Types of Inheritance and examples

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called **inheritance**. The aim of inheritance is to provide the re-usability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.

Child Class:

The class that extends the features of another class is known as child class, sub class or derived class.

Parent Class:

The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

> Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.
> Inheritance allows us to reuse of code, it improves re-usability in your java application.
> Note: The biggest **advantage of Inheritance** is that the code that is already present in base class need not be rewritten in the child class.

This means that the data members(instance variables) and methods of the parent class can be used in the child class as.

Syntax: Inheritance in Java

To inherit a class we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class.

```
class XYZ extends ABC
{
}
```

Inheritance Example

In this example, we have a base class `Teacher` and a sub class `PhysicsTeacher`. Since class `PhysicsTeacher` extends the designation and college properties and work() method from base class, we need not to declare these properties and method in sub class.
Here we have collegeName, designation and work() method which are common to all the teachers so we have declared them in the base class, this way the child classes like `MathTeacher`, `MusicTeacher` and `PhysicsTeacher` do not need to write this code and can be used directly from base class.

```
class Teacher {
   String designation = "Teacher";
   String collegeName = "Beginnersbook";
   void does(){
        System.out.println("Teaching");
   }
}

public class PhysicsTeacher extends Teacher{
```

```
   String mainSubject = "Physics";
   public static void main(String args[]){
        PhysicsTeacher obj = new PhysicsTeacher();
        System.out.println(obj.collegeName);
        System.out.println(obj.designation);
        System.out.println(obj.mainSubject);
        obj.does();
   }
}
```

Output:

```
Beginnersbook
Teacher
Physics
Teaching
```

> Based on the above example we can say that `PhysicsTeacher` **IS-A** `Teacher`. This means that a child class has IS-A relationship with the parent class. This is inheritance is known as **IS-A relationship** between child and parent class

Note:

The derived class inherits all the members and methods that are declared as public or protected. If the members or methods of super class are declared as private then the derived class cannot use them directly. The private members can be accessed only in its own class. Such private members can only be accessed using public or protected getter and setter methods of super class as shown in the example below.

```
class Teacher {
   private String designation = "Teacher";
   private String collegeName = "Beginnersbook";
   public String getDesignation() {
        return designation;
   }
   protected void setDesignation(String designation) {
        this.designation = designation;
   }
   protected String getCollegeName() {
        return collegeName;
   }
   protected void setCollegeName(String collegeName) {
        this.collegeName = collegeName;
   }
   void does(){
        System.out.println("Teaching");
   }
}

public class JavaExample extends Teacher{
   String mainSubject = "Physics";
   public static void main(String args[]){
        JavaExample obj = new JavaExample();
        /* Note: we are not accessing the data members
         * directly we are using public getter method
         * to access the private members of parent class
         */
        System.out.println(obj.getCollegeName());
        System.out.println(obj.getDesignation());
        System.out.println(obj.mainSubject);
        obj.does();
```

```
    }
}
```

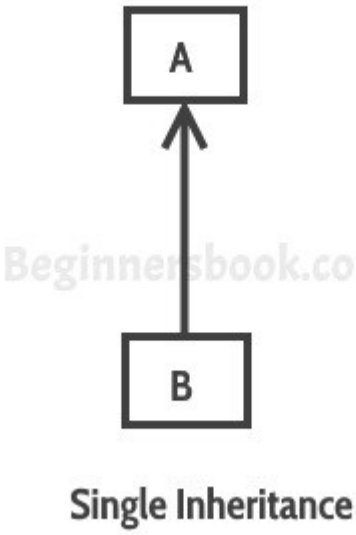
The output is:

```
Beginnersbook
Teacher
Physics
Teaching
```

The important point to note in the above example is that the child class is able to access the private members of parent class through **protected methods** of parent class. When we make a instance variable(data member) or method **protected**, this means that they are accessible only in the class itself and in child class. These public, protected, private etc. are all access specifiers and we will discuss them in the coming tutorials.

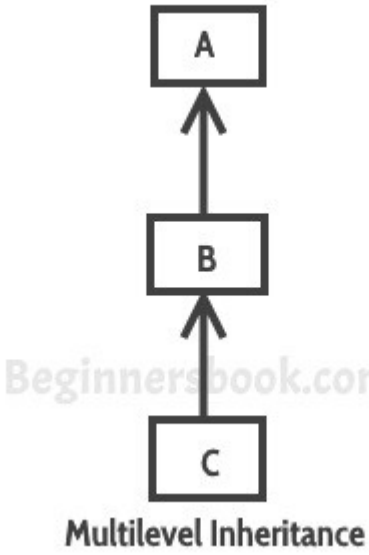
Types of inheritance

To learn types of inheritance in detail, refer: Types of Inheritance in Java.
Single Inheritance: refers to a child and parent class relationship where a class extends the another class.

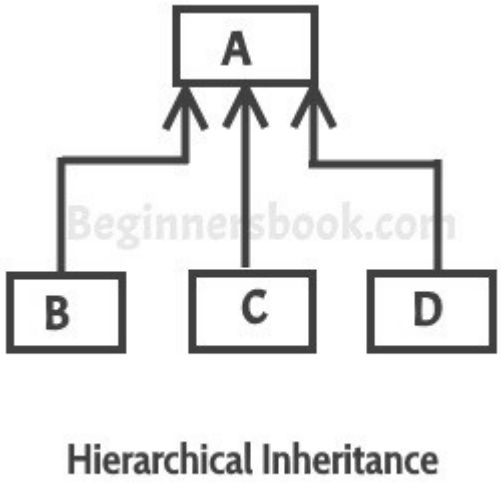


Single Inheritance

Multilevel inheritance: refers to a child and parent class relationship where a class extends the child class. For example class C extends class B and class B extends class A.

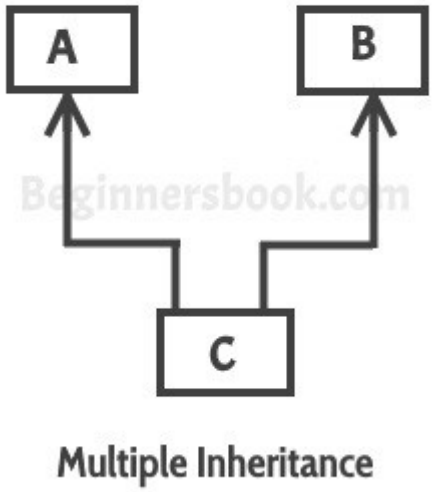


Hierarchical inheritance: refers to a child and parent class relationship where more than one classes extends the same class. For example, classes B, C & D extends the same class A.



Multiple Inheritance: refers to the concept of one class extending more than one classes, which means a child class has two parent classes. For example class C extends both classes A and B. Java

doesn’t support multiple inheritance.



Multiple Inheritance

Hybrid inheritance: Combination of more than one types of inheritance in a single program. For example class A & B extends class C and another class D extends class A then this is a hybrid inheritance example because it is a combination of single and hierarchical inheritance.

Constructors and Inheritance

constructor of sub class is invoked when we create the object of subclass, it by default invokes the default constructor of super class. Hence, in inheritance the objects are constructed top-down. The superclass constructor can be called explicitly using the super keyword, but it should be first statement in a constructor. The super keyword refers to the superclass, immediately above of the calling class in the hierarchy. The use of multiple super keywords to access an ancestor class other than the direct parent is not permitted.

```
class ParentClass{
   //Parent class constructor
   ParentClass(){
        System.out.println("Constructor of Parent");
   }
}
class JavaExample extends ParentClass{
   JavaExample(){
        /* It by default invokes the constructor of parent class
         * You can use super() to call the constructor of parent.
         * It should be the first statement in the child class
         * constructor, you can also call the parameterized constructor
         * of parent class by using super like this: super(10), now
         * this will invoke the parameterized constructor of int arg
         */
        System.out.println("Constructor of Child");
   }
   public static void main(String args[]){
        //Creating the object of child class
        new JavaExample();
```

```
   }
}
```

Output:

```
Constructor of Parent
Constructor of Child
```

Inheritance and Method Overriding

When we declare the same method in child class which is already present in the parent class the 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. However we can call the parent class method using super keyword as I have shown in the example below:

```
class ParentClass{
   //Parent class constructor
   ParentClass(){
        System.out.println("Constructor of Parent");
   }
   void disp(){
        System.out.println("Parent Method");
   }
}
class JavaExample extends ParentClass{
   JavaExample(){
        System.out.println("Constructor of Child");
   }
   void disp(){
        System.out.println("Child Method");
        //Calling the disp() method of parent class
        super.disp();
   }
   public static void main(String args[]){
        //Creating the object of child class
        JavaExample obj = new JavaExample();
        obj.disp();
   }
}
```

The output is :

```
Constructor of Parent
Constructor of Child
Child Method
Parent Method
```