Advanced Programming (CSE201), Quiz -2

Time allocated: 09:45am – 10:15am (30 minutes) + 5 minutes for uploading solution

Name	
Roll Number	

Instructions:

- You must follow all the instructions sent to you earlier over the email.
- Only reasonable and clearly mentioned assumptions (if any) would be accepted.
- For justifications, please be as concise as possible (2-3 sentences only)

Question-1) What will be the output of the program shown in Figure-1? Correct the compilation error / runtime exception (**if any**) in the program shown in Figure-1. Brief justification. **[2 marks]**

```
public interface Storage {
    public void storageType();
}
public class Phone {
    public void storageType() {
        System.out.println("SD card");
    }
}
public class iPhone
    extends Phone implements Storage {
    public static void main(String[] args) {
        Storage S = new iPhone();
        Phone P = new iPhone();
        S.storageType();
        P.storageType();
    }
}
```

Figure-1

Rubric-Q1

There is no error/exception and the o/p will be SD Card (printed twice) [+1 marks]

```
public class Shape {
    private String color;
    private double area;
    public Shape(String c, double a) {
        color = c;
        area = a;
    }
    public boolean equals(Shape o) {
        return o.color == this.color && o.area == this.area;
    }
}
public final class Circle extends Shape {
    public static final double PI = 3.14;
    public Circle(String c, int r) {
        super(c, PI * r * r);
    }
    public static void main(String[] args) {
        Circle c1=new Circle("RED", 10);
        Circle c2=new Circle("RED", 10);
        System.out.println(c1.equals(c2));          //true
        System.out.println(c1.equals("ABC"));       //false
    }
}
```

Figure-2

The interface method has exactly same signature with the method inside Phone class. As iPhone inherits storageType method from Phone, it doesn't need to override the interface method again. [+1 marks]

Question-2) Assume the code in the classes Shape and Circle are exactly as shown in Figure-2. What is/are the **bare minimum** change(s) required in this code such that **only** the statement highlighted in the main() method returns **true and false**, respectively? "Bare minimum changes", means all those changes specific **only** to this sample scenario, but not for general scenario. Rewrite the corrected code/method only (if required). No need to rewrite the entire class in that case. **[3 marks]**

NOTE: Question-2 also has -VE partial marking components for any extra changes you do than required. However, total marks obtained in this question cannot be negative.

Rubric-Q2

Option-1: If someone mentioned there is no issue with the code and it will work without errors then full marks will be awarded! [+3 marks]

Option-2: If someone is providing the implementation of equals method:

Bare minimum change is to simply correct the equals() method inside **Shape** class, exactly as following. Likewise, overriding equals inside Circle is not required as Circle has only got a static variable that is associated with a class, but not with instances.

```
public boolean equals(~~Shape~~ Object o) {          //+0.5 marks
   if(o instanceof Shape) {                          //+0.75 marks
      Shape s = (Shape) o;                           //+0.75 marks
      return s.color == this.color && s.area == this.area;   //+0.5 marks
   } else return false;                              //+0.5 marks
}
```

OR

```
public boolean equals(~~Shape~~ Object o) {          //+0.5 marks
   if(o instanceof String) {                         //+0.75 marks
      return false;                                  //+0.5 marks
   } else {
      Shape s = (Shape) o;                           //+0.75 marks
      return s.color == this.color && s.area == this.area;   //+0.5 marks
   }
}
```

OR

```
public boolean equals(~~Shape~~ Object o) {          //+0.5 marks
   if(o!=null && o.getClass() == getClass()) {                    //+0.75 marks
      Shape s = (Shape) o;                           //+0.75 marks
      return s.color == this.color && s.area == this.area;   //+0.5 marks
   } else return false;                              //+0.5 marks
}
```

Both "**s.**color == this.color" and "**s.**color.equals(this.color)" are fine in this particular case.

If any number of extra code than above, then -0.5 marks as negative marking (in total, i.e. max deduction). Total marks obtained cannot be less than zero. **Recall this question asked Bare Minimum changes.**

Question-3) Can two different Strings S1 and S2 can return different results in equality check when this check is performed either as S1==S2 or as S1.equals(S2)? Printing both S1 and S2 gives the **exact** same result. Brief justification. **[2 marks]**

Rubric-Q3: (This was discussed in Tuesday's lecture)
Yes. [+1 marks]
Reason: [+1 marks]
String S1 = "abc"; String S2 = new String("abc");
S1==S2 =>false.
S1.equals(S2) => true.

Question-4) What is/are the **bare minimum** change(s) required in the program shown in Figure-3 such that the print inside main outputs **ChildParent**? You must adhere to OOP principles. Ignore the CloneNotSupportedException and assume that there is no issue with it. "Bare minimum changes", means all those changes specific **only** to this sample scenario, but not for general scenario. **[3 marks]**

```
public class Parent {
   private String name = "Parent";
}
public class Child extends Parent implements Cloneable {
   private String name = "Child";
   public Child clone throws CloneNotSupportedException {
      return super.clone();
   }
   public static void main(String[] args)
                        throws CloneNotSupportedException  {
      Child C1 = new Child();
      Parent P2 = C1.clone();
      System.out.println(P2);
   }
}
```

Figure-3

Rubrics-Q4:

Bare minimum change is to simply implement toString() method in both these classes. There is no need to provide a clone() implementation inside Parent as the Child class has already provided the clone method. Had it been that: a) Parent had the clone() implementation, but not the Child, OR b) the declared-type for C1 was Parent instead of Child, then clone() implementation was mandatory in Parent as well. The actual-type of the object P2 is clone of "Child" type object. Hence, the clone implementation inside Parent is optional in this scenario. Calling super.clone() inside Child will lead the call to Object class where the JVM would prepare the object and copy the attributes of all the classes down in the hierarchy up to Child class.

Code inside Parent class:

```
public String toString() {              //+0.5 marks
   return name;
}
```

Code inside Child class (you must use super.clone(), but not a getter):

```
public String toString() {              //+0.5 marks
   return name+super.toString();        //+1 marks
}
```

Typecasting super.clone() as: "(Child) super.clone()" //+1 marks