Unit-2

Inheritance, Packages, Interfaces

Inheritance : Basics, Using Super, Creating Multilevel hierarchy, Method Overriding, Dynamic Method Dispatch, Using Abstract classes, Using final with inheritance, Object class.

Packages : Basics, Finding packages and classpath, Access Protection, Importing packages.

Interfaces : Definition, Implementing Interfaces, Extending Interfaces, Nested Interfaces, Applying Interfaces, Variables in Interfaces.

Packages :-

A java package is a group of similar types of classes, interfaces and subpackages.

Packages in java can be categorized in two forms

(i) Build-in Packages.

(ii) User defined packages.

Build-in Packages :-

Build-in packages are the packages from Java API. The Java API is a library of pre-defined classes, interfaces & sub-packages. The built-in packages were included in the JDK.

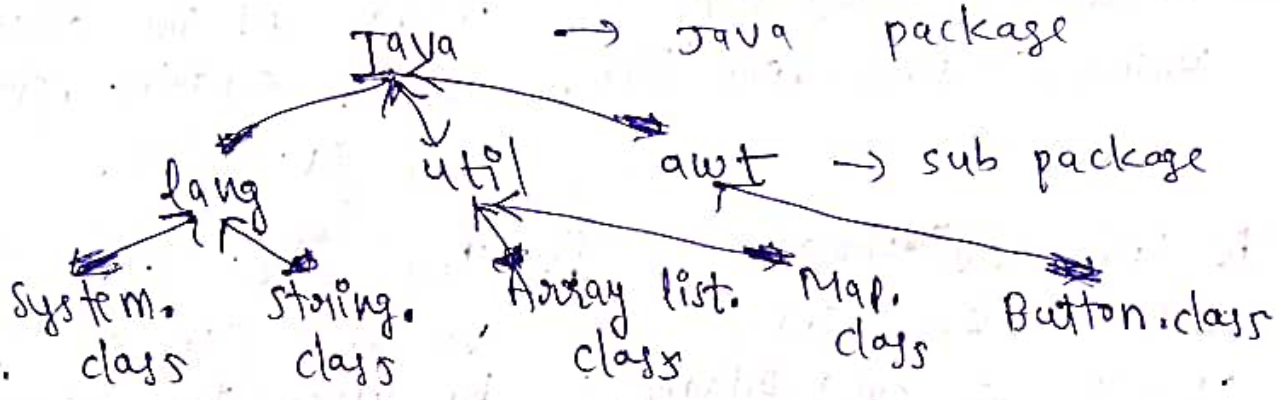
User defined packages :-

User defined packages are the packages created by the user, is free to create their own packages.

There are many built-in packages such as java, lang, awt, javax, swing, net, IO, util, sql.

Advantages of Java packages:-

1) A Java package is used to categorized the classes and interfaces, so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.



Class Path:-

It is an environment variable which is used by application class loader to locate and load the class files. The class path defines the path to find third-party & user defined classes that are not extensions are part of java platform include all the directories which contain .class files and JAR files, when the setting the class path.

Difference b/w path and class path.

Path	Class path.
(i) path is an environment variable.	(i) class path is also an environment variable
(ii) It is used by the operating system to find the executable files (.exe)	(ii) It is used by the application class loader to locate the .class files.
(iii) You are required to include the directory which contains .exe files.	(iii) You are required to include all the directory. which contains .class and jar files.
(iv) Path environment variable once set, cannot be overridden.	(v) The class path environment variable can be overridden by using the commandline option -cp (or) class path to both javac and java command.

(i) Private:-

The access level of a private modifier is only within the class. It can not be accessed from outside the class (or). The private members can be accessed only inside the same class.

Eg:- class A {
    private int data=40;
    private void msg()
    {
        System.out.println("Hello Java");
    }
}

```
public class Simple {
 public static void main (String args[]) {
  A obj = new A();
  System.out.println (obj.data);
   obj.msg();
 }
}
```

(iii) Protected :-

The protected access modifier is accessible within package & outside the package but through inheritance only. The protected access modifier can be applied on the data member, method & constructor applied on the class. It provides are accessibility than the default modifier. (or)

The protected members are accessible to every child class (same packages (or) other packages)

```
package my pack;
 import pack.*;
 class B extends A{
 public static void main(String args[])
 {
   B obj = new B();
   obj.msg();
 }
}
```

```
package pack;
public class A {
 protected void msg()
 {
  System.out.println ("Hello");
 }
}
```

(iii) Default :- (or) No modifier

If you don't use any modifier, it is treated as "default" by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. (or) The default members are accessible within the same packages but not outside the package.

Ex:- package my pack;

```
import pack.*;
class B{
public static void main(String args[])
{
A a=obj;
obj.msg();
}
}
package pack;
class A{
void msg()
{
System.out.println("Hello");
}
}
```

(iv) Public :-

The public access modifier is accessible every where.

Ex:-
```
package pack;
public class A{
public void msg(){
System.out.println("Hello");
}
}
package mypack;
import pack.*;
class B{
public static void main(String args[])
{
A obj=new A();
obj.msg();
}
}
```

Method Signature:-

```
class Test{
public int m1(int i)  // method initialization
{
}
public void m2(String s) // method initialization
{
}
public static void main(String args[])
{
Test t=new Test(); // creating an object.
t.m1(10);
```

```
t.m2("Hello");
System.out.println(t.m1);
System.out.println(t.m2);
}
}
```

Inheritance:-

Inheritance in java is a mechanism in which one object acquires all the properties and behaviours of parent object. It is an important of oops.

Syntax:-

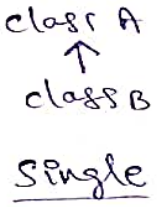
```
class subclassname extends Superclassname
{
  // methods and fields
}
```

Ex:-

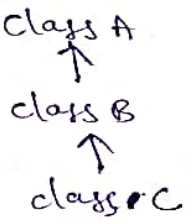
```
class Animal{
    void eat(){
      System.out.println("Eating");
    }
}
class Dog extends Animal{
  void Bark(){
    System.out.println("Barking");
  }
}
class Test{
  public static void main(String args[])
  {
    Dog d=new Dog();
```

```
        d.bark();
        d.eat();
    }
}
```

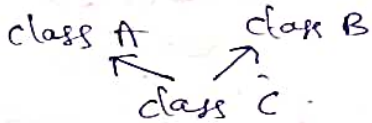
Types of inheritance:-



Single



Multilevel inheritance.



Multiple.

Ex:-
```
class GrandParent {
    void Land() {
        System.out.println("Own the lands");
    }
}
class Parent extends GrandParent {
    void Cars() {
        System.out.println("Own cars");
    }
}
class Son extends Parent {
    void use() {
        System.out.println("Uses properties of parent
                            and grand parent");
    }
}
```

```
class Family {
public static void main(String args[])
{
 Son s=new Son();
  s.Cars();
  s.Land();
  s.use();
}
}
```

Method Overriding:-

If the sub class (child class) as the same method as declared in the parent class. It is known as overriding.

Method Overloading:-

If a class has multiple methods having same name but different in parameters (signature). It is known as method overloading.

If we have to perform only one operation having same name of the methods increases the readability of the program.

Advantages of method overloading:-

(i) Method over loading increases the readability of the program.
(ii) Increases the flexibility of the program.
(iii) Reduces the complexity of the program.

Method Overloading :-

```
class Casio {
    public void add(int i, int j)
    {
     System.out.println(i+j);
    }
  public void add(int i,int j,int k)
    {
     System.out.println(i+j+k);
    }
}
public class Method Overloading{
    public static void main(String args[])
    {
      Casio obj=new Casio();
       obj.add(1,2);
       obj.add(1,2,3);
    }
}
```

Method Overriding :-

```
class A{
    public void show(){
     System.out.println("A is running");
    }
}
class B extends A{
    public void show(){
```

```
        System.out.println("B is running");
    }
}
public class Overriding {
    public static void main(String args[]) {
        B obj = new B();
        obj.show();
    }
}
```

In a subclass provides the specific implementation of the method that has been declared by one of its parent class it is known as method overriding.

Uses of Java Method Overriding :-

Method overriding is used to provide the specific implementation of a method which is already provided by its super class. Method overriding is used for run time polymorphism.

Rules for Java method overriding :-

(i) The method must have the same name as in the parent class.

(ii) The method must have the same parameter as in the parent class.

(iii) There must be an "is a relationship" [Inheritance]

Constructor Overloading :-

Constructor overloading in java is a technique of having more than one constructor with different parameter lists. They are arrange in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list.

In Java a constructor is just like a method, but ~~without~~ without return type. It can also be overloaded like java methods.

Super Keyword :-

The super keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you creat the instance of sub class and instance of parent class is created implicitly. which is referred by the super reference variable.

Usage of Java Super keyword :-

(i) Super can be used to refer immediate parent class instance variable.

(ii) Super can be used to invoke immediate parent class method.

(iii) Super can be used to invoke immediate parent class constructor.

Ex:-
```
class A{
    void show(){
        System.out.println("This is A");
    }
}
class B extends A{
    void show(){
        super.show();
        System.out.println("This is B");
    }
}
public class Superdemo{
public static void main(String args[]){
    B obj=new B();
    obj.show();
}
}
```
O/P:- This is A
This is B

Ex:-
```
class A{
    void show(){
        System.out.println("This is A");
    }
}
class B extends A{
    void show(){
        System.out.println("This is B");
    }
}
```

```
class C extends B{
    void show(){
        super.show();
        System.out.println("This is C");
    }
}
public class Superdemo{
    public static void main(String args[]){
        C obj=new C();
        obj.show();
    }
}
```

replace with B.

Runtime Polymorphism / Dynamic Method Dispatch:-

Runtime polymorphism in java is achieved by method overriding in which a child class overrides a method in its parent, an overriden method is essentially hidden in the parent class and it is not invoked unless the child class uses the super keyword with in the overriding method.

The method call resolution happends at run time and is termed as dynamic dispatch mechanism.

In the above example you can see that even though a1 is a type of animal it runs the eat method in the lion class. The reason for this is in compile time the check is made on the reference time.

However in the run time JVM figures out the object type and would run the method that belongs to that particular object.

```
Ex:- class Animal{
        void eat(){
     System.out.println("Animals eat both plants
                                & flesh");
     }
}
class lion extends Animal{
        void eat{
        super.eat();
        System.out.println("Lion eats flesh because
                            they are convivore");
        }
public static void main(String args[]){
   Animal a=new Animal();
      a.eat();
   Animal al=new Animal();
      al.eat();
   }
}
}
```

o/p:- Animal eat both plants & flesh
Lion eats flesh

Abstract Class:-

A class which is declared as abstract is known as abstract class. It can have abstract

and non-abstract methods. It needs to be extended and its method implemented. It can not be instantiated.

An abstract class must be declared with an abstract keyword, It can have constructors and static methods also, It can have final methods which will force the sub class not to change the body of the method.

Syntax:- abstract class <classname> {
}

Ex:-
```
abstract class Animal{
    void eat(){
        System.out.println("Animal eat both plants & flesh");
    }
}
class Lion extends Animal{
    void eat(){
        super.eat();
        System.out.println("Lion eats flesh");
    }
    public static void main(String args[]){
        Animal s1=new Animal();
        s1.eat();
    }
}
```

o/p:- Compile Time Error

Final Keyword:-

The final keyword in java is used to restrict the user, the java final keyword can be used in many contexts like → variable
→ method
→ class.

The final keyword can be applied with the variable a final variable that has no value, it is called blank final variable or an initialized final variable. It can initialized in the constructor only, the blank final variable can be static also which will be initialized static block only.

Syntax:- final int i=5;
i++;

Blank final variable

final int i;
i++;
i=5;

Final Method:-

```
class A{
  final void run(){
    System.out.println("This is a final method");
  }
}
class B extends A{
  void run(){
    System.out.println("This is illegal");
  }
}
```

```
public static void main(String args[]){
    B s1=new B();
    s1.run();
  }
}
```

o/p:- This is final method

If you make any method final you can't override that method.

Final class:-

If you make any class final you can't extend it.

Syntax:-

```
final class <class name>{
  ------}

class <child class> extends <parent class>{
...}
```

Ex:-

```
final class A{
    void run(){
    System.out.println("This is final method");
  }
}
class B extends A{
    void main(){
    System.out.println("This is illegal");
  }
  public static void main(String args[]){
    B s1=new B();
    s1.run();
  }
}
```

o/p:- Compile time error

Final Method using Inheritance:-

The final method is inherited. We can't override that final method.

Eg:-
```
class A{
    final void run(){
        System.out.println("Final method using inheritance");
    }
}
class B extends A{
    public static void main(String args[]){
        new B().run();
    }
}
```

o/p:- Final method using Inheritance.

Interface:-

An interface in Java is a blue print of a class. It has static constants and abstract methods. The interface in Java is a mechanism to achieve abstraction. They can be only abstract methods in the Java interface not method body. It is used to achieve abstraction and multiple Inheritance in Java.

Interface can have abstract methods and variables and "It can not have a method body". Java Interface represents IS a relation.

```
interface JNTUA{
    public void bTech();
    public void mTech();
}
class SJCET implements JNTUA{
    public void bTech(){
        System.out.println("Engineering");
    }
    public void mTech(){
        System.out.println("Masters");
    }
    public static void main(String args[]){
        SJCET stu=new SJCET();
        stu.bTech();
        stu.mTech();
    }
}
```

o/p:- Engineering
Masters

Uses of Java Interface:-

There are mainly 3 reasons to use interface.

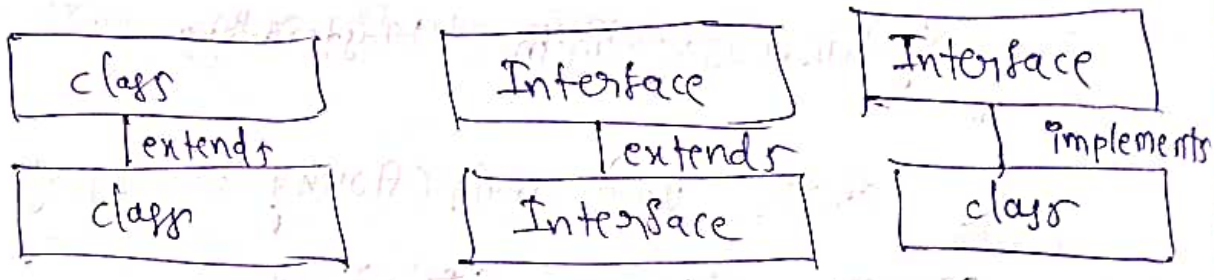
(1) It is used to achieve abstraction.

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

Declaring an Interface:-

An interface is declared by using the interface keyword. It provides total abstraction that means all the methods in an interface are declared with an empty body and all the fields are public, static, and final by default.

A class that implements an interface must implements all the methods declared in the interface.



Extending Interface:

Syntax:- interface A {

```
    ========();// Declaring methods
}
interface B extends A {
    ========();// Declaring methods
}
```

Eg:-

```
interface A {
public void funA();
}
Interface B extends A {
    public void funB();
}
```

```
class C implements B {
    public void funA() {
        System.out.println("This is funA");
    }
    public void funB() {
        System.out.println("This is funB");
    }
}

public class InterfaceDemo {
    public static void main(String args[]) {
        C s1=new C();
        s1.funA();
        s1.funB();
    }
}
```

Ex:-

```
interface TATA {
    public void Nexon();
}
interface TATAev extends TATA {
    public void NexonEV();
}
class Showroom implements TATAev {
    public void Nexon() {
        System.out.println("gets Nexon");
    }
```

```
public void Nexon4V(){
    System.out.println("Gets Nexon tv");
}
}

public class Agent {
    public static void main(String args[]){
        Showroom buyer=new Showroom();
        buyer.Nexon();
        buyer.Nexon4V();
    }
}
```

Extending Interface:-

* The Interface A has an abstract method function A. The interface B extends the Interface A and has an abstract method function B.

The class C Implements the Interface B.

* The main method class an object of class C is created then the methods function A and function B called extending interface.

Def^{n}:- An interface containing variables & methods like a class but the methods in an interface are abstract by default unlike a class. An interface extends another interface like a class implements an interface in interface inheritance.

Nested Interface:-

An interface declare within the another interface or class is known as Nested Interface. The nested interfaces are used to group related interfaces. So, that they can be easy to maintain. The nested interface must be referred by the outer interface or class.

* It can't be accessed directly.

The nested interface must be public. If it is declared inside the interface but it can have the any access modifier if declared within the class.

* Nested interface are declared static

Interface Variables:-

* Interface variables are static because Java interfaces can not be instantiated on their own. The value of the variable must be assigned in a static context in which no instance exists.
* The final modifier ensures the value assigned to the interface variable is a true constant that can't be re-assigned. In other words, interfaces can declare only constants not instance variables.

Object class:-

The object class is the parent class of all the classes in java by default. In other words, It is the topmost class of java.

The object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee, student etc. We can use object class reference to refer that object.

```
Eg:- Object obj = getObject();
     // we don't know what object will be returned
        from this method.
```

Nested Interface:-

```
Syntax:- interface interface.name{
         ------> method s/ only declaring.
         interface nested interface.name{
         ---------
         }
}
```

using class:-

```
class interface.name {

    interface nested Interface.name{

    }
}
```

Ex:- (1)

```
interface showable{
    void show();
    interface message{
        void msg();
    }
}
class Test Nested Interface 1 implements
                        showable
{
    message{
        public void msg(){
            System.out.println("Hello nested interface");
        }
    public static void main(String args[]){
    showable.message message=new TestNestedInterface1();

    message.msg();
    }
}
```

Output:- Hello nested interface

(ii)
```
class A{
    interface message{
        void msg();
    }
class TestNestedInterface2 implements A.message{
    public void msg(){
    System.out.println("Hello nested Interface");
    }
public static void main(String args[]){
    A.message message=new TestNestedInterface2();
    Message.msg();
}}
```

output:- Hello Nested Interface

Interface Variables:-

Eg:-
```
Interface m1{   // abstract
    void print();
    a=10;   // public static final
}
class Test implements m1,m2{
    public void print(){
        System.out.println(m1.a);
        System.out.println(m2.a);
    public static void main(String args[]){
        new Test().print();
}}
```