**Encapsulation**

**What is Encapsulation?**

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. 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.

**Why you need Encapsulation?**

The main advantages of encapsulation are

* It improves the maintainability of an application.
* Offers flexibility to the user to use the system very easily
* Helps the developers to organize the code better
* Makes the overall coding process easier, as you are only concerned with what another class does, not how it does it
* This method helps the developers to be more 'objective' and result oriented.
* Encapsulated Code is quite flexible and easy to change with new requirements.
* Encapsulation makes unit testing easy.
* It allows you to reduce coupling of modules and increases cohesion inside a module as all piece of one thing are encapsulated in one place.
* Encapsulation helps you to change a part of code without affecting other parts of the code.
* Improves the code readability of the Application
* Enhanced security and makes maintenance of application easier
* The encapsulation interface only allows for well-defined interaction.

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.

|  |
| --- |
| // Java program to demonstrate encapsulation    public class Encapsulate {        // private variables declared      // these can only be accessed by      // public methods of class      private String geekName;      private int geekRoll;      private int geekAge;        // get method for age to access      // private variable geekAge      public int getAge()      {          return geekAge;      }        // get method for name to access      // private variable geekName      public String getName()      {          return geekName;      }        // get method for roll to access      // private variable geekRoll      public int getRoll()      {          return geekRoll;      }        // set method for age to access      // private variable geekage      public void setAge(int newAge)      {          geekAge = newAge;      }        // set method for name to access      // private variable geekName      public void setName(String newName)      {          geekName = newName;      }        // set method for roll to access      // private variable geekRoll      public void setRoll(int newRoll)      {          geekRoll = newRoll;      }  }    // Class to access variables  // of the class Encapsulate  class TestEncapsulation {      public static void main(String[] args)      {          Encapsulate obj = new Encapsulate();            // setting values of the variables          obj.setName("Harsh");          obj.setAge(19);          obj.setRoll(51);            // Displaying values of the variables          System.out.println("Geek's name: " + obj.getName());          System.out.println("Geek's age: " + obj.getAge());          System.out.println("Geek's roll: " + obj.getRoll());            // Direct access of geekRoll is not possible          // due to encapsulation          // System.out.println("Geek's roll: " + obj.geekName);      }  } |

**Output:**

Geek's name: Harsh

Geek's age: 19

Geek's roll: 51

**Abstraction**

## What is Abstraction

Data Abstraction is the property by virtue of which only the essential details are displayed to the user. The trivial or the non-essentials units are not displayed to the user. Ex: A car is viewed as a car rather than its individual components.

Data Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. The properties and behaviours of an object differentiate it from other objects of similar type and also help in classifying/grouping the objects.

## Why you need Abstraction?

Here, are the main reasons why abstraction is needed for Object-Oriented Programming:

* Helps you to simplify the representation of the domain models.
* Abstraction hides the irrelevant details found in the code.
* Abstraction helps you to partition the program into many independent concepts.
* Offers the greatest flexibility when using ADT(Abstract Data Type) objects in different situations

|  |
| --- |
| // Java program to illustrate the concept of Abstraction    abstract class Shape {      String color;        // these are abstract methods      abstract double area();      public abstract String toString();        // abstract class can have a constructor      public Shape(String color)      {          System.out.println("Shape constructor called");          this.color = color;      }        // this is a concrete method      public String getColor()      {          return color;      }  }  class Circle extends Shape {      double radius;        public Circle(String color, double radius)      {            // calling Shape constructor          super(color);          System.out.println("Circle constructor called");          this.radius = radius;      }        @Override      double area()      {          return Math.PI \* Math.pow(radius, 2);      }        @Override      public String toString()      {          return "Circle color is "              + super.color              + "and area is : "              + area();      }  }    class Rectangle extends Shape {        double length;      double width;        public Rectangle(String color,                       double length,                       double width)      {            // calling Shape constructor          super(color);          System.out.println("Rectangle constructor called");          this.length = length;          this.width = width;      }        @Override      double area()      {          return length \* width;      }        @Override      public String toString()      {          return "Rectangle color is "              + super.color              + "and area is : "              + area();      }  }    public class Test {      public static void main(String[] args)      {          Shape s1 = new Circle("Red", 2.2);          Shape s2 = new Rectangle("Yellow", 2, 4);            System.out.println(s1.toString());          System.out.println(s2.toString());      }  } |

**Output:**

Shape constructor called

Circle constructor called

Shape constructor called

Rectangle constructor called

Circle color is Redand area is : 15.205308443374602

Rectangle color is Yellowand area is : 8.0

**Difference between Abstraction and Encapsulation:**

|  |  |
| --- | --- |
| Abstraction | Encapsulation |
| Abstraction is the process or method of gaining the information. | While encapsulation is the process or method to contain the information. |
| In abstraction, problems are solved at the design or interface level. | While in encapsulation, problems are solved at the implementation level. |
| Abstraction is the method of hiding the unwanted information. | Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside. |
| We can implement abstraction using abstract class and interfaces. | Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public. |
| In abstraction, implementation complexities are hidden using abstract classes and interfaces. | While in encapsulation, the data is hidden using methods of getters and setters. |
| The objects that help to perform abstraction are encapsulated. | Whereas the objects that result in encapsulation need not be abstracted. |

# Constructors in Java

Constructors are used to initialize the object’s state. Like [methods](https://www.geeksforgeeks.org/methods-in-java/), a constructor also contains **collection of statements(i.e. instructions)** that are executed at time of Object creation.

**Need of Constructor**  
Think of a Box. If we talk about a box class then it will have some class variables (say length, breadth, and height). But when it comes to creating its object(i.e Box will now exist in computer’s memory), then can a box be there with no value defined for its dimensions. The answer is no.  
So constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

**When is a Constructor called ?**  
Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.

A constructor is invoked at the time of object or instance creation. For Example:

class Animal

{

.......

// A Constructor

new Animal() {}

.......

}

// We can create an object of the above class

// using the below statement. This statement

// calls above constructor.

Geek obj = new Animal();

**Rules for writing Constructor:**

* Constructor(s) of a class must has same name as the class name in which it resides.
* A constructor in Java can not be abstract, final, static and Synchronized.
* Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

**Types of constructor**

There are two type of constructor in Java:

1. **No-argument constructor:** A constructor that has no parameter is known as default constructor. If we don’t define a constructor in a class, then compiler creates **default constructor(with no arguments)** for the class. And if we write a constructor with arguments or no-arguments then the compiler does not create a default constructor.  
   Default constructor provides the default values to the object like 0, null, etc. depending on the type.

|  |
| --- |
| // Java Program to illustrate calling a  // no-argument constructor  import java.io.\*;    class Animal  {      int age;      String name;        // this would be invoked while an object      // of that class is created.      Animal()      {          System.out.println("Constructor called");      }  }    class XYZ  {      public static void main (String[] args)      {          // this would invoke default constructor.          Animal animal1 = new Animal();            // Default constructor provides the default          // values to the object like 0, null          System.out.println(animal1.name);          System.out.println(animal1.age);      }  } |

**Output :**

Constructor called

null

0

1. **Parameterized Constructor:** A constructor that has parameters is known as parameterized constructor. If we want to initialize fields of the class with your own values, then use a parameterized constructor.

// Java Program to illustrate calling of

// parameterized constructor.

import java.io.\*;

class Animal

{

// data members of the class.

String name;

int id;

// constructor would initialize data members

// with the values of passed arguments while

// object of that class created.

Animal(String name, int id)

{

this.name = name;

this.id = id;

}

}

class XYZ

{

public static void main (String[] args)

{

// this would invoke the parameterized constructor.

Animal animal1 = new Animal("Dog", 1);

System.out.println("AnimalName :" + animal1.name +

" and AnimalId :" + animal1.id);

}

}

**Output:**

AnimalName :Dog and AnimalId :1

## What is THIS Keyword in Java?

Keyword THIS is a reference variable in Java that refers to the current object.

The various usages of 'THIS' keyword in Java are as follows:

* It can be used to refer instance variable of current class
* It can be used to invoke or initiate current class constructor
* It can be passed as an argument in the method call
* It can be passed as argument in the constructor call
* It can be used to return the current class instance