Getting Started in Java

Bill Pugh
Dept. of Computer Science
Univ. of Maryland, College Park
Hello, World

- In HelloWorld.java
  ```java
  public class HelloWorld {
      public static void main(String[] args) {
          System.out.println("Hello world");
      }
  }
  ```
- Enter `javac HelloWorld.java`
- Enter `java HelloWorld`
Not in this talk

• Why Java made the front page of Business week
• Why Microsofts J++ isn’t Java
• Creating graphical user interfaces or applets
  – next week
• Drawing graphics
  – some next week, see Ben Bederson’s talk on Java 2D Graphics API later as well
What you need to know

• How to program in C
  – basic language pretty much the same
• How to program in C++
  – some things the same
  – some things very different
• Having taken CMSC330 is helpful
  – some terms will be unfamiliar if you haven’t taken 330
No global variables/methods

• All code and variables are attached to objects/classes
  – there are no global variables/functions
• use static methods/variables instead
  • a static method/variable of a class C is not associated with an instance of class C
Class names

• Java class names consist of a package name and a simple class name
  – e.g., java.io.PrintStream
    • Package java.io
    • Simple class name PrintStream
• package names are lower case
• simple class names are Capitalized
• full class name = packageName.SimpleClassName
  – by default, need to use full class name
Null package

• You can have an empty package name – the null package
• full class name same as simple class name
• Code that won’t be used by anyone else can put in the null package
• I’ll assume all of your development will be in the null package
Java Classes

• Every class extends some other class
  – class java.lang.Object if not specified
  – except for class java.lang.Object

• java.lang.Object has a number of methods defined
  – toString(), equals(Object o), hashKey()

• You may wish to override these methods
Java Object Model

• All objects/(instances of a class) are allocated on the heap
• A method/class/instance variable is
  – a primitive value
    • (int, long, float, double, boolean,…)
  – a reference to a heap allocated object
• No objects contained in another object
Java Objects

class A {
    int x;
    B b;
}
class B {
    int y;
}
A a = new A();
a.x = 42;
// allocation of an A
// doesn’t allocate a B
a.b = new B();
a.b.y = 17;

Getting started in Java
Writing Java Classes

• The simple way to do things
  – Not the only way
• A class with the simple name C
  – goes into a file C.java
• run javac C.java
  – to produce C.class
Using import

• Put import statements at beginning of .java file

• import java.io.PrintStream;
  – allows use of PrintStream rather than java.io.PrintStream

• import java.io.*;
  – does the above for all classes in java.io

• import java.lang.*; implicit
Java Applications

• Invoke the java virtual machine:
  – java fullClassName arguments

• Expects that the class have a method:
  – public static void main(String [] args) {

• arguments to main are command line arguments
Arrays

• Java arrays are objects
  – they have a variable length
  – can be indexed as a[i]
    • where $0 \leq i < a\text{.length}$

• Can have arrays of primitive types or of references

• All array references are bounds checked
Array Example

```java
int primes[];
// declaring primes doesn’t allocate it
primes = new int[100];
int p = 0;

for (nextPrime = 2; p < primes.length; nextPrime++) {
    for (int r=0; r<p; r++) if (nextPrime % primes[r] == 0) break;
    if (r == p) primes[p++] = nextPrime;
}
```
java.lang.String

• Widely used class
• Represents an immutable string
• “String constants” are java.lang.String objects
• All objects have a toString() method
• + does string concatenation
  – System.out.println(“x = “ + x);
Text Output

• PrintStream/PrintWriter
  – classes for text output

• both provide print/println methods
  – for all single arguments
  – println with no arguments

• PrintStream deprecated
  – but java.lang.System.out and java.lang.System.err are PrintStreams
Text Input

• System.in is a java.io.InputStream
  – reads raw bytes
    BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
  – InputStreamReader converts from bytes to Unicode
  – BufferedReader supports String readLine()
    • returns null at EOF
File Text I/O

- PrintWriter out = new PrintWriter(new FileWriter(fileName));
- BufferedReader in = new BufferedReader(new FileReader(fileName));
java.StreamTokenizer

• Breaks input up into words, numbers and characters
  – by default ignores whitespace
  – can ignore comments
  – can handle quoted strings
  – Can control which characters make up words
Finding doubled words

• When writing papers, I often leave doubled words in documents
• Find repeated words in a list of text files
  – Files specified as command line arguments
  – Create a StreamTokenizer for each file
import java.io.*;
public class Doubled {
    public static void main(String args[]) throws Exception {
        for(int i = 0; i < args.length; i++) {
            boolean fileNamePrinted = false;
            StreamTokenizer stream = new StreamTokenizer( new FileReader(args[i]));
            String prevWord = null;
            while (true) {
                int token = stream.nextToken();
                if (token == StreamTokenizer.TT_EOF) break;
                if (token == StreamTokenizer.TT_WORD) {
                    if (stream.sval.equals(prevWord)) {
                        if (!fileNamePrinted) {
                            System.out.println("File " + args[i]);
                            fileNamePrinted = true;
                        }
                        System.out.println("  " + stream.lineno() + " : " prevWord);
                    } else prevWord = stream.sval;
                } else prevWord = null;
            }
        }
    }
}
Subtyping and extension

• A class can extend another class
  – class B extends A {...}
  – a B can be used where ever an A is expected
  – A variable of type A can point to an object of type B

• A variable of type Object can point to any Object
Method overriding

• If you redefine a method with exactly the same name and arguments
  – you override the previous definition
• All instance methods are virtual methods
  – If both class A and B define an f() method
  – A a = new B(); a.f(); invokes B’s version
Interfaces

• An interface is a class
  – with no executable code
  – with no instance variables

• Only method names and arguments
  – public interface Comparable {
    public int compareTo(Object o);
  }
Implementing an interface

• A class an implement any number of interfaces
  – Can extend only one class
• Must support all methods defined by interface
• A variable of an interface type
  – can only reference an object that implements that interface
Using interfaces

• void sort(Comparable[] a) { ...}
Object []

- An object variable can point to any object
  - e.g., a String
- Consider B b[] = new B[100];
  - b[42] can point to a B
  - or to an instance of a subclass of B
Reverse an Object array

```java
public static void reverse(Object objs[]) {
    int i = 0; int j = objs.length-1;
    while (i < j) {
        Object tmp = objs[i];
        objs[i++] = objs[j];
        objs[j++] = tmp;
    }
}
```
Subtyping among arrays

• If class B is a subclass of A
  – B[] is a subclass of A[]
• E.g., String[] is a subclass of Object[]
  – can pass a String[] to reverse(Object[])
• Object[] is a generic container
Collection Classes

• All java collections are collections of Objects
  – no templates
• java.util.Dictionary implements a map
  – from keys to values
• Much better Collection classes in 1.2
Dealing with exceptions

• Some methods can throw exceptions
  – e.g., some IO methods throw IOException
• A method can’t throw an exception unless it declares the exception
  – unless it is a subclass of Error or RuntimeException
  – When overriding, can’t add new exceptions
  – Must catch errors you don’t declare
Declaring exceptions

• Simple way to handle exceptions
  public static void main(String[] args)
  throws Exception { ... }

• Uncaught exceptions kill Thread
Catching Exceptions

• You can catch exceptions:
  
  ```java
  try {
      ...
  } catch (IOException e) {
      System.out.println("Caught IO error");
      e.printStackTrace();
  }
  ```
Network Fun

• network libraries in java.net
• java.net.URL
  – used for Uniform Resource Locations
  – e.g., http://www.cs.umd.edu
• java.net.Socket
  – used to open to connection to a server
• java.net.ServerSocket
  – used to host a socket connection
import java.io.*;
import java.net.*;

public class URLGet {
    public static void main(String [] args) throws Exception {
        if (args.length != 1) {
            System.out.println("Please supply one URL as an argument");
            return;
        }

        URL u = new URL(args[0]);
        BufferedReader in = new BufferedReader(new InputStreamReader(u.openStream()));
        String s;
        while((s = in.readLine()) != null) System.out.println(s);
    }
}

Getting started in Java
A little bit more work

• Get information such as last modified date for a URL