BSCCS2005: Graded Assignment
Questions with Solutions
Week 3

1. Consider the following code: [Nalini: MCQ: 2points]

```
public class A {
    public void display( ){
        System.out.print("Hii ");
    }
}
public class B extends A{
    public void display(String s){
        display();
        System.out.println(s);
    }
    public static void main(String[] args) {
         A a = new B( );
        // Line 1
    }
}
```

What is the correct instruction to be written in *Line 1* in order to print `Hii Ram` as output?

- ◯ a.display("Ram");
- ◯ a.display("Hii Ram");
- ◯ ((A)a).display("Hii Ram")
- √ **((B)a).display("Ram");**

Solution:

- To print Hii Ram display method of subclass must be called.
- But `a` is a reference variable of type A
- So `a` must be made of type B, this can be done by type casting.
- **type casting is happening on reference variable not on the object**
- And then the display( ) method of subclass should be called

Consider the following code and answer the questions 2 and 3.

```
public class Polygon{
    public static void perimeter( ){
        System.out.print("In Polygon perimeter");
    }
    public void area( ){
        System.out.println("In Polygon area");
    }
    public void sides( ){
         System.out.println("In Polygon sides");
    }
    public void angleSum( ){
        System.out.println("In Polygon angleSum");
    }
}
public class Pentagon extends Polygon{
    public static void perimeter( ){
        System.out.println("In Pentagon perimeter");
    }
    public void area( ) {
         System.out.println("In Pentagon area");
    }
    public int sides( ){
         return 5;
    }
    public void angleSum(int x) {
        System.out.println("In Pentagon angleSum");
    }
}
```

2. Which method/s override/s methods in the parentclass?

[Nalini: MSQ: 2points]

- ◯ perimeter
- √ **area**
- ◯ sides
- ◯ angleSum

Solution:

- Method signature must be same in both parentclass and subclass to override the method

3. Which method/s is/are new methods of the subclass?

[Nalini: MSQ: 2points]

- ◯ perimeter
- ◯ area
- √ **sides**
- √ **angleSum**

4. Consider the following code: [Nalini: MSQ: 2points]

```
1    public class Employee {
2        public void display( ){
3            System.out.print("In Employee class");
4        }
5    }
6    public class TeamLead extends Employee{
7        public void display( ){
8            System.out.println("In TeamLead class");
9        }
10   }
11   public class Manager extends Employee, TeamLead{
12        public void display( ){
13            System.out.println("In Manager class");
14        }
15        public static void main(String[] arg) {
16            TeamLead t = new Employee( );
17        }
18   }
```

Identify the line/s which has/have error.

- ◯ Line 1
- ◯ Line 2
- ◯ Line 6
- ◯ Line 7
- √ **Line 11**
- ◯ Line 12
- ◯ Line 15
- √ **Line 16**

Solution:

- A class cannot inherit two classes, multiple inherits is not allowed in java.
- An object of type parent class cannot refer subclass. Because the parent class may not have all the functionalities of subclass

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

```
public class Bird{
    public void fly(){
        System.out.println("it can fly");
    }
}
public class Duck extends Bird{
    public void swim(){
        System.out.println("it can swim");
    }
}
public class FClass{
    public static void doIt(Bird b){
        b.fly();
        if(b instanceof Duck)
            ((Duck) b).swim();
    }
    public static void main(String[] args){
        Duck d = new Duck();
        doIt(d);
    }
}
```

What will be the output?

◯ `it can fly`

◯ `it can swim`

√ `it can fly`
`it can swim`

◯ It generates compiler error

Solution:

In function call `doIt(d);`, `d` is of `Duck` type and implicitly type casted to the parent type `Bird`. However, `b` refers to a subclass object, i.e. object of `Duck`. Thus, `b.fly()` prints `it can fly`. The condition `b instanceof Duck` returns `true` and `((Duck) b).swim()` prints `it can swim`.

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

```
public class Shape{
    public void area(){
        System.out.println("area is unknown");
    }
    public void volume(){
        System.out.println("volume is unknown");
    }
}
public class Rectangle extends Shape{
    public void area(){
        System.out.println("area of Rectangle");
    }
}
public class Cube extends Shape{
    public void area(){
        System.out.println("area of Cube");
    }
    public void volume(){
        System.out.println("volume of Cube");
    }
}
public class FClass{
    public static void compute(Shape s){
        s.area();
        s.volume();
    }
    public static void main(String[] args){
        Rectangle r = new Rectangle();
        Cube c = new Cube();
        compute(r);
        compute(c);
    }
}
```

What will be the output?

◯
```
area is unknown
volume is unknown
area is unknown
volume is unknown
```

◯
```
area of Rectangle
area of Cube
volume of Cube
```

√ `area of Rectangle`
`volume is unknown`
`area of Cube`
`volume of Cube`

◯ It generates compiler error

Solution: For the call `compute(r);`, although the static type of `s` is `Shape`, the dynamic type is `Rectangle` (since it refers to object of `Rectangle`). Thus, it prints

```
area of Rectangle
volume is unknown
```

For the call `compute(c);`, although the static type of `s` is `Shape`, the dynamic type is `Cube` (since it refers to object of `Cube`). Thus, it prints

```
area of Cube
volume of Cube
```

7. Consider the following abstract types.

   - `Scanner`, with method `scanDocuments()`.
   - `Printer`, with method `printDocuments()`.
   - `Copier`, with methods `scanDocuments()` and `printDocuments()`.

   Identify the correct subtype and inheritance relationships between the classes.

   [ARUP: MSQ: 2 points]

   ◯ `Scanner` and `Printer` are subtypes of `Copier`

   √ `Copier` **is a subtype of** `Scanner` **and** `Printer`

   √ `Scanner` **and** `Printer` **both inherit from** `Copier`

   ◯ `Copier` inherits from both `Scanner` and `Printer`

   Solution: Since `Copier` has more functionality than `Scanner` and `Printer`, `Copier` is a subtype of `Scanner` and `Printer`.
   Since we can suppress one of the functions in `Copier` and use it as a `Scanner` or `Printer`, both `Scanner` and `Printer` inherit `Copier`.

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

```
public class Employee{
    private int empid;
    private String name;

    public Employee(int empid_, String name_){
        empid = empid_;
        name = name_;
    }
    public Employee(){
        this(0, "unknown");
    }
    public void print(){
        System.out.print(empid + " : " + name + " : ");
    }
}
public class Manager extends Employee{
    private String department;

    public Manager(int empid_, String name_, String department_){
        _________________________   //line:1
        department = department_;
    }
    public void print(){
        super.print();
        System.out.println(department);
    }
}
public class FClass{
    public static void main(String[] args){
        Manager m = new Manager(101, "Nutan", "HR");
        m.print();
    }
}
```

Identify the appropriate option to fill in the blank at `line:1` such that output of the code will be:

```
101 :  Nutan :  HR
```

◯ `Employee(empid_, name_);`

◯ `this(empid_, name_);`

√ `super(empid_, name_);`

◯ `empid = empid_; name = name_;`

Solution: Since the instance variables `empid` and `name` are declared as `private` in `class Employee`, they are invisible in `class Manager`. Thus, option-4 is incorrect. The only way to initialize the `private` instance variables in `class Employee` by calling it's constructor, which can be done by the subclass `Manager` using `super` keyword.

9. Consider the following code. [Bhaskar:MSQ:2points]

```
public class Student{
    public String sname;
    public String sid;
    public int sclass;

    public Student(String s_name,String s_id, int s_class){
        this.sname = s_name;
        this.sid =s_id;
        this.sclass = s_class;
    }
    public void display() {
        System.out.println("name:"+sname);
        System.out.println("id:"+sid);
        System.out.println("class:"+sclass);
    }
}

public class Toppers extends Student{
    public int marks;

    public Toppers(int marks){
        this.marks = marks;
    }

    public void display() {
        super.display();
        System.out.println("marks:"+marks);
    }
}

public class FClass{
    public static void main(String[] args){
        Toppers t = new Toppers(30);
        t.display();
    }
}
```

Choose all the correct option/s which mention/s the required modification (if any) in the code for successful compilation.

◯ `super.display` must be removed.

√ **Define an appropriate no argument constructor of Student class.**

√ **Call the Student class parameterized constructor explicitly inside Topper class's constructor.**

◯ No modification required, code compiles successfully.

Solution: While we define an object of Topper class the constructor of Topper class is executed. This constructor will first call the no argument constructor of its superclass implicitly if an explicit call to superclass constructor is missing.
We have defined a parameterized constructor in the superclass which hides the default no argument constructor and therefore the no argument constructor call by the child class constructor will not get resolved and subsequently generate an error.
There can be two solutions to this problem:

1. Defining a no argument constructor in superclass explicitly
2. Call the parameterised constructor (already defined) of superclass explicitly, inside the constructor's body of child class

10. Consider the following code. [Bhaskar:MCQ:2ppoints]

```
1.  public class Mammal{
2.     public String name;
3.     public int lifespan;
4.
5.     public Mammal(){
6.          name = "Tiger";
7.          lifespan = 45;
8.     }
9.
10.    public void show() {
11.         System.out.format("name = %s : lifespan = %d",name,lifespan);
12.    }
13.    public void display() {
14.         System.out.println("Mammal details");
15.    }
16. }
17.
18. public class Endangered extends Mammal{
19.     public boolean endanger_status;
20.
21.     public Endangered(){
22.         endanger_status = false;
23.     }
24.
25.     public void show() {
26.         System.out.println("endanger status of "+
                                this.name + " is " + endanger_status);
27.     }
28.     public void display(String species) {
29.         System.out.println("Endanger status");
30.     }
31. }
32.
33. public class FClass{
34.     public static void main(String args[]) {
35.         Mammal m1 = new Endangered();
36.         m1.show();
37.         m1.display();
38.     }
39. }
```

Choose the correct option regarding the given code.

- ◯ This code produces a compilation error on line 37
- ◯ This code produces a compilation error on line 36
- ◯ This code will generate the output:
  endanger status of Tiger is false
  Endanger status
- √ **This code will generate the output:**
  endanger status of Tiger is false
  Mammal details
- ◯ This code will generate the output:
  endanger status of *null* is false
  Mammal details

Solution: The `display() and display(String) are two overloaded methods in Endangered class whereas the show() method of Mammal class is an overridden method.`
`Overloading is done during compile time and method invocation is resolved depending upon reference type whereas for overridden methods instance type decides the version to be invoked during run time.`
`As the instance type to which` *`m1`* `is referring to is Endangered therefore show() method of Endangered class is executed.`