## Principles of OOPs

# Inheritance

# **Inheritance** is one of the features of Object-Oriented Programming ([OOPs](http://beginnersbook.com/2013/04/oops-concepts/)). Inheritance allows a child class to use the properties and methods of parent class. In other words, the derived class inherits the states and behaviours from the base class. The derived class is also called subclass and the base class is also known as super-class. The derived class can add its own additional variables and methods. These additional variable and methods differentiates the derived class from the base class.

Inheritance is a [compile-time](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) mechanism. A super-class can have any number of subclasses. But a subclass can have only one super class. This is because Java does not support multiple inheritance.

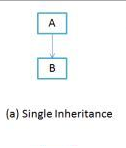
The superclass and subclass have **“is-a”** relationship between them.

**Note:**The derived class inherits all the members and methods that are declared as public or protected. If declared as private it cannot be inherited by the derived classes. The private members can be accessed only in its own class. The private members can be accessed through assessor methods. The derived class cannot inherit a member of the base class if the derived class declares another member with the same name.

**Types of Inheritance**

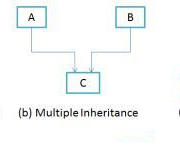
**1) Single Inheritance**

Single inheritance is damn easy to understand. When a class extends another one class only then we  call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a parent class of B and B would be  a child class of A.

**[](http://beginnersbook.com/wp-content/uploads/2013/05/Single-Inheritance.png)**

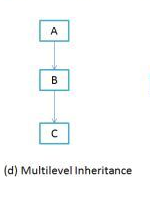
**2) Multiple Inheritances**

“Multiple Inheritance” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.

[](http://beginnersbook.com/wp-content/uploads/2013/05/Multiple-Inheritance.png)

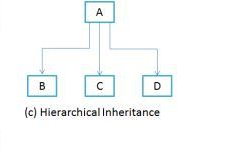
**3) Multilevel Inheritance**

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.

**[](http://beginnersbook.com/wp-content/uploads/2013/05/Multilevel-Inheritance.png)**

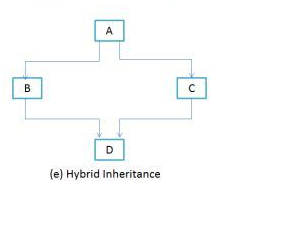
**4) Hierarchical Inheritance**

In such kind of inheritance one class is inherited by many sub classes. In below example class B, C and D inherits the same class A. A is parent class (or base class) of B,C & D

**[](http://beginnersbook.com/wp-content/uploads/2013/05/Hierarchical-Inheritance.png)**

**5) Hybrid Inheritance**

In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritances. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritances can be.Using interfaces. Yes you heard it right. By using interfaces you can have multiple as well as hybrid inheritance in Java.

**[](http://beginnersbook.com/wp-content/uploads/2013/05/Hybrid-inheritance.png)**

# Association

In this article we will discuss **Association in Java.** Association establish relationship between two **classes** through their **objects**. The relationship can be one to one, One to many, many to one and many to many.

## Aggregation

[Aggregation](http://beginnersbook.com/2013/05/aggregation/) is a special form of association which is a unidirectional one way relationship between classes (or entities), for e.g. Wallet and Money classes. Wallet has Money but money doesn’t need to have Wallet necessarily

## Composition

**Composition** is a restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other. For e.g. Human and Heart. A human needs heart to live and a heart needs a Human body to survive.

## Encapsulation

The whole idea behind encapsulation is to hide the implementation details from users. If a data member is private it means it can only be accessed within the same class. No outside class can access private data member (variable) of other class. However if we setup public getter and setter methods to update void setSSN(intssn)and read (intgetSSN()) the privatedata fields then the outside class can access those private data fields via public methods. This way data can only be accessed by public methods thus making the private fields and their implementation hidden for outside classes. That’s why encapsulation is known as **data hiding**

#### Advantages of encapsulation:

1. It improves maintainability and flexibility and re-usability: the implementation is purely hidden for outside classes they would still be accessing the private field empName using the same methods (setEmpName(String name) and getEmpName()). Hence the code can be maintained at any point of time without breaking the classes that uses the code. This improves the re-usability of the underlying class.
2. The fields can be made read-only (If we don’t define setter methods in the class) or write-only (If we don’t define the getter methods in the class). For e.g. If we have a field(or variable) which doesn’t need to change at any cost then we simply define the variable as private and instead of set and get both we just need to define the get method for that variable. Since the set method is not present there is no way an outside class can modify the value of that field.
3. User would not be knowing what is going on behind the scene. They would only be knowing that to update a field call set method and to read a field call get method but what these set and get methods are doing is purely hidden from them.

**Encapsulation is also known as “data Hiding”.**

1. Objects encapsulate data and implementation details. To the outside world, an object is a black box that exhibits a certain behavior.
2. The behavior of this object is what which is useful for the external world or other objects.
3. An object exposes its behavior by means of public methods or functions.
4. The set of functions an object exposes to other objects or external world acts as the interface of the object.

## Polymorphism

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. I know it sounds confusing. Don’t worry we will discuss this in detail.

* It is a  feature that allows one interface to be used for a general class of  actions.
* An operation may exhibit different behaviour in different instances.
* The behaviour depends on the types of data used in the operation.
* It plays an important role in allowing objects having different internal structures to share the same external interface.
* Polymorphism is extensively used in implementing inheritance

**Following concepts demonstrate different types of polymorphism in java.**

1) [Method Overloading](http://beginnersbook.com/2013/05/method-overloading/)  
2) [Method Overriding](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/)

**1) Method Overloading**

Method Overloading is a feature that allows a class to have two or more methods having same name, if their argument lists are different.

**Argument lists could differ in –**

1. Number of parameters.  
2. Data type of parameters.  
3. Sequence of Data type of parameters.

Method overloading is also known as **Static Polymorphism**.

**Points to Note:**1. [Static Polymorphism](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) is also known as compile time binding or early binding.  
2. [Static binding](http://beginnersbook.com/2013/04/java-static-dynamic-binding/) happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

**Rules for Method Overloading**

1. Overloading can take place in the same class or in its sub-class.
2. Constructor in Java can be overloaded
3. Overloaded methods must have a different argument list.
4. The parameters may differ in their type or number, or in both.
5. They may have the same or different return types.
6. It is also known as compile time polymorphism**.**

**OR**

1. Must have different argument lists
2. May have different return types, if argument lists are also different
3. May have different access modifiers
4. May throw different exceptions

**2) Method Overriding**

Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.

**OR**

Child class has the same method as of base class. In such cases child class overrides the parent class method without even touching the source code of the base class. This feature is known as method overriding.

**OR**

1. Must have the same argument list.
2. Must have the same return type,.
3. Must not have a more restrictive access modifier.
4. May have a less restrictive access modifier.
5. Must not throw new or broader checked exceptions.
6. May throw fewer or narrower checked exceptions, or any unchecked exception

#### Rules of method overriding in Java

1. **Argument list:** The argument list of overriding method must be same as that of the method in parent class. The data types of the arguments and their sequence should be maintained as it is in the overriding method.
2. [**Access Modifier**](http://beginnersbook.com/2013/05/java-access-modifiers/)**:** The Access Modifier of the overriding method (method of subclass) cannot be more restrictive than the overridden method of parent class.  For e.g. if the Access Modifier of base class method is public then the overriding method (child class method ) cannot have private, protected and default Access modifier as all of the three are more restrictive than public.
3. Private, static and final methods cannot be overridden as they are local to the class. However static methods can be re-declared in the sub class, in this case the sub-class method would act differently and will have nothing to do with the same static method of parent class.
4. Overriding method (method of child class) can throw any[unchecked exceptions](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/), regardless of whether the overridden method(method of parent class) throws any exception or not. However the overriding method should not throw [checked exceptions](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) that are new or broader than the ones declared by the overridden method. We will discuss this in detail with example in the upcoming tutorial.
5. Binding of overridden methods happen at runtime which is known as [dynamic binding](http://beginnersbook.com/2013/04/java-static-dynamic-binding/).
6. If a class is extending an [abstract class](http://beginnersbook.com/2013/05/java-abstract-class-method/) or implementing an[interface](http://beginnersbook.com/2013/05/java-interface/) then it has to override all the abstract methods unless the class itself is a abstract class.

# Exception handling in Method overriding

**Rule:**An overriding method (the method of child class) can throw any[unchecked exceptions](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/), regardless of whether the overridden method (method of base class) throws exceptions or not. However the overriding method should not throw [checked exceptions](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) that are new or broader than the ones declared by the overridden method. The overriding method can throw those checked exceptions, which have less scope than the exception(s) declared in the overridden method

**If base class doesn’t throw any exception but child class throws an unchecked exception.**

**If base class doesn’t throw any exception but child class throws a checked exception**

**When child class method is throwing border checked exception compared to the same method of base class**

**When base class and child class both throws a checked exception**

## Overloading vs. Overriding in Java

**1)** First and most important difference between method overloading and overriding is that, In case of method overloading in Java, signature of method changes while in case of method overriding it remain same.  
  
2) Second major difference between method overloading vs overriding in Java is that you can overload method in one class but overriding can only be done on subclass.  
  
3) You cannot override static, final and private method in Java but you can overload static, final or private method in Java.  
  
4) Overloaded method in Java is bonded by static binding and overridden methods are subject to dynamic binding.  
  
5) Private and final method can also be not overridden in Java.

**Abstraction**

The purpose of abstraction is to hide information that is not relevant and  show only the relevant information and to simplify it by comparing it to something similar in the real world.

Abstraction means “The process of forming of general and relevant concept from more complex scenarios”.

Following concepts demonstrate different types of Abstractionin java.

**1)**[**Abstract**](http://beginnersbook.com/2013/05/method-overloading/) **Class  
2)**[**Interface**](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/)

# Abstract Classes

A class that is declared using “abstract” keyword is known as abstract class. It may or may not include abstract methods which means in abstract class you can have concrete methods (methods with body) as well along with abstract methods ( without an implementation, without braces, and followed by a semicolon). An abstract class cannot be instantiated (you are not allowed to create object of Abstract class).

**Abstract class declaration**

Specifying abstract keyword before the class during declaration, makes it abstract. Have a look at below code:

// Declaration using abstract keyword

abstract class AbstractDemo{

// Concrete method: body and braces

public void myMethod(){

//Statements here

}

// Abstract method: without body and braces

abstract public void anotherMethod();

}

Since abstract class allows concrete methods as well, it does not provide 100% abstraction. You can say that it provides partial abstraction. [Interfaces](http://beginnersbook.com/2013/05/java-interface/) are used for 100% abstraction (full [abstraction](http://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/))

Remember two rules:  
1) If the class is having few abstract methods and few concrete methods: declare it as abstract class.  
2) If the class is having only abstract methods: declare it as interface.

## Why we need an abstract class?

If you want to provide common, implemented functionality among with all implementations of your component, use an abstract class. Abstract classes allow you to partially implement your class.

Abstract class has concrete method which is commonlyshared by all the inherited class, ifwe updating the base class,all inheriting classes are automatically updated with the change.

Abstract classes should be used primarily for objects that are closely related.

Key Points:

1. An abstract class has no use until unless it is extended by some other class.
2. If you declare an abstract method (discussed below) in a class then you must declare the class abstract as well. you can’t have abstract method in a non-abstract class. It’s vice versa is not always true: If a class is not having any abstract method then also it can be marked as abstract.
3. Abstract class can have non-abstract method (concrete) as well.

## Abstract methods

Well, we already discussed about abstract methods in the above section. Lets take few examples to understand it better.

**syntax:**

publicabstractvoid display();

**Points to remember about abstract method:**

1) Abstract method has no body.  
2) Always end the declaration with a **semicolon**(;).  
3) It must be [overridden](http://beginnersbook.com/2014/01/method-overriding-in-java-with-example/). An abstract class must be extended and in a same way abstract method must be overridden.  
4) Abstract method must be in an abstract class.

**Note:**The class which is extending abstract class must override (or implement) all the abstract methods.

# Interface in java

An interface in java is a blueprint of a class. It has static constants and abstract methods only.OR Interface is a contract.

The interface in java is a mechanism to achieve fully abstraction. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.

Java Interface also represents IS-A relationship.

It cannot be instantiated just like abstract class.

#### The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

**What is the use of interfaces?Or why we use Interface in java.**

A particular advantage of using interface in Java is that it allows multiple **inheritance.**

Java programming language does not support multiple inheritance, using interfaces we can achieve this as a class can implement more than one interfaces, however it cannot extend more than one classes.

**Declaration**Interfaces are declared by specifying a keyword “interface”. E.g.:

Interface MyInterface {

/\* All the methods are public abstract by default

\* Note down that these methods are not having body

\*/

public void method1();

public void method2();

}

**Key points**1) We can’t instantiate an interface in java.

2) Interface provides complete [abstraction](http://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/) as none of its methods can have body. 3) implements keyword is used by classes to implement an interface.

4) While providing implementation in class of any method of an interface, it needs to be mentioned as public.

5) Class implementing any interface must implement all the methods, otherwise the class should be declared as “abstract”.

6) Interface cannot be declared as private, protected.

7) All the interface methods are by default **abstract and public**.

8) Variables declared in interface are **public, static and final** by default.

9) Interface variables must be initialized at the time of declaration otherwise compiler will through an error.

interfaceTry

{

int x;//Compile-time error

}

10) Any interface can extend any interface but cannot implement it. Class implements interface and interface extends interface.

11) A **class** can implement any **number of interfaces**.

13) If there are **two or more same methods** in two interfaces and a class implements both interfaces, implementation of the method once is enough.

interface A

{

Public void aaa();

}

interface B

{

Public void aaa();

}

Class Central implements A,B

{

Public void aaa()

{

//Any Code here

}

Public static void main(String args[])

{

//Statements

}

}

14) A class cannot implement two interfaces that have methods with same name but different return type.

interface A

{

Public void aaa();

}

interface B

{

Public int aaa();

}

classCentralimplements A,B

{

Public void aaa()// error

{

}

Public int aaa()// error

{

}

publicstaticvoid main(Stringargs[])

{

}

}

#### Benefits of having interfaces:

1. In java, [**multiple inheritance**](http://beginnersbook.com/2013/05/java-multiple-inheritance/) is not allowed, However by using interfaces you can achieve the same . A class can extend only one class but can implement any number of interfaces. It saves you from Deadly Diamond of Death(DDD) problem.

# Difference Between Abstract Class and Interface in Java

|  |  |
| --- | --- |
| Abstract Class | Interface |
| abstract  class  can  have  both  abstract and concrete methods | interface can  have only abstract methods |
| A class can extend only one abstract class | A class can implement any number of interfaces |
| In abstract class keyword ‘abstract’ is mandatory to declare a method as an abstract | In an interface keyword ‘abstract’ is optional to declare a method as an abstract |
| abstract  class can have  protected , public and public abstract methods | Interface can have only public abstract methods i.e. by default |
| abstract class can have  static, final  or static final  variable with any access specifier | interface  can  have only static final (constant) variable i.e. by default |

**What is constructor in Java**

Constructor in Java is block of code which is executed at the time of Object creation.

## Constructor in Java – things to remember

**1)** First and most important rule of declaring constructor is that name of constructor in Java must be exactly same with the class on which you declare constructor. A class in Java can have as many constructor as it and that is called constructor overloading in Java but signature of two constructor must not be same. here is an example of having multiple constructors in Java and how they are called using new() operator:

**2)** Another important rule of declaring constructor is that constructor in Java doesn't have return type

**3)** Every Class in Java has constructor, if no explicit constructor is specified by Programmer, Java Compiler inserts a no argument constructor inside class. This is also called default Constructor in Java. if you provide any constructor in Java e.g. with one argument or two argument than compiler will not add default constructor or no arguments constructor,

Another drawback of not providing no argument constructor is chances of having restricted hierarchy. Suppose another sub class is created and you don't add constructor over there than compiler tries to create a default constructor which calls super() at first line. super() means call to no argument constructor of super class and since there is no such constructor in your class it will fail with compilation error. This is like making your class final in Java

**4)** You can use any access modifier with Java constructor. they can be public, protected or private. Default or no argumentconstructor has same access modifier as class.

**5**) Constructor in Java cannot be abstract, [static](http://javarevisited.blogspot.sg/2012/10/what-is-static-import-in-java-5-example-tutorial.html), [final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) or [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html). These modifiers are not allowed for constructor.

6) Since parent class is initialized before child class in Java, Constructor of parent class is executed before constructor of child class, that explains why super() is first statement in default no argument constructor.

**Protected constructor**

Protected constructor can be accessed from its own class, its subclasses, all other classes belonging to the same package and subclasses of other packages.

# Java – private constructor example

The **use of private constructor** is to serve singleton classes. A singleton class is one which limits the number of objects creation to one. Using private constructor we can ensure that no more than one object can be created at a time. We will see in the below example how to use private constructor for limiting the number of objects for a singleton class.

# Java – Constructor in Interface?

This is a most frequently asked java interview question. The answer is No, interface cannot have constructors. In this post we will discuss why [constructors](http://beginnersbook.com/2013/03/constructors-in-java/) are not allowed in [interface](http://beginnersbook.com/2013/05/java-interface/)?

As we know that all the methods in interface are public abstract by default which means the method implementation cannot be provided in the interface itself. It has to be provided by the implementing class.

# Java static constructor– Is it really Possible to have them in Java?

Have you heard of **static constructor in Java**? I guess yes but the fact is that they are not allowed in Java. A constructor cannot be static in Java.

**Why java doesn’t support static constructor?**

It’s actually pretty simple to understand – Everything that is marked static belongs to the class only, for example static method cannot be inherited in the sub class because they belong to the class in which they have been declared.

# Static KeyWord

In this tutorial we will discuss the use of **static keyword in Java**. It can be used along with Class name, Variables, Methods and block.

1. static class  
2. static block  
3. static methods  
4. static variables

## Static Class

A Class can be made **static** only if it is a nested Class. The nested static class can be accessed without having an object of outer class.

**Example 1:**

classExample1{

//Static class

staticclass X{

staticStringstr="Inside Class X";

}

publicstaticvoid main(Stringargs[])

{

X.str="Inside Class Example1";

System.out.println("String stored in str is- "+X.str);

}

}

Output:

String stored instris-InsideClassExample1

## Static Block

Static block is mostly used for changing the default values of static variables.This block gets executed when the class is loaded in the memory.  
A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

**Example: Multiple Static blocks**

classExample4{

staticintnum;

staticStringmystr;

//First Static block

static{

System.out.println("Static Block 1");

num=68;

mystr="Block1";

}

//Second static block

static{

System.out.println("Static Block 2");

num=98;

mystr="Block2";

}

publicstaticvoid main(Stringargs[])

{

System.out.println("Value of num="+num);

System.out.println("Value of mystr="+mystr);

}

}

**Output:**

StaticBlock1

StaticBlock2

Value of num=98

Value of mystr=Block2

## Static Methods

Static Methods can access class variables without using object of the class. It can access non-static methods and non-static variables by using objects. Static methods can be accessed directly in static and non-static methods.

**Example 1: public static void main itself is a static method**

classExample5{

staticint i;

staticString s;

publicstaticvoid main(Stringargs[])//Its a Static Method

{

Example5obj=newExample5();

//Non Static variables accessed using object obj

System.out.println("i:"+obj.i);

System.out.println("s:"+obj.s);

}

}

**Output:**

i:0

s:null

**Static Variables**

* Static variables are also known as Class Variables.
* Such variables get default values based on the data type.
* Data stored in static variables is common for all the objects( or instances ) of that Class.
* Memory allocation for such variables only happens once when the class is loaded in the memory.
* These variables can be accessed in any other class using class name.
* Unlike non-static variables, such variables can be accessed directly in static and non-static methods.

**Example 1: Static variables can be accessed without reference in Static method**

class Example7{

staticint var1;

static String var2;

//Its a Static Method

public static void main(String args[])

{

System.out.println("Var1 is:"+Var1);

System.out.println("Var2 is:"+Var2);

}

}

Output:

Var1 is:0

Var2 is:null

**As you can see in the above example that both the variables are accessed in void main method without any object(reference).**

# Access Modifiers in java

There are 4 types of java access modifiers:

1. private
2. default
3. protected
4. public

**1) private access modifier**

|  |
| --- |
| The private access modifier is accessible only within class. |

### 2) default access modifier

|  |
| --- |
| If you don't use any modifier, it is treated as default bydefault. The default modifier is accessible only within package |

**3) protected access modifier**

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

**4) public access modifier**

|  |
| --- |
| The public access modifier is accessible everywhere. It has the widest scope among all other modifiers. |

Let's understand the access modifiers by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | within class | within package | outside package by subclass only | outside package |
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |

# Final Keyword In Java

## 1) final variable

final variables are nothing but constants. We cannot change the value of a final variable once it is initialized.

**Blank final variable**

A final variable that is not initialized at the time of declaration is known as blank final variable. We must initialize the blank final variable in constructor of the class otherwise it will throw a compilation error

**Blank static final variable**

A static final variable that is not initialized during declaration can only be initialized in [static block](http://beginnersbook.com/2013/04/java-static-class-block-methods-variables/)

## 2) final method

A final method cannot be overridden. Which means even though a sub class can call the final method of parent class without any issues but it cannot override it.

## 3) final class

We cannot extend a final class.

**Points to Remember:**

1) A [constructor](http://beginnersbook.com/2013/03/constructors-in-java/) cannot be declared as final.  
2) Local final variable must be initializing during declaration.  
3) All variables declared in an [interface](http://beginnersbook.com/2013/05/java-interface/) are by default final.  
4) We cannot change the value of a final variable.  
5) A final method cannot be overridden.  
6) A final class not be inherited.  
7) If method parameters are declared final then the value of these parameters cannot be changed.  
8) It is a good practice to name final variable in all CAPS.  
9) final, [finally](http://beginnersbook.com/2013/04/java-finally-block/) and finalize are three different terms. finally is used in exception handling and finalize is a method that is called by JVM during [garbage collection](http://beginnersbook.com/2013/04/java-garbage-collection/).

# Super Keyword In Java

super is a keyword in java which refers to the immediate super class object.

## *****Use of super keyword in java:*****

1. **1. super can be used to call immediate super class constructor (constructor chaining).**
2. **2. super can be used to call immediate super class method on a subclass object from a subclass method.**
3. **3. super can be used to access immediate super class instance variable.**

## *****1. super can be used to call immediate super class constructor (constructor chaining).*****

## *****Example:*****

**SuperExample1.java**

|  |
| --- |
| **class** Display {  Display(){  System.out.println("Super class constructor called.");  }  }    **publicclass** SuperExample1 **extends** Display {  SuperExample1(){  *//super keyword will call super class constructor.*  **super**();    System.out.println("Current class constructor called.");  }    **publicstaticvoid** main(Stringargs[]){  SuperExample1 obj=**new** SuperExample1();  }  } |

## *****Output:*****

|  |
| --- |
| **Superclass** constructor called.  Current**class** constructor called. |

## *****If super is not used explicitly compiler will automatically add super as the first statement.*****

## *****Example:*****

**SuperExample2.java**

|  |
| --- |
| **class** Display {  Display(){  System.out.println("Super class constructor called.");  }  }    **publicclass** SuperExample2 **extends** Display {  SuperExample2(){  *//compiler automatically add super here.*  System.out.println("Current class constructor called.");  }    **publicstaticvoid** main(Stringargs[]){  SuperExample2 obj=**new** SuperExample2();  }  } |

## *****Output:*****

|  |
| --- |
| **Superclass** constructor called.  Current**class** constructor called. |

## *****2. super can be used to call immediate super class method on a subclass object from a subclass method.*****

##### If super class and subclass have same methods and that method is called from subclass method than subclass method is called.

## *****Example:*****

**SuperExample4.java**

|  |
| --- |
| **class** Display {  **publicvoid** display(){  System.out.println("display method of super class.");  }  }    **class** Show **extends** Display {  **publicvoid** display(){  System.out.println("display method of sub class.");  }    **publicvoid** show(){  System.out.println("show method of sub class.");  *//subclass display method is called.*  display();  }  }    **publicclass** SuperExample4 {  **publicstaticvoid** main(Stringargs[]){  *//create Show class object.*  Show obj=**new** Show();  *//method call*  obj.show();  }  } |

## *****Output:*****

|  |
| --- |
| show method of sub **class**.  display method of sub **class**. |

**Above problem can be solved with super keyword.**

## *****Example:*****

**SuperExample3.java**

|  |
| --- |
| **class** Display {  **publicvoid** display(){  System.out.println("display method of super class.");  }  }    **class** Show **extends** Display {  **publicvoid** display(){  System.out.println("display method of sub class.");  }    **publicvoid** show(){  System.out.println("show method of sub class.");  *//super class display method is called.*  **super**.display();  }  }    **publicclass** SuperExample3 {  **publicstaticvoid** main(Stringargs[]){  *//create Show class object.*  Show obj=**new** Show();  *//method call*  obj.show();  }  } |

## *****Output:*****

|  |
| --- |
| show method of sub **class**.  display method of **superclass**. |

## *****If super class and subclass not have same methods and method of super class is called from subclass method than super class method is called. There is no need of super keyword.*****

## *****Example:*****

**SuperExample5.java**

|  |
| --- |
| **class** Display {  **publicvoid** display(){  System.out.println("display method of super class.");  }  }    **class** Show **extends** Display {    **publicvoid** show(){  System.out.println("show method of sub class.");  *//no need of super keyword here.*  display();  }  }    **publicclass** SuperExample5 {  **publicstaticvoid** main(Stringargs[]){  *//create Show class object.*  Show obj=**new** Show();  *//method call*  obj.show();  }  } |

## *****Output:*****

|  |
| --- |
| show method of sub **class**.  display method of **superclass**. |

## *****3. super can be used to access immediate super class instance variable.*****

##### If super class and subclass have same instance variables and that variable is called from subclass than subclass instance variable will be referred.

**SuperExample6.java**

|  |
| --- |
| **class** Display {  **int**num=100;  }    **class** Show **extends** Display {  **int**num=200;    **publicvoid** show(){  *//sub class instance variable will be referred.*  System.out.println("num = "+num);  }  }    **publicclass** SuperExample6 {  **publicstaticvoid** main(Stringargs[]){  *//create Show class object.*  Show obj=**new** Show();  *//method call*  obj.show();  }  } |

## *****Output:*****

|  |
| --- |
| num=200 |

##### Above problem can be solved with super keyword.

## *****Example:*****

**SuperExample7.java**

|  |
| --- |
| **class** Display {  **int**num=100;  }    **class** Show **extends** Display {  **int**num=200;    **publicvoid** show(){  *//super class instance variable will be referred.*  System.out.println("num = "+**super**.num);  }  }    **publicclass** SuperExample7 {  **publicstaticvoid** main(Stringargs[]){  *//create Show class object.*  Show obj=**new** Show();  *//method call*  obj.show();  }  } |

## *****Output:*****

|  |
| --- |
| num=100 |

## *****If sub class and super class instance variables are not same than there is no need of super keyword.*****

**SuperExample8.java**

|  |
| --- |
| **class** Display {  **int**num=100;  }    **class** Show **extends** Display {    **publicvoid** show(){  *//super class instance variable will be referred.*  System.out.println("num = "+num);  }  }    **publicclass** SuperExample8 {  **publicstaticvoid** main(Stringargs[]){  *//create Show class object.*  Show obj=**new** Show();  *//method call*  obj.show();  }  } |

## *****Output:*****

|  |
| --- |
| num=100 |

### this keyword

* **this** keyword is used to refer to current object.
* **this** is always a reference to the object on which method was invoked.
* **this** can be used to invoke current class constructor.
* **this** can be passed as an argument to another method.

*Example* :

class Box

{

Double width, weight, dept;

Box (double w, double h, double d)

{

this.width = w;

this.height = h;

this.depth = d;

}

}

Here the **this** is used to initialize member of current object.

#### The this is used to call overloaded constructor in java

class Car

{

private String name;

public Car()

{

this("BMW"); //oveloaded constructor is called.

}

public Car(Stting n)

{

this.name=n; //member is initialized using this.

}

}

#### The this is also used to call Method of that class.

public void getName()

{

System.out.println("Studytonight");

}

public void display()

{

this.getName();

System.out.println();

}

#### this is used to return current Object

public Car getCar()

{

return this;

}