Internet Programming
300130

WORKSHOP-1

(5 Marks, due by Week 4)

EXERCISE 1.1 (2 marks)

Class, Object and Instance variable

1) Write a Java program that includes

i) A **Vehicle** class containing four **private** fields:

id (a unique integer number, used to identify the vehicle)

nextId (**static**, used for the sequence number)

colour (used to describe the color of a vehicle)

type (used to describe the type of a vehicle, e.g. car, van, or truck)

and a constructor (with no arguments) that assigns **nextId** to **id** and then increments the **nextId**. You may add more methods or variables when it is necessary. For example, you may add methods to get id, to set/get the colour or type.

ii) A **VehicleTest** class containing a **main** method and creating some objects, such as cars and trucks, in the **VehicleTest** class.

iii) Set and print the information of the objects of **Vehicle** in the **VehicleTest** class. The output may look like:

```
Vehicle:  1,  Colour:  Red,  Type:  Car
Vehicle:  2,  Colour:  Blue, Type:  truck
```

2) Modify the code in the previous question by writing an additional constructor in such a way that a **Vehicle** object with color and type can be constructed directly via, say,

```
Vehicle("Green", "Van");
```

3) The sample code is provided.

EXERCISE 1.2 (3 marks)

Overloading vs Overriding

Examine the following piece of code **test.java**, and write out the output of the program when executed. Answer the following questions.

```
// filename: test.java
class A {
    int i=0;
    private void f(){ i=1; };
    void g(){ i=11; };
    A() { f(); }
    A(boolean x) { g(); }
}

class B extends A {
    void f(){ i=9; };
    public void g(){ i=99; };
    public int h() { return i;}
    B() { super(); }
    B(boolean x) { super(x); }

    public static void main(String[] args) {
      System.out.println( new B().h() );
      System.out.println( new B(true).h() );
    }
}
```

1) What new file(s) would be generated if you compile **`test.java`**?

```
javac test.java
```

2) What are the constructors of class **`B`**? Are they overloaded or overridden?

3) Run the program by

```
java B
```

What results do you get? What happens if you try to run

```
java A
```

Can you explain the result?

4) If `g( )` of **`A`** is defined as private method, what results the program will produce? Can you explain why?

5) If you place the keyword **`public`** right in front of the definition of the class **`B`** so that you have

```
public class B extends A {
...

}
```

can you still compile it by executing **javac test.java**? If not, explain why.

EXERCISE 1.3 (not to be assessed)

Java Basics

1) The following code was saved as Vehicle.java.

```
public class Vehicle {
    System.out.println("This is a testing class");
}
```

Please compile and then run the program. If the compilation is not successful, please explain the failure and fix the code. After fixing the code, you delete the type **public** in the class and save the code as **Vehi.java**. Please compile and run the code, and explain what has happened.

2) Explain the meaning of **a.next.value** in the code:

```
public class Node {
    public int value;
    public Node next;
    public Node(int i) { value=i; }
}

class NodeList {
   public static void main(String [] args) {
      Node a=new Node(1), b=new Node(2), c=new Node(3);
      a.next=b;
      b.next=c;
      System.out.println(a.next.value);
```

```
    }
}
```

What object does it refer to?

3) Recall that command line parameters are passed to

```
public static void main(String[] args)
```

in **args**. The number of parameters is given by **args.length**. For instance the following Java program displays all the command line parameters.

```
class Test2 {
  public static void main(String[] args) {
     for(int i=0; i<args.length; i++)
       System.out.println("args["+i+"]="+args[i]);
  }
}
```

If you compile the program by

```
javac Test2.java
```

and run it by

```
java Test2 para1 para2 para3
```

then the output is simply

```
args[0]=para1
args[1]=para2
args[2]=para3
```

4) Parameters passed from command line are of **String** type. To convert from a string **s** to an integer *i*, just do

```
i= Integer.parseInt(s);
```

5) Write a Java program **ShowStars.java** so that

```
java ShowStars numA1 numB1 ....
```

display a sequence of stars "*", starting from position *numA1* and ending at position *numB1*.

i. If there are further such integer pairs, say, *numA2* and *numB2*, do the same for the next line. For example,

```
java ShowStars 3 3 2 4 1 5
```

gives

```
  *
 ***
*****
```

ii. For convenience we assume all *numA1*, *numB1* etc are positive integers.

iii. If the number of command line parameters is an odd number, then ask users to input a number to form an additional pair.