WIA1002/WIB1002 Data Structure

Tutorial 1: Programming Fundamentals (Revision)

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

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

   public Telephone(String areaCode, String number){
      this.areaCode = areaCode;
      this.number = number;
      numberOfTelephoneObject++;
   }

   // Accessor
   public String getAreaCode(){
      return areaCode;
   }

   public String getNumber(){
      return number;
   }

   // Mutator
   public void setAreaCode(String areaCode){
      this.areaCode = areaCode;
   }

   public void setNumber(String number){
      this.number = number;
   }

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

   public static void main(String[] args) {
      Telephone[] telephones = {new Telephone("03","79676300"), new Telephone("03",
"79676301"),  new Telephone("03", "79676302"), new Telephone("03", "79676303"), new
Telephone("03","79676304")};

      for (Telephone t: telephones) {
         System.out.println(t.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();
    }
}
```

The output of the following program is

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

This is because the constructors are executed in the order of the inheritance hierarchy where Person → Employee → Faculty. In this case, the Faculty class which is a derived class of Employee is instantiated in the main method. The class Employee is derived from Person which triggers Person no-argument constructor that outputs "(1) Performs Person's tasks". After that, Employee will execute the no-argument constructor which calls the overloaded argument constructor pass down with the String parameter that prints "(2) Invoke Employee's overloaded constructor". before printing "(3) Performs Employee's tasks ". Lastly, the program will execute Faculty constructor and outputs "(4) Performs Faculty's tasks".

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

The output for the following program is **a. AB**. This is because the derived class A overrides the base class B so the variable o[0] which is an instance of A uses A's toString() method which returns "A". On the other hand, o[1] which is an instance of B uses B's toString() method directly since there's no overriding involved for the base class itself which later returns "B".

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?*

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

   public 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();
         if (currentSpeed > maxSpeed) {
            currentSpeed = maxSpeed; // limits to maxSpeed only
         }
      }
   }
}
```

An instance of an abstract class like Vehicle cannot be instantiated directly. To access an abstract class, it must be inherited from another class. Furthermore, the abstract class may contain abstract methods without complete definitions that serve as a placeholder to be overridden by a subclass. Thus, creating and instantiating a subclass that extends to the abstract class by implementing it with complete structure and definitions can access the functionality of both abstract class and subclass.

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*.

```
// Assume the existence of an interface
interface Account {
   public int deposit(int amount);
   public boolean withdraw(int amount);
}

public class BankAccount implements Account{
   private int balance;

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

   public int deposit(int amount){
      this.balance += amount;
      return balance;
   }

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