Java Review

EECS 484

Winter 2015

Today's Agenda

- Introductions
- Overview of Java vs C++
- Group exercises

Introduce Yourself

- Please tell us:
  - Your name
  - Year
  - Department

- Thanks!

Simple Java Program

```
/* This is a simple Java program */
public class HelloWorld {
    //Your program begins with a call to main()
   public static void main(String[] args) {
      System.out.println("Hello, World");
   }
}
```

Simple Java Program (Contd.)

- Java has no ``top-level'' or ``global'' variables or functions. A Java program is *always* a set of class definitions.
- By convention, the name of that class should match the name of the file that holds the program.
- **Compiling a Java program:**
  - javac HelloWorld.java
- **Executing a Java program:**
  - java HelloWorld
  - java HelloWorld 10 20 will pass [“10”, “20”] as arguments to the main function

Simple Java Program (Contd.)

- In Java, the first thing executed is the method called `main` of the indicated class on the command line (here, HelloWorld).
- The `main` method does not return any value (it is of type void).
- The `main` method takes only one parameter, an array of strings (denoted `String argv[]`). This array will have one element for each word on the command line *following* the name of the class being executed.

Simple Java Program (Contd.)

- System.out in Java is roughly equivalent to cout in C++ (or stdout in C)
- System.out.println(whatever);is (even more) roughly equivalent to
- **cout << data<< endl;**
- **Multiline comment -** /* data*/
- **Single line comment** - // data

Java vs C++

- Java is much like C++, with a few differences that mainly make life easier
- Biggest differences:
  - Garbage Collection
  - Variables and Pointers
  - String Operations
  - Constructors

1. Garbage Collection

- There is no delete operator in Java.
- The Java system automatically deletes objects when no references to them remain.
- This is a much more important convenience than it may at first seem. Delete is extremely error-prone.
  - Deleting objects too early can lead to a *dangling* reference
  - Deleting them too late (or not at all) can lead to a *memory leak*.

2. Variables, Objects and Pointers

- Objects can *only* be referenced with pointers. Variables can **hold primitive values** (such as integers or floating-point numbers) or *references* (like pointers) to objects. A variable cannot hold an object, and you cannot make a pointer to a primitive value.
- Since you don't have a choice, Java doesn't have a special notation like C++ does to indicate when you want to use a pointer.
- There are exactly eight **primitive types** in Java: boolean, char, byte, short, int, long, float, and double.
- Note that '**int**' is a primitive type, whereas '**Integer**' is an object.

Variables, Objects and Pointers - Example

```
class Pair { int x, y; }

public class Example{
void f() {        int n = 1;
        Pair p = new Pair();
        p.x = 2; p.y = 3;
        System.out.println(n);     // prints 1
        System.out.println(p.x);  // prints 2
        g(n,p);
        System.out.println(n);     // still prints 1
        System.out.println(p.x);  // prints 100
    }
void g(int num, Pair ptr) {
      System.out.println(num);   // prints 1
      num = 17;                          // changes only the local copy
      System.out.println(num);  // prints 17
      System.out.println(ptr.x); // prints 2
      ptr.x = 100;                     // changes the x field of caller's Pair
      ptr = null;                      // changes only the local ptr
   }
}
```

Intermission: Group Exercise!

- ```
  class Test {
  static void Swap(Integer j, Integer k) {
        int tmp = k.intValue();
        k = new Integer(j.intValue());
        j = new Integer(tmp);
     }
     public static void main(String[] args) {
        Integer n = new Integer(5), m = new Integer(6);
        Swap(n, m);
        System.out.println("n = " + n + "; m = " + m);
     }
  }
  ```
- The person who wrote this program expected the output to be: n = 6; m = 5. However, the actual output is: n = 5; m = 6.

- Make a group with 2 neighbors. Spend 3 minutes discussing why.
  ```

Group Exercise: Explanation

- In Java, all parameters are passed by value, so changes to the parameters themselves in the function do not affect the values that were passed.

3. String Operations

- The + operator is overloaded to mean concatenation
- You can concatenate anything with a string (Java automatically converts the data to a string).
  - Built-in types such as numbers are converted in the obvious way
  - Objects are converted by calling their toString() method

3. String Operations

- CREATING STRINGS

```
String a = new String ("Hello World!");
System.out.println(a);

String S1 = "hello",              // initialize from a string literal
S2 = new String("bye"),    // use new and the String constructor
S3 = new String(S1);       // use new and a different constructor
```

- STRING CONCATENATION (+ Operator)

```
String S1 = "hello" + "bye",
 S2 = S1 + "!",
 S3 = S1 + 10;
```

4. Constructors

- A method with the same name as the class. If a constructor has arguments, you supply corresponding values when using **new**. Even if no arguments, you still need the parentheses (unlike C++).
- There can be multiple constructors, with different numbers or types of arguments. The same is true for other methods.

4. Constructors

```
class Pair {
    int x, y;
    Pair(int u, int v) {
       x = u;  // the same as this.x = u
       y = v;
    }
    Pair(int x) {
       this.x = x;  // not the same as x = x
       y = 0;
    }
    Pair() {
       x = 0;
       y = 0;
    }
  }
```

```
class Test {
    public static void main(String[] argv)
{
       Pair p1 = new Pair(3,4);
       Pair p2 = new Pair();  // same as
new Pair(0,0)
       Pair p3 = new Pair;    // error!
    }
  }
```

Intermission 2: Group Exercise!

- Rewrite this C++ program in Java.
- Make a group with 2 neighbors and spend 5 minutes doing this

```
#include<iostream>

using namespace std;

int main(int argc, char** argv) {
	int i1 = 3;
	int i2 = 4;
	if (i1 > i2) {
		cout << "i1 is larger.";
	} else if (i2 > i1) {
		cout << "i2 is larger";
	} else {
		cout << "Both are equal";
	}
	return 0;
}
```

Intermission 2: Answer!

```
public class Tutorial {
    public static void main(String[] args) {
        int i1 = 3;
        int i2 = 4;
        if (i1 > i2) {
            System.out.print("i1 is larger.");
        } else if (i2 > i1) {
            System.out.print("i2 is larger");
        } else {
            System.out.print("Both are
          equal");
        }
        return;
    }
}
```

```
#include<iostream>

using namespace std;

int main(int argc, char** argv) {
    int i1 = 3;
    int i2 = 4;
    if (i1 > i2) {
        cout << "i1 is larger.";
    } else if (i2 > i1) {
        cout << "i2 is larger";
    } else {
        cout << "Both are equal";
    }
    return 0;
}
```

Programming Hints: Simple Wrappers

- s = new Integer(i).toString();
  - Make i into an Integer object, then use toString() on the object
- s = "" + i;
  - The addition forces i to be converted to a String since we add an empty string (""), the result string contains just the integer value
- int I = Integer.parseInt(“123“);
  - Convert a String “123” to an integer using the static method parseInt() of the Integer class.

Programming Hints: Style Guidelines

- Names of classes
  - CamelCase starting with a capital letter.
  - If the most natural name for the class is a phrase, start each word with a capital letter, as in StringBuffer.
- Names of methods
  - **mixed case letters**, beginning with a lower case letter and starting each subsequent word with an upper case letter.
- Names of "constants”
  - ALL_UPPER_CASE. Separate words of phrases with underscores as in MIN_VALUE.
- Each class definition should go in a separate file, and the name of the source file must be exactly the same (including case) as the name of the class, with ".java" appended.

Programming Hints: Manual Compilation (1)

```
class HelloTest {
        public static void main(String[] args) {
       Hello greeter = new Hello();
              greeter.speak();
   }
}
class Hello {
   void speak() {
      System.out.println("Hello World!");
   }
}
```

- Put each class in a separate file (HelloTest.java and Hello.java). Then try this:

  javac HelloTest.java

  java HelloTest

Programming Hints: Manual Compilation (2)

- You should see a cheery greeting. If you type **ls** you will see that you have both HelloTest.class and Hello.class even though you only asked to compiled HelloTest.java. The Java compiler figured out that class HelloTest uses class Hello and automatically compiled it. Try this to learn more about what's going on:

```
rm -f *.class
javac -verbose HelloTest.java
java HelloTest
```

Additional Resources

- Java for C++ Programmers
  - http://triton.towson.edu/~mzimand/os/Lect2-java-tutorial.html

- Exercises
  - http://pages.cs.wisc.edu/~hasti/cs368/JavaTutorial/