

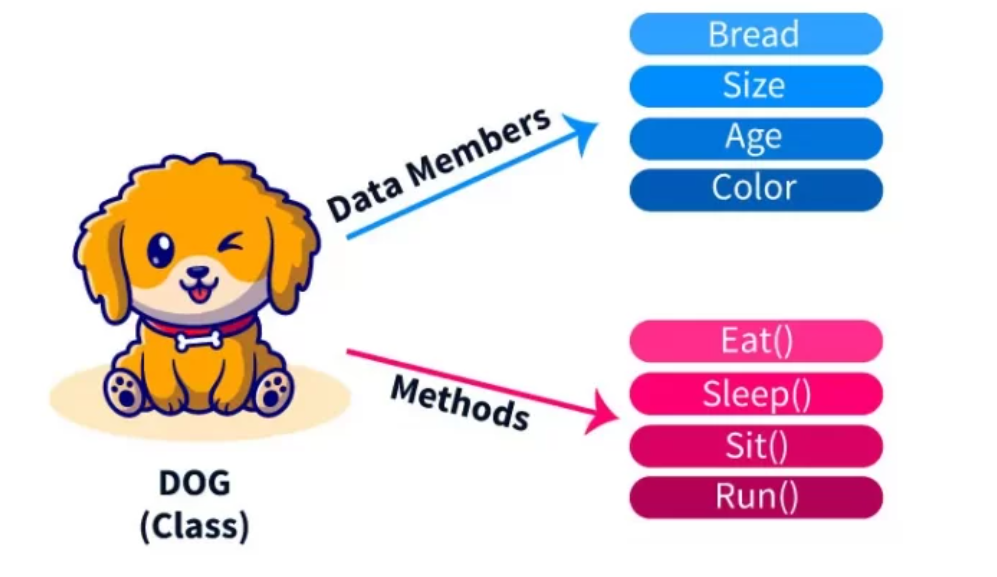
## Class

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.

(Or)

Class is collection of Data members(variables) and member functions (methods).



**The basic syntax** used for creating a class is as follows:

class <class\_name>

{

data members; //hold the data or information

member functions; //determine the behaviour

}

## Object



Any entity that has state and behaviour 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.

**Object and Class Example:**

*File: TestStudent1.java*

//Java Program to demonstrate having the main method in

//another class

//Creating Student class.

**class** Student{

**int** id;

 String name;

}

//Creating another class TestStudent1 which contains the main method

**class** TestStudent1{

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

  Student s1=**new** Student();

  System.out.println(s1.id);

  System.out.println(s1.name);

 }

}

**Test it Now**

Output:

0

null

# Method in Java

In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods**. In this section, we will learn **what is a method in Java, types of methods, method declaration,** and **how to call a method in Java**.

## What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

### **Method Declaration**

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as we have shown in the following figure.361

Difference between JDK, JRE, and JVM



**Method Signature:** Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

**Access Specifier:** Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

* **Public:** The method is accessible by all classes when we use public specifier in our application.
* **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
* **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
* **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

**Return Type:** Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

**Method Name:** It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

**Parameter List:** It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

**Method Body:** It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

## Naming a Method

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

**Single-word method name:** sum(), area()

**Multi-word method name:** areaOfCircle(), stringComparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

## Types of Method

There are two types of methods in Java:

* Predefined Method
* User-defined Method

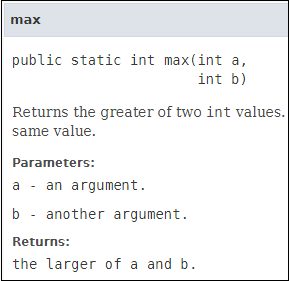
### **Predefined Method**

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Each and every predefined method is defined inside a class. Such as **print()** method is defined in the **java.io.PrintStream** class. It prints the statement that we write inside the method. For example, **print("Java")**, it prints Java on the console.

In the above example, we have used three predefined methods **main(), print(),** and **max()**. We have used these methods directly without declaration because they are predefined. The print() method is a method of **PrintStream** class that prints the result on the console. The max() method is a method of the **Math** class that returns the greater of two numbers.

We can also see the method signature of any predefined method by using the link <https://docs.oracle.com/>. When we go through the link and see the max() method signature, we find the following:



In the above method signature, we see that the method signature has access specifier **public**, non-access modifier **static**, return type **int**, method name **max(),** parameter list **(int a, int b).** In the above example, instead of defining the method, we have just invoked the method. This is the advantage of a predefined method. It makes programming less complicated.

Similarly, we can also see the method signature of the print() method.

### **User-defined Method**

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

### **Static Method**

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

### **Instance Method**

The method of the class is known as an **instance method**. It is a **non-static** method defined in the class. Before calling or invoking the instance method, it is necessary to create an object of its class. Let's see an example of an instance method.

There are two types of instance method:

* **Accessor Method**
* **Mutator Method**

**Accessor Method:** The method(s) that reads the instance variable(s) is known as the accessor method. We can easily identify it because the method is prefixed with the word **get**. It is also known as **getters**. It returns the value of the private field. It is used to get the value of the private field.

**Mutator Method:** The method(s) read the instance variable(s) and also modify the values. We can easily identify it because the method is prefixed with the word **set**. It is also known as **setters** or **modifiers**. It does not return anything. It accepts a parameter of the same data type that depends on the field. It is used to set the value of the private field.

### **Abstract Method**

The method that does not has method body is known as abstract method. In other words, without an implementation is known as abstract method. It always declares in the **abstract class**. It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword **abstract**.

**Syntax**

1. **abstract** **void** method\_name();

### **Factory method**

It is a method that returns an object to the class to which it belongs. All static methods are factory methods. For example, **NumberFormat obj = NumberFormat.getNumberInstance()**

# **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](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 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](https://www.javatpoint.com/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:**

1. <class\_name>(){}

#### 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.

### **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.

## 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](https://www.javatpoint.com/method-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.

## 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 the class name. |



### **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.

### **Is there Constructor class in Java?**

Yes.

### **What is the purpose of Constructor class?**

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

## Abstraction

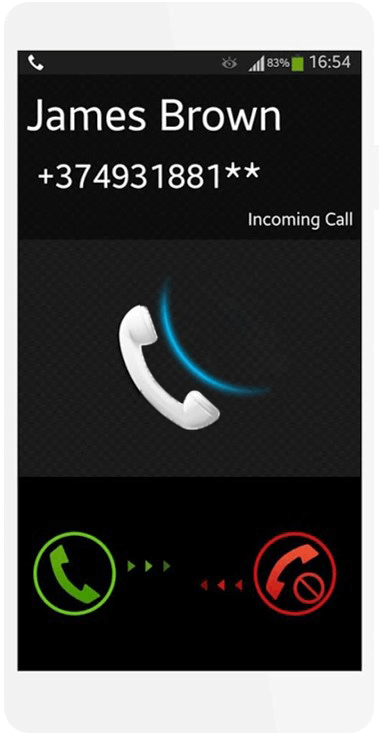
# **Abstract class in Java**

A class which is declared with the abstract keyword is known as an abstract class in [Java](https://www.javatpoint.com/java-tutorial). It can have abstract and non-abstract methods (method with the body).

Before learning the Java abstract class, let's understand the abstraction in Java first.

### **Abstraction in Java**

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.



Abstraction refers to the quality of dealing with ideas rather than events. It basically deals with hiding the details and showing the essential things to the user. If you look at the image here, whenever we get a call, we get an option to either pick it up or just reject it. But in reality, there is a lot of code that runs in the background. So you don’t know the internal processing of how a call is generated, that’s the beauty of abstraction. Therefore, abstraction helps to reduce complexity.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.MapReduce and Design Patterns - Binning Pattern Example

Abstraction lets you focus on what the [object](https://www.javatpoint.com/object-and-class-in-java) does instead of how it does it.

### **Ways to achieve Abstraction**

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

### **Abstract class in Java**

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

#### Points to Remember

* An abstract class must be declared with an abstract keyword.
* It can have abstract and non-abstract methods.
* It cannot be instantiated.
* It can have [constructors](https://www.javatpoint.com/java-constructor) and static methods also.
* It can have final methods which will force the subclass not to change the body of the method.



**Example of abstract class**

1. **abstract** **class** A{}

### **Abstract Method in Java**

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

1. **abstract** **void** printStatus();//no method body and abstract

### **Understanding the real scenario of Abstract class**

In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.

Mostly, we don't know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the **factory method**.

A **factory method** is a method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

#### Rule: If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract

# **Interface in Java**

1. [Interface](https://www.javatpoint.com/interface-in-java)
2. [Example of Interface](https://www.javatpoint.com/interface-in-java#interfaceex)
3. [Multiple inheritance by Interface](https://www.javatpoint.com/interface-in-java#interfacemultiple)
4. [Why multiple inheritance is supported in Interface while it is not supported in case of class.](https://www.javatpoint.com/interface-in-java#interfacewhynot)
5. [Marker Interface](https://www.javatpoint.com/interface-in-java#interfacemarker)
6. [Nested Interface](https://www.javatpoint.com/nested-interface)

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve [*abstraction*](https://www.javatpoint.com/abstract-class-in-java). There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java).

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also **represents the IS-A relationship**.

45.5M

It cannot be instantiated just like the abstract class.

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

## Why use Java interface?

There are mainly three reasons to use interface. They are given below.

* It is used to achieve abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.



## How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

### **Syntax:**

1. **interface** <interface\_name>{
3. // declare constant fields
4. // declare methods that abstract
5. // by default.
6. }

## Java 8 Interface Improvement

Since [Java 8](https://www.javatpoint.com/java-8-features), interface can have default and static methods which is discussed later.

## Internal addition by the compiler

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

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



#### The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



## Java Interface Example: Drawable

In this example, the Drawable interface has only one method. Its implementation is provided by Rectangle and Circle classes. In a real scenario, an interface is defined by someone else, but its implementation is provided by different implementation providers. Moreover, it is used by someone else. The implementation part is hidden by the user who uses the interface.

## Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



As you can see in the above example, Printable and Showable interface have same methods but its implementation is provided by class TestTnterface1, so there is no ambiguity.

# **Difference between abstract class and interface**

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

Encapsulation

[**Encapsulation**](https://www.edureka.co/blog/object-oriented-programming/#encapsulation) is a process of binding your data and code together as a single unit using getter and setter methods.

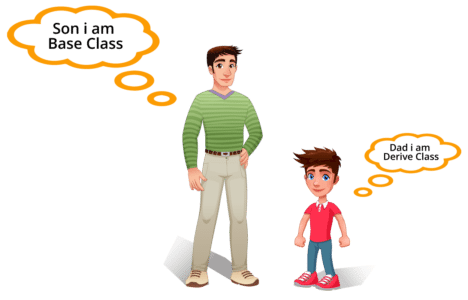
You need to perform two steps to achieve encapsulation:

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

## Inheritance

**Object Oriented Programming: Inheritance**

In OOP, computer programs are designed in such a way where everything is an object that interact with one another. Inheritance is one such concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes.



As we can see in the image, a child inherits the properties from his father. Similarly, in Java, there are two classes:

1. Parent class ( Super or Base class)

2. Child class (Subclass or Derived class )

A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

Inheritance is further classified into 4 types:

## Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



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

When one class inherits multiple classes, it is known as multiple inheritance. For Example:

## Single Inheritance Example

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

## Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

## Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

## Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

## Hybrid Inheritance in Java

It seems that because of this diagram people are finding it difficult to understand this topic because this diagram shows combination of hierarchical and multiple inheritance and [**multiple inheritance**](https://beginnersbook.com/2013/05/java-multiple-inheritance/) is not supported in java.  
The diagram is just for the representation, since multiple inheritance is not possible in java, It is not correct to show that as a part of hybrid inheritance. I will update the diagram whenever I get the time. You can refer the example that I have given at the beginning of post representing combination of single and hierarchical inheritance

**Program:** This example is just to demonstrate the hybrid inheritance in Java. Although this example is meaningless, you would be able to see that how we have implemented two types of inheritance(single and hierarchical) together to form hybrid inheritance.  
Class A and B extends class C → [**Hierarchical inheritance**](https://beginnersbook.com/2013/10/hierarchical-inheritance-java-program/)  
Class D extends class A → Single inheritance

## Polymorphism

**Polymorphism in Java** is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

[**Polymorphism**](https://www.edureka.co/blog/object-oriented-programming/#polymorphism) is the ability of a variable, function or an object to take multiple forms. It allows you to define one interface or method and have multiple implementations. There are two types of polymorphism in Java.

1. Compile Time Polymorphism
2. Runtime Polymorphism

## Compile Time Polymorphism

Also called static binding, as the type of the object is determined at the compile time by the compiler itself. Example: Method Overloading

## Runtime Polymorphism

Also called dynamic binding as the overridden method is resolved at runtime rather than compile-time.  In this, a reference variable is used to call an overridden method of a superclass at run time. Example: Method Overriding.

## **What is Method Overloading in Java?**

Method overloading allows the method to have the same name which differs on the basis of arguments or the argument types. It can be related to compile-time polymorphism. Following are a few pointers that we have to keep in mind while overloading methods in Java.

* We cannot overload a return type.
* Although we can overload [static methods](https://www.edureka.co/blog/static-keyword-in-java/), the arguments or input parameters have to be different.
* We cannot overload two methods if they only differ by a static keyword.
* Like other static methods, the main() method can also be overloaded.

In the above program, we have two methods with the same name but different parameters. This is how the method overloading works in [Java](https://www.edureka.co/blog/java-tutorial/).

[Explore Curriculum](https://www.edureka.co/java-j2ee-training-course" \t "_blank)

## **Why Method Overloading?**

The main advantage of using method overloading in Java is that it gives us the liberty to not define a function again and again for doing the same thing. In the below example, the two methods are basically performing division, so we can have different methods with the same name but with different parameters. It also helps in compile-time polymorphism.

## **What Is Method Overriding in Java?**

Inheritance in java involves a relationship between parent and child classes. Whenever both the classes contain methods with the same name and arguments or parameters it is certain that one of the methods will override the other method during execution. The method that will be executed depends on the object.

If the child class object calls the method, the child class method will override the parent class method. Otherwise, if the parent class object calls the method, the

Method overriding also helps in implementing runtime polymorphism in java. Let’s take a simple example to understand how method overriding works in java.

## **Rules For Method Overriding**

* The [access modifier](https://www.edureka.co/blog/access-modifiers-in-java/) can only allow more access for the overridden method.
* A [final](https://www.edureka.co/blog/final-finally-and-finalize-in-java/) method does not support method overriding.
* A static method cannot be overridden.
* Private methods cannot be overridden.
* The return type of the overriding method must be the same.
* We can call the parent class method in the overriding method using the super keyword.
* A [constructor](https://www.edureka.co/blog/constructor-in-java/) cannot be overridden because a child class and a parent class cannot have the constructor with the same name.

## Difference between Method Overloading and Method Overriding in Java

|  |  |
| --- | --- |
| **Method Overloading** | **Method Overriding** |
| When a class having several methods with the same name but with different arguments then it is known as method overloading. | When parent and child class have methods with same name and arguments then it is called as method overriding. |
| Method overloading occur within one class only. | Method overriding occur within two classes (parent and child). |
| It is an example of compile time polymorphism. | It is an example of run time polymorphism. |
| Methods must have different signature (same name but different number of arguments and their type). | Methods must have same signature (same name and arguments list). |
| Used to add more to method’s behavior. | Used to change the method’s existing behavior. |