Lab Task 3: Object Oriented Programming Concepts (Encapsulation)

OBJECTIVES:

i. To be familiar with encapsulation
ii. To be familiar with Get and Set methods

Problem:
Create a class called Car with properties such as make, model, year, color, price. Include getter and setter methods for each property (Encapsulation).

Title:
A Class Implementation for Car Properties and Methods with Encapsulation

Objective:
The objective of this Java program is to demonstrate and apply one of the primary concepts of object-oriented programming, encapsulation, by creating a car class with various properties, constructor and methods.

Introduction:
This report presents a class implementation in Java for a car object with encapsulation. The class called "Car" will have properties such as make, model, year, color, and price. Additionally, getter and setter methods will be provided for each property to ensure data encapsulation and controlled access.

Problem Understanding:
The goal is to design a class that represents a car object with various private properties. Encapsulation is essential to ensure that the properties are accessed and modified through controlled methods, providing data integrity and security. We need to use getter and setter methods for each property to enforce data validation and provide a consistent interface for accessing and updating the car properties.

Background Theory:
In Java, access modifiers, encapsulation, and getter/setter methods are key concepts in object-oriented programming that help achieve data hiding, encapsulation, and control over the accessibility of class members. Let's explore each of these concepts:

a. **Access Modifiers:**
   Access modifiers determine the accessibility of classes, variables, methods, and constructors within Java. There are four types of access modifiers in Java:
      i. Public: The public access modifier allows unrestricted access to a class member from any other class or package.
      ii. Private: The private access modifier restricts access to the member within the same class. It is the most restrictive modifier.
      iii. Protected: The protected access modifier allows access within the same class, derived/subclass, and package.
      iv. Default (no modifier): When no access modifier is specified, it is referred to as the default access modifier. It allows access within the same package only.
b. **Encapsulation:**
   Encapsulation is a mechanism that combines data and methods (or behaviors) within a class, hiding the internal details and providing controlled access to the class members. It helps in achieving data abstraction and protecting the data from unauthorized access and modifications.

To achieve encapsulation, one typically marks the class variables (fields) as private and provides public getter and setter methods to access and modify the data. This ensures that the internal state of the object is controlled and maintained consistently.

c. **Getter and Setter Methods:**
   Getter and setter methods (also known as accessors and mutators) are public methods used to retrieve (get) and modify (set) the values of private variables, respectively. They provide controlled access to the class fields while encapsulating the implementation details.

The naming convention for getter and setter methods is based on the field name, with "get" and "set" prefixes, respectively. For example, if there is a private field called "name," the corresponding getter and setter methods would be "getName()" and "setName(String name)".

Getter methods are used to access the value of a field, while setter methods are used to set or modify the value of a field. These methods can include additional logic, such as validation or calculations, before accessing or modifying the field.

By using getter and setter methods, one can enforce data validation rules, implement read-only or write-only properties, and provide a level of abstraction to the internal representation of data.

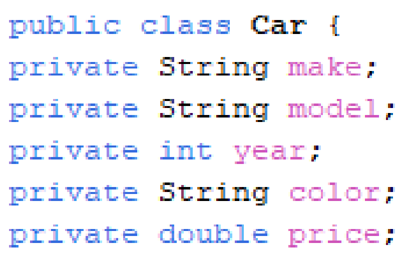
Overall, access modifiers, encapsulation, and getter/setter methods in Java enable a programmer to control the visibility and accessibility of class members, protect data integrity, and provide a standardized way to access and modify private variables while encapsulating the implementation details.

Algorithm Design:

1. Create a class called "Car" with the following properties:
    i. Make
    ii. Model
    iii. Year
    iv. Color
    v. price
2. Implement getter and setter methods for each property:
    i. For each property, create a getter method that returns the current value of the property.
    ii. For each property, create a setter method that takes a parameter and assigns the new value to the property.
3. Instantiate a Car object and set its properties using the user input:
    i. Create an instance of the Car class.
    ii. Use the setter methods to set the values of the car properties based on the user input.
4. Repeat step 3 for 2 more objects.
5. Display objects using getter methods.

Code:

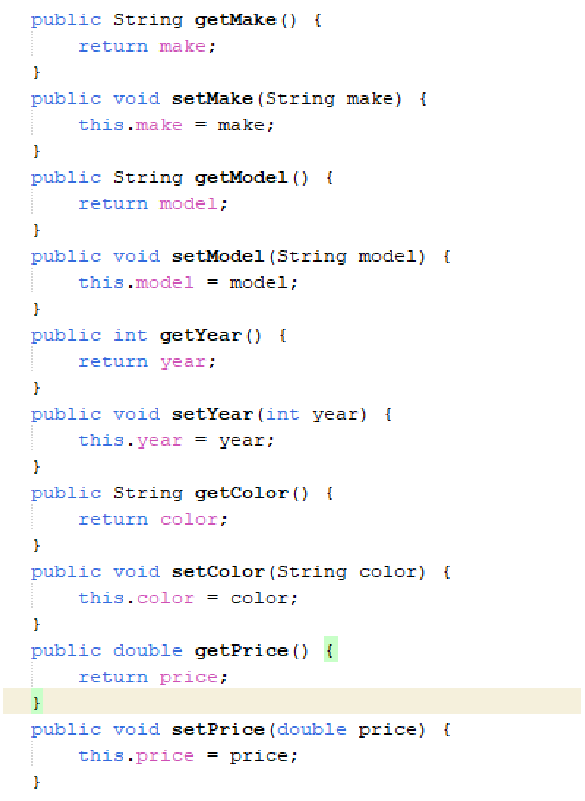
The Car class encapsulates the properties of a car using private access modifiers. This ensures that the properties cannot be accessed directly from outside the class, promoting encapsulation. Each property has a corresponding private field: make, model, year, color, and price.



```
public class Car {
private String make;
private String model;
private int year;
private String color;
private double price;
```

figure 1: Class Car

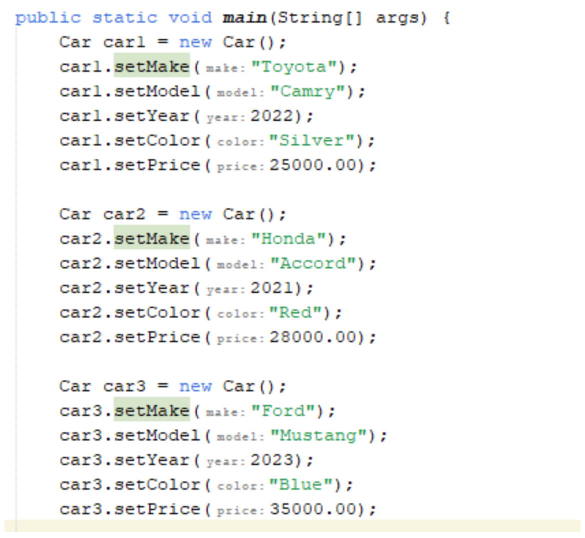
For each property, there are two methods: a getter method and a setter method.
The getter methods, such as getMake(), getModel(), getYear(), getColor(), and getPrice(), return the values of the respective properties.
The setter methods, such as setMake(String make), setModel(String model), setYear(int year), setColor(String color), and setPrice(double price), allow you to set the values of the respective properties.



```
public String getMake() {
    return make;
}
public void setMake(String make) {
    this.make = make;
}
public String getModel() {
    return model;
}
public void setModel(String model) {
    this.model = model;
}
public int getYear() {
    return year;
}
public void setYear(int year) {
    this.year = year;
}
public String getColor() {
    return color;
}
public void setColor(String color) {
    this.color = color;
}
public double getPrice() {
    return price;
}
public void setPrice(double price) {
    this.price = price;
}
```

figure 2: Getter and Setter methods

Three Car objects (car1, car2, and car3) are created and their properties are set using the setter methods inside the main method.



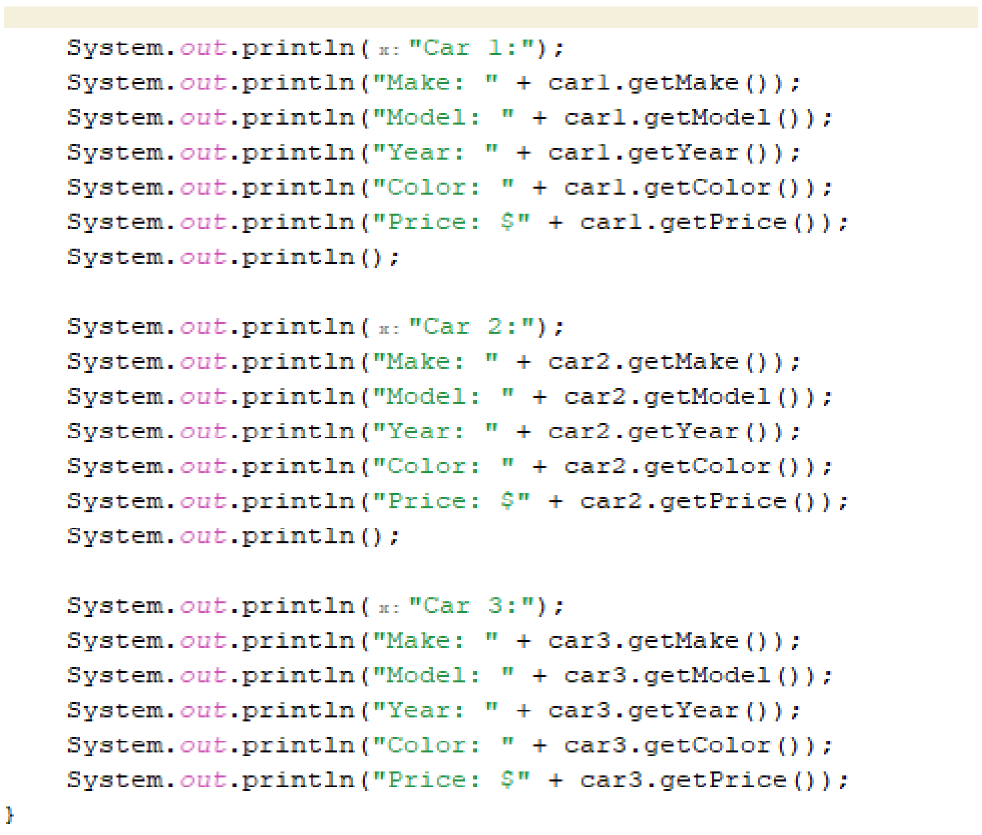
```
public static void main(String[] args) {
    Car car1 = new Car();
    car1.setMake( make: "Toyota");
    car1.setModel( model: "Camry");
    car1.setYear( year: 2022);
    car1.setColor( color: "Silver");
    car1.setPrice( price: 25000.00);

    Car car2 = new Car();
    car2.setMake( make: "Honda");
    car2.setModel( model: "Accord");
    car2.setYear( year: 2021);
    car2.setColor( color: "Red");
    car2.setPrice( price: 28000.00);

    Car car3 = new Car();
    car3.setMake( make: "Ford");
    car3.setModel( model: "Mustang");
    car3.setYear( year: 2023);
    car3.setColor( color: "Blue");
    car3.setPrice( price: 35000.00);
```

figure 3: Object creation using setter methods

The System.out.println statements are used to display the details of each car object, including make, model, year, color, and price using the getter methods inside the main method.



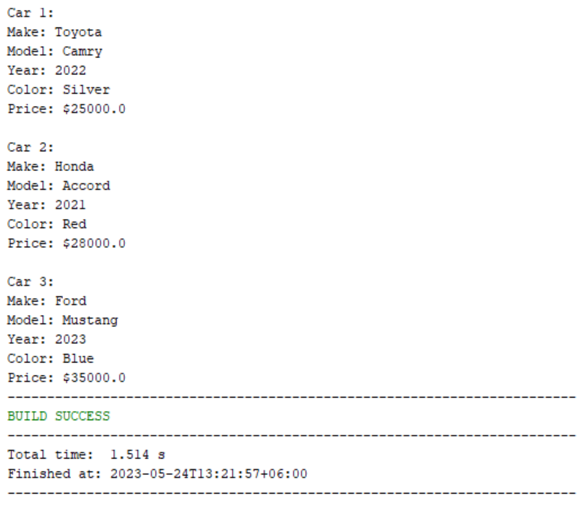
```
    System.out.println( x: "Car 1:");
    System.out.println("Make: " + car1.getMake());
    System.out.println("Model: " + car1.getModel());
    System.out.println("Year: " + car1.getYear());
    System.out.println("Color: " + car1.getColor());
    System.out.println("Price: $" + car1.getPrice());
    System.out.println();

    System.out.println( x: "Car 2:");
    System.out.println("Make: " + car2.getMake());
    System.out.println("Model: " + car2.getModel());
    System.out.println("Year: " + car2.getYear());
    System.out.println("Color: " + car2.getColor());
    System.out.println("Price: $" + car2.getPrice());
    System.out.println();

    System.out.println( x: "Car 3:");
    System.out.println("Make: " + car3.getMake());
    System.out.println("Model: " + car3.getModel());
    System.out.println("Year: " + car3.getYear());
    System.out.println("Color: " + car3.getColor());
    System.out.println("Price: $" + car3.getPrice());
}
```

figure 4: Object display using getter methods

The output of the program is shown below:



```
Car 1:
Make: Toyota
Model: Camry
Year: 2022
Color: Silver
Price: $25000.0

Car 2:
Make: Honda
Model: Accord
Year: 2021
Color: Red
Price: $28000.0

Car 3:
Make: Ford
Model: Mustang
Year: 2023
Color: Blue
Price: $35000.0
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.514 s
Finished at: 2023-05-24T13:21:57+06:00
------------------------------------------------------------------------
```

figure 5: Code output

Conclusion: In conclusion, we have successfully implemented a Java class named "Car" with encapsulation. The class provides properties such as make, model, year, color, and price, along with getter and setter methods for each property. The user's input is obtained through the Scanner class, and the car object's properties are set accordingly. Encapsulation ensures that the properties are accessed and modified using controlled methods, promoting data integrity and security.

Practice Problems

1. Create a class called Person with properties such as name, age, gender, address. Include getter and setter methods for each property.
2. Create a class called Employee with properties such as name, id, salary, designation. Include methods to get and set the properties.