Java Basics

Tom Wheeler

What I'm Going to Cover

- What is Java
- Define terms like JDK, JRE and VM
- Describe a couple of basic programs
- Visibility modifiers
- Memory management and threads
- Input and Output
- Exceptions, Collections API and generics
- The popular Ant build tool

What is Java?

- A language
- A class library
- A virtual machine
- All of these form the “Java Platform”
- JavaScript is an unrelated language!
  - Mainly for client-side Web programming
  - It does have a similar syntax though

History of Java

- First there was ALGOL (1958)
- Then there was C (1972)
- Then there was C++ (1983)
- Then there was Java (1995)
- Just as C++ has syntax similar to C
  - Java has a syntax similar to C++

Version Number Nonsense

- Java 1.0
- Java 1.1
- Java 1.2 (AKA Java 2 SE or J2SE)
- Java 1.3 (AKA J2SE version 1.3)
- Java 1.4 (AKA J2SE version 1.4)
- Java 5 (AKA J2SE version 1.5)
- Java 6 (AKA Java SE 6)

Java Concepts

- Object-oriented
- Not native code
  - But not interpreted either
  - Compilation target is a virtual machine (VM)
  - Result of compilation is called “bytecode”
  - Can run unchanged on nearly any system
- Because its platform-independent
  - Sizes of data types are always the same
  - Tend towards “lowest common denominator”

Java Tools

- JDK = Java Development Kit (developer)
- JRE = Java Runtime Environment (user)
- JDK and JRE both have a `bin` directory
  - Contains tools needed to compile/run
  - You need to add the `bin` directory to PATH
- The two most important tools are:
  - `javac` Java compiler (for JDK only)
  - `java` Java runtime (virtual machine)

Applications and Applets

- There are two main program types
  - Applets and applications
- In the mid-1990s, applets were the rage
  - Client-side app which runs in browser
- Now applets are somewhat rare
  - And applications are the most common
- Various types of applications
  - Console, GUI (Swing) and Web apps

Data Types

- There are two main data types
  - Primitives (int, float, double, byte, etc.)
  - Objects (everything else)
- There are "wrapper classes" for primitives
  - e.g. int <-> Integer, double <-> Double
  - Autoboxing in Java 5 makes this easier
- Arrays are also objects
- Strings are immutable

A Basic Program (1)

- Class (and public modifier)
- Also filename must be <Class>.java

```java
package com.tomwheeler.example;

/**
 * This is a sample class for demonstration.  It just
 * prints 'Hello world' to standard output.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Example {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

A Basic Program (2)

- Package (directory/namespace)

```java
package com.tomwheeler.example;

/**
 * This is a sample class for demonstration.  It just
 * prints 'Hello world' to standard output.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Example {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

A Basic Program (3)

- Javadoc

```java
package com.tomwheeler.example;

/**
 * This is a sample class for demonstration.  It just
 * prints 'Hello world' to standard output.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Example {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

A Basic Program (4)

- The main method/signature
- `System.out.println` is like `cout <<`

```
package com.tomwheeler.example;

/**
 * This is a sample class for demonstration.  It just
 * prints 'Hello world' to standard output.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Example {

    public static void main(String[] args) {
        System.out.println("Hello world");
    }

}
```

Java Visibility Modifiers

- Can use on class, field or method def.
- Purpose is to support encapsulation
- Java has four visibility modifiers
  - private
  - default (package-private)
  - protected
  - Public
- NetBeans improves upon this!

Object-Oriented Programming in Java

- Previous class didn't illustrate OOP
  - Class had no fields and only a static method
  - Static method belongs to class, not object
  - It was basically procedural
- All objects derive from java.lang.Object
  - Inheritance in this case is implicit
  - It defines just a few methods
- Let's look at another example...

A Simple Class: Person (1)

- import statement (*wildcards/java.lang*)

```
package com.tomwheeler.example;

import java.util.Date;

/**
 * This is a sample class which models a person.  It has no
 * main method, so it can be run by itself -- you will need
 * to construct it and invoke its methods from another class.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Person {

    private String name;
    private int age;
    private float weight;

    public Person() {
        // zero-argument constructor: all fields set to default
    }

    public Person(String name) {
        // one-argument constructor: only name is set; others default
        this.name = name;
    }

    public Person(String name, int age, float weight) {
        // constructor which allows all values to be set
        this.name = name;
        this.age = age;
        this.weight = weight;
    }
```

A Simple Class: Person (2)

- The fields (and private modifier)

```
package com.tomwheeler.example;

import java.util.Date;

/**
 * This is a sample class which models a person.  It has no
 * main method, so it can be run by itself -- you will need
 * to construct it and invoke its methods from another class.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Person {

    private String name;
    private int age;
    private float weight;

    public Person() {
        // zero-argument constructor: all fields set to default
    }

    public Person(String name) {
        // one-argument constructor: only name is set; others default
        this.name = name;
    }

    public Person(String name, int age, float weight) {
        // constructor which allows all values to be set
        this.name = name;
        this.age = age;
        this.weight = weight;
    }
```

A Simple Class: Person (3)

- The constructors (and public modifier)

```java
package com.tomwheeler.example;

import java.util.Date;

/**
 * This is a sample class which models a person.  It has no
 * main method, so it can be run by itself -- you will need
 * to construct it and invoke its methods from another class.
 *
 * @author Tom Wheeler
 * @version 1.1
 */
public class Person {

    private String name;
    private int age;
    private float weight;

    public Person() {
        // zero-argument constructor: all fields set to default
    }

    public Person(String name) {
        // one-argument constructor: only name is set; others default
        this.name = name;
    }

    public Person(String name, int age, float weight) {
        // constructor which allows all values to be set
        this.name = name;
        this.age = age;
        this.weight = weight;
    }
```

A Simple Class: Person (4)

- Some instance methods
- String concatenation
  - But *you* can't do operator overloading in Java
- Override toString and annotation

```
    public void tellMeTheDate() {
        System.out.println("The date is now " + new Date());
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Person: '" + name + "', age=" + age;
    }
}
```

Inheritance in Java

- Java supports two types
  - Implementation (“extends”)
  - Interface (“implements”)
- Abstract and final classes
- There is no multiple inheritance in Java
  - But you can implement multiple interfaces
  - NetBeans Platform has some tricks to simulate multiple inheritance

Memory Management in Java

- Memory management is automatic
  - Unlike C or C++
- Garbage collection
  - Runs periodically to free up memory
- Works well... but
  - It's still possible to have memory leaks!
- There are also no pointers in Java
  - Not that you can see, anyway (NPE)

Exceptions

- Exception handling is built into Java
- Throwable: Error and Exception
- Two types of exceptions
  - Checked and unchecked
- try/catch/finally

Threads

- Allows you to run things simultaneously
- Built into the language since 1.0
  - Improved greatly in Java 5
  - Still a very complex topic though
- General Rules
  - Access GUI from event dispatch thread only!
  - Execute long-running tasks in BG thread

Input and Output

- Streams are byte-oriented
  - Read from InputStreams
  - Write to OutputStreams
  - Many types: common to wrap them
    - e.g. FileInputStream, FilterInputStream, CipherInputStream, ZipInputStream, etc.
    - Similar classes for output
- Readers/Writers are character-oriented

Collections API

- Originally just had Vector and Hashtable
- Collections API introduced in Java 1.2
- Main interfaces
  - There are various implementations
  - Collection (base class) and Iterator
  - List: ordered collection (dupes allowed)
  - Set: unordered collection (no dupes)
  - Map: maps keys to values

Generics

- Generics were introduced in Java 5
- Allows enhanced type specification
  - Similar to STL, though not as powerful
- Especially for collections classes
- Before generics (what's in the list?)
  - `List stuff = new ArrayList()`
- After generics
  - `List<Dog> pets = new ArrayList<Dog>()`

Ant

- Having to compile each class is tedious
- Ant is a tool that helps to automate this
- Roughly analogous to “make”
  - But Ant uses an XML syntax
- Ant is open source (Apache)
- Ant is very widely used for Java projects
- Writing Ant build files is also tedious
  - NetBeans does this for you!

Review Questions

- What is the difference between JDK/JRE?
- What are the two main data types?
- What is meant by wrapper class?
- What are the four visibility modifiers?
- Does Java support multiple inheritance?
- Why would you use a “finally” block?
- Contrast a List and a Set.

Recap

- Java is an object-oriented language
  - Syntax similar to C and C++
  - Automatic memory management (GC)
  - Primitive types/wrapper classes
  - Four visibility modifiers
  - Implementation/interface inheritance
  - Streams and Readers
  - Generics and collections
  - Ant is a build tool like make

Exercise (20 minutes)

- Write a simple Java program which
  - Adds three numbers together
  - Prints the result to standard output
- You can use NetBeans IDE to help you
  - Hint: Create a new Java application project
  - Hint: Add a new class to your project
  - Hint: Add a main method to your class