IS 2104 - Rapid Application Development

Object Oriented Concepts

Encapsulation

M.V.P. Thilini Lakshika
University of Colombo School of Computing
tlv@ucsc.cmb.ac.lk

Lesson Outline

- What is Encapsulation ?
- Why Encapsulation is important?
- Access Modifiers
- Abstraction vs. Encapsulation
- Advantage of Encapsulation

Introduction

Assume that you are going to a vending machine and pay 100 rupees for an orange juice which cost only 80 rupees.

How the machine knows I gave it a valid 100 rupees note instead of a piece of printed paper ?

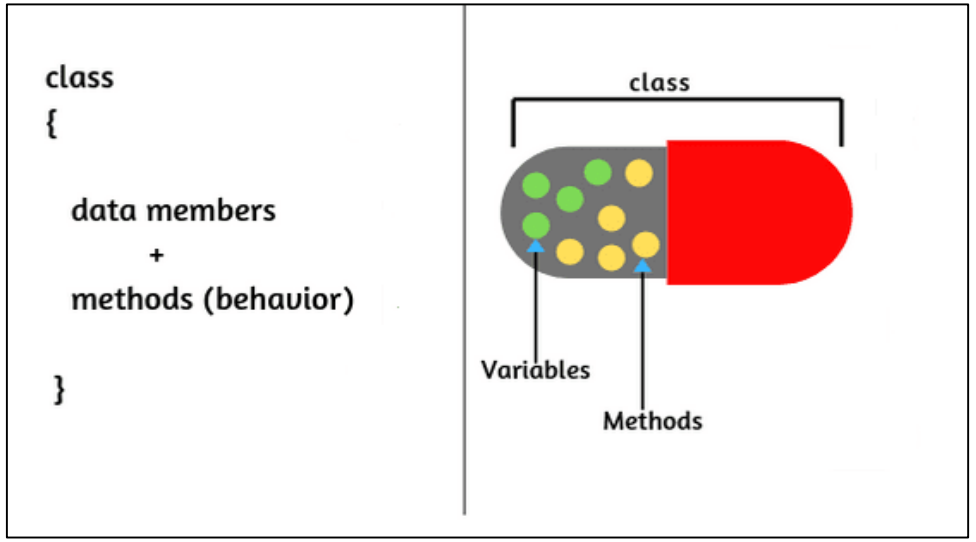
How the machine does the math to give me the balance?

What is the amount of money the machine has in it ?

You are unaware of these information. Its functions are **encapsulated**.

Encapsulation

- Encapsulation is the process of wrapping the data (variables) and code (methods) together into a single unit.
- Encapsulation can be seen as a protective shield that prevents the data from being accessed by the outside this shield.
- The variables of a class will be hidden from other classes, and can be accessed only any member function of own class in which they are declared.
- Therefore, encapsulation also known as data hiding or information hiding.



A capsule which is mixed of several medicines

Why Encapsulation is important?

- Machine is keeping track of how much money it has available, how much each item costs and how much money you give it.
- Machine has a function to distribute the right item if you provide it enough money and a function to calculate how much change to give you.
- This allows you to simply walk up, give it money, select an item, get the item and your change and go enjoy life without worrying about the complex calculations going on behind the scenes.
- Hide the hard work from the user while providing them access to the functionality they need.
- This is why encapsulation is so great in programming.

Encapsulation

- How to achieve encapsulation in Java?
  - Declare members or methods of a class as private.
  - Provide public setter and getter methods to modify (set) and view (get) the variables values.
  - The class is exposed to the world using the abstraction concept (combination of data-hiding and abstraction).
- You can create a fully encapsulated class in Java by making all the data members of the class private.

Encapsulation

- If a data member is declared private, then it can only be accessed within the same class.
- No outside class can access data member of that class.
- If you need to access these variables, you have to use public getter and setter methods.
- Getter and Setters methods are used to create, modify, delete and view the variables values.

```
class Account{

private int account_number;
private int account_balance;

// getter method
public int getBalance() {
        return this.account_balance;
            }

// setter method
public void setNumber(int num) {
        this.account_number = num;
            }
}
```

Encapsulation Example

Consider the given bank account class with deposit and showData methods.

```
class Account {
   private int account_number;
   private int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
      if (a < 0) {
         //show error
      } else
         account_balance = account_balance + a;
   }
}
```

Encapsulation Example

- Assume that, a hacker managed to gain access to the code of your bank account.
- Now, hacker tries to manipulate your account by two ways.

Attempt 1

- Hacker tries to deposit an invalid amount of money (-100) into a bank account by manipulating the code.
- Is that possible?

```
class Hacker {
        Account a = new Account ();
        a.account_balance= -100;
}
```

Encapsulation Example

- Usually, a variable in a class are set as private as shown in the code.
- If a data member is private, it means it can only be accessed with the methods defined in the same class.
- No outside class can access private data member or variable of other class.
- So, hacker cannot deposit amount of -100 to the account.

```
class Account {
   private int account_number;
   private int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
      if (a < 0) {
         //show error
      } else
         ac
   }
}
```

```
class Hacker {
        Account a = new Account ();
        a.account_balance= -100;
}
```

Encapsulation Example

Attempt 2

- Hacker's first attempt failed to deposit the amount in to the bank account.
- Next, he tries to do deposit a amount of -100 by using deposit method.

```
class Hacker {
        Account a = new Account ();
        a.deposit(-100);
}
```

Encapsulation Example

- But the implementation of the deposit method has a condition to check the negative values.
- So, the second attempt also fails.

```
class Hacker {
        Account a = new Account ();
        a.deposit(-100);
}
```

```
class Account {
   private int account_number;
   private int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
      if (a < 0) {
         //show error
      } else
         account_balance = account_balance + a;
   }
}
```

Encapsulation Example

- You never expose your data to an external party.
- These measures make your application secure.
- The entire code can be considered as a capsule, and you can only communicate through the messages.

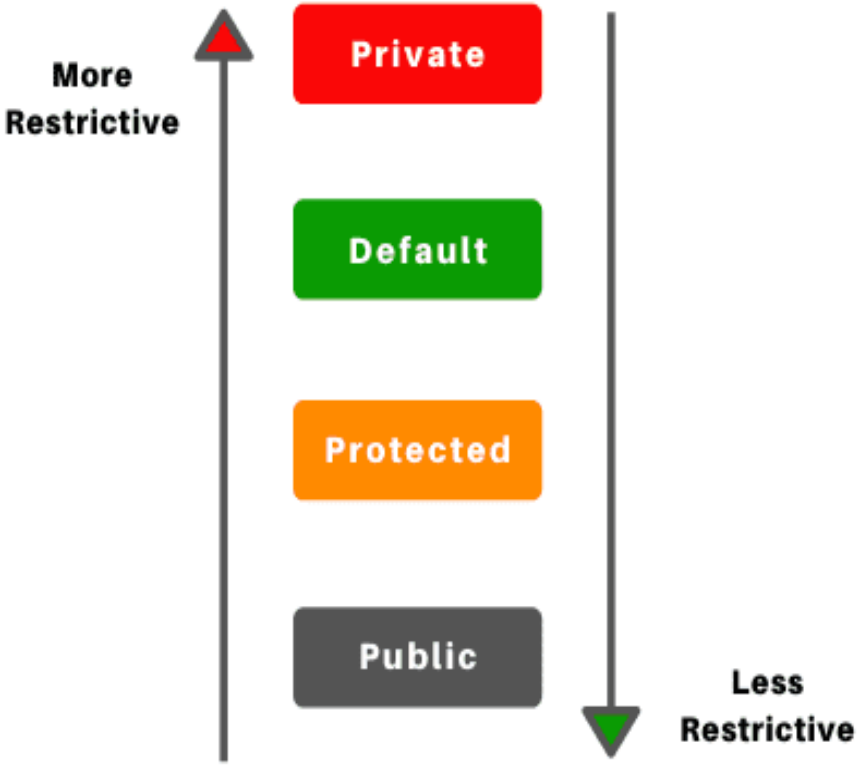
```
class Account {
   private int account_number;
   private int account_balance;

   public void showData() {
      //code to show data
   }

   public void deposit(int a) {
      if (a < 0) {
         //show error
      } else
         account_balance = account_balance + a;
   }
}
```

Access Modifiers

- Encapsulation in Java would not be as effective without access modifiers.
- Access modifiers specify how parts of your class and class members can be accessed by other classes in your program.
- Access modifiers : private, default, protected, public
- Access modifiers can be used to restrict access to critical data members in your code, which improves security.
- To achieve a lesser degree of encapsulation in Java, you can use modifiers like protected or public.



Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

Access Modifiers

Default access modifier

- When no any access modifier mentioned, it is called default access modifier.
- The scope of this modifier is limited to the package only.
- A class, method or data member with the default access modifier in a package, would not be visible in the class of another package.
- Only other classes and methods that are in the same package can access those.

Access Modifiers

Example - Default access modifier

```
package operations;

public class Addition {
  /* No any access modifier here, consider as default*/
  int addTwoNumbers(int a, int b){
            return a+b;
  }
}
```

```
package testOperation;

import operations.*;
public class Test {
  public static void main(String args[]){
            Addition obj = new Addition();
            /* Throw error because we are trying to access the default method in another package */
            obj.addTwoNumbers(10, 21);
  }
}
```

Output : Exception in thread "main" java.lang.Error: Unresolved compilation problem: The method addTwoNumbers(int, int) from the type Addition is not visible at testOperation.Test.main

Access Modifiers

Private access modifier

- The scope of private modifier is limited to the class only.
- Class and Interface cannot be declared as private.
- If a class has private constructor then you cannot create the object of that class from outside of the class.

Example - Private access modifier

```
class Square{
  private string color= “red”;
  private int area(int a){
            return a*a;
  }
}
public class TestSquare{
  public static void main(String args[]){
            Square obj = new Square();
            System.out.println(obj.color);
            System.out.println(obj.area(10));
  }
}
```

Output : Compile - time error

Access Modifiers

Protected Access Modifier

- Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package.
- Protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes.
- Classes cannot be declared protected.
- This access modifier is generally used in a parent-child relationship.

Access Modifiers

Example - Protected Access Modifier

```
package operations;

public class Addition {
        protected int addTwoNumbers(int a, int b){
                return a+b;
        }
}
```

```
package testOperation;

import operations.*;
class Test extends Addition {
  public static void main(String args[]){
        Test obj = new Test();
        System.out.println(obj.addTwoNumbers(10, 20));
        }
}
```

Output : 30

Abstraction vs. Encapsulation

- Often encapsulation is misunderstood with abstraction.
  - Encapsulation is more about How to achieve a functionality.
  - Abstraction is more about What a class can do.

Ex: A mobile phone

- All the complex logic in the circuit board is encapsulated in a touch screen.
- The interface is provided to abstract it out.

Advantage of Encapsulation

- Allows to create read-only or write-only classes depending on the requirements.

  - Using setter or getter method, you can make the class read-only or write-only.

  - You can skip the getter or setter methods accordingly.

- Provides the control over the data.

  - You can write the logics inside the setter method to control data.

  Ex: Set the value of student_id which should be greater than 100 only.

  Set the student_age not to store the negative numbers.

Advantage of Encapsulation

- Facilitate Data Hiding.

  - How the class is storing values in the variables is not visible to the user.

- Improves the reusability and easy to change with new requirements.

- Testing code is easy.

  - Encapsulated code is easy to test for unit testing.

- Provide better management or grouping of related data.

- Developers can change one part of the code easily without affecting other.

Exercise

Conduct a detailed comparison between Abstraction and Encapsulation concepts in OOP.

END