CMPU4021 Distributed Systems - Labs

Introduction to Java

This is an optional introductory Java lab.

Learning Outcomes:

1. Be able to use the Java compiler
2. Be able to use the Java Runtime Environment
3. Be able to browse the Java API Documentation to find packages, classes and methods.
4. Be able to use the Java tutorial
5. Be able to create a Hello World example program
6. Be able to create a sample class hierarchy
7. Be able to create a program that uses a set of classes
8. Be able to understand the Java notion of *interfaces*.
9. Be able to package classes together
10. Be able to use the JAR tool to archive class files
11. Be able to use the JavaDoc tool.

Note 1: You may need to setup the environment for smooth running. Read `Lab_Environment_Setup.txt` how to create a BAT file that you need to run (for Windows). It sets the PATH and CLASSPATH for Java.

You may need to modify it the PATH and CLASSPATH – depending where you are running the lab tasks (i.e. in DIT labs or on your lap top).

Note 2: If you copy/paste command lines from a PDF/Word document it may not work, as invisible characters are often added. Please type it manually or copy/paste as TEXT ONLY

Tasks

1. Extract T1/HelloWorld.java and save it to a directory on your F: drive.

2. Open up a command prompt (Start -> Run -> Type "cmd"). Change to the directory where you saved the file and type the following command:
:/> javac -classpath . HelloWorld.java

The classpath argument lists the directories where all the other classes that may be used in the class are physically located. In this case we list only the current directory.

3. Execute the program by invoking the Java Runtime Environment (JRE).
:/> java -classpath . HelloWorld

The "java" command will convert the bytecode in HelloWorld.class into the machine code for your platform and will execute this machine code.

4. Extract the program T4/CommandLineArguments.java and save as before. Compile and execute the program. When executing, pass in the appropriate arguments as follows.
:/> java -classpath . CommandLineArguments 10 Words

5. Extract the classes T5/BankAccount.java, T5/CreditCardAccount.java and T5/SavingsAccount.java and store in a separate directory.
Compile these classes using one of the following commands:

:/> javac -classpath . BankAccount.java CreditCardAccount.java SavingsAccount.java

or

:/> java -classpath . *.java

Go through the code to see how Java supports the following OO methods:

- Inheritance
- Overriding
- Abstract classes and methods
- Access modifiers (public, private, protected)
- Overloading - multiple methods with the same name
- Superconstruction - calling the parent constructor from within a constructor

6. Extract T6/BankMain.java and see how Java creates and accesses objects. Change so that the deposit and withdraw methods are called - possibly by passing in values through the command line.
Compile and run BankMain.

Exceptions

7. Extract these classes from T7 (BankMain.java, BankAccount.java, CreditCardAccount.java and SavingsAccount.java) and notice the changes to the withdraw methods i.e. the addition of exception code for handling situations where problems arise. Notice how calling code must change when the method being called throws an exception.

Interfaces

8. Extract these classes from T8 (BankMain.java, BankAccount.java, CreditCardAccount.java, SavingsAccount.java and InterestGainer.java). InterestGainer is an interface rather than a class. An interface is a completely abstract class that can be implemented by classes is a manner similar to C++ multiple inheritance. Notice how when SavingsAccount implements this interface (in the class declaration) it must then implement the add interest method. Thereafter, it functions exactly the same as normal inheritance.

Packages

9. Packages are used for grouping related classes together. They are constructed by putting classes in particular directories, the names of which effect the names of the classes as a whole.

Create two folders, named src and classes. src is used to store the source code and classes is used for the compiled code.

Inside the `src` directory, create a directory named `ie`, inside which you put a directory named `dit`, inside which you put a directory named `comp`, inside which you put a directory named `bank`. These directories identify your `package`.

Put the following files in the bank directory: T9/BankAccount.java, T9/CreditCardAccount.java, T9/SavingsAccount.java and T9/InterestGainer.java

Note how the first line of each of these files identifies the package.

Return to the command prompt, and the src directory. Compile all the files with:

:/> javac -classpath ..\classes;. -d ..\classes ie/dit/comp/bank/*.java

Go one level up from the src directory and save T9/BankMain.java. This file imports all the classes in the package (see the first line).

The -d switch identifies the directory where the compiled code should be put (until now it was put in the same directory).

Compile this with

:/> javac -classpath classes;. BankMain.java

Run it with

:/> java -classpath classes;. BankMain

Note that the classpath must list the directories that store class files or packages.

JAR

10. A JAR (Java Archive) stores a class hierarchy inside a file, like a zip file. In this task you will store the ie.dit.comp.bank package in an archive. This would usually be used to distribute the classes throughout the network or internet.

Go to the classes directory at the command prompt. Type the following command:

:/> jar cvf bank_classes.jar ie/dit/comp/bank/*.class

This will create a file named bank_classes.jar which you can include in your classpath instead of the location of the package.

Go one directory layer up from classes, where BankMain is, and type the following command:

:/> java -classpath classes/bank_classes.jar;. BankMain

11. JavaDoc
At the same level as src and classes create a new directory named docs.

Go to the src directory and type the following command:

:/> javadoc -d ..\docs ie/dit/comp/bank/*.java

In Windows Explorer, double click on the file named index.html in the docs directory. You will see that automatic documentation was created using the special comments you put in your code.

Java API

12. Develop a set of classes and a program for storing information about students. Follow all the above steps again, to create a package inside a JAR which can then be deployed.