Java Encapsulation

Topics : JAVA
Written on April 08, 2023

Java encapsulation is a fundamental principle of object-oriented programming that allows classes to control their internal data and functionality, while hiding them from other classes. It is achieved through the use of access modifiers (public, private, protected, default) and methods (getters and setters).

Encapsulation is important because it helps maintain the integrity of an object's state, ensuring that it cannot be accessed or modified in an unexpected or unwanted way. It also enables objects to be used in a modular way, so that changes to one part of a program do not affect other parts.

Here is an example of how encapsulation can be used in Java:


```
public class BankAccount {
  private int accountNumber;
  private double balance;

  public BankAccount(int accountNumber, double balance) {
    this.accountNumber = accountNumber;
    this.balance = balance;
  }

  public int getAccountNumber() {
    return accountNumber;
  }

  public double getBalance() {
    return balance;
  }

  public void deposit(double amount) {
    balance += amount;
  }

  public void withdraw(double amount) {
    if (balance >= amount) {
      balance -= amount;
    } else {
```

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

In this example, the `BankAccount` class has two private instance variables (`accountNumber` and `balance`) that can only be accessed from within the class. To provide access to these variables from outside the class, the class provides two public getter methods (`getAccountNumber` and `getBalance`) that return the values of the variables.

To modify the values of these variables, the class provides two public methods (`deposit` and `withdraw`) that modify the `balance` variable in a controlled way. The `withdraw` method, for example, checks to make sure that there are sufficient funds before making the withdrawal.

By encapsulating the internal data and functionality of the `BankAccount` class in this way, the class is able to control how its data is accessed and modified, preventing unintended changes and ensuring that the class can be used in a modular and maintainable way.