Java OOPs Concept

Object-Oriented Programming is a paradigm that provides many concepts such as **inheritance**, **data binding**, **polymorphism**, etc.

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

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

The popular object-oriented languages are Java, C#, PHP, Python, C++ etc.

The main aim of object-oriented programming is to implement real-world entities for example object, classes, abstraction, inheritance, polymorphism, etc.

OOPs (Object-Oriented Programming System)

**Object** means a real-world entity such as a pen, chair, table, computer, watch, 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:

o Object

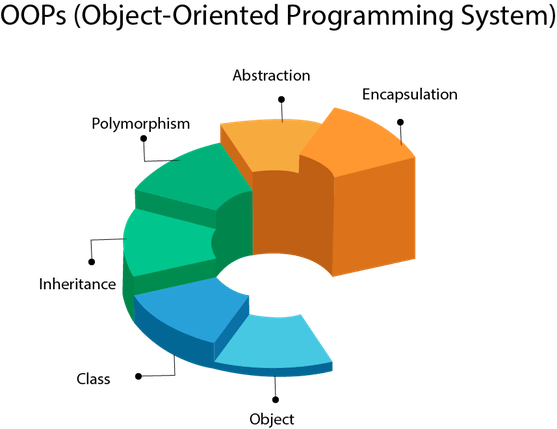
o Class

o Inheritance

o Polymorphism

o Abstraction

o Encapsulation



Object



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

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class

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

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Inheritance

*When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.



Polymorphism

If *one task is performed by different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

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

Another example can be to speak something; for example, a cat speaks meow, 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 are 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 whereas in a procedure-oriented programming language it is not easy to manage if code grows as project size increases.

2) OOPs provides data hiding whereas in a procedure-oriented programming language a global data can be accessed from anywhere.

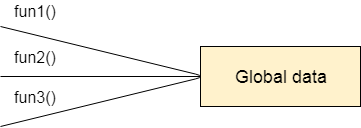


Figure: Data Representation in Procedure-Oriented Programming

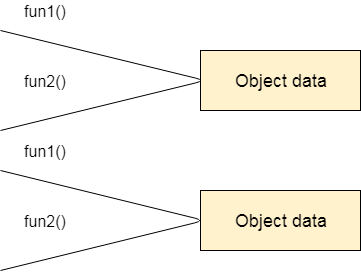


Figure: Data Representation in Object-Oriented Programming

3) OOPs provides the 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.

What is the difference between an 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. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

Advantage of naming conventions in java

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

The following are the key rules that must be followed by every identifier:

* The name must not contain any white spaces.
* The name should not start with special characters like & (ampersand), $ (dollar), \_ (underscore).

Let's see some other rules that should be followed by identifiers.

Class

* It should start with the uppercase letter.
* It should be a noun such as Color, Button, System, Thread, etc.
* Use appropriate words, instead of acronyms.
* **Example: -**

**public** **class** Employee

{

//code snippet

}

Interface

* It should start with the uppercase letter.
* It should be an adjective such as Runnable, Remote, ActionListener.
* Use appropriate words, instead of acronyms.
* **Example: -**

**interface** Printable

{

//code snippet

}

Method

* It should start with lowercase letter.
* It should be a verb such as main(), print(), println().
* If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
* **Example:-**

**class** Employee

{

//method

**void** draw()

{

//code snippet

}

}

Variable

* It should start with a lowercase letter such as id, name.
* It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore).
* If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
* Avoid using one-character variables such as x, y, z.
* **Example :-**

**class** Employee

{

//variable

**int** id;

//code snippet

}

Package

* It should be a lowercase letter such as java, lang.
* If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
* **Example :-**

**package** com.javatpoint; //package

**class** Employee

{

//code snippet

}

Constant

* It should be in uppercase letters such as RED, YELLOW.
* If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY.
* It may contain digits but not as the first letter.
* **Example :-**

**class** Employee

{

//constant

**static** **final** **int** MIN\_AGE = 18;

//code snippet

}

CamelCase in java naming conventions

Java follows camel-case syntax for naming the class, interface, method, and variable.

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

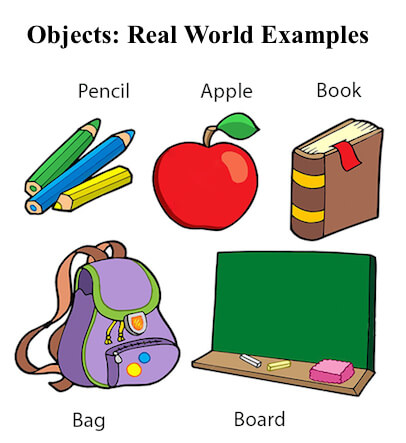
Objects and Classes in Java

1. [Object in Java](https://www.javatpoint.com/object-and-class-in-java#object)
2. [Class in Java](https://www.javatpoint.com/object-and-class-in-java#class)
3. [Instance Variable in Java](https://www.javatpoint.com/object-and-class-in-java#objectinstancevariable)
4. [Method in Java](https://www.javatpoint.com/object-and-class-in-java#objectmethod)
5. [Example of Object and class that maintains the records of student](https://www.javatpoint.com/object-and-class-in-java#objectex2)
6. [Annonymous Object](https://www.javatpoint.com/object-and-class-in-java#objectannonymous)

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

An object in Java is the physical as well as logical entity whereas a class in Java is a logical entity only.

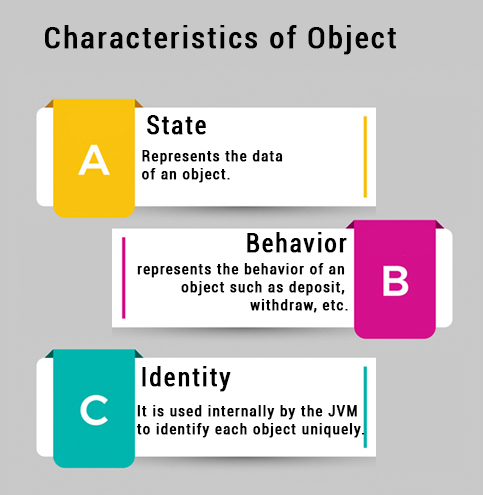
What is an object in Java



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 (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

* **State:** represents the data (value) of an object.
* **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
* **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, 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, known as its state. It is used to write, so writing is its behavior.

**An object is an instance of a class.** A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

**Object Definitions:**

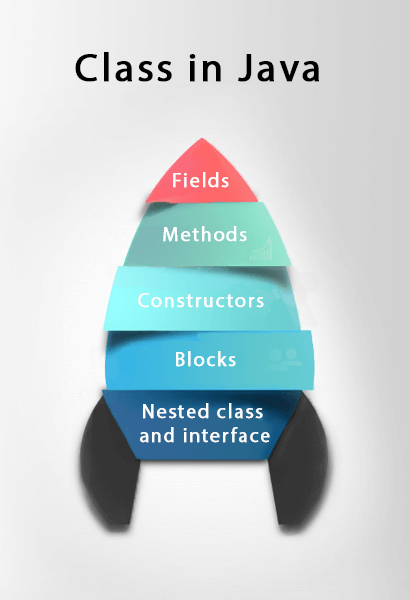
* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**



Syntax to declare a class:

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

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

* Code Reusability
* Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object and Class Example: main within the class

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

Here, we are creating a main() method inside the class.

*File: Student.java*

1. //Java Program to illustrate how to define a class and fields
2. //Defining a Student class.
3. **class** Student{
4. //defining fields
5. **int** id;//field or data member or instance variable
6. String name;
7. //creating main method inside the Student class
8. **public** **static** **void** main(String args[]){
9. //Creating an object or instance
10. Student s1=**new** Student();//creating an object of Student
11. //Printing values of the object
12. System.out.println(s1.id);//accessing member through reference variable
13. System.out.println(s1.name);
14. }
15. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student" \t "_blank)**

Output:

0

null

Object and Class Example: main outside the class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

*File: TestStudent1.java*

1. //Java Program to demonstrate having the main method in
2. //another class
3. //Creating Student class.
4. **class** Student{
5. **int** id;
6. String name;
7. }
8. //Creating another class TestStudent1 which contains the main method
9. **class** TestStudent1{
10. **public** **static** **void** main(String args[]){
11. Student s1=**new** Student();
12. System.out.println(s1.id);
13. System.out.println(s1.name);
14. }
15. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestStudent1" \t "_blank)**

Output:

0

null

3 Ways to initialize object

There are 3 ways to initialize object in java.

1. By reference variable
2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

Initializing an object means storing data into the object. Let's see a simple example where we are going to initialize the object through a reference variable.

*File: TestStudent2.java*

1. **class** Student{
2. **int** id;
3. String name;
4. }
5. **class** TestStudent2{
6. **public** **static** **void** main(String args[]){
7. Student s1=**new** Student();
8. s1.id=101;
9. s1.name="Sonoo";
10. System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestStudent2" \t "_blank)**

Output:

101 Sonoo

We can also create multiple objects and store information in it through reference variable.

*File: TestStudent3.java*

1. **class** Student{
2. **int** id;
3. String name;
4. }
5. **class** TestStudent3{
6. **public** **static** **void** main(String args[]){
7. //Creating objects
8. Student s1=**new** Student();
9. Student s2=**new** Student();
10. //Initializing objects
11. s1.id=101;
12. s1.name="Sonoo";
13. s2.id=102;
14. s2.name="Amit";
15. //Printing data
16. System.out.println(s1.id+" "+s1.name);
17. System.out.println(s2.id+" "+s2.name);
18. }
19. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestStudent3" \t "_blank)**

Output:

101 Sonoo

102 Amit

2) Object and Class Example: Initialization through method

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

*File: TestStudent4.java*

1. **class** Student{
2. **int** rollno;
3. String name;
4. **void** insertRecord(**int** r, String n){
5. rollno=r;
6. name=n;
7. }
8. **void** displayInformation(){System.out.println(rollno+" "+name);}
9. }
10. **class** TestStudent4{
11. **public** **static** **void** main(String args[]){
12. Student s1=**new** Student();
13. Student s2=**new** Student();
14. s1.insertRecord(111,"Karan");
15. s2.insertRecord(222,"Aryan");
16. s1.displayInformation();
17. s2.displayInformation();
18. }
19. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestStudent4" \t "_blank)**

Output:

111 Karan

222 Aryan



As you can see in the above figure, object gets the memory in heap memory area. The 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.

3) Object and Class Example: Initialization through a constructor

We will learn about constructors in java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

*File: TestEmployee.java*

1. **class** Employee{
2. **int** id;
3. String name;
4. **float** salary;
5. **void** insert(**int** i, String n, **float** s) {
6. id=i;
7. name=n;
8. salary=s;
9. }
10. **void** display(){System.out.println(id+" "+name+" "+salary);}
11. }
12. **public** **class** TestEmployee {
13. **public** **static** **void** main(String[] args) {
14. Employee e1=**new** Employee();
15. Employee e2=**new** Employee();
16. Employee e3=**new** Employee();
17. e1.insert(101,"ajeet",45000);
18. e2.insert(102,"irfan",25000);
19. e3.insert(103,"nakul",55000);
20. e1.display();
21. e2.display();
22. e3.display();
23. }
24. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestEmployee" \t "_blank)**

Output:

101 ajeet 45000.0

102 irfan 25000.0

103 nakul 55000.0

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

*File: TestRectangle1.java*

1. **class** Rectangle{
2. **int** length;
3. **int** width;
4. **void** insert(**int** l, **int** w){
5. length=l;
6. width=w;
7. }
8. **void** calculateArea(){System.out.println(length\*width);}
9. }
10. **class** TestRectangle1{
11. **public** **static** **void** main(String args[]){
12. Rectangle r1=**new** Rectangle();
13. Rectangle r2=**new** Rectangle();
14. r1.insert(11,5);
15. r2.insert(3,15);
16. r1.calculateArea();
17. r2.calculateArea();
18. }
19. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestRectangle1" \t "_blank)**

Output:

55

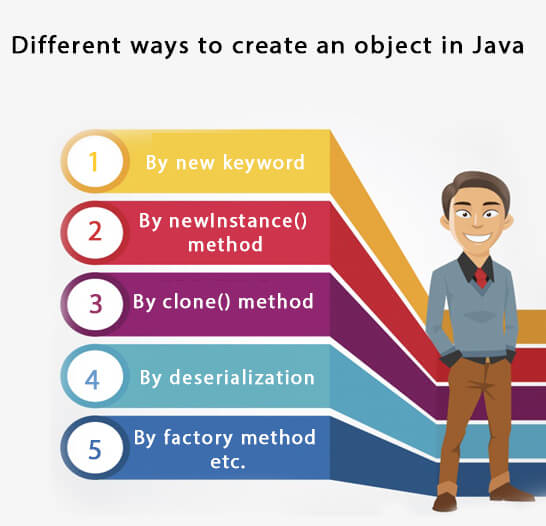
45

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 deserialization
* By factory method etc.

We will learn these ways to create object later.



Anonymous object

Anonymous simply means nameless. An object which has no reference is known as an anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, an anonymous object is a good approach. For example:

**new** Calculation();//anonymous object

Calling method through a reference:

Calculation c=**new** Calculation();

c.fact(5);

Calling method through an anonymous object

**new** Calculation().fact(5);

Let's see the full example of an anonymous object in Java.

**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 anonymous 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.

Initialization of primitive variables:

**int** a=10, b=20;

Initialization of refernce variables:

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

Let's see the example:

//Java Program to illustrate the use of Rectangle class which

//has length and width data members

**class** Rectangle{

**int** length;

**int** width;

**void** insert(**int** l,**int** w){

  length=l;

  width=w;

 }

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

}

**class** TestRectangle2{

**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

Real World Example: Account

*File: TestAccount.java*

//Java Program to demonstrate the working of a banking-system

//where we deposit and withdraw amount from our account.

//Creating an Account class which has deposit() and withdraw() methods

**class** Account{

**int** acc\_no;

String name;

**float** amount;

//Method to initialize object

**void** insert(**int** a,String n,**float** amt){

acc\_no=a;

name=n;

amount=amt;

}

//deposit method

**void** deposit(**float** amt){

amount=amount+amt;

System.out.println(amt+" deposited");

}

//withdraw method

**void** withdraw(**float** amt){

**if**(amount<amt){

System.out.println("Insufficient Balance");

}**else**{

amount=amount-amt;

System.out.println(amt+" withdrawn");

}

}

//method to check the balance of the account

**void** checkBalance(){System.out.println("Balance is: "+amount);}

//method to display the values of an object

**void** display(){System.out.println(acc\_no+" "+name+" "+amount);}

}

//Creating a test class to deposit and withdraw amount

**class** TestAccount{

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

Account a1=**new** Account();

a1.insert(832345,"Ankit",1000);

a1.display();

a1.checkBalance();

a1.deposit(40000);

a1.checkBalance();

a1.withdraw(15000);

a1.checkBalance();

}}

Output:

832345 Ankit 1000.0

Balance is: 1000.0

40000.0 deposited

Balance is: 41000.0

15000.0 withdrawn

Balance is: 26000.0

Constructors in Java

1. [Types of constructors](https://www.javatpoint.com/java-constructor#constypes)
   1. [Default Constructor](https://www.javatpoint.com/java-constructor#consdef)
   2. [Parameterized Constructor](https://www.javatpoint.com/java-constructor#conspara)
2. [Constructor Overloading](https://www.javatpoint.com/java-constructor#consoverloading)
3. [Does constructor return any value?](https://www.javatpoint.com/java-constructor#consdoesreturn)
4. [Copying the values of one object into another](https://www.javatpoint.com/java-constructor#conscopy)
5. [Does constructor perform other tasks instead of the initialization](https://www.javatpoint.com/java-constructor#consothertask)

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the object is created, and memory is allocated for the object.

It is a special type of method which is used to initialize the object.

When is a constructor called

Every time an object is created using new() keyword, at least one constructor is called. It calls a default constructor.

**Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

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. |

//Java Program to create and call a default constructor

**class** Bike1{

//creating a default constructor

Bike1(){System.out.println("Bike is created");}

//main method

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

//calling a default constructor

Bike1 b=**new** Bike1();

}

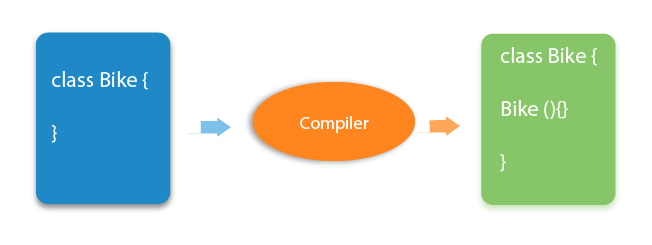
}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Bike1" \t "_blank)**

Output:

Bike is created

Rule: If there is no constructor in a class, compiler automatically creates a default constructor.



Q) What is the purpose of a default constructor?

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example of default constructor that displays the default values

//Let us see another example of default constructor

//which displays the default values

**class** Student3{

**int** id;

String name;

//method to display the value of id and name

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

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

//creating objects

Student3 s1=**new** Student3();

Student3 s2=**new** Student3();

//displaying values of the object

s1.display();

s2.display();

}

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student3" \t "_blank)**

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 which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

The parameterized constructor is used to provide different values to the distinct objects. However, you can provide the same values also.

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.

//Java Program to demonstrate the use of parameterized constructor

**class** Student4{

**int** id;

    String name;

    //creating a parameterized constructor

    Student4(**int** i,String n){

    id = i;

    name = n;

    }

    //method to display the values

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

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

    //creating objects and passing values

    Student4 s1 = **new** Student4(111,"Karan");

    Student4 s2 = **new** Student4(222,"Aryan");

    //calling method to display the values of object

    s1.display();

    s2.display();

   }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student4" \t "_blank)**

Output:

111 Karan

222 Aryan

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

//Java program to overload constructors in java

**class** Student5{

**int** id;

    String name;

**int** age;

    //creating two arg constructor

    Student5(**int** i,String n){

    id = i;

    name = n;

    }

    //creating three arg constructor

    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();

   }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student5" \t "_blank)**

Output:

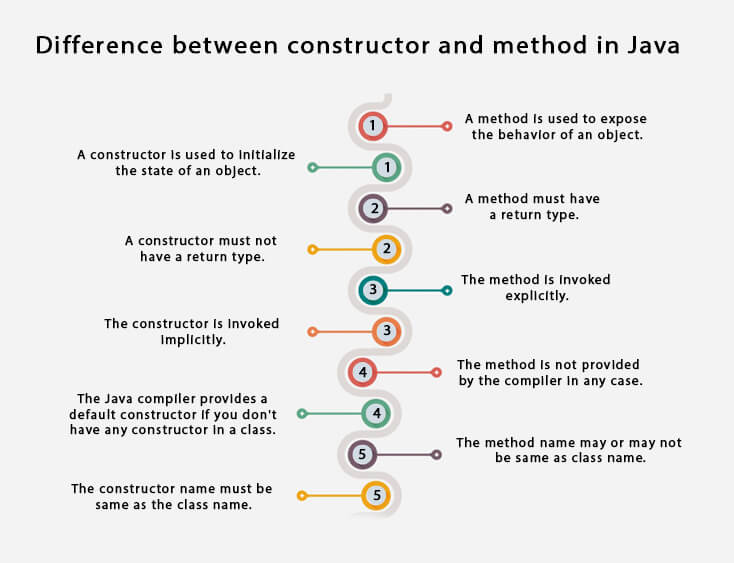
111 Karan 0

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** |
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as class name. |



Java Copy Constructor

There is no copy constructor in java. However, we can copy the values from 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.

//Java program to initialize the values from one object to another

**class** Student6{

**int** id;

    String name;

    //constructor to initialize integer and string

    Student6(**int** i,String n){

    id = i;

    name = n;

    }

  //constructor to initialize another object

    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();

   }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student6" \t "_blank)**

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();

   }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student7" \t "_blank)**

Output:

111 Karan

111 Karan

Q) Does constructor return any value?

Yes, it is the 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 a method, etc. You can perform any operation in the constructor as you perform in the method.

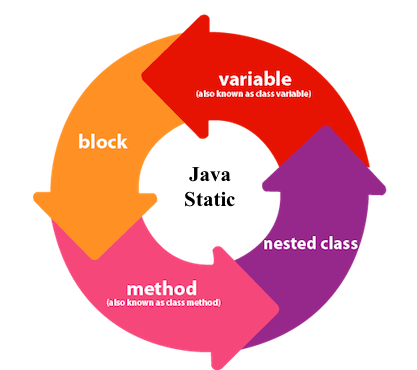
Java static keyword

1. [Static variable](https://www.javatpoint.com/static-keyword-in-java#staticv)
2. [Program of the counter without static variable](https://www.javatpoint.com/static-keyword-in-java#staticvcounter1)
3. [Program of the counter with static variable](https://www.javatpoint.com/static-keyword-in-java#staticvcounter2)
4. [Static method](https://www.javatpoint.com/static-keyword-in-java#staticm)
5. [Restrictions for the static method](https://www.javatpoint.com/static-keyword-in-java#staticmr)
6. [Why is the main method static?](https://www.javatpoint.com/static-keyword-in-java#staticwhymain)
7. [Static block](https://www.javatpoint.com/static-keyword-in-java#staticblock)
8. [Can we execute a program without main method?](https://www.javatpoint.com/static-keyword-in-java#staticwithoutmain)

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 an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class



1) Java static variable

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

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

Advantages of static variable

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

Understanding the 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 the object is created. All students have its unique rollno and name, so instance data member is good in such case. Here, "college" refers to the common property of all objects. If we make it static, this field will get the memory only once.

Java static property is shared to all objects.

Example of static variable

//Java Program to demonstrate the use of static variable

**class** Student{

**int** rollno;//instance variable

   String name;

**static** String college ="ITS";//static variable

   //constructor

   Student(**int** r, String n){

   rollno = r;

   name = n;

   }

   //method to display the values

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

}

//Test class to show the values of objects

**public** **class** TestStaticVariable1{

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

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

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

 //we can change the college of all objects by the single line of code

 //Student.college="BBDIT";

 s1.display();

 s2.display();

 }

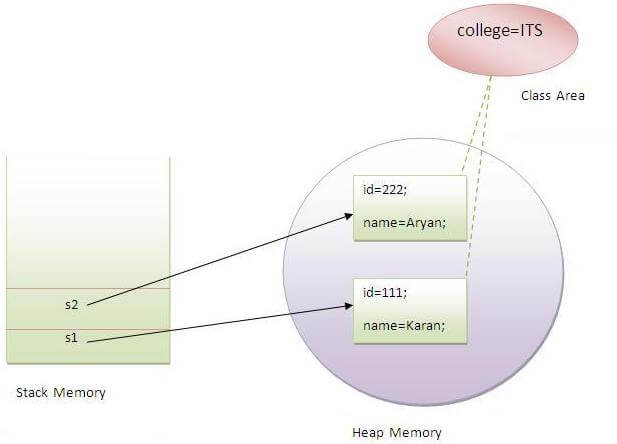
}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestStaticVariable1)

Output:

111 Karan ITS

222 Aryan ITS



Program of the 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 other objects. So each object will have the value 1 in the count variable.

//Java Program to demonstrate the use of an instance variable

//which get memory each time when we create an object of the class.

**class** Counter{

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

Counter(){

count++;//incrementing value

System.out.println(count);

}

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

//Creating objects

Counter c1=**new** Counter();

Counter c2=**new** Counter();

Counter c3=**new** Counter();

}

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Counter" \t "_blank)**

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.

//Java Program to illustrate the use of static variable which

//is shared with all objects.

**class** Counter2{

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

Counter2(){

count++;//incrementing the value of static variable

System.out.println(count);

}

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

//creating objects

Counter2 c1=**new** Counter2();

Counter2 c2=**new** Counter2();

Counter2 c3=**new** Counter2();

}

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Counter2" \t "_blank)**

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

Example of static method

//Java Program to demonstrate the use of a static method.

**class** Student{

**int** rollno;

 String name;

**static** String college = "ITS";

     //static method to change the value of static variable

**static** **void** change(){

     college = "BBDIT";

     }

     //constructor to initialize the variable

     Student(**int** r, String n){

     rollno = r;

     name = n;

     }

     //method to display values

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

}

//Test class to create and display the values of object

**public** **class** TestStaticMethod{

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

    Student.change();//calling change method

    //creating objects

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

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

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

    //calling display method

    s1.display();

    s2.display();

    s3.display();

    }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestStaticMethod" \t "_blank)**

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

Another example of a static method that performs a normal calculation

//Java Program to get the cube of a given number using the 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);

  }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Calculate" \t "_blank)**

Output:125

Restrictions for the 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);

 }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A" \t "_blank)**

Output:Compile Time Error

Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an 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 the 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");

  }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A2" \t "_blank)**

Output:static block is invoked

Hello main

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

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a java class without the main method.

**class** A3{

**static**{

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

  System.exit(0);

  }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A3" \t "_blank)**

Output:

static block is invoked

Since JDK 1.7 and above, output would be:

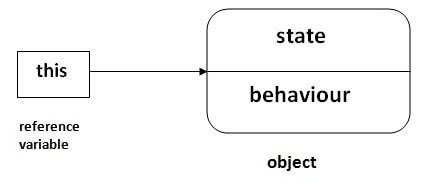
Error: Main method not found in class A3, please define the main method as:

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

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.

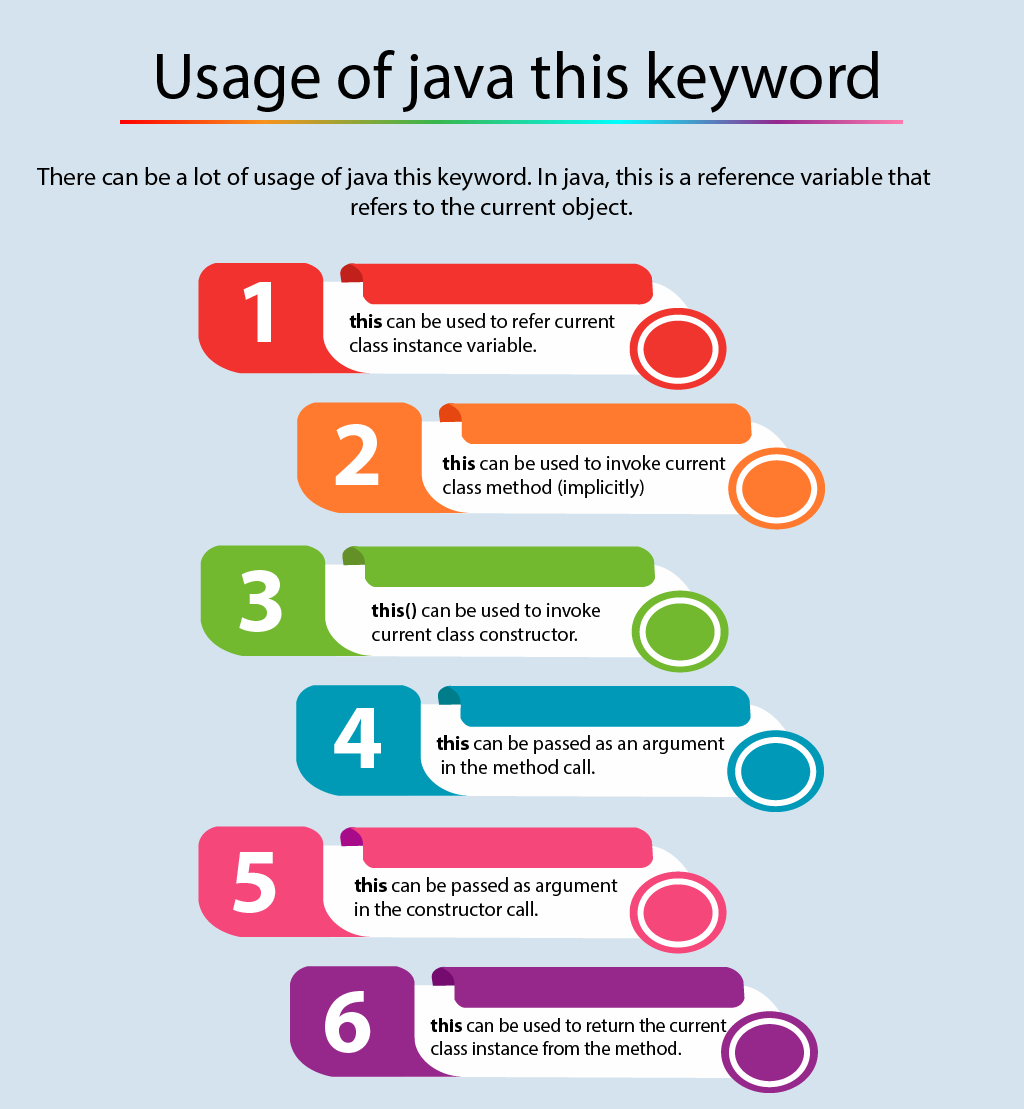


Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
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 can be used to return the current class instance from the method.

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



1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, 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: |

1. **class** Student{
2. **int** rollno;
3. String name;

**float** fee;

Student(**int** rollno,String name,**float** fee){

rollno=rollno;

name=name;

fee=fee;

}

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

}

**class** TestThis1{

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

Student s1=**new** Student(111,"ankit",5000f);

Student s2=**new** Student(112,"sumit",6000f);

s1.display();

s2.display();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis1" \t "_blank)**

Output:

0 null 0.0

0 null 0.0

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

**class** Student{

**int** rollno;

String name;

**float** fee;

Student(**int** rollno,String name,**float** fee){

**this**.rollno=rollno;

**this**.name=name;

**this**.fee=fee;

}

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

}

**class** TestThis2{

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

Student s1=**new** Student(111,"ankit",5000f);

Student s2=**new** Student(112,"sumit",6000f);

s1.display();

s2.display();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis2" \t "_blank)**

Output:

111 ankit 5000

112 sumit 6000

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** Student{

**int** rollno;

String name;

**float** fee;

Student(**int** r,String n,**float** f){

rollno=r;

name=n;

fee=f;

}

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

}

**class** TestThis3{

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

Student s1=**new** Student(111,"ankit",5000f);

Student s2=**new** Student(112,"sumit",6000f);

s1.display();

s2.display();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis3" \t "_blank)**

Output:

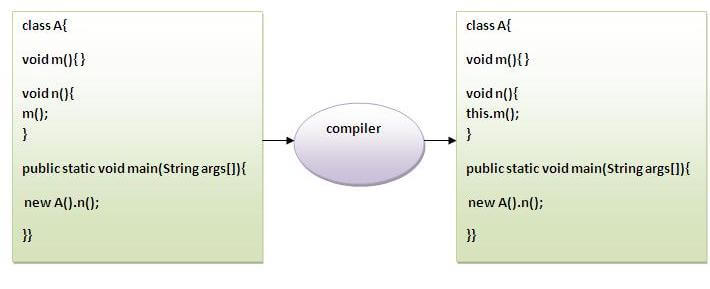
111 ankit 5000

112 sumit 6000

It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.

2) this: to invoke current class method

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** A{

**void** m(){System.out.println("hello m");}

**void** n(){

System.out.println("hello n");

//m();//same as this.m()

**this**.m();

}

}

**class** TestThis4{

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

A a=**new** A();

a.n();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis4" \t "_blank)**

Output:

hello n

hello m

3) this() : to invoke current class constructor

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

**Calling default constructor from parameterized constructor:**

**class** A{

A(){System.out.println("hello a");}

A(**int** x){

**this**();

System.out.println(x);

}

}

**class** TestThis5{

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

A a=**new** A(10);

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis5" \t "_blank)**

Output:

hello a

10

**Calling parameterized constructor from default constructor:**

**class** A{

A(){

**this**(5);

System.out.println("hello a");

}

A(**int** x){

System.out.println(x);

}

}

**class** TestThis6{

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

A a=**new** A();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis6" \t "_blank)**

Output:

5

hello a

Real usage of this() constructor call

The this() constructor call should be used to reuse the constructor from 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** Student{

**int** rollno;

String name,course;

**float** fee;

Student(**int** rollno,String name,String course){

**this**.rollno=rollno;

**this**.name=name;

**this**.course=course;

}

Student(**int** rollno,String name,String course,**float** fee){

**this**(rollno,name,course);//reusing constructor

**this**.fee=fee;

}

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

}

**class** TestThis7{

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

Student s1=**new** Student(111,"ankit","java");

Student s2=**new** Student(112,"sumit","java",6000f);

s1.display();

s2.display();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis7" \t "_blank)**

Output:

111 ankit java null

112 sumit java 6000

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

**class** Student{

**int** rollno;

String name,course;

**float** fee;

Student(**int** rollno,String name,String course){

**this**.rollno=rollno;

**this**.name=name;

**this**.course=course;

}

Student(**int** rollno,String name,String course,**float** fee){

**this**.fee=fee;

**this**(rollno,name,course);//C.T.Error

}

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

}

**class** TestThis8{

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

Student s1=**new** Student(111,"ankit","java");

Student s2=**new** Student(112,"sumit","java",6000f);

s1.display();

s2.display();

}}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestThis8" \t "_blank)**

Compile Time Error: Call to this must be first statement in constructor

4) this: to pass 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();

  }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=S2" \t "_blank)**

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. It is used to reuse one object in many methods.

5) this: to pass 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();

  }

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A4" \t "_blank)**

Output:10

6) this keyword can be used to return current class instance

We can return 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();

}

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Test1" \t "_blank)**

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();

}

}

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=A5" \t "_blank)**

Output:

A5@22b3ea59

A5@22b3ea59