JAVA BASICS Required for Automation

Contents

[1.Object: 2](#_Toc35684630)

[Constructor: 5](#_Toc35684631)

[Default Constructor: 6](#_Toc35684632)

[No arg Constructor: 6](#_Toc35684633)

[Parameterized constructor : 6](#_Toc35684634)

[Constructor Overloading: 6](#_Toc35684635)

[2. Class: 7](#_Toc35684636)

[3. Method: 8](#_Toc35684637)

[1.Modifier: 8](#_Toc35684638)

[2.The return type : 9](#_Toc35684639)

[3. Parameter list : 9](#_Toc35684640)

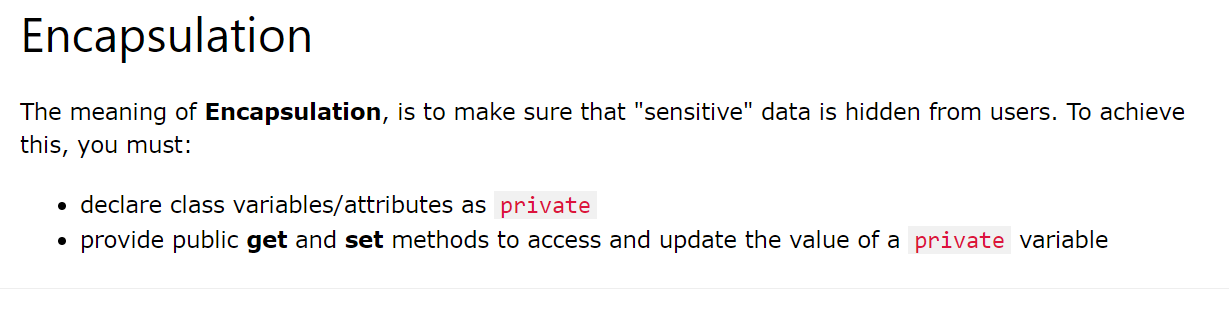
[4.Exception list : 9](#_Toc35684641)

[5.Overriding 9](#_Toc35684642)

[6.Overloading: 9](#_Toc35684643)

[4.Abstraction: 10](#_Toc35684644)

[5.Encapsulation:(security of the data) 10](#_Toc35684645)

[ 10](#_Toc35684646)

[6.Inheritance: 11](#_Toc35684647)

[7.Polymorphism: 14](#_Toc35684648)

The core OOPs concepts:

1. Object
2. Class
3. Method
4. Abstraction
5. Encapsulation
6. Inheritance
7. Polymorphism

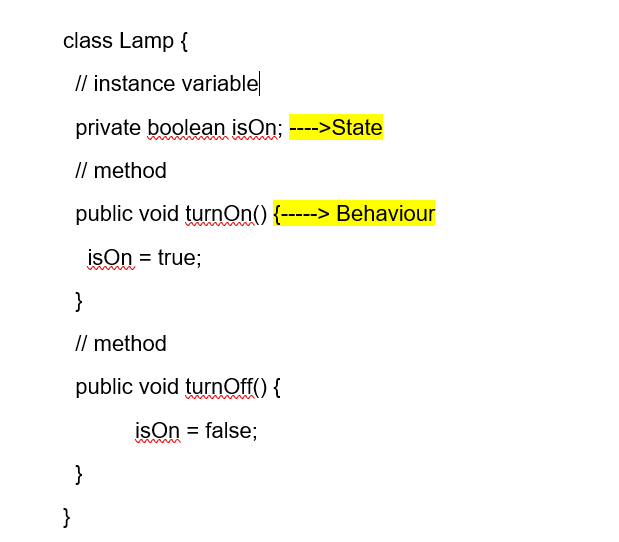
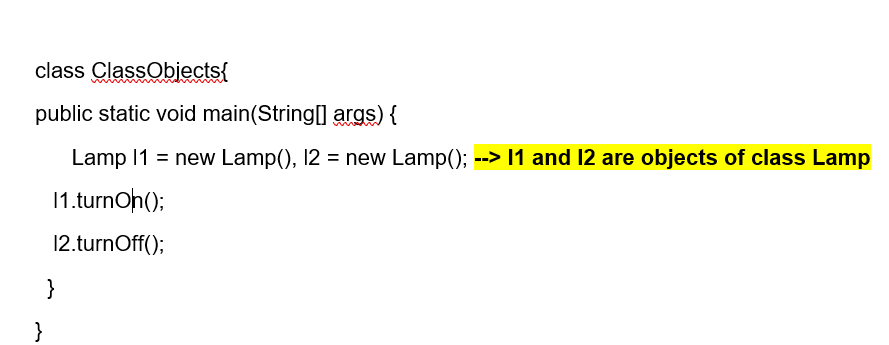
# 1.Object:

The object is an instance(Copy of Original variable or method) of a class. These objects share two characteristics:

* State (Variables of the class)
* Behaviour(Methods )

Consider lamp for example:

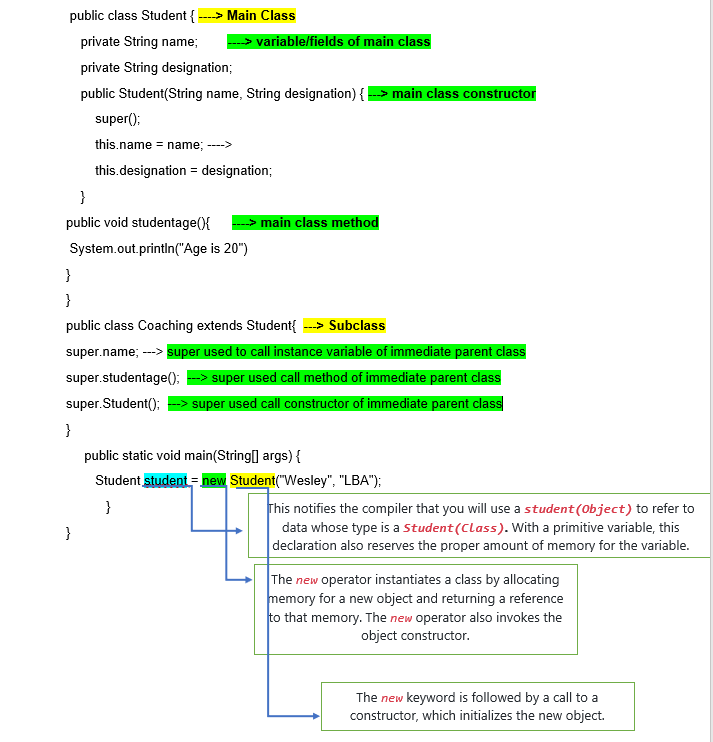
* It can be in on or off (state).
* You can turn on and turn off lamp (behaviour).

* When an object is initialized, it's called an instance. Each instance contains its own copy of these variables. For example, isOn variable for objects l1 and l2 are different.

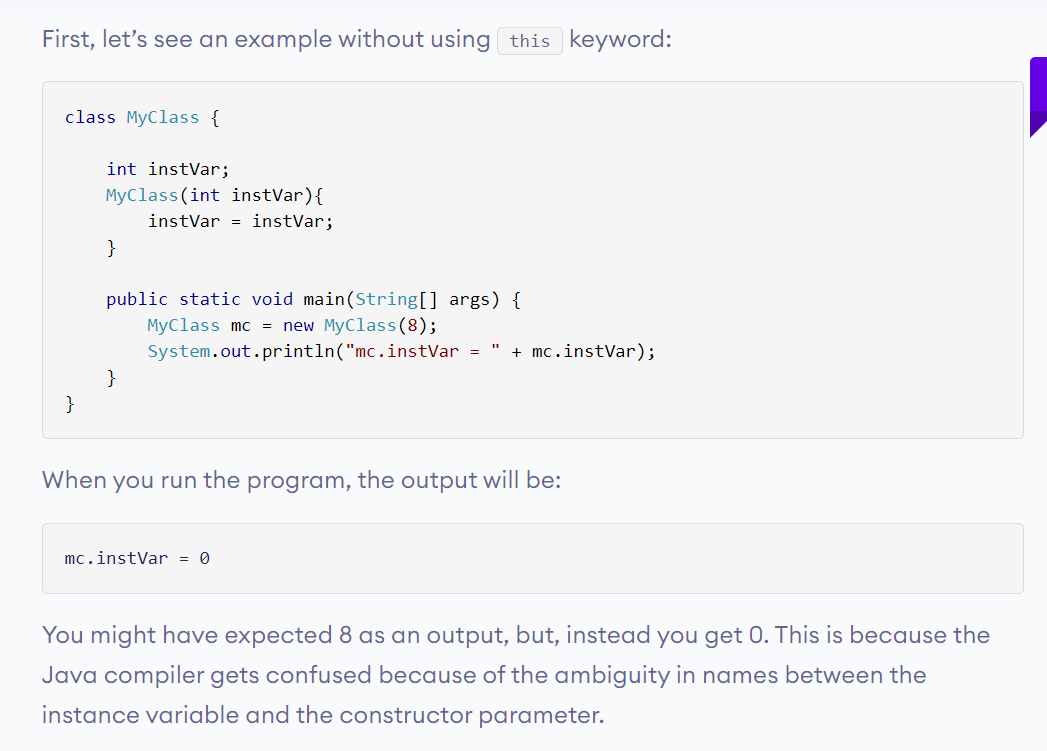
**Note:** **Variables defined within a class are called instance variable for a reason**.

Consider the following programming example:

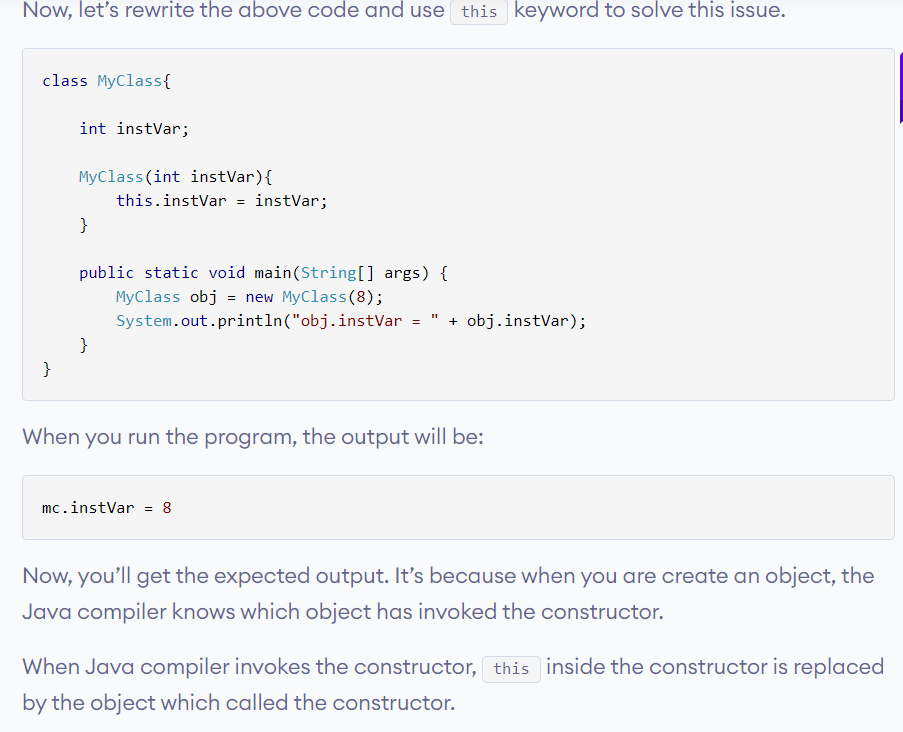


**Use of super(dot) or this(dot) Keyword**

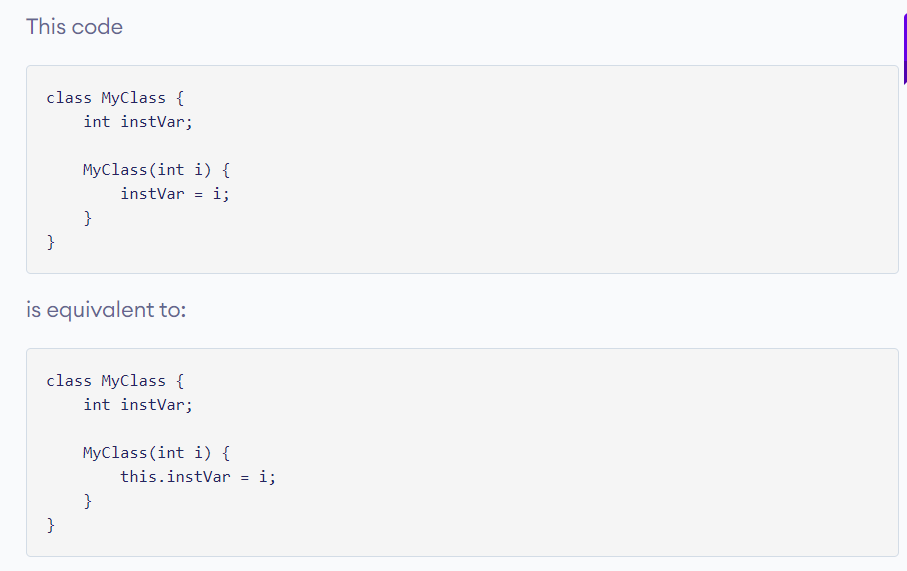
* super can be used to refer immediate parent class instance variable.
* super can be used to invoke immediate parent class method.
* super() can be used to invoke immediate parent class constructor.
* Similarly, **this** keyword is used for referring same class instance variable, method or constructor
* These super and this Keywords used to avoid ambiguity of reference to the compiler which variable belongs to which consider below example.

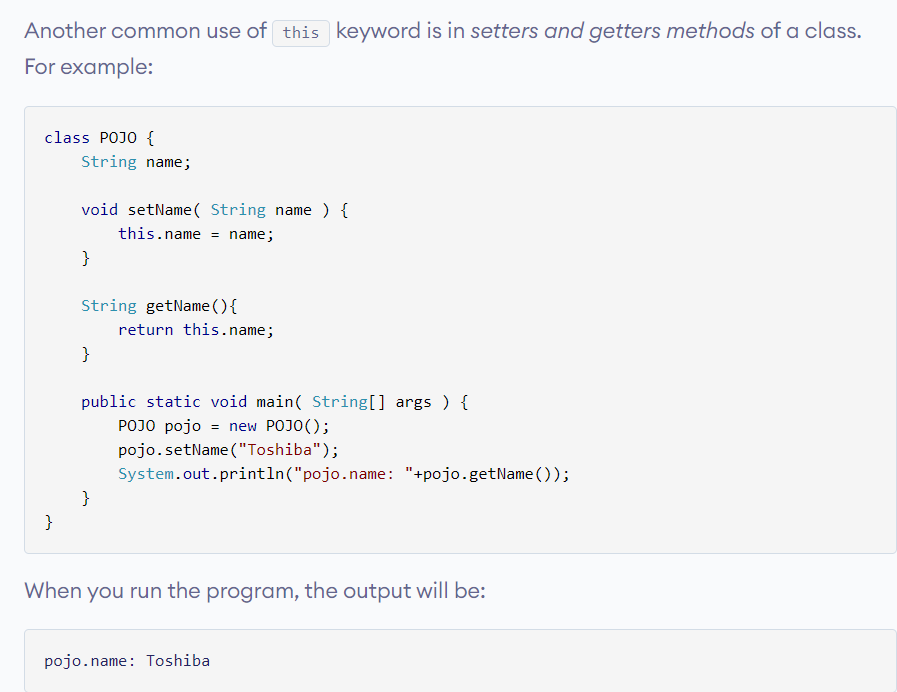


With use of this keyword this ambiguity can be solved.



**Note:** If you pass a parameter that has different name than that of instance variables, the compiler automatically appends this keyword. Consider below example:

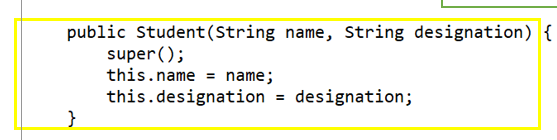




## Constructor:

A constructor is similar to a method (but not actually a method) that is invoked automatically when an object is instantiated.

Java compiler distinguish between a [method](https://www.programiz.com/java-programming/methods) and a constructor by its name and return type. In Java, a constructor has same name as that of the class and doesn’t return any value.



There are different types of Constructor:

1. Default Constructor

2. No arg Constructor

3. Parameterized constructor

### Default Constructor:

If you do not create constructors yourself, the Java compiler will automatically create a no-argument constructor during run-time. This constructor is known as default constructor. The default constructor initializes any uninitialized instance variables.

### No arg Constructor:

If a Java constructor does not accept any parameters, it is a no-arg constructor. It's syntax is:

public/private/protected ClassName() {

// constructor body

### Parameterized constructor :

A constructor may also accept parameters. It's syntax is:

public/private/protected ClassName(arg1,arg 2) {

// constructor body

}

### Constructor Overloading:

While working with constructor overloading, you may find it useful to invoke one constructor from another constructor. But, constructors can not be called explicitly. Hence, to accomplish this, you can use another form of this keyword this().



# 2. Class:

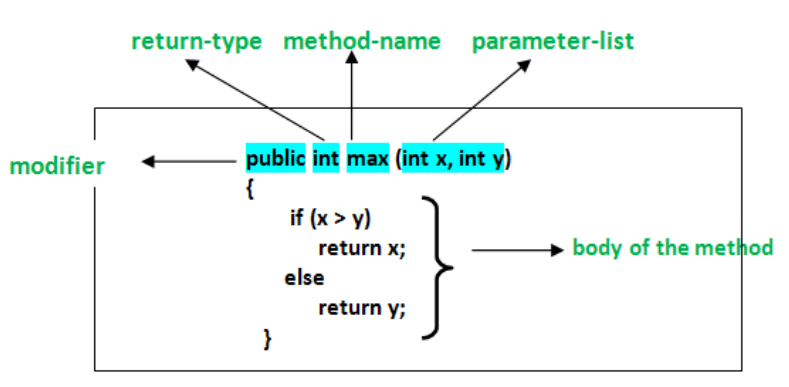
Before you create objects in Java, you need to define a class.

A class is a blueprint for the object. Consider class is a protype of House. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house.

Since, many houses can be made from the same description, we can create many objects from a class.

The variables and methods defined within a class are called members of the class.

# 3. Method:

 In programming, a function is a block of code that performs a specific task.In object-oriented programming, method is a jargon used for function. Methods are bound to a class and they define the behaviour of a class.

## 1.Modifier:

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 class in which it is defined and, in its subclass(es)

**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.

## 2.The return type :

The data type of the value returned by the method or void if does not return a value.

## 3. 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 ().

## 4.Exception list :

The exceptions you expect by the method can throw, you can specify these exception(s).

5.Overriding **:** Overriding is a feature of OOP languages like Java that is related to run-time polymorphism. A subclass (or derived class) provides a specific implementation of a method in superclass (or base class).

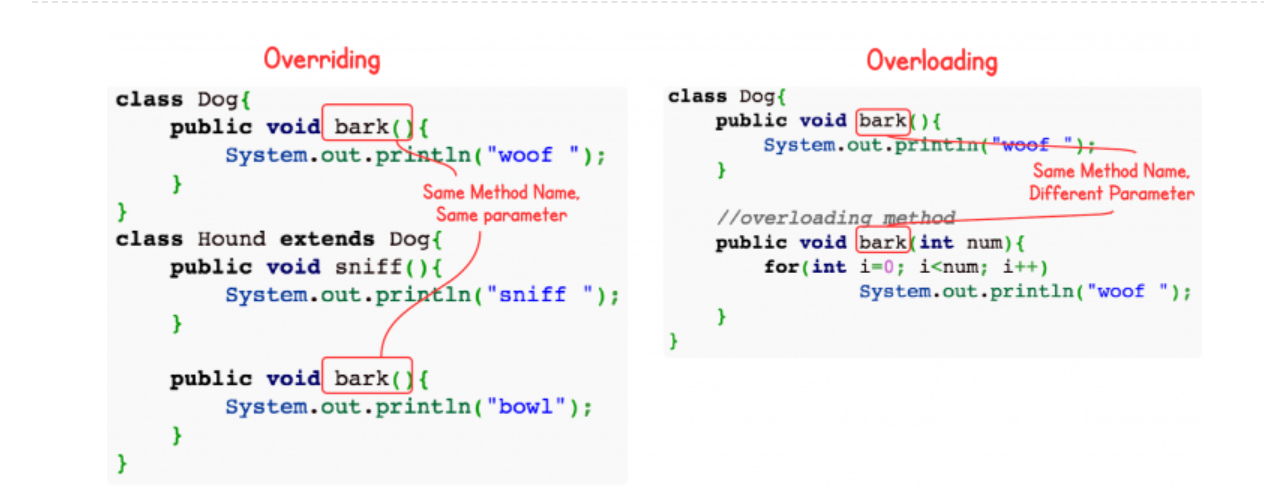
The implementation to be executed is decided at run-time and decision is made according to the object used for call. Note that signatures(arguments) of both methods must be same.

## 6.Overloading:

Overloading is also a feature of OOP languages like Java that is related to compile time (or static) polymorphism. This feature allows different methods to have same name, but different signatures(arguments), especially number of input parameters and type of input parameters.

**Note:** In both C++ and Java, methods cannot be overloaded according to return type.

This happens when running the pgm This happens while compiling the pgm **Runtime Polymorphism**  **Compile time Polymorphism**



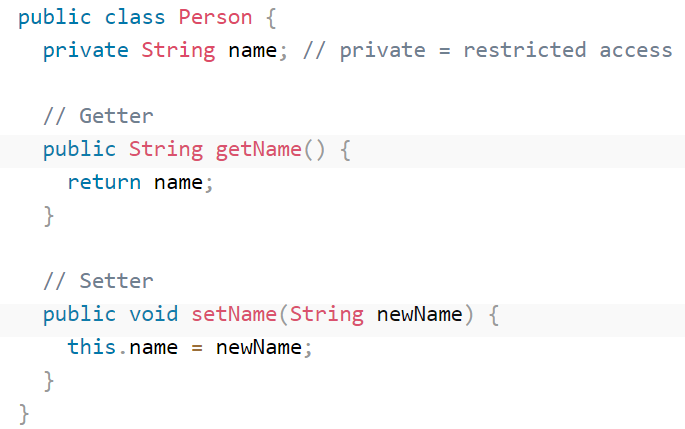
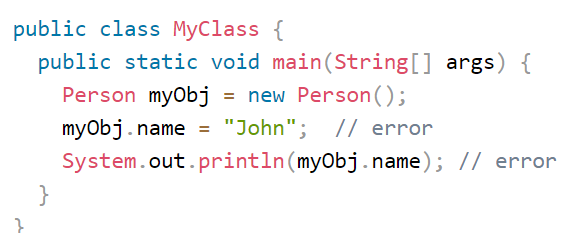
# 4.Abstraction:

# 5.Encapsulation:(security of the data)

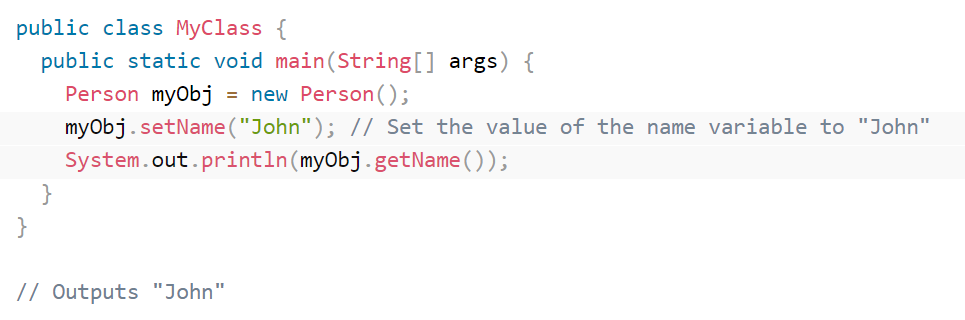
# 

private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public get and set methods.

Look at the following example: error is thrown if we access pvt variables.

However, we can access these variables using getter and setter methods.

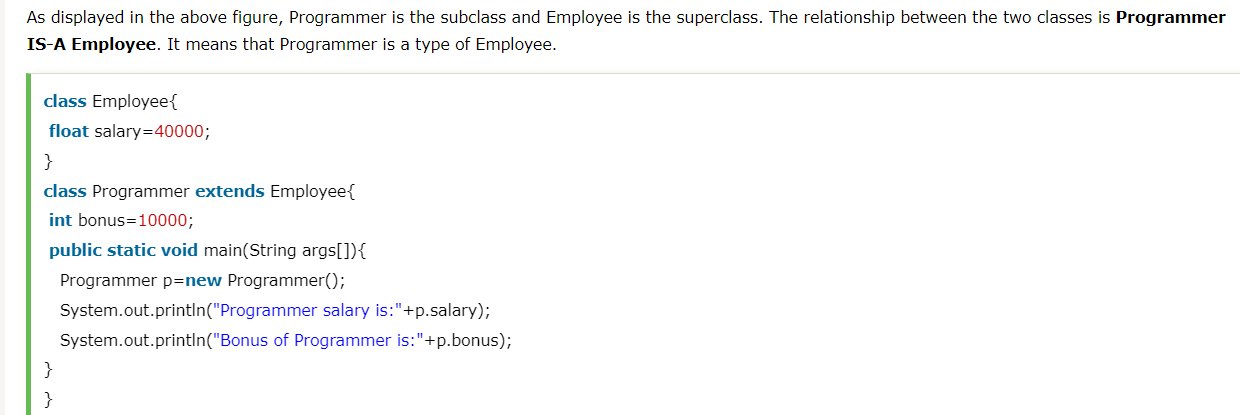
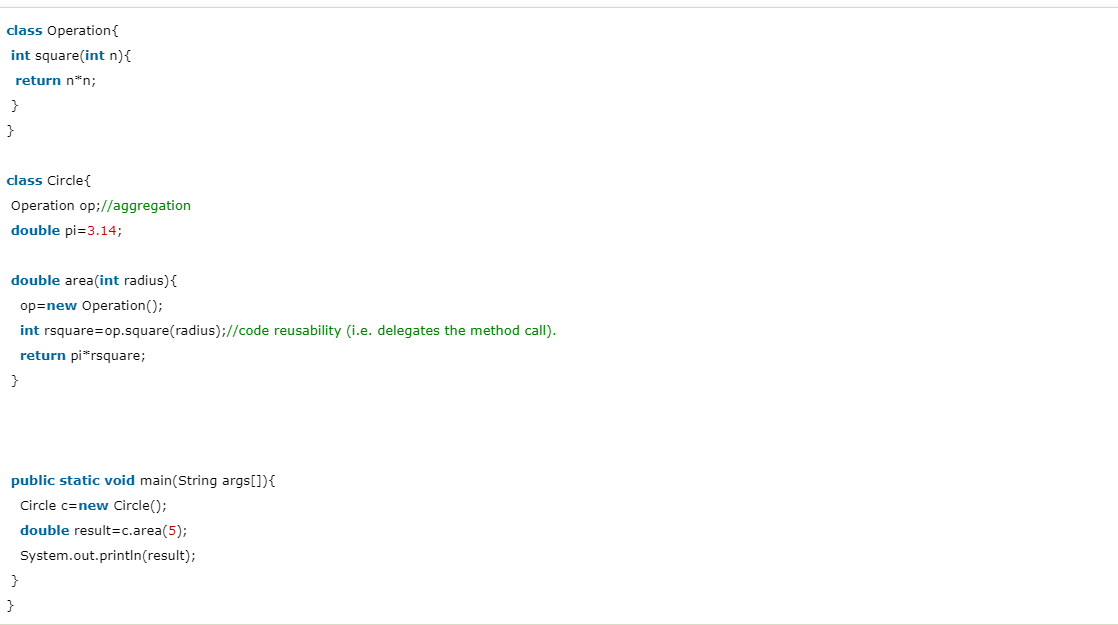


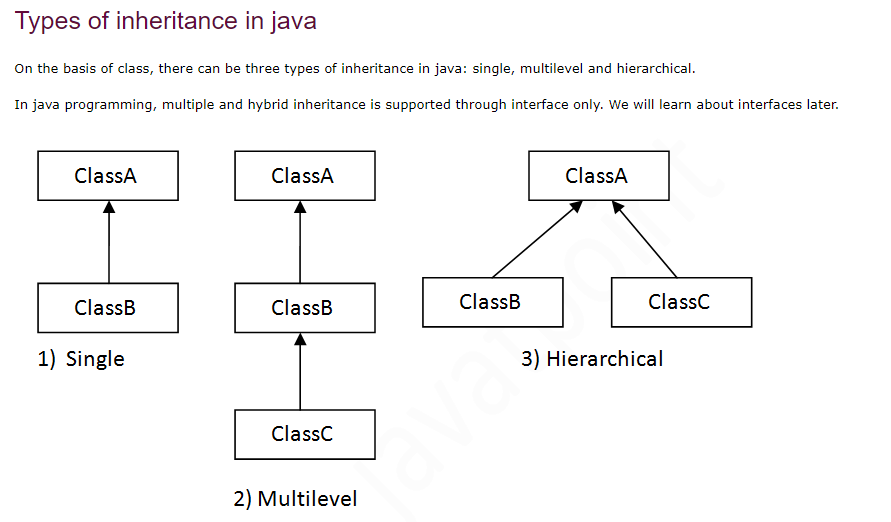
# 6.Inheritance:

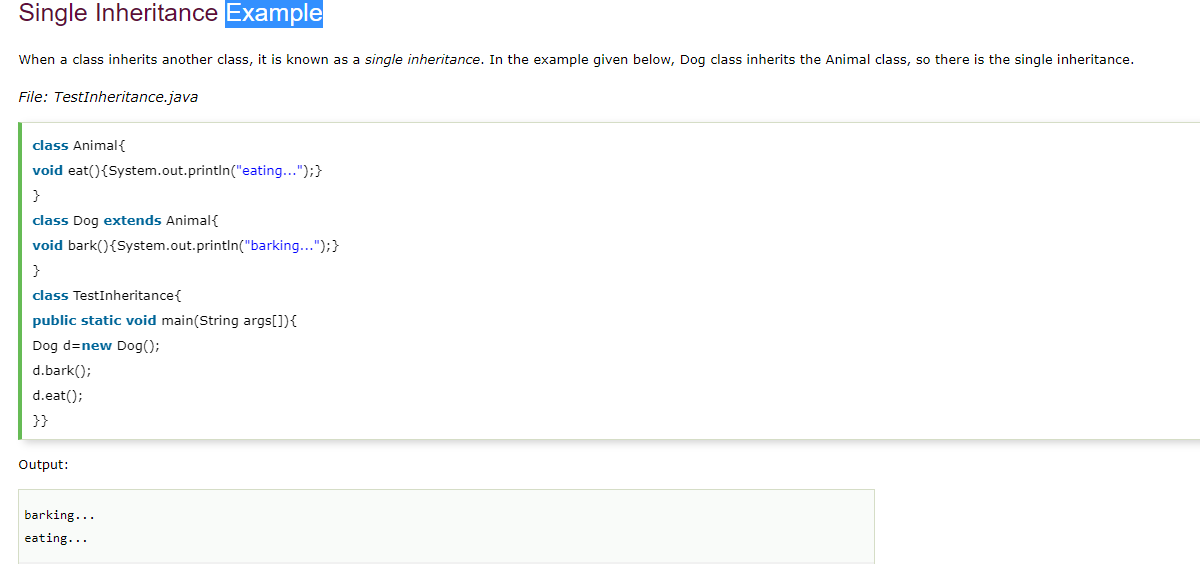
Inheritance in java can be achieved in two ways:

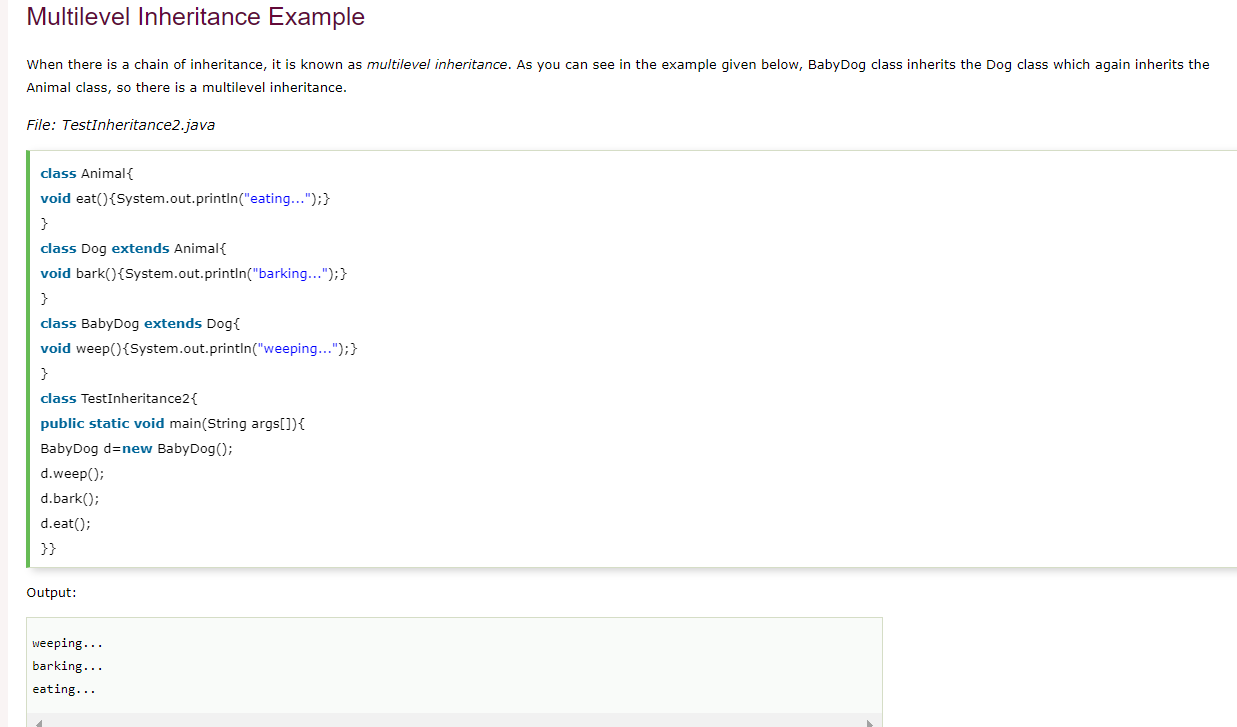
1. By extending the subclass by Parent class using extends keyword(IS-A relation) called inhertance
2. By Creating object for the class which has to be inherited(HAS-A relation) called Aggregation

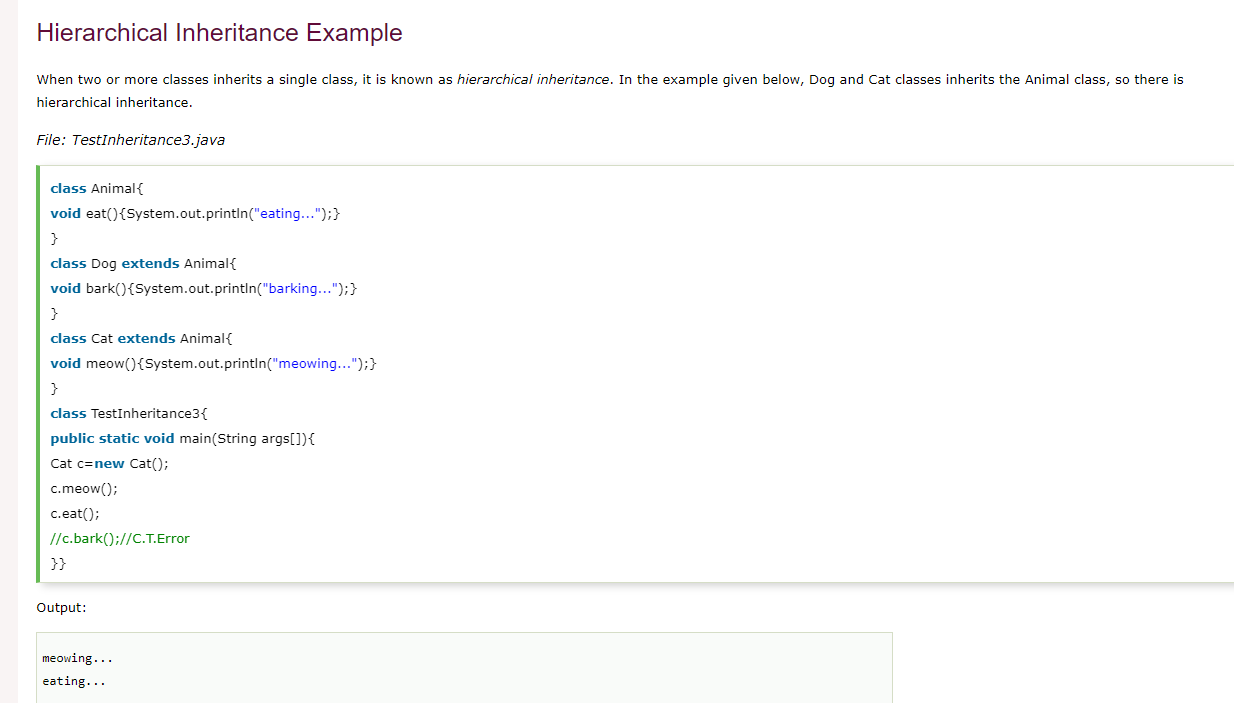
**Type 1:**

**Type 2:** 











# 7.Polymorphism: