BSCCS2005: Graded Assignment with Solutions
Week 4

1. Consider the code given below. [MCQ:2 points]

```
public interface Printable{
    public default void print() {
        System.out.println("not implemented");
    }
}
public abstract class Collection{
    public void print() {
        System.out.println("no element");
    }
}
public class Queue extends Collection implements Printable{
    public void print() {
        super.print();
        System.out.println("print the queue");
    }
}
public class FClass{
    public static void main(String[] args) {
        Queue q = new Queue();
        q.print();
    }
}
```

What will be the output?

- ◯
  ```
  no element
  not implemented
  print the queue
  ```
- ◯
  ```
  not implemented
  print the queue
  ```
- ✓
  ```
  no element
  print the queue
  ```
- ◯ It generates compiler error

Solution: Whenever there is a conflict between a default method of an interface and a method in a class to be inherited, method inherited from the class gets the priority.

2. Consider the code given below. [MCQ:2 points]

```
public interface Shape{
    public double area();
    public default double volume() {
        return -1.0;
    }
}

public interface Printable{
    public default void print() {
        System.out.println("not implemented");
    }
}

public class Rectangle implements Shape, Printable{
    private double w, h;
    public Rectangle(double w_, double h_) {
        w = w_;
        h = h_;
    }
    public double area() {
        return w * h;
    }
    public void print() {
        System.out.print(area() + " ");
        System.out.print(volume());
    }
}

public class FClass{
    public static void main(String[] args) {
        Shape s = new Rectangle(20.0, 50.0);
        s.print();
    }
}
```

What will be the output?

- ◯ `1000.0` followed by a runtime error
- ◯ `not implemented`
- ◯ `1000.0 -1.0`
- [✓ It generates a compiler error]

Solution: Since the static type of `s` is `Shape`, and `print()` method does not belong to `Shape`, the call `s.print()` is illegal.

3. Consider the code given below. [MSQ:2 points]

```
public interface NumberType{
    public double norm();
    public static void print(double x) {
        System.out.println("norm is : " + x);
    }
}

public class ComplexNum implements NumberType{
    private int r, i;
    public ComplexNum(int r_, int i_) {
        r = r_;
        i = i_;
    }
    public double norm() {
        return Math.sqrt(r*r + i*i);
    }
}

public class FClass{
    public static void main(String[] args) {
        ComplexNum c = new ComplexNum(3, 4);
        ___________________________________    //LINE1
    }
}
```

Identify the appropriate option(s) to fill in the blank at `LINE1`, such the output is:
`norm is :  5.0`

- ○ `print(c.norm());`
- ○ `c.print(c.norm());`
- ✓ `NumberType.print(c.norm());`
- ○ `ComplexNum.print(c.norm());`

Solution: A `static` method belongs to an interface, must be invoked with the same interface name.

4. Consider the code given below. [MSQ:2 points]

```
public interface DBIF{
    public void executeStatement(String qry);
}

public class Database{
    private ConnectionObj cobj = null;
    ---------------------------------- {
        if(isValidate(u, p))
            cobj = new ConnectionObj();
        return cobj;
    }
    public boolean isValidate(String u, String p) {
        //assume the vaidation is always true
        return true;
    }
    private class ConnectionObj implements DBIF{
        public void executeStatement(String qry) {
            System.out.println("Execute: " + qry);
        }
    }
}

public class FClass{
    public static void main(String[] args) {
        DBIF con = new Database().connectDB("test", "test");
        con.executeStatement("fetch students");
    }
}
```

Identify the appropriate option to fill in blank at `LINE1` such that the output becomes `Execute:  fetch students`

- ✓ `public DBIF connectDB(String u, String p)`
- ✓ `public ConnectionObj connectDB(String u, String p)`
- ◯ `public Database connectDB(String u, String p)`
- ◯ `public static ConnectionObj connectDB(String u, String p)`

Solution: Since the method `connectDB()` is called with reference to an object of `Database`, `connectDB()` must be a non-static method.
Since the object returned by `connectDB()` need to be referred by `DBIF`, thus it should return either `DBIF` or `ConnectionObj`.

5. Consider the code given below. [MCQ:2 points]

```
public interface Inter{
  public abstract void greet();
}
public class Greetings{
  private String country;
  public void setCountry(String s){
    this.country = s;
  }
  public String getCountry(){
    return country;
  }
  public Inter checkCountry(){
    if(getCountry() == "India"){
      return new IndiaGreetings();
    }
    return new WorldGreetings();
  }
  private class IndiaGreetings implements Inter{
    public void greet(){
      System.out.println("Hello"+ " "+"India");
    }
  }
  private class WorldGreetings implements Inter{
    public void greet(){
      System.out.println("Hello"+ " "+"World");
    }
  }
}
public class Example {
  public static void main(String[] args) {
    Greetings g = new Greetings();
    g.setCountry("India");
    --------- Line 1 ---------

  }
}
```

Identify the appropriate option to fill in the blank at `Line 1` to print `Hello India` as the output.

- ○ `g.greet();`
- [✓ `g.checkCountry().greet();`]

- ◯ `g.IndiaGreetings.greet();`
- ◯ `g.Inter.IndiaGreetings.greet();`

Solution: To print `Hello India` greet() must be called. `IndiaGreetings` class not visible outside of `Greetings` class, but the interface `Inter` is visible which has greet() abstract method. checkCountry() is called which returns `India Greetings` object. And the greet() is called using that object.

6. Consider the code given below.

```
public class Language{
    public void show(){
        System.out.println("In Language class");
    }
    public class Programming{
        public void show(){
            System.out.println("In Programming class");
        }
    }
}
public class Example {
    public static void main(String[] args) {
        -----------Line 1-----------
    }
}
```

Identify the appropriate option to fill in the blank at `Line 1` to print `In Programming class` as the output.

- ◯ `Language.Programming.show();`
- ◯ `new Language().Programming.show();`
- √ `new Language().new Programming().show();`
- ◯ `Language.new Programming().show();`

7. Consider the Java program given below and predict the output.

[MCQ : 2 points]

```
public interface Animal{
    public void sound();
    default void eat(String animal){
        System.out.println(animal+" eats every day");
    }
}
public class Cat implements Animal{
    public void sound(){
        System.out.println("Cat meows");
    }
}
public class Dog implements Animal{
    public void sound(){
        System.out.println("Dog barks");
    }
}
public class Example{
    public static void main(String[] args){
        Animal oa1=new Cat();
        oa1.sound();
        oa1.eat("Cat");
        Animal oa2=new Dog();
        oa2.sound();
        oa2.eat("Dog");
    }
}
```

- ◯ This code generates a compilation error because an interface can't have fully implemented methods.
- ◯ Cat meows
  Dog barks
- [✓ Cat meows
  Cat eats every day
  Dog barks
  Dog eats every day]
- ◯ Cat eats every day
  Dog eats every day

Solution: Option 3
From JDK 1.8 onwards, the interface supports default methods. Program executes successfully and all methods called.

8. Consider the Java program given below and select the correct option from among the given choices. [MCQ : 2 points]

```
public interface One{
    public void print();
}
public interface Two extends One{
    public void display();
}
public class Three implements Two{
    public void display(){
        System.out.println("This is display");
    }
}
public class  Example{
    public static void main(String[] args){
        Three three=new Three();
        three.display();
    }
}
```

- ✓ Compilation fails because `print()` method has not been overridden in class Three.
- ○ Compilation fails because one interface cannot extend another interface.
- ○ Generates no output.
- ○ This code generates the output: `This is display`

Solution: class Three implements interface Two, interface Two extends from interface One.
class Three should override both `print()` and `display()`.

9. Choose all the correct option(s) regarding the code given below.

```
public class University{
    public int univ_id;

    private class College{
        private String college_name;

        public void getName(){
            System.out.println(college_name);
        }

        public College(String name){
            college_name = name;
        }
    }

    //As the class is private so we are using a public method
    //to return an object of the the inner class

    public College getReference(){
        return new College("IITMadras");
    }

}
public class FClass{
    public static void main(String[] args) {
        University uni = new University();
        (uni.getReference()).getName() ;

    }
}
```

[MSQ:2 points]

- ◯ This code compiles successfully and generates output:
  IITMadras
- ◯ This code generates a compilation error because `college_name` is a private variable.
- ◯ This code generates a compilation error because `getReference()` is actually not returning a valid object of `College` type.
- ✓ This code generates a compilation error because the method `getName()` is not overridden and hence not accessible.

Solution:
Method `getName` is a member function of the nested class `College` which is private, hence the only way we can access this method is by applying dynamic dispatch using method overriding. Therefore we have to use either an abstract class or interface which the inner class would inherit/implement and then we can access the `getName` method as an overridden implementation.
So correct choice would be option 4.

The Java code below models a certain functionality of a ropeway system, carrying tourists, that allows 6 travelers in each cabin. The travelers are let in one after the other into a cabin until the maximum capacity is reached (class CabinCounter). Then, the cabin is allowed to move on, and the next cabin is readied for boarding. A counter (class MasterCounter) is used to track the total number of travelers using the ropeway in a day. Based on the above information and the following code, answer questions 10 and 11.

```
public interface Counter{
    public abstract void inform();
}
public class CabinCounter{
    private MasterCounter master;

    public CabinCounter(MasterCounter mc){
        master = mc;
    }
    public void performCount(){
        int currentCabinCount = 0;
        while (currentCabinCount < 7) {
            Traveler t = new Traveler();
            if t.hasValidTicket(){
                //Let the traveler get inside the current cabin
               currentCabinCount++;
            }
        }
        //let the cabin start moving
        .................... //Blank Line 1
    }
}
public class MasterCounter implements Counter{
    static int dayCount = 0;
    private CabinCounter cc;

    public MasterCounter(){
        cc = new CabinCounter(this); //A new cabin arrives
        cc.performCount();
    }
    public void inform(){
        this.incrementDailyCounter();
        cc = new CabinCounter(this); //A new cabin arrives
        .................... //Blank Line 2
    }
    public void incrementDailyCounter(){
        dayCount = dayCount + 6;
    }

    public static void main(String args[]) {
        MasterCounter masterCounter = new MasterCounter();
    }
}
public class Traveler{
    ....
    public boolean hasValidTicket(){
        //This method checks if the traveler has a valid ticket.
    }
}
```

10. Which should be the line at `Blank Line 1` so that the MasterCounter is notified of a cabin becoming full?

[MCQ: 2points]

- ○ `mc.inform();`
- ✓ `master.inform();`
- ○ `cc.incrementDailyCounter();`
- ○ MasterCounter cannot be notified because `master` is a private object inside CabinCounter.

11. Which should be the line at `Blank Line 2` so that the CabinCounter will start checking for the new cabin becoming full?

[MCQ: 2points]

- ○ There is no need to add any line because the constructor of CabinCounter will initiate the counting for the new cabin.
- ✓ `cc.performCount();`
- ○ `master.incrementDailyCounter();`
- ○ `cc.inform();`