



Green University of Bangladesh
Department of Computer Science and Engineering (CSE)
Faculty of Sciences and Engineering
Semester: (Summer, Year:2021), B.Sc. in CSE (Day)

Lab Report NO #2

Course Title: Object Oriented Programming

Course Code: CSE202 Section:242_d2

Lab Experiment Name: Encapsulation

Student Details

	Name	ID
1.	Tushar Rajbongshi	242002140-CSE202-242D2

Submission Date : 14/12/2025
Course Teacher's Name : Sabbir Hosen Mamun

<u>Lab Report Status</u>	
Marks:	Signature:
Comments:	Date:

1. TITLE OF THE LAB REPORT EXPERIMENT:Encapsulation

2.OBJECTIVES:

1. We will be able to understand the concept of Encapsulation in Java.
2. We will be able to learn how data hiding is achieved using access modifiers.
3. We will be able to demonstrate controlled access to class data using getter and setter methods.

3.Theory:

Encapsulation means binding data and the methods (or functions) that operate on that data together into a single unit (such as a class), so that external parts cannot directly modify the data. Instead, the data can only be accessed through specific methods. As a result, data security is ensured and code management becomes easier. Encapsulation is one of the core principles of Object-Oriented Programming (OOP).

Example:

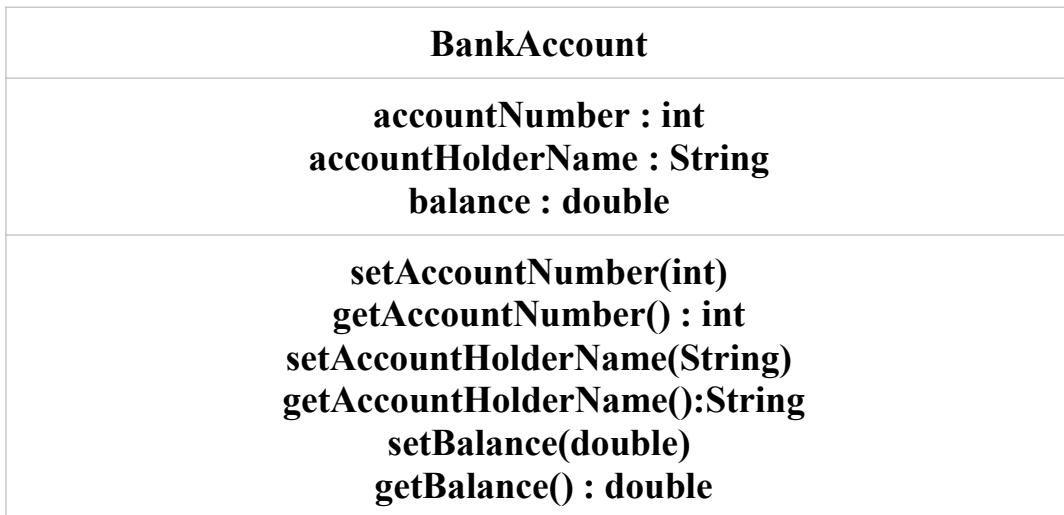
Just as a car's engine cannot be modified directly from outside and can only be controlled using specific controls (such as the steering wheel and pedals), encapsulation works in a similar way.

Main advantages of Encapsulation:

1. Data security (Data Hiding / Security)
2. Code modularity (Modularity)
3. Maintainability
4. Control and validation
5. Code reusability (Reusability)

4.PROCEDURE / ANALYSIS / DESIGN :

4.1: UML class diagram:



5.IMPLEMENTATION:

```
class BankAccount
{
    private int accountNumber;
    private String accountHolderName;
    private double balance;

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

    public void setAccountNumber(int accountNumber )
    {
        this.accountNumber=accountNumber;
    }
    public int getAccountNumber()
    {
        return this.accountNumber;
    }
}
```

```

}

public void setAccountHolderName(String HolderName )
{
    this.accountHolderName=HolderName;
}
public String getAccountHolderName()
{
    return this.accountHolderName;
}

public void setBalance(double balance)
{
    if(balance>0) {
        this.balance=balance;
    }
    else {
        System.out.println("Wroung balance");
    }
}
public double getBalance()
{
    return this.balance;
}

public void displayInfo()
{
    System.out.println("Account      Name:      "+this.accountHolderName      +
"\nAccount Number: " +this.accountNumber+"\nAccount Balance: "+this.balance+"\n");
}
}

public class Encapsulation {
    public static void main(String[] args) {
        BankAccount B1=new BankAccount(242002140,"Tushar Rajbongshi",
50000);
        BankAccount B2=new BankAccount(242002100,"Sojal Rajbongshi", 60000);
        BankAccount B3=new BankAccount(242002000,"Drjoy Rajbongshi", 30000);
        B1.displayInfo();
        B2.displayInfo();
    }
}

```

```
        B3.displayInfo();
    }
}
```

6.TEST RESULT / OUTPUT:

```
[Running] cd "c:\Users\USER\Downloads\" && javac Encapsulation.java && java Encapsulation
Account Name: Tushar Rajbongshi
Account Number: 242002140
Account Balance: 50000.0

Account Name: Sojal Rajbongshi
Account Number: 242002100
Account Balance: 60000.0

Account Name: Drjoy Rajbongshi
Account Number: 242002000
Account Balance: 30000.0

[Done] exited with code=0 in 0.616 seconds
```

7. ANALYSIS AND DISCUSSION:

The main objective of this experiment was to analyze the concept of Encapsulation in Java OOP and to observe its practical implementation through a real-world example. For this purpose, a BankAccount class was designed, where all important data members were declared as private. As a result, the internal data of the class cannot be accessed or modified directly from outside the class.

During the implementation of the program, it was observed that Encapsulation plays a vital role in ensuring data security and data integrity. For example, the balance variable cannot be modified directly from the main method. Instead, the balance must be updated through the setBalance() method, where any negative input value is automatically rejected. This demonstrates that Encapsulation ensures data validation and prevents invalid or incorrect input.

Furthermore, this program highlights another important aspect of Encapsulation—maintainability and flexibility. If the internal logic of the

BankAccount class is changed in the future (such as adding transaction history or modifying balance calculation rules), no changes will be required in the external code as long as the structure of the public methods remains unchanged.

In conclusion, the use of Encapsulation provides the program with a well-organized and modular structure. Each BankAccount object functions as a self-contained unit, making the code more readable, easier to debug, and extensible for future enhancements. This analysis clearly demonstrates that Encapsulation is a crucial OOP principle, essential for developing secure, reliable, and maintainable software.

8. GitHub Repository Link:

<https://github.com/Rajbongshi2005/LabReport02-Encapsulation.git>