Introduction to Java (part 1)

Jules White

Bradley Dept. of Electrical and Computer Engineering
Virginia Tech
Jules.white@vt.edu

http://www.ece.vt.edu/faculty/white.php

Virginia Tech

Java Resources

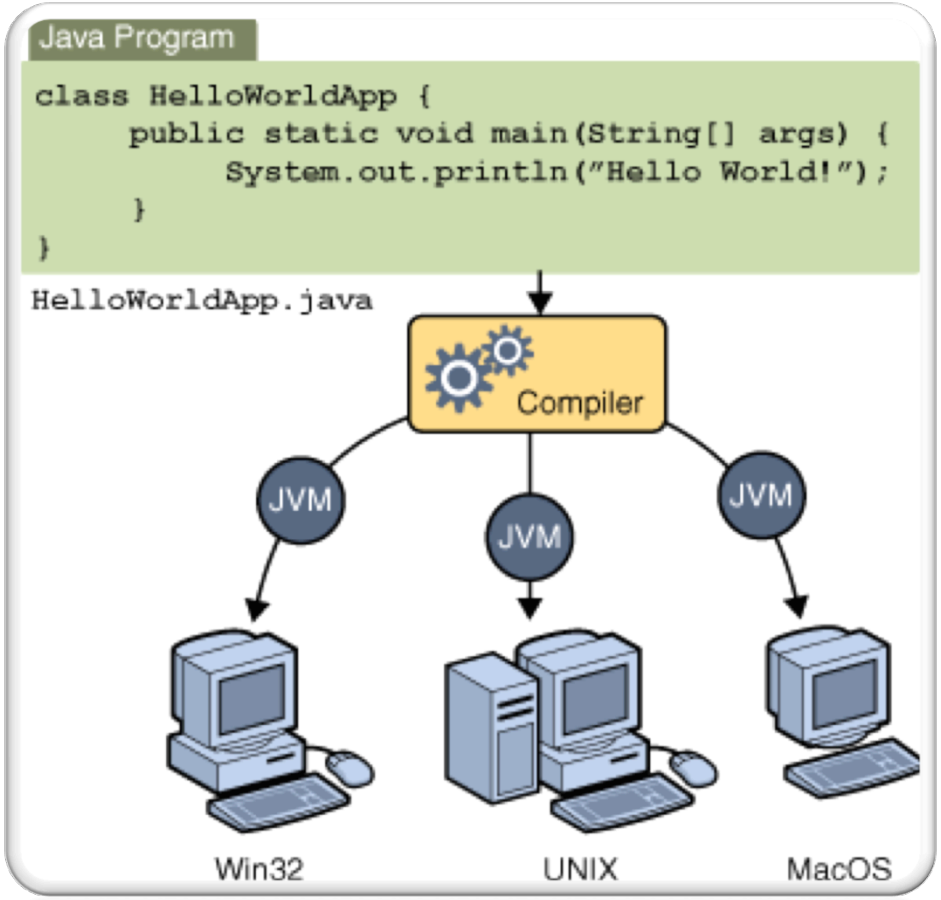
- Possibly the best online resource for Java is:
  - The Java Tutorial http://download.oracle.com/javase/tutorial/
- Many of the figures/terms in this lecture come from the Java tutorial
- If you need documentation on the standard APIs, use the Javadocs:
  - http://download.oracle.com/javase/6/docs/api/

Why Use Java for Network Applications?

- Java is an object-oriented programming language (like C++)
- Java code is compiled into bytecode (not machine code like C++) and interpreted by the Java Virtual Machine (JVM)
  - Each Java source code file (Foo.java) is compiled into a bytecode file (Foo.class)
- Java's cross-platform nature is critical to its success as a platform for writing networked applications
- Writing cross-platform networking code in C++/C is HARD
  - Major platform variations make developing and maintaining a cross-platform network application extremely difficult

The Java Virtual Machine

- Each Java source code file (Foo.java) is compiled into a bytecode file (Foo.class)
- The Java Virtual Machine (JVM) interprets the bytecode and dynamically translates it into platform-specific instructions
- Once a Java class has been compiled it can be run on any platform that has a JVM
- The JVM and the standard APIs serve as a common interface to the underlying platform



Managing Memory in Network Applications

- A major concern in network application is memory management
  - Figuring out when to release memory can be challenging when some objects may be referenced by remote clients
- Java has garbage collection, which means you don't have to manage memory
- Garbage collection makes writing most network applications much easier

Java on Android

- The Java language is the same on Android but the JVM works a bit differently
- As a developer, you probably will not be aware of the differences
- The key difference is that Java on Android uses the Dalvik Virtual Machine
- Dalvik allows Google to avoid paying royalties to Sun/Oracle
    - Although Sun/Oracle is now suing Google for big $$$$$
- Dalvik is supposedly better for mobile devices b/c it consumes less power

Creating a Java Class

- Java is an object-oriented language
- If you are not familiar with OO, you should seriously consider whether or not you are prepared for this class!
- Each ***top-level*** Java class must be declared in its own “.java” file.
- ***There is not a separate header file – everything is declared in one place***
- The class Foo would be defined in the file “Foo.java”
- Comments start with “//”
- Here is the class declaration for Foo:

```
//this class would live in the file Foo.java

public class Foo {
   //I am a class
}
```

Java Class Anatomy

```
//this class would live in the file Foo.java

package org.vt.ece4564;

public class Foo {

}
```

Every class should live in some package

This is the basic syntax of a class declaration. Everything within these brackets is part of the class' definition

Importing Classes

- You can refer to any class that is in the same package by its name
- If the class that you are referring to, is not in the same package, you must either:
  - Use the fully qualified name of that class (poor form)
  - Add an import statement to import that class
- Import statements follow the package statement in a class and specify the fully qualified name of a class that you would like to use
- Examples of using the class org.vt.something.Bar with/without an import

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {}
```

```
package org.vt;
public class Foo extends org.vt.something.Bar {
   //this example uses the fully qualified name rather than an import
}
```

Java Class Anatomy

```
//this class would live in the file Foo.java

package org.vt.ece4564;


import java.util.List;
import java.util.ArrayList;

public class Foo {


}
```

Imports, follow the package declaration. Imports allow you to use classes in other packages

Java Class Anatomy

```
//this class would live in the file Foo.java

package org.vt.ece4564;


import java.util.List;
import java.util.ArrayList;

public class Foo {
     private List myList_ = new ArrayList();

     //these two are equivalent
     private List myOtherList_;
     private List anotherList_ = null;

}
```

Member variables are declared right after the opening class declaration. You can either initialize the variable to a value or leave it uninitialized (null).

Creating a Java Package

- A Java package is a way of grouping related Java classes (similar to a C++ namespace)
- You will always use packages
- For example, “com.android” is a package
- You can name packages anything you want and use any arbitrary organization scheme for grouping classes into packages
- The fully qualified name of a class is <package_name>.<class_name> (e.g. org.vt.Foo)
- A package will have a corresponding file system folder hierarchy (e.g. the classes in org.vt would be contained in the folder “org/vt/”
- A class is added to a package by including at package declaration at the very top of the file:

```
//this class would be defined in the file org/vt/Foo.java
package org.vt;

public class Foo {
   //I am a class
}
```

Exercise

Create 2 Java classes in different packages with at least 2 constructors, 3 member variables, and 1 import statement each.

Post your answer on the class web page's wiki:

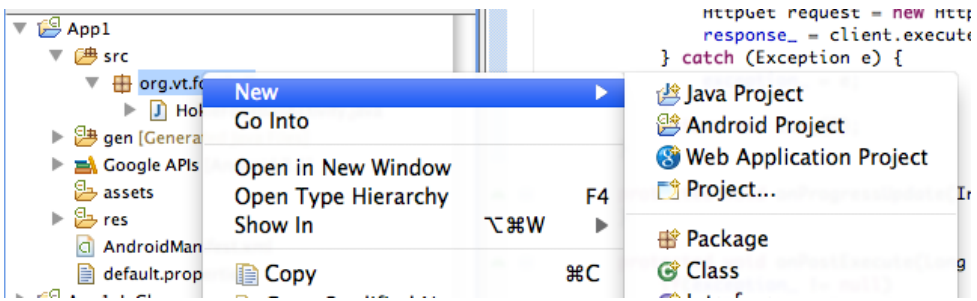
1. Name your wiki page: <your_name>JavaEx1 (e.g. JulesWhiteJavaEx1)
2. Enclose your code in a verbatim block:

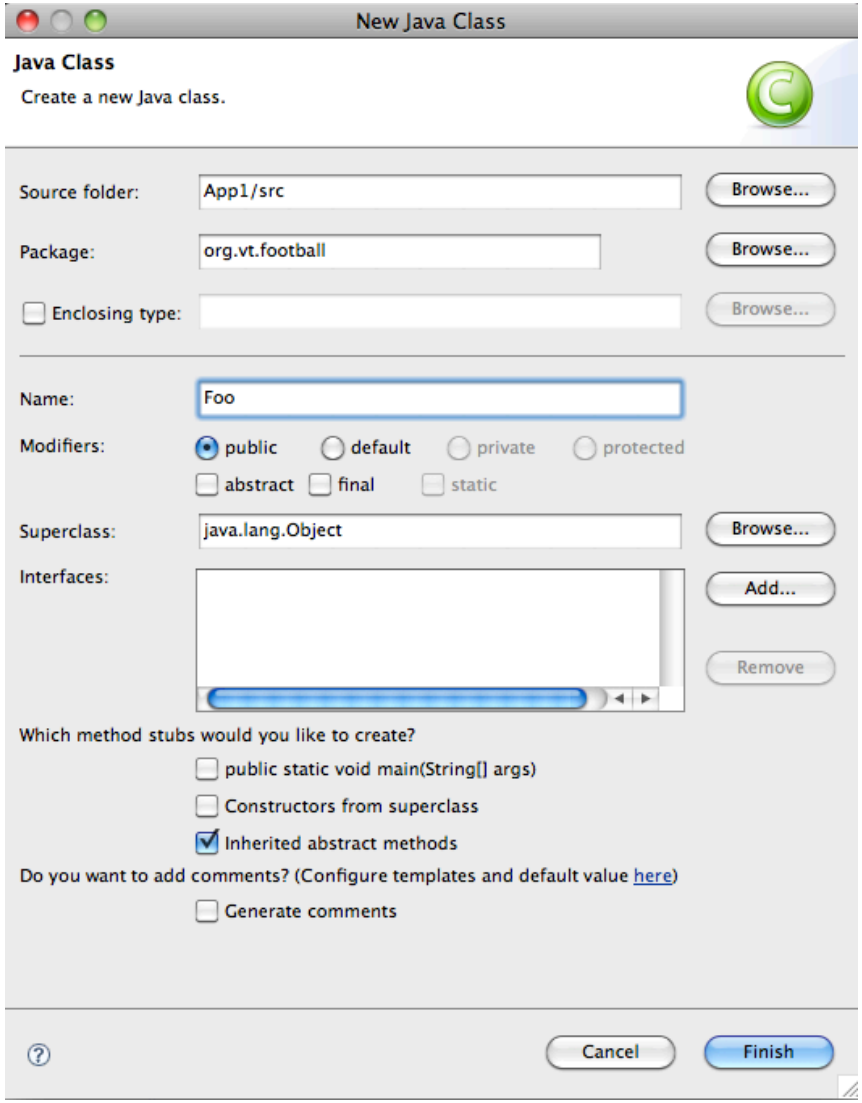
```
{{{
    //your code here
}}}
```

3. Do not look at anyone else's answer
4. Your grade will be based on completion of the exercise (e.g. 100 or zero – all or nothing)

How to Create a Class in Eclipse

- Select a “src” folder in a project
- Right-click, New->Class





Inheritance

- A Java class can inherit from another class using the “extends” key word
- All classes inherit from java.lang.Object. If you do not explicitly specify a super class, your class will inherit from java.lang.Object
  - This means that every class has java.lang.Object at the root of its inheritance hierarchy
- In this example, we make Foo extend the class Bar:

```
package org.vt;

public class Foo extends Bar {
   //I am a class
}
```

Creating Objects

- You create new instances of a class using the “new” keyword:

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {

    public void createAFoo() {
       Foo myfoo = new Foo(); //create an instance of Foo

       List l2 = new ArrayList(4);
       Foo anotherfoo = new Foo(new ArrayList(),l2);
       //create another instance
    }

}
```

Member Variables

- Member variables are declared inside of a class declaration and should typically come before all method declarations
- A member variable can be private, public, or protected
- A private member variable can only be accessed within the class
- A public member variable can be accessed by anyone
- 99% of your variables will be private
- You can set a value for the variable when you declare it or do it later
- This class will use a naming scheme where all member variable names end in an underscore (e.g. foo_)

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {
    private String someData_ = “foobar”;  //set the value
    private String someOtherData_; //set the value later
}
```

Setting Member Variables

- From within a class, you can refer directly to a member variable by name
- If the member variable is in another object instance, you use dot notation:

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {
    private String someData_ = “foobar”;  //set the value
    private String someOtherData_; //set the value later

    public void doSomething(){
        someData_ = “some new value”;   //set a member variable in
                                        //this instance
        Foo myfoo = new Foo();
        myfoo.someData_ = “some value”; //set a member variable in
                                        //another instance
    }

}
```

The “this” Member Variable

- Every object can refer to itself using the built-in “this” member variable
- “this” refers to the current object

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {
    private String someData_ = “foobar”;  //set the value
    private String someOtherData_; //set the value later

    public void doSomething(){
        someData_ = “some new value”;   //set a member variable in
                                        //this instance

        this.someData_ = “some new value”; //equivalent
    }

}
```

Methods

- Every class can define one or more methods
- Methods can be public, private, or protected
- Examples:

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {
    private String someData_ = “foobar”;  //set the value

    public void someMethod(){
       //a method that takes no parameters and does not return anything
    }

    public void anotherMethod(String arg, Foo myfoo){
       myfoo.someData_ = arg;
    }

    public String aMethodToGetSomeData(){
       return myfoo.someData_;
    }
}
```

Invoking a Method

- A method can be invoked on any object instance
- Examples:

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {
    private String someData_ = “foobar”;  //set the value

    public void someMethod(){
       Foo somefoo = new Foo();
       String val = somefoo.aMethodToGetSomeData();
    }

    public void anotherMethod(String arg, Foo myfoo){
       myfoo.someMethod();
    }

    public String aMethodToGetSomeData(){
       return myfoo.someData_;
    }
}
```

Constructors

- A constructor is a method that gets called when an instance of your class is created with “new”
- A constructor has the same name as the enclosing class and does not specify a return type:

```
package org.vt;
import org.vt.something.Bar;

public class Foo extends Bar {
    private String someData_ = “foobar”;  //set the value

    public Foo(){
       //a no argument constructor
    }

    public Foo(String someval){
       someData_ = someval;
    }

    public Foo(String a, String b){
       this(a); //invoke another constructor first
    }
}
```

Using Constructors

- When you use the “new” keyword, you can specify a constructor to call by passing different arguments:

```
Foo myfoo = new Foo(); //if no constructors are specified in a class
                       //there will always be this default constructor

Foo somefoo = new Foo(“abcdef”); //call the constructor that takes 1
                                 //String argument

Foo someotherfoo = new Foo(“aasdf”,”asdfe”);
```

Primitive Types

- String is a special class that is also an object
- int, double, byte, boolean, long, …what you would expect
- Integer, Double, Boolean, Long are the object equivalents of the primitive types.

```
int i = 0;
double j = 0.0;
boolean ok = true;
byte val = 1;

int is = Integer.parseInt("1"); //convert a string to an int
int ds = Double.parseDouble("1.0");

//automatically convert between primitive and object representation
int j = 0;
Integer jo = j;
j = jo;
```

Arrays

- An array is declared with “[]” and a length
- Examples:

```
Foo[] myfoos = new Foo[10]; //create an array to hold 10 Foo objects
myfoos[0] = new Foo(); //create a Foo and stick it in the array at index
                       //0
Foo somefoo = myfoos[0]; //get a Foo instance from the array

if(myfoos[1] == null){
  //log something to the console for a normal Java app (not android)
  System.out.println(“arrays of objects do not initialize the indexes”);
}

int[] someints = new int[5]; //you can create arrays of primitives
```

Arrays

- An array is declared with “[]” and a length
- Examples:

```
Foo[] myfoos = new Foo[10]; //create an array to hold 10 Foo objects
myfoos[0] = new Foo(); //create a Foo and stick it in the array at index
                       //0
Foo somefoo = myfoos[0]; //get a Foo instance from the array

if(myfoos[1] == null){
  //log something to the console for a normal Java app (not android)
  System.out.println(“arrays of objects do not initialize the indexes”);
}

int[] someints = new int[5]; //you can create arrays of primitives
```

Printing Output to the Console

- You can print to the console with System.out.println
- In Android, you can log to logcat with Log.d
- Examples:

```
//In normal Java
System.out.println("some message for debugging");

//In Android
Log.d("My Class","this will be printed to log cat");
```

Inheritance

- A Java class can inherit from another class using the “extends” key word
- A derived class can override methods from a super class (e.g. all methods are the equivalent of virtual methods in C++)

```
package org.vt;

public class Foo extends Bar {
   public void do(){}
}
```

```
package org.vt;

public class FooBar extends Foo {
   public void do(){} //this will get called instead of Foo’s version
}
```

Inheritance

```
package org.vt;

public class Foo extends Bar {
   public void do(){System.out.println("a");}
}
```

```
package org.vt;

public class FooBar extends Foo {
   public void do(){System.out.println("b");}
}
```

```
Foo f = new Foo();
Foo f2 = new FooBar();
FooBar f3 = new FooBar();
f.do(); //this will print "a"
f2.do(); //this will print "b"
f3.do(); //this will print "b"
```

Exercise

Modify your original exercise answer to include a new base class for the other two classes. Include a method that constructs an instance of java.util.ArrayList, invokes its add method, and passes it a single String argument.

Post your modifications in the wiki page you created

The “super” Member Variable

- Every object can refer to its parent type using “super”

```
package org.vt;

public class Foo extends Bar {
   public void do(){System.out.println(“a”);}
}
```

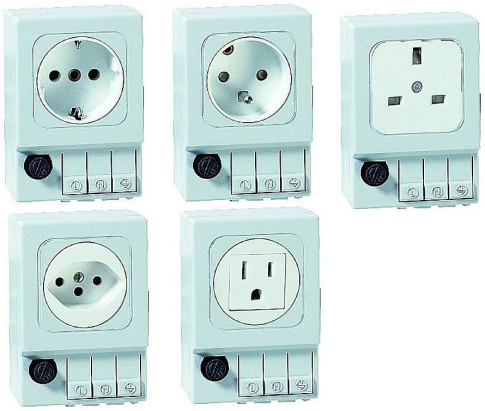
```
package org.vt;
public class FooBar extends Foo {
   public void do(){
    super.do();
   }
}
```

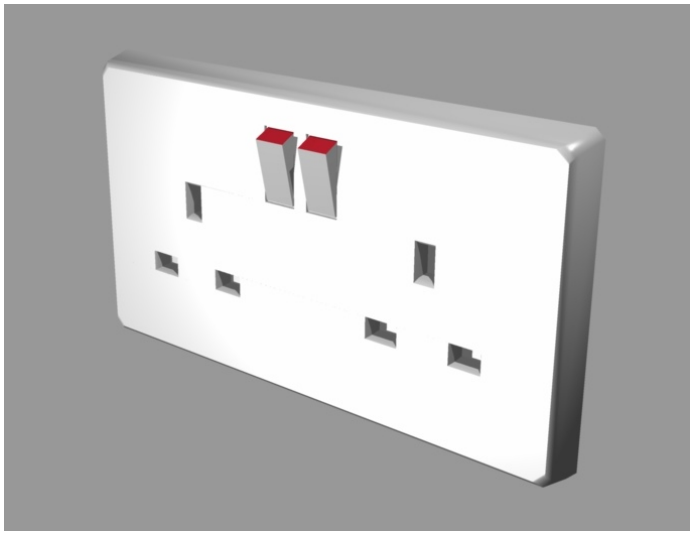
```
Foo f = new Foo();
Foo f2 = new FooBar();
FooBar f3 = new FooBar();
f.do(); //this will print “a”
f2.do(); //this will print “a”
f3.do(); //this will print “a”
```

Frameworks & Callbacks

- A framework is a semi-complete application that your code plugs into
- Frameworks exhibit inversion of control, they control the main thread of execution and decide when your code should execute
  - e.g. If you plug a listener for button clicks into Android's GUI framework, the framework decides when to call your listener
  - e.g. Android decides when to call your app's onStart, onCreate, etc.

- How do you define the shape/size of the receptacles you plug into?





Java Interfaces

- A Java interface is a contract that specifies methods that an implementer of the interface must have
- A Java interface is in its own file exactly like a class
- An example interface definition:

```
package org.vt;

public interface Runnable {
   public void run(); //method signature with no body
}
```

```
package org.vt;

public class RunnableImpl implements Runnable {
   public void run(){
      System.out.println("this is the run implementation");
   }
}
```

Java Interfaces

```
package org.vt;

public class RunnableImpl extends Foo implements Runnable {
   public void run(){
      System.out.println("this is the run implementation");
   }
}
```

```
package org.vt;

public class AnotherRunnableImpl implements Runnable {
   public void run(){
      System.out.println("this is another run implementation");
   }
}
```

```
Runnable r = new RunnableImpl();
r.run(); //prints "this is the run implementation"
r = new AnotherRunnableImpl();
r.run(); //prints "this is another run implementation"
```

If, ==, and .equals() in Java

- java.lang.Object defines the .equals() method
- Every object has this method
- If you use “==“ to compare for equality of two objects, it returns true if the two objects have the same memory address
- If you use .equals() to compare for equality, the implementation can be overriden to do other things, like check for equal values

```
String a = “foo”;
String b = “foo”;
String c = “bar”;

if(a == b){
 //this may or may not get executed
 //never compare Strings for equality using “==“
}
if(a.equals(b)){
 //this will definitely execute because the class java.lang.String
 //overrides the equals() method to check the internal values of the
 //Strings
}
```

for loops in Java

- Java has the standard for loop that you see in C++

```
String[] myStrings = new String[]{“a”,”b”,”c”};
for(int i = 0; i < myStrings.length; i++){
   System.out.println(myStrings[i]);
}
```

- Java also has an enhanced for loop for iterating over things that implement the Iterable interface

```
String[] myStrings = new String[]{“a”,”b”,”c”};
for(String somestring : myStrings){
   System.out.println(somestring);
}
```

Exercise

Create a class that implements the java.lang.Runnable interface. The implementation should create an array of three strings and iterate over the array to print them out. You should always refer to the Javadocs: http://download.oracle.com/javase/6/docs/api/ if you need more information about the standard Java APIs (e.g. Runnable)

Post your modifications in a separate verbatim code block on the <your_name>JavaEx1 wiki page that you created

ArrayLists

- An ArrayList is a variable sized list of items similar to an array

```
import java.util.ArrayList; //add this import in the appropriate place
…
ArrayList mylist = new ArrayList();
mylist.add("foo");
mylist.add("bar");
mylist.add("foobar");

//This list stores objects of type java.lang.Object, so we have to
//cast the item back to a String
String itemone = (String)mylist.get(0);
mylist.remove(0);

//Casting example
String a = "asdf";
Object b = a;

//illegal b/c b is of type java.lang.Object
a = b;

//legal, b/c we are explicitly casting b to a String
a = (String)b;
```

ArrayList Item Types

- ArrayLists can be typed to include specific types of items
- The < > defines the type
  - E.g. ArrayList<String> is a list of Strings
  - ArrayList<Foo> is a list of objects of class Foo

```
import java.util.ArrayList; //add this import in the appropriate place
…
ArrayList<String> mylist = new ArrayList<String>();
mylist.add("foo");
mylist.add("bar");
mylist.add("foobar");

//throws an exception b/c this list holds strings
mylist.add(new Foo());

//no need to cast the value b/c the list has a type
String value = mylist.get(0);
```

Iterating Over List Items

- ArrayLists can iterated over in for loops

```
import java.util.ArrayList; //add this import in the appropriate place
…
List<String> mylist = new ArrayList<String>();

for(String item : mylist){
      System.out.println(item);
}

//equivalent to
for(int i = 0; i < mylist.size(); i++){
   String item = mylist.get(i);
   System.out.println(item);
}
```

HashMaps

- A HashMap stores key/value pairs

```
import java.util.HashMap; //add this import in the appropriate place
…
HashMap<String,Object> mymap = new HashMap<String,Object>();

Foo f1 = new Foo();
Foo f2 = new Foo();
Foo f3 = new Foo();
mymap.put(“one”,f1);
mymap.put(“two”,f1);
mymap.put(“three”,f3);
mymap.put(f1,f2);

//what is the output?
System.out.println(mymap.get(“one”) == mymap.get(“two”));

mymap.put(“two”, f3);

//what is the output?
System.out.println(mymap.get(“one”) == mymap.get(“two”));
```

HashMap Key / Value Typing

- A HashMap stores key/value pairs
- A HashMap type is specified by two types
  - One type for the keys
  - One type for the values
- HashMap<String,Foo> would be a map that uses Strings for the keys and objects of class Foo for the values

```
import java.util.HashMap; //add this import in the appropriate place
…
HashMap mymap = new HashMap();

Foo f1 = new Foo();
Foo f2 = new Foo();
Foo f3 = new Foo();
mymap.put("one",f1);
mymap.put("two",f2);
mymap.put("some key",f3);
```

Exercise

Create a class, called ContactManager, for managing contacts on a phone. The class should have three methods:

```
public void addNumber(String person, String number)
public String getNthNumber(String person, int numberindex)
public void printNumbers(String person)
```

Your class should be able to store multiple numbers for each person. The getNthNumber method should return the phone number at index “numberindex” in the list of that person’s numbers.

Post your modifications in a separate verbatim code block on the <your_name>JaveEx1 wiki page that you created

Java Class Anatomy

```
//this class would live in the file Foo.java

package org.vt.ece4564;


import java.util.List;
import java.util.ArrayList;

public class Foo {
     private List myList_ = new ArrayList();

     private List myOtherList_;

     public Foo(){}

     public Foo(List somelist){
        myOtherList_ = somelist;
     }

     public Foo(List ml, List sl){
        myList_ = ml;
        myOtherList_ = sl;
     }

}
```

Three example constructors

Java Class Anatomy

```
//this class would live in the file Foo.java

package org.vt.ece4564;


import java.util.List;
import java.util.ArrayList;

public class Foo {
     …
     //this method can only be called inside Foo
     private void doSomething(){
          //your code here
     }

     public List getMyList(){
          return myList_;
     }

     public void setMyList(List l){
          myList_ = l;
     }

     public void setBothLists(List l, List l2){
          myList_ = l;
          myOtherList_ = l2;
     }
}
```

Example method declarations

Java Class Anatomy

```
//this class would live in the file Foo.java

package org.vt.ece4564;

public class Foo {
   //I am a class

   //a private member variable;
   private String name_;
   private Foo anotherFoo_;
   private Foo someOtherFoo_ = new Foo("some name",false);

   //a basic constructor
   public Foo(){

      //call the superclass constructor
      super();
   }

   //another constructor
   public Foo(String aparam, boolean someval){
      name_ = aparam;
   }

   public Foo getAnotherFoo(){
      return this.anotherFoo_;
   }
}
```

Every class should live in some package

Member variables are declared right after the class declaration

You can call the super class' constructor with "super()" .. you can also pass params to it super(someparam,another)

Member variables are declared right after the class declaration

Java Class Anatomy (cont.)

```
//this class would live in the file Foo.java

package org.vt.ece4564;

public class Foo {
   //I am a class

   //a private member variable;
   private String name_;
   private Foo anotherFoo_;
   private Foo someOtherFoo_ = new Foo("some name",false);

   //a basic constructor
   public Foo(){

      //call the superclass constructor
      super();
   }

   //another constructor
   public Foo(String aparam, boolean someval){
      name_ = aparam;
   }

   public Foo getAnotherFoo(){
      return anotherFoo_;
   }
}
```

Declaring Methods

```
//this class would live in the file Foo.java

package org.vt.ece4564;


import java.util.List;
import java.util.ArrayList;

public class Foo {
     …
     //this method can only be called inside Foo
     private void doSomething(){
          //your code here
     }

     public List getMyList(){
          return myList_;
     }

     public void setMyList(List l){
          myList_ = l;
     }

     public void setBothLists(List l, List l2){
          myList_ = l;
          myOtherList_ = l2;
     }
}
```

Example method declarations