Java OOPs Concepts

In this page, we will learn about basics of OOPs. Object Oriented Programming is a paradigm that provides many concepts such as**inheritance**, **data binding**, **polymorphism** etc.

**Simula** is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

**Smalltalk** is considered as the first truly object-oriented programming language.

## **OOPs (Object Oriented Programming System)**

**Object** means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

* Object
* Class
* Inheritance
* Polymorphism
* Abstraction
* Encapsulation

#### Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

#### Class

**Collection of objects** is called class. It is a logical entity.

#### Inheritance

**When one object acquires all the properties and behaviours of parent object** i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

#### Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

#### Abstraction

**Hiding internal details and showing functionality** is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

#### Encapsulation

**Binding (or wrapping) code and data together into a single unit is known as encapsulation**. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

### **Advantage of OOPs over Procedure-oriented programming language**

|  |
| --- |
| 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows. |
| 2)OOPs provides data hiding whereas in Procedure-oriented prgramming language a global data can be accessed from anywhere. |
| 3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language. |

|  |  |
| --- | --- |
| Global Data | Object Data |

### What is difference between object-oriented programming language and object-based programming language?

|  |
| --- |
| Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages. |

# Java Naming conventions

Java **naming convention** is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method etc.

But, it is not forced to follow. So, it is known as convention not rule.

All the classes, interfaces, packages, methods and fields of java programming language are given according to java naming convention.

## **Advantage of naming conventions in java**

By using standard Java naming conventions, you make your code easier to read for yourself and for other programmers. Readability of Java program is very important. It indicates that **less time** is spent to figure out what the code does.

|  |  |
| --- | --- |
| Name | Convention |
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |

## **CamelCase in java naming conventions**

Java follows camelcase syntax for naming the class, interface, method and variable.

If name is combined with two words, second word will start with uppercase letter always e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

# Object and Class in Java

In this page, we will learn about java objects and classes. In object-oriented programming technique, we design a program using objects and classes.

Object is the physical as well as logical entity whereas class is the logical entity only.

### **Object in Java**



An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.

An object has three characteristics:

* **state:** represents data (value) of an object.
* **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
* **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

|  |
| --- |
| For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior. |

|  |
| --- |
| **Object is an instance of a class.** Class is a template or blueprint from which objects are created. So object is the instance(result) of a class. |

### **Class in Java**

|  |
| --- |
| A class is a group of objects that has common properties. It is a template or blueprint from which objects are created. |

A class in java can contain:

* **data member**
* **method**
* **constructor**
* **block**
* **class and interface**

### Syntax to declare a class:

1. **class** <class\_name>{
2. data member;
3. method;
4. }

### **Simple Example of Object and Class**

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

**class** Student1{

**int** id;//data member (also instance variable)

 String name;//data member(also instance variable)

**public** **static** **void** main(String args[]){

  Student1 s1=**new** Student1();//creating an object of Student

  System.out.println(s1.id);

  System.out.println(s1.name);

 }

}

Output:0 null

### Instance variable in Java

|  |
| --- |
| A variable that is created inside the class but outside the method, is known as instance variable.Instance variable doesn't get memory at compile time.It gets memory at runtime when object(instance) is created.That is why, it is known as instance variable. |

### Method in Java

|  |
| --- |
| In java, a method is like function i.e. used to expose behaviour of an object. |

#### Advantage of Method

* Code Reusability
* Code Optimization

### new keyword

|  |
| --- |
| The new keyword is used to allocate memory at runtime. |

### **Example of Object and class that maintains the records of students**

|  |
| --- |
| In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method on it. Here, we are displaying the state (data) of the objects by invoking the displayInformation method. |

**class** Student2{

**int** rollno;

 String name;

**void** insertRecord(**int** r, String n){  //method

  rollno=r;

  name=n;

 }

**void** displayInformation(){System.out.println(rollno+" "+name);}//method

**public** **static** **void** main(String args[]){

  Student2 s1=**new** Student2();

  Student2 s2=**new** Student2();

  s1.insertRecord(111,"Karan");

  s2.insertRecord(222,"Aryan");

  s1.displayInformation();

  s2.displayInformation();

 }

}

111 Karan

222 Aryan



|  |
| --- |
| As you see in the above figure, object gets the memory in Heap area and reference variable refers to the object allocated in the Heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory. |

### **Another Example of Object and Class**

|  |
| --- |
| There is given another example that maintains the records of Rectangle class. Its exaplanation is same as in the above Student class example. |

**class** Rectangle{

**int** length;

**int** width;

**void** insert(**int** l,**int** w){

  length=l;

  width=w;

 }

**void** calculateArea(){System.out.println(length\*width);}

**public** **static** **void** main(String args[]){

  Rectangle r1=**new** Rectangle();

  Rectangle r2=**new** Rectangle();

  r1.insert(11,5);

  r2.insert(3,15);

  r1.calculateArea();

  r2.calculateArea();

}

}

Output:55

45

### What are the different ways to create an object in Java?

|  |
| --- |
| There are many ways to create an object in java. They are:   * By new keyword * By newInstance() method * By clone() method * By factory method etc.   We will learn, these ways to create the object later. |

### **Annonymous object**

|  |
| --- |
| Annonymous simply means nameless.An object that have no reference is known as annonymous object. |
| If you have to use an object only once, annonymous object is a good approach. |

**class** Calculation{

**void** fact(**int**  n){

**int** fact=1;

**for**(**int** i=1;i<=n;i++){

   fact=fact\*i;

  }

 System.out.println("factorial is "+fact);

}

**public** **static** **void** main(String args[]){

**new** Calculation().fact(5);//calling method with annonymous object

}

}

Output:Factorial is 120

### **Creating multiple objects by one type only**

|  |
| --- |
| We can create multiple objects by one type only as we do in case of primitives. |

Rectangle r1=**new** Rectangle(),r2=**new** Rectangle();//creating two objects

|  |
| --- |
| Let's see the example: |

**class** Rectangle{

**int** length;

**int** width;

**void** insert(**int** l,**int** w){

  length=l;

  width=w;

 }

**void** calculateArea(){System.out.println(length\*width);}

**public** **static** **void** main(String args[]){

  Rectangle r1=**new** Rectangle(),r2=**new** Rectangle();//creating two objects

  r1.insert(11,5);

  r2.insert(3,15);

  r1.calculateArea();

  r2.calculateArea();

}

}

Output:55

45

# Method Overloading in Java

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

## java method overloadingAdvantage of method overloading?

Method overloading **increases the readability of the program**.

### Different ways to overload the method

|  |
| --- |
| There are two ways to overload the method in java |

|  |
| --- |
| 1. By changing number of arguments 2. By changing the data type |

#### In java, Methood Overloading is not possible by changing the return type of the method.

### **1) Example of Method Overloading by changing the no. of arguments**

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

**class** Calculation{

**void** sum(**int** a,**int** 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[]){

  Calculation obj=**new** Calculation();

  obj.sum(10,10,10);

  obj.sum(20,20);

  }

}

Output:30

40

### **2) Example of Method Overloading by changing data type of argument**

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

**class** Calculation2{

**void** sum(**int** a,**int** b){System.out.println(a+b);}

**void** sum(**double** a,**double** b){System.out.println(a+b);}

**public** **static** **void** main(String args[]){

  Calculation2 obj=**new** Calculation2();

  obj.sum(10.5,10.5);

  obj.sum(20,20);

  }

}

Output:21.0

40

### Que) Why Method Overloaing is not possible by changing the return type of method?

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

**class** Calculation3{

**int** sum(**int** a,**int** b){System.out.println(a+b);}

**double** sum(**int** a,**int** b){System.out.println(a+b);}

**public** **static** **void** main(String args[]){

  Calculation3 obj=**new** Calculation3();

**int** result=obj.sum(20,20); //Compile Time Error

  }

}

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

### Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

**class** Overloading1{

**public** **static** **void** main(**int** a){

  System.out.println(a);

  }

**public** **static** **void** main(String args[]){

  System.out.println("main() method invoked");

  main(10);

  }

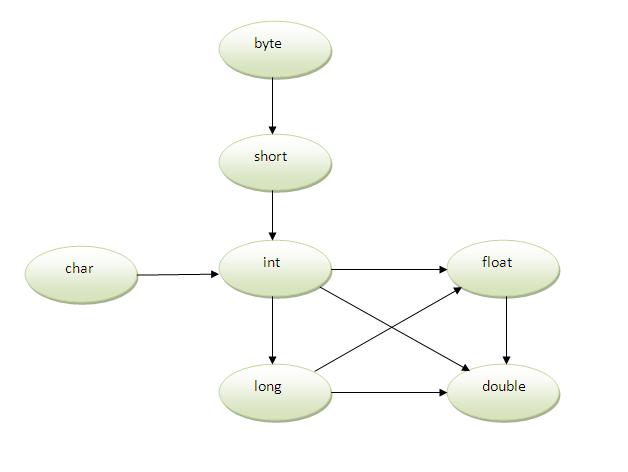
}

Output:main() method invoked

10

### **Method Overloading and TypePromotion**

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



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int,long,float or double. The char datatype can be promoted to int,long,float or double and so on.

### **Example of Method Overloading with TypePromotion**

**class** OverloadingCalculation1{

**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[]){

  OverloadingCalculation1 obj=**new** OverloadingCalculation1();

  obj.sum(20,20);//now second int literal will be promoted to long

  obj.sum(20,20,20);

  }

}

Output:40

60

### **Example of Method Overloading with TypePromotion if matching found**

If there are matching type arguments in the method, type promotion is not performed.

**class** OverloadingCalculation2{

**void** sum(**int** a,**int** b){System.out.println("int arg method invoked");}

**void** sum(**long** a,**long** b){System.out.println("long arg method invoked");}

**public** **static** **void** main(String args[]){

  OverloadingCalculation2 obj=**new** OverloadingCalculation2();

  obj.sum(20,20);//now int arg sum() method gets invoked

  }

}

Output:int arg method invoked

### **Example of Method Overloading with TypePromotion in case ambiguity**

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

**class** OverloadingCalculation3{

**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[]){

  OverloadingCalculation3 obj=**new** OverloadingCalculation3();

  obj.sum(20,20);//now ambiguity

  }

}

Output:Compile Time Error

#### One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitely.

# Constructor in Java

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Constructor in java** is a special type of method that is used to initialize the object.  Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor. **Rules for creating java constructor** There are basically two rules defined for the constructor.   1. Constructor name must be same as its class name 2. Constructor must have no explicit return type  **Types of java constructors** There are two types of constructors:   1. Default constructor (no-arg constructor) 2. Parameterized constructor   java constructor **Java Default Constructor**  |  | | --- | | A constructor that have no parameter is known as default constructor. |  Syntax of default constructor: <class\_name>(){} **Example of default constructor**  |  | | --- | | In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation. |   **class** Bike1{  Bike1(){System.out.println("Bike is created");}  **public** **static** **void** main(String args[]){  Bike1 b=**new** Bike1();  }  }  Output:  Bike is created Rule: If there is no constructor in a class, compiler automatically creates a default constructor. default constructor Q) What is the purpose of default constructor? Default constructor provides the default values to the object like 0, null etc. depending on the type. **Example of default constructor that displays the default values** **class** Student3{  **int** id;  String name;    **void** display(){System.out.println(id+" "+name);}    **public** **static** **void** main(String args[]){  Student3 s1=**new** Student3();  Student3 s2=**new** Student3();  s1.display();  s2.display();  }  }  Output:  0 null  0 null  **Explanation:**In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor. **Java parameterized constructor**  |  | | --- | | A constructor that have parameters is known as parameterized constructor. |  Why use parameterized constructor?  |  | | --- | | Parameterized constructor is used to provide different values to the distinct objects. |  **Example of parameterized constructor**  |  | | --- | | In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor. |   **class** Student4{  **int** id;      String name;        Student4(**int** i,String n){      id = i;      name = n;      }  **void** display(){System.out.println(id+" "+name);}    **public** **static** **void** main(String args[]){      Student4 s1 = **new** Student4(111,"Karan");      Student4 s2 = **new** Student4(222,"Aryan");      s1.display();      s2.display();     }  }  Output:  111 Karan  222 Aryan **Constructor Overloading in Java**  |  | | --- | | Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. |  **Example of Constructor Overloading** **class** Student5{  **int** id;      String name;  **int** age;      Student5(**int** i,String n){      id = i;      name = n;      }      Student5(**int** i,String n,**int** a){      id = i;      name = n;      age=a;      }  **void** display(){System.out.println(id+" "+name+" "+age);}    **public** **static** **void** main(String args[]){      Student5 s1 = **new** Student5(111,"Karan");      Student5 s2 = **new** Student5(222,"Aryan",25);      s1.display();      s2.display();     }  }  Output:  111 Karan 0  222 Aryan 25 **Difference between constructor and method in java** There are many differences between constructors and methods. They are given below.   |  |  | | --- | --- | | Java Constructor | Java Method | | Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. | | Constructor must not have return type. | Method must have return type. | | Constructor is invoked implicitly. | Method is invoked explicitly. | | The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. | | Constructor name must be same as the class name. | Method name may or may not be same as class name. |  **Java Copy Constructor** There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor in C++.  There are many ways to copy the values of one object into another in java. They are:   * By constructor * By assigning the values of one object into another * By clone() method of Object class   In this example, we are going to copy the values of one object into another using java constructor. |

**class** Student6{

**int** id;

    String name;

    Student6(**int** i,String n){

    id = i;

    name = n;

    }

    Student6(Student6 s){

    id = s.id;

    name =s.name;

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student6 s1 = **new** Student6(111,"Karan");

    Student6 s2 = **new** Student6(s1);

    s1.display();

    s2.display();

   }

}

Output:

111 Karan

111 Karan

## **Copying values without constructor**

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

**class** Student7{

**int** id;

    String name;

    Student7(**int** i,String n){

    id = i;

    name = n;

    }

    Student7(){}

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student7 s1 = **new** Student7(111,"Karan");

    Student7 s2 = **new** Student7();

    s2.id=s1.id;

    s2.name=s1.name;

    s1.display();

    s2.display();

   }

}

Output:

111 Karan

111 Karan

### **Q) Does constructor return any value?**

**Ans:**yes, that is current class instance (You cannot use return type yet it returns a value).

### **Can constructor perform other tasks instead of initialization?**

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

# Java static keyword

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

## **1) Java static variable**

If you declare any variable as static, it is known static variable.

* The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc.
* The static variable gets memory only once in class area at the time of class loading.

### Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

#### Understanding problem without static variable

**class** Student{

**int** rollno;

     String name;

     String college="ITS";

}

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once.

#### Java static property is shared to all objects.

### **Example of static variable**

//Program of static variable

**class** Student8{

**int** rollno;

   String name;

**static** String college ="ITS";

   Student8(**int** r,String n){

   rollno = r;

   name = n;

   }

**void** display (){System.out.println(rollno+" "+name+" "+college);}

**public** **static** **void** main(String args[]){

 Student8 s1 = **new** Student8(111,"Karan");

 Student8 s2 = **new** Student8(222,"Aryan");

 s1.display();

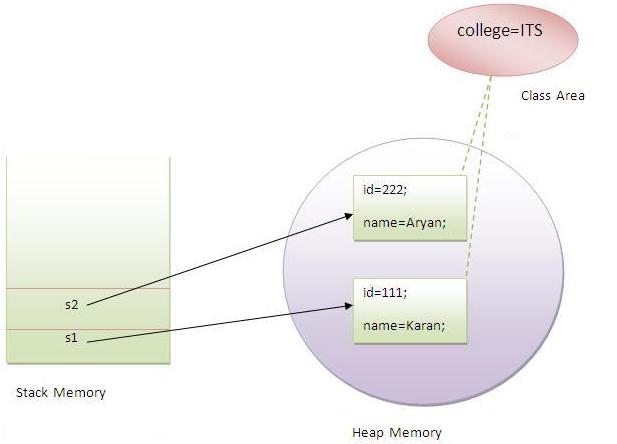
 s2.display();

 }

}

Output:111 Karan ITS

222 Aryan ITS



### **Program of counter without static variable**

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

**class** Counter{

**int** count=0;//will get memory when instance is created

Counter(){

count++;

System.out.println(count);

}

**public** **static** **void** main(String args[]){

Counter c1=**new** Counter();

Counter c2=**new** Counter();

Counter c3=**new** Counter();

 }

}

Output:1

1

1

### **Program of counter by static variable**

|  |
| --- |
| As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value. |

**class** Counter2{

**static** **int** count=0;//will get memory only once and retain its value

Counter2(){

count++;

System.out.println(count);

}

**public** **static** **void** main(String args[]){

Counter2 c1=**new** Counter2();

Counter2 c2=**new** Counter2();

Counter2 c3=**new** Counter2();

 }

}

Output:1

2

3

## **2) Java static method**

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* static method can access static data member and can change the value of it.

### **Example of static method**

//Program of changing the common property of all objects(static field).

**class** Student9{

**int** rollno;

     String name;

**static** String college = "ITS";

**static** **void** change(){

     college = "BBDIT";

     }

     Student9(**int** r, String n){

     rollno = r;

     name = n;

     }

**void** display (){System.out.println(rollno+" "+name+" "+college);}

**public** **static** **void** main(String args[]){

    Student9.change();

    Student9 s1 = **new** Student9 (111,"Karan");

    Student9 s2 = **new** Student9 (222,"Aryan");

    Student9 s3 = **new** Student9 (333,"Sonoo");

    s1.display();

    s2.display();

    s3.display();

    }

}

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

### **Another example of static method that performs normal calculation**

//Program to get cube of a given number by static method

**class** Calculate{

**static** **int** cube(**int** x){

**return** x\*x\*x;

  }

**public** **static** **void** main(String args[]){

**int** result=Calculate.cube(5);

  System.out.println(result);

  }

}

Output:125

### Restrictions for static method

|  |
| --- |
| There are two main restrictions for the static method. They are: |

|  |
| --- |
| 1. The static method can not use non static data member or call non-static method directly. 2. this and super cannot be used in static context. |

**class** A{

**int** a=40;//non static

**public** **static** **void** main(String args[]){

  System.out.println(a);

 }

}

Output:Compile Time Error

### **Q) why java main method is static?**

|  |
| --- |
| Ans) because object is not required to call static method if it were non-static method, jvm create object first then call main() method that will lead the problem of extra memory allocation. |

## **3) Java static block**

* Is used to initialize the static data member.
* It is executed before main method at the time of classloading.

### **Example of static block**

**class** A2{

**static**{System.out.println("static block is invoked");}

**public** **static** **void** main(String args[]){

   System.out.println("Hello main");

  }

}

Output:static block is invoked

Hello main

### Q) Can we execute a program without main() method?

Ans) Yes, one of the way is static block but in previous version of JDK not in JDK 1.7.

**class** A3{

**static**{

  System.out.println("static block is invoked");

  System.exit(0);

  }

}

Output:static block is invoked (if not JDK7)

In JDK7 and above, output will be:

Output:Error: Main method not found in class A3, please define the main method as:

public static void main(String[] args)

# this keyword in java

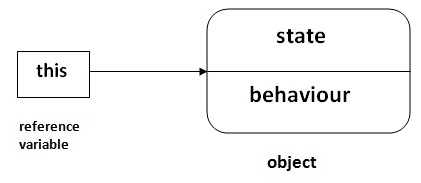
There can be a lot of usage of **java this keyword**. In java, this is a**reference variable** that refers to the current object.

## **Usage of java this keyword**

Here is given the 6 usage of java this keyword.

1. this keyword can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this keyword can be used to invoke current class method (implicitly)
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this keyword can also be used to return the current class instance.

**Suggestion:** If you are beginner to java, lookup only two usage of this keyword.



### **1) The this keyword can be used to refer current class instance variable.**

|  |
| --- |
| If there is ambiguity between the instance variable and parameter, this keyword resolves the problem of ambiguity. |

#### Understanding the problem without this keyword

|  |
| --- |
| Let's understand the problem if we don't use this keyword by the example given below: |

**class** Student10{

**int** id;

    String name;

    Student10(**int** id,String name){

    id = id;

    name = name;

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student10 s1 = **new** Student10(111,"Karan");

    Student10 s2 = **new** Student10(321,"Aryan");

    s1.display();

    s2.display();

    }

}

Output:0 null

0 null

|  |
| --- |
| In the above example, parameter (formal arguments) and instance variables are same that is why we are using this keyword to distinguish between local variable and instance variable. |

#### Solution of the above problem by this keyword

//example of this keyword

**class** Student11{

**int** id;

    String name;

    Student11(**int** id,String name){

**this**.id = id;

**this**.name = name;

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student11 s1 = **new** Student11(111,"Karan");

    Student11 s2 = **new** Student11(222,"Aryan");

    s1.display();

    s2.display();

}

}

Output111 Karan

222 Aryan



|  |
| --- |
| If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program: |

#### Program where this keyword is not required

**class** Student12{

**int** id;

    String name;

    Student12(**int** i,String n){

    id = i;

    name = n;

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student12 e1 = **new** Student12(111,"karan");

    Student12 e2 = **new** Student12(222,"Aryan");

    e1.display();

    e2.display();

}

}

Output:111 Karan

222 Aryan

### **2) this() can be used to invoked current class constructor.**

The this() constructor call can be used to invoke the current class constructor (constructor chaining). This approach is better if you have many constructors in the class and want to reuse that constructor.

//Program of this() constructor call (constructor chaining)

**class** Student13{

**int** id;

    String name;

    Student13(){System.out.println("default constructor is invoked");}

    Student13(**int** id,String name){

**this** ();//it is used to invoked current class constructor.

**this**.id = id;

**this**.name = name;

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student13 e1 = **new** Student13(111,"karan");

    Student13 e2 = **new** Student13(222,"Aryan");

    e1.display();

    e2.display();

   }

}

Output:

default constructor is invoked

default constructor is invoked

111 Karan

222 Aryan

### **Where to use this() constructor call?**

|  |
| --- |
| The this() constructor call should be used to reuse the constructor in the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword. |

**class** Student14{

**int** id;

    String name;

    String city;

    Student14(**int** id,String name){

**this**.id = id;

**this**.name = name;

    }

    Student14(**int** id,String name,String city){

**this**(id,name);//now no need to initialize id and name

**this**.city=city;

    }

**void** display(){System.out.println(id+" "+name+" "+city);}

**public** **static** **void** main(String args[]){

    Student14 e1 = **new** Student14(111,"karan");

    Student14 e2 = **new** Student14(222,"Aryan","delhi");

    e1.display();

    e2.display();

   }

}

Output:111 Karan null

222 Aryan delhi

#### Rule: Call to this() must be the first statement in constructor.

**class** Student15{

**int** id;

    String name;

    Student15(){System.out.println("default constructor is invoked");}

    Student15(**int** id,String name){

    id = id;

    name = name;

**this** ();//must be the first statement

    }

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    Student15 e1 = **new** Student15(111,"karan");

    Student15 e2 = **new** Student15(222,"Aryan");

    e1.display();

    e2.display();

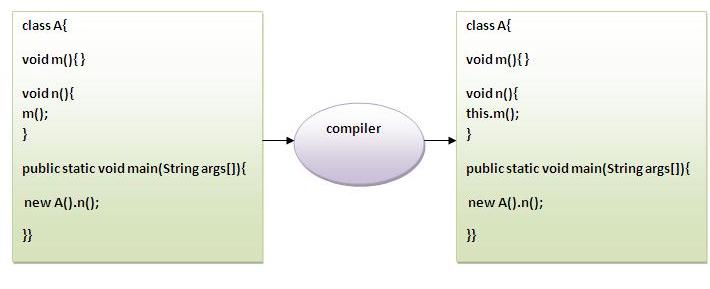
   }

}

Output:Compile Time Error

### **3)The this keyword can be used to invoke current class method (implicitly).**

|  |
| --- |
| You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method. Let's see the example |



**class** S{

**void** m(){

  System.out.println("method is invoked");

  }

**void** n(){

**this**.m();//no need because compiler does it for you.

  }

**void** p(){

  n();//complier will add this to invoke n() method as this.n()

  }

**public** **static** **void** main(String args[]){

  S s1 = **new** S();

  s1.p();

  }

}

Output:method is invoked

### **4) this keyword can be passed as an argument in the method.**

|  |
| --- |
| The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example: |

**class** S2{

**void** m(S2 obj){

  System.out.println("method is invoked");

  }

**void** p(){

  m(**this**);

  }

**public** **static** **void** main(String args[]){

  S2 s1 = **new** S2();

  s1.p();

  }

}

Output:method is invoked

### Application of this that can be passed as an argument:

|  |
| --- |
| In event handling (or) in a situation where we have to provide reference of a class to another one. |

|  |
| --- |
|  |

### **5) The this keyword can be passed as argument in the constructor call.**

|  |
| --- |
| We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example: |

**class** B{

  A4 obj;

  B(A4 obj){

**this**.obj=obj;

  }

**void** display(){

    System.out.println(obj.data);//using data member of A4 class

  }

}

**class** A4{

**int** data=10;

  A4(){

   B b=**new** B(**this**);

   b.display();

  }

**public** **static** **void** main(String args[]){

   A4 a=**new** A4();

  }

}

Output:10

### **6) The this keyword can be used to return current class instance.**

|  |
| --- |
| We can return the this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example: |

### Syntax of this that can be returned as a statement

return\_type method\_name(){

**return** **this**;

}

### **Example of this keyword that you return as a statement from the method**

**class** A{

A getA(){

**return** **this**;

}

**void** msg(){System.out.println("Hello java");}

}

**class** Test1{

**public** **static** **void** main(String args[]){

**new** A().getA().msg();

}

}

Output:Hello java

### **Proving this keyword**

|  |
| --- |
| Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same. |

**class** A5{

**void** m(){

System.out.println(**this**);//prints same reference ID

}

**public** **static** **void** main(String args[]){

A5 obj=**new** A5();

System.out.println(obj);//prints the reference ID

obj.m();

}

}

Output:A5@22b3ea59

A5@22b3ea59