Java Notes

Blake McBride (blake@mcbride.name)

Mechanics

Files

.java $\Longrightarrow$ .class $\Longrightarrow$ .jar

Each .java file can have any number of class definitions, however, each class will create a `<class>.class` file when compiled. The name of the .java file does not matter. It is, however, customary to make the class name correspond to the file name.

Oracle's Java or OpenJDK

```
Compiling:  javac  <file>.java
Running:    Java   <file>

<file> should be the same as the name of the class being defined internally.
```

If . is not in your CLASSPATH environment variable, you must include it when running any Java program.

If file uses other classes during compile or run time, you must specify the directory of the .class files it uses or the .jar file. Multiple directories / jar files can be separated by a ';' (or ':' on Linux).

```
Compiling:  javac  -classpath  .;lib.jar  <file>.java
Running:    Java   -cp  .;lib.jar  <file>
```

Jar files are built by combining .class files with the 'jar' program.

If the class is in a package, the relative directory path of the file must match the package. For example, my.pack.MyClass must reside in: `my/pack/MyClass.java`

```
Compiling:  javac  my/pack/MyClass.java
                     or
            javac  my/pack/*.java
Running:    Java   my.pack.MyClass
```

Hello World Program

```
class Test1 {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Data Types

Type	Size in bytes	Example	Boxed Type
void	0	no value	Void
boolean		true/false	Boolean
char	2 (unicode)	‘H’	Character
byte	1	(byte) 22	Byte
short	2	(short) 17	Short
int	4	27	Integer
long	8 (L suffix)	55L	Long
float	4 (F suffix)	36.7F	Float
double	8	3.14	Double
String		“Hello”	

All numeric types are signed. Strings are immutable.

Arrays

```
int[]  x;
x = new int[100];
x[0] = 44;
x.length
int[]  y = { 4, 3, 6 };
int[]  z = y;   //  z and y point to the same array

double[][]  balance;
balance = new double[33][44];
balance[4][16] = 4.55;

MyClass [] i = new MyClass[10];
```

Classes

```
/*  Sample class definition and use  */

class Test2 {
    public static void main(String[] args) {
        MyClass i = new MyClass(4);
        System.out.println(i.getIV());
        MyClass.setCV(88);
        System.out.println(MyClass.getCV());
    }
}

class MyClass {
    private static int cv;   // class variable
    private int iv;          // instance variable

    public MyClass(int v) {  // constructor
        iv = v;
    }
    public static void setCV(int v) {
        cv = v;
    }
    public static int getCV() {
        return cv;
    }

    public int getIV() {
        return iv;
    }
}
```

Use "*this*" to refer to the object being referred to.

Use "*super.method(args)*" to call a super method.

Use "*super(args)*" to execute a super constructor.

"*Object*" is the root class.

if (i instanceof MyClass) ...

i.getClass()

Constructors

Class constructors are indicated by methods with no return types and having the same name as the class name. They are used to initialize instance variables within a newly constructed class. Each class can have any number of constructors provided each has different arguments.

Subclasses

Subclasses (or extended classes, i.e. classes that extend other classes) can explicitly execute their superclass constructors via:

```
super();     // to execute the default superclass constructor
  or
super(...);  // to execute a specific superclass constructor
```

Alternatively, a subclass may call one of its own constructors with the following:

```
this();     // to execute the default constructor for this class
  or
this(...);  // to execute a specific constructor for this class
```

There are two important things however. The first is that calling one of your own constructors or calling your superclass' constructor must be the first executable line of any constructor you create if you do so at all. The second is that if you omit a call to your superclass' constructor, a call to your superclass' default constructor is implied and will be called. In other words, all constructors without explicit calls to their superclass constructor will have the following first line implied:

```
super();     // to execute the default superclass constructor
```

If a superclass has defined constructors with arguments and no default argument constructor, subclass constructors must explicitly call the desired superclass' constructor.

Constructors are not inherited.

Modifiers

```
Class:    [public]  [abstract/final]  class  <className>
          [extends  <superclass>]
          [implements  <interface> [, <interface>]...]  {  ...  }

          final = class cannot be subclassed


Inner Class:   [public/private/protected]  [static]  class  <className>
          [extends  <superclass>]
          [implements  <interface> [, <interface>]...]  {  ...  }


Interface:[public]   interface  <interfaceName>
          [extends  <superinterface>] [, <superinterface>]... {  ...  }


Method:   [public/private/protected]  [static]  [abstract/final]
          returnType  <methodName>  (  [args] )  {  ...  }

          final = method cannot be overridden and no dynamic lookup


Field:    [public/private/protected]  [static]  [final]
          <type>  <fieldName>;

          final = value cannot change


Variable: [final]  <type>  <variableName>;

          final = value cannot change
```

“static” always means item is associated with the class instead of an instance. Default access for all types is “package”.

Packages

To put classes in a file into a package use the following. (It must be the first statement in the file.)

```
package  com.company.package;
```

To access a class use:

```
com.company.package.class  var;
```

Or use:

```
import com.company.package.class;
      or
import com.company.package.*;

class  var;
```

If an import statement causes two classes with the same name in two different packages to collide, those classes must be fully qualified to be used.

The package name should dictate a corresponding directory structure of the files. For example package `com.company.package` would be located in the `com/company/package` directory.

Use static import to avoid having to specify the class of an external class method as follows:

```
import static com.company.package.class.method;
      or
import static com.company.package.class.*;
```

You can now use:

```
method(x);
      instead of
class.method(x);
```

Nested Classes

Static Nested Classes

Static nested classes are the same as outer classes except that they have access to the enclosing class' static variables. If an inside class is passed an instance of the outer class, the inside class will have direct access to the outer class' instance variables too.

References are made to the inside class via:

```
    OutsideClass.InsideClass
```

Inner Classes

Non-static or inner classes are associated with instances of the outer class. Inner classes cannot have static variables or methods. Inner classes can access the outer class' instance and class variables. Instances of inner classes can only be created in the context of an instance of the outer class. This can be done by executing the new on the inner class within an instance of the outer class, or via:

```
    InsideClass var = outsideInstance.new InsideClass();
```

The outer class would normally hold the reference to the inside class.

Exceptions

```
try {
  // code to execute, can throw exception
}
catch (Exception-type ex) {
  // handle exception of exception-type, ex is exception object
}
catch (Exception-type ex) {
  // handle another exception type
}
finally {
  //  code to execute (after everything) whether exception or not
}
```

Exceptions must be of type *java.lang.Throwable.*

Use the following to throw an exception:

```
throw new ExceptionClass(args);
```

where *ExceptionClass* is one of the classes related to *java.lang.Throwable.*

Methods that throw exceptions must declare it in their header as follows:

```
public int myMethod(args)  throws  SomeException  { ... }
```

Java IO

Binary IO Class with *Stream* in their name

Text IO Classes with *Reader* or *Writer* in their name

Classes that open files include:

- PrintWriter (has printf)
- PrintStream
- FileReader
- FileWriter
- FileInputStream
- FileOutputStream

The *File* class can be used to delete or inspect files.

Classes that convert streams to text IO include:

- OutputStreamWriter
- InputStreamReader

Classes that buffer IO include:

- BufferedReader
- BufferedWriter
- BufferedInputStream
- BufferedOutputStream

stdin System.in (InputStream)

stdout System.out (PrintStream)

stderr System.err (PrintStream)

Buffered printf with new PrintWriter(new BufferedWriter(new FileWriter("file.txt")))

JavaDoc

JavaDocs start with '/**' and end with '*/'. They must immediately precede the declaration being documented. The first sentence should be a summary.

Text inside a JavaDoc can contain the following conventions.

```
\{@code XXXXXX\}
<code>XXXXXXX</code>
<p>
```

Sample JavaDoc for a class.

```
/**
 * Description of the class....
 *
 * @author Sam Jones
 * @version 1/1/08
 * @see some.other.class
 * @see some.other.class2
 */
```

Sample JavaDoc for a method.

```
/**
 * Description of the method....
 *
 * @param param1 description-of-param1
 * @param param2 description-of-param2
 * @return description
 * @exception exception description
 * @throws exception description
 * @see
 */
```

Sample JavaDoc for a class or instance variable.

```
/**
 * Description of the variable
 */
```