CPSC 324
Topics in Java Programming

Lecture 3:
Basic Java Programming Constructs
Continued

Agenda

• Commenting
• Exercises
• More on basic language constructs
  – Arrays, Math functions, Static Methods, Strings
• Reading Assignments
  – Core: Ch. 3: 50-51, 53-58, 63-64, 90-104
Challenge question

- What is the signature for a Java main method?

Answer: public static void main string bracket bracket args

Assignment #0

Documenting your Java source code (see all 66 rules in “The Elements of Java Style”, Chapter 4)

/**
 * The <code>HelloWorld</code> class implements the standard
 * hello world program.
 * @author Shawn Bowers
 * @version Assignment 0, 01/14/2010
 */
public class HelloWorld {

/**
 * Prints the hello world string.
 * @param args the command line arguments
 */
public static void main(String[] args) {
    System.out.println("Hello World!");
    }
}
Running javadoc

• Run the “javadoc” command to get the documentation
  
  % javadoc -author -version HelloWord.java -d docs

• Open docs/index.html file

Exercise 1
Exercise 1 Answers

```java
public class DooBee {
    public static void main(String[] args) {
        int x = 1;
        while(x < 3) {
            System.out.print("Doo");
            System.out.print("Bee");
            x = x + 1;
        }
        if(x == 3) {
            System.out.print("Do");
        }
    }
}
```

What would happen if we used `println`?

Exercise 2
Exercise 2 Answers (paren removed)

```java
public class Shuffle1 {
    public static void main(String[] args) {
        int x = 3;
        while(x > 0) {
            if(x > 2) {
                System.out.print("a");
                x = x - 1;
            }
            System.out.print("-");
            if(x == 2) {
                System.out.print("b c");
            }
            if(x == 1) {
                System.out.print("d");
                x = x - 1;
            }
        }
    }
}
```

Java Arrays

- An array is used to store a collection of values:
  ```java
  int[] theArray = {100, 10, 1000, 20, 200};
  ```

- You access elements in an array by index
  ```java
  theArray[0] = 100; // the first element in the array
  theArray[1] = 10; // the second element in the array
  ```

- You can get the length of the array using “length”
  ```java
  System.out.print(theArray.length); // prints “5”
  ```

This is very useful... and is not available in C++!
Java Arrays

• Using arrays in loops
  
  ```java
  for(int i = 0; i < theArray.length; i++)
      System.out.println(theArray[i]);
  ```

• You can also use arrays in a for-each loop
  
  ```java
  for(int a : theArray)
      System.out.println(a);
  ```

This is much easier to read ... Use foreach whenever possible

Declaring Java Arrays

• These are all equivalent ways to declare the array:
  
  ```java
  int[] theArray; // this is the “standard”
  ```

  ```java
  int [] theArray;
  ```

  ```java
  int theArray[];
  ```

  ```java
  int theArray [];
  ```

• This is not an allowed declaration in Java:
  
  ```java
  int[5] theArray;
  ```
Initializing Java Arrays

- You can use a constant (like we did before)
  ```java
  int[] theArray = {100, 10, 1000, 20, 200};
  ```

- Or you can explicitly allocate memory:
  ```java
  int[] theArray = new int[5];
  ```

- This creates an array of 5 elements, each initialized to the value 0
  - Boolean arrays are initialized to false
  - Arrays of object types are initialized to null (more later)

Initializing Java Arrays

- Once you declare an array
  ```java
  int[] theArray = {100, 10, 1000, 20, 200};
  // or
  int[] theArray = new int[5];
  ```

- You cannot initialize it again with a const
  ```java
  theArray = {1, 2, 3, 4, 5};  // compile error!
  ```

- You can assign it a new array
  ```java
  theArray = new int[5];
  theArray = new int[] {1, 2, 3, 4, 5};  // an "anonymous" array
  ```
Java Arrays

• Array vars are really references (pointers) to arrays
  
  ```java
  int[] smallPrimes = {2, 3, 5, 7};
  ```

  ```
  smallPrimes = [2, 3, 5, 7]
  ```

  "smallPrimes" holds a "reference" to the array in memory

  ```java
  int[] luckyNumbers = smallPrimes;
  luckyNumbers[3] = 8;
  ```

  ```java
  int smallPrimes[] = "smallPrimes" holds a "reference" to the array in memory
  ```

  • This means we have to be careful when we copy (assign) one array to another array var
  
  ```java
  int[] luckyNumbers = smallPrimes
  luckyNumbers[3] = 8;
  ```

Copying Arrays

• One way to copy an array:
  
  ```java
  int len = primeNumbers.length;
  int[] luckyNumbers = new int[len];
  for(int i = 0; i < len; i++)
      luckyNumbers[i] = primeNumbers[i];
  ```

  ```java
  Could foreach be used here?
  ```

  • Java provides helper functions for copying:
  
  ```java
  int [] luckyNumbers = Array.copyOf(primeNumbers, len);
  ```

• For more Array helpers, see the online api docs
Command-Line Parameters

Passed to main as an array of Strings

```java
public static void main(String[] args)
```

So, if we issue the command:

```
java MyEliza terms.txt defaults.txt
```

- How do we check if we have the *right number* of args?
  ```java
  if(args.length < 2) {
    // print usage message, and call System.exit(64);
  }
  ```

- How do we get the *first and second* arg?
  ```java
  String termsFileName = args[0];
  String defaultsFileName = args[1];
  ```

Math Functions

- The Math class provides various *helper functions*
  ```java
  double x = Math.random();  // random value in [0.0, 1)
  double y = Math.sqrt(15);
  double z = Math.pow(10, 3.2);
  etc.
  ```

- For example, return a *random* value between 0 and 9
  ```java
  int x = (int) (Math.random() * 10);
  ```
Static Methods

• Classes can have static fields and methods

• The Math methods are examples of static methods
  
  ```java
  public class Math {
    ...
    public static double random() {
      ...
    }
  }
  ```

• Static methods are called directly on the class, without creating an object
  
  ```java
  double x = Math.random();
  ```

• Static methods can only work at the “class” level

• They can only access local variables (including arguments), static fields, and static methods

This one is a bit tricky
We’ll talk about it later

Each method must provide

– Access level/visibility … public, protected, private, or none
– Whether it is static or non-static (default)
– Return type (primitive, array, or object type)
– Method name
– Zero or more arguments (type and name)
– Method body (in {}’s)
Static Methods

• Is this OK?

```java
public class HelloAgainWorld {
    public static String theString = "Hello World!";
    public static String getString() {
        return theString;
    }
    public static void setString(String aString) {
        theString = aString;
    }
    public static void main(String[] args) {
        setString("bonjour monde !");
        System.out.println(getString());
    }
}
```

Static Methods

• What about this version?

```java
public class HelloAgainWorld {
    public String theString = "Hello World!";
    public String(getString) {
        return theString;
    }
    public void setString(String aString) {
        theString = aString;
    }
    public void main(String[] args) {
        setString("bonjour monde !");
        System.out.println(getString());
    }
}
```
Java Strings

• A String is a “blend” of object and primitive value
  – Strings are really objects
  – But they have a special representation (using “”)
  – Strings in Java denote sequences of Unicode characters

Examples

String empty = ""; // an empty string
String shortstr = "a"; // a string of one char
String course = "CPSC 324"; // a string of multiple chars
String title = "Topics in Java Programming";

Java Strings

• A character sequence in double quotes is an object of type String
  – “CPSC 324” is a String object
  – As is “” (the empty string)

• Like in other languages, strings do not span multiple lines

• Strings can be concatenated using the ‘+’ operator

  String fullTitle = course + ":" + title;
  String abc = "a" + "b" + "c"; // “abc"
  String mixed = "a" + 1 + 2 + "c"; // “a12c"
  String nested = "a" + (1 + 2) + "c"; // “a3c"
Java Strings

• Strings are **immutable**
  – Once a String object is created, it cannot be modified!
  – For efficiency reasons
    • common strings can be shared
    • can reduce memory use (same string only stored once in memory)
  – Also helps with concurrency (threads)

• This means that operations on String objects create new String objects
  – This was the case for concatenation (+)
  – str1 + str2 creates a new String object

String Operations

• There are a large number of String helper methods

• Most of these are non-static (work over objects)

  String course = "CPSC 324";
  String dept = course.substring(0, 4); // "CPSC"
  String cno = course.substring(5, 8); // "324"
  String lower = "CPSC 324".toLowerCase(); // "cpsc 324"
  String trimmed = " CPSC 324 " .trim(); // "CPSC 324"
  String title = "Topics in Java Programming";
  int index = title.toLowerCase().indexOf("java"); // 10

• Be sure to look at the String class in the java api