Swami Sahajanand College of Computer Science

B.C.A. SEM-V[NEP]

Subject: CORE JAVA
Major (Core) - 25617

UNIT 3

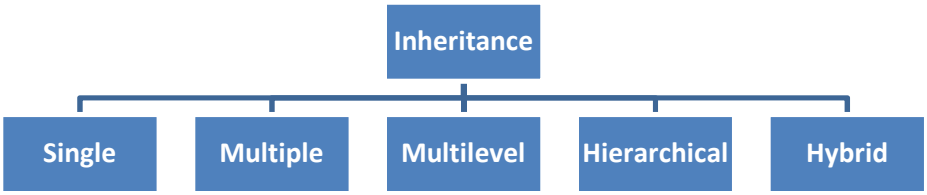
INHERTANCE AND INTERFACE

- Inheritance Basic, Types of Inheritance.
- Method Overriding. Run Time Polymorphism: Dynamic Method Dispatch.
- Abstract Method and Class.
- Interfaces: Defining Interface, Implementing Interface.
- Implementation of Multiple and Hybrid Inheritance using Interface. Extending Interface

Inheritance Basic, Types of Inheritance.

Inheritance **:**

- Inheritance is the concept of object oriented programming language.
- It provides the concept of reusability.
- "The mechanism of deriving a new class from an old one is called inheritance or derivation."
- In other words we can say when one class gets the properties of another class it is called inheritance.
- Old class is known as base class or parent class or super class.
- New class is known as derived class or subclass or child class.
- Derived class inherits all of the properties from the base class.
- **Types of Inheritance:-**



- How we can define derived class?
- Derived class can be defined by specifying its relationship with the base class in addition to its own details.

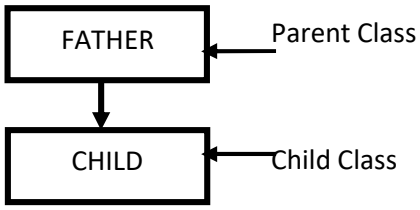
- **Syntax** to define derived class:-

```
class sub_class_name extends super_class_name
{
        ……
        …… //Members of derived class
}
```

- Keyword **extends** signifies that the properties of the super_class_name are inherited to the sub_class_name.
- **Example**:-

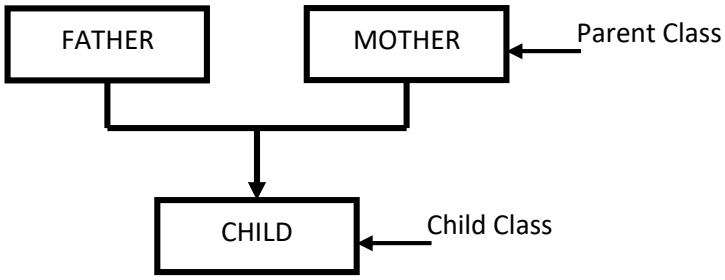
```
class ABC extends  XYZ
{
      members of ABC
}
```

- **<u>Single Inheritance:-</u>**
  - "A derived class with only one base class is called single inheritance".
  - Example:-



- In above example CHILD class gets the all or some of the properties from the parent class FATHER.

- **Multiple Inheritance:-**
- "When derived class has more than one base class it is called multiple inheritance."
- In java it is not possible to implement multiple inheritance directly.
- We can use interface to defining multiple inheritance.
- **Example**:-

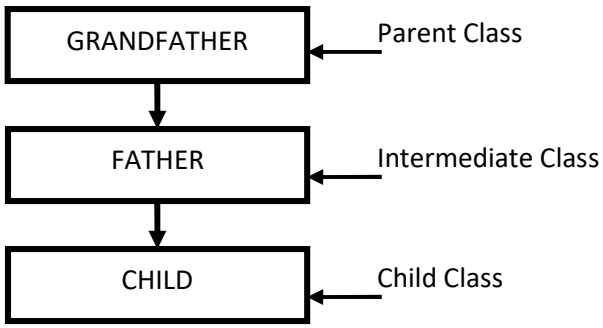


- In above example derived class CHILD gets some or all of the properties of base class FATHER & MOTHER.
- Such as CHILD inheriting physical features of one parent & intelligence of another.

- Defining multiple inheritance:-

```
class child implements interface_name extends super_class_name
{
        Members of child class
}
```

- **Multilevel inheritance:-**
- "The mechanism of deriving a new class from another derived class is called multilevel inheritance."
  - Example:-

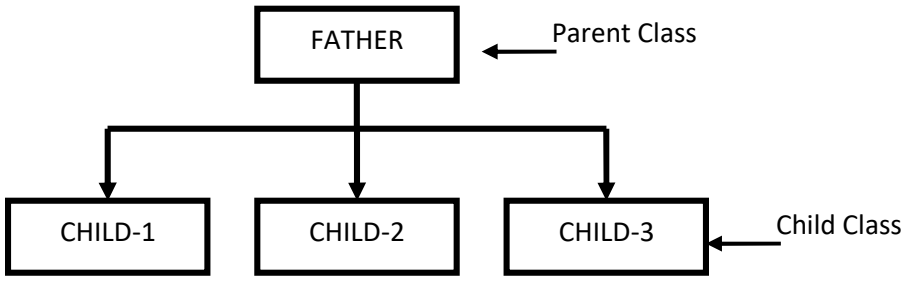


- In above example FATHER gets the properties from the GRANDFATHER and CHILD gets the properties of FATHER and GRANDFATHER both.

- Defining multilevel inheritance:-

```
class GRANDFATHER  //Base class
{
      Members of GRANDFATHER
}
class FATHER extends GRANDFATHER  //Intermediate base class
{
      Members of FATHER
}
class CHILD extends FATHER //Derived class
{
      Members of CHILD
}
```

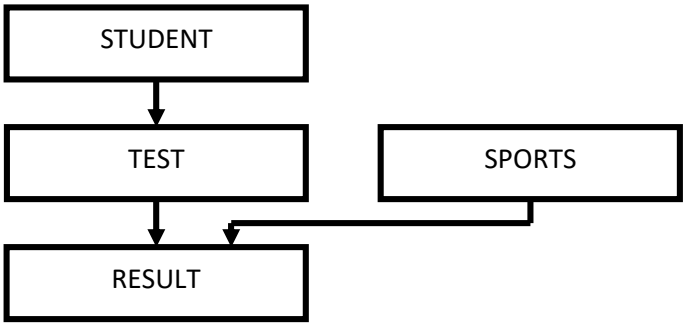
- **Hierarchical inheritance:-**
- "Properties of one class may be inherited by more than one class is called hierarchical inheritance."
- Example:-



- Defining Hierarchical inheritance:-

```
class FATHER
{
      Member of FATHER
}
class CHILD-1 extends FATHER
{
      Member of CHILD-1
}
class CHILD-2 extends FATHER
{
      Member of CHILD-2
};
class CHILD-3 extends FATHER
{
      Member of CHILD-3
}
```

- **Hybrid inheritance:-**
- "Combination of two or more type of inheritance to design a program is called hybrid inheritance."
- Example:-



- In above example we define multilevel & multiple inheritance together so it becomes hybrid inheritance.

Method Overloading:

- "Method name is same but method signature is different is known as method overloading."
- Another powerful object-oriented technique is method overloading.
- Method overloading enables you to specify different types of information (parameters) to send to a method.
- To overload a method, you declare another version with the same name but different parameters.
- The compiler keeps up with the parameters for each method along with the name.
- When a call to a method is encountered in a program, the compiler checks the name and the parameters to determine which overloaded method is being called.
- Ex :-

```
class OverloadDemo
{
        // Overload test for no parameters.
    void test()
        {
                System.out.println("No parameters");
        }

        // Overload test for two integer parameters.
        void test(int a, int b)
        {
                System.out.println("a and b: " + a + " " + b);
        }

        // overload test for a double parameter

        void test(double a)
        {
                System.out.println("Inside test(double) a: " + a);
```

```
        }
        int test(int x,int y,int z)
        {
                return x*y*z;
        }

}

class MethodOverloading
{
        public static void main(String args[])
        {
                OverloadDemo ob = new OverloadDemo();
                ob.test();
                ob.test(10, 20);
                ob.test(20.5); // this will invoke test(double)
                int n;
                n=ob.test(5,10,15);
                System.out.println("multiplication="+n);
        }
}
```

Output :

```
No parameters
a and b: 10 20
Inside test(double) a: 20.5
multiplication=750
```

Explain Method overriding:

- "When same method signature is used in the subclass and the superclass then this is called as method overriding."
- When an overridden method is called from within a subclass, it will always refers to the version of that method defined in the subclass and will hide the version of that method defined in the superclass.
- A subclass can define a method that has exactly the same method signature as a method in its superclass.
- In that case, the method in the subclass overrides the method in the method in the superclass.
- Prog :-

```
class A
{
    int i, j;
    A(int a, int b)
    {
            i = a;
            j = b;
    }
    // display i and j
    void show()
```

```
    {
        System.out.println("i and j: " + i + " " + j);
    }
}
class B extends A
{
    int k;
    B(int a, int b, int c)
    {
        super(a, b);
        k = c;
    }
    // display k – this overrides show() in A
    void show()
    {
        //super.show();
        System.out.println("k: " + k);

    }
}
class Override
{
    public static void main(String args[])
    {
        B subOb = new B(1, 2, 3);
        subOb.show(); // this calls show() in B
    }
}
```

O/P→ k: 3

Runtime polymorphism. Or Dynamic method dispatch:

- "Runtime polymorphism is used to avoid method overriding."
- Dynamic method dispatch is the mechanism by which a call to an overridden function is resolved at run time.
- It implements runtime polymorphism.
- A superclass reference variable can refer to a subclass object.
- When an overridden method is called through a superclass reference, java determines which version of that method to execute based upon the type of the object being referred to at runtime.
- It is the type of the object being referred to that determines, which version of an overridden method will be executed.
- Therefore, if a superclass contains a method that is overridden by a subclass, then different types of objects are referred through a superclass.

```
class A
{
    void callme()
    {
```

```
                System.out.println("Inside A's callme method");
        }
}
class B extends A
{
        // override callme()
        void callme()
        {
                System.out.println("Inside B's callme method");
        }
}
class C extends A
{
        // override callme()
        void callme()
        {
                System.out.println("Inside C's callme method");
        }
}
class Dispatch
{
        public static void main(String args[])
        {
        A a = new A(); // object of type A
        B b = new B(); // object of type B
        C c = new C(); // object of type C
        A r; // obtain a reference of type A
        r = a; // r refers to an A object
        r.callme(); // calls A's version of callme
        r = b; // r refers to a B object
        r.callme(); // calls B's version of callme
        r = c; // r refers to a C object
        r.callme(); // calls C's version of callme
        }
}
```

Abstract classes and method:

- "If we don't want to allow creating a super class's object then we create that class as abstract class."
- An abstract class is typically used to declare a common set of methods for a group of classes when there are no useful implementations.
- The advantage of the abstraction is that it allows you to write code to as per the requirement.
- Methods defined in an abstract class can be declared abstract.
- An abstract method is declared without any implementation.

```
// abstract class prog.
abstract class A
{
```

```
        abstract void callme();
        // concrete methods are still allowed in abstract classes
        void callmetoo()
        {
                System.out.println("This is a concrete method.");
        }
}
class B extends A
{
        void callme()
        {
                System.out.println("B's implementation of callme.");
        }
}
class AbstractDemo
{
        public static void main(String args[])
        {
                //A a=new A();
                B b = new B();
                b.callme();
                b.callmetoo();
        }
}
```

Interfaces

- An interface defines a protocol of behavior that can be implemented by any class anywhere in the class hierarchy.
- An interface defines a set of methods but does not implement them.
- A class that implements the interface agrees to implement all the methods defined in the interface.
- An interface is a named collection of method definitions (without implementations).
- An interface cannot implement any methods.
- A class can implement many interfaces but can have only one super class.

Important points about Interface

- Java does not support multiple inheritances.
- We can implement multiple inheritances in java indirectly, by using interface.
- We can define Interface by using “Interface” keyword.
- Syntax for interface,

```
Interface interface_name
{
}
```

- Example,

```
Interface student
{
```

```
}
```

- We can only define method in interface; we cannot implements method in interface.
- It means we cannot write any line of code in method in interface.
- Data member in interface must define as constant.
- Use final keyword to define constants variable.
- Interface can also be inherited by other classes.
- To inherited interface "Implements" keyword is used.
- Syntax,

```
Class class_name implements interface
{
}
```

- Example,

```
Class subject implements student
{
}
```

Advantages of Interface

- Interface provides a means to define the structure for a class without worrying about the implementation details.
- This is simple benefit can make large projects much easier to manage.
- Ones interfaces have been designed, the class development can take place without worrying about communication among classes.
- Another important use of interfaces is the capacity for a class to implement multiple interfaces.
- The major difference between inheriting multiple interfaces and true multiple inheritance is that the interface approach enables you to inherit only method description, not implementations.

Packages

- **"A package is a collection of related classes and interfaces providing access protection and namespace management."**
- The syntax for the package statement follows:
- package Identifier;
- The classes and interfaces that are part of the Java platform are members of various packages that bunch classes are: fundamental classes are in java.lang, classes for reading and writing (input and output) are in java.io, and so on.
- You can put your classes and interfaces in packages, too.
- Package provides reusability of code is one of the most important requirements because it saves time, effort also ensures consistency.
- A package allows creating a various class without concern that it will collide with some other class stored elsewhere.
- Packages are stored in a hierarchical manner and are explicitly imported into new class definition.

Basic points / Rules about Package

- Package program must be stored in directory, name of directory must be same as package name.
- To define package "Package" key word is used.
- Class in package must as public.
- All methods of class in package must also be defined as public.
- To use any class, any method of package in any other program, "Import" keyword is used.

- Syntax for define package is,
  Package package_name;
- Example,
  Package student;
- Syntax for import,
  Import package_name.class_name.mehtod_name;
  OR
  Import package_name.*;
- Example,
  Import student.*;

Advantages of package

- Package allows you to organize your classes into smaller units and make it easy to locate and use the appropriate class file.
- It helps to avoid naming conflicts, when you are working with number of classes it becomes difficult to decide on names of the classes and methods.
- At times, you would have to use the same name, which belongs to another class.
- A package basically, hides the class's data and methods in a larger way then on a class-to-class basis.
- A package mechanism is to partitioning the class name into more manageable manner.
- Package names can be used to identify your classes.
- It can be easily determined that these types are related.
- Built in package

  1. java.util - Includes classes like ArrayList, HashMap, Date, Scanner, and more.
  Example: import java.util.Scanner; (for reading user input).

  2. java.io - Includes classes for reading from and writing to files, such as File, BufferedReader, PrintWriter.
  Example: import java.io.File;

  3. java.lang - Automatically imported in every Java program. It includes fundamental classes such as String, System, Math, and Object.
  Example: String name = "Java";

  4. java.net - Includes classes for networking, like URL, Socket, etc.
  Example: import java.net.URL;

  5. java.time - Introduced in Java 8 for modern date and time handling.
  Example: import java.time.LocalDate;

Why Use Packages?

1. Organizing Code: Packages help group similar functionality into modules. For example, all database-related classes can go into a `com.example.database` package.

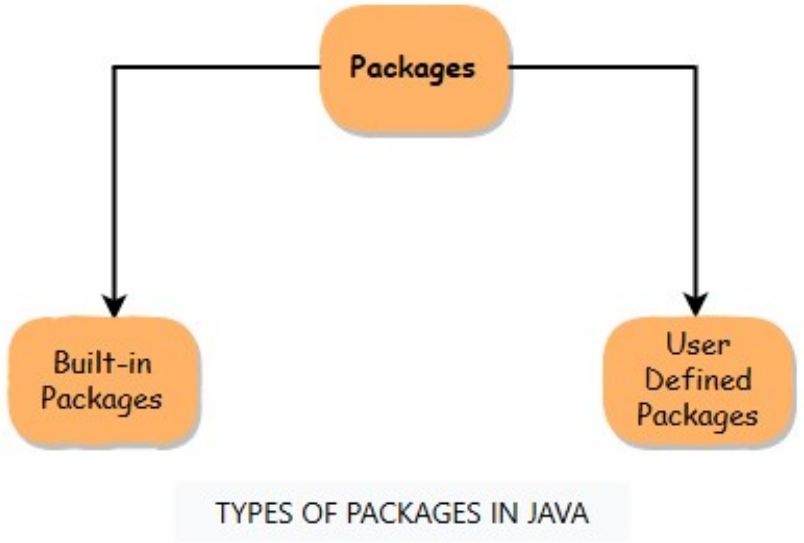
2. Avoiding Naming Conflicts: In large projects, it's possible to have classes with the same names. Packages provide namespaces, so two classes with the same name can exist in different packages.

3. Access Control: Java packages provide access control mechanisms. Classes, methods, and fields can have package-level visibility.

4. Reusability: Code can be organized into packages that can be reused across projects.

Types of Packages in Java
Java has two types of packages:

1. Built-in Packages: These are provided by the Java API (e.g., `java.util`, `java.io`, `java.lang`).
2. User-defined Packages: These are created by users to organize their classes and interfaces.



TYPES OF PACKAGES IN JAVA

What are built-in packages?

Built-in packages are provided for the program to reduce the burden. There are various in-built packages accessible in Java. Some of the built-in packages in Java are:

- ***Java.util:*** Holds the collection structures, some of these packages support classes, properties, random number generation classes. Classes like HashSet, LinkedList, Queue, HashMap, Collection, Arrays, Date, Optional, etc., are part of this package.

- ***Java.io:*** Produces classes for system input/output operations. Classes like BufferedInputStream, BufferedReader, CharArrayWriter, BufferedWriter, FileReader, InputStream, OutputStream, PrintStream, PipedReader, Serializable, etc., are part of this package.

- ***Java.lang:*** Provides classes and interfaces that are rudiments to the design of Java programming language. Classes like Float, Boolean, Integer, Long, String, StringBuffer, StringBuilder, etc are part of this package.

- ***Java.sql:*** Accommodates the classes and interfaces for obtaining and processing data stored in a database. Classes or interfaces like Struct, PreparedStatement, Wrapper, Connection, DriverAction, DriverManager, Driver, ResultSet, Statement, etc are part of this package.

What are user-defined packages?

User-defined packages are the packages created by the users. Users are free to create their packages. To create your package, you need to understand that Java uses a file system directory to store them, just like folders and sub-folders on your computer.

How to create/write packages in Java

We use the **package keyword** to create or define a package in Java programming language.

1. The package keyword must come before the package name.
2. The package name must be a solitary word. For example 'my package' should be written as 'mypackage'. There should be no space between my and the package.
3. The package name should use lower case notation.
4. The package statement should be the first line of statement in the program.

Program:

num1.java

```
package number_pack;
import java.util.Scanner;
public class num1
{
    public int n1;
    public void getn1()
    {
            Scanner sc = new Scanner(System.in);
            System.out.println("**********************");
            System.out.print("Enter Number 1 : ");
            n1 = sc.nextInt();
    }
    public void displayn1()
    {
    System.out.print("Value of Number 1 is .. " + n1);
    }
}
```

num2.java

```
package number_pack;
import java.util.Scanner;
public class num2
{
    public int n2;
    public void getn2()
    {
```

```
            Scanner sc = new Scanner(System.in);
            System.out.println("**********************");
            System.out.print("Enter Number 2 : ");
            n2 = sc.nextInt();
    }
    public void displayn2()
    {
    System.out.print("Value of Number 2 is .. " + n2);
    }
}
```

Sum_p.java

```
import number_pack.*;
class sum_p
{
    public static void main(String args[])
    {
            num1 a  = new num1();
            a.getn1();
            num2 b  = new num2();
            b.getn2();
            a.displayn1();
            System.out.println();
            b.displayn2();
            int t=a.n1+b.n2;
            System.out.println();
            System.out.print("Total is .." + t);
    }
}
```

```
/*
Output
number_pack> javac num1.java
number_pack> javac num2.java
> javac sum_p.java
> java sum_p
**********************
Enter Number 1 : 10
**********************
Enter Number 2 : 20
Value of Number 1 is .. 10
Value of Number 2 is .. 20
Total is ..30
*/
```