**Encapsulation** is one of the fundamental concepts in Object-Oriented Programming (OOP). It refers to the practice of **bundling the data (attributes)** and the **methods (behaviors)** that operate on that data into a **single unit** (usually a class). It also involves **restricting access** to some of the object's components and protecting its internal state by making some fields **private** and providing **public methods** to access or modify those fields.

**Key Points of Encapsulation:**

1. **Data Hiding**: You **hide** the internal details of the object (e.g., its fields) from outside interference. This is usually done by making the fields private.
2. **Controlled Access**: You **provide controlled access** to the data via **public getter and setter methods** (also known as accessors and mutators).
3. **Improved Maintainability**: By using encapsulation, you can change the internal implementation of a class without affecting its external usage, since users of the class only interact with the public methods.

**Java Example of Encapsulation:**

Let's consider an example where we have a **BankAccount** class, and we use **encapsulation** to hide the account balance and PIN from direct access.

// BankAccount class demonstrating Encapsul

ation

public class BankAccount {

// Private fields, encapsulated from outside access

private double balance;

private int pin;

// Constructor to initialize the account with a balance and a PIN

public BankAccount(int pin, double initialBalance) {

this.pin = pin; // Encapsulated, can only be accessed via getter/setter

this.balance = initialBalance; // Encapsulated, can only be accessed via getter/setter

}

// Public method to get the balance (getter method)

public double getBalance() {

return balance; // We control how the balance is accessed

}

// Public method to deposit money into the account

public void deposit(double amount) {

if (amount > 0) {

balance += amount; // We change the balance only via this method

System.out.println("$" + amount + " deposited. New balance: $" + balance);

} else {

System.out.println("Invalid deposit amount.");

}

}

// Public method to withdraw money from the account

public void withdraw(double amount, int enteredPin) {

// Validate the PIN before allowing withdrawal

if (this.pin == enteredPin) {

if (amount <= balance) {

balance -= amount; // We change the balance via this method

System.out.println("$" + amount + " withdrawn. Remaining balance: $" + balance);

} else {

System.out.println("Insufficient funds.");

}

} else {

System.out.println("Incorrect PIN. Access denied.");

}

}

// Public method to set a new PIN (if needed)

public void setPin(int newPin) {

this.pin = newPin; // Allows updating the PIN with controlled access

System.out.println("PIN updated successfully.");

}

// Public method to get the current PIN (getter method)

public int getPin() {

return pin; // Access to the PIN is controlled

}

}

**How Encapsulation Happens in This Code:**

1. **Private Fields**:
   * The fields balance and pin are marked as private, which means they cannot be accessed directly from outside the class. This ensures that no external code can change the balance or PIN without going through the appropriate methods.
   * Example:

private double balance;

private int pin;

1. **Public Methods (Getter and Setter)**:
   * The getBalance() method is a **getter** that provides read-only access to the balance field, allowing outside code to check the balance without directly accessing it.

public double getBalance() {

return balance;

}

* + The deposit() and withdraw() methods are **mutators** that allow changing the balance field. But they perform checks to ensure the balance is modified correctly (e.g., checking if the withdrawal amount is valid).

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public void withdraw(double amount, int enteredPin) {

if (this.pin == enteredPin) {

if (amount <= balance) {

balance -= amount;

} else {

System.out.println("Insufficient funds.");

}

} else {

System.out.println("Incorrect PIN.");

}

}

1. **Control Over Access**:
   * The **PIN** is not directly accessible or modifiable by external code. If someone needs to change the PIN, they must use the setPin() method, which provides a controlled way of changing the PIN. This method allows you to add additional logic if needed, such as validating the new PIN or adding a security check.

public void setPin(int newPin) {

this.pin = newPin;

}