Introduction to Java

Student Exercises

Introduction

Instructions for all the student exercises are given below. It is assumed that you will be using Sun's JDK throughout from a NT command window (cmd.exe). If you wish to use an IDE you will need to make adjustments accordingly.

The exercises should be installed and run under the student directory

```
C:\Student
```

To run the Java Compiler and Java Virtual Machine you should use the batch file provided in

```
C:\Student\Commands\Dos Session.cmd
```

This should be launched from Windows Explorer.

Ch01.Hello

This is a simple exercises to get you started with your first java program. All you have to do is to print the *Hello* on the console.
Create your source file in the directory

```
C:\Student\com\ssl\ch01
```

and call it

```
Hello.java
```

At the start of your source file include the package statement

```
package com.ssl.ch01;
```

We will learn about packages later in the course.

Start up a console session (using the batch file provided), *cd* to the student directory and run the Java compiler.

```
cd c:\Student
javac com\ssl\ch01\Hello.java
```

Correct any mistakes and then execute the program

```
java com.ssl.ch01.Hello
```

Ch01.Args

Now try experimenting with passing parameters on the command line. Write a program the prints out the first 2 parameters typed on the command line. You can pick these up in your program using

```
args[0]
args[1]
```

Run your program and pass your first and last name as parameters. What happens if you forget to supply the command line arguments?

Ch02.Person

In this exercise you have to write a class called

```
class Person
```

that has two instance variables

```
private String name
private int age
```

and two methods
```
public constructor
public void Display()
```

The constructor should be written to initialise the two instance parameters (it takes two arguments) and the *Display()* method should simply print the object's name and age to the console.

Create a test harness in the file
```
C:\Student\com\ssl\ch02\TestPerson.java
```

In this file create two classes
```
public class TestPerson
class Person
```

Set up the package name to be
```
package com.ssl.ch02;
```

In the *main* method of the *TestPerson* class create two *Person* objects and call their *Display()* methods. Build and run as before
```
cd c:\Student
javac com\ssl\ch02\TestPerson.java
java com.ssl.ch02.TestPerson
```

Ch02.Point

This next exercises is similar, but you will write a few more methods for your class. Create a test harness in the file
```
C:\Student\com\ssl\ch02\TestPoint.java
```

with the package name
```
package com.ssl.ch02
```

Your class is called
```
class Point
```

with two instance variables
```
private int x
private int y
```

two constructors
```
public Point()
public Point(int x0, int y0)
```

and two methods
```
public void WhereAmI()
public void MoveBy(int dx, int dy)
```

In the default constructor initialise the Point object to co-ordinates (0, 0). In the other constructor initialise the Point object to co-ordinates (x0, y0). The *WhereAmI()* method should display the Points co-ordinates and the *MoveBy()* method should change the internal co-ordinates.

In the test harness create several Point objects using both constructors. After all the Points have been initialised use *WhereAmI()* to display their locations. Then use *MoveBy()* on all the Points and display their locations again. Compile and run from
```
cd c:\Student
```

Ch03.Point

In this example you will be repeating the code from the previous exercise, but this time splitting up the source files and making use of packages. Copy the *TestPoint.java* file to the directory
```
C:\Student\com\ssl\ch03\Point
```

and separate out the test code and the *Point* class into the different files

```
C:\Student\com\ssl\ch03\Point\Point.java
C:\Student\com\ssl\ch03\Point\TestPoint.java
```

Now remove the test code from *Point.java* and the Point class from *TestPoint.java*. Change the package name to reflect the new directory structure. Compile and run

```
javac com\ssl\ch03\Point\Point.java
javac com\ssl\ch03\Point\TestPoint.java
java com.ssl.ch03.Point.TestPoint
```

Check out the classpath using

```
set classpath
```

This classpath was set by the Dos Session.cmd utility.

Ch03.Jar Utility

In this exercise you will experiment with jar files. You can reuse the code from the previous exercise.
When you run the JVM any classes that you use must be included on the classpath. However it is perfectly acceptable to place your class files in a jar file and include this jar file on the class path instead.

Run the jar command to create a jar file using the command

```
jar -cvf Point.jar com\ssl\ch03\Point\*.class
```

This should create the jar file

```
C:\Student\Point.jar
```

Check out its contents by running the WinZip command or jar with the -t option. You should see you class files zipped up and a manifest (acts like an index). Once your class files are safely inside the jar file you can should the original class files (not the source files!).

```
del com\ssl\ch03\Point\*.class
```

and try running your program again

```
java com.ssl.ch03.Point.TestPoint
```

Your program should fail because the jar file is not included in the classpath. Fix the problem with

```
set CLASSPATH=%CLASSPATH%;C:\Student\Point.jar
```

and try again. It should work this time.

Ch04.TestOverloading

In this example you will be working with a single file that contains the test harness and your class

```
C:\Student\com\ssl\ch04\TestOverloading.java
```

What you have to do is write a class called

```
class Clock
```

with instance variables

```
private int hours
private int minutes
private int seconds
```

that has overloaded constructors and overloaded methods. If you study the test harness you will be able to work out what is required.

```
public class TestOverloading {
   static void main(String[] args) {
      Clock t1, t2, t3, t4;
      t1 = new Clock(6, 0, 15);
      t2 = new Clock(7, 30);
      t3 = new Clock(7, 58, 30);
      t4 = new Clock();

      t1.backward(15);
```

```
      t2.backward(0, 30, 0);
      t3.forward(90);
      t4.forward(9, 0, 0);

      t1.WhatIsTheTime();
      t2.WhatIsTheTime();
      t3.WhatIsTheTime();
      t4.WhatIsTheTime();
   }
}
```

In the calls to *forward()* and *backward()* if a single parameter is passed the parameter is in seconds, but if three parameters are passed then the parameters denote *hours*, *minutes* and *seconds* respectively.

Implement the Clock class in the same file as the test harness. Build and run from

```
cd c:\Student
```

You should get the following output

```
The time is: 6:0:0
The time is: 7:0:0
The time is: 8:0:0
The time is: 9:0:0
```

Ch04.TestObjectRefs

This exercise simulates a horse race run over 5 furlongs. Work entirely in the file

```
C:\Student\com\ssl\ch04\TestObjectRefs.java
```

Your job is to write a class called

```
class Race
```

with instance variables

```
private Horse horse1
private Horse horse2
private Horse horse3
```

and a class

```
class Horse
```

with instance variables

```
private String name
private int distance
```

Study the test harness to work out which methods need implementing in each class

```
public class TestObjectRefs {
   static void main(String[] args) {
      Horse theNag     = new Horse("The Nag");
      Horse lightening = new Horse("Lightening");
      Horse thunder    = new Horse("Thunder");
      Race theRace     = new Race(theNag, lightening, thunder);

      lightening.move();
      thunder.move();
      lightening.move();
      theNag.move();
      lightening.move();
      thunder.move();
      thunder.move();
      lightening.move();
      theNag.move();
      thunder.move();
      lightening.move();

      theRace.PrintResult();
   }
}
```

Implement your classes. Build and run from

```
cd c:\Student
```

You should get the following output

```
The Nag at 2 furlongs
Lightening at 5 furlongs
Thunder at 4 furlongs
```

Ch04.TestStatics

This next exercise is to give you some experience with static methods and static class variables. Study the test harness

```
public class TestStatics {
   static void main(String[] args) {
      Phone dicksPhone = new Phone("Dick");
      Phone johnsPhone = new Phone("John");
      Phone marysPhone = new Phone("Mary");
      dicksPhone.call( 3, 15);
      johnsPhone.call( 2, 10);
      dicksPhone.call(15, 30);
      marysPhone.call(10,  0);
      dicksPhone.call( 3, 15);
      johnsPhone.call( 1,  0);
      marysPhone.call( 2, 15);
      marysPhone.call(12, 15);
      dicksPhone.call( 9,  0);
      johnsPhone.call(30, 45);
      dicksPhone.call(17, 15);

      dicksPhone.PrintTotals();
      johnsPhone.PrintTotals();
      marysPhone.PrintTotals();

      Phone.HowManyCalls();
   }
}
```

Work entirely in the file

```
C:\Student\com\ssl\ch04\TestStatics.java
```

Your job is to write a class called

```
class Phone
```

with instance variables

```
private int minutes;
private int seconds;
private String name;
```

Decide what static variables, static methods and normal methods are required. Implement your class. Build and run from

```
cd c:\Student
```

You should get the following output

```
Dick has used 17:15
John has used 30:55
Mary has used 12:30
There have been 11 calls
```

Ch05.TestArrays

This next exercise is to give you some experience with arrays. Study the test harness

```
public class TestArrays {
   static void main(String[] args) {

      MyArray a = new MyArray(2, 4, 8, 16, 32);
      a.Print();

      a.Double();
      a.Print();

      a.Reverse();
      a.Print();

      MyArray b = new MyArray();
      b.CopyFrom(a);
      a.Clear();
      a.Print();
      b.Print();
   }
}
```

Work entirely in the file

```
C:\Student\com\ssl\ch05\TestArrays.java
```

Your job is to write a class called

```
class MyArray
```

with instance variables

```
final int size = 5;
private int[] array;
```

Decide what methods are required and then implement your class. Build and run from

```
cd c:\Student
```

You should get the following output

```
2 4 8 16 32
4 8 16 32 64
64 32 16 8 4
0 0 0 0 0
64 32 16 8 4
```

Ch05.TestStrings

This next exercise is to give you some experience with strings. Study the test harness

```
public class TestStrings {
   static void main(String[] args) {
   String[] plays = { "Hamlet", "Othello", "Macbeth", "Cymbeline", "Coriolanus" };
   Play.Print(plays);

   Play.ToLower(plays);
   Play.Print(plays);

   Play.ToUpper(plays);
   Play.Print(plays);

   Play.PrintFirst(plays);
   Play.PrintLast(plays);

   String all = Play.Concatenate(plays);
   System.out.println(all);
   }
}
```

Work entirely in the file

```
C:\Student\com\ssl\ch05\TestStrings.java
```

Your job is to write a class called

```
class Play
```

This class has no instance variables nor standard methods, only static methods.

```
static void ToUpper(String[] array);           // convert everything to uppercase
static void ToLower(String[] array);           // convert everything to lowercase
static void PrintFirst(String[] array);        // print first string (lexically)
static void PrintLast(String[] array);         // print last string (lexically)
static void Print(String[] array);             // print entire array
static String Concatenate(String[] array);     // concatenate array
```

Implement all the static methods. Build and run from

```
cd c:\Student
```

You should get the following output

```
Hamlet Othello Macbeth Cymbeline Coriolanus
hamlet othello macbeth cymbeline coriolanus
HAMLET OTHELLO MACBETH CYMBELINE CORIOLANUS
First string is: CORIOLANUS
Last string is: OTHELLO
HAMLETOTHELLOMACBETHCYMBELINECORIOLANUS
```

Ch06.Marriage

Complete the project discussed in the notes about getting married. Study the code in

```
C:\Student\com\ssl\ch06\TestMarriage.java
```

Your job is to fill in all the missing code fragments denoted by TODO labels.

The test harness creates 4 objects and then marries them off. At the end of the test harness we print out the names of various partners. You should get the following output

```
Carol
Ted
Bob
Alice
```

Ch06.GrandPrix

This next exercise is to give you some more experience with associative relationships. Study the code in

```
C:\Student\com\ssl\ch06\TestGrandPrix.java
```

Your job is to fill in all the missing code fragments denoted by TODO labels.

The test harness creates several Driver and Manufacturer objects and then sets up the teams. The make up of the teams is displayed as we progress through the test harness. You should get the following output

```
Ferrari's team is: Michael Schumacher and Eddie Irvine
Williams's team is: Ralph Schumacher and Jenson Button
Ferrari's team is: Michael Schumacher and Jenson Button
Williams's team is: Ralph Schumacher and Eddie Irvine
```

Ch07.Q1.Cars1

This next exercise is to give you experience using inheritance. Study the code in

```
C:\Student\com\ssl\ch07\TestCars1.java
```

Your job is to fill in all the missing code fragments denoted by TODO labels.

You should get the following output

```
Ford 1500cc
Morgan 2000cc 120 tops
Ferrari 6500cc 220 tops Champions
```

Ch07.Q2.Cars2

This is an extension to the previous exercise. Make a copy the previous source file

```
cd C:\Student\com\ssl\ch07
copy TestCars1.java TestCars2.java
```

Remove the lines

```
System.out.println(black.Details());
System.out.println(green.Details());
System.out.println(red.Details());
```

and replace them with a call to a **static** method called *Print()* in the *TestCars2* class that prints the same information

```
public class TestCars2 {
   static void main(String[] args) {
      Car black = new Car("Ford", 1500);
      Car green = new SportsCar("Morgan", 2000, 120);
      Car red = new RacingCar("Ferrari", 6500, 220, "Champions");
      Print(black);
      Print(green);
      Print(red);
   }
   ...
```

What parameter will *Print()* take and why?

Ch07.Q3.Arrays

This next exercise is to give you experience using polymorphic collections. Study the code in

```
C:\Student\com\ssl\ch07\Arrays.java
```

You need to write 3 classes (*Circle*, *Rectangle* and *Polygon*) that are subclasses of *Shape* and fill in the static method *Print()* in the test harness. Notice that the array contains references to different kinds of *Shape*. The array is an example of a polymorphic collection.

Your job is to fill in all the missing code fragments denoted by TODO labels.

You should get the following output

```
x=0,y=0,r=50
x=1,y=1,h=20,w=25
x=2,y=2,sides=5
x=3,y=3,r=50
x=4,y=4,r=80
x=5,y=5,sides=12
x=6,y=6,r=35
x=7,y=7,h=78,w=51
```

Ch08.Q1.FileCopy

This exercise is to give you experience using exception handling and to practice file I/O. Study the code in

```
C:\Student\com\ssl\ch08\Q1\TestFileCopy.java
```

The idea is to take an example source file and make a copy of it. The names of the files will be specified on the command line. You'll find a test file that you can copy called

```
C:\Student\com\ssl\ch08\Q1\Test.txt
```

When you are attempting to copy files, things can often go wrong. You might not be able to find the source file or perhaps the target file has read only access. To make your program bullet proof you need exception handling.

Fill in all the missing code fragments denoted by TODO labels and then test your work. In this exercise you need to generate some errors for the exception handling to kick in. So try the following tests

Test 1: Copy occurs

```
java com.ssl.ch08.Q1.TestFileCopy Test.txt Test2.txt
```

Test 2: Copy fails because you forgot to pass the command line parameters (*ArrayIndexOutOfBoundsException*)
```
java com.ssl.ch08.Q1.TestFileCopy
```

Test 3: Copy fails because the target file is read only (*FileNotFoundException*)
```
attrib +r com\ssl\ch08\Q1\Test2.txt
java com.ssl.ch08.Q1.TestFileCopy Test.txt Test2.txt
```

Note: You can compare two files using the command
```
comp com\ssl\ch08\Q1\Test.txt com\ssl\ch08\Q1\Test2.txt
```

Ch08.Q2.Team

This exercise is to give you experience at throwing and catching exceptions. Study the code in
```
C:\Student\com\ssl\ch08\Q2\TestTeam.java
```

Fill in all the missing code fragments denoted by TODO labels and then test your work.
You should get the following output
```
Error: Team is full
Team is:
       Paul Scholes
       Ryan Giggs
       Dwight Yorke
```

Ch09.Q1.Shapes

This exercise is to give you experience at using polymorphism. Study the code in
```
C:\Student\com\ssl\ch09\Q1\TestShapes.java
```

Fill in all the missing code fragments denoted by TODO labels and then test your work.
You should get the following output
```
Area is: 314.159
Area is: 1256.636
Area is: 225.0
Area is: 375.0
Area is: 1963.49375
Area is: 981.746875
```

Ch10.Q1.Polymorphism

This exercise is to give you experience at using collections. You will be working with the file
```
C:\Student\com\ssl\ch10\Q1\TestPolymorphism.java
```

The idea is to take your solution from *Ch09.Q1.Shapes* and update it to use a collection class instead of an array. You could use any collection class, but we suggest that you use the *ArrayList*. During the conversion you will need to use iterators to access the elements of the collection.

Build and test your work. You should get the same output as before.

Ch10.Q2.Squash

This exercise is to give you experience at using collections. You will be working with the file
```
C:\Student\com\ssl\ch10\Q1\TestSquash.java
```

The idea in this example is to use an *ArrayList* to keep track of the players involved in a squash league. As usual you have to fill in all the missing code fragments denoted by TODO labels and then test your work.
You should get the following output
```
John has a score of 8
Peter has a score of 20
Larry has a score of 16
Tom has a score of 16
```

Ch11.Q1.SimpleApplet

This exercise is to give you experience at creating simple applets and passing parameters from your HTML file. You will be working with the files

```
C:\Student\com\ssl\ch11\Q1\SimpleApplet.html
C:\Student\com\ssl\ch11\Q1\SimpleApplet.java
```

Study the code in the java source file and in the HTML file. You have to fill in all the missing code fragments denoted by TODO labels. Check out the *PARAM* tag in the HTML file to see what gets passed to your applet.

Use the AppletViewer to test your applet

```
javac com\ssl\ch11\Q1\SimpleApplet.java
appletviewer com\ssl\ch11\Q1\SimpleApplet.html
```

Ch11.Q2.PieApplet

This exercise is to give you more experience at creating applets. You will be working with the files

```
C:\Student\com\ssl\ch11\Q2\PieApplet.html
C:\Student\com\ssl\ch11\Q2\PieApplet.java
```

Study the code in the java source file. Your job is to draw a Pie at a visible position within the applet (e.g. x=150, y =100). Use the random number generator in the *java.lang.Math* package to change the colour of the Pie every time the page containing the applet becomes active (i.e. in the applet's *start()* method). Don't forget to call *repaint()* each time you change the colour of the Pie.

Use the AppletViewer to test your applet

```
javac com\ssl\ch11\Q2\PieApplet.java
appletviewer com\ssl\ch11\Q2\PieApplet.html
```

Ch11.Q3.Rabbit

Now for some audio and images. This example is based almost entirely on the code given in the notes.

Create an applet from scratch in the file

```
C:\Student\com\ssl\ch11\Q3\Rabbit.java
```

by copying and pasting one of the other applets you have just written. Change it such that it loads the image

```
rabbit.gif
```

and play the audio clip

```
cuckoo.au
```

The code to do this is given in the notes. The only thing you'll need to change is the URL of the 2 files. These files are located in the same directory as the java source file and HTML file. So change the call to *getDocumentBase()* to use the URL defined by

```
URL thisDirectory = new URL("file:///.");
```

Set up an HTML file in the same directory that will start the applet

```
C:\Student\com\ssl\ch11\Q3\Rabbit.html
```

by copying and pasting one of the other HTML files you have just written. Change it so that it refers to the *Rabbit* applet.

Use the AppletViewer to test your applet

```
javac com\ssl\ch11\Q3\Rabbit.java
appletviewer com\ssl\ch11\Q3\Rabbit.html
```

Experiment with other images and audio files, but don't annoy your neighbour too much!

Ch12.Q1.Pie

Now for some animation in an applet. You will be working with the files

```
C:\Student\com\ssl\ch12\Q1\TestPie.html
C:\Student\com\ssl\ch12\Q1\TestPie.java
```

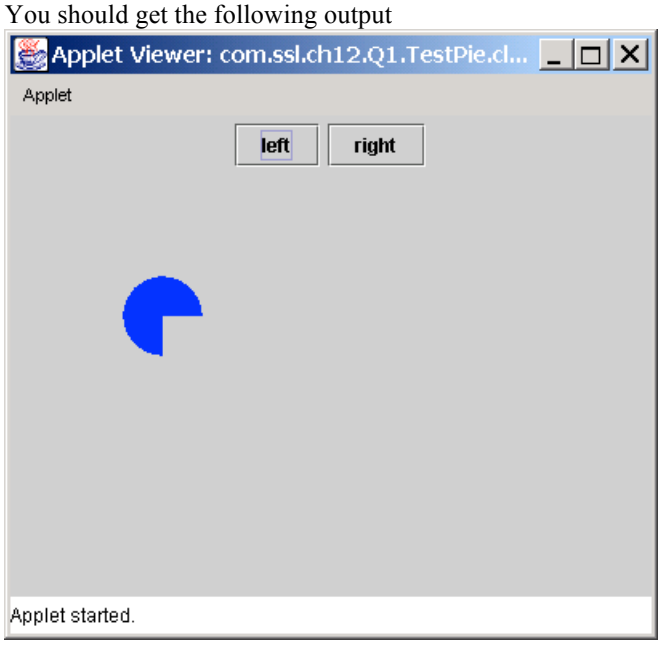
Study the code in the java source file. Your job is to add two buttons to the applet to move a Pie left and right. Start with the Pie at a visible position within the applet (e.g. x=150, y =100) and move it 10 pixels to the left each time the left button is pressed and move it 10 pixels to the right each time the right button is pressed.

The only tricky bit is too call the superclass *paint()* method in your applet's own *paint()* method. This enables the buttons to paint themselves.

There are the usual TODO comments in the source file. Use the AppletViewer to test your applet

```
javac com\ssl\ch12\Q1\TestPie.java
appletviewer com\ssl\ch12\Q1\TestPie.html
```

You should get the following output



Ch13.Q1.Gridbag

This exercise is to give you experience at using Layout Managers. You will be working with the files

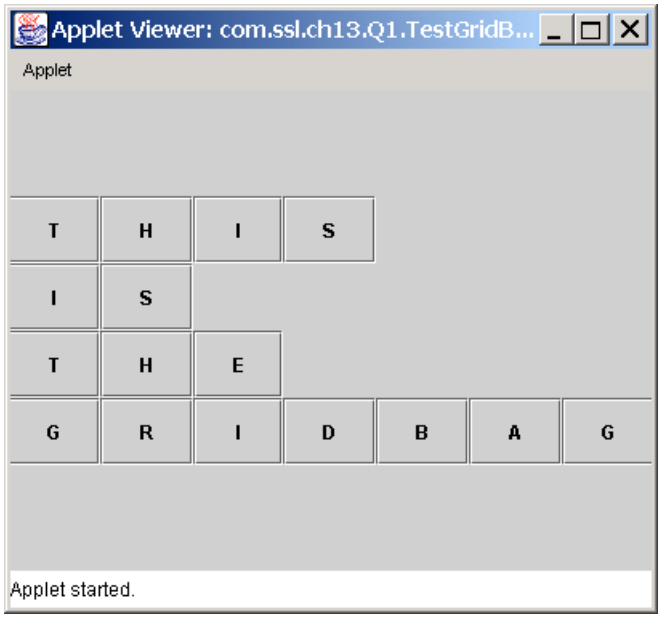
```
C:\Student\com\ssl\ch13\Q1\TestGridBag.html
C:\Student\com\ssl\ch13\Q1\TestGridBag.java
```

You will be working with the GridBagLayout. Take a look in the Java SDK for details of this class. Then study the code in the java source file and then fill in all the missing code fragments denoted by TODO labels.

There are the usual TODO comments in the source file. Use the AppletViewer to test your applet

```
javac com\ssl\ch13\Q1\TestGridBag.java
appletviewer com\ssl\ch13\Q1\TestGridBag.html
```

You should get the following output



Ch14.Q1.PieApplet

This exercise is to give you experience at threads. You will be working with the files

```
C:\Student\com\ssl\ch14\Q1\PieApplet.html
C:\Student\com\ssl\ch14\Q1\PieApplet.java
```

Your job is to draw a Pie that moves around the screen describing a square. To make thing interesting try changing the colour of the Pie as it moves. To get an idea of what is required run the example solution

```
javac com\ssl\Solutions\ch14\Q1\PieApplet.java
appletviewer com\ssl\Solutions\ch14\Q1\PieApplet.html
```

There are the usual TODO comments in the source file. Use the AppletViewer to test your applet

```
javac com\ssl\ch14\Q1\PieApplet.java
appletviewer com\ssl\ch14\Q1\PieApplet.html
```