An Introduction to Java for C++ Programmers[1]

1 How to program in Java

To play with Java, first make sure you have two things:

- A text editor to edit Java source code files
- An Java compilation and execution environment or tool set

We may always download JDK (Java Development Kit) from http://java.sun.com/, which includes a compiler, `javac`, and an interpreter, `java`. The compiler processes Java source files and generates class files. Class files contains instructions in the format of so-called bytecode. These instruction cannot be directly executed on traditional CPUs, instead they have to run on a JVM (Java Virtual Machine). The interpreter, `java`, implements a JVM, which translates the bytecode instructions to binary instructions and finally runs them on local microprocessors.

To give you a sense how to program in Java, let us start with a “Hello World” example.

1. Create a Java source file, named `HelloWorldApp.java`, with the following content:

```
/**
 * The HelloWorldApp class implements an application that
 * displays "Hello World!" to the standard output.
 */
public class HelloWorldApp {
    public static void main(String[] args) {
        // Display "Hello World!"
        System.out.println("Hello World!");
    }
}
```

   Note that the Java compiler and interpreter are case-sensitive, so don’t mess up the capital letters and small ones.

[1]This introduction contains materials originally from http://info.baeumle.com/java/intro/ and http://java.sun.com/.

2. Compile the source file.

   On Windows, a DOS console is needed to use the above command-line tool, the compiler, `javac`. At the prompt, enter the directory you store `HelloWorldApp.java` first, and then input the following command to compile the source file:

   ```
   javac HelloWorldApp.java
   ```

   Normally, you will see the prompt again. Congratulations! You have successfully compiled your program and you should have another file in the same directory now, `HelloWorldApp.java`.

   It is possible that Windows cannot find the command `javac`. It is because the path of `javac` is not contained in the PATH environment variable. Please do so if this indeed happens.

3. Run the Program.

   In the same directory, enter at the prompt:

   ```
   java HelloWorldApp
   ```

   You are supposed to see "Hello world!" coming out after the command is issued.

Based on my experience, it is convenient to have the following batch file to set up all the environment variables that the development of Java applications needs.

```
@ECHO OFF
set java_home=%SystemDrive%\j2sdk1.4.0
set path=.;%java_home%\bin;%path%
set CLASSPATH=.;%java_home%\jre\lib\rt.jar
%SystemRoot%\system32\cmd.exe
```

2 Principles and Foundations of Java

After getting the first Java application working, we then turn our attention to the philosophy and the foundations of Java, the basic ideas of Java and its important differences compared with C++.

2.1 Simple!

What makes Java simpler include:

- There is no pointer!

- There is no macro, conditional compilation and function prototypes, so there is no need of header files!
- There is no template and no multiple inheritance in Java either! To have the flexibility of multiple inheritance, Java supports *interface*, which may be viewed as virtual classes including only function declarations with any implementation.
- Java doesn't support variable length parameter lists like in C++.

2.2 Almost Pure Object Orientation

You are assumed to have known the basic concepts of object orientation and known how to answer the following questions:

- What is a *class*? What is an *object* or *instance*? And the relationship between them?
- What a *public* or *private method* or *attribute* of a class/object?
- What is *inheritance* and *overriding*?

The issues we are going to cover in class includes:

- How to define a class in Java? How to create an object? The starting point of a Java application?

```
public class Bicycle extends Object implements Merchandise{
  public Bicycle() {
    ...
  }

  public static void main(String args[]) {
    Bicycle myBike = new Bicycle();
    ...
  }
}
```

- What is an interface?

  In English, an interface is a device or a system that unrelated entities use to interact. According to this definition, a remote control is an interface between you and a television set, and the English language is an interface between two people. Within the Java programming language, an interface is a device that unrelated objects use to interact with each other. An interface is probably most analogous to a protocol (an agreed on behavior).

For example, a bicycle, though can run, if stocked in a store, is only required to answer questions that may be asked by an inventory program, such as "what's your unit price?". The program does not even care if the item is a bicycle or not. Actually every item sold by the store should has this capability, which can be defined as an interface:

```
public interface Merchandise {
  public double getPrice();
}
```

And each item, whatever it is, should support/implement this interface in the following way:

```
public class Bicycle extends Object implements Merchandise{
  ...
}
```

In this case, we also say those items are the instances of `Merchandise`. Generally, interfaces are useful for the following:

– Capturing similarities among unrelated classes without artificially forcing a class relationship

– Declaring methods that one or more classes are expected to implement.

– Revealing an object's programming interface without revealing its class.

3 Java Data Types, Operators, and Control Structures

3.1 Primitive Types and Reference Types

The following example illustrates some typical Java artifacts:

```
public class BasicsDemo {
    public static void main(String[] args) {
        int sum = 0;
        for (int current = 1; current <= 10; current++) {
            sum += current;
        }
        System.out.println("Sum = " + sum);
    }
}
```

The primitive data types that Java supports includes: *byte*, *short*, *int*, *long*, *double*, *float*, *boolean*, and *char*.

Note that in other languages, the format and size of primitive data types may depend on the platform on which a program is running. In contrast, the Java programming language specifies the size and format of its primitive data types. Hence, you don't have to worry about system-dependencies.

Arrays, *classes*, and *interfaces* are *reference* types. The value of a reference type variable, in contrast to that of a primitive type, is a reference to (an address of) the value or set of values represented by the variable.

A reference is called a pointer, or a memory address in other languages. The Java programming language does not support the explicit use of addresses like other languages do. You use the variable's name instead. For example,

```
int array[10];
System.out.println("Length: "+array.length);
```

Note that a variable may be defined anywhere in a statement block in Java, instead of limited to the start of a block in C.

3.2 Control Flow Statements

The following control flow statements are supported in Java: looping (while, do-while and for), decision making (if-else, switch-case), exception handling (try-catch-finally), throw branching (break, continue, label:, return).

The Java programming language provides a mechanism known as *exceptions* to help programs report and handle errors. When an error occurs, the program throws an exception. What does this mean? It means that the normal flow of the program is interrupted and that the runtime environment attempts to find an exception handler - a block of code that can handle a particular type of error. The exception handler can attempt to recover from the error or, if it determines that the error is unrecoverable, provide a gentle exit from the program. The structure of try-catch-finally statement is:

```
try {
    statement(s)
} catch (exceptiontype name) {
    statement(s)
} finally {
    statement(s)
}
```

4 Java Class Library

The following gives a list of classes scattered in different packages, which I believe are must-know and you should start with them to get gradually familiar with Java packages.

4.1 java.system

```
System
  public static final PrintStream out;
  public static final InputStream in;

Runtime:
  public Process exec(String command) throws IOException;

String
Object
```

4.2 java.util

```
Vector
Stack
```

4.3 java.io

```
File
InputStream
PrintStream
Reader
Writer
IOException
```

4.4 java.net

```
ServerSocket
  public ServerSocket(int port) throws IOException;
  public Socket accept() throws IOException;
Socket
  public Socket(InetAddress address, int port) throws IOException;
  public InputStream getInputStream() throws IOException;
  public OutputStream getOutputStream() throws IOException;
DatagramSocket
DatagramPacket
InetAddress
```

5 Exercise: Accessing the class roster

Requirement:

1. From the course website, first download `roster.zip` and uncompress the files into your working directory. `roster.zip` contains several classes files that you are supposed to use to do the work explained next.

2. By accessing the classes whose public interfaces are explained below, implement a Java program which can let you obtain your magic number.

   Registrar Capable of generating our class roster.

   ```
   public static Roster getRoster();
   ```

   Roster An interface presenting any class roster that can be queried for a specific student with his/her last name, first name, or id provided.

   ```
   public Student getStudentByLastName(String lastName);
   public Student getStudentByFirstName(String firstName);
   public Student getStudentByID(String id);
   public String toString();
   ```

   Student An interface presenting a student who has a last name, a first name, an id and a magic number.

   ```
   public String getLastName();
   public String getFirstName();
   public String getID();
   public int getMagicNumber();
   ```