SYNTAX TECHNOLOGIES

JAVA

Class 29

Agenda

Encapsulation in Java

Practice on JAVA OOPS concepts

Encapsulation

Encapsulation is one of the four fundamental OOP concepts.

Encapsulation in Java is a mechanism for wrapping the data (variables) and code acting on the data (methods) together as a single unit.

Encapsulation is achieved in java language by class concept.

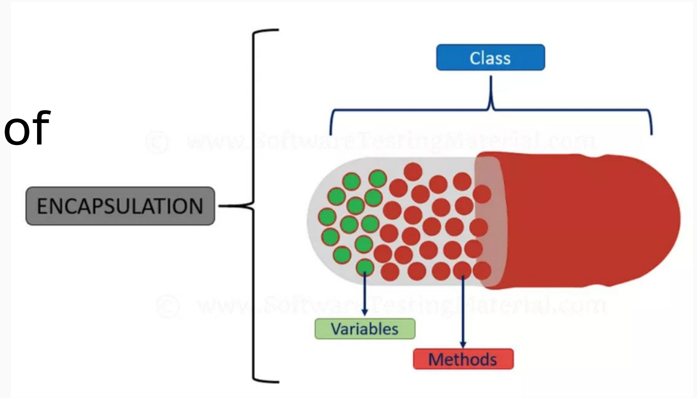
Combining of state and behavior in a single container is known as encapsulation.

In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

Encapsulation

The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real life example of Encapsulation
The common example of encapsulation is capsule. In capsule all medicine are encapsulated inside capsule.



Encapsulation

To achieve encapsulation in Java

- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Encapsulation

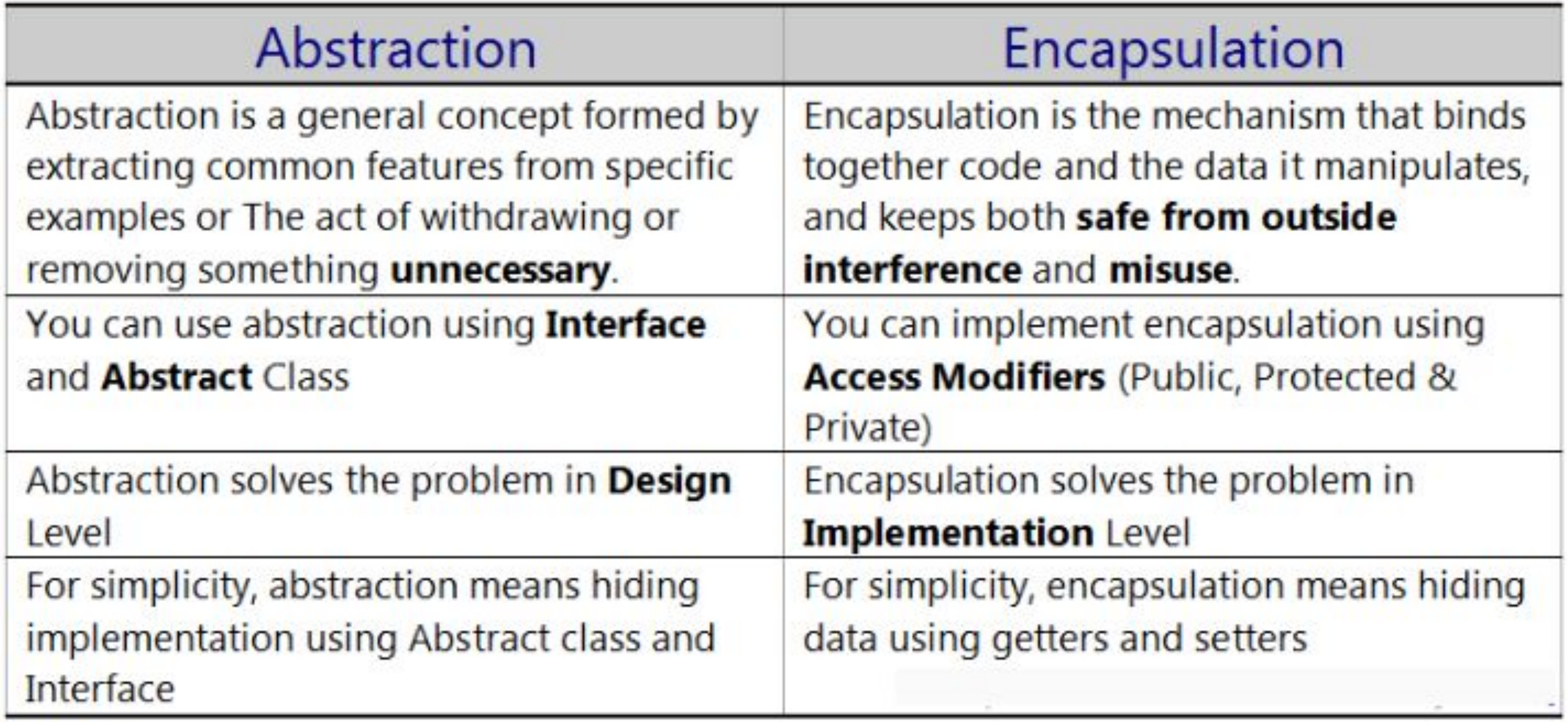
```
class Employee {

    private String name;

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

class Demo {
       public static void main(String[] args) {
               Employee e=new Employee();
               e.setName("Harry");
               System.out.println(e.getName());
        }
}
```

Encapsulation

Benefits of Encapsulation

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.

Abstraction vs Encapsulation



Abstraction	Encapsulation
Abstraction is a general concept formed by extracting common features from specific examples or The act of withdrawing or removing something **unnecessary**.	Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both **safe from outside interference** and **misuse**.
You can use abstraction using **Interface** and **Abstract** Class	You can implement encapsulation using **Access Modifiers** (Public, Protected & Private)
Abstraction solves the problem in **Design** Level	Encapsulation solves the problem in **Implementation** Level
For simplicity, abstraction means hiding implementation using Abstract class and Interface	For simplicity, encapsulation means hiding data using getters and setters

Task

1. Create an Interface 'Shape' with undefined methods as calculateArea and calculatePerimiter . Create 2 child classes: Circle & Square that should have an implementation of area and perimeter calculation. Test your code.

2. We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create class 'Marks' with an abstract method 'getPercentage'. It is inherited by classes 'A' and 'B' each having a method with the same name which **returns** the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Test your code

Task

1. Create Registration Class in which you would have variables as email, userName and password that have an access scope only within its own class. After creating an object of the class user should be able to call methods and in each method separately pass values to set users email, username and password.

   Requirements:

A. Valid email consider to be only gmail
B. Valid userName and password cannot be empty and should be of length larger than 6 characters. Also valid password cannot contain userName.

Task

1. Create a Class Car that would have the following fields: carPrice and color and method calculateSalePrice() which should be returning a price of the car.

   Create 2 sub classes: Sedan and Truck. The Truck class has field as weight and has its own implementation of calculateSalePrice() method in which returned price calculated as following: if weight>2000 then returned price car should include 10% discount, otherwise 20% discount.

   The Sedan class has field as length and also does it is own implementation of calculateSalePrice(): if length of sedan is >20 feet then returned car price should include 5% discount, otherwise 10% discount