

IFT 6800

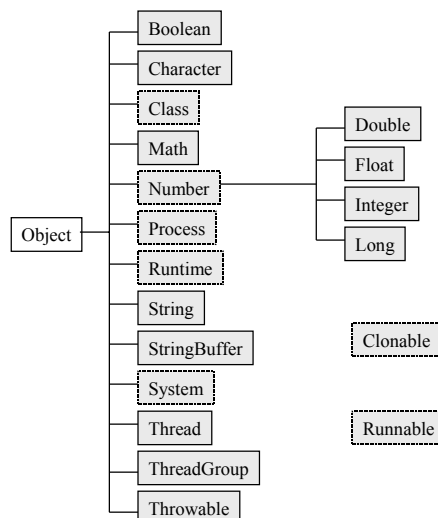
Atelier en Technologies d'information

Cours 7: Le langage de programmation Java
chapitre 5: Les API Java



1

java.lang.*



java.lang.String (1)

- La classe `String` gère des chaînes de caractères (`char`).
- Une `String` n'est pas modifiable.
- Toute modification entraîne la création d'une nouvelle `String`.
- Les valeurs littérales ("`abc`") sont transformées en `String`.
- L'opérateur `+` permet la concaténation de 2 `String`.

Java.lang.String (2)

```
String s = "\u00c4tre ou ne pas \u00eatre"; // s = "Être ou ne pas être"
int lg = s.length();                       // lg = 19
String s = "Java" + "Soft";                 // s = "JavaSoft"

String s = (String) new URL("http://server/big.txt").getContent();

char[] data = {'J', 'a', 'v', 'a'};
String name = new String(data);

String s = String.valueOf(2 * 3.14159); // s = "6.28318"
String s = String.valueOf(new Date()); // s = "Sat Jan 18 12:10:36 GMT+0100 1997"
int i = Integer.valueOf("123");         // i = 123

String s = "java";

if (s == "java") {...}                    // Erreur
if (s.equals("java") {...})               // Ok
```

java.lang.StringBuffer

- La classe `StringBuffer` gère des chaînes de caractères (`char`) modifiable (`setCharAt()`, `append()`, `insert()`)
- La méthode `toString()` convertie une `StringBuffer` en `String` (pas de copie, le même tableau est partagé, jusqu'à modification)

```
StringBuffer sb = "abc"; // Error: can't convert String to StringBuffer
StringBuffer sb = new StringBuffer("abc");

sb.setCharAt(1, 'B'); // sb= "aBc"
sb.insert(1, "1234"); // sb = "a1234Bc"
sb.append("defg"); // sb = "a1234Bcdefg"
String s = sb.toString(); // s = "a1234Bcdefg"
sb.append("hij"); // sb = "a1234Bcdefghij" s =
```

java.lang.Class

- La classe `Class` représente une classe java.
- Elle n'est pas instanciable
- Elle permet de créer dynamiquement des nouvelles instances (mais seul le constructeur par défaut est appelé)

```
Class classname = Class.forName("java.util.Date");
Date d = (Date)classname.newInstance();

System.out.println("Date : " + d);

Integer i = classname.getMethod("getMinutes", null).invoke(d, null);
```

java.lang.Thread (1)

- Cette classe permet de déléguer le traitement d'un objet par une nouvelle thread.
- 2 possibilités : soit hériter de `Thread` soit implémenter `Runnable`

```
class C1 extends Thread
{
    public C1() { this.start(); }
    public void run() {...}
}

class C2 implements Runnable
{
    public C2() {Thread t = new Thread(this); t.start(); }
    public void run() {...}
}
```

java.lang.Thread (2)

- Méthodes :
 - `void start()`
 - `void stop()`
 - `void suspend()`
 - `void resume()`
 - `static void sleep()`

java.lang.Thread (3)

- Le mot réservé `synchronized` permet de synchroniser l'accès à une partie de code où à une méthode.

```
class Banque {
    synchronized void ajouter(int montant) {...}
    synchronized void retirer(int montant) {...}
}

class Client implements Runnable {
    Banque b;
    public Client(Banque b) {
        this.b = b;
        Thread t = new Thread(this); t.start();
    }
    public void run() { ... b.ajouter(100); ... b.retirer(10); ... }
}

Banque b = new Banque();
Client c1 = new Client (b); Client c2 = new Client(b);
```

java.lang.Thread (4)

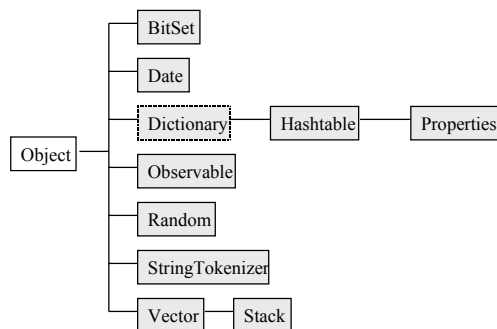
- Chaque objet possède les méthode `wait()`, `notify()` et `notifyAll()`
- Dans une une partie `synchronized` d'un objet O :
 - `wait()` relache le verrou et se met en attente.
 - `notify()` reveille un *thread* en attente (fifo)
 - `notifyAll()` reveille tous les *threads* en attente

```
class MyThing {
    synchronized void waiterMethod() {...; wait(); ...}
    synchronized void notifyMethod() {...; notify(); ...}
    synchronized void anOtherMethod() {...}
}
```

java.lang.Thread (5)

- Scheduling et priorité :
 - Le scheduling est en partie dépendant des implémentations
 - `setPriority()` permet de fixer la priorité d'une *thread*
 - Pour 2 threads de même priorité, par défaut : *round robin*
 - T1 cède la place à T2 quand `sleep()`, `wait()`, bloque sur un `synchronized`, `yield()`, `stop()`
 - Certaines JVM implémentent un *time slicing* (Win32, NN, IE, ...)

java.util.*



java.util.Hashtable

- Cette classe gère une collection d'objets au travers d'une table de hachage dont les clés sont des `String` et les valeurs associées des `Object`.

```
Hashtable ht = new Hashtable();

ht.put("noel", new Date("25 Dec 1997"));

ht.put("un vecteur", new Vector());

Vector v = (Vector)ht.get("un vecteur");

for(Enumeration e = ht.keys(); e.hasMoreElements();){
    String key = (String)e.nextElement;
    ...
}
```

java.util.Properties

- Cette classe gère une collection d'objets au travers d'une table de hachage dont les clés et les valeurs sont des `String`.

```
Properties p = new Properties();

p.put("&acute;", "\'e");
p.put("&grave;", "`e");
p.put("&circ", "^e");

String s = p.getProperty("&acute;");

for(Enumeration e = p.keys(); e.hasMoreElements();){
    String key = (String)e.nextElement;
    ...
}
```

java.util.StringTokenizer

- Cette classe permet de découper une `String` selon des séparateurs.

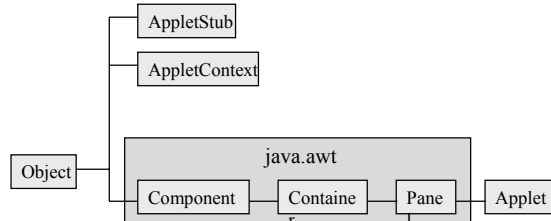
```
String str = "avion, bateau ; train ";  
  
StringTokenizer st = new StringTokenizer(str, ";, ");  
  
System.out.println(st.nextToken()); // --> avion  
System.out.println(st.nextToken()); // --> bateau  
System.out.println(st.nextToken()); // --> train
```

java.util.Vector

- Cette classe gère une collection d'objets dans un tableau dynamique.

```
Vector v = new Vector();  
  
v.addElement("une chaine");  
v.addElement(new Date());  
v.addElement(new String[]);  
v.addElement(new Vector());  
  
v.setElementAt("abcde", 2);  
System.out.println(v.elementAt(2)); // --> abcde
```

java.applet.*



java.applet.Applet (1)

- Une applet est une classe compilée héritant de `java.applet.Applet`
- Elle est diffusé par un serveur web dans une page HTML

```
<APPLET code='TiffViewer.class' width=50 height=50>  
  <PARAM name='imagesource' value='mon_image.tiff'>  
</APPLET>
```

- Elle est téléchargée puis exécutée par le browser.
- Elle est soumise au *Security Manager* du browser :
 - pas d'accès en lecture ni en écriture sur le disque du browser.
 - connexion réseau uniquement sur le serveur d'origine.
 - pas de chargement de librairie native.
 - pas de lancement de processus, ...

java.applet.Applet (2)

■ Structure d'une applet

```
public class MyApplet extends java.applet.Applet
{
    public void init() {...}
    public void start() {...}
    public void paint(java.awt.graphics g) {...}
    public void stop() {...}
    public void destroy() {...}
}
```

java.applet.Applet (3)

■ Diffusion de l'applet

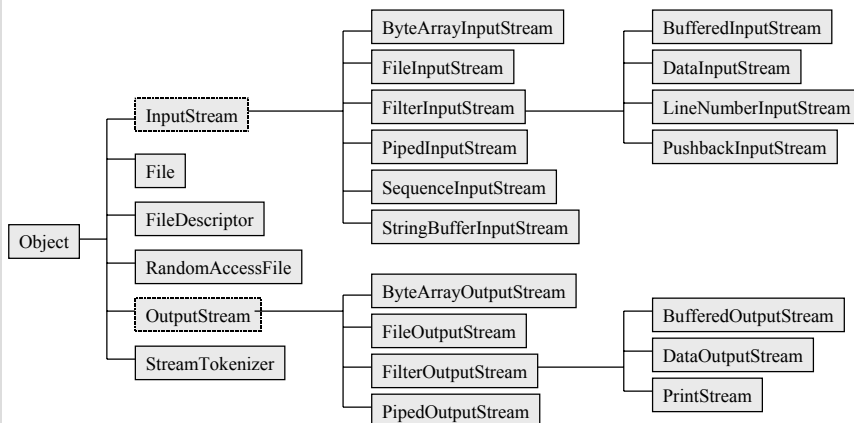
```
<HTML>
  <BODY>
    <APPLET code="MyApplet"
      codebase="http://falconet.inria.fr/~dedieu/applets/"
      width=300 height=200>
    <PARAM name="message" value="Hello World">
    </APPLET>
  </BODY>
</HTML>
```

java.applet.Applet (4)

■ Quelques methodes :

```
String msg = this.getParameter("message");  
  
this.showStatus("Applet en cours");  
  
Image img = this.getImage(new URL("http://falconet/image.gif"));  
  
AppletContext ctxt = this.getAppletContext();  
ctxt.showDocument(new URL("http://falconet/page.html"), "frame");
```

java.io.*



java.io.File

- Cette classe fournit une définition *platform-independent* des fichiers et des répertoires.

```
File f = new File("/etc/passwd");
System.out.println(f.exists()); // --> true
System.out.println(f.canRead()); // --> true
System.out.println(f.canWrite()); // --> false
System.out.println(f.getLength()); // --> 11345

File d = new File("/etc/");
System.out.println(d.isDirectory()); // --> true

String[] files = d.list();
for(int i=0; i < files.length; i++)
    System.out.println(files[i]);
```

java.io.File(InputStream|OutputStream)Stream

- Ces classes permettent d'accéder en lecture et en écriture à un fichier.

```
FileInputStream fis = new FileInputStream("source.txt");
byte[] data = new byte[fis.available()];
fis.read(data);
fis.close();

FileOutputStream fos = new FileOutputStream("cible.txt");
fos.write(data);
fos.close();
```

java.io.Data(Input|Output)Stream

- Ces classes permettent de lire et d'écrire des types primitifs et des lignes sur des flux.

```
FileInputStream fis = new FileInputStream("source.txt");
DataInputStream dis = new DataInputStream(fis);

int i    = dis.readInt();
double d = dis.readDouble();
String s = dis.readLine();

FileOutputStream fos = new FileOutputStream("cible.txt");
DataOutputStream dos = new DataOutputStream(fos);

dos.writeInt(123);
dos.writeDouble(123.456);
dos.writeChars("Une chaine");
```

java.io.PrintStream

- Cette classe permet de manipuler un `OutputStream` au travers des méthode `print()` et `println()`.

```
PrintStream ps = new PrintStream(new FileOutputStream("cible.txt"));

ps.println("Une ligne");
ps.println(123);
ps.print("Une autre ");
ps.print("ligne");
ps.flush();
ps.close();
```

java.io.Object(Input|Output)Stream (1)

- Ces classes permettent de lire et d'écrire des objets, implémentant `java.io.Serializable`, sur des flux.

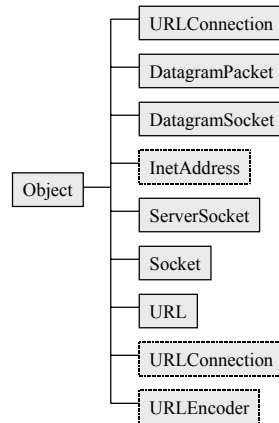
```
// Ecriture
FileOutputStream fos = new FileOutputStream("tmp");
ObjectOutput oos = new ObjectOutputStream(fos);
oos.writeObject("Today");
oos.writeObject(new Date());
oos.flush();

// Lecture
FileInputStream fis = new FileInputStream("tmp");
ObjectInputStream ois = new ObjectInputStream(fis);
String today = (String)ois.readObject();
Date date = (Date)ois.readObject();
```

java.io.Object(Input|Output)Stream(2)

- Par défaut, tous les champs sont sérialisés (y compris `private`)
- Cela peut poser des problèmes de sécurité
- 3 solutions :
 - Ne pas implémenter `Serializable`
 - Réécrire les méthodes `writeObject()` et `readObject()`
 - Le mot clé `transient` permet d'indiquer qu'un champ ne doit pas être sérialisé.

java.net.*



java.net.Socket

- Cette classe implémente une socket TCP coté client.

```
String serveur = "www.inria.fr";
int port = 80;

Socket s = new Socket(serveur, port);

PrintStream ps = new PrintStream(s.getOutputStream());
ps.println("GET /index.html");

DataInputStream dis = new DataInputStream(s.getInputStream());

String line;
while((line = dis.readLine()) != null)
    System.out.println(line);
```

java.net.ServerSocket

- Cette classe implémente une socket TCP coté serveur.

```
int port_d_ecoute = 1234;
ServerSocket serveur = new ServerSocket(port_d_ecoute);

while(true)
{
    Socket socket_de_travail = serveur.accept();
    new ClasseQuiFaitLeTraitement(socket_travail);
}
```

java.net.DatagramSocket (1)

- Cette classe implémente une socket UDP

```
// Client

Byte[] data = "un message".getBytes();

InetAddress addr = InetAddress.getByName("falconet.inria.fr");

DatagramPacket packet = new DatagramPacket(data, data.length, addr,
1234);

DatagramSocket ds = new DatagramSocket();

ds.send(packet);
ds.close();
```

java.net.DatagramSocket (2)

```
// Serveur

DatagramSocket ds = new DatagramSocket(1234);

while(true)
{
    DatagramPacket packet = new DatagramPacket(new byte[1024],
    1024);
    s.receive(packet);
    System.out.println("Message: " + packet.getData());
}
```

java.net.MulticastSocket (1)

- Cette classe implémente une socket multicast (UDP)

```
// Client

Byte[] data = "un message".getBytes();

InetAddress addr = InetAddress.getByName("falconet.inria.fr");

DatagramPacket packet = new DatagramPacket(data, data.length, addr,
1234);

MulticastSocket s = new MulticastSocket();

s.send(packet, (byte)1);
s.close();
```

java.net.MulticastSocket (2)

```
// Serveur

MulticastSocket s = new MulticastSocket(1234);
System.out.println("I listen on port " + s.getLocalPort());

s.joinGroup(InetAddress.getByName("falconet.inria.fr"));

DatagramPacket packet = new DatagramPacket(new byte[1024], 1024);

s.receive(packet);

System.out.println("from: " + packet.getAddress());
System.out.println("Message: " + packet.getData());

s.leaveGroup(InetAddress.getByName("falconet.inria.fr"));
s.close();
```

java.net.URL

```
URL url = new URL("http://falconet.inria.fr/index.html");

DataInputStream dis = new DataInputStream(url.openStream());

String line;
while ((line = dis.readLine()) != null)
    System.out.println(line);
```