Encapsulation in Java

Encapsulation is a fundamental concept in object-oriented programming (OOP) that refers to the bundling of data and methods that operate on that data within a single unit, which is called a class in Java. Encapsulation is a way of hiding the implementation details of a class from outside access and only exposing a public interface that can be used to interact with the class.

In Java, encapsulation is achieved by declaring the instance variables of a class as private, which means they can only be accessed within the class. To allow outside access to the instance variables, public methods called getters and setters are defined, which are used to retrieve and modify the values of the instance variables, respectively. By using getters and setters, the class can enforce its own data validation rules and ensure that its internal state remains consistent.

Here's an example of encapsulation:

Java

```
    private String name;
    private int age;

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }
}

public class Main {
    public static void main(String[] args)
    {
        Person person = new Person();
        person.setName("John");
        person.setAge(30);

        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());
    }
}
```

Output

```
Name: John
Age: 30
```

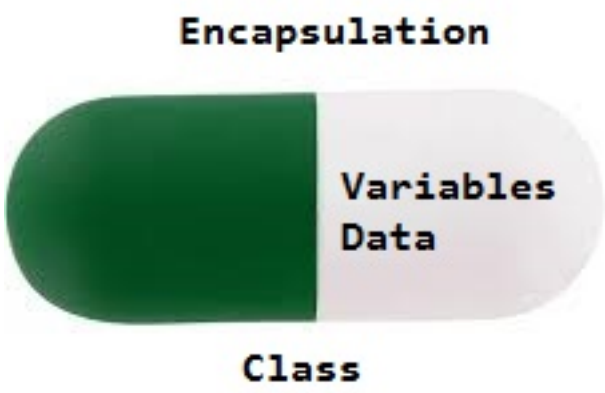
Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds

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 its own class in which it is declared.
- As in encapsulation, the data in a class is hidden from other classes using the data hiding concept which is achieved by making the members or methods of a class private, and the class is exposed to the end-user or the world without providing any details behind implementation using the abstraction concept, so it is also known as a **combination of data-hiding and abstraction**.
- 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.
- It is more defined with the setter and getter method.

Advantages of Encapsulation:

- **Data Hiding:** it is a way of restricting the access of our data members by hiding the implementation details. Encapsulation also provides a way for data hiding. The user will have no idea about the inner implementation of the class. It will not be visible to the user how the class is storing values in the variables. The user will only know that we are passing the values to a setter method and variables are getting initialized with that value.
- **Increased Flexibility:** We can make the variables of the class read-only or write-only depending on our requirements. If we wish to make the variables read-only then we have to omit the setter methods like setName(), setAge(), etc. from the above program or if we wish to make the variables write-only then we have to omit the get methods like getName(), getAge(), etc. from the above program
- **Reusability:** Encapsulation also improves the re-usability and is easy to change with new requirements.
- **Testing code is easy:** Encapsulated code is easy to test for unit testing.
- **Freedom to programmer in implementing the details of the system:** This is one of the major advantage of encapsulation that it gives the programmer freedom in implementing the details of a system. The only constraint on the programmer is to maintain the abstract interface that outsiders see. *For example, the programmer of the edit menu code in a text-editor GUI might at first, implement the cut and paste operations by copying actual screen images in and out of an external buffer. Later, he/she may be dissatisfied with this implementation, since it does not allow compact storage of the selection, and it does not distinguish text and graphic objects. If the programmer has designed the cut-and-paste interface with encapsulation in mind, switching the underlying implementation to one that stores text as text and graphic objects in an appropriate compact format should not cause any problems to functions that need to interface with this GUI. Thus encapsulation yields adaptability, for it allows the implementation details of parts of a program to change without adversely affecting other parts.*



Java

```
// fields to calculate area
class Area {

    int length;
    int breadth;

    // constructor to initialize values
    Area(int length, int breadth)
    {
        this.length = length;
        this.breadth = breadth;
    }
```

```
    {
        int area = length * breadth;
        System.out.println("Area: " + area);
    }
}

class Main {
    public static void main(String[] args)
    {

        Area rectangle = new Area(2, 16);
        rectangle.getArea();
    }
}
```

Output

```
Area: 32
```

The program to access variables of the class EncapsulateDemo is shown below:

Java

```
// Java program to demonstrate encapsulation

class Encapsulate {
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String geekName;
```

```
    // 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; }
}

public class TestEncapsulation {
    public static void main(String[] args)
```

```
        // 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
```

In the above program, the class Encapsulate is encapsulated as the variables are declared private. The get methods like getAge(), getName(), and getRoll() are set as public, these methods are used to access these variables. The setter methods like setName(), setAge(), setRoll() are also declared as public and are used to set the values of the variables.

```
class Name {

    private int age; // Private is using to hide the data

    public int getAge() { return age; } // getter

    public void setAge(int age)
    {
        this.age = age;
    } // setter
}

class GFG {
    public static void main(String[] args)
    {

        Name n1 = new Name();

        n1.setAge(19);

        System.out.println("The age of the person is: "
                           + n1.getAge());
    }
}
```

Output

```
The age of the person is: 19
```

```
class Account {
    // private data members to hide the data
    private long acc_no;
    private String name, email;
    private float amount;
    // public getter and setter methods
    public long getAcc_no() { return acc_no; }
    public void setAcc_no(long acc_no)
    {
        this.acc_no = acc_no;
    }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email)
    {
        this.email = email;
    }
    public float getAmount() { return amount; }
    public void setAmount(float amount)
    {
        this.amount = amount;
    }
}
public class GFG {
    public static void main(String[] args)
    {
        // creating instance of Account class
        Account acc = new Account();
        // setting values through setter methods
```

```
        acc.setEmail("mdfaiz689@gmail.com");
        acc.setAmount(100000f);
        // getting values through getter methods
        System.out.println(
            acc.getAcc_no() + " " + acc.getName() + " "
            + acc.getEmail() + " " + acc.getAmount());
    }
}
```

Output

```
7310805450 MD FAIZ mdfaiz689@gmail.com 100000.0
```

Advantages of Encapsulation in Java:

1. Improves security of an object's internal state by hiding it from the outside world.
2. Increases modularity and maintainability by making it easier to change the implementation without affecting other parts of the code.
3. Enables data abstraction, allowing objects to be treated as a single unit.
4. Allows for easy addition of new methods and fields without affecting the existing code.
5. Supports the object-oriented principle of information hiding, making it easier to change the implementation without affecting the rest of the code.

Disadvantages of Encapsulation in Java:

1. Can lead to increased complexity, especially if not used properly

3. May limit the flexibility of the implementation.

This article is contributed by **Harsh Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 07 Apr, 2023

398

Similar Reads

1. Difference between Abstraction and Encapsulation in Java with Examples
2. Understanding Encapsulation, Inheritance, Polymorphism, Abstraction in OOPs
3. Difference Between java.sql.Time, java.sql.Timestamp and java.sql.Date in Java
4. How to Convert java.sql.Date to java.util.Date in Java?
5. Different Ways to Convert java.util.Date to java.time.LocalDate in Java
6. How to Convert java.util.Date to java.sql.Date in Java?

7. Java.util.BitSet class methods in Java with Examples | Set 2
8. Java.io.BufferedInputStream class in Java
9. Java.util.BitSet class in Java with Examples | Set 1
10. Java.io.BufferedWriter class methods in Java

Related Tutorials

1. Spring MVC Tutorial
2. Spring Tutorial
3. Spring Boot Tutorial
4. Java 8 Tutorial
5. Java IO Tutorial

Previous

Abstraction in Java

Next

Polymorphism in Java

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : sai003, anshikabhatnagar, bipin_kumar, CHANDRASEKHARVARMA, rithwiksv3700, phanitanshi12, mdfaiz589, avinashrat55252, rathoadavinash, swanandbhuskute2003

Article Tags : Java-Object Oriented, Java

Practice Tags : Java

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

Sorting

Searching

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

DevOps

Git

AWS

Docker

Kubernetes

NLP Tutorial

Deep Learning Tutorial

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

Interview Corner

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

Commerce

UPSC

Microeconomics

Macroeconomics

Statistics for Economics

Indian Economic Development

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship