What is **Inheritance** ?

The process by which one class acquires the properties (data members) and functionalities (methods) of another class is called inheritance.

In OOP, we often organize classes in **hierarchy** to avoid **duplication** and reduce **redundancy**. The classes in the lower hierarchy inherit all the variables (attributes/state) and methods (dynamic behaviors) from the higher hierarchies.

A class in the lower hierarchy is called a **subclass** (or derived, child, extended class). A class in the upper hierarchy is called a **superclass** (or base, parent class).

By pulling out all the common variables and methods into the superclasses, and leaving the specialized variables and methods in the subclasses, redundancy can be greatly reduced or eliminated as these common variables and methods do not need to be repeated in all the subclasses. **Re usability** is maximum.

A subclass inherits all the member variables and methods from its superclasses (the immediate parent and all its ancestors). It can use the inherited methods and variables as they are. It may also **override** an inherited method by providing its own version, or **hide an inherited variable** by defining a variable of the same name.

It is important to note that a subclass is not a "**subset**" of a superclass. In contrast, subclass is a "**superset**" of a superclass. It is because a subclass inherits all the variables and methods of the superclass; in addition, it extends the superclass by providing more variables and methods.

Inheritance is a process from **generalization → specialization**. (in a top down manner)

It represents

- **IS - A** Relationship between the classes / objects

The keyword used for inheritance

- **extends**

Summary : Sub class IS-A super class , and something more (additional state,

additional methods) and something modified(behavior via method overriding)

Examples of inheritance hierarchy

Person,Student,Faculty

Emp,Manager,SalesManager,HRManager,Worker,TempWorker,PermanentWorker

Shape, Circle,Rectangle,Cyllinder,Cuboid

BankAccount ,LoanAccount,HomeLoanAccount,VehicleLoanAccount,

Student,GradStudent,PostGradStudent

Fruit , Apple , FujiApple

Types of inheritance

1. Single inheritance

eg : class A {...} class B extends A{...}

2. Multi level inheritance

eg : class A{...} class B extends A{...} class C extends B{...}

3. Hierarchical inheritance

When more than one class inherits a same class then this is called hierarchical inheritance.

eg : class A{..} class B extends A{ ..} class C extends A{...} class D extends A {...}

4. Multiple inheritance

- NOT supported in Java (For simplicity)

Eg. class A extends B,C{...}

- Assuming B & C are classes
- Results in compiler err

It avoids Diamond problem (ambiguity issue)

- Suppose we have two classes B and C inheriting from A. Assume that B

and C are overriding an inherited method and they provide their own implementation. Now D inherits from both B and C doing multiple inheritance. D should inherit that overridden method. BUT which overridden method will be used? Will it be from B or C? Here we have an ambiguity. This is known as Diamond Problem.

To avoid this problem , Java does not support multiple inheritance through classes.

Constructor invocations in inheritance hierarchy

- Refer to a diagram
- " day4_help\diagrams\constr invocation in inheritance hierarchy.png"

"**super**" is a keyword related to inheritance.

Usages

1. To access super class's visible members(data members as well as methods)

eg : package p1;

```
class A { void show(){System.out.println("in A's show");}}
```

```
package p1;

class B extends A {

  //overriding form in a sub class

 void show(){

   super.show(); //invokes A's show.

   System.out.println("in B's show");

}

}
```

eg : B b1=new B();b1.show();

2. To invoke immediate super class's matching constructor

- super(args)

- It is accessible only from sub class constructor

- Java compile implicitly adds super() in the sub class constructor, if its not invoked explicitly.

Regarding this & super keywords

this(...) implies invoking constructor from the same class.

super(...) implies invoking constructor from the immediate super class.

Note -

1. You can use this(...) or super(..) , only from the constructor.

2. It has to be 1st statement in the constructor.

3. Any constructor can never have both this() & super().

Enter polymorphism

It means one name with multiple (changing) forms.

What is method binding ?

- Linking a method call to actual method definition.

Different forms of polymorphism

1. **static or compile time polymorphism**

- Uses early binding

- Exact form of the method to be used for execution , is resolved by java compiler , based on the type of the reference.

- It is achieved via method overloading

About **Method Overloading**

- Overloaded methods can be in same class or in the inheritance hierarchy.
- They have same method name
- Method signature must be different , meaning either no of arguments or type of arguments or both need to be different.
- Method return type is ignored by compiler , while resolving overloaded methods.
- You can overload private or final or static methods.

```
Eg. class TestMe {

void test(int i,int j){...}

void test(int i) {..}

void test(double i){..}

void test(int i,double j,boolean flag){..}

int test(int a,int b){...}     //javac error

}
```

2. Dynamic or run time polymorphism

- Uses late binding

- The form of method to be executed is decided **at runtime**, based on the object's **actual type**, not the reference type(dynamic method dispatch)

- Achieved via - method overriding

What is Method Overriding ?

When we declare the same method in child class which is already present in the parent class then this is called method overriding. In this case when we call the method from child class object, the child class version of the method is called.

eg : class Fruit {String taste() {return "No Specific Taste";}

class Orange extends Fruit {String taste() {return "Sour in Taste";}

In main method -

Fruit f=new Orange();

System.out.println(f.taste());

- It will print - Sour in Taste

Important points regarding method overriding

- There is NO "virtual" keyword in java. (meaning all java methods are implicitly virtual)
- All java methods can be overridden to achieve run time polymorphism, if they are not marked as **private or static or final**
- **private or static or final or overloaded methods use early binding** (i.e method call is resolved at compile time , using the type of the reference)

Method overriding related

1. Overridden and overriding methods can't exist in the same class , they can only exist in an inheritance hierarchy (in super class & sub class)
2. Overriding form of the method should have
- same method name, same signature, return type must be same or can be co-variant (meaning sub type of the return type of the super class method)

Eg. of **co-variance**

```
package p1;

class A {

    A getInstance() {return new A();

    }

}
```

```
package p1;

class B extends A

{

    B getInstance()

      {

           return new B();

      }

}
```

3. Overriding form of the method can't restrict the access specifier.

4. It can’t throw any **new** or **broader checked exceptions**.

   Can add any new unchecked exceptions

   Can add any subset or sub-class of checked exceptions.

IMPORTANT Statements

Java compiler resolves method binding(early binding) using type of reference.

JVM (run time) resolves method binding (late binding) using the actual type of object