Core Java Notes

Topic – Encapsulation (Data Hiding)

Encapsulation in Java: Detailed Notes

Definition:where the data (variables) and methods (functions) that operate on the data are bundled together into a single unit, known as a class. It restricts direct access to some of the object's components, which helps in protecting the internal state of the object and controlling how the data is accessed or modified.

Key Components:

1. **Private Variables:**
   - Variables are declared as `private` to prevent direct access from outside the class. This ensures that the internal representation of the object is hidden from other classes.
   - **Example:**

```
java
Copy code
private String name;
private int age;
```

2. **Public Methods (Getters and Setters):**
   - **Getters:** Methods that retrieve the values of private variables. They are publicly accessible, allowing other classes to read the values.
   - **Setters:** Methods that modify the values of private variables. They are also publicly accessible but allow controlled modification of the data.
   - **Example:**

```
public String getName()
    {
        return name;
    }

    public void setName(String name)
    {
        this.name = name;
    }

    public int getAge()
    {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

Benefits of Encapsulation:

1. **Control Over Data:**
   - Encapsulation allows for controlled access to the data by using getter and setter methods. This can help ensure that the data is manipulated in a controlled and predictable way.
2. **Flexibility and Maintenance:**
   - The internal implementation of a class can be changed without affecting other classes that use it. As long as the public interface (getters and setters) remains the same, changes to the internal logic will not impact external code.
3. **Increased Security:**
   - By hiding the internal details of the class and exposing only necessary methods, encapsulation helps protect the integrity of the data and prevents unauthorized access or modification.
4. **Reduced Complexity:**
   - Encapsulation simplifies the interaction with the class by hiding complex internal details. It provides a clear interface for interacting with the object, making the code easier to understand and use.

Example Without Conditional Logic in Setter

Here is a straightforward example of encapsulation where the setter methods do not include any conditional logic:

```
class Encapsulation
{
    private int age;
    private String name;
    Encapsulation(int a, String n)
    {
        this.age=a;
        this.name=n;
    }
    public void setAge(int age)
    {
        this.age=a;
    }
    public int getAge()
    {
        return age;
    }
    public void setName(String name)
    {
        this.name=n;
    }
    public String getName()
    {
        return name;
    }
```

```
    public static void main(String args[])
    {
        Encapsulation EN=new Encapsulation(23,"Reema");
        Encapsulation EN1=new Encapsulation(20,"Vishwa");
        System.out.println("Age is " + EN.getAge());
        System.out.println("name is " + EN.getName());
        System.out.println("Age is " + EN1.getAge());
        System.out.println("name is " + EN1.getName());

    }
}
```

Explanation:

1. **Private Variables:**
   - `name` and `age` are private variables, meaning they can only be accessed and modified within the `Person` class.
2. **Constructor:**
   - Initializes a new `Person` object with the specified `name` and `age`.
3. **Getter Methods:**
   - `getName()` returns the value of `name`.
   - `getAge()` returns the value of `age`.
4. **Setter Methods:**
   - `setName(String name)` sets the value of `name` without any checks or constraints.
   - `setAge(int age)` sets the value of `age` without any validation.
5. **Main Method:**
   - Demonstrates creating an instance of the `Person` class and using the getters and setters to access and modify the private variables.

Conclusion:

Encapsulation helps in creating well-defined classes with controlled access to their internal data. Even without conditional logic in the setters, encapsulation still provides benefits such as data hiding, improved maintainability, and simplified interaction with class objects. This approach emphasizes the core principle of encapsulation: keeping the internal state private and exposing it through public methods.