WIA1002/WIB1002 Data Structure

Tutorial 1: Programming Fundamentals (Revision)
Sample Answer

Instruction: Bring your solutions for all the questions below to your tutorial class. You might be asked to present your solutions to the class.

1. Write the definition of a class *Telephone* that contains:
   - An instance variable *areaCode*
   - An instance variable *number*
   - A static variable *numberOfTelephoneObject* that keeps track of the number of *Telephone* objects created
   - A constructor that accepts two arguments used to initialize the two instance variables
   - The accessor and mutator methods for *areaCode* and *number*
   - A method *makeFullNumber* that does not accept any argument, concatenates areaCode and number with a dash in between, and returns the resultant *String*.

   Write the statements to:
   - Instantiate 5 *Telephone* objects and store them in an array. Iterate through the array to print the full number of the 5 *Telephone* objects on the console. Your output should look as below:

```
03-79676300
03-79676301
03-79676302
03-79676303
03-79676304
```

Sample Answer:

```
public class Telephone {
    private String areaCode;
    private int number;
    private static int numberOfTelephoneObject = 0;

    Telephone(String areaCode, int number){
        this.areaCode = areaCode;
        this.number = number;
        numberOfTelephoneObject++;
    }
    public void setAreaCode(String areaCode)
    {
            this.areaCode = areaCode;
    }
    public void setNumber(int number)
```

```
            {
                    this.number = number;
            }

            public String getAreaCode()
            {
                return areaCode;
            }

            public int getNumber()
            {
                return number;
            }

            public String makeFullNumber() {
                return areaCode + "-" + number;
            }

            /**
             * @param args the command line arguments
             */
            public static void main(String[] args) {
                // TODO code application logic here
                Telephone[] phoneArray= new Telephone[5];
                int number = 79676300;
                for (int i = 0; i < 5; i++) {
                    phoneArray[i] = new Telephone("03", number++);
                }
                for (int i = 0; i < numberOfTelephoneObject; i++) {
                     System.out.println(phoneArray[i].makeFullNumber());
                }
            }
 }
```

2. What is the output for the following? Explain.

```
class Person {
    public Person() {
        System.out.println("(1) Performs Person's tasks");
    }
}
class Employee extends Person {
    public Employee() {
        this("(2) Invoke Employee's overloaded constructor");
        System.out.println("(3) Performs Employee's tasks ");
    }
    public Employee(String s) {
        System.out.println(s);
    }
}
public class Faculty extends Employee {
    public Faculty() {
        System.out.println("(4) Performs Faculty's tasks");
    }
    public static void main(String[] args) {
        new Faculty();
    }
}
```

Sample Answer:

(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks

In any case, constructing an instance of a class invokes the constructors of all the superclasses along the inheritance chain. When constructing an object of a subclass, the subclass constructor first invokes its superclass constructor before performing its own tasks. If the superclass is derived from another class, the superclass constructor invokes its parent-class constructor before performing its own tasks. This process continues until the last constructor along the inheritance hierarchy is called. This is called *constructor chaining*.

3. What is the output for the following? Explain.

```
public class C {
   public static void main(String[] args) {
    Object[] o = {new A(), new B()};
    System.out.print(o[0]);
    System.out.print(o[1]);
  }
}

 class A extends B {
    public String toString() {
        return "A";
     }
}

 class B {
     public String toString() {
        return "B";
    }
}
```

a. AB
b. BA
c. AA
d. BB

Sample Answer: AB.

A variable of reference type is a polymorphic variable (o[0] and o[1] in the code above), since its dynamic type can differ from its static type and change during execution. The method definition of the dynamic type will be executed during runtime.

4. Write a class definition for an abstract class, *Vehicle*, that contains:
   - a double instance variable, *maxSpeed*
   - a protected double instance variable, *currentSpeed*
   - a constructor accepting a double used to initialize the *maxSpeed* instance variable
   - an abstract method, accelerate, that accepts no parameters and returns nothing.
   - a method *getCurrentSpeed* that returns the value of *currentSpeed*
   - a method *getMaxSpeed* that returns the value of *maxSpeed*
   - a method *pedalToTheMetal*, that repeatedly calls accelerate until the speed of the vehicle is equal to *maxSpeed*. *pedalToTheMetal* returns nothing.

   Can you create an instance of *Vehicle?*

Sample Answer:

```
public abstract class Vehicle {
    private double maxSpeed;
    protected double currentSpeed;

    Vehicle(double maxSpeed) {
        this.maxSpeed = maxSpeed;
    }

    public abstract void accelerate();

    public double getCurrentSpeed() {
        return currentSpeed;
    }

    public double getMaxSpeed() {
        return maxSpeed;
    }

    public void pedalToTheMetal() {
      while(currentSpeed < maxSpeed)
         accelerate();
    }
}
```

No

5. Assume the existence of an interface, *Account*, with the following methods:
   - *deposit*: accepts an integer parameter and returns an integer
   - *withdraw*: accepts an integer parameter and return a boolean

   Define a class, *BankAccount*, that implements the above interface and has the following members:
   - an instance variable named *balance*
   - a constructor that accepts an integer that is used to initialize the instance variable

- an implementation of the *deposit* method that adds its parameter to the *balance* variable. The new balance is returned as the value of the method.
- an implementation of the *withdraw* method that checks whether its parameter is less than or equal to the *balance* and if so, decreases the *balance* by the value of the parameter and returns *true*; otherwise, it leaves the *balance* unchanged and returns *false*.

Sample Answer:

```
interface Account {
   public abstract int deposit(int depositAmount);
   public abstract boolean withdraw(int withdrawAmount);
}

public class BankAccount implements Account {
     private int balance;

     BankAccount(int balance) {
         this.balance = balance;
     }

     public int deposit(int depositAmount) {
         return balance = balance + depositAmount;
     }

     public boolean withdraw(int widthdrawAmount) {
         if(widthdrawAmount < balance) {
             balance = balance - widthdrawAmount;
             return true;
         }
         else
             return false;
     }
}
```