**Encapsulation** is one of the four fundamental Object-Oriented Programming (OOP) concepts, and it refers to the bundling of data (attributes) and methods (functions) that operate on the data into a single unit known as a class. The key idea behind encapsulation is to hide the internal details of an object and to provide a well-defined interface for interacting with that object. Access modifiers like private, public, and protected are commonly used to control the visibility of members within a class.

**Let's break down encapsulation in Java:**

**Private Variables:**

Encapsulation often involves using private variables to restrict direct access from outside the class.

Example:

**Java code**

**public class Car {**

**private String model;**

**private int year;**

**}**

**Public Methods (Getters and Setters):**

**Public methods are used to provide controlled access to the private variables.**

**Getter methods allow retrieving the values of private variables.**

**Setter methods allow modifying the values of private variables.**

**Example:**

java code

public class Car {

private String model;

private int year;

// Getter for model

public String getModel() {

return model;

}

// Setter for model

public void setModel(String model) {

this.model = model;

}

// Getter for year

public int getYear() {

return year;

}

// Setter for year

public void setYear(int year) {

this.year = year;

}

}

**Data Hiding:**

**By encapsulating data, you hide the internal details of the class from the outside world, preventing unintended interference.**

**Example:**

Java code

**public class Car {**

**private String model;**

**private int year;**

**// Methods and logic here...**

**}**

**Information Hiding:**

**Encapsulation helps in achieving information hiding, as the internal implementation details are hidden from the users of the class.**

**Users only need to know how to interact with the public methods, not the internal workings of the class.**

**Example Usage:**

**Using an instance of the Car class:**

**Java code**

**public class Main {**

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

**Car myCar = new Car();**

**myCar.setModel("Toyota");**

**myCar.setYear(2022);**

**// Accessing information using public methods**

**System.out.println("Car Model: " + myCar.getModel());**

**System.out.println("Car Year: " + myCar.getYear());**

**}**

**}**

**In summary,** encapsulation in Java involves grouping data and methods within a class, using access modifiers to control access, and providing well-defined public interfaces for interacting with objects. It promotes modularity, code organization, and maintenance.

**A tightly encapsulated class** **in Java is a class where the internal details, such as the fields (variables) and methods, are strongly encapsulated and hidden from the outside world. This is achieved by using access modifiers like private for fields and methods, allowing only essential operations to be performed through well-defined public interfaces. The concept of a tightly encapsulated class aligns with the principles of information hiding and data protection.**

**Here's an example of a tightly encapsulated class:**

**Java code**

public class Student {

// Private fields

private String name;

private int age;

private double gpa;

// Public constructor

public Student(String name, int age, double gpa) {

this.name = name;

this.age = age;

this.gpa = gpa;

}

// Public getter for name

public String getName() {

return name;

}

// Public getter for age

public int getAge() {

return age;

}

// Public getter for GPA

public double getGpa() {

return gpa;

}

**// Public method to set GPA (with validation)**

**public void setGpa(double newGpa) {**

**if (newGpa >= 0.0 && newGpa <= 4.0) {**

**this.gpa = newGpa;**

**} else {**

**System.out.println("Invalid GPA value. GPA should be between 0.0 and 4.0");**

**}**

**}**

**// Other public methods and business logic...**

**}**

In this example:

The **fields** (name, age, and gpa) are declared as **private**, restricting direct access from outside the Student class.

**Public getter methods** (getName(), getAge(), and getGpa()) provide controlled access to retrieve the values of the private fields.

The setGpa(double newGpa) method allows modifying the GPA with a validation check to ensure it falls within a valid range.

By encapsulating the internal details of the Student class and providing a limited and controlled interface for interaction, the class is considered tightly encapsulated. Users of the class can access and modify information only through the specified methods, maintaining the integrity of the class's data.