Chapter 12: Data Structures

Presentation slides for

Java Software Solutions

Foundations of Program Design

Second Edition

by John Lewis and William Loftus

Java Software Solutions is published by Addison-Wesley

Data Structures

- **We can now explore some advanced techniques for organizing and managing information**

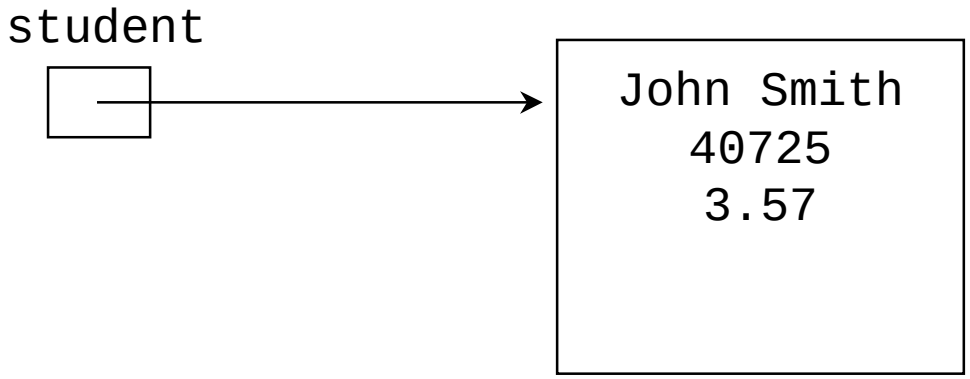
- **Chapter 12 focuses on:**
  - **dynamic structures**
  - **Abstract Data Types (ADTs)**
  - **linked lists**
  - **queues**
  - **stacks**

Static vs. Dynamic Structures

- **A *static* data structure has a fixed size**
- **This meaning is different than those associated with the `static` modifier**
- **Arrays are static; once you define the number of elements it can hold, it doesn't change**
- **A *dynamic* data structure grows and shrinks as required by the information it contains**

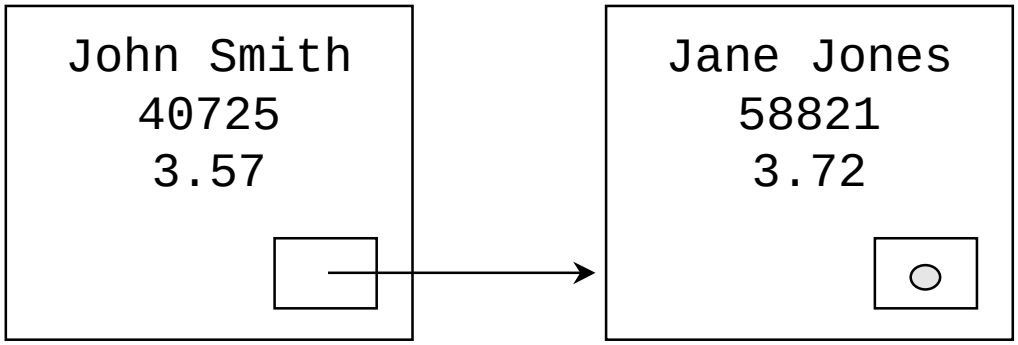
Object References

- **Recall that an *object reference* is a variable that stores the address of an object**
- **A reference can also be called a *pointer***
- **They are often depicted graphically:**



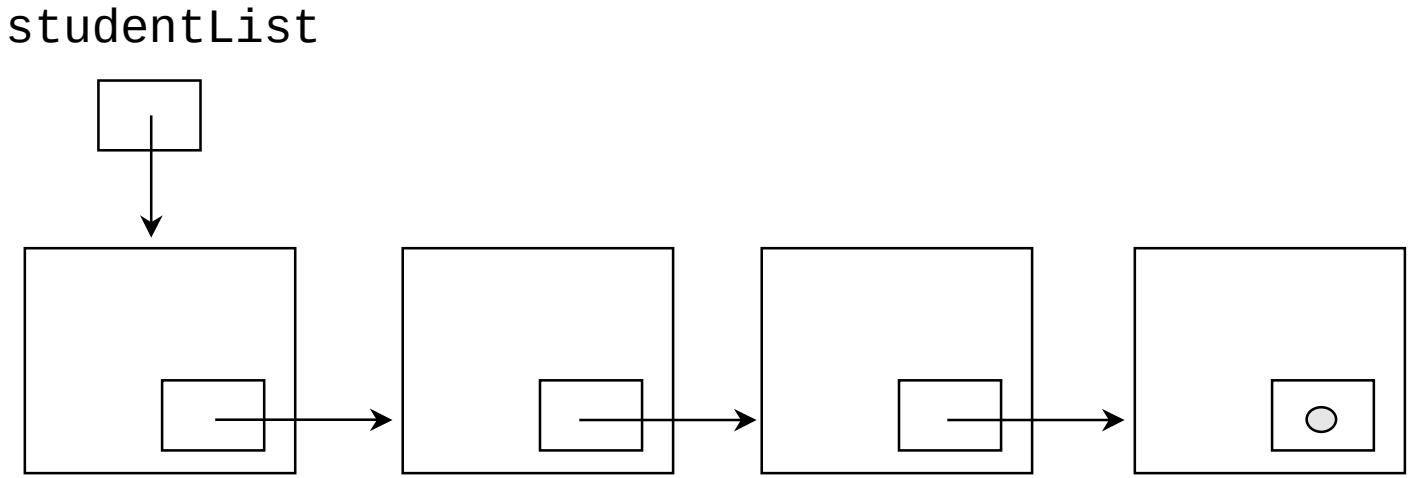
References as Links

- Object references can be used to create *links* between objects
- Suppose a `Student` class contained a reference to another `Student` object



References as Links

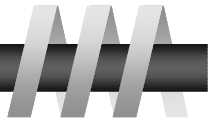
- **References can be used to create a variety of linked structures, such as a *linked list*:**



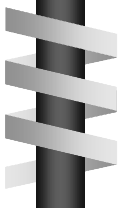
Abstract Data Types

- **An *abstract data type* (ADT) is an organized collection of information and a set of operations used to manage that information**
- **The set of operations define the *interface* to the ADT**
- **As long as the ADT accurately fulfills the promises of the interface, it doesn't really matter how the ADT is implemented**
- **Objects are a perfect programming mechanism to create ADTs because their internal details are *encapsulated***

Abstraction



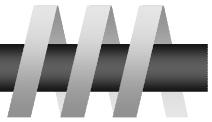
- Our data structures should be abstractions
- That is, they should hide details as appropriate
- We want to separate the interface of the structure from its underlying implementation
- This helps manage complexity and makes the structures more useful



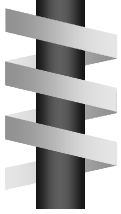
Intermediate Nodes

- The objects being stored should not have to deal with the details of the data structure in which they may be stored
- For example, the `Student` class stored a link to the next Student object in the list
- Instead, we can use a separate node class that holds a reference to the stored object and a link to the next node in the list
- Therefore the internal representation actually becomes a linked list of nodes

Book Collection

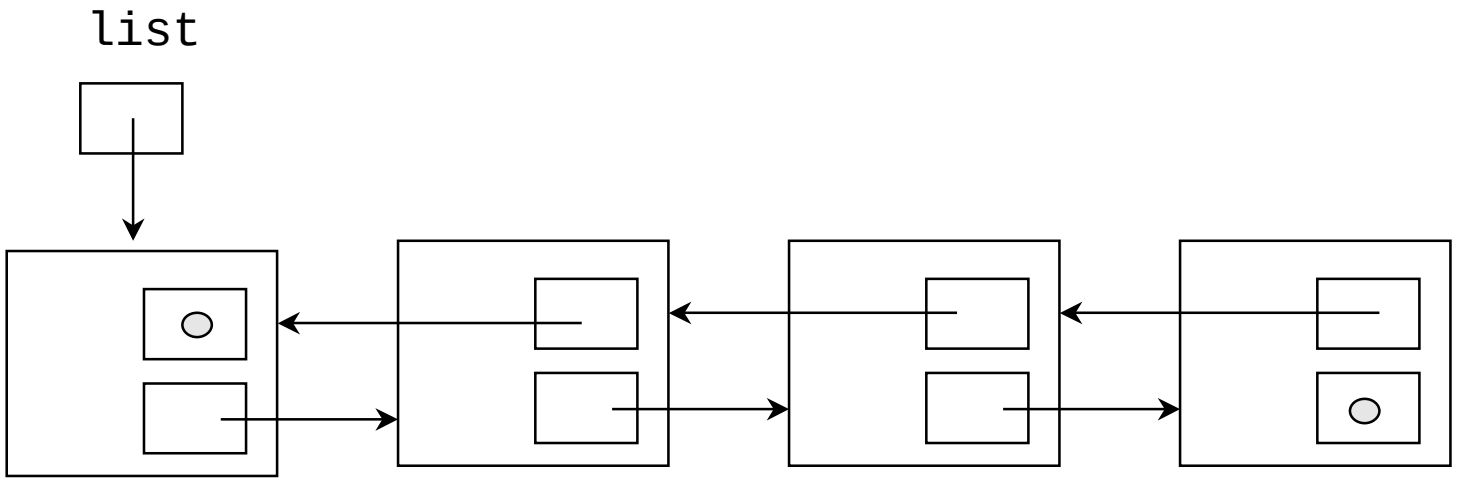


- Let's explore an example of a collection of `Book` objects
- The collection is managed by the `BookList` class, which has an private inner class called `BookNode`
- Because the `BookNode` is private to `BookList`, the `BookList` methods can directly access `BookNode` data without violating encapsulation
- See `Library.java` (page 500)
- See `BookList.java` (page 501)
- See `Book.java` (page 503)



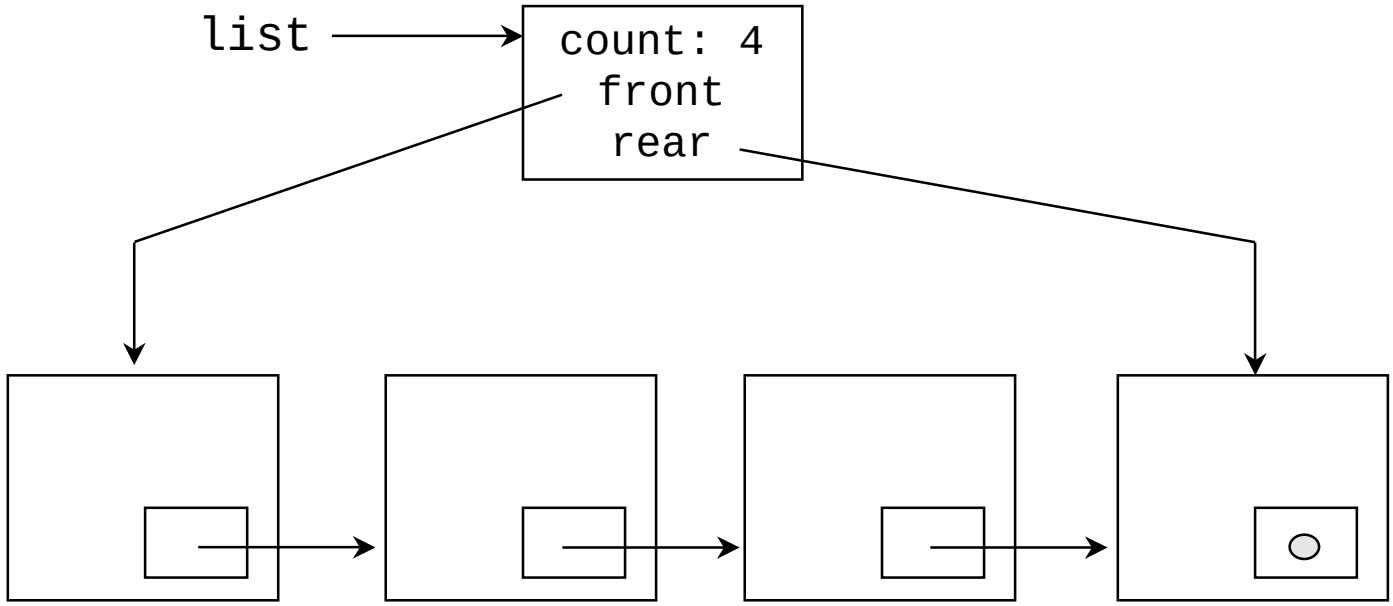
Other Dynamic List Implementations

- **It may be convenient to implement as list as a *doubly linked list*, with next and previous references:**



Other Dynamic List Implementations

- **It may also be convenient to use a separate header node, with references to both the front and rear of the list**



Queues

- **A *queue* is similar to a list but adds items only to the end of the list and removes them from the front**
- **It is called a FIFO data structure: First-In, First-Out**
- **Analogy: a line of people at a bank teller's window**

Queues

- We can define the operations on a queue as follows:
  - enqueue - add an item to the rear of the queue
  - dequeue - remove an item from the front of the queue
  - empty - returns true if the queue is empty

- As with our linked list example, by storing generic `Object` references, any object can be stored in the queue

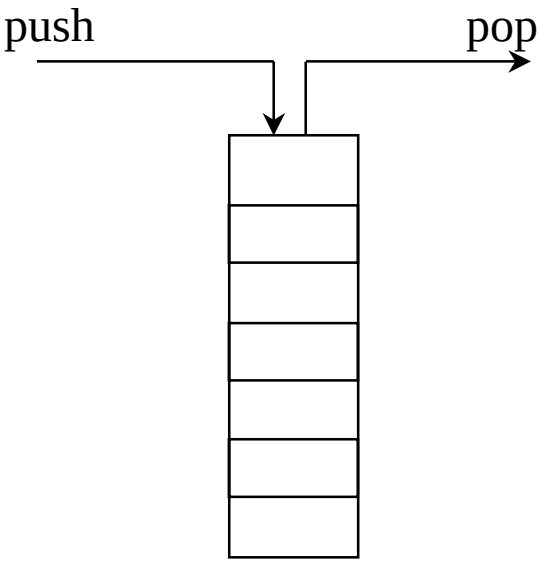
- Queues are often helpful in simulations and any processing in which items get "backed up"

Stacks

- **A *stack* ADT is also linear, like a list or queue**
- **Items are added and removed from only one end of a stack**
- **It is therefore LIFO: Last-In, First-Out**
- **Analogy: a stack of plates**

Stacks

- **Stacks are often drawn vertically:**



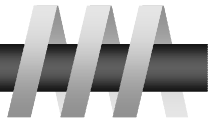
Stacks

- **Some stack operations:**
  - **push - add an item to the top of the stack**
  - **pop - remove an item from the top of the stack**
  - **peek - retrieves the top item without removing it**
  - **empty - returns true if the stack is empty**

- **The `java.util` package contains a `Stack` class, which is implemented using a `Vector`**

- **See `Decode.java` (page 508)**

Collection Classes



- The Java 2 platform contains a Collections API
- This group of classes represent various data structures used to store and manage objects
- Their underlying implementation is implied in the class names, such as `ArrayList` and `LinkedList`
- Several interfaces are used to define operations on the collections, such as `List`, `Set`, `SortedSet`, `Map`, and `SortedMap`