

Introduction to Java

CMSC 331

updated October 2015

Introduction

- Present the syntax of Java
- Introduce the Java API
- Demonstrate how to build
 - stand-alone Java programs
 - Java applets, which run within browsers e.g. Netscape
- Example programs

Why Java?

- It's the current “hot” language
- It's almost entirely object-oriented
- It has a vast library of predefined objects and operations
- It's more platform independent
 - this makes it great for Web programming
- It's more secure
- It isn't C++

Applets, Servlets and Applications

- An *applet* is designed to be embedded in a Web page, and run by a browser
- Applets run in a *sandbox* with numerous restrictions; for example, they can't read files and then use the network
- A *servlet* is designed to be run by a web server
- An *application* is a conventional program

Building Standalone JAVA Programs (on UNIX)

- Prepare the file `foo.java` using an editor
- Invoke the compiler: `javac foo.java`
- This creates `foo.class`
- Run the java interpreter: `java foo`

Java Virtual Machine

- The .class files generated by the compiler are not executable binaries
 - so Java combines compilation and interpretation
- Instead, they contain “byte-codes” to be executed by the Java Virtual Machine
 - other languages have done this, e.g. UCSD Pascal
 - other languages use the JVM, e.g. Clojure
- This approach provides platform independence, and greater security

HelloWorld (standalone)

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

- Note that String is a built in type
- println is a member function for the System.out class

Comments are almost like C++

- `/* This kind of comment can span multiple lines
*/`
- `// This kind is to the end of the line`
- `/**
 * This kind of comment is a special
 * 'javadoc' style comment
 */`

Primitive data types are like C

- Main data types are `int`, `double`, `boolean`, `char`
- Also have `byte`, `short`, `long`, `float`
- `boolean` has values `true` and `false`
- Declarations look like C, for example,

```
double x, y;  
int count = 0;
```

Expressions are like C

- Assignment statements mostly look like those in C; you can use `=`, `+=`, `*=` etc.
- Arithmetic uses the familiar `+` `-` `*` `/` `%`
- Java also has `++` and `--`
- Java has boolean operators `&&` `||` `!`
- Java has comparisons `<` `<=` `==` `!=` `>=` `>`
- Java does *not* have pointers or pointer arithmetic

Control statements are like C

- `if (x < y) smaller = x;`
- `if (x < y){ smaller=x; sum += x; }
else { smaller = y; sum += y; }`
- `while (x < y) { y = y - x; }`
- `do { y = y - x; } while (x < y)`
- `for (int i = 0; i < max; i++)
 sum += i;`
- BUT: conditions must be **boolean** !

Control statements II

```
switch (n + 1) {  
    case 0: m = n - 1; break;  
    case 1: m = n + 1;  
    case 3: m = m * n; break;  
    default: m = -n; break;  
}
```

- Java also introduces the **try** statement, about which more later

Java isn't C!

- In C, almost everything is in functions
- In Java, almost everything is in classes
- There is often only one class per file
- There *must* be only one **public** class per file
- The file name *must* be the same as the name of that public class, but with a **.java** extension

Java program layout

- A typical Java file looks like:

```
import java.util.*;
import java. //other imports

public class SomethingOrOther {
    // object definitions go here
    . . .
}
```

This must be in a file named `SomethingOrOther.java` !

What is a class?

- Early languages had only arrays
 - all elements had to be of the same type
- Then languages introduced structures (called **records**, or **structs**)
 - allowed different data types to be grouped
- Then Abstract Data Types (ADTs) became popular
 - grouped operations along with the data

So, what is a class?

- A class consists of
 - a collection of *fields*, or *variables*, very much like the named fields of a struct
 - all the operations (called *methods*) that can be performed on those fields
 - can be *instantiated*
- A class describes objects and operations defined on those objects

Name conventions

- Java is case-sensitive; `maxval`, `maxVal`, and `MaxVal` are three different names
- Class names begin with a capital letter
- All other names begin with a lowercase letter
- Subsequent words are capitalized: `theBigOne`
- Underscores are not used in names
- These are *very strong* conventions!

The class hierarchy

- Classes are arranged in a hierarchy
- The root, or topmost, class is **Object**
- Every class but **Object** has at least one superclass
- A class may have subclasses
- Each class *inherits* all the fields and methods of its (possibly numerous) superclasses

An example of a class

```
class Person {  
    String name;  
    int age;  
  
    void birthday ( ) {  
        age++;  
        System.out.println (name +  
            ' is now ' + age);  
    }  
}
```

Another example of a class

```
class Driver extends Person {  
    long driversLicenseNumber;  
    Date expirationDate;  
}
```

Creating and using an object

- ```
Person john;
john = new Person ();
john.name = "John Smith";
john.age = 37;
```
- ```
Person mary = new Person ( );  
mary.name = "Mary Brown";  
mary.age = 33;  
mary.birthday ( );
```

An array is an object

- `Person mary = new Person ();`
- `int myArray[] = new int[5];`
 - or:
- `int myArray[] = {1, 4, 9, 16, 25};`
- `String languages [] = {"Prolog", "Java"};`