inner map.

Throws

`NullPointerException` if `innerMap` is `null`.

Methods

```
public Map<IK, IV> getInnerMap()
```

Returns the inner map associated with this map.

Returns the inner map.

```
public abstract OK convertKeyFromInnerType (IK key)
```

Converts the key for the inner map to a key for the outer map.

Parameter

key the inner key.

Returns the outer key.

```
public abstract IK convertKeyToInnerType (OK key)
```

Converts the key for the outer map to a key for the inner map.

Parameter

key the outer key.

Returns the inner key.

```
public abstract OV convertValueFromInnerType (IV value)
```

Converts the value for the inner map to a value for the outer map.

Parameter

value the inner value.

Returns the outer value.

```
public abstract IV convertValueToInnerType (OV value)
```

Converts the value for the outer map to a value for the inner map.

Parameter

value the outer value.

Returns the inner value.

MergedCollection

Represents a collection providing a view of two collections merged together. The merged collection contains the elements of collections `col1` and `col2`, and its iterator traverses both collections. A merged collection is immutable, but any change to the inner collections is reflected on the merged collection.

Type parameter

E the type of the elements in the merged collection.

```
package umontreal.iro.lecuyer.collections;  
  
public class MergedCollection<E> extends AbstractCollection<E>
```

Constructor

```
public MergedCollection (Collection<? extends E> col1, Collection<?  
                        extends E> col2)
```

Constructs a collection merging collections `col1` and `col2`.

Parameters

`col1` the first collection.

`col2` the second collection.

Throws

`NullPointerException` if `col1` or `col2` are `null`.

Methods

```
public Collection<? extends E> getFirstCollection()
```

Returns a reference to the first collection of this merged collection.

Returns the first collection.

```
public Collection<? extends E> getSecondCollection()
```

Returns a reference to the second collection of this merged collection.

Returns the second collection.

MergedList

Represents a list providing a view of two lists side by side. This extends the merged collection for implementing the `List` interface.

Type parameter

`E` the type of the element in the merged list.

```
package umontreal.iro.lecuyer.collections;

public class MergedList<E> extends MergedCollection<E>
    implements List<E>
```

Constructor

```
public MergedList (List<? extends E> list1, List<? extends E> list2)
```

Constructs a new merged list from lists `list1` and `list2`.

Parameters

`list1` the first list.

`list2` the second list.

Throws

`NullPointerException` if `list1` or `list2` are null.

Methods

```
public List<? extends E> getFirstCollection()
```

Returns the reference to the first list in this merged list.

Returns the first list.

```
public List<? extends E> getSecondCollection()
```

Returns the reference to the second list in this merged list.

Returns the second list.

```
public MergedList<E> tryToMakeRandomAccess()
```

Attempts to make this merged list random-accessible, i.e., supporting fast random access. If both inner lists implement the `RandomAccess` interface, this method returns a merged list implementing `RandomAccess`. Otherwise, the method returns this reference.

Returns the merged list, possibly random-accessible.

```
public static <E> MergedList<E> newRandomAccess (List<? extends E> list1,  
                                                List<? extends E> list2)
```

Attempts to construct a random-accessible merged list. If `list1` and `list2` both implement `RandomAccess`, this constructs and returns a list implementing `RandomAccess`. Otherwise, the constructed list does not implement the interface.

Type parameter

`E` the type of elements in the merged list.

Parameters

`list1` the first list.

`list2` the second list.

Returns the constructed list.

Throws

`NullPointerException` if `list1` or `list2` are null.

MergedSet

Represents a set providing a view of two sets. This extends the merged collection for implementing the `Set` interface.

Type parameter

`E` the type of the elements in the merged set.

```
package umontreal.iro.lecuyer.collections;

public class MergedSet<E> extends MergedCollection<E>
    implements Set<E>
```

Constructor

```
public MergedSet (Set<? extends E> set1, Set<? extends E> set2)
    Constructs a set merging sets set1 and set2.
```

Parameters

`set1` the first set.

`set2` the second set.

Throws

`NullPointerException` if `set1` or `set2` are null.

Methods

```
public Set<? extends E> getFirstCollection()
    Returns a reference to the first set of this merged set.
```

Returns the first set.

```
public Set<? extends E> getSecondCollection()
    Returns a reference to the second set of this merged set.
```

Returns the second set.

```
public Iterator<E> iterator()
```

Constructs and returns an iterator for the merged set. The returned iterator enumerates all the elements in the first set, then it enumerates the elements of the second set not present in the first set.

MergedMap

Represents a map merging two maps. A merged map is constructed from two maps `map1` and `map2`, contains the keys of both maps, and is immutable. However, any change to the inner maps is reflected on the merged map. Note that if both maps contain the same key, the key appears only once in the merged map, and the corresponding value comes from the entry in the first map. The iterators returned by this implementation enumerates the elements of the first map, then the elements of the second map.

Type parameters

K the type of the keys in the merged map.

V the type of the values in the merged map.

```
package umontreal.iro.lecuyer.collections;
```

```
public class MergedMap<K, V> extends AbstractMap<K, V>
```

Constructor

```
public MergedMap (Map<? extends K, ? extends V> map1, Map<? extends K, ?  
                  extends V> map2)
```

Constructs a new merged map from maps `map1` and `map2`.

Parameters

`map1` the first map.

`map2` the second map.

Throws

`NullPointerException` if `map1` or `map2` are null.

Methods

```
public Map<? extends K, ? extends V> getFirstMap()
```

Returns a reference to the first map in this merged map.

Returns the first map.

```
public Map<? extends K, ? extends V> getSecondMap()
```

Returns a reference to the second map in this merged map.

Returns the second map.

FilteredIterator

Represents an iterator traversing a restricted subset of the elements enumerated by another iterator. A filtered iterator encapsulates an ordinary iterator, and uses it to enumerate objects. However, this iterator only returns objects passing the test implemented in the user-defined `filter (Object)` method. For example, this class could be used to iterate over objects of a certain subclass or having certain properties. Note that this iterator does not support the `remove()` operation.

Type parameter

E the type of the accepted objects.

```
package umontreal.iro.lecuyer.collections;

public abstract class FilteredIterator<E> implements Iterator<E>
```

Constructors

```
public FilteredIterator (Iterator<? super E> it)
```

Constructs a new filtered iterator from the iterator `it`.

Parameter

`it` the iterator being filtered.

Throws

`NullPointerException` if `it` is null.

```
public FilteredIterator (Iterator<? super E> it, int maxNumElements)
```

Constructs a new filtered iterator from the iterator `it`, and returning at most `maxNumElements` elements.

Parameters

`it` the iterator being filtered.

`maxNumElements` the maximal number of elements the iterator can return.

Throws

`NullPointerException` if `it` is null.

`IllegalArgumentException` if `maxNumElements` is negative.

Methods

```
public Iterator<? super E> getInnerIterator()
```

Returns the inner iterator used by this iterator.

Returns the inner iterator.

```
public int getMaxNumElements()
```

Returns the maximal number of elements that can be traversed by this iterator, or `Integer.MAX_VALUE` if the number of elements is not bounded. This does not affect the number of elements traversed by the inner iterator and passed to the `filter (Object)` method.

Returns the maximal number of elements the iterator can return.

```
public abstract boolean filter (Object o)
```

Determines if the object `o` is returned by this iterator. Returns `true` if the object is accepted, `false` otherwise. This iterator assumes that this method returns `true` for a particular object `o` only if `o` can be cast into an instance of `E`.

Parameter

- `o` the tested object.

Returns the result of the test.

FilteredListIterator

Represents a list iterator traversing a restricted subset of the elements enumerated by another list iterator. A filtered list iterator encapsulates an ordinary list iterator, and uses it to enumerate objects. However, this iterator only returns objects passing the test implemented in the user-defined `FilteredIterator.filter (Object)` method. For example, this class could be used to iterate over objects of a certain subclass or having certain properties. Note that this iterator does not support the `add (E)`, `set (E)`, and `remove()` operations.

Type parameter

`E` the type of the accepted objects.

```
package umontreal.iro.lecuyer.collections;

public abstract class FilteredListIterator<E> extends FilteredIterator<E>
    implements ListIterator<E>
```

Constructors

```
public FilteredListIterator (ListIterator<? super E> it)
```

Constructs a new filtered iterator from the iterator `it`. Note that if `it` is not positioned at the beginning of the list, this method has to enumerate all elements of `it` to set the initial value of `nextIndex()`.

Parameter

`it` the iterator being filtered.

Throws

`NullPointerException` if `it` is null.

```
public FilteredListIterator (ListIterator<? super E> it, int
                             maxNumElements)
```

Constructs a new filtered iterator from the iterator `it`, and returning at most `maxNumElements` elements. Note that if `it` is not positioned at the beginning of the list, this method has to enumerate all elements of `it` to set the initial value of `nextIndex()`, which cannot exceed `maxNumElements`.

Parameters

`it` the iterator being filtered.

`maxNumElements` the maximal number of elements the iterator can return.

Throws

`NullPointerException` if `it` is null.

`IllegalArgumentException` if `maxNumElements` is negative.

Method

```
public ListIterator<? super E> getInnerIterator()
```

Returns the inner iterator used by this iterator.

Returns the inner iterator.

ObjectTypeIterator

Represents an iterator traversing objects of a particular class enumerated by another iterator.

Type parameter

E the type of the objects.

```
package umontreal.iro.lecuyer.collections;
```

```
public class ObjectTypeIterator<E> extends FilteredIterator<E>
```

Constructors

```
public ObjectTypeIterator (Iterator<? super E> it, Class<E> objectClass)
```

Constructs a new iterator traversing objects of class `objectClass` enumerated by the inner iterator `it`.

Parameters

`it` the inner iterator.

`objectClass` the object class.

Throws

`NullPointerException` if `it` or `objectClass` are null.

```
public ObjectTypeIterator (Iterator<? super E> it, Class<E> objectClass,  
                           int maxNumElements)
```

Constructs a new iterator traversing at most `maxNumElements` objects of class `objectClass` enumerated by the inner iterator `it`.

Parameters

`it` the inner iterator.

`objectClass` the object class.

`maxNumElements` the maximal number of traversed objects.

Throws

`NullPointerException` if `it` or `objectClass` are null.

`IllegalArgumentException` if `maxNumElements` is negative.

Method

```
public Class<E> getObjectClass()
```

Returns the class of the objects returned by this iterator.

Returns the class of the iterated objects.

ObjectTypeListIterator

Represents a list iterator traversing objects of a particular class enumerated by another iterator.

Type parameter

E the type of the objects.

```
package umontreal.iro.lecuyer.collections;
```

```
public class ObjectTypeListIterator<E> extends FilteredListIterator<E>
```

Constructors

```
public ObjectTypeListIterator (ListIterator<? super E> it, Class<E>  
                                objectClass)
```

Constructs a new iterator traversing objects of class `objectClass` enumerated by the inner iterator `it`.

Parameters

`it` the inner iterator.

`objectClass` the object class.

Throws

`NullPointerException` if `it` or `objectClass` are null.

```
public ObjectTypeListIterator (ListIterator<? super E> it, Class<E>  
                                objectClass, int maxNumElements)
```

Constructs a new iterator traversing at most `maxNumElements` objects of class `objectClass` enumerated by the inner iterator `it`.

Parameters

`it` the inner iterator.

`objectClass` the object class.

`maxNumElements` the maximal number of traversed objects.

Throws

`NullPointerException` if `it` or `objectClass` are null.

`IllegalArgumentException` if `maxNumElements` is negative.

Method

```
public Class<E> getObjectClass()
```

Returns the class of the objects returned by this iterator.

Returns the class of the iterated objects.

Matrix

Represents a two-dimensional matrix of objects. Each element of a matrix can be referenced using a two-dimensional index (r, c) , where $r = 0, \dots, R - 1$ is the row index, and $c = 0, \dots, C - 1$ is the column index. Methods are specified by this interface to resize matrices by adding or removing rows or columns. A class implementing this interface might implement the `RandomAccess` interface. This means that the `get (int, int)` operation is efficient, and that the returned views also implement `RandomAccess`.

Type parameter

E the type of the elements.

```
package umontreal.iro.lecuyer.collections;
```

```
public interface Matrix<E> extends Collection<E>
```

Methods

```
public int rows()
```

Returns the number of rows in this matrix.

Returns the number of rows in this matrix.

```
public int columns()
```

Returns the number of columns in this matrix.

Returns the number of columns in this matrix.

```
public void setRows (int numRows)
```

Sets the number of rows of this matrix to `numRows`. If `numRows` is smaller than `rows()`, the last rows of the matrix are removed. If `numRows` is greater than `rows()`, new rows filled with `null` references are added to the matrix. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Parameter

`numRows` the new number of rows in the matrix.

Throws

`IllegalArgumentException` if `numRows` is negative.

`UnsupportedOperationException` if this method is not supported.

```
public void setColumns (int numColumns)
```

Sets the number of columns of this matrix to `numColumns`. If `numColumns` is smaller than `columns()`, the last columns of the matrix are removed. If `numColumns` is greater than `columns()`, new columns filled with `null` references are added to the matrix. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Parameter

`numColumns` the new number of columns in the matrix.

Throws

`IllegalArgumentException` if `numColumns` is negative.

`UnsupportedOperationException` if this method is not supported.

```
public E get (int r, int c)
```

Returns the element at index (`r`, `c`) of the matrix.

Parameters

`r` the row index.

`c` the column index.

Returns the value of the element.

Throws

`IndexOutOfBoundsException` if `r` or `c` are negative, if `r` is greater than or equal to `rows()`, or if `c` is greater than or equal to `columns()`.

```
public E set (int r, int c, E value)
```

Sets the element at index (`r`, `c`) of the matrix to `value`, and returns the element previously at that position. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Parameters

`r` the row index.

`c` the column index.

`value` the value of the element.

Returns the previous value of the element.

Throws

`IndexOutOfBoundsException` if `r` or `c` are negative, if `r` is greater than or equal to `rows()`, or if `c` is greater than or equal to `columns()`.

`UnsupportedOperationException` if this method is not implemented.

```
public Iterator<E> iterator()
```

Constructs and returns an iterator traversing the elements of this matrix rowise.

Returns the constructed iterator.

```
public List<E> asList()
```

Returns a list view of this matrix. Element $i=rC + c$ of the returned list must correspond to element (r, c) of the matrix. As a result, when considering a matrix as a list and iterating over elements, elements are enumerated rowise. However, elements cannot be added or removed from the returned list, because the backing matrix must remain rectangular. One can however use `List.set (int, E)` to change elements. The behavior of the returned list is undefined if the dimensions of the backing matrix are changed. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Returns the list view.

Throws

`UnsupportedOperationException` if this method is not implemented.

```
public List<E> viewRow (int r)
```

Returns a list representing a view of row r of this matrix. Any change to this matrix is reflected on the returned list while the elements of the returned list can be modified using `List.set (int, E)`. The behavior of the returned list is undefined if the dimensions of the backing matrix are changed. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Parameter

r the index of the row to get a view for.

Returns the view of the selected row.

Throws

`IndexOutOfBoundsException` if r is negative or greater than or equal to `rows()`.

`UnsupportedOperationException` if this method is not implemented.

```
public List<E> viewColumn (int c)
```

Returns a list representing a view of column c of this matrix. Any change to this matrix is reflected on the returned list while the elements of the returned list can be modified using `List.set (int, E)`. The behavior of the returned list is undefined if the dimensions of the backing matrix are changed. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Parameter

c the index of the column to get a view for.

Returns the view of the selected column.

Throws

`IndexOutOfBoundsException` if `c` is negative or greater than or equal to `columns()`.

`UnsupportedOperationException` if this method is not implemented.

```
public Matrix<E> viewPart (int fromRow, int fromColumn, int toRow, int  
                           toColumn)
```

Returns a view of a portion of this matrix containing rows `fromRow` (inclusive) to `toRow` (exclusive), and columns `fromColumn` (inclusive) to `toColumn` (exclusive). The behavior of the returned matrix is undefined if the dimensions of the backing matrix are changed. This method is optional, and throws an `UnsupportedOperationException` if not implemented.

Parameters

`fromRow` the starting row.

`fromColumn` the ending row.

`toRow` the starting column.

`toColumn` the ending column.

Returns the view of the matrix.

Throws

`IndexOutOfBoundsException` if row or column indices are out of bounds.

`IllegalArgumentException` if `fromRow` is greater than `toRow`, or `fromColumn` is greater than `toColumn`.

`UnsupportedOperationException` if this method is not implemented.

```
public Object[] [] to2DArray()
```

Returns a 2D array containing the elements of this matrix in the proper sequence.

Returns the array containing the elements of this matrix.

```
public E[] [] to2DArray (E[] [] array)
```

Returns a 2D array containing the elements of this matrix in the proper sequence; the runtime type of the returned array is the same as the runtime type of the given array.

Parameter

`array` the array to use.

Returns the array containing the elements of this matrix.

`public boolean equals (Object o)`

Compares the specified object with this matrix for equality. Returns `true` if and only if the specified object is also a matrix, both matrices have the same dimensions, and all corresponding pairs of elements in the two matrices are equal. (Two elements `e1` and `e2` are equal if `(e1 == null ? e2 == null : e1.equals (e2))`.) In other words, two matrices are defined to be equal if they contain the same elements in the same order. This definition ensures that the `equals` method works properly across different implementations of the `Matrix` interface.

`public int hashCode()`

Returns the hash code value for the matrix. The hash code of a matrix is defined to be $31 * (31 * R + C) + H$, where H is the hash code of the list view returned by `asList()`. This ensures that `matrix1.equals (matrix2)` implies that `matrix1.hashCode() == matrix2.hashCode()` for any two matrices, `matrix1` and `matrix2`, as required by the general contract of `Object.hashCode`.

AbstractMatrix

Provides default implementation for most methods of the `Matrix` interface.

Type parameter

`E` the type of the elements.

```
package umontreal.iro.lecuyer.collections;

public abstract class AbstractMatrix<E> extends AbstractCollection<E>
    implements Matrix<E>
```

Field

```
protected int modCount
```

This must be incremented each time `Matrix.setRows (int)` or `Matrix.setColumns (int)` modify the number of rows or columns.

Methods

```
public List<E> asList()
```

Returns a list using `size()` to get the number of elements, and `Matrix.get (int, int)` to access elements.

```
public List<E> viewRow (int r)
```

Returns a list using `Matrix.columns()` to get the number of elements, and `Matrix.get (int, int)` to access elements.

```
public List<E> viewColumn (int c)
```

Returns a list using `Matrix.rows()` to get the number of elements, and `Matrix.get (int, int)` to access elements.

```
public int size()
```

Returns the product of `Matrix.rows()` and `Matrix.columns()`.

```
public boolean isEmpty()
```

Returns `true` if `Matrix.rows()` or `Matrix.columns()` return 0.

DenseMatrix

Represents a matrix storing elements in an array.

Type parameter

`E` the element type.

```
package umontreal.iro.lecuyer.collections;
```

```
public class DenseMatrix<E> extends AbstractMatrix<E>
    implements Cloneable, Serializable, RandomAccess
```

Constructors

```
public DenseMatrix (int rows, int columns)
```

Constructs a new dense matrix with `rows` rows, `columns` columns, and filled with `null` elements.

Parameters

`rows` the number of rows.

`columns` the number of columns.

```
public DenseMatrix (E[] [] elements)
```

Constructs a new matrix from the 2D array `elements`.

Parameter

`elements` the elements of the matrix.

```
public DenseMatrix (Matrix<? extends E> matrix)
```

Constructs a new matrix from the matrix `matrix`.

Parameter

`matrix` the source matrix.

DescendingOrderComparator

Represents a comparator sorting objects in descending natural order. More specifically, when comparing two objects, this comparator returns the result of `Comparable.compareTo (T)` multiplied by -1 .

Type parameter

T

```
package umontreal.iro.lecuyer.collections;
```

```
public class DescendingOrderComparator<T extends Comparable<? super T>>  
    implements Comparator<T>
```

References

- [1] S. Bailliez, N. K. Barozzi, et al. *Apache Ant User Manual*. Apache, 2004. See <http://ant.apache.org/manual>.
- [2] Wolfgang Hoschek. *The Colt Distribution: Open Source Libraries for High Performance Scientific and Technical Computing in Java*. CERN, Geneva, 2004. Available at <http://dsd.lbl.gov/~hoschek/colt/>.
- [3] A. Le Hors, Philippe Le Hégarret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. *Document Object Model Level 2 Core*, November 2000. See <http://www.w3.org/TR/DOM-Level-2-Core>.
- [4] P. L’Ecuyer. *SSJ: A Java Library for Stochastic Simulation*, 2004. Software user’s guide, available at <http://www.iro.umontreal.ca/~lecuyer>.
- [5] F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. *Extensible Markup Language (XML) 1.0*. W3C Recommendation, third edition, February 2004. Also available from <http://www.w3.org/TR/REC-xml>.