MARRI LAXMAN REDDY
Institute of Technology and Management
(Approved By AICTE, New Delhi & Affiliated to JNTU Hyderabad)
Dundigal,Medchal(M),Hyderabad,Telangana, India – 500 043.
(UGC - AUTONOMOUS)

UNIT-II-PART-II

Inheritance, Packages and Interfaces – Inheritance basics, Using super, Creating a multilevel hierarchy, method overriding, Dynamic method dispatch, abstract classes, Using final with inheritance, Defining a package, Finding package and classpath, Access protection, importing packages, Defining an interface, implementing interface, applying interfaces, variables in interface and extending interfaces.

Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.

When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class.
- It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features.
- It is also called a base class or a parent class.

- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class.
- You can use the same fields and methods already defined in the previous class.

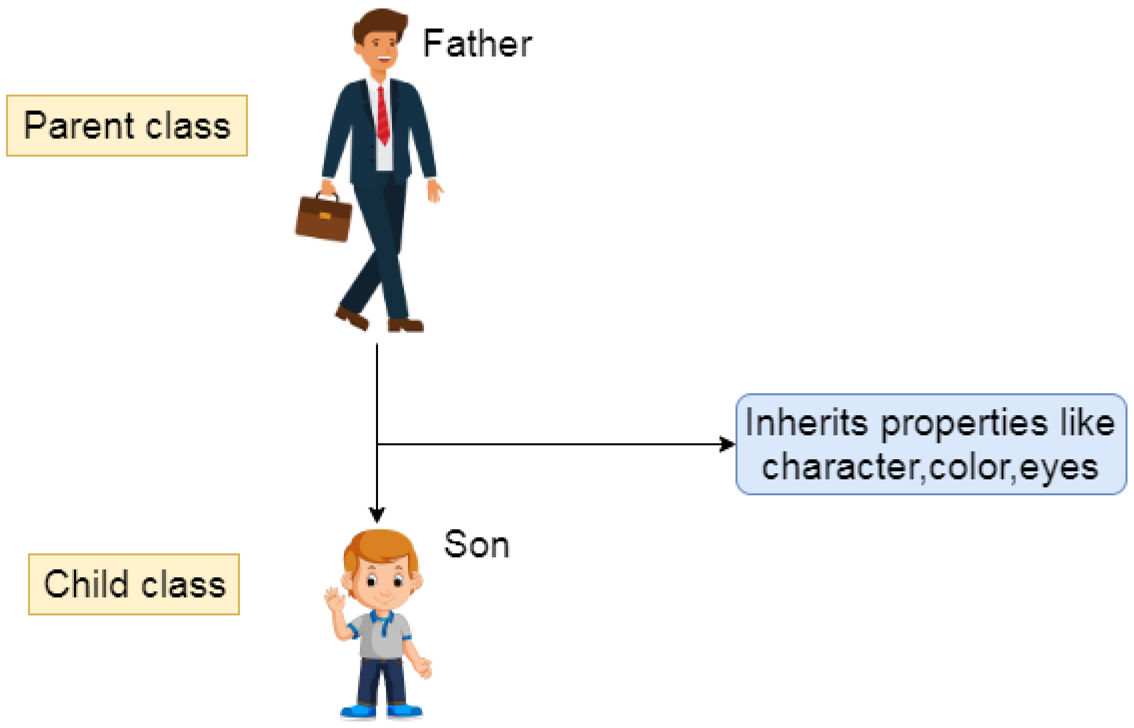
The syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
 {
  //methods and fields
 }
```

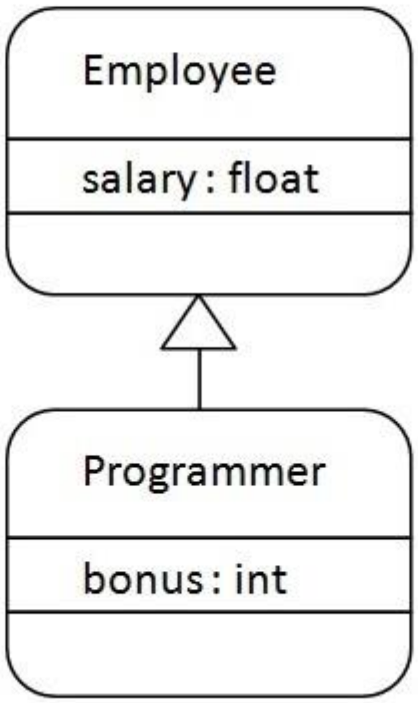
The **extends keyword** indicates that you are making a new class that derives from an existing class.

The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.



Java Inheritance Example



As displayed in the above figure, Programmer is the subclass and Employee is the superclass.

The relationship between the two classes is **Programmer IS-A Employee**. I

It means that Programmer is a type of Employee.

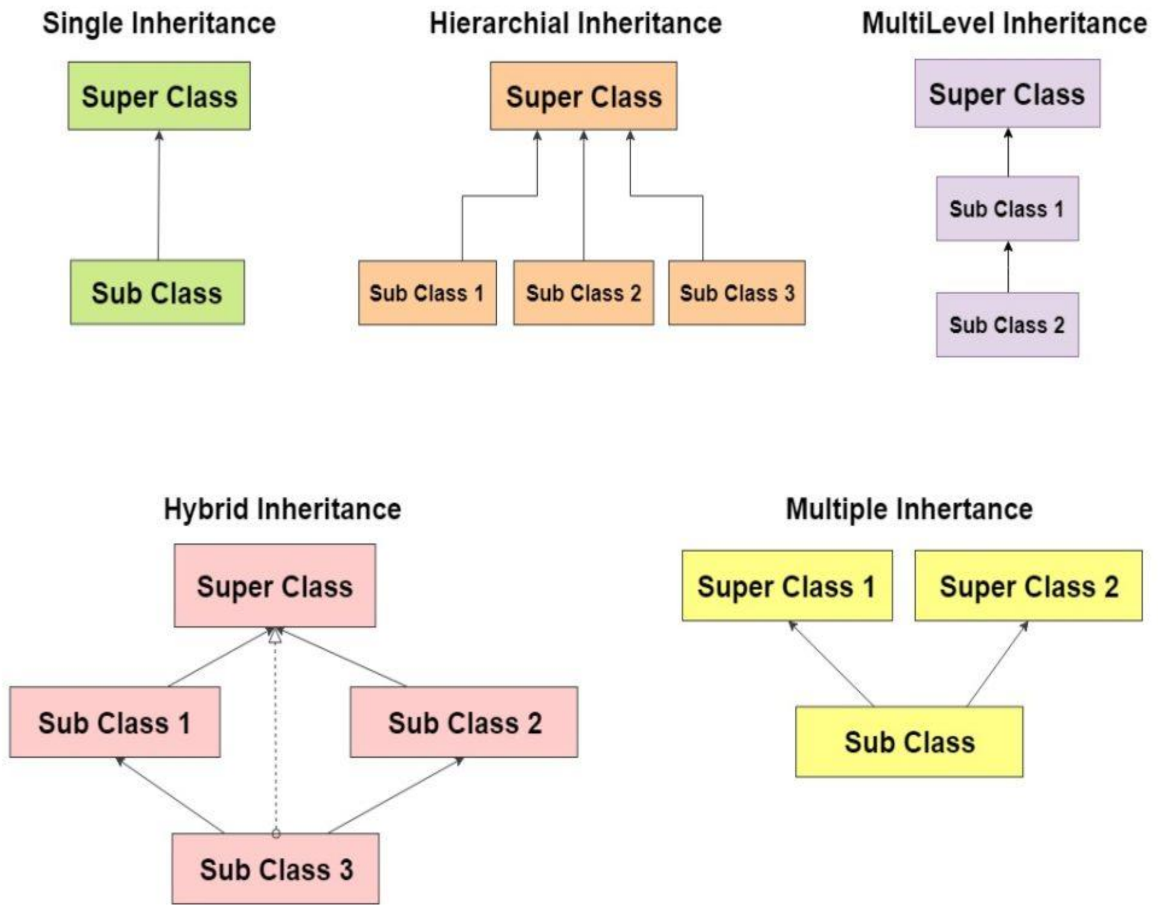
Example:-

```
class Employee
{
 float salary=40000;
}
class Programmer extends Employee
{
 int bonus=10000;
}

 class single
{
 public static void main(String args[])
{
 Programmer p=new Programmer();
 System.out.println("Programmer salary is:"+p.salary);
 System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

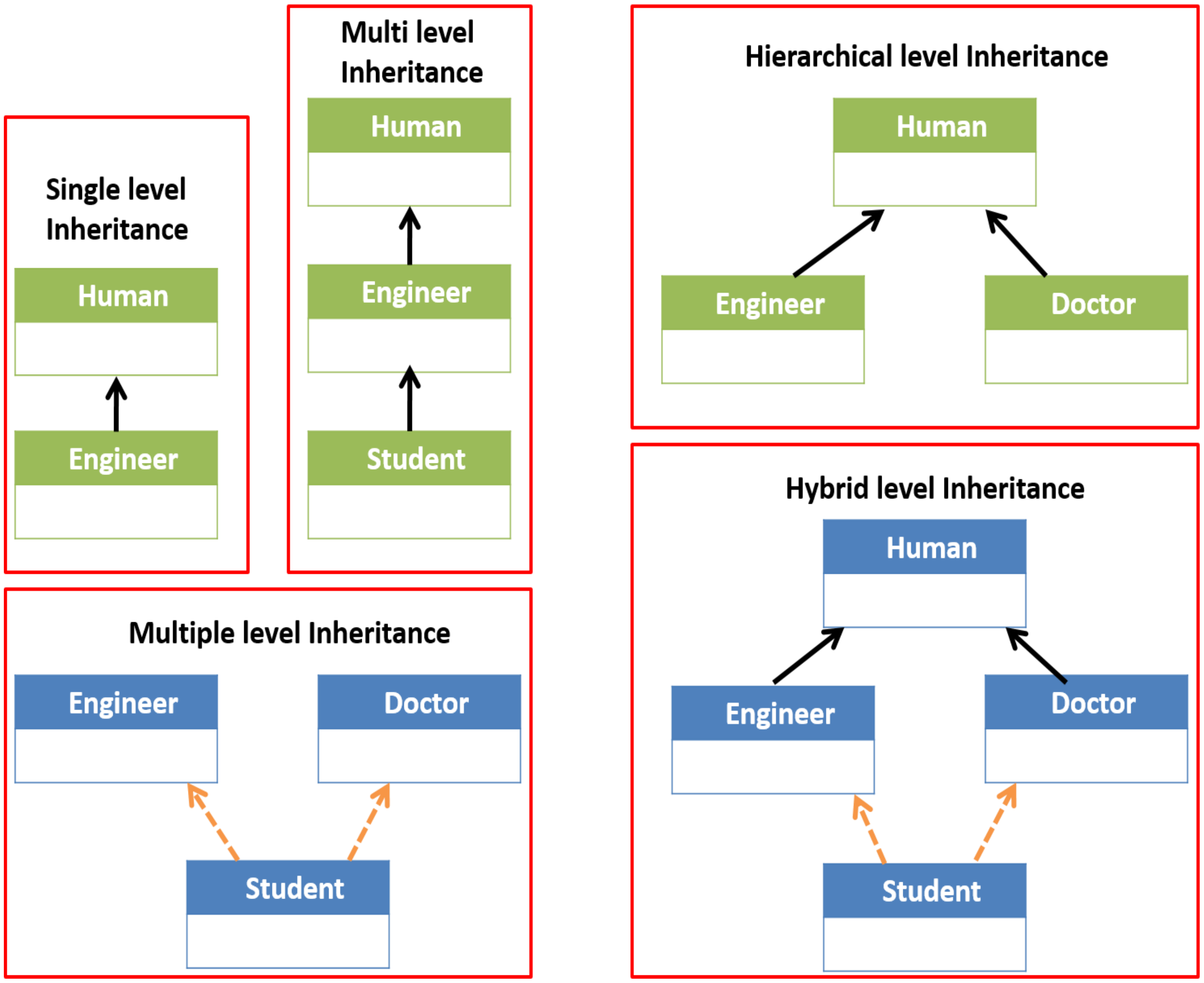
```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

In the above example, Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

Types of inheritance in java



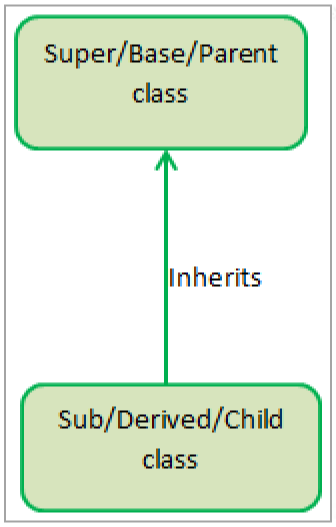
Example2:-

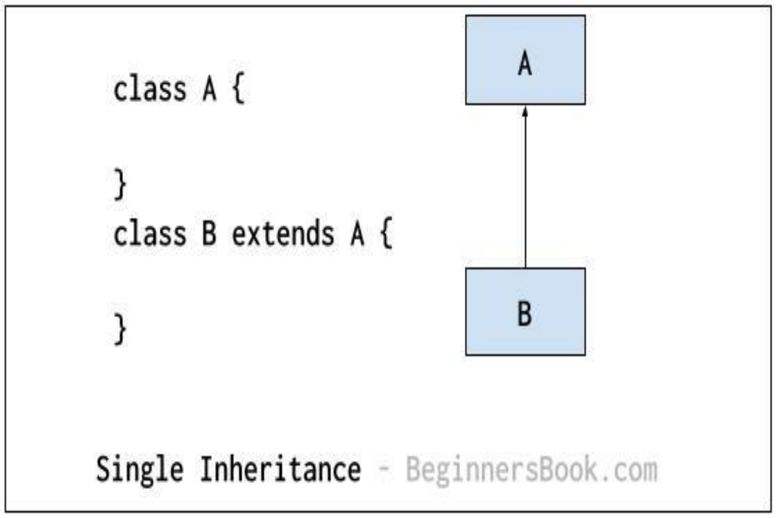

Single level Inheritance
Human
Engineer
Multi level Inheritance
Human
Engineer
Student
Hierarchical level Inheritance
Human
Engineer
Doctor
Multiple level Inheritance
Engineer
Doctor
Student
Hybrid level Inheritance
Human
Engineer
Doctor
Student

SingleLevel Inheritance

When a class inherits another class, it is known as a *single inheritance*.

In the example given below, Dog class inherits the Animal class, so there is the single inheritance.





```
class A {

}
class B extends A {

}
```

Example:-

```
class Animal
{
void eat()
{
System.out.println("eating...");
}
}
class Dog extends Animal
{
void bark()
{
System.out.println("barking...");
}
}
class TestInheritance
{
public static void main(String args[])
{
Dog d=new Dog();
d.bark();
d.eat();
}
}
```

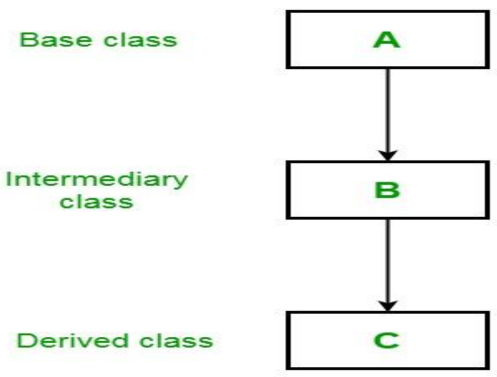
Output:-

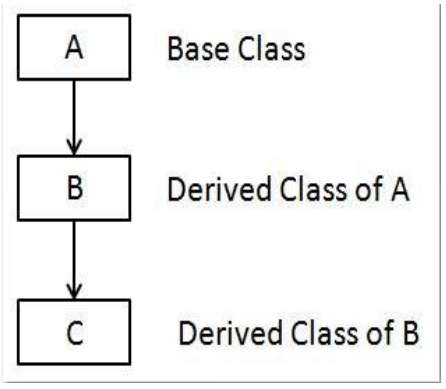
```
Barking

Eating
```

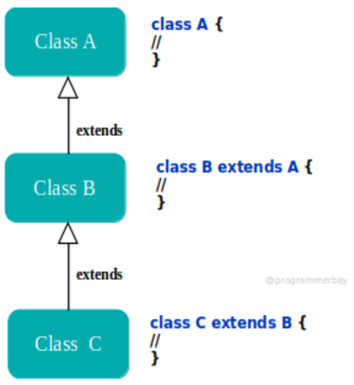
Multilevel Inheritance Example

In Multi-Level Inheritance in Java, a class extends to another class that is already extended from another class.

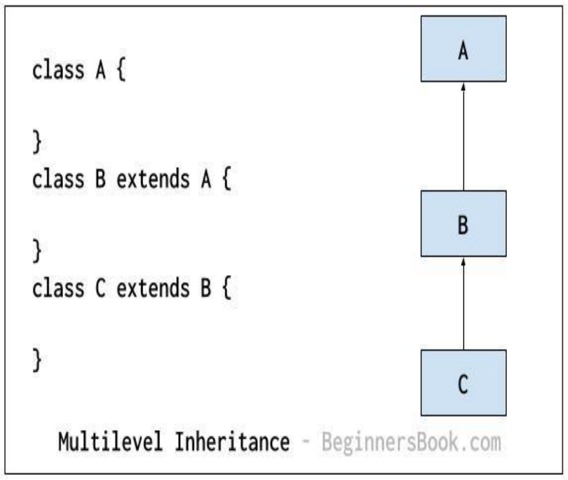
For example, if there is a class A that extends class B and class B extends from another class C, then this scenario is known to follow Multi-level Inheritance.







```
class A {
//
}
class B extends A {
//
}
class C extends B {
//
}
```



```
class A {

}
class B extends A {

}
class C extends B {

}
```

Multilevel inheritance example:-

```
class Shape
{
  public void display()
  {
     System.out.println("Inside display");
   }
}
class Rectangle extends Shape
{
  public void area()
  {
     System.out.println("Inside area");
   }
}
class Cube extends Rectangle
{
  public void volume()
  {
     System.out.println("Inside volume");
   }
}
public class Tester
{
  public static void main(String[] arguments)
  {
     Cube c = new Cube();
     c.display();
     c.area();
     c.volume();
   }
}
```

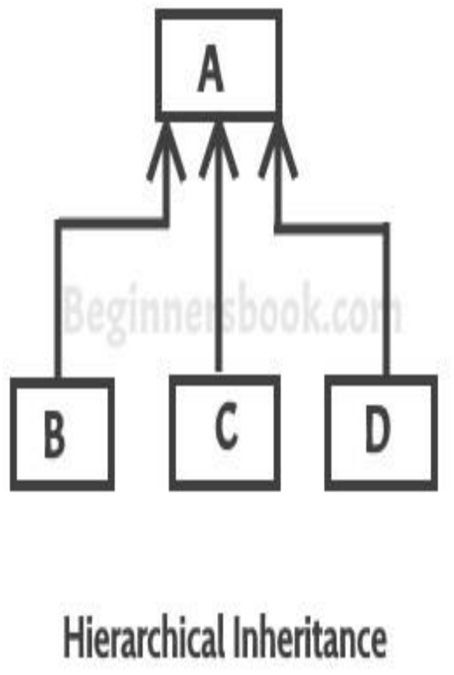
Output

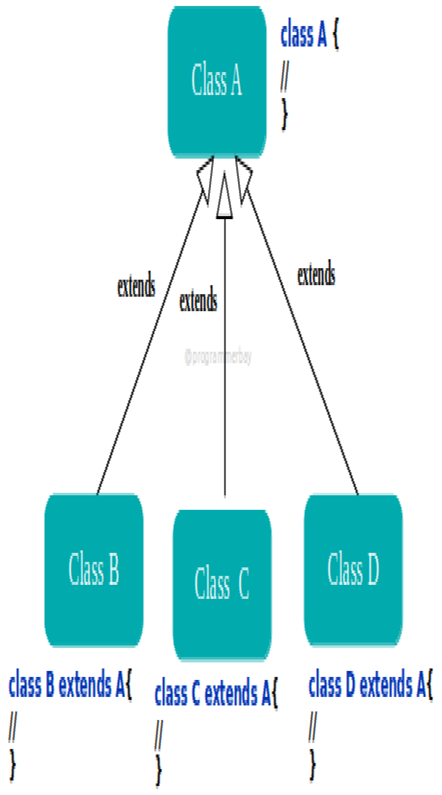
```
Inside display
Inside area
Inside volume
```

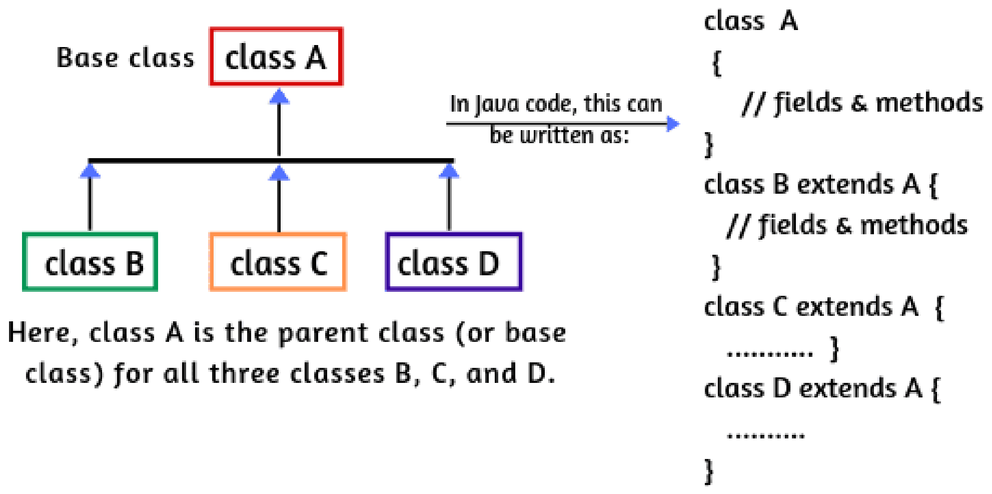
Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*

Hierarchical inheritance in Java is a type of inheritance in which the same class is inherited by more than one class.



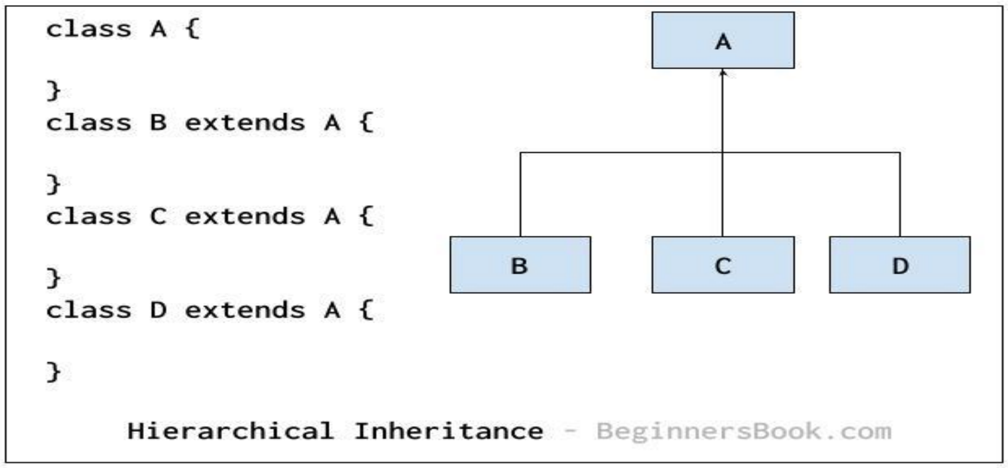




Here, class A is the parent class (or base class) for all three classes B, C, and D.

```
class  A
 {
    // fields & methods
}
class B extends A {
   // fields & methods
 }
class C extends A  {
   ........... }
class D extends A {
   ..........
}
```

Fig: Hierarchical Inheritance in Java



```
class A {

}
class B extends A {

}
class C extends A {

}
class D extends A {

}
```

Hierarchical Inheritance - BeginnersBook.com

Hierarchical inheritance example:-

```
public class A

{

    public void display()

   {

        System.out.println("I am a method from class A");

    }

}

class B extends A

 {

    public void print()

   {

        System.out.println("I am a method from class B");

    }

}

class C extends A

{

    public void show()

 {

        System.out.println("I am a method from class C");

    }

}
```

```
class D extends A

{

    public void outPut()

   {

        System.out.println("I am a method from class D");

    }

    Class hierar

   {

    public static void main(String[] args) {

        B objB = new B();

        C objC = new C();

        D objD = new D();

        objB.display();

        objC.display();

        objD.display();

    }

}
```

OUTPUT:-

```
I am a method from class A

I am a method from class A

I am a method from class A
```

Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes.

The C class inherits A and B classes.

If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
 class A
 {
    void msg()
    {
    System.out.println("Hello");
    }
    }
    class B
    {
    void msg()
  {
System.out.println("Welcome");}
}
class C extends A,B
{//suppose if it were
   }
```

```
Class multiple
{
 public static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

```
Compile Time Error
```

Java Forms of Inheritance

The inheritance concept used for the number of purposes in the java programming language.

One of the main purposes is substitutability.

The substitutability means that when a child class acquires properties from its parent class, the object of the parent class may be substituted with the child class object.

For example, if B is a child class of A, anywhere we expect an instance of A we can use an instance of B.

The substitutability can achieve using inheritance, whether using extends or implements keywords.

The following are the different forms of inheritance in java.

- Specialization
- Specification
- Construction
- Extension
- Limitation
- Combination

Specialization

It is the most ideal form of inheritance. The subclass is a special case of the parent class. It holds the principle of substitutability.

Specification

This is another commonly used form of inheritance. In this form of inheritance, the parent class just specifies which methods should be available to the child class but doesn't implement them. The java provides concepts like abstract and interfaces to support this form of inheritance. It holds the principle of substitutability.

Construction

This is another form of inheritance where the child class may change the behavior defined by the parent class (overriding). It does not hold the principle of substitutability.

Eextension

This is another form of inheritance where the child class may add its new properties. It holds the principle of substitutability.

Limitation

This is another form of inheritance where the subclass restricts the inherited behavior. It does not hold the principle of substitutability.

Combination

This is another form of inheritance where the subclass inherits properties from multiple parent classes. Java does not support multiple inheritance type.

Benefits of Inheritance

- Inheritance helps in code reuse. The child class may use the code defined in the parent class without re-writing it.
- Inheritance can save time and effort as the main code need not be written again.
- Inheritance provides a clear model structure which is easy to understand.
- An inheritance leads to less development and maintenance costs.
- With inheritance, we will be able to override the methods of the base class so that the meaningful implementation of the base class method can be designed in the derived class.

An inheritance leads to less development and maintenance costs.

Creating a Multilevel Inheritance Hierarchy in Java

Inheritance involves an object acquiring the properties and behaviour of another object. So basically, using inheritance can extend the functionality of the class by creating a new class that builds on the previous class by inheriting it.

Multilevel inheritance is when a class inherits a class which inherits another class. An example of this is class C inherits class B and class B in turn inherits class A.

A program that demonstrates a multilevel inheritance hierarchy in Java is given as follows:

Example

```
class A
 {
   void funcA()
  {
      System.out.println("This is class A");
   }
}
class B extends A
{
   void funcB()
{
      System.out.println("This is class B");
   }
}
class C extends B
 {
   void funcC()
 {
      System.out.println("This is class C");
   }
}
public class Demo
{
   public static void main(String args[])
  {
      C obj = new C();
      obj.funcA();
      obj.funcB();
      obj.funcC();
   }
}
```

Output

```
This is class A
This is class B
This is class C
```

Method Overriding:

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

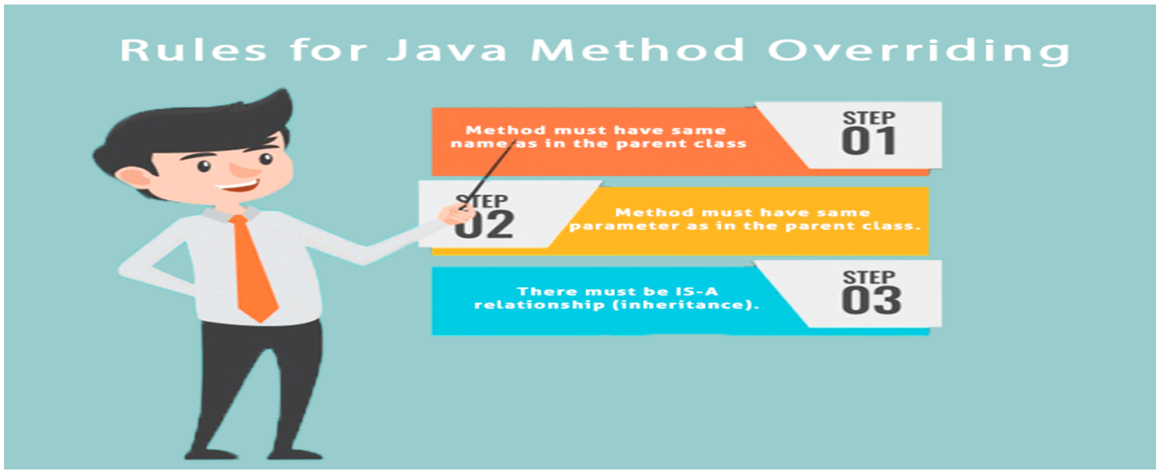
In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).


Rules for Java Method Overriding
Method must have same name as in the parent class
STEP 01
STEP 02
Method must have same parameter as in the parent class.
There must be IS-A relationship (inheritance).
STEP 03

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
//Java Program to illustrate the use of Java Method Overriding
//Creating a parent class.
class Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child class
class Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output:

```
Bike is running safely
```

Difference between Method Overloading and Method Overriding
Following are the key differences between method overloading and overriding in Java.

Method Overloading	Method Overriding
• **It is used to increase the readability of the program**	• **Provides a specific implementation of the method already in the parent class**
• **It is performed within the same class**	• **It involves multiple classes**
• **Parameters must be different in case of overloading**	• **Parameters must be same in case of overriding**
• **Is an example of compile-time polymorphism**	• **It is an example of runtime polymorphism**
• **Return type can be different but you must change the parameters as well.**	• **Return type must be same in overriding**
• **Static methods can be overloaded**	• **Overriding does not involve static methods.**

Using super :-

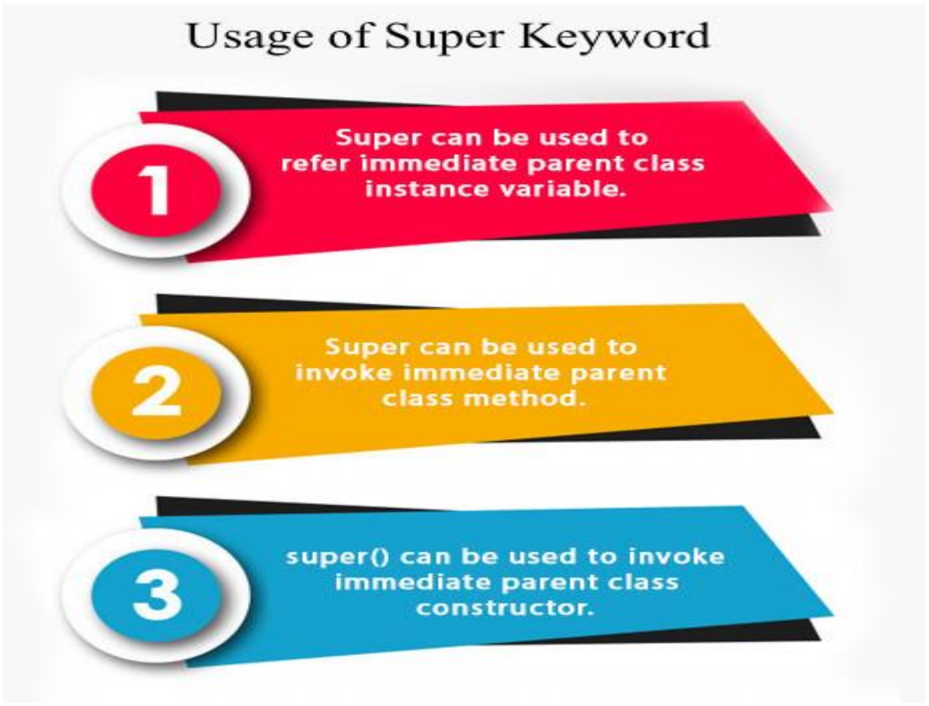
Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.


Usage of Super Keyword
1
Super can be used to refer immediate parent class instance variable.
2
Super can be used to invoke immediate parent class method.
3
super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor()
{
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

o/p:-black ,white

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat()
{
System.out.println("eating bread...");
}
void bark()
{
System.out.println("barking...");
}
void work()
{
super.eat();
bark();
}
}
class TestSuper2
{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}  Output:eating...  barking...
```

In the above example Animal and Dog both classes have eat() method if we call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, we need to use super keyword.

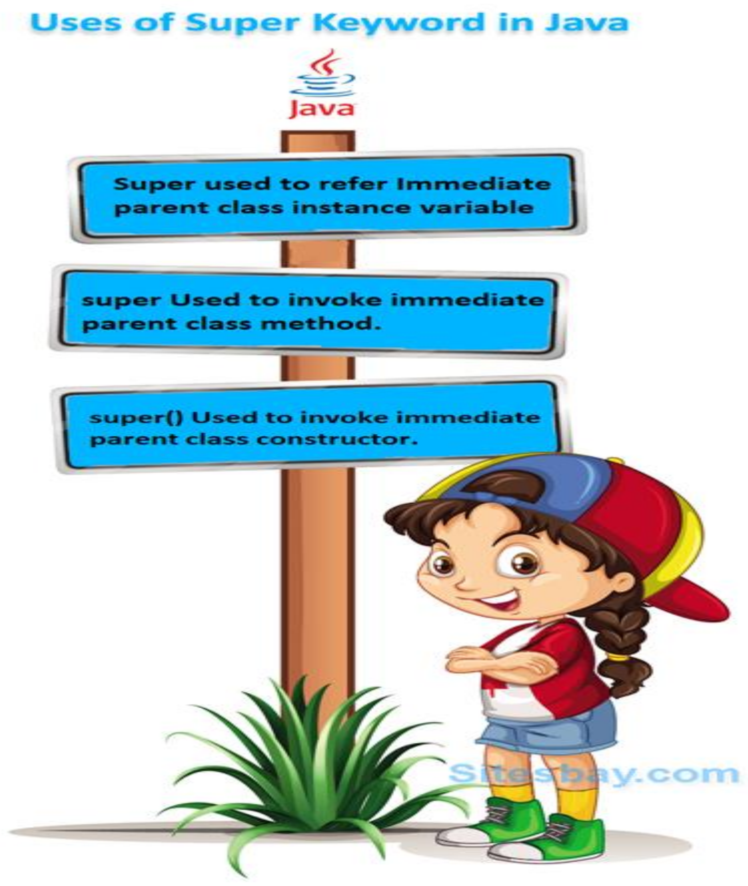
3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Let's see a simple example:

```
class Animal{
Animal()
{
System.out.println("animal is created");}
}
class Dog extends Animal
{
Dog()
{
  super();
System.out.println("dog is created");
}
}
class TestSuper3
{
public static void main(String args[]){
Dog d=new Dog();
}}
```

output:

```
animal is created
dog is created
```


Uses of Super Keyword in Java
Java
Super used to refer Immediate parent class instance variable
super Used to invoke immediate parent class method.
super() Used to invoke immediate parent class constructor.
Sitesbay.com

When Need of super keyword

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

Syntax

```
super.baseclass features.
```

Dynamic method dispatch

Dynamic Method Dispatch in Java is the process by which a call to an overridden method is resolved at runtime (during the code execution). The concept of method overriding is the way to attain runtime polymorphism in Java. During the code execution, JVM decides which implementation of the same method should be called.

Polymorphism in Java

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Runtime Polymorphism in Java

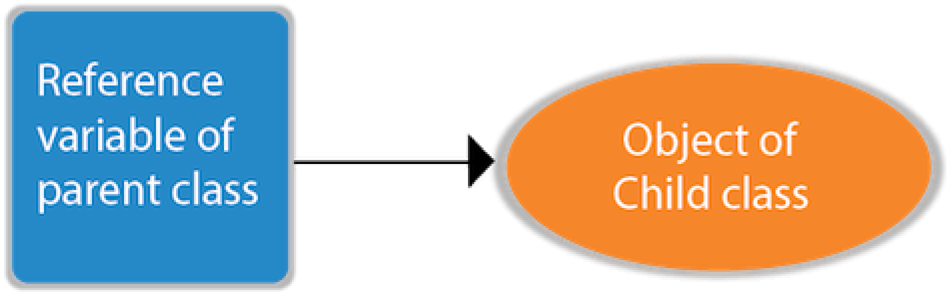
Runtime polymorphism or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

Let's first understand the upcasting before Runtime Polymorphism.

Upcasting

If the reference variable of Parent class refers to the object of Child class, it is known as upcasting. For example:



```
1. class A{}
2. class B extends A{}
1. A a=new B();//upcasting
```

For upcasting, we can use the reference variable of class type or an interface type.

Example of Java Runtime Polymorphism

In this example, we are creating two classes Bike and Splendor.

Splendor class extends Bike class and overrides its run() method.

We are calling the run method by the reference variable of Parent class.

Since it refers to the subclass object and subclass method overrides the Parent class method, the subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
  class Bike
 {
  void run()
  {
  System.out.println("running");}
  }
 class Splendor extends Bike
{
 void run()
{
System.out.println("running safely with 60km");
}

 public static void main(String args[])
 {
  Bike b = new Splendor();//upcasting
  b.run();
 }
}
```

Output:

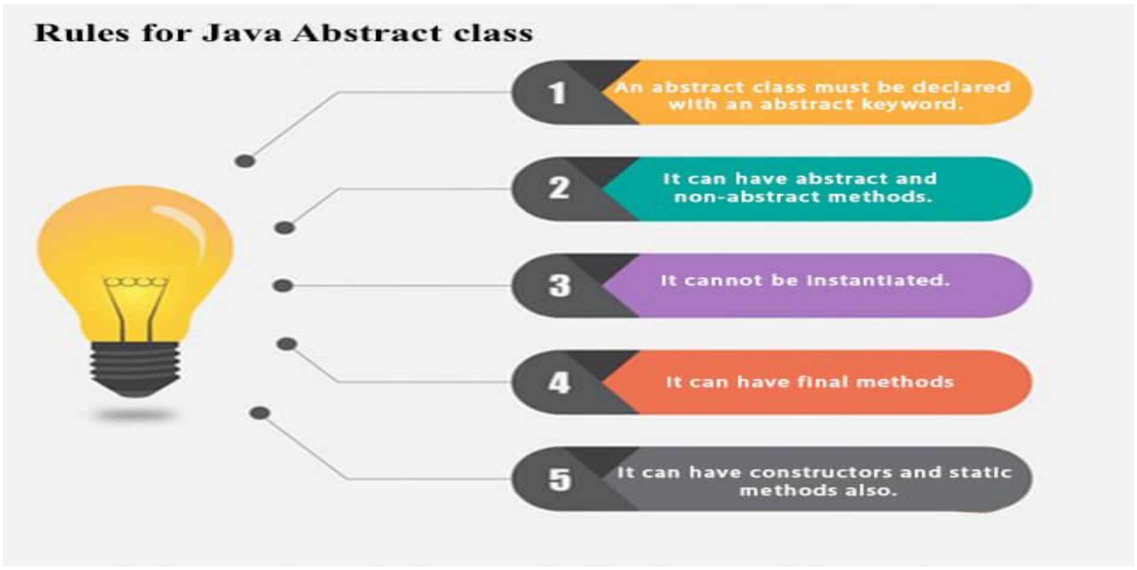
```
running safely with 60km.
```

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

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


Rules for Java Abstract class
1
An abstract class must be declared with an abstract keyword.
2
It can have abstract and non-abstract methods.
3
It cannot be instantiated.
4
It can have final methods
5
It can have constructors and static methods also.

Example of abstract class

```
abstract class A{   }
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Example of abstract method

```
abstract void printStatus();//no method body and abstract
```

Example of Abstract class that has an abstract method

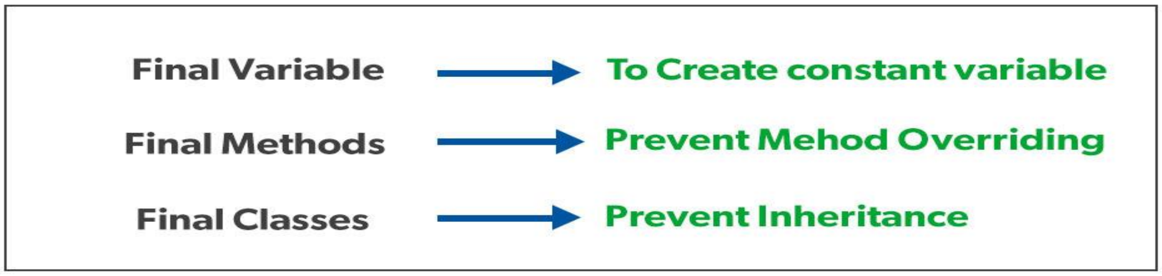
In this example, Bike is an abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

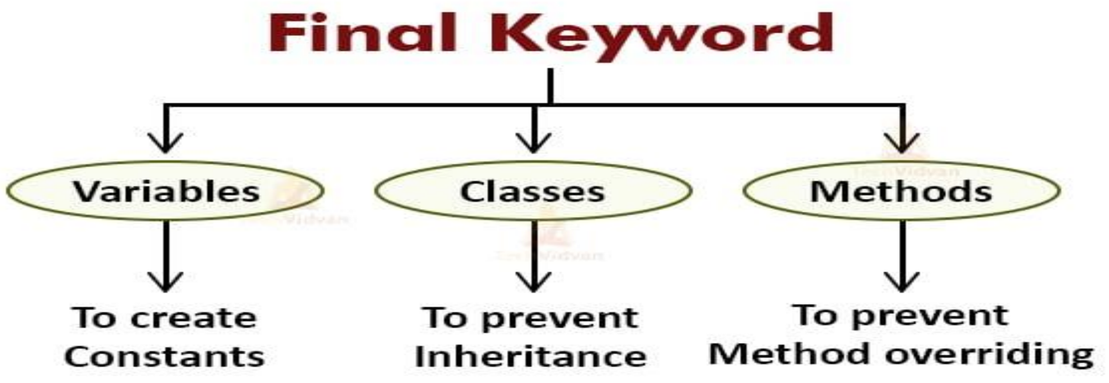
```
abstract class Bike
{
  abstract void run();
}
class Honda4 extends Bike
{
void run(){System.out.println("running safely");
}
public static void main(String args[])
{
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output:-

```
running safely
```

Final with inheritance:-





1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9
{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

```
Output:Compile Time Err
```

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
class Bike
{
  final void run()
{
System.out.println("running");}
}

class Honda extends Bike
{
   void run(){System.out.println("running safely with 100kmph");
}
   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}  Output:Compile Time Error
```

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Bike
{
}
 class Honda1 extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}
   public static void main(String args[])
{
  Honda1 honda= new Honda1();
  honda.run();
  }
}
```

```
Output:Compile Time Error
```

Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it.