Le-1, 23, 32, 60, 242

Binding:- Connecting a method call to the method body is called Binding.

Binding
- static
- Dynamic.

static binding: this binding happens during compilation time.
- ⇒ it also called early binding, because binding happens before a program actually runs.
- ⇒ Binding of all static, private and final methods are done at compile-time. Eg:- Method overloading.

Dynamic binding:- in this compiler doesn't decide which method to be called. Eg:- overriding.
- ⇒ it also called late binding, because binding happens during execution time.
- ⇒ it allocates memory at run-time.

Eg:-
```
class person {
    void eat()
    {
        sopln("person is eating");
    }
}
class Doctor extends person {
    void eat() {
        sopln("Doctor is eating");
    }
}
Public class DynamicEx {
    P SV m() {
        Person P = new Doctor();
        P.eat();
```

o/p:-
Doctor is eating.

abstract classes & methods:- Abstract is a keyword.

Abstraction:- it is a process of hiding the implementation details and showing only functionality to the user.

⇒ it hides internal details and only shows essential things to user. Eg:- sending SMS.
you type text and send the message, but you don't know the internal processing about the message delivery.

Abstract class:- A class which is declared as abstract is called as abstract class. it can have abstract & non-abstract methods. it needs to be extended and its methods implemented.

⇒ it contains variables, methods and at least one abstract method.

Ex:-
```
abstract class shape
{
  int a=2;        → variables
  int b=4;
  abstract void printArea();   → at least one abstract method.
}
```

abstract class Rules:-

1. abstract class must be declared with an abstract keyword.
2. it can have abstract & non-abstract methods.
3. it can have final methods
4. to use an abstract class, you have to inherit it from sub classes.
5. if a class contain partial implementation, then we should declare a class as abstract.

Syntax:-

```
abstract class class_name {
    // abstract methods & non-abstract methods
}
```

Abstract Method:- A method which is declared as abstract and does not have implementation is known as abstract method.

→ abstract method does not contain any code or body.

```
{
}
```

X no code, no body.

→ all abstract class methods implemented immediate sub class. it contains only declaration but implementation done is sub class.

Eg- → method declaration should be done in super class.

abstract void printArea();

→ here we are declaring only defination but implementation done immediate sub class.

Eg:-

```
abstract class Shap
{
  int a=2;
  int b=4;
  abstract void printArea();
}
class Rectangle extends Shape
{
  void printArea()
  {int r=a*b;
  sopln("area of rectangle=" +r);
  }
}
```

its implementation done here. only defination declareted.

this is Immidiate sub class program

L3

Syntax of abstract method :-

abstract access modifier return type method_name();

no method body & method

Eg :- abstract void printArea();

Eg 2 :- Bike :- Bike is an abstract class that contains only one abstract method run. its implementation is provided by Honda class.

```
abstract class Bike {          → class abstract
    abstract void run();       → method abstract
}
class Honda4 extends Bike {
    void run() {
        System.out.println("running safely");
    }
    public static void main() {
        Bike obj = new Honda4();
        obj.run();
    }
}
```

O/P :- running safely

Packages :- Defining a Package, creating & Accessing a Package :-

Defining a Package :- it is necessary in software development to create several classes & interfaces. after creating these classes & interfaces, it is better if they are divided into some groups depending on their relationship. thus, the classes and interfaces which handle similar or same task are put into the same directory. this directory or folder is also called as Package.

Eg:- in your laptop, you want to store bunch of data. Data includes favorite music, songs etc.

favorite music, songs, images.
↓ ↓ ↓
folder 1 folder 2 folder 3

⇒ we are creating seperate folders for each section to retrive the data immediately & to find data immediatly.

⇒ if we mix all music & songs, images in only 1 folder it is not possible to search particular file immedia

... class available in java.sql package. ...g4 also

Package Defination :- A Java package is a group of similar types of classes, interfaces & sub-packages.

advantages:-

1. Java Package is used to categorize the classes & interfaces so that they can be easily maintained.
2. Java Package provides access protection.
3. Java Package removes naming collision.

⇒ Packages are useful to arrange related classes & interfaces into a group. this makes all the classes & interfaces performing the same task to put together in the same package.

Eg:- in Java, all the classes & interfaces which perform input & output operations are stored in java.io. package.

⇒ Packages hide the classes & interfaces in a separate sub directory, so that accidental deletion of classes & interfaces will not take place.

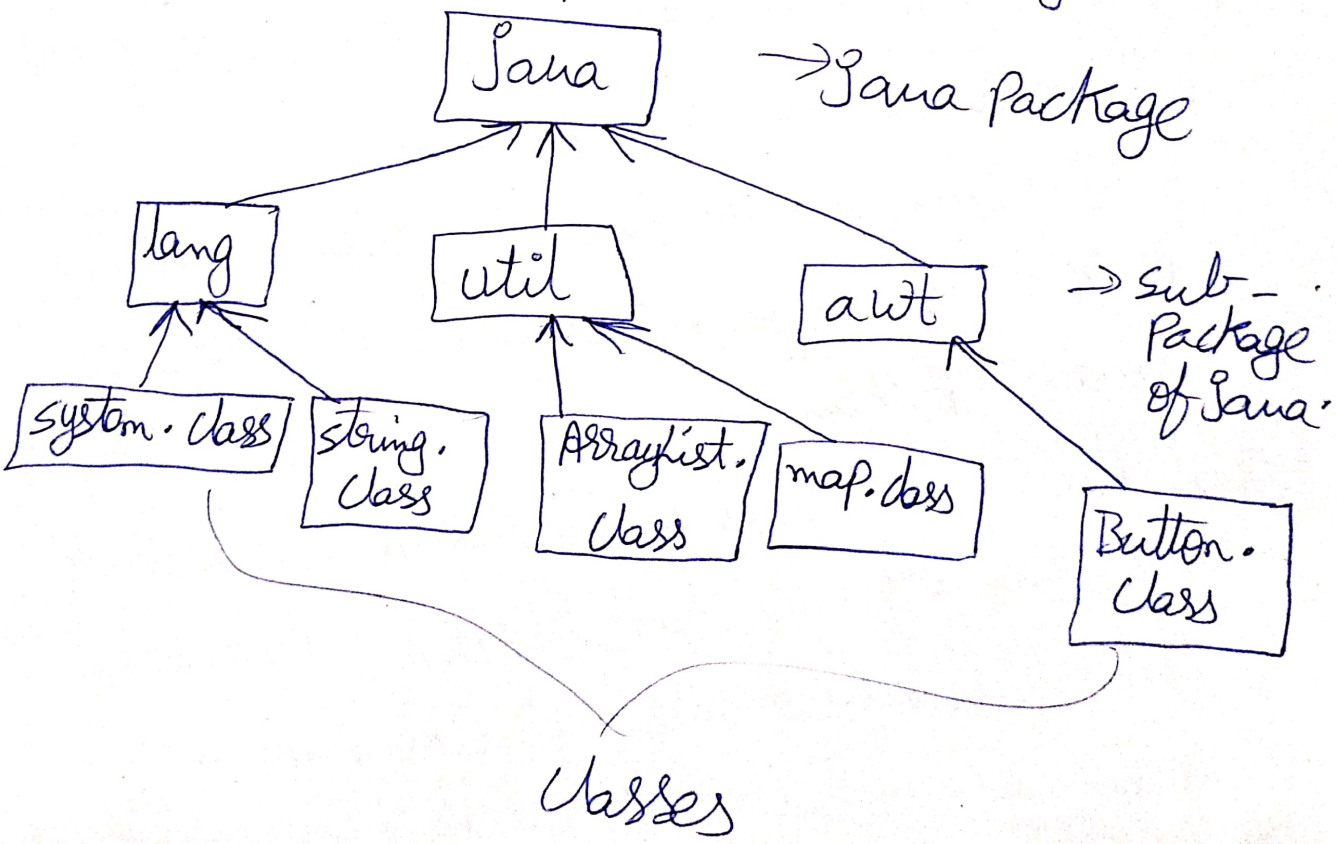
⇒ we can use same names for classes of two different classes. Eg:- Date class in java.util Package & also Date class available in java.sql package.

→ A group of packages is called a library. The classes & interfaces of a package are like books in a library & can be re-used several times. This re-usability nature of package makes programming easy.

Different types of Packages:-

1. Built-in 2. User-defined

1. Built-in Package :- the package which are already created by Java developer people are called pre-defined / Built-in package.



java.lang.*:— this package as primary classes & interfaces essential for Java language. it consists of wrapper classes. (wrapper is a piece of paper/thin metal covers & protects something you buy. eg: food cover)
Ex:- system, string, Integer, character etc.

java.util.*:— this package consists useful classes & interfaces like stack, LinkedList, ArrayList, Date etc.

java.io.*:—this package handles files and input, output related tasks. Ex:- File, FileInputStream, FileOutputStream, FileReader, FileWriters.

java.awt.*:— awt means Abstract window Toolkit. this package helps to develop GUI (Graphical User Interface). it consist 2 important packages:

1. java.awt.event.*;
2. java.awt.image.*;

Javax.swing.*:- this package helps to develop GUI.
it consists some additional features of awt.

java.net.*:- client & server programming can be done using packages it uses TCP/IP, TCP protocals means (transmission control protocol & Internet protocol).

java.applet.*:- applets are small intelligent program which travel from one place to another place on Internet & executed in client side.

java.sql.*:- this package helps to connect to the data base like Oracle, MS Access etc.

java.beans.*:- Beans are software re-usable components in the network. they can develop the package.

java.rmi.*:- # rmi stands for Remote method Invocation. this object which exists one computer in the network can be invoked from another computer & can be used.

javax.servlet.*:- servlets are server side programming which handles clients.

Creating and Accessing a Package :-

User defined Package :-

Creating a Package :- The users of the Java language can also create their own packages. They are called user-defined packages. User defined Packages can also be imported into other classes & used exactly in the same way as the Built-in Packages.

Syntax :-

1. Package Packagename; // to create a Package.
2. Package Packagename.subPackagename; // to create a sub Package within a Package.

Ex :- → keyword

```
Package mypack;
Public class Simple {
PSVm() {
S.O.Pln("welcome to Package");
}}
```

o/p :- welcome to Package.

TO Compile :- → javac -d . javafilename

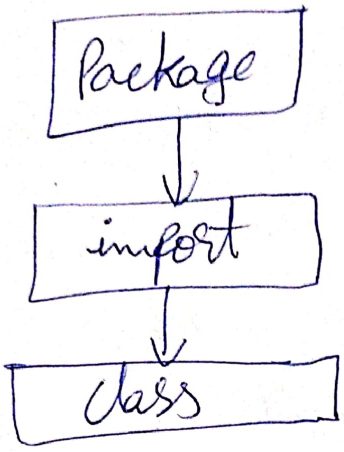
→ directory

Ex :- javac -d . Simple.java

(space → space.)

TORun :- Java mypack.Simple

Note:- sequence of the program must be Package then import then class.



Sub-Package in Java:- Package inside the Package is called sub-Package.

Ex:- Syntax:- Package Packagename.Subpackage; //

to create a Sub Package within a Package

Understanding CLASSPATH:-

The CLASSPATH is an environment variable that tells the Java compiler where to look for class files import.

→ CLASSPATH is generally set to a directory or a Jar (Java Archive) file.

Interface:- it is a blue print of a class, used to specify the behavior of a class. it has static constants & abstract methods.

⇒ the interface in Java is a mechanism to achieve multiple inheritance & abstraction there can be only abstract methods in Java interface, not method body. it also represents the IS-A relationship.

⇒ we can have default, static and private methods in interface

⇒ we decide on a type of entity by its behavior & not via attribute.

Uses:- 1. it is used to achieve abstraction.

2. By interface, we can support the functionality of multiple inheritance.

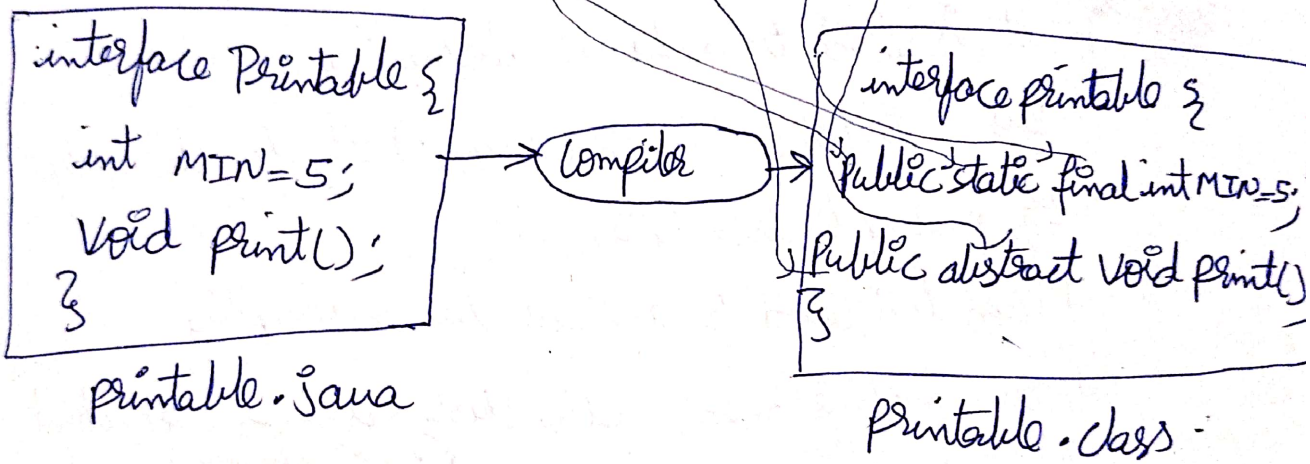
3. it can be used to achieve loose coupling.
(loose coupling means the classes are independent of each other. they have less dependent on each other.

How to declare an interface:- An interface is declared by using the interface keyword. it provides total abstraction, means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.

Syntax:-

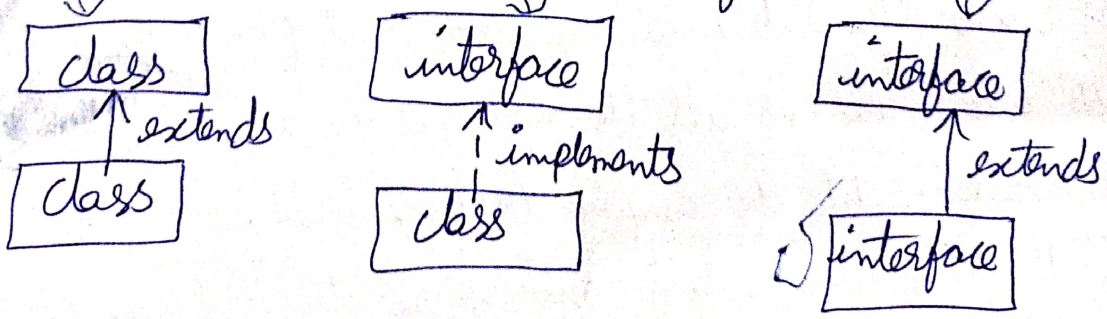
```
interface <interface_name> {
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

⇒ In interface, the Java compiler adds interface fields are public, static and final by default, and the methods are public and abstract.



relationship between classes & interfaces :-

A class extends another class, an interface extends another interface but a class implements an interface.



Ex:- The printable interface has only one method, and its implementation is provided in the A6 class.

Ex1:-
```
interface printable{                    → rectangle etc.
void print();
}
class A6 implements printable{
Public void print(){
System.out.println("Hello");
}
Public static void main(String args[]){
A6 obj = new A6();
obj.print();
}}
```
O/P: Hello.

Ex2:- Bank

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
Public float rateOfInterest(){
return 9.15f;
}
class PNB implements Bank{
Public float rateOfInterest(){
return 9.7f;}
}
```

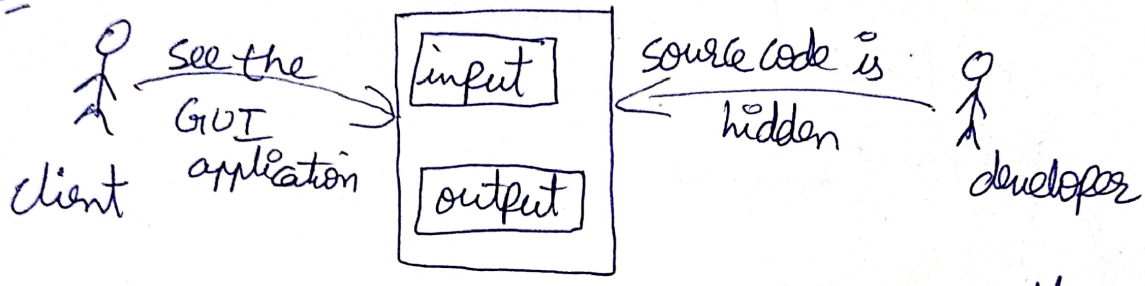
```
class TestInterface2{
P S V m(){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOf
                    Interest());
}}
```
O/P:- ROI: 9.15.

Implementing interface :-

⇒ implementing interface using "implements" Keyword. the class implements all the methods at interface. because the interface methods are abstract methods.

⇒ interface is nothing but it deals between client & developers.

Ex:-



Ex:- putable.

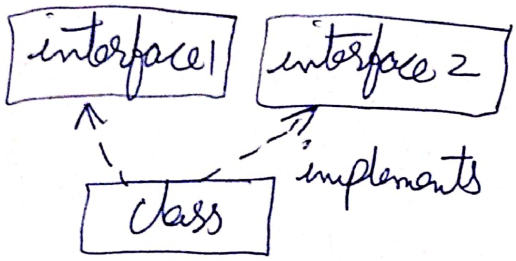
```
interface client
{
void input();   // Public + abstract
void output();
}
class Raju implements client
{
  String name;      // Public + static + final
  double sal;       // Global variables.
Public void input();
{
Scanner s = new Scanner(System.in);
SOPln("enter username:");
    name = s.nextLine();
SOPln("enter salary:");
  sal = s.nextDouble();
}
Public void output()
{
SOPln(name+" "+sal);
}
PSVm(){
client c = new Raju;
interface obj name  new memory  class obj
c.input();
c.output();
}}
```

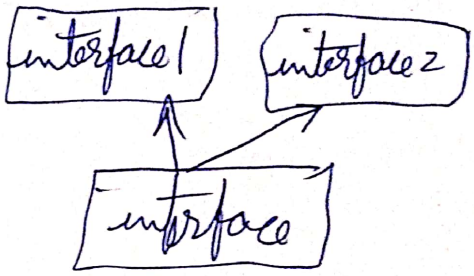
O/P:

Enter username:
Ankit
Enter salary:
50,000.00
Ankit 50.000.0

Multiple inheritance by Interface:-

if a class implements multiple interface, or an interface extends multiple interfaces, it is known as multiple inheritance.





Ex:- printable, showable.

Variables in Interface:- the variables are default in public + static + final in interface. lets go to understood various variables in interface.

Ex:-

```
interface CustomerRaj {
    int amt = 100;      // public + static + final
    void purchase();    // public + abstract
}
class SellerSanjii implements CustomerRaj
{
    @Override // over-writing base class method
    public void purchase()
    {
        SOPln("Raj needs" + amt + " Kg rice");
    }
}
class check { PSVm() {
    CustomerRaj c = new SellerSanju();
    c.purchase(); SOPln(customerRaj.amt);
}}
```

o/p:-
Raj needs 100 Kg rice ~~100~~
100

diff b/w class & interface

class	interface
1. The keyword use to create a class is "class"	1. The keyword used to create an interface is "interface"
2. A class can be instantiated i.e., objects of a class can be created.	2. An Interface cannot be instantiated i.e., objects cannot be created.
3. it does not support multiple inheritance	3. it support multiple inheritance
4. it can be inherited by another class using the keyword "extends".	4. it can be inherited by a class by using the keyword "implements" & it can be inherited by an interface using the keyword "extends".
5. it contain constructors	5. it cannot contain constructor
6. it cannot contain abstract methods	6. it contains abstract methods only
7. variables & methods in a class can be declared using any acess specifiers (Public, private, protected, default)	7. all variables & methods in a interface are declare as Public