Chapter 11: Inheritance & PolyMorphism

Question 11.1 :

True or False? A subclass is a set of a superclass?

False;

Subclass usually contains more information and methods than its superclass.

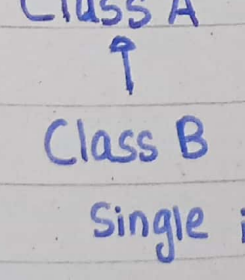
Question 11.2 :

What keyword do you use to define a subclass?

We use ~~inheri~~ extend keyword to inherit from parent class.

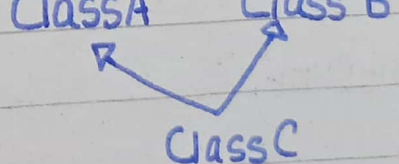
Question 11.3 :

What is single level inheritance? and multiple inheritance? Does java support multiple inheritance?



Single inheritance

When a subclass can extend only one superclass.



Multiple inheritance.

Multiple inheritance allows a subclass to extend multiple class.

Java doesn't support multiple inheritance

Question 11-4:

What is output of running class C in (a)? What problem arises in compiling the program in (b).

a

```
class A {
    public A() {
        System.out.println("A is no-arg constructor");
    }
}

class B extends A {
    }
public class C {
    public static void main (String [] args){
        B b = new B();
    }
    }
```

Output:

```
A is no-arg constructor
```

```
class A{
    public A(int x){
        }
}
class B extends A {
        public B(){
            }
    }
public class C {
      public static void main (String [] args){
          B b = new B();
        }
    }
```

Answer:-

Problem arises in compiling the above program is; when the object of class B is initialized because no argument is passed for constructor of class A.

Question 11.5:

How does a subclass invoke its superclass constructor?

Subclass invoke its superclass constructor using super keyword.

```
class SuperClass{
        Superclass(){
          }
    }
class SubClass extends SuperClass{
          SubClass(){
            super(); // invoking superclass constructor using super //
      }
}
```

Question 11.6:

True or False ? When invoking a constructor from a subclass. Its superclass's no arg constructor is always invoked.

False, We can explicitly call constructor with arguments. In that case no-arg constructor wouldn't be called.

Question 11.7

True or False? You can override a private method defined in superclass

Private method is not accesible from sub class so it can't be overridden. (False)

Question 11.8 :-

True or False ? You can override a ~~private~~ static method defined in superclass ?

Overriding static method is not allowed. (False).

Question 11.9 :-

How do you explicitly invoke a superclass's constructor from a subclass ?

We can explicitly invoke a superclass's constructor from a subclass using super(), keyword.

Question 11-10 :-

How do you invoke an overridden superclass method from a subclass?

Ans:-

To invoke an overriden superclass from a subclass method by using super keyword.

Example :-

If there is a method circle(), defined in a superclass and overridden in a subclass, to invoke method from a superclass, we need to write super.circle().

Question 11.11 :-

Identify problems in following code?

```
1        public class Circle{
2          private double radius;
3
4          public Circle (double radius){
5                this.radius = radius;
6           }
7
8          public doule getRadius(){
9                return radius;
10          }
11
```

```
12      public double getArea(){
13          return radius * radius * Math.PI;
14          }
15  }
16
17  class B extends Circle{
18          private double length;
19
20          B(double radius, double length){
21              Circle(radius);
22              length = length;
23          }
24
25          public double getArea(){
26              return getArea() * length;
27          }
28  }
```

Answer:-

Line 5 and 22, missing this keyword.
Line 21 - super(radius) instead of Circle(radius).

Question 12 :-

Difference b/w Method overloading and overriding ?

	Method Overloading	Method OverWriting
1	Method overloading is used to increase readability of program.	Method overriding is used to provide specific implementation of the method that is provide by its super class.
2	Method overloading is performed within class.	Method overriding is occur in two classes that has is-a relationship.
3	In the case method overloading, parameter must be different	In the case method overriding parameter must be same.
4	Method overloading is the example of compile time polymorphism.	Method overriding is example of runtime polymorphism.

Question 11.13:-

If a method in a subclass has same signature as a method in its superclass with same return type, is method overridden or overloaded?

If a method in a subclass has same signature and same return type as method in superclass then method is overridden

Question 11.14:-

If a method ~~is~~ in a subclass has same signature as a method in its superclass with ~~same~~ different return type. will this be a problem.

This will result in a syntax error because to override method the return type also has to be same.

Question 11.15:-

If a method in a subclass has same name as a method in its superclass with different parameter type, is the method overloading or overridden?

It is method overloading

Question 11.17: What is polymorphism? What is dynamic binding?

Polymorphism allows object of different classes to be treated as objects of a common base class.

Dynamic binding when type of object is determine at runtime.

Question 11.18: Describe difference b/w method matching and method binding?

Method matching in java, there are multiple method which have same name Depending on what arguments are provided to method, what are type of those arguments and return type different method is invoked. This is decided by compiler.

Method Binding may be implemented in several classes along implements in several classes along inheritance chain. during execution of program.

Question 11.19:

Can you assign new int[50], new Integer[50], new String[50] or new Object[50], into a variable of Object[] type?

Yes can assign all of them except, int[50] because it will give you an error

: Object[] o = new int[50];

Question 11.20:

What is wrong in the following code?

```
1  public class Test{
2      public static void main(String[]args){
3          Integer [] list1 = {2,1}
4          Double[] list2 = {3.0, 2.0}
5          int[]    list3 = {100, 20}
6          printArray (list1);
7          printArray (list2);
8          printArray (list3);
9      }
10
11
```

```
12 public static void printArray (Object[] list){
13         for (Object o: list)
14             sout (o + " ");
15             sout( )
16         }
17 }
```

Answer:-

Error is in line 5

because,

int is a primitive and object isn't its superclass.

Assigning list 3 of int[] type to object[] is not possible.

Question 11.21:- Show Output of following code

```
public class Test{
    psvm{
    new Person().printPerson();
    new Student().printPerson();
    }
}

    class Student extends Person{
        @Override
        public | private String getInfo(){
            return "Student";
        }
    }

class Person{
private | public String getInfo(){
                return "Person";
            }
        public void printPerson(){
            sout(getInfo());
        }
    }
```

Part a | Part b

Part b | Part a

Answer:-

Part (a):- Person
Student

Part (b):- Person
Person

Question 11.22: Show output of following program

```
public class Test {
    public static void main (String[] args){
        A a = new A(3);
    }
}

class A extends B {
    public A(int t){
        System.out.println("A's constructor");
    }
}

class B{
    public B(){
        System.out.println("B's constructor");
    }
}
```

Output:

```
B's constructor
A's constructor
```

Question 11.23:

Show Output of following code:

```
public class Test {
        public static void main (String [] args){
            new A();
            new B();
    }
  }
class A {
    int i = 7
      public A(){
          set I(20);
          System.out.println ("i from A is" + i)
      }
      public void setI(int i){
          this.i = 2*i;
        }
      }
class B extends A {
      public B(){
          sout ("i from B is" + i);
      }
      public void setI(int i){
          this.i = 3*i;
        }
}
```

Output: i from A is 40
i from A is 60
i from B is 60

- Question 11.24:

  Indicate true or false for following statement.

  - You can always successfully cast an instance of a subclass to a superclass

    True

  - You can always successfully cast an instance of a superclass to a subclass.

    False

- Question 11.25:

  For the geometric object and Circle classes, answer following question:

a Assume are circle and object created as follows:

```
Circle circle = new Circle();
GeometricObject object = new GeometricObject();
```

Are the following Boolean Expression are true or false

(circle instanceOf GeometricObject) True
(object instanceof GeometricObject) True
(circle instanceof Circle) True
(object instance of Circle) False

b Can following statements be compiled ?

```
Circle circle = new Circle(5);
GeometricObject object = circle;
```

The given statement can be compiled

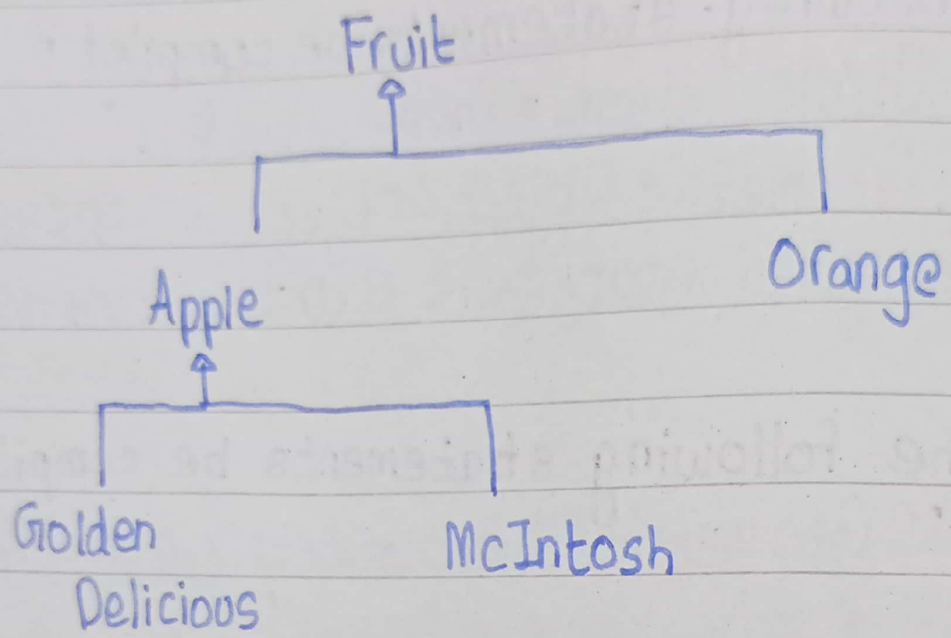
c Can the following statements be compiled?

```
GeometricObject object = new GeometricObject();
Circle circle = (Circle) object;
```

The given statement cannot be compiled

Q Question 11.26:

Suppose that Fruit, Apple, Orange, Golden Delicious and McIntosh are defined in following inheritance hierarchy:



Assume that following code is given:

```
Fruit fruit = new Golden delicious();
Orange orange = new Orange();
```

Answer following question:

a Is fruit instance of Fruit ?

True

Is fruit instanceof Orange?

False

Is fruit instance of Apple?

True

Suppose the method makeAppleCider is defined in Apple class. Can fruit invoke this method? Can Orange invoke this method?

Answer :-

Fruit can not invoke method AppleCider because fruit is variable of type Fruit. In order to invoke method, it needs to be down casted.

(Apple) fruit.AppleCider();

Orange also cannot invoke method.

Suppose method makeOrangeJuice is defined in Orange class? can orange invoked this method? Can fruit invoke this method?

Orange can invoke this method. and fruit cannot invoke this method?

. Is statement Orange p = new Apple(); legal?

Illegal

. Is statement McIntosh p = new Apple() legal.

Illegal

. Is statement Apple p = new McIntosh() legal.

Yes legal?

Q 11.27:

What is wrong in following code?

```
1  public class Test {
2      psvm{
3      Object fruit = new Fruit();
4      Object apple = (Apple) fruit;
5    }
6  }
7  class Apple extends Fruit {
8    }
9  class Fruit{
10   }
```

Answer :-

Error is in line 4 :- because Fruit cannot be cast to Apple as it is a superclass of Apple

11.37 :- What modifier should you use on a class so that a class in same package can access it, but a class in different package cannot access the cl it?

No Modifier (default)

Q 11.38 :- What modifier should you use so that a class in a different package cannot access the class, but its subclasses in any packag can access it?

Protected

Q Question 11.39:

In the following code, the classes A & B are in same package. If question marks in (a) are replaced by blanks, can class B be compiled? If question marks are replaced by private, can class B be compiled? If question marks are replaced by protected, can class B be compiled?

```
package p1;

public class A{
  ?  class$ A{/
  ?    int i ;
  ?    void m(){
        .....
  }
}
```

(a)

```
package p1;

public class B extends A {
    public void m1(String[] args) {
        System.out.println(i);
        m();
    }
}
```

(b)

Ans:-

If Question Marks are replaced by private, class B cannot compiled as it does not have access to class A. Modifier private allows access to the variable only inside class.

If Question Marks are replaced by ~~private~~ protected, class B can~~not~~ be compiled. as it allows access to the member of class by subclasses even if subclas-es are outside of package.

Question 11.41:

- How do you prevent a class from being extended? How do you prevent a method from being extended?

We need to use final statement.

- Final class can have ~~extended~~ .instance

Yes you can instantiate but you cannot extend a class.