CS1004: Intro to CS in Java, Spring 2005

Lecture #11: Java OO

Janak J Parekh
janak@cs.columbia.edu

Administrivia

- HW#2 due now
- HW#3 out this afternoon
- I didn't cover the theory book's 5.3 in detail, I'll leave that as read-only
- Nifty trick on emacs: how to set Java-specific coloring
  - http://home.janak.net/cs1004/phpBB2/viewtopic.php?p=314

Let's try this again

- I swear, I have an example that works. Let's give it a try
- It's easy to make a check to see if ints are equal
- Not so easy for Strings, or for any other *object*

Here's why

- A class name can be used as a type to declare an *object reference variable*

```
String title;
```

- An *object reference variable* holds the address of an object
- The object itself must be created separately
  - Either via a *constructor* or via another entity that creates it for you
  - *Instantiation* == creation
  - Multiple ways to create Strings in particular

Invoking Methods

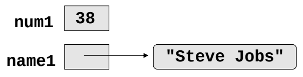
- We've seen that once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
count = title.length();
```

- A method may *return a value*, which can be used in an assignment or expression
- A method invocation can be thought of as asking an object to perform a service
- Primitive types *have no methods*

References

- Note that a primitive variable contains the value itself, but an object variable contains the *address* of the object
  - Memory location of the object, to be precise
- An object reference can be thought of as a "pointer" to the location of the object
- Rather than dealing with arbitrary addresses, we often depict a reference graphically

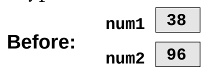


Why bother?

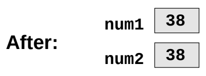
- Often, with objects, we don't know how much memory we're going to use up until we actually create it
- As a result, it goes into a *separate part* of the memory (known as the *heap*) when it's instantiated
- Primitive types and the reference itself are stored, on the other hand, on the *stack*
- Don't worry about these terms right now

Assignment Revisited

- The act of assignment takes a *copy* of a value and stores it in a variable
- For primitive types:

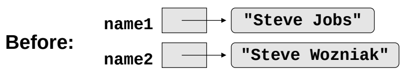


```
num2 = num1;
```

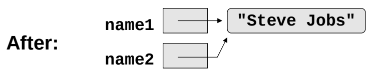


Reference Assignment

- For object references, assignment copies the *address*:

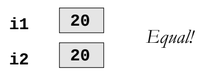


```
name2 = name1;
```

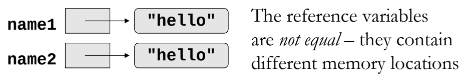


So in my examples…

- Here's what happened with integers:



- And with Strings:


name1
"hello"
name2
"hello"
The reference variables are *not equal* – they contain different memory locations

Aliases

- Two or more references that refer to the same object are called *aliases* of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object
- How do we fix our program?

Null

- We can tell Java to explicitly set a reference to "nothing"
- In fact, it does it by default sometimes
- Useful if you have a variable which you *will* eventually fill in, but don't know what to put in yet
  - **`String test = null;`**
  - **`test = new String("Now we have data");`**
- You can test for equality with null, too
- What happens if we did the opposite? Let's draw it out
  - **`String test = new String("Data!");`**
  - **`test = null;`**

Garbage Collection

- When an object no longer has any valid references to it, it can no longer be accessed by the program
- The object is useless, and therefore is called *garbage*
- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use
- In some other languages, the programmer is responsible for performing garbage collection

The String Class

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

```
title = "Java Software Solutions";
```

- This is special syntax that works *only* for strings
- Each string literal (enclosed in double quotes) represents a `String` object

String Methods

- Once a `String` object has been created, neither its value nor its length can be changed
- Thus we say that an object of the `String` class is *immutable*
- However, several methods of the `String` class return new `String` objects that are modified versions of the original
  - Concatenation also does this
- See the book or the String Java documentation

String Indexes

- It is occasionally helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index*
- The indexes begin at zero in each string
- In the string `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4
- What happens if we supply too large or too small an index?

Other classes and class libraries

- A *class library* is a collection of classes that we can use when developing programs
- The *Java standard class library* is part of any Java development environment
  - Not "Java language" per se, but closely associated
  - Various classes we've already used (`System` , `Scanner`, `String`) are part of the Java standard class library
  - Other class libraries can be obtained through third party vendors, or you can create them yourself

Packages

- The classes of the Java standard class library are organized into *packages*
- Some examples:

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

The import Declaration

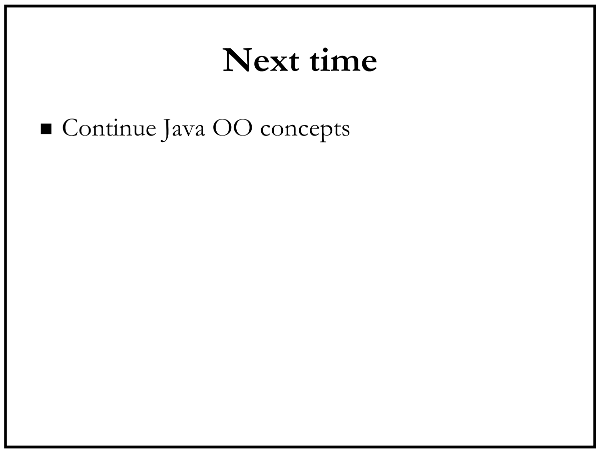
- When you want to use a class from a package, you could use its *fully qualified name*:
```
    java.util.Scanner scan = new
    java.util.Scanner(System.in);
```
- Or you can *import* the class at the top:
```
      import java.util.Scanner;
```
- To import all classes in a particular package, you can use the * *wildcard character*:
```
          import java.util.*;
```

java.lang package is special

- All classes of the `java.lang` package are imported automatically into all programs, as if we had typed `import java.lang.*;`
- That's why we don't import the `System` or `String` classes explicitly
- `Scanner`, on the other hand, is part of `java.util`

The Random Class

- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A `Random` object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values
- Let's try a quick example (more complex one on page 126)



Next time

- Continue Java OO concepts