**Object-oriented programming:** As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

**OOPs Concepts:**

* Class
* Object
* [Method](https://www.geeksforgeeks.org/methods-in-java/)
* [Polymorphism](https://www.geeksforgeeks.org/polymorphism-in-java/)
* [Inheritance](https://www.geeksforgeeks.org/inheritance-in-java/)
* [Encapsulation](https://www.geeksforgeeks.org/encapsulation-in-java/)
* [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)
* [Message Passing](https://www.geeksforgeeks.org/message-passing-in-java/)

[**Class**](https://www.geeksforgeeks.org/classes-objects-java/)**:** A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

* 1. **Modifiers**: A class can be public or has default access .
  2. **Class name:** The name should begin with a initial letter (capitalized by convention).
  3. **Superclass(if any):** The name of the class’s parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
  4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
  5. **Body:** The class body surrounded by braces, { }.

**Object**

**Object** is an instance of a class. An object in OOPS is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. An object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

* **Declaration** − A variable declaration with a variable name with an object type.
* **Instantiation** − The 'new' keyword is used to create the object.
* **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

[**Method**](https://www.geeksforgeeks.org/methods-in-java/)**:** A method is a collection of statements that perform some specific task and return result to the caller. A method can perform some specific task without returning anything. Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class which is different from languages likeC ,C++ and Python.

Methods are **time savers**and help us to **reuse** the code without retyping the code.

In general, method declarations has six components:

* [**Access Modifier**](https://www.geeksforgeeks.org/access-modifiers-java/): Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
  + **public:** accessible in all class in your application.
  + **protected:** accessible within the package in which it is defined and in its **subclass(es)(including subclasses declared outside the package)**
  + **private:** accessible only within the class in which it is defined.
  + **default (declared/defined without using any modifier):** accessible within same class and package within which its class is defined.
* **The return type**: The data type of the value returned by the method or void if does not return a value.
* **Method Name**: the rules for field names apply to method names as well, but the convention is a little different.
* **Parameter list**: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
* **Exception list**: The exceptions you expect by the method can throw, you can specify these exception(s).
* **Method body**: it is enclosed between braces. The code you need to be executed to perform your intended operations.



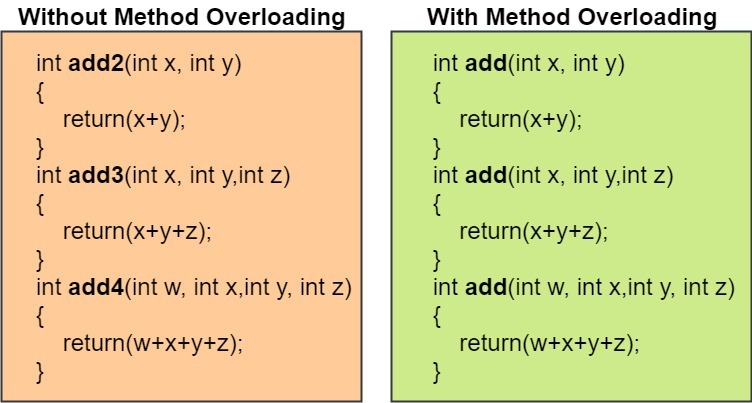
[**Polymorphism**](https://www.geeksforgeeks.org/polymorphism-in-java/)**:** Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. This is done by Java with the help of the signature and declaration of these entities.

Polymorphism in Java are mainly of 2 types:

* [Overloading in Java](https://www.geeksforgeeks.org/overloading-in-java/)
* [Overriding in Java](https://www.geeksforgeeks.org/overriding-in-java/)

**Method Overloading**.

If a [class](https://www.javatpoint.com/object-and-class-in-java) has multiple methods having same name but different in parameters, it is known as **Method Overloading**.



### Usage of Java Method Overloading

It is **used** when a class that extends from another class wants to **use** most of the feature of the parent class and wants to implement specific functionality in certain cases.

**Overloading** in Java is the ability to create multiple methods of the same name, but with different parameters.

Method Overloading Rules:

Two methods will be treated as overloaded if:

* Both must have the same method name.
* Both must have different argument lines.

**And if both methods follow the above mandatory rules, then they may or may not:**

* Have different return types.
* Have different access modifiers.
* Throw different checked or unchecked exceptions.

# Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

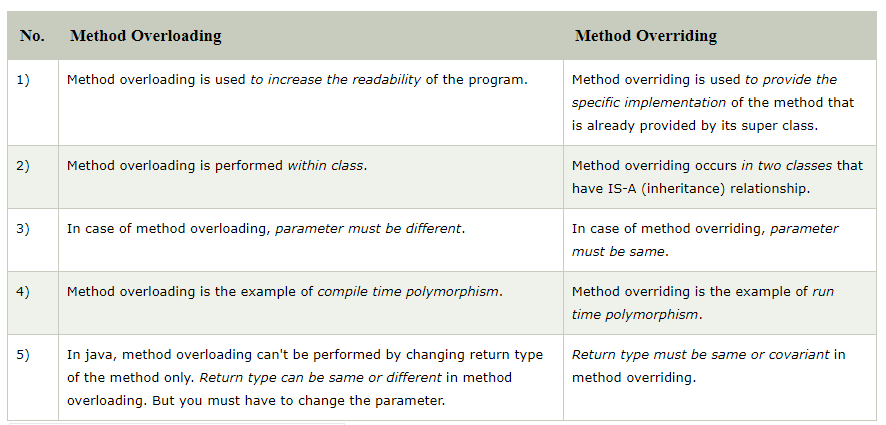
In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

### Usage of Java Method Overriding

* Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
* Method overriding is used for runtime polymorphism

#### Rules for Java Method Overriding

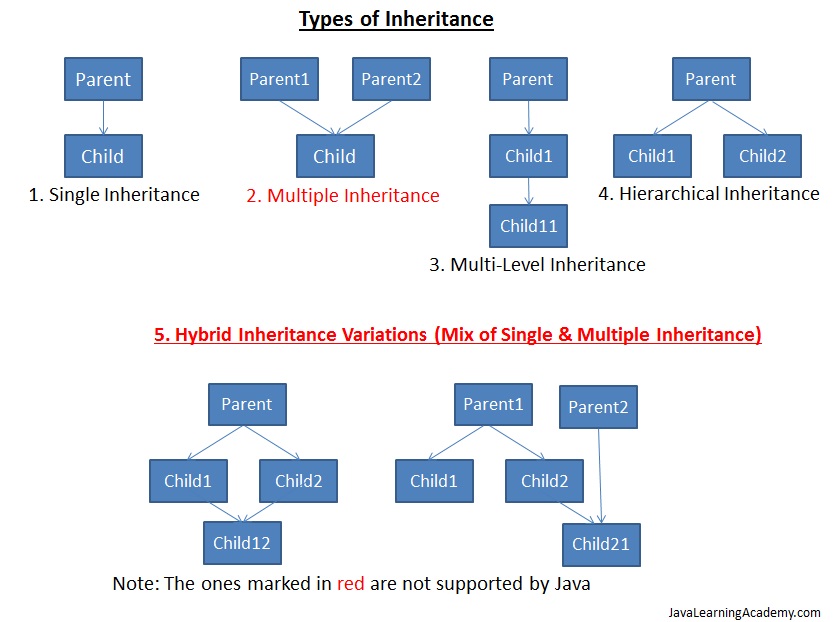
1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).



[**Inheritance**](https://www.geeksforgeeks.org/inheritance-in-java/)**:** Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields(data) and methods) of another class.  
**Important terminology:**

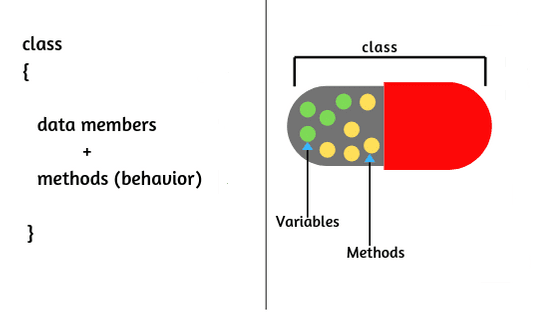
* **Super Class:**The class whose features are inherited is known as superclass(or a base class or a parent class).
* **Sub Class:** The class that inherits the other class is known as subclass(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability:**Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

The keyword used for inheritance is **extends**.



[**Encapsulation**](https://www.geeksforgeeks.org/encapsulation-in-java/)**:**

**Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.



We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.

* Technically in encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared.
* As in encapsulation, the data in a class is hidden from other classes, so it is also known as **data-hiding**.
* Encapsulation can be achieved by Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables.

### Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

### Abstraction in Java

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

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.

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
2. Interface

### 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.

**An interface in Java**

An interface in Java is defined as an abstract type that specifies class behavior**.** An interface is a kind of a protocol that sets up rules regarding how a particular class should behave.

An interface in Java can contain abstract methods and static constants. By default, all the methods in the interface are public and abstract.

|  |  |
| --- | --- |
| **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(); } |

**Message Passing:**

**Message Passing:** Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function and the information to be sent.

