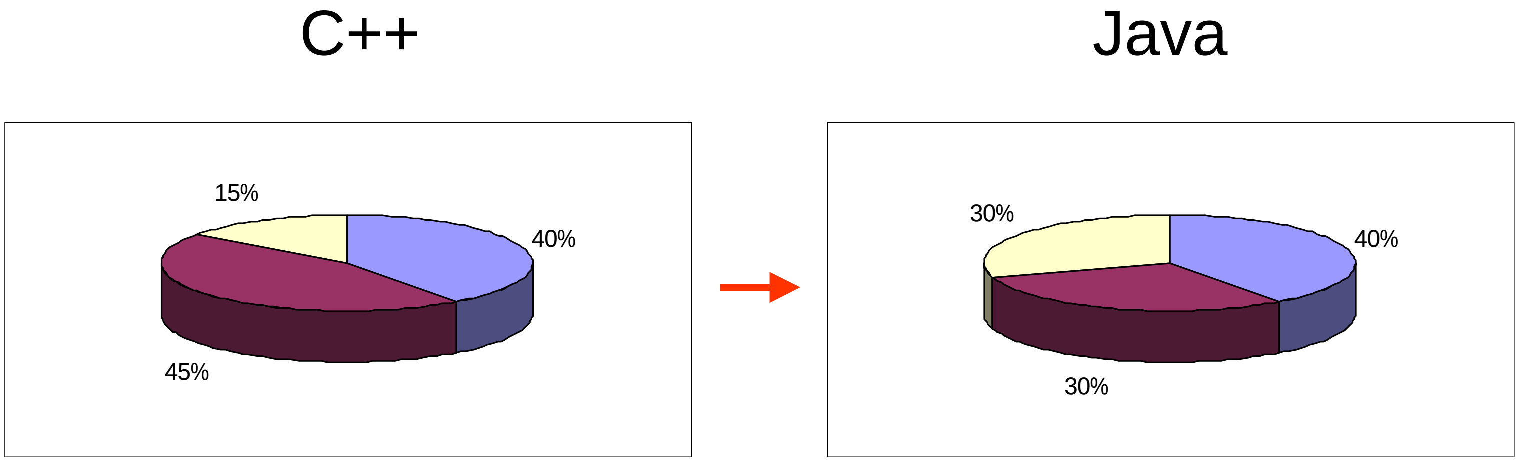
Data Structures in Java

Maria Litvin

Phillips Academy, Andover, MA

mlitvin@andover.edu

College and AP Emphasis:



🟦 Concepts

🟥 Implementation

🟨 Library classes and interfaces

Collections Framework

- The AP subset:
  - List, Stack, Queue
  - Set, Map, Priority Queue
- In Java, these collections hold objects (not ints or doubles)
- In Java, abstract collections are represented by *interfaces*

Interfaces

- An *interface* lists methods without code
- A class that implements an interface must supply definitions (code) for all the methods of the interface
- interface and implements are Java reserved words
- Different classes can implement the same interface in different ways

Example of an interface:

```
public interface Stack*
{
   boolean isEmpty ();
   void push (Object x);
   Object pop ();
   Object peekTop ();
}
```

Describes what any class that implements this interface can do

* Adapted from The College Board’s *AP Computer Science AB: Implementation Classes and Interfaces*

Example of a class that implements the Stack interface:

```
public class MyStack implements Stack
{
   // Constructor:
   public MyStack()  { items = new Object [16]; sp = 0; }

   // Methods:
   public isEmpty ()  { return sp == 0; }
   public void push (Object x)  { items [sp] = x; sp++; }
   public Object pop ()  { sp--; return items [sp]; }
   public Object peekTop ()  { return items [sp - 1]; }

   // Fields (data members):
   private Object [ ]  items;
   private int sp;
}
```

The same methods as in the Stack interface; additional methods are allowed

Six Collections → Java Interfaces

- List → java.util.List
- Stack → The College Board’s Stack
- Queue → The College Board’s Queue
- Set → java.util.Set
- Map → java.util.Map
- Priority Queue → The CB’s PriorityQueue

List

- Holds numbered (indexed) items

  $x_0, x_1, \ldots, x_{n\text{-}1}$

- Can hold duplicate values
- Provides methods to retrieve, replace, add, and remove items

List Applications

- A mailing list
- A waiting list
- etc.

java.util.List Interface

(AP Subset of methods)

```
int size ();                       // Returns the number of items
Object get (index);                // Returns the value at index
Object set (index, obj);           // Replaces the value at index
                                   //    and returns the old value
boolean add (obj);                 // Appends obj at the end,
                                   //    returns true
void add (index, obj);             // Inserts obj at index
Object remove (index);             // Removes and returns
                                   //    the value at index

Iterator iterator ();              // Returns an Iterator
ListIterator listIterator ();      // Returns a ListIterator
```

A reminder:

- In Java, all objects are represented by references to them (their addresses)
- References are similar to pointers in C++

A misconception:

- “Java has no pointers...”

In truth:

- Java has only pointers (for objects)

C++:

```
ListNode node (...);

ListNode *nodePtr =
    new ListNode (...);
```

Java:

```
ListNode node =
    new ListNode (...);
```

- Understanding of pointers is helpful for handling objects in Java

In particular:

- A list “of objects” actually holds references to (addresses of) objects
- A list can hold several references for the same object
- A list can hold “equal” objects — `obj1.equals(obj2)`
- The same object can belong to several lists
- An object can change after it is added to a list

List Implementation 1: java.util.ArrayList

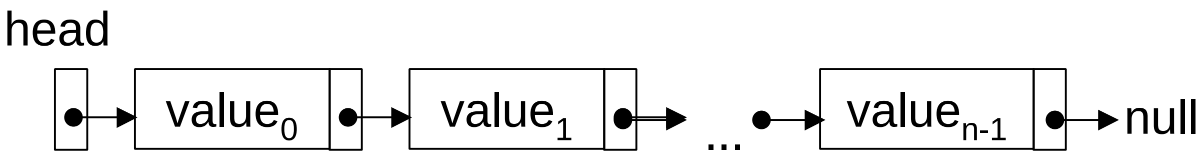
- Implements a list as an array with direct-access to elements
- The no-args constructor creates an empty list of some default capacity
- The capacity is doubled automatically (and all the elements are copied into the new array) when the array runs out of space
- Throws IndexOutOfBoundsException if an index is out of range

ArrayList Example: Traversal

```
import java.util.ArrayList;
...

   ArrayList list = new ArrayList ();
   list.add ("Austin");
   list.add ("Boston");
   ...
   int i;
   for (i = 0; i < list.size(); i++)
   {
       System.out.println ( list.get (i) );
   }
```

List Implementation 2: java.util.LinkedList

- Implements a list as a linked list



- Easy to insert and remove values in the middle of the list
- Takes time to go to the *k*-th element
- The no-args constructor creates an empty list

LinkedList Example: Traversal

```
import java.util.LinkedList;
...

   LinkedList list = new LinkedList ();
   list.add ("Austin");
   list.add ("Boston");
   ...
   int i;
   for (i = 0; i < list.size(); i++)
   {
       System.out.println ( list.get (i) );
   }
```

Inefficient!

(Each time starts counting from the beginning to find *i*-th node)

Solution?

Continued ⇒

Iterators

```
import java.util.LinkedList;
import java.util.Iterator;
...

   LinkedList list = new LinkedList ();
   list.add ("Austin");
   list.add ("Boston");
   ...
   Iterator it = list.iterator ();
   while ( it.hasNext () )
   {
      System.out.println ( it.next () );
   }
```

iterator is another method in the List interface

Iterators work for both ArrayList and LinkedList

LinkedList’s

Additional AP Subset Methods

```
void addFirst (Object obj);
void addLast (Object obj);
Object getFirst ();
Object getLast ();
Object removeFirst ();
Object removeLast ();
```

“Do-It-Yourself” Programming of Linked Lists

- The College Board provides a class, ListNode, which implements a node of a linked list
- AP (AB) exam may include free-response questions that involve writing methods that manipulate a linked list (with nodes represented by ListNode objects)

```
public class ListNode*
{
  private Object value;
  private ListNode next;

  // Constructor:
  public ListNode (Object initValue, ListNode initNext)
           { value = initValue;  next = initNext; }

  public Object getValue ()  { return value; }
  public ListNode getNext ()  {  return next; }
  public void setValue (Object theNewValue)
                      { value = theNewValue;  }
  public void setNext (ListNode theNewNext)
                      {  next = theNewNext; }
}
```

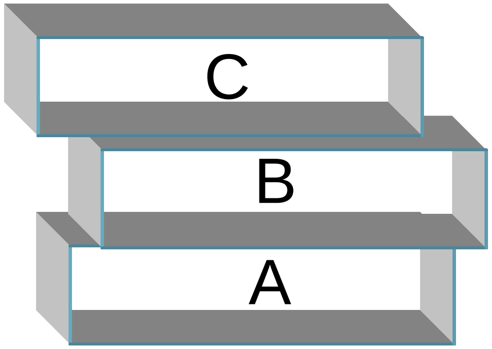
*From *AP CS AB: Implementation Classes and Interfaces*

ListNode Example: Traversal

```
public class SomeClass
{
   private ListNode head = null;
   ...
   public displayList ()
   {
      ListNode node;
      for (node = head; node != null;
                              node = node.getNext () )
      {
         System.out.println ( node.getValue () );
      }
   }
   ...
```

Stack

- Implements LIFO (last-in-first-out) access method
- Provides push and pop methods: push adds an item on top, pop removes and returns the item from the top of the stack



Stack Applications

- Processing nested structures (directories within directories, GUI components within GUI components, etc.)
- Implementing branching processes (tracing a path in a graph)
- “Back” / “Forward” buttons in a browser

Stack Interface

```
public interface Stack*
{
   boolean isEmpty ();
   void push (Object obj);
   Object pop ();
   Object peekTop ();         //  Returns the top element
                              //     but leaves it on the stack
}
```

*From *AP CS AB: Implementation Classes and Interfaces*

Stack Implementation:

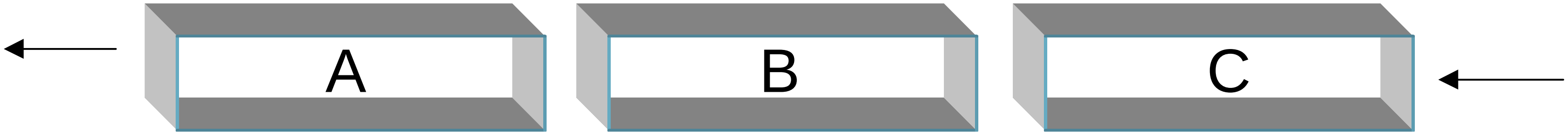
(Based on java.util.ArrayList)

```
public class ArrayStack* implements Stack
{
   private ArrayList items;

   // Constructor:
   public ArrayStack ()  { items = new ArrayList(); }

   public boolean isEmpty ()  { return items.size() == 0; }
   public void push (Object obj)  {  items.add(obj); }
   public Object pop ()  { return items.remove(items.size() - 1); }
   public Object peekTop ()  { return items.get(items.size() - 1); }
}
```

*Adapted from *AP CS AB: Implementation Classes and Interfaces*

Queue

- Implements FIFO (first-in-first-out) access method
- Provides enqueue and dequeue methods: enqueue adds an item at the rear, dequeue removes the item from the front of the queue



Queue Applications

- Simulation of real-time events
- Operating system tasks
  - Printer queue
  - Keyboard buffer
- e-mail mailbox

Queue Interface

```
public interface Queue*
{
   boolean isEmpty ();
   void enqueue (Object obj);
   Object dequeue ();
   Object peekFront ();        //  Returns the front element
                               //     but leaves it in the queue
}
```

*From *AP CS AB: Implementation Classes and Interfaces*

Queue Implementation:

(Based on java.util.LinkedList)

```
public class ListQueue* implements Queue
{
   private LinkedList items;

   // Constructor:
   public ListQueue ()  { items = new LinkedList(); }

   public boolean isEmpty ()  { return items.size() == 0; }
   public void enqueue (Object obj)  {  items.addLast (obj); }
   public Object dequeue ()  { return items.removeFirst(); }
   public Object peekFront ()  { return items.getFirst(); }
}
```

*Adapted from *AP CS AB: Implementation Classes and Interfaces*

Set

- Implements a set of objects
- Cannot hold duplicate objects (neither the same object twice nor “equal” objects)
- Provides methods to add an object, to find out whether an object is in the set, and to remove a given object

Set Applications

- A set of logged-in users
- A Scrabble^{TM} dictionary
- A set of prime numbers

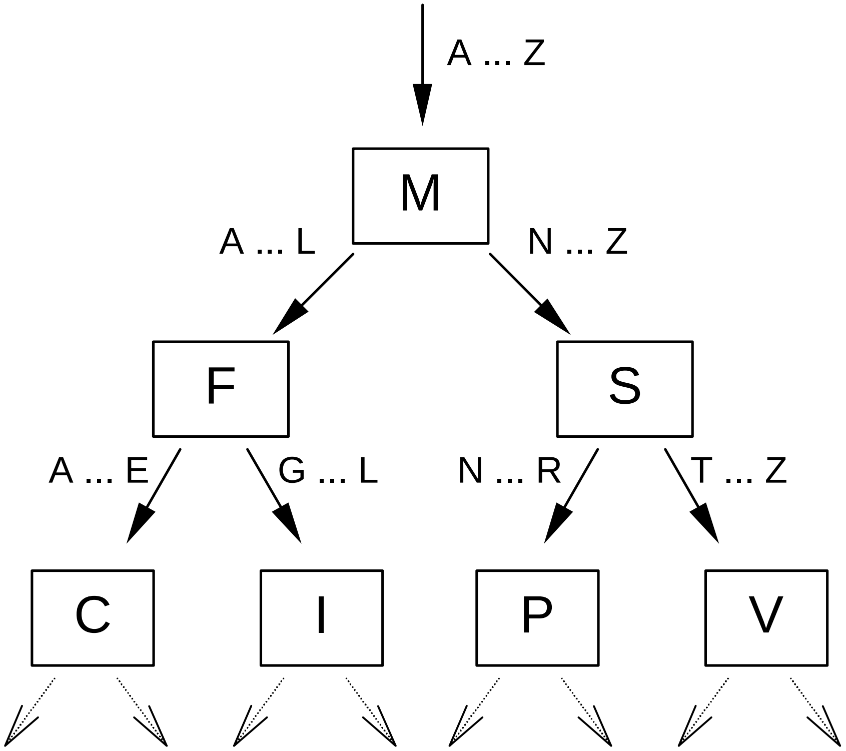
java.util.Set Interface

(AP subset methods)

```
int size ();                  // Returns the number
                              //    of objects in the set
boolean add (obj);            // Adds obj to the set;
                              //    returns true if success
boolean contains (obj);       // Returns true if obj is in
                              //     the set
boolean remove (obj);         // Removes obj from the set;
                              //    returns true if success
Iterator iterator ();         // Returns an iterator
                              //   (the sequence depends
                              //    on implementation)
```

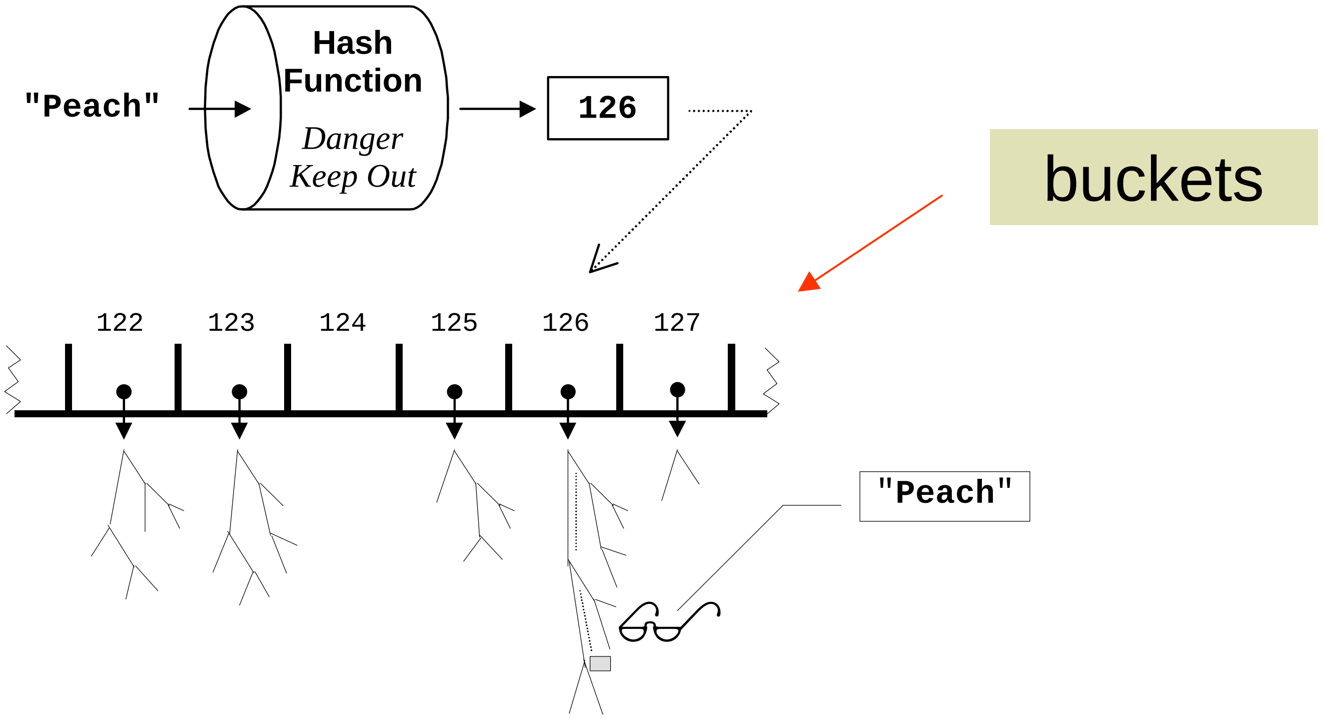
Set Implementation 1: java.util.TreeSet

- Implements Set as a Binary Search Tree



Set Implementation 2: java.util.HashSet

- Implements Set as a hash table



Comparing Objects

- TreeSet: uses compareTo method for Comparable objects (or a Comparator object and its compare method)
- HashSet: uses hashCode + equals methods
- For a well-behaved class, the designer provides all three (hashCode, compareTo, equals) that consistent with each other

Map

- Implements a set of keys; each key is associated with a value
- Cannot hold duplicate keys
- Provides methods to add a (key, value) pair, to find out whether a key is in the set of keys, and to retrieve a value for a given key

Map Applications

- Login info + Subscriber
- Word + Definition
- ID + Student
- etc.

Maps are more flexible and useful than sets

java.util.Map Interface

(AP subset methods)

```
int size ();                   // Returns the number
                               //    of pairs in the map
Objct put (key, value);        // Adds the pair to map
                               //    returns old value or null
Object get (key);              // Returns the value for key
                               //     or null
boolean containsKey (key);
                               // Returns true if key is in
                               //    the set
Set keySet ();                 // Returns the set of all keys
```

Map Implementations:

- java.util.TreeMap — A Binary Search Tree

  (based on the order of keys)
- java.util.HashMap — A hash table (based on hashcodes for keys)

The TreeNode class

- Provided by the College Board for “do-it-yourself” implementations of binary trees
- Is likely to come up in AB free-response questions
- Similar to ListNode, but has getLeft, getRight, setLeft, and setRight methods instead of getNext and setNext

TreeNode Examples:

```
...
public void traverseInOrder (TreeNode root)
{
   if ( root != null )
   {
       traverseInOrder ( root.getLeft () );
       System.out.println ( root.getValue () );
       traverseInOrder ( root.getRight () );
   }
}

public TreeNode copy (TreeNode root)
{
   if ( root == null )
      return null;
   return new TreeNode (root.getValue (),
                copy ( root.getLeft () ), copy ( root.getRight () ) );
}
```

Priority Queue

- Holds items that are ranked in some way according to their “priority”; the items are Comparable objects (or a Comparator is provided)
- Provides methods to add an item and to remove the minimum (highest priority) item

Priority Queue Applications

- Handling prioritized events
- Processing of auction bids, trading orders, etc.

PriorityQueue Interface

```
public interface PriorityQueue*
{
   boolean isEmpty ();
   void add (Object obj);
   Object removeMin ();
   Object peekMin ();          //  Returns the min element
                               //     but leaves it in the queue
}
```

*From *AP CS AB: Implementation Classes and Interfaces*

Priority Queue Implementations

- Simplistic implementations can use an ArrayList or a LinkedList
- A more efficient implementation is based on *heaps*
- A heap is a complete binary tree, stored in an array; the smallest item is in the root; the same property holds for each subtree

Summary: What We Need To Know for the AP (AB) Exam

- Abstract data collections:
  - List, Stack, Queue, Set, Map, Priority Queue

  and their applications

- Interfaces (AP subset)
  - java.util.List, Stack, Queue, java.util.Set, java.util.Map, PriorityQueue

What We Need To Know (cont'd)

- Java library classes
  - java.util.TreeSet and java.util.TreeMap,
  - java.util.HashSet and java.util.HashMap

  (their methods are the same as in the interfaces they implement)
- Comparable objects, hashCode, equals
- Working with linked lists using ListNode
- Working with binary trees using TreeNode

These slides are posted at:

`http://www.skylit.com/oop/`

e-mail questions about the above to me:

`mlitvin@andover.edu`

Other Java resources are available at:

`http://www.skylit.com/javamethods.html`

For *Java Methods* evaluation copies e-mail:

support@skylit.com

Java Methods AB evaluations are online only (see the preface, table of contents and several chapters online):

http://www.skylit.com/jmethods.html