Java Primer

CS2810 – Advanced Programming Lab

Akhilesh Godi

Slides adapted from Matt Hibbs' classes

Why Java?

- Cross Platform
  - Bytecode and the JVM run on most platforms
- Security
  - JVM can restrict access to local machine
- Safety
  - Garbage Collection (fewer memory leaks)
- Ease of Coding
  - Many available packages

Java Basics

http://docs.oracle.com/javase/tutorial/

- Development Environments
  - Eclipse
  - Emacs/Vim
  - gedit and Command Line

Hello World

- In C

```
int main (int argc, char** argv) {
   printf(“Hello World!\n”);
   return 0;
}/* end main */
```

- In Java

```
public class HelloWorld {
   public static void main (String[] args) {
        System.out.println(“Hello World!”);
   }/* end main */
}//end HelloWorld
```

- Things to notice:
  - Similar syntax
  - Classes
  - `System.out.println()`
  - `main` is `void: public static void main`

How Java Differs from C

- Exclusively Object-Oriented Language
  - EVERYTHING must live in a class (mostly)
  - No Global Variables
- No Pointers
  - Also, no ‘*’ ‘->’ or ‘&’ operators
  - Blessing and a Curse
    - Garbage Collection
    - Loss of Power

How Java Differs from C

- No Preprocessor (no `#include`, `#define`, etc.)
- No `goto` statement
- Declare/Define Variables & Methods anywhere (within a class)
- No `struct`, `enum`, or `typedef`
- Can’t overload Operators
- Use `new` rather than `malloc()`

Java – Data Types

- Primitives
  - `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`
    - char is Unicode (16 bits)
    - boolean is true/false (not 1/0)

```
int i = 1;
if (i) { } //BAD
if (i == 1) { } //GOOD
```

Java – Data Types

- Primitives
  - Type conversions
    - Can’t convert boolean

```
boolean b = false;
int i = b; //BAD
int j = b ? 1 : 0; //GOOD
```

    - Converting ‘up’ is automatic
    - Converting ‘down’ requires a cast

```
double f = 37.5; // it used to be float but
  // gave compiler error
int i = f; //BAD
int j = (int) f; //GOOD (truncates)
```

Java – Data Types

- References (by reference vs. by value)
  - Everything that's not primitive (classes and arrays)
  - Think of these as hidden pointers

```
Foo a = new Foo();
Foo b = a;
```

    - a and b now reference the same object

Reading From the Console

- Java's Scanner object reads in input that the user enters on the command line.

```
Scanner input = new Scanner(System.in);
```

- System.in is a reference to the standard input buffer.
- We can read values from the Scanner object using the dot notation to invoke a number of functions.
  - nextInt() — returns the next integer from the buffer
  - nextFloat() — returns the next float from the buffer
  - nextLine() — returns the entire line as a String
- In order to use Scanner class you need to add the line

```
import java.Util.Scanner
```

Java Development

Java Platform

- Lots of built-in objects and functions for various purposes
  - Graphics (java.awt)
  - Math Functions (java.math)
  - Networking (java.net)
  - Databases (java.sql)
- We'll mostly use the platform for data structures (java.util) and I/O (java.io)

Java Packages

- Java organizes classes into larger groups called packages
- Both the Java Platform and any classes that you write are organized into packages
- For code that you write, don’t worry so much about what package you’re in (the default is fine)

Using the Java Platform

- You can always use the full package path name to access classes

```
java.io.FileReader fr = new java.io.FileReader (“test.txt”);
```

- But that’s pretty annoying, so you can use import statements at the beginning of your code to avoid this

```
import java.io.FileReader;
//...
   FileReader fr = new FileReader(“test.txt”);
//...
```

Object Orientation

Java - Classes

Here's a simple Java class that we'll break down:

```
public class Circle {
  private float radius;

  public static final double PI = 3.14159;

  public Circle(float r) {
   radius = r;
  }

  public float area() {
   return PI * radius * radius;
  }
}
```

Java - Classes

- Naming

  `public class Circle {`

  – The basic form is:
    - <modifiers> `class` <name> `{`

  – Typically need an *access modifier*
    - `public`, `protected`, `private`

  – Can have additional modifiers
    - `static`, `final`, `abstract`

- If a Class is `public`, it MUST be the only `public` class in its file, and this file MUST be called <name>.java

Java – Access Modifiers

- Access modifiers control who has access to a variable or class

  `public` – anything can access it

  `protected` – only classes in same package can access (not important for 402)

  `private` – only accessible within the class that defines it

Java - Classes

- Instance Field

```
private float radius;
```

  - Any variable declared in a class is a field
  - Typically have an *access modifier*
  - In this case the field is uninitialized, so it will get the default value (0).

Java - Classes

- Class fields and methods
  - Use `static` modifier
  - Are instance-independent
  - Refer to using class name e.g. Circle.PI
  - No need to include class name if using within class
  - If only used static methods, would not be OO

Java - Classes

- Constant class field

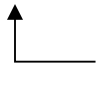
```
public static final double PI =
3.14159;
```

  - Again, typically have an *access modifier*
  - If you want to have a constant variable, also use the `static` and `final` modifiers
    - `static` means that there is only one `PI` no matter how many instances of `Circle` we make
    - `final` means that `PI` cannot be changed

Java - Classes

- Constructors

```
public Circle(float r) {radius = r;}
```



Name of the class

  - This is what gets called when an instance of your class is created
  - Typically used to just initialize fields
  - If you don’t write one, you get a *default constructor* for free, which does nothing
  - Refer to other constructors with this()

Java - Classes

- Methods

```
public float area() {
 return PI * radius * radius;
}
```

  - Again, need an *access modifier*
  - Need a return type (`void` is a valid choice)
  - Can optionally pass in arguments

Java - Classes

- Here's the whole thing again:

```
public class Circle {
  private float radius;

  public static final double PI = 3.14159;

  public Circle(float r) {
   radius = r;
  }

  public float area() {
   return PI * radius * radius;
  }
}
```

Java - Objects

- So, we've made a class, now what?
- We can start making circles…

```
public Circle c1;
public Circle c2 = new Circle();
public Circle c3 = new Circle(2.5);
```

  - `c1` didn't actually make a circle, it just made a circle reference
  - `c2` used the default constructor, and has radius 0
  - `c3` used our constructor, and has radius 2.5

Java - Objects

- Now that we have circles, we can call methods on them

```
public float f = c3.area();
System.out.print(f); //19.6349
```

Questions?