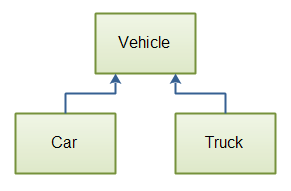
**Inheritance**

Inheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes.

A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.

 in Java, a subclass inherits the characteristics (properties and methods) of its superclass (ancestor).

For example, a **vehicle is a superclass** and a **car and Truck are subclasses.**



The car , truck (subclasses) inherits all of the vehicle’s properties.

The inheritance mechanism is very useful in code reuse.

**The Limitations of Java class inheritance**:

1. A subclass cannot inherit private members of its superclass.
2. Constructor and initializer blocks cannot be inherited by a subclass.
3. A subclass can have only one superclass.

The keyword “extends” is used to derive a subclass from the superclass, as illustrated by the following syntax: class Name\_of\_subclass extends Name\_of superclass { //new fields and methods that would define the subclass go here } If you want to derive a subclass Rectangle from a superclass Shapes, you can do it as follows: class Rectangle extends Shapes { …. }



**Terms used in Inheritance**

Class: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub Class/Child Class: Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

Super Class/Parent Class: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability: As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

**The syntax of Java Inheritance**

class Subclass-name extends Superclass-name

{

//methods and fields

}

The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

### Access Control and Inheritance

The following rules for inherited methods are enforced −

* Methods declared public in a superclass also must be public in all subclasses.
* Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
* Methods declared private are not inherited at all, so there is no rule for them.

super keyword

The **super** keyword is similar to **this** keyword. Following are the scenarios where the super keyword is used.

* It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
* It is used to **invoke the superclass** constructor from subclass.

Differentiating the Members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

super.variable

super.method();

Sample Code

This section provides you a program that demonstrates the usage of the **super** keyword.

In the given program, you have two classes namely *Sub\_class* and *Super\_class*, both have a method named display() with different implementations, and a variable named num with different values. We are invoking display() method of both classes and printing the value of the variable num of both classes. Here you can observe that we have used super keyword to differentiate the members of superclass from subclass.

Copy and paste the program in a file with name Sub\_class.java.

**Example**

[Live Demo](http://tpcg.io/C04irr)

class Super\_class {

int num = 20;

// display method of superclass

public void display() {

System.out.println("This is the display method of superclass");

}

}

public class Sub\_class extends Super\_class {

int num = 10;

// display method of sub class

public void display() {

System.out.println("This is the display method of subclass");

}

public void my\_method() {

// Instantiating subclass

Sub\_class sub = new Sub\_class();

// Invoking the display() method of sub class

sub.display();

// Invoking the display() method of superclass

super.display();

// printing the value of variable num of subclass

System.out.println("value of the variable named num in sub class:"+ sub.num);

// printing the value of variable num of superclass

System.out.println("value of the variable named num in super class:"+ super.num);

}

public static void main(String args[]) {

Sub\_class obj = new Sub\_class();

obj.my\_method();

}

}

Compile and execute the above code using the following syntax.

javac Super\_Demo

java Super

On executing the program, you will get the following result −

**Output**

This is the display method of subclass

This is the display method of superclass

value of the variable named num in sub class:10

value of the variable named num in super class:20

Invoking Superclass Constructor

If a class is inheriting the properties of another class, the subclass automatically acquires the default constructor of the superclass. But if you want to call a parameterized constructor of the superclass, you need to use the super keyword as shown below.

super(values);

Sample Code

The program given in this section demonstrates how to use the super keyword to invoke the parametrized constructor of the superclass. This program contains a superclass and a subclass, where the superclass contains a parameterized constructor which accepts a integer value, and we used the super keyword to invoke the parameterized constructor of the superclass.

Copy and paste the following program in a file with the name Subclass.java

**Example**

[Live Demo](http://tpcg.io/iTN0iC)

class Superclass {

int age;

Superclass(int age) {

this.age = age;

}

public void getAge() {

System.out.println("The value of the variable named age in super class is: " +age);

}

}

public class Subclass extends Superclass {

Subclass(int age) {

super(age);

}

public static void main(String argd[]) {

Subclass s = new Subclass(24);

s.getAge();

}

}

Compile and execute the above code using the following syntax.

javac Subclass

java Subclass

On executing the program, you will get the following result −

**Output**

The value of the variable named age in super class is: 24

**PREVENTING INHERITANCE**

1. Using final keyword
2. By making a class constructor private:

##### **1: Using final keyword**

Using final keyword before a class declaration we can stop a class to be inherited by other classes. For example,

**public final class A**

**{**

**}**

If we try to extend the class A which is final, compiler will flash an error i.e.  
**“The Type B cannot the subclass the final Class A”,**if class B is trying to extend final class A.

**public class B extends A**

**{//Error :The Type B cannot the subclass the  Final Class A**

|  |  |
| --- | --- |
|  |  |

##### **2: By making a class constructor private:**

We can also stop a class to be extended/inherited by other classes in Java by making the class constructor private.

If we make the class constructor private we’ll not be able to create the object of this class from outside of this class. But, our purpose is to just prevent a class to be inherited and not to stop object creation. Hence, we need a method that can create an object of this class and return it.

We need to put a static method that will create and return an object. Why Static method? Because, from outside of a class, to call a normal method we need an object of the class, but, as constructor is private, we cannot create an object, hence, only solution is to have a static method that can be called using class name.

So, as a solution to stop a class to be extended, we need to make a constructor private and have one static method that will create an object of this class and return it.

**class A {**

**// Make constructor private to prevent object creation**

**//From outside of this class.**

**private A() {**

**}**

**//Static method to create and return an object.**

**//this method will be called from outside by using**

**//class name only.**

**public static A GetInstance() {**

**return new A();**

**}**

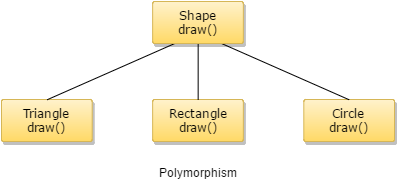
**}**

# 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.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.





### Example of Java Runtime Polymorphism

In this example, we are creating two classes **Bike and Splendor**.

Splendor class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, the subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

**class** Bike{

**void** run(){System.out.println("running");}

}

**class** Splendor **extends** Bike{

**void** run(){System.out.println("running safely with 60km");}

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

    Bike b = **new** Splendor();//upcasting

    b.run();

  }

}

Output:

running safely with 60km.

## Java Runtime Polymorphism Example: Bank

Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest.

****

Types of Polymorphism in Java

* Static Polymorphism
* Dynamic Polymorphism.

Polymorphism can be achieved in two of the following ways:

* **Method Overloading**(Compile time Polymorphism)
* **Method Overriding**(Run time Polymorphism)

## Method Overloading

* To call an overloaded method in Java, it is must use the type and/or the number of arguments to determine which version of the overloaded method actually to call.
* The overloaded methods may have varied return types and the return type single-handedly is insufficient to make out two versions of a method.
* As and when Java compiler encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.
* It permits the user to obtain compile time polymorphism with name method name.
* An overloaded method can throw different kinds of exceptions.
* A method which is overloaded can contain different access modifiers.

### Variations in Overloading a Method

Overloading method's argument lists might differ in:

* Number of parameters passed
* Data type of actual parameters
* Sequence of data type of actual parameters

### Program for Method Overloading in Java

Example:

class Mltply {

void mul(int a, int b) {

System.out.println("Sum of two=" + (a \* b));

}

void mul(int a, int b, int c) {

System.out.println("Sum of three=" + (a \* b \* c));

}

}

class Polymorphism {

public static void main(String args[]) {

Mltply m = new Mltply();

m.mul(6, 10);

m.mul(10, 6, 5);

}

}

## Method Overriding

### Rules of method overriding in Java

* **Argument list**: The argument list at the time of overriding method need to be same as that of the method of the parent class. The data types of the arguments along with their sequence must have to be preserved as it is in the overriding method.
* **Access Modifier**: The Access Modifier present in the overriding method (method of subclass) cannot be more restrictive than that of an overridden method of the parent class.
* The private, static and final methods can't be overridden as they are local to the class.
* Any method which is overriding can throw any unchecked exceptions, in spite of whether the overridden method usually method of parent class might throw an exception or not.

### Program for Method Overriding in Java

Example:

class parent {

public void work() {

System.out.println("Parent is under retirement from work.");

}

}

class child extends parent {

public void work() {

System.out.println("Child has a job");

System.out.println(" He is doing it well");

}

public static void main(String argu[]) {

child c1 = new child();

c1.work();

}

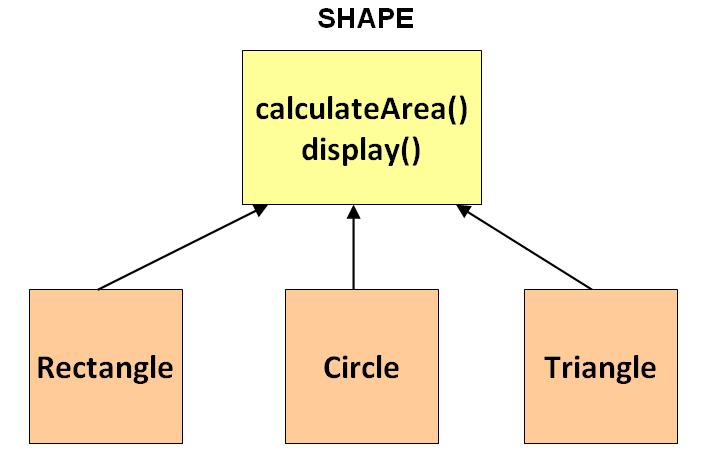
}

# Abstract classes and abstract methods

# Abstract Class

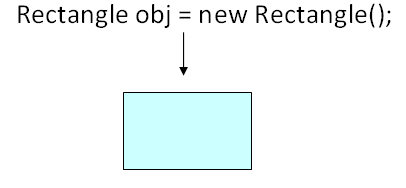
Abstract Classes are classes in Java, that declare one or more abstract methods.

Consider the following class hierarchy consisting of a Shape class which is inherited by three classes Rectangle, Circle, and Triangle. The Shape class is created to save on common attributes and methods shared by the three classes Rectangle, Circle, and Triangle. calculateArea() is one such method shared by all three child classes and present in Shape class.

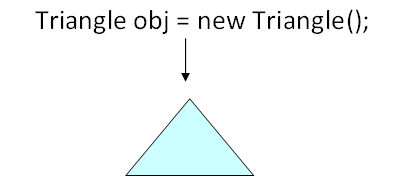
[](https://www.guru99.com/images/uploads/2012/06/java-abstract-classes.jpg)

Now, assume you write code to create objects for the classes depicted above. Let's observe how these **objects will look in a practical world.**

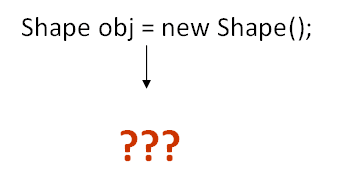
An object of the class rectangle will give a rectangle, a shape we so commonly observed in everyday life.

[](https://www.guru99.com/images/uploads/2012/06/Java_Abstract.png)

An object of the class triangle will give a triangle, again a common everyday shape.

[](https://www.guru99.com/images/uploads/2012/06/Java_Abstract_1.png)

But what would an object of Class Shape look like in a practical world ??

[](https://www.guru99.com/images/uploads/2012/06/java_abstract_method.png)

If you observe the Shape class serves in **our goal of achieving inheritance and polymorphism.**But it was not built to be instantiated.Such classes can be labeled **Abstract**. An abstract class can not be instantiated.

**Syntax:**

abstract class Shape{

// code

}

**It is possible that you DO NOT label Shape class as Abstract and then instantiate it. But such object will have no use in your code and will open a room for potential errors. Hence this is not desirable.**

Abstract Methods

An Abstract Method is a method that has just the method definition but does not contain implementation.

As we all know, the formula for calculating area for rectangle, circle, & triangle is different. The calculateArea() method will have to be overridden by the inheriting classes. It makes no sense defining it in the Shape class, **but we need to make sure that all the inheriting classes do have the method.**

Such methods can be labeled **abstract.**

**Syntax:**

abstract public void calculateArea();

For an **abstract method, no implementation is required.** Only the signature of the method is defined.

## Abstract Class in Java: Important Points

* An abstract class **may**also have concrete (complete) methods.
* For design purpose, a class can be declared abstract even if it does not contain any abstract methods
* Reference of an abstract class can point to objects of its sub-classes thereby achieving run-time polymorphism Ex: Shape obj = new Rectangle();
* A class must be compulsorily labeled abstract, if it has one or more abstract methods.
* abstract classes can’t be instantiated, only subclassed.
* other classes extend abstract classes.
* can have both abstract and concrete methods.
* similar to interfaces, but (1) can implement methods, (2) fields can have various access modifiers, and (3) subclasses can only extend one abstract class.

**Abstract methods:**

* abstract method bodies must be empty (no curly braces)
* subclasses must implement the abstract class’s abstract methods
* Eclipse example: abstract\_classes\_methods