Objects and Classes in Java

An object in Java is the physical as well as a 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:**

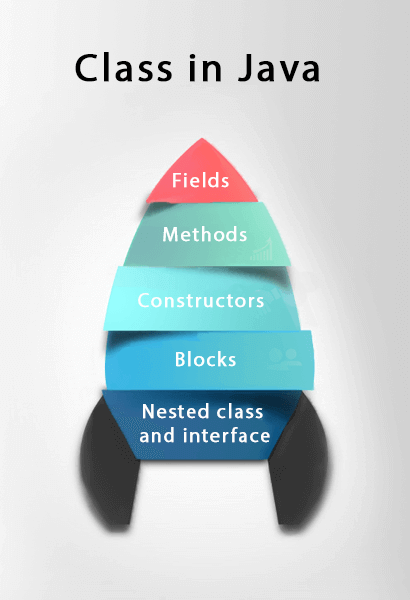
* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

## **What is a class in Java**

A class is a type which leads to have 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.

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. }

### **Object and Class Example: main outside the concerned 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. }

## **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="Sayan";
10. System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12. }

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

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. }

### **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.

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. }



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.

### **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. }

### **Object and Class Example: Rectangle**

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

*File: TestRectangle1.java*

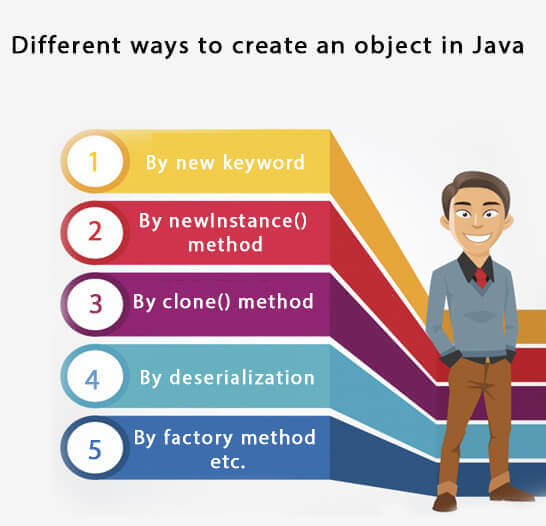
1. **class** Rectangle{
2. **int** length;
3. **int** width;
4. **void** insert(**int** l, **int** w){
5. length=l;
6. width=w;
7. }
8. **void** calculateArea(){System.out.println(length\*width);}
9. }
10. **class** TestRectangle1{
11. **public** **static** **void** main(String args[]){
12. Rectangle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

## **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:

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

Calling method through a reference:

1. Calculation c=**new** Calculation();
2. c.fact(5);

Calling method through an anonymous object

1. **new** Calculation().fact(5);

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

1. **class** Calculation{
2. **void** fact(**int**  n){
3. **int** fact=1;
4. **for**(**int** i=1;i<=n;i++){
5. fact=fact\*i;
6. }
7. System.out.println("factorial is "+fact);
8. }
9. **public** **static** **void** main(String args[]){
10. **new** Calculation().fact(5);//calling method with anonymous object
11. }
12. }

### **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:

1. **int** a=10, b=20;

Initialization of refernce variables:

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

Let's see the example:

1. //Java Program to illustrate the use of Rectangle class which
2. //has length and width data members
3. **class** Rectangle{
4. **int** length;
5. **int** width;
6. **void** insert(**int** l,**int** w){
7. length=l;
8. width=w;
9. }
10. **void** calculateArea(){System.out.println(length\*width);}
11. }
12. **class** TestRectangle2{
13. **public** **static** **void** main(String args[]){
14. Rectangle r1=**new** Rectangle(),r2=**new** Rectangle();//creating two objects
15. r1.insert(11,5);
16. r2.insert(3,15);
17. r1.calculateArea();
18. r2.calculateArea();
19. }
20. }

### **Real World Example: Account**

*File: TestAccount.java*

1. //Java Program to demonstrate the working of a banking-system
2. //where we deposit and withdraw amount from our account.
3. //Creating an Account class which has deposit() and withdraw() methods
4. **class** Account{
5. **int** acc\_no;
6. String name;
7. **float** amount;
8. //Method to initialize object
9. **void** insert(**int** a,String n,**float** amt){
10. acc\_no=a;
11. name=n;
12. amount=amt;
13. }
14. //deposit method
15. **void** deposit(**float** amt){
16. amount=amount+amt;
17. System.out.println(amt+" deposited");
18. }
19. //withdraw method
20. **void** withdraw(**float** amt){
21. **if**(amount<amt){
22. System.out.println("Insufficient Balance");
23. }**else**{
24. amount=amount-amt;
25. System.out.println(amt+" withdrawn");
26. }
27. }
28. //method to check the balance of the account
29. **void** checkBalance(){System.out.println("Balance is: "+amount);}
30. //method to display the values of an object
31. **void** display(){System.out.println(acc\_no+" "+name+" "+amount);}
32. }
33. //Creating a test class to deposit and withdraw amount
34. **class** TestAccount{
35. **public** **static** **void** main(String[] args){
36. Account a1=**new** Account();
37. a1.insert(832345,"Ankit",1000);
38. a1.display();
39. a1.checkBalance();
40. a1.deposit(40000);
41. a1.checkBalance();
42. a1.withdraw(15000);
43. a1.checkBalance();
44. }}

**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.

## **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:

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

Calling method through a reference:

1. Calculation c=**new** Calculation();
2. c.fact(5);

Calling method through an anonymous object

1. **new** Calculation().fact(5);

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

1. **class** Calculation{
2. **void** fact(**int**  n){
3. **int** fact=1;
4. **for**(**int** i=1;i<=n;i++){
5. fact=fact\*i;
6. }
7. System.out.println("factorial is "+fact);
8. }
9. **public** **static** **void** main(String args[]){
10. **new** Calculation().fact(5);//calling method with anonymous object
11. }
12. }

### **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:

1. **int** a=10, b=20;

Initialization of refernce variables:

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

Let's see the example:

1. //Java Program to illustrate the use of Rectangle class which
2. //has length and width data members
3. **class** Rectangle{
4. **int** length;
5. **int** width;
6. **void** insert(**int** l,**int** w){
7. length=l;
8. width=w;
9. }
10. **void** calculateArea(){System.out.println(length\*width);}
11. }
12. **class** TestRectangle2{
13. **public** **static** **void** main(String args[]){
14. Rectangle r1=**new** Rectangle(),r2=**new** Rectangle();//creating two objects
15. r1.insert(11,5);
16. r2.insert(3,15);
17. r1.calculateArea();
18. r2.calculateArea();
19. }
20. }

### **Real World Example: Account**

*File: TestAccount.java*

1. //Java Program to demonstrate the working of a banking-system
2. //where we deposit and withdraw amount from our account.
3. //Creating an Account class which has deposit() and withdraw() methods
4. **class** Account{
5. **int** acc\_no;
6. String name;
7. **float** amount;
8. //Method to initialize object
9. **void** insert(**int** a,String n,**float** amt){
10. acc\_no=a;
11. name=n;
12. amount=amt;
13. }
14. //deposit method
15. **void** deposit(**float** amt){
16. amount=amount+amt;
17. System.out.println(amt+" deposited");
18. }
19. //withdraw method
20. **void** withdraw(**float** amt){
21. **if**(amount<amt){
22. System.out.println("Insufficient Balance");
23. }**else**{
24. amount=amount-amt;
25. System.out.println(amt+" withdrawn");
26. }
27. }
28. //method to check the balance of the account
29. **void** checkBalance(){System.out.println("Balance is: "+amount);}
30. //method to display the values of an object
31. **void** display(){System.out.println(acc\_no+" "+name+" "+amount);}
32. }
33. //Creating a test class to deposit and withdraw amount
34. **class** TestAccount{
35. **public** **static** **void** main(String[] args){
36. Account a1=**new** Account();
37. a1.insert(832345,"Ankit",1000);
38. a1.display();
39. a1.checkBalance();
40. a1.deposit(40000);
41. a1.checkBalance();
42. a1.withdraw(15000);
43. a1.checkBalance();
44. }}

## **Java Constructors**

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes

In [Java](https://www.javatpoint.com/java-tutorial), a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java) is created. At the time of calling constructor, memory for the object is allocated in the memory.

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

**Every time an object is created using the new() keyword, at least one constructor is called.**

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized 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 fp=or 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

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:**

1. <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. |

1. //Java Program to create and call a default constructor
2. **class** Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. **public** **static** **void** main(String args[]){
7. //calling a default constructor
8. Bike1 b=**new** Bike1();
9. }
10. }

### 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

1. **class** Student3{
2. **int** id;
3. String name;
4. //method to display the value of id and name
5. **void** display(){System.out.println(id+" "+name);}
7. **public** **static** **void** main(String args[]){
8. //creating objects
9. Student3 s1=**new** Student3();
10. Student3 s2=**new** Student3();
11. //displaying values of the object
12. s1.display();
13. s2.display();
14. }
15. }

### **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 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.

1. //Java Program to demonstrate the use of the parameterized constructor.
2. **class** Student4{
3. **int** id;
4. String name;
5. //creating a parameterized constructor
6. Student4(**int** i,String n){
7. id = i;
8. name = n;
9. }
10. //method to display the values
11. **void** display(){System.out.println(id+" "+name);}
13. **public** **static** **void** main(String args[]){
14. //creating objects and passing values
15. Student4 s1 = **new** Student4(111,"Karan");
16. Student4 s2 = **new** Student4(222,"Aryan");
17. //calling method to display the values of object
18. s1.display();
19. s2.display();
20. }
21. }

# Method Overloading in Java

If a class has multiple methods having same name but different in 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 behavior of the method because its name differs.

## **Advantage 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, Method Overloading is not possible by changing the return type of the method only.**

### **1) Method Overloading: changing no. of arguments**

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating [static methods](https://www.javatpoint.com/static-keyword-in-java)

so that we don't need to create instance for calling methods.

1. **class** Adder{
2. **static** **int** add(**int** a,**int** b){**return** a+b;}
3. **static** **int** add(**int** a,**int** b,**int** c){**return** a+b+c;}
4. }
5. **class** TestOverloading1{
6. **public** **static** **void** main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(11,11,11));
9. }}

### **2) Method Overloading: changing data type of arguments**

In this example, we have created two methods that differs in [data type](https://www.javatpoint.com/java-data-types)

. The first add method receives two integer arguments and second add method receives two double arguments.

1. **class** Adder{
2. **static** **int** add(**int** a, **int** b){**return** a+b;}
3. **static** **double** add(**double** a, **double** b){**return** a+b;}
4. }
5. **class** TestOverloading2{
6. **public** **static** **void** main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(12.3,12.6));
9. }}

### **Q) Why Method Overloading is not possible by changing the return type of method only?**

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

1. **class** Adder{
2. **static** **int** add(**int** a,**int** b){**return** a+b;}
3. **static** **double** add(**int** a,**int** b){**return** a+b;}
4. }
5. **class** TestOverloading3{
6. **public** **static** **void** main(String[] args){
7. System.out.println(Adder.add(11,11));//ambiguity
8. }}

Compile Time Error: method add(int,int) is already defined in class Adder

#### **Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.**

### **Can we overload java main() method?**

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But [JVM](https://www.javatpoint.com/jvm-java-virtual-machine)

calls main() method which receives string array as arguments only. Let's see the simple example:

1. **class** TestOverloading4{
2. **public** **static** **void** main(String[] args){System.out.println("main with String[]");}
3. **public** **static** **void** main(String args){System.out.println("main with String");}
4. **public** **static** **void** main(){System.out.println("main without args");}
5. }

## **Method Overloading and Type Promotion**

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



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**

1. **class** OverloadingCalculation1{
2. **void** sum(**int** a,**long** b){System.out.println(a+b);}
3. **void** sum(**int** a,**int** b,**int** c){System.out.println(a+b+c);}
5. **public** **static** **void** main(String args[]){
6. OverloadingCalculation1 obj=**new** OverloadingCalculation1();
7. obj.sum(20,20);//now second int literal will be promoted to long
8. obj.sum(20,20,20);
10. }
11. }

### **Example of Method Overloading with Type Promotion if matching found**

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

1. **class** OverloadingCalculation2{
2. **void** sum(**int** a,**int** b){System.out.println("int arg method invoked");}
3. **void** sum(**long** a,**long** b){System.out.println("long arg method invoked");}
5. **public** **static** **void** main(String args[]){
6. OverloadingCalculation2 obj=**new** OverloadingCalculation2();
7. obj.sum(20,20);//now int arg sum() method gets invoked
8. }
9. }

### **Example of Method Overloading with Type Promotion in case of ambiguity**

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

1. **class** OverloadingCalculation3{
2. **void** sum(**int** a,**long** b){System.out.println("a method invoked");}
3. **void** sum(**long** a,**int** b){System.out.println("b method invoked");}
5. **public** **static** **void** main(String args[]){
6. OverloadingCalculation3 obj=**new** OverloadingCalculation3();
7. obj.sum(20,20);//now ambiguity
8. }
9. }

Output:Compile Time Error

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