String is a Class; Quoted Text is an Object

- String s = "abc d";
- Indexed characters:
  012345
  abc d
  s.length() is 5
  s.charAt(2) is 'c'
  s.substring(2) is "c d"
  s.substring(1,3) is "bc"

String Has a Lot of Useful Methods

- String s = "abc d";
- Indexed characters:
  012345
  abc d
  s.substring(2,4) is "c " (NOT "c d")
  s.substring(2) is "c d"
  "  bcd   ".trim() is "bcd" (trim beginning and ending blanks)
  s.indexOf("bc") is 1

String Variables Hold Folder Names

- Create two Strings
  - String s = "hello";
  - String t = "hello";
- Do not use == to test equality of s and t
  - s.equals() tests if same object
  - Not useful for Strings
- Use equals() instead
  - s.equals() tests if they have the same text

Containers

- Container: an object that holds a list of objects
  - But cannot hold primitive values (e.g. int, double, etc.)!
- Java has several container classes
  - The are all in package java.util!
  - Generic classes: type depends on what is contained
  - Put contained type in < >
- Example: Vector
  - Vector<String>: Vector that holds String objects
  - Vector<AcornProfile>: Holds AcornProfile objects
  - Vector<Vector<String>>: ???
  - Vector<int>: NOT ALLOWED!

Wrappers: Turn Primitives into Objects

- Want Vector<int>
  - int is primitive type, not class
  - Need to convert an int value (e.g. 9) into an object
- Integer: a wrapper class
  - Contains or wraps one value
  - Value cannot be changed: it is immutable
  - Many useful static features
    - Integer.MAX_VALUE
    - Integer.parseInt(String)

Each Primitive Type Has a Wrapper

- When you need to treat a primitive value as an object, then just wrap the value in an object of the wrapper class.

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

Each wrapper class has:

- Instance methods (e.g. equals, constructors, toString)
- Static variables and methods (for useful computations)

Integer k = new Integer(63);  int j = k.intValue();

You don't have to memorize the methods of the wrapper classes. But be aware of them, See Section 5.1 and PLive 5.1 and 5.2 for more.
Example: Vector

- Create an empty vector instance (of Strings)
  import java.util;
  Vector vec = new Vector<String>();
- Add some strings to it
  vec.add("abc"); // Adds 2 at position 0
  vec.add("ghi"); // Adds 7 at position 1
  vec.add("jkl"); // Adds -3 at position 2
- Get the String at position 1
  vec.get(1); // Function call, gives 7
- Search vector for number 5
  vec.indexOf(new Integer(5)); // Not found; gives -1

Vectors Can Add and Remove

- Do the following:
  import java.util;
  Vector vec = new Vector<String>();
  vec.add("abc");
  vec.add("def");
  vec.add("ghi");
  vec.add("jkl");
  vec.remove(1);
- After all this, what is the value of vec.get(2)?
  A: Function gives "abc"
  B: Function gives "ghi"
  C: Function gives "jkl"
  D: I have no clue

Arrays

- Array: an object that holds a fixed number of values of the same type
  - Type of an array is written:
    - <type>[], (e.g. int[])
  - Declare a variable x that holds the name of an array of ints:
    - <type> <name>; (e.g., int x)
  - Elements of array x are numbered:
    - 0, 1, 2, …, n - 1
  - To refer to an element of an array:
    - <name>[<index>]; (e.g. x[3])

Arrays vs. Vectors vs. Strings

- Declaration:
  - int[] x;
  - (contains int[])
- Creation:
  - a = new int[3];
  - (size fixed forever)
- Reference:
  - x = a[0];
- Change:
  - a[0] = x;

Variables a[0], a[1], … are at successive locations in memory. Element type can be class or primitive type.

Arrays

- Array length is a field of the object
  - x.length
  - x.length()  [not x.length(i)]
- The length field is final: it never changes after the array is created
  - Length is not part of the array type
  - An int[] variable can hold arrays of different lengths at different times
- Declaring x does not create array
  - As an object it starts out null
  - Need a special new-expression:
    - new <type>[<lengths>]
    - (e.g. x = new int[5])
    - Length is not part of the array type
  - As an object it starts out null
  - Need a special new-expression:
    - new <type>[<lengths>]
    - (e.g. x = new int[5])

Array Initializers

- Initializing a newly created array:
  - int[] arr = new int[5];
  - (create array of 5 ints initialized with default (0))
  - (assign new values to elements)
- Instead, use an array initializer:
  - new int[] { [5, 4, 7, 6, 5] };
  - (create array of 5 ints and initialize all elements)
  - Types must agree with array's type
  - arr = new int[] { [5, 4, 7, 6, 5] };

In a declaration, short form is available:

- int[] arr;
  - arr = new int[] { [5, 4, 7, 6, 5] };
  - arr = new int[] { [5, 4, 7, 6, 5] };
  - arr = [ [5, 4, 7, 6, 5] ];

The last two do the same thing.