Inheritance In Java

What is Inheritance ?

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

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.

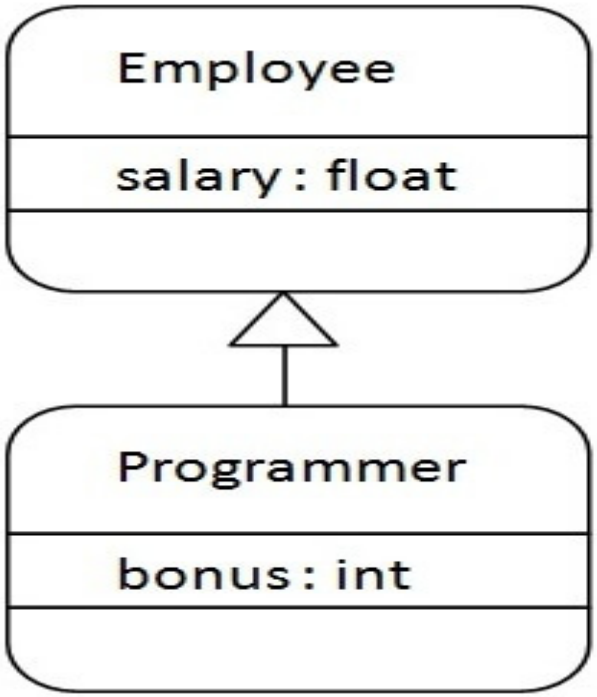
Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same

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.

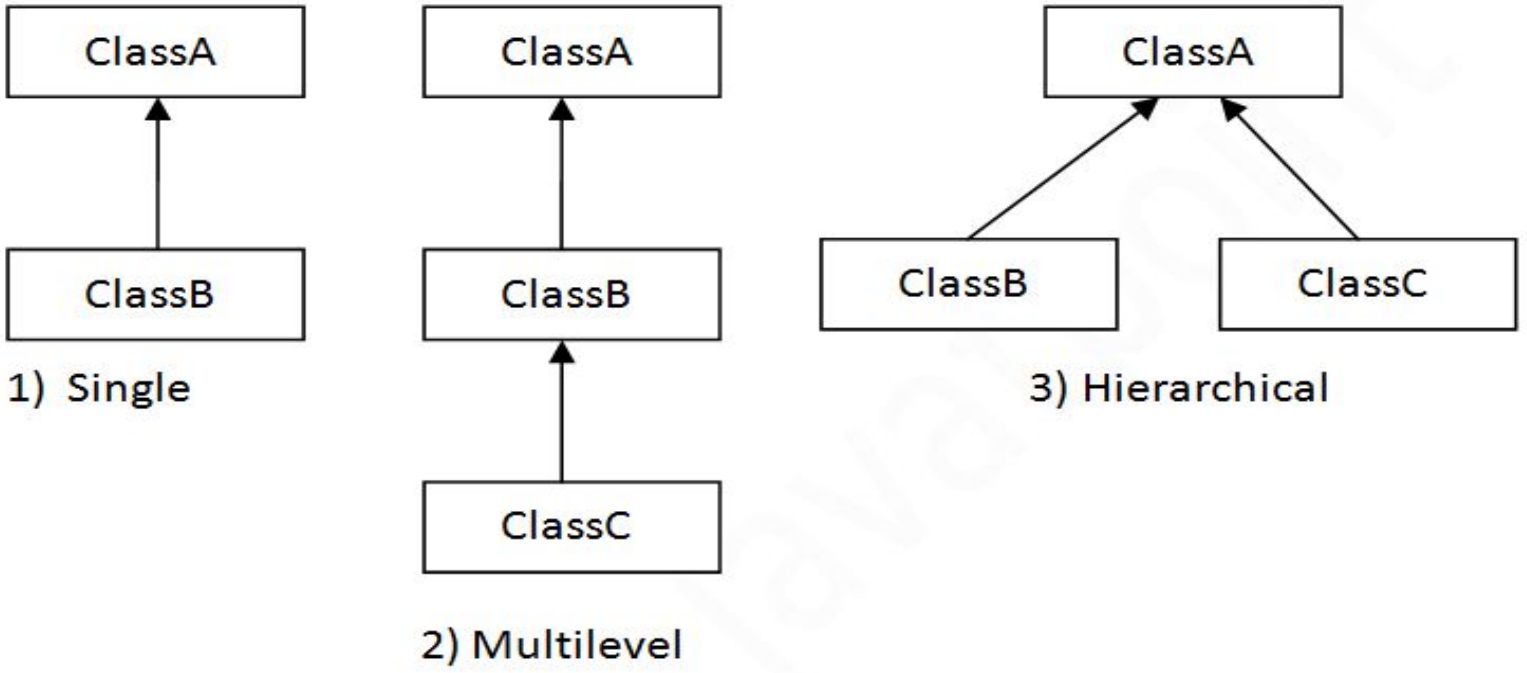
Java Inheritance Example



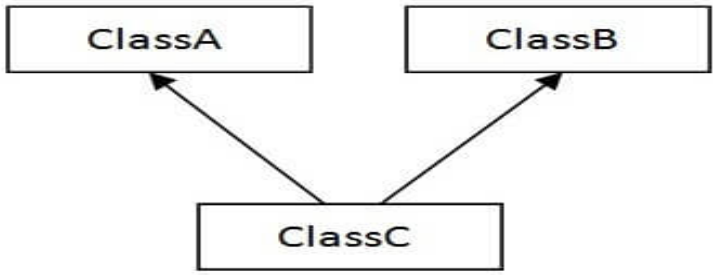
Java Inheritance Example

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

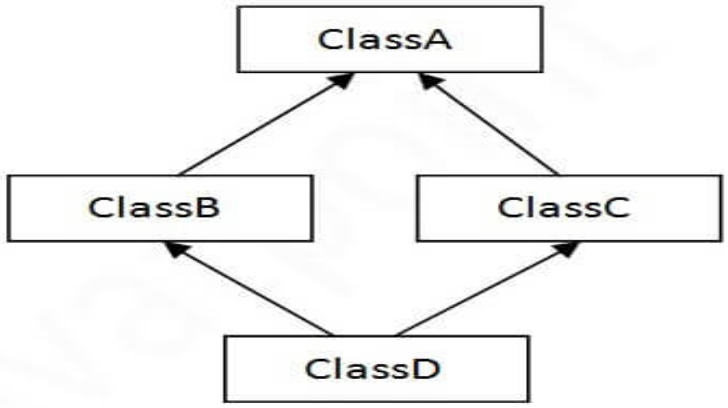
Java Inheritance Example



Java Inheritance Example



4) Multiple



5) Hybrid

Note: Multiple inheritance is not supported in Java through class.

Single Inheritance 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();  }}
```

Multilevel Inheritance Example

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

Hierarchical Inheritance Example

```
class Animal{
void eat(){System.out.println("eating...");}  }
class Dog extends Animal{
void bark(){System.out.println("barking...");}   }
class Cat extends Animal{
void meow(){System.out.println("meowing...");}  }
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Inheritance In Java

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.

Inheritance In Java

Why multiple inheritance is not supported in java?

```
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

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

Aggregation In Java

Aggregation in Java :

If a class have an entity reference, it is known as Aggregation.

Consider a situation, Employee object contains many informations such as id, name, emailId etc. It contains one more object named address, which contains its own informations such as city, state, country, zipcode etc. as given below.

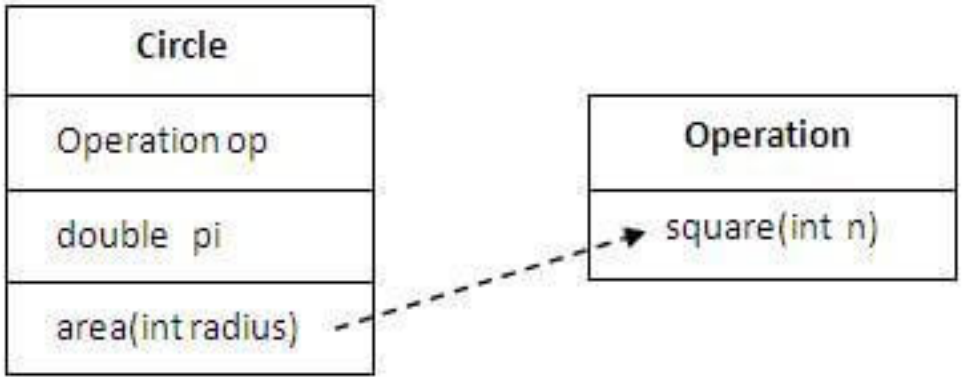
```
class Employee{
int id;
String name;
Address address;//Address is a class
}
```

Aggregation In Java

Why use Aggregation ?

For Code Reusability.

Simple Example of Aggregation :



Aggregation In Java

Example of Aggregation

```
class Operation{
 int square(int n){
  return n*n;  }
}
class Circle{
 Operation op;//aggregation
 double pi=3.14;
 double area(int radius){
   op=new Operation();
   int rsquare=op.square(radius);//code reusability (i.e.
delegates the method call).
   return pi*rsquare; }
```

Aggregation In Java

Example of Aggregation

```
 public static void main(String args[]){
   Circle c=new Circle();
   double result=c.area(5);
   System.out.println(result);
 }
}
```

Aggregation In Java

Example of Aggregation 1

```
public class Address {
String city,state,country;

public Address(String city, String state, String country)
{
    this.city = city;
    this.state = state;
    this.country = country;
}

}
```

Aggregation In Java

Example of Aggregation 1

```
public class Emp {
int id;
String name;
Address address;
public Emp(int id, String name,Address address) {
    this.id = id;
    this.name = name;
    this.address=address;
}
void display(){
System.out.println(id+" "+name);
System.out.println(address.city+" "+address.state+"
"+address.country);  }
```

Aggregation In Java

Example of Aggregation 1

```
public static void main(String[] args) {
Address address1=new Address("gzb","UP","india");
Address address2=new Address("gno","UP","india");

Emp e=new Emp(111,"varun",address1);
Emp e2=new Emp(112,"arun",address2);

e.display();
e2.display();
}
}
```

Method overloading In Java

Can we overload java main() method?

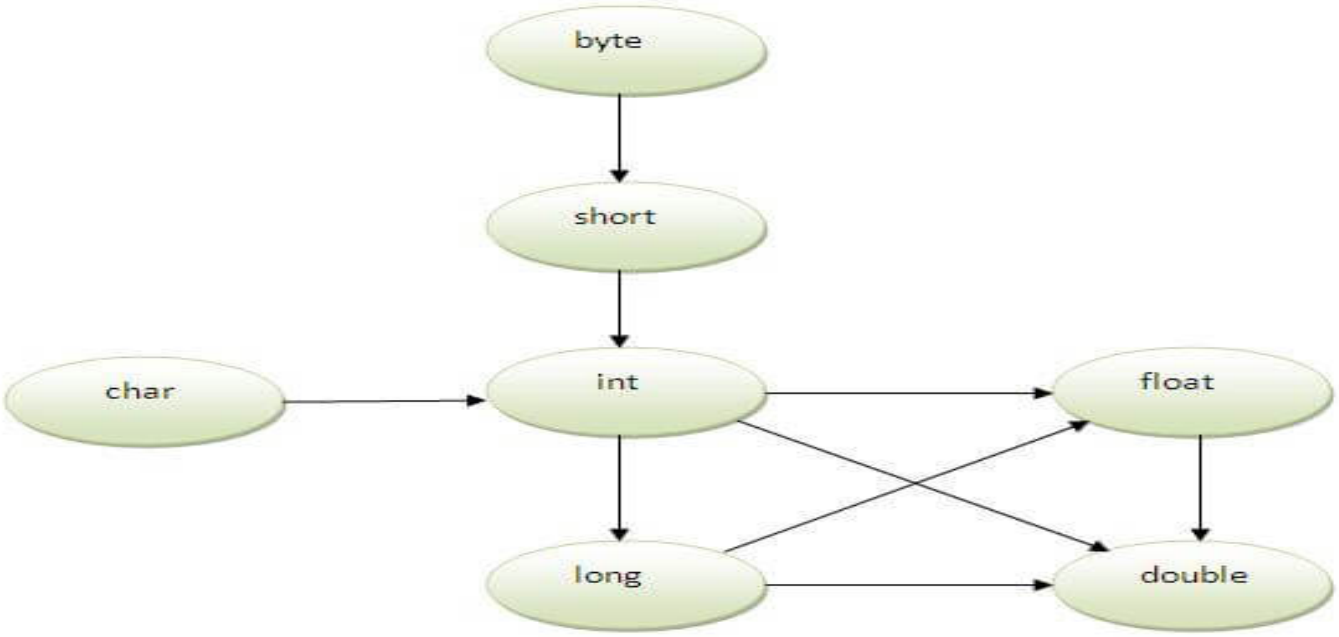
Yes, by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

```
   class TestOverloading{
   public static void main(String[] args)
{System.out.println("main with String[]");}
   public static void main(String args)
{System.out.println("main with String");}
   public static void main()
{System.out.println("main without args");}
   }
```

Method overloading In Java

Method Overloading and Type Promotion :

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



Method overloading In Java

Example of Method Overloading with TypePromotion

```
class OverloadingCal{
void sum(int a,long b)
 {System.out.println(a+b);}
void sum(int a,int b,int c)
{System.out.println(a+b+c);}

public static void main(String args[]){
  OverloadingCal obj=new OverloadingCal();
  obj.sum(20,20);//now second int literal will be promoted to
long
  obj.sum(20,20,20);
  }
}
```

Method overloading In Java

Example of Method Overloading with Type Promotion in case of ambiguity

```
class OverloadingCal{
  void sum(int a,long b)
  {System.out.println("a method invoked");}
  void sum(long a,int b)
  {System.out.println("b method invoked");}

  public static void main(String args[]){
  OverloadingCal obj=new OverloadingCal();
  obj.sum(20,20);//now ambiguity
  }
}
```

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

Usage of Java Method Overriding :

- **Method overriding is used to provide specific implementation of a method that is already provided by its super class.**
- **Method overriding is used for runtime polymorphism**

Method overriding In Java

Understanding the problem without method overriding

```
class Vehicle
{
  void run(){System.out.println("Vehicle is running");}
}

class Bike extends Vehicle
{
  public static void main(String args[])
  {
  Bike obj = new Bike();
  obj.run();
  }
}
```

Method overriding In Java

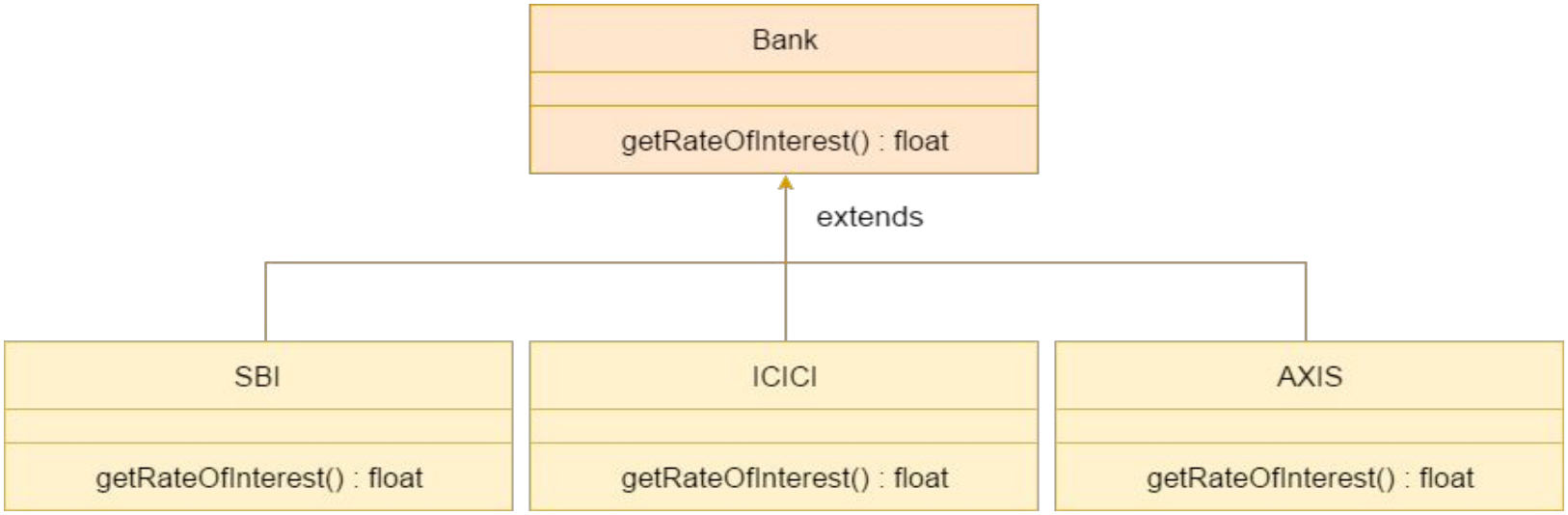
Example of method overriding

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

Method overriding In Java

Example of method overriding 1

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.



Method overriding In Java

Why we cannot override static method ?

Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

Can we override java main method ?

No, because main is a static method.

Method overriding In Java

Covariant Return Type

The covariant return type specifies that the return type may vary in the same direction as the subclass.

Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type.

Method overriding In Java

Simple example of Covariant Return Type

```
class A{
A get(){return this;}
}

class B1 extends A{
B1 get(){return this;}
void message()
{System.out.println("welcome to covariant return type");}

public static void main(String args[]){
new B1().get().message();
}
}
```

Method overriding In Java

Simple example of Covariant Return Type

As you can see in the above example, the return type of the get() method of A class is A but the return type of the get() method of B1 class is B1. Both methods have different return type but it is method overriding. This is known as covariant return type.

Method overriding In Java

How is Covariant return types implemented ?

Java doesn't allow the return type based overloading but JVM always allows return type based overloading. JVM uses full signature of a method for lookup/resolution. Full signature means it includes return type in addition to argument types. i.e., a class can have two or more methods differing only by return type. javac uses this fact to implement covariant return types.

super keyword in java

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.

super keyword in Java

Usage of java super Keyword :

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

super keyword in Java

super is used to refer immediate parent class instance variable.

- **super can be used to refer immediate parent class instance variable.**
- **super can be used to invoke immediate parent class method.**
- **super() can be used to invoke immediate parent class constructor.**

super keyword in java

super is used to refer immediate parent class instance variable

```
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 TestSuper{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
```

super keyword in java

super can be used to invoke parent class method

```
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 TestSuper{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

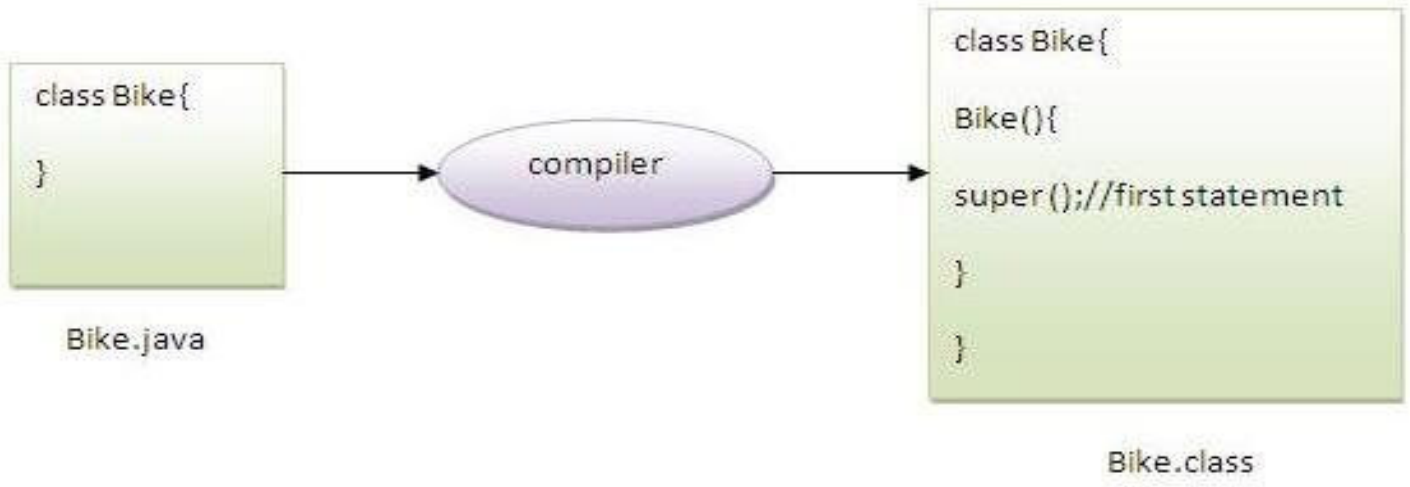
super keyword in java

super is used to invoke parent class constructor

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

super keyword in Java

Note: super() is added in each class constructor automatically by compiler if there is no super()



As we know well that default constructor is provided by compiler automatically if there is no constructor. But, it also adds super() as the first statement.

super keyword in java

Another example of super keyword where super() is provided by the compiler implicitly.

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

super keyword in java

Another example of super keyword

```
class Person{
int id;
String name;
Person(int id,String name){
this.id=id;
this.name=name;
} }
class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);//reusing parent constructor
this.salary=salary;
}
```

super keyword in java

Another example of super keyword

```
void display(){System.out.println(id+" "+name+" "+salary);}
}
class TestSuper{
public static void main(String[] args){
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}}
```

Instance Initializer block in Java

Instance initializer block :

Instance Initializer block is used to initialize the instance data member. It run each time when object of the class is created.

The initialization of the instance variable can be done directly but there can be performed extra operations while initializing the instance variable in the instance initializer block.

Why use instance initializer block?

Suppose I have to perform some operations while assigning value to instance data member e.g. a for loop to fill a complex array or error handling etc.

super keyword in java

Example of instance initializer block :

```
class Bike{
    int speed;

    Bike(){System.out.println("speed is "+speed);}

    {speed=100;}
    public static void main(String args[]){
    Bike b1=new Bike();
    Bike b2=new Bike();
    }
}
```

Output:speed is 100
speed is 100

super keyword in java

What is invoked first, instance initializer block or constructor ?

```
class Bike{
    int speed;
    Bike(){System.out.println("constructor is invoked");}

   {System.out.println("instance initializer block invoked");}

    public static void main(String args[]){
    Bike b1=new Bike();
    Bike b2=new Bike();
    }
}
```

super keyword in java

What is invoked first, instance initializer block or constructor ?

```
class Bike{
    int speed;
    Bike(){System.out.println("constructor is invoked");}

   {System.out.println("instance initializer block invoked");}

    public static void main(String args[]){
    Bike b1=new Bike();
    Bike b2=new Bike();
    }
}
```

super keyword in java

What is invoked first, instance initializer block or constructor ?

Output:

```
instance initializer block invoked
constructor is invoked
instance initializer block invoked
constructor is invoked
```

In the above example, it seems that instance initializer block is firstly invoked but NO. Instance intializer block is invoked at the time of object creation. The java compiler copies the instance initializer block in the constructor after the first statement super(). So firstly, constructor is invoked.

super keyword in java

Note: The java compiler copies the code of instance initializer block in every constructor.

```
Class B{
B(){
System.out.println("constructor");}
}
{System.out.println("instance initializer block");}
}
```


```
class B{
B(){
super();
{System.out.println("instance initializer block");}
System.out.println("constructor");}
}
}
```

super keyword in Java

Rules for instance initializer block :

There are mainly three rules for the instance initializer block. They are as follows:

- **The instance initializer block is created when instance of the class is created.**
- **The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).**
- **The instance initializer block comes in the order in which they appear.**

super keyword in java

Program of instance initializer block that is invoked after super()

```
class A{
A()
{System.out.println("parent class constructor invoked");}
}
class B2 extends A{
B2(){
super();
System.out.println("child class constructor invoked"); }

{System.out.println("instance initializer block is invoked");}

public static void main(String args[]){
B2 b=new B2();  }  }
```

super keyword in Java

Another example of instance block

```
class A{
A(){
System.out.println("parent class constructor invoked");
}
}
class B3 extends A{
B3(){
super();
System.out.println("child class constructor invoked");
}
```

super keyword in java

Another example of instance block

```
B3(int a){
super();
System.out.println("child class constructor invoked "+a);
}

{System.out.println("instance initializer block is invoked");}

public static void main(String args[]){
B3 b1=new B3();
B3 b2=new B3(10);
}
}
```

Final Keyword In Java

Final :

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- **variable**
- **method**
- **class**

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

Final Keyword In Java

Java final variable :

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

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

Output:Compile Time Error

Final Keyword In Java

Java final method :

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

```
  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

Final Keyword In Java

Java final class :

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

```
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

Final Keyword In Java

Is final method inherited ?

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

```
class Bike{
  final void run()
  {System.out.println("running...");}
}
class Honda extends Bike{
   public static void main(String args[]){
    new Honda().run();
   }
}
```

Output: running...

Final Keyword In Java

What is blank or uninitialized final variable ?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

```
class Student{
int id;
String name;
final String PAN_CARD_NUMBER;
... }
```

Final Keyword In Java

Can we initialize blank final variable ?

```
class Bike{
  final int speedlimit;//blank final variable

  Bike(){
  speedlimit=70;
  System.out.println(speedlimit);
  }

  public static void main(String args[]){
    new Bike();
 }
}
```

Final Keyword In Java

static blank final variable :

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

```
class A{
  static final int data;//static blank final variable
  static{ data=50;}
  public static void main(String args[]){
    System.out.println(A.data);
 }
}
```

Final Keyword In Java

What is final parameter ?

If you declare any parameter as final, you cannot change the value of it.

```
class Bike{
  int cube(final int n){
   n=n+2;//can't be changed as n is final
   n*n*n;
  }
  public static void main(String args[]){
    Bike b=new Bike();
    b.cube(5);
  } }
```

Output: Compile Time Error

Final Keyword In Java

Can we declare a constructor final ?

No, because constructor is never inherited.