Java programming - 22412)

Unit III : Inheritance, Interface and Package

(Weightage - 12 marks)

PRANJAL SAVE (ST-CO)

3.1 Inheritance : concept of inheritance, Types of Inheritance

Questions ----

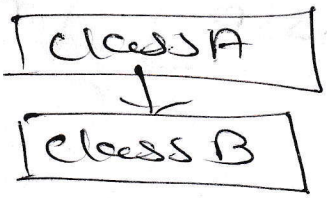
Q) List the types of inheritance which is supported by java. (2m)

Inheritance

* The ability of object of one class to acquire properties of object of another class is known as inheritance.
* The existing class is known as base class (parent class) and the new class derived from exist class is known as derived class. (child class)
* The inheritance can be achieved by incooperating the definition of one class into another using the keyword extends.
* The inheritance is a mechanism in which the child class is derived from parent class.
* This derivation is using keyword extends.

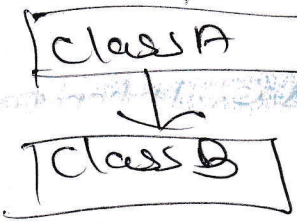
Syntax

```
class A          // base class
  {
  ----
  }
class B extends A   // derived class.
  {
  ----
  }
```



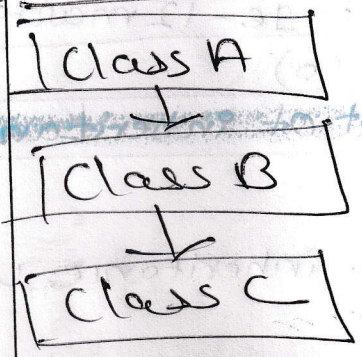
Types of inheritances

Single inheritance



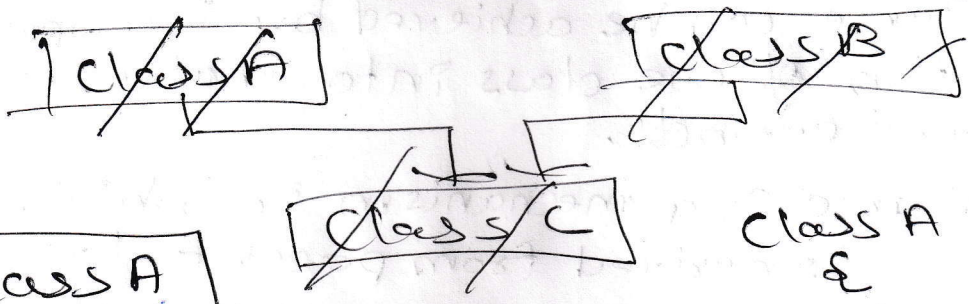

```
class A
{
}
class B extends A
{
}
```

Multilevel inheritance




```
class A
{
}
class B extends A
{
}
class C extends B
{
}
```

Hierarchical inheritance




```
class A
{
}
class B extends A
{
}
class C extends A
{
}
```

* Does not support multiple & Hybrid inheritance in JAVA.

3.2 Single inheritance & multilevel inheritance, Hierarchical inheritance. method overloading and overriding, dynamic method dispatch, final variables, final methods, use of super, abstract method and classes, static members.

Questions ---

Q1 State the use of final keyword with respect to inheritance. (4M)

Q2 Develop a program to create a class 'Book' having data members 'author', 'title' and 'price'. Derive a class 'BookInfo' having data member 'stockposition' and method to initialize and display the information for three objects. (4M).

Q3 Explain single and multilevel inheritance with proper example. (4M)

Q4 Describe final variable and final method. (4M)

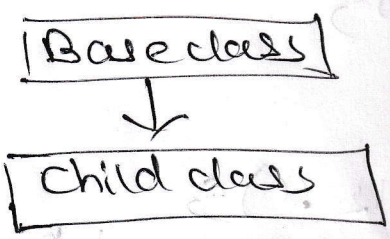
Q5 Explain the concept of Dynamic method dispatch with suitable example. (6M/4M)

Q6 Differentiate between method overloading and method overriding. (4M)

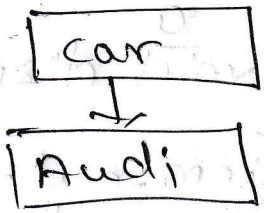
Q7 WAP to show the Hierarchical inheritance. (4M)

Single inheritance

A derived class with only one base class is single inheritance.



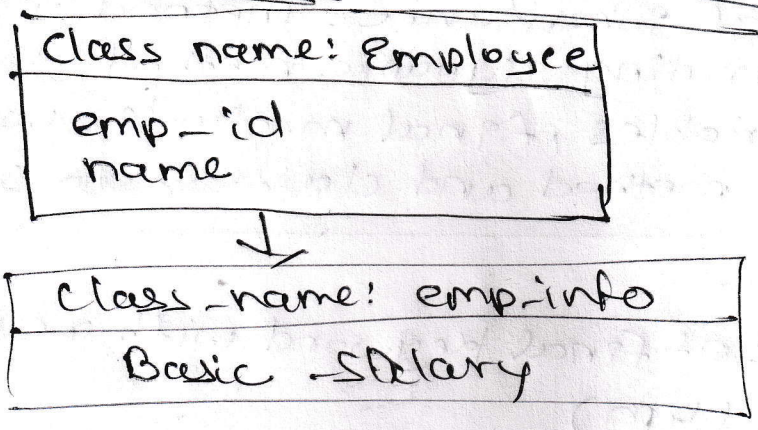
example,



Syntax

```
class Base
{
}
class Child extends Base
{
}
```

program to implement Single inheritance

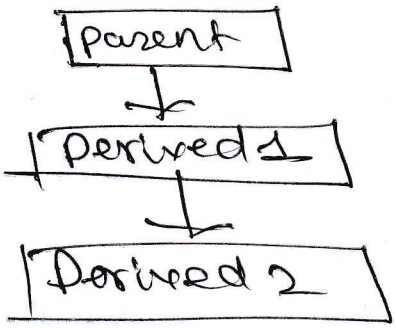


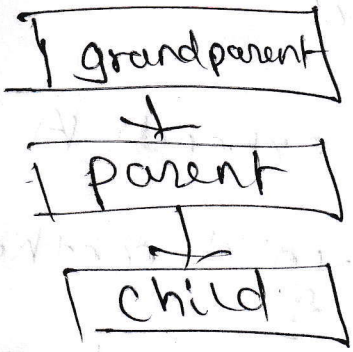

```
import java.util.*;
public class employee
    {
      int emp_id;
      string name;
    }
  class emp-info extends employee
    {
     int basic_salary;
     void getdata()
       {
       cout << "Enter emp-id"
System.out.println("Enter emp_id");
Scanner sc = new Scanner(System.in);
emp_id = sc.nextInt();
System.out.println("Enter name");
Scanner sp = new Scanner(System.in);
name = sp.nextline();
System.out.println("Enter salary");
Scanner st = new scanner(System.in);
basic_salary = st.nextInt();
    }
     void display()
    {
System.out.println("Details");
System.out.println(+emp_id +name, +basic_
                                salary);
    }
}
```

```
class S_I
{
public static void main (String args[])
  {
  emp-info e = new emp-info();
  e.getdata();
  e.display();
  }
}
```

Multilevel inheritance

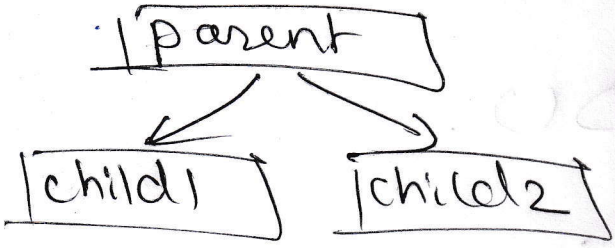
The mechanism of deriving class from another derived class is known as multilevel inheritance.

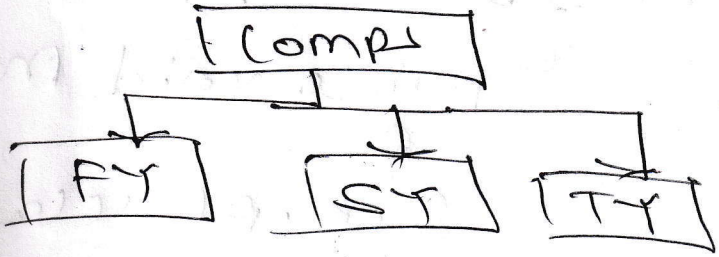




Hiearchical inheritance

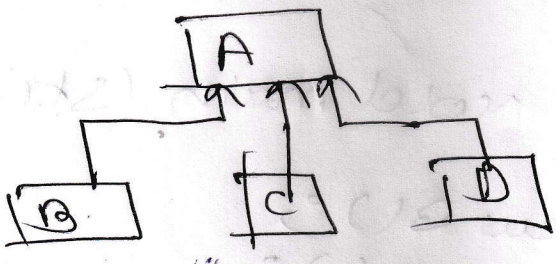
If one Base class is inherited by more than one derived class is called Hiearchical inheritance





Question: Write a program to show Hiearchical Inheritance. ..... 4m.

⇒



```
class A
{
   void methodA()
    {
   System.out.println("class A");
    }

}
class B extends A
    {
       void method B()
       {
    System.out.println("class B");
       }
    }
class C extends A
      {
        void method C()
         {
    system.out.println("class C");
        }
     }
class D extends A
    {  public void method D()
         {
         S.O.P("class D");
         }

    }
class Java
   { public static void main.(String Args[])
      {
      B obj1 = new B();
      C obj2 = new C();
      D obj3 = new D();
      obj4.methodA();
      obj2.methodB();
}}      obj3.methodC();
```

Method overloading

- Method Overloading means to define different methods with the same name but different parameters lists and different definitions.
- It is used when objects are required to perform similiar task but using different input parameters thay may vary either in number or type of arguements.
- Overloaded methods may have different return types.
- It is way of achieving polymorphism in JAVA.

```
int add (int a , int b)
int add (int a , int b, int c)
int a double add( double a, double b)
```

Example

```
class Sample
{
    int addition (int i , int j)
        {
         return i+j;
        }
    string addition (string s1 , string s2)
        {
         return s1+s2;
        }
    double addition( double d1 , double d2)
        {
         return d1+d2;
        }
}
```

```
Class AddOperation
  {
  public static void main (String Args[])
    {
    Sample obj = new Sample();
  System.out.println(obj.addition(1,2));
  System.out.println(obj.addition("Hello","world");
  System.out.println(obj.addition(1.5,2.5);
    }
  }
```

Method Overriding

- IF subclass (childclass) has same method as declared in parent class, it is known as as method Overriding.
- IF subclass provides the specific implementation of method that has been provided by one of parent class, it is known as method overridin
- Method overriding is used for runtime polymorphism

Example

```
class Vechicle
    {
      void run()
        {
        S.O.P("vechile");
        }
    }
class Bike extends vechicle
    {
      void run()
        {
        S.O.P("Bike");
        }
public static void main(String args[])
    { Bike obj = new Bike();
      obj.run();
```

Method Overloading	Method Overriding
Method overloading is a compile time polymorphism.	Method overriding is a run-time polymorphism.
It occurs within the class	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding requires inheritance
Private and final method can be overloaded.	Private and final method can't be overridden
Poor performance, due to compile time polymorphism.	Better performance.
The arguement list should be different while doing method overloading.	The arguement list should be same while doing method overriding
In method overloading, method name - same and different signatures.	In method overriding, same name and same signature.
Return type can or can not be same, but just have to change parameter	Return type must be same ~~name~~.
Efficient & faster	less efficient & slower.

Dynamic Method Dispatch (Runtime Polymorphism)

(4m/6m)

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refer to. In simple words

the type of object which it referred determines which version of overridden method will be called.

*Example

```
Class Game
{
public void type()
{ System.out.println("Indoor & Outdoor"); }
}
Class Cricket extends Game
{
public void type()
{
   System.out.println("outdoor game");
{
Public static void main(String[] args)
{
Game gm = new Game();
Cricket ck = new Cricket();
gm.type();
ck.type();
//gm = ck;  //gm refers to Cricket object
//gm.type(); // calls Cricket's version of type
}
}
```

Final keyword

* Final is a keyword in JAVA, which generally means, cannot be changed once created.
* Final behaves very differently when variables, methods and classes specifically means
  - A final variable cannot be reassigned once initalized.
  - A final method cannot be overriden.
  - A final class cannot be extended.

Final keyword with variables

- Final variable works like const of C-language that can't be altered in the whole program.
- That is, final variable once created can't be changed and they must be used as it is by all the program code.
- When a variable is declared final, it is constant which will not and cannot change.
- Example : final PI = 3.14;
- The value of PI is declared as final so it cannot be changed in the course of program.

Program

```
class Final {
    public static void main(String args[])
    {
        final int y=20;
        int x=10;
    System.out.println("x="+x);
    System.out.println("y="+y);
    x=30;
    y=40;
    System.out.println("x="+x);
    System.out.println("y="+y);
    }
}
```

Output

Cannot assign a value to final variable y

Final with Methods

- You can also declare method as final.
- A method that is declared final cannot be overridden in subclass.
- The syntax is simple, just put the keyword final after the access specifier and before the return type.
- **Syntax:** access_specifier final returntype methodname()
  {
  //body.
  }
- When you try to override method that has been declared final it will generate compile-time error.

Program

```
class A
{
  final void meth()
    {
      int a=10;
    }
}
class B extends A
{
  void meth() //Error!
  {
    int a=15;
  }
}
```

O/P

method cannot be overriden.

Final with Classes

- If we want the class not be sub-classed (or extended) by any other class, declare it final.
- Classes declared final can not be extended.
- That is, any class can use the method of a final class by creating object of final class and call the methods with the object.

Syntax

```
final class class-name
    {
      //body.
    }
```

program

```
final class Demo1
 {
   public void display()
     {
   System.out.println("hi");
     }
 }
public class Demo2 extends Demo1 // Error
 {
   public static void main(String args[])
     {
       Demo1 d = new Demo1();
        d.display();
     }
 }
```

O/P

Cannot inherit from final Demo1

Question : Develop a program to create a class 'Book' having data members 'author', 'title' and 'price'. Derive a class 'BookInfo' having datamember 'stockposition' and method to initalize and display the info. for three objects.

Solution

```
class Book {
string author, title;
 double price;
public Book(String author, String title, double price)
   {
    this.author = author;
    this.title = title;
     this.price = price;
    }
}
class Book_Info extends Book{
  int stockposition;
 public Book_Info(string author, string title, double
                  price, int stockposition)

    {
     super(author, title, price);
     this.stockposition = stockposition;
    }
public void display()
{
System.out.println("Title : " +title);
System.out.println("Author : " +author);
System.out.println("Price : " +price);
System.out.println("stock Position : " +stockpositio
   }
}
public class BookDemo {
    public static void main(String args[])
        {
BookInfo ~~book~~ b1 = new BookInfo("Ak", "JPR", 28.99, 10
 b1.display();
```

```
BookInfo b2 = new BookInfo("DK","CYD",30.99,200);
b2.display();
BookInfo b3 = new BookInfo("XY","OOP",69.11,300);
b3.display();
}
}
```

3.3 Interfaces: Define interface, implementing interface, accessing interface, variables and methods, extending interfaces, interface references, nested interfaces.

Questions.....

Q// Write a program to create a class 'salary' with data members 'empid' 'name' and 'basicsalary'. Write an interface "Allowance" which stores rates of calculation for da as 90% of basic salary, hra as 10% of basic salary and pf as 8.33% of basic salary. Include a method to calculate net salary and display. ...6m

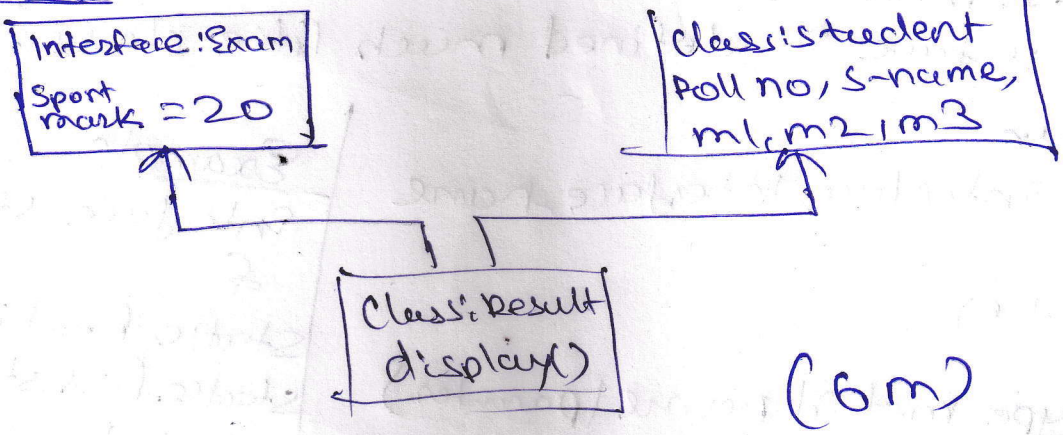
Q// Define interface in Java. (2m)

Q// Difference between class & interface. (4m)

Q// Develop and interest interface which contains simple interest and compound interest methods and static final field of rate 25%. write a class to implement those methods. (6m)

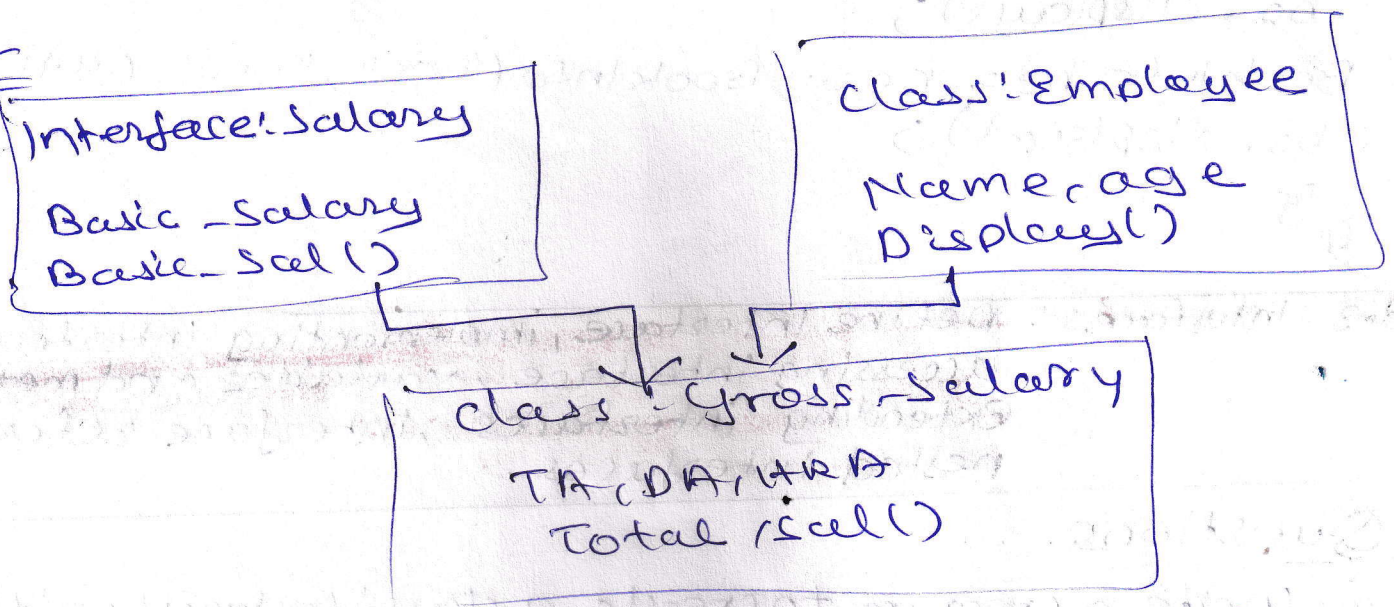
programs

Q1



(6m)

Q2



6M

Interface

- Interface is also known as kind of class.
- Interface also contains methods and variables but with major difference, the interface consist of only abstract method. (i.e they're declared not defined. they're just outlines waiting to be filled in by classes that implement the interface) and final fields.
- This means that interface do not specify any code to implement those methods and data fields contains only constants.
- Therefore, it is responsibility of class that implements an interface to define the code for implementation of these methods.
- An interface is defined much like class.

Syntax

```
Access interface interface_name
 {
method()
---
return type method name (parmter)
 type final variable 1 = value 1
 type final variable N = value n
}
```

Example

```
interface student
 {
static final int roll_no=
static final string name="
 void show();
}
```

Need of Interface

- Achieve multiple Interface.
- We can implement more than one interface in one class.
- Methods can be implemented by one or more class

Class	Interface
The keyword used to create a class is "class".	The keyword used to create a interface is "interface"
Object of can be created of class.	whereas, Object of interface cannot be created.
~~Supports~~ Does not support multiple inheritance	supports multiple inheritance.
It can contain constructors	it cannot contain constructors
Cannot contain abstract methods.	contain abstract methods only
variables in a class can be static, final or either	All variables are static and final.
It has instant variable	it has final variable
Access specifier can be public, private & protected.	Interface has only public access specifier
Classes are always extended	Interfaces are implemented
Memory is allocated for class.	memory is not allocated for ~~class~~ interface.

Class Example
{
void method1()
{
body
}
void method2()
{
body
}
}
	interface Example
	{
	int x=5;
	void method1();
	void method2();
	}

Supports Hybrid inheritance	Does not support Hybrid inheritance

Question

Develop and Interest Inteface which contains simple interest and compound interest methods and static final field of rate 25%. write class to implement those methods. ... 6m.

⇒
```
public interface Interest
{
double RATE = 0.25;
double calculateSimpleInterest(double principal, double time);
double calculateSimpleInterest(double principal, double time);
}

public class InterestCalculator implements Interest
{
public double calculateSimpleInterface(double principal, double time)
{
return principal * RATE * time;
}
public double calculateSimpleInterface(double principal, double time)
{
double amount = principal * Math.pow(1+Rate, time);
return amount - principal;
}
public static void main(String args[])
{
InterestCalculator c = new InterestCalculator();
double principal = 1000;
double time = 2;
double simpleInterest = calculator.calculateSimpleInterest(principal, time);
double simpleInterest = calculator.calculateSimpleInterest(principal, time);
S.O.P("simple Interest = "+simpleinterest);
S.O.P("Compound Interest = "+compoundinterest);
}}
```

Question

Write a program to create a class 'salary' with data members 'empid' 'name' and 'basic salary'. Write an interface 'Allowance' which stores rate of calculation for da as 90% of basic salary, hra as 10% of basic salary and pf of ces 8.33% of basic salary. Include a method to calculate a net salary and display. (6M)

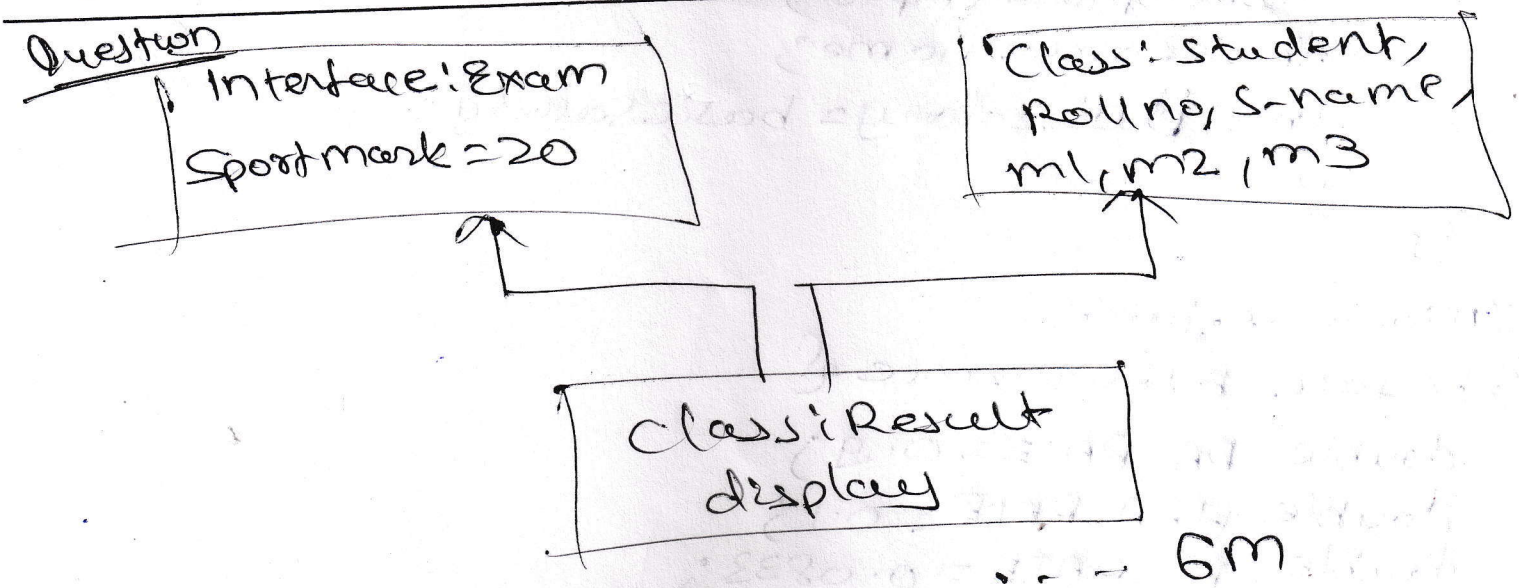
⇒ Class Salary

```
{
int empId;
String name;
double basicsalary;
Salary (int empId, String name, double basicsalary)
{
this.empId = empId;
this.name = name;
this.Basicsalary = basicsalary;
}
}
```

```
//Allowance.java
interface Allowanance {
double DA_RATE = 0.9;
double HRA_RATE = 0.1;
double PF_RATE = 0.0833;
default double CalculateNetSalary (Salary salary)
{
double da = salary.basicSalary * DA_RATE;
double hra = salary.basicsalary * HRA_RATE;
double pf = Salary.basicSalary * PF_RATE;
return salary.basicsalary * da+hra - pf;
}
}
```

```
//main.java
public class main implements Allowance {
  public static void main (String args[])
    {
   Salary employee = new salary(101, "John Doe", 50000
    main m = new main();
  double netsalary = main.calculateNetSalary(employee
  S.O.P (employee.empId);
  S.O.P (employee.name);
  S.O.P (netsalary);
  }
}
```

Question

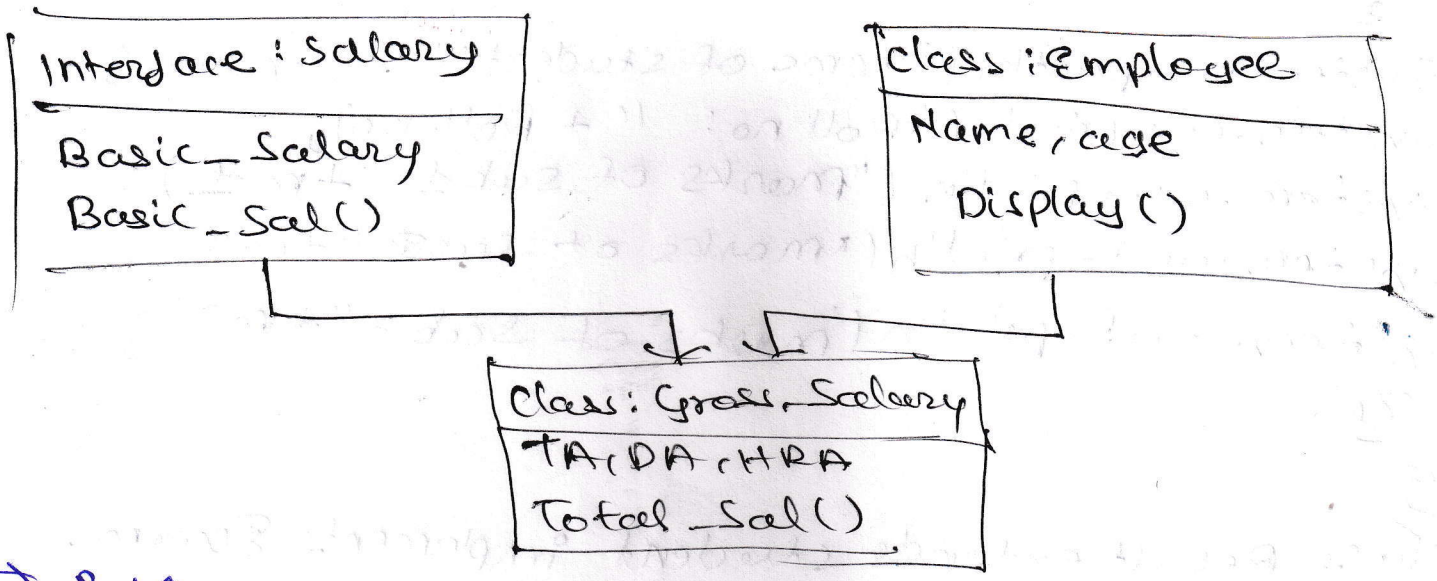


... 6M

```
interface Exam {
   int sport_mark=20
  }
class {student
   {
  String S-name;
   int Roll_no, m1, m2, m3;
   student(string n, int a, int b, int c, int d)
   {
   S-name = n;
   Roll_no = a;
    m1 = b;
    m2 = c;
    m3 = d;
```

```
    void showdata()
    {
System.out.println("Name of student:" +s_name);
System.out.println("Roll no: " + Roll_no);
System.out.println("marks of sub1 " +m1);
System.out.println("marks of sub2 " +m2);
System.out.println("marks of sub3 "+m3);
    }
}
class Result extends student implements Exam.
    {
     Result(string n, int a, int b, int c, int d)
        {
          super(n,a,b,c,d)
        }
      void display()
        {
        super.showdata();
        int total = m1+m2 tm3;
        float result =(total +Sport_mark)/total * 100;
      System.out.println("result of student "+result);
        }
}
class Stud_Details
    {
      public static void main(String args[])
        {
        Result r = new Result("Sachin",14,78,85,97);
        r.display();
      }
}
```

Question



⇒
```
interface salary
{
double Basic_Salary=10000.0;
void Basic_Sal();
{
System.out.println("Basic Salary = "+Basic_Salary);
}
}
class Employee
{
String Name;
int age;
Employee(String n, int a)
{
Name=n;
age=a;
}
void Display()
{
System.out.println("Name : "+Name);
System.out.println("Age : "+age);
}
}
```

```
class Gross_salary extends Employee implements Salary
{
    double HRA, TA, DA;
    Gross_Salary (String n, int b, double h, double t,
                  double d)
    {
        super(n, a);
        HRA = h;
        TA = t;
        DA = d;
    }
    void Total Basic_Sal()
    {
        double Total_Sal = Basic_Salary + TA + DA + HRA;
        System.out.println("Total Salary : "+Total_Sal);
    }
}
class EmpDetails
{
    public static void main (String args[])
    {
        Gross_Salary s = new Gross_Salary("Sachin",
        20, 10000, 2000, 7000);
        s.totalSal();
    }
}
```

3.4 Package : Define package, type of package naming and creating package, accessing package, import statement, static import, adding class and interface to a packages.

Questions ----

Q1 Enlist any four inbuilt packages in Java. (2m).

Q2 Explain how to create a package and how to import it. (4m)

Q3 How to create user defined package in Java. Explain with suitable example. (6m).

Q4 Describe package in java with suitable example. (4m)

Q5 Give a syntax to create a package and accessing package in java. (2m)

Q6 Define packages. How to create user define package? Explain with example. (6m)

Q7

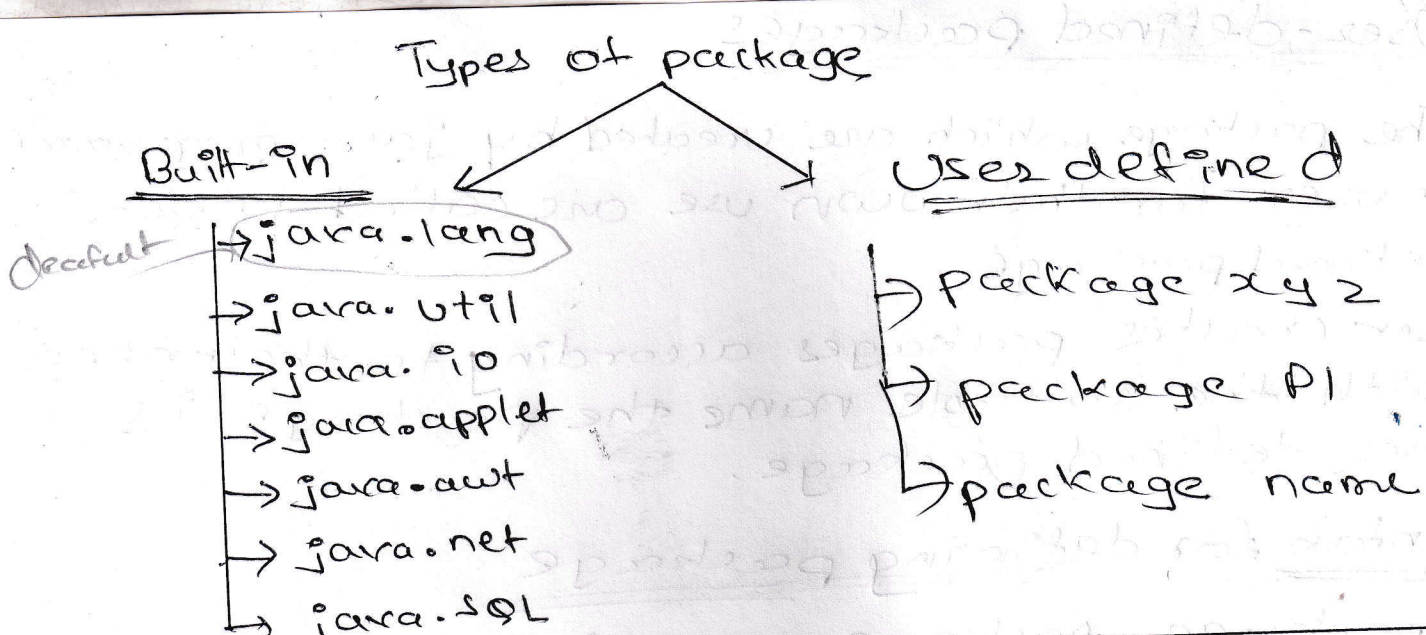
Package

A package arrange number of classes, interface and sub-package of same type into particular group.

Note: Package is nothing but folder in windows.

In simpler terms, a package in Java is like a folder that helps organize and manage classes.

- It's like putting similiar things together in Box
- When you create a package in Java, you're grouping related classes under one name.
- This helps avoiding naming conflicts and makes its easier to organize and find your classes.
- Just like how you organize files into folders on your computer, you can organize classes into package in Java.



Predefined / Built in Packages - created by Java developer already

- Predefined packages are packages that come with Java and provide wide-range of built-in functionalities.
- These packages are part of the Java Development kit (JDK) and are ready to use without any additional setup.
- Some examples are.

java.lang	contains fundamental classes and interfaces that are automatically imported into every Java language. Example, String, Integer etc.
java.util	Contains utility classes and data structures such as list, set, maps and includes Scanner class for input
java.io	Provide classes for input and output operations, such as reading and writing into files
java.net	contains classes for networking operations, such as creating network connecting and communicating over internet
java.applet	for applet using in program
java.awt	GUI/Standalone program
java.SQL	JDBC purpose.

User-defined packages

The package which are created by java programmer or user for their own use are called user-defined package.
user creates packages according to their need with the suitable name the package is user-defined package.

Syntax for defining package

```
package package-name;
```

Here, package_name is name of package.
example, `package xyz;`

Rules for package

i) Package statement must be first line of program.
ii) The way of compilation of these would be different.

Compilation

```
javac -d . classname.java
```

Package are mirrored by directories. - A file system structure containing files of other directories.
Java system uses file system directories to store packages.
The class file of any classes which are declared in package must be stored in directory which has same name as package.
The directory must match with package name exactly.
A hierarchy can be created by seperating package name and sub package name by a period(.) as pkg1.pkg2.pkg3; which requires a directory structure as pkg1/pkg2/pkg3.

Syntax

To access package. In Java source file import statements occur immediately following the package statement and before any class definition

```
import pkg1.pkg2.(classname(*));
```

(* — everything)

Example `import airish.A;`

airish — package_name; A — class_name.

Example

```
package package_xyz;  //Declaration
public class Box
  {
    int l=2, b=7, h=8;
      public void display()
       {
  System.out.println("Volume="+(l*b*h));
       }
  }
```

Source file

```
import package_xyz.Box;
 Class volume
  {
  public static void main(String args[])
    {
      Box b = new Box()
       b.display();
    }
}
```