CodInClub
Powered By BridgeLabz

Best Programming Practices

Encapsulation

- Use `private` access modifiers for class fields to restrict direct access.
- Provide `public` getter and setter methods to access and modify private fields.
- Implement validation logic in setters to ensure data integrity.
- Use `final` fields and avoid setters for immutable classes.
- Follow naming conventions for methods (e.g., `getX`, `setX`).

Polymorphism

- Program to an interface, not an implementation.
- Ensure overridden methods adhere to the base class method's contract.
- Avoid explicit casting; rely on polymorphic behavior.
- Leverage covariant return types for overriding methods.
- Keep inheritance hierarchies shallow to maintain simplicity.

Interfaces

- Use interfaces to define a contract or behavior.
- Prefer default methods only when backward compatibility or shared implementation is necessary.
- Combine interfaces to create modular, reusable behaviors.
- Favor composition over inheritance when combining multiple behaviors.

Abstract Classes

- Use abstract classes for shared state and functionality among related classes.
- Avoid overusing abstract classes; use them only when clear shared behavior exists.
- Combine abstract classes with interfaces to separate behavior and implementation.
- Avoid deep inheritance hierarchies; keep designs flexible and maintainable.

General Practices

- Follow Java naming conventions for classes, methods, and variables.
- Document code with comments and Javadoc to improve readability.
- Ensure consistency and readability by adhering to team or industry coding standards.
- Apply SOLID principles, particularly Single Responsibility and Interface Segregation. (We will learn it in coming days)

Tips for Implementation

- **Encapsulation**: Ensure all sensitive fields are private and accessed through well-defined getter and setter methods. Include validation logic where applicable.
- **Polymorphism**: Use abstract class references or interface references to handle objects of multiple types dynamically.
- **Abstract Classes**: Use them to define a common structure and behavior while deferring specific details to subclasses.
- **Interfaces**: Use them to define additional capabilities or contracts that are not tied to the class hierarchy.

Problem Statements

1. Employee Management System

- **Description**: Build an employee management system with the following requirements:
  - Use an abstract class `Employee` with fields like `employeeId`, `name`, and `baseSalary`.
  - Provide an abstract method `calculateSalary()` and a concrete method `displayDetails()`.
  - Create two subclasses: `FullTimeEmployee` and `PartTimeEmployee`, implementing `calculateSalary()` based on work hours or fixed salary.
  - Use encapsulation to restrict direct access to fields and provide getter and setter methods.
  - Create an interface `Department` with methods like `assignDepartment()` and `getDepartmentDetails()`.
  - Ensure polymorphism by processing a list of employees and displaying their details using the `Employee` reference.

```
abstract class Employee{
    private int employeeId;
    private String name;
    private int baseSalary;

    Employee(int employeeId, String name, int baseSalary){
        this.employeeId = employeeId;
        this.name = name;
        this.baseSalary = baseSalary;
    }

    abstract void calculateSalary();
```

```
    public void displayDetails(){
        System.out.println("Employee name is : "+name);
        System.out.println("Employee id is : "+employeeId);
        System.out.println("Employee base salary is : "+baseSalary);
    }

    void setBaseSalary(int salary){
        this.baseSalary = salary;
    }

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

    String getName(){
        return name;
    }

    int getBaseSalary(){
        return baseSalary;
    }

}
```

```
interface Department{

    public void assignDepartment(String newDepartment);

    public String getDepartmentDetails();

}
```

```
class FullTimeEmployee extends Employee implements Department{

    private int workHours;
    private String department;
```

```
    FullTimeEmployee(int employeeId, String name, int baseSalary, int
workHours, String department){
            super(employeeId, name, baseSalary);
            this.workHours = workHours;
            this.department = department;

    }

    void calculateSalary(){
        System.out.println("Salary of Employee "+getName()+" is :
"+(workHours*5*4*500));
    }

    void setWorkHours(int workHours){
        this.workHours = workHours;
    }

    public void assignDepartment(String newDepartment){
        this.department = newDepartment;
    }

    int getWorkHours(){
        return workHours;
    }

    public String getDepartmentDetails(){
        return this.department;
    }

}
```

```
class PartTimeEmployee extends Employee implements Department{

    private int workHours;
    private String department;

    PartTimeEmployee(int employeeId, String name, int baseSalary, int
workHours, String department){
            super(employeeId, name, baseSalary);
            this.workHours = workHours;
            this.department = department;
```

```
    }

    void calculateSalary(){
        System.out.println("Salary of Employee "+getName()+" is :
"+(workHours*5*4*500));
    }

    void setWorkHours(int workHours){
        this.workHours = workHours;
    }

    public void assignDepartment(String newDepartment){
        this.department = newDepartment;
    }

    int getWorkHours(){
        return workHours;
    }

    public String getDepartmentDetails(){
        return this.department;
    }

}
```

```
import java.util.*;
class Main{
    public static void main(String[] args){
        Employee fe1 = new FullTimeEmployee(5433, "Kushagra Sharma", 80000, 5,
"Software Developer");

        Employee pe1 = new PartTimeEmployee(2436, "Naman Agarwal", 75000, 5,
"FullStack Developer");

        List<Employee> l1 = new ArrayList<>();

        l1.add(fe1);
        l1.add(pe1);

        for(int i=0;i<l1.size();i++){
```

```
            l1.get(i).displayDetails();

            if(l1.get(i) instanceof FullTimeEmployee){


System.out.println(((FullTimeEmployee)l1.get(i)).getWorkHours());

System.out.println(((FullTimeEmployee)l1.get(i)).getBaseSalary());

                ((FullTimeEmployee)l1.get(i)).setWorkHours(6);
                ((FullTimeEmployee)l1.get(i)).setBaseSalary(76000);


System.out.println(((FullTimeEmployee)l1.get(i)).getWorkHours());

System.out.println(((FullTimeEmployee)l1.get(i)).getBaseSalary());

                ((FullTimeEmployee)l1.get(i)).assignDepartment("Artificial
Intelligence");


System.out.println(((FullTimeEmployee)l1.get(i)).getDepartmentDetails());
            }
            if(l1.get(i) instanceof PartTimeEmployee){


System.out.println(((PartTimeEmployee)l1.get(i)).getWorkHours());

System.out.println(((PartTimeEmployee)l1.get(i)).getBaseSalary());

                ((PartTimeEmployee)l1.get(i)).setWorkHours(6);
                ((PartTimeEmployee)l1.get(i)).setBaseSalary(76000);


System.out.println(((PartTimeEmployee)l1.get(i)).getWorkHours());

System.out.println(((PartTimeEmployee)l1.get(i)).getBaseSalary());

                ((PartTimeEmployee)l1.get(i)).assignDepartment("Artificial
Intelligence");


System.out.println(((PartTimeEmployee)l1.get(i)).getDepartmentDetails());
```

```
            }
        }

    }
}
```

---

2. E-Commerce Platform

- **Description**: Develop a simplified e-commerce platform:
    - Create an abstract class `Product` with fields like `productId`, `name`, and `price`, and an abstract method `calculateDiscount()`.
    - Extend it into concrete classes: `Electronics`, `Clothing`, and `Groceries`.
    - Implement an interface `Taxable` with methods `calculateTax()` and `getTaxDetails()` for applicable product categories.
    - Use encapsulation to protect product details, allowing updates only through setter methods.
    - Showcase polymorphism by creating a method that calculates and prints the final price (price + tax - discount) for a list of `Product`.

```
abstract class Product{
    private int productId;
    private String name;
    private int price;

    Product(int productId, String name, int price){
            this.productId = productId;
            this.name = name;
            this.price = price;

    }

    abstract void calculateDiscount();

    public void displayDetails(){
        System.out.println("Product name is : "+name);
        System.out.println("Product id is : "+productId);
        System.out.println("Product price is :"+price);
```

```
    }

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

    void setPrice(int price){
        this.price = price;
    }

    String getName(){
        return name;
    }

    int getPrice(){
        return price;
    }

    int getProductId(){
        return productId;
    }

}
```

```
interface Taxable{

    public void calculateTax();
    public void getTaxDetails();



}
```

```
class Clothing extends Product implements Taxable{

    private double tax;
    private double discount;
```

```
    Clothing(int productId, String name, int price){
        super(productId, name, price);
    }

    public void calculateDiscount(){
        discount = 0.2*getPrice();
    }

    public void calculateTax(){
        tax = 0.1*getPrice();
    }

    public void getTaxDetails(){
        System.out.println("Tax on the Product is : "+tax);
    }

    public void getDiscountDetails(){
        System.out.println("Discount on the Product is : "+discount);
    }

    public void showFinalPrice(){
        System.out.println("Final Price of the Product is :
"+(getPrice()+tax-discount));
    }

}
```

```
class Electronics extends Product implements Taxable{

    private double tax;
    private double discount;

    Electronics(int productId, String name, int price){
        super(productId, name, price);
    }

    public void calculateDiscount(){
        discount = 0.2*getPrice();
    }

    public void calculateTax(){
```

```
        tax = 0.1*getPrice();
    }

    public void getTaxDetails(){
        System.out.println("Tax on the Product is : "+tax);
    }

    public void getDiscountDetails(){
        System.out.println("Discount on the Product is : "+discount);
    }

    public void showFinalPrice(){
        System.out.println("Final Price of the Product is :
"+(getPrice()+tax-discount));
    }

}
```

```
class Groceries extends Product implements Taxable{

    private double tax;
    private double discount;

    Groceries(int productId, String name, int price){
        super(productId, name, price);
    }

    public void calculateDiscount(){
        discount = 0.2*getPrice();
    }

    public void calculateTax(){
        tax = 0.1*getPrice();
    }

    public void getTaxDetails(){
        System.out.println("Tax on the Product is : "+tax);
    }

    public void getDiscountDetails(){
        System.out.println("Discount on the Product is : "+discount);
```

```
    }

    public void showFinalPrice(){
        System.out.println("Final Price of the Product is :
"+(getPrice()+tax-discount));
    }

}
```

```
import java.util.*;
class Main{
    public static void main(String[] args){
        Product ec1 = new Electronics(4235, "Controller", 1000);

        Product c1 = new Clothing(2352, "T-Shirt", 500);

        Product g1 = new Groceries(6168, "Cup Set", 1500);

        List<Product> l1 = new ArrayList<>();

        l1.add(ec1);
        l1.add(c1);
        l1.add(g1);

        for(int i=0;i<l1.size();i++){
            l1.get(i).displayDetails();

            if(l1.get(i) instanceof Electronics){
                ((Electronics)l1.get(i)).calculateTax();
                ((Electronics)l1.get(i)).getTaxDetails();
                ((Electronics)l1.get(i)).calculateDiscount();
                ((Electronics)l1.get(i)).getDiscountDetails();
                ((Electronics)l1.get(i)).showFinalPrice();
            }
            if(l1.get(i) instanceof Clothing){
                ((Clothing)l1.get(i)).calculateTax();
                ((Clothing)l1.get(i)).getTaxDetails();
                ((Clothing)l1.get(i)).calculateDiscount();
                ((Clothing)l1.get(i)).getDiscountDetails();
                ((Clothing)l1.get(i)).showFinalPrice();
            }
```

```
            if(l1.get(i) instanceof Groceries){
                ((Groceries)l1.get(i)).calculateTax();
                ((Groceries)l1.get(i)).getTaxDetails();
                ((Groceries)l1.get(i)).calculateDiscount();
                ((Groceries)l1.get(i)).getDiscountDetails();
                ((Groceries)l1.get(i)).showFinalPrice();
            }

        }

    }

}
```

---

3. Vehicle Rental System

- **Description**: Design a system to manage vehicle rentals:
  - Define an abstract class `Vehicle` with fields like `vehicleNumber`, `type`, and `rentalRate`.
  - Add an abstract method `calculateRentalCost(int days)`.
  - Create subclasses `Car`, `Bike`, and `Truck` with specific implementations of `calculateRentalCost()`.
  - Use an interface `Insurable` with methods `calculateInsurance()` and `getInsuranceDetails()`.
  - Apply encapsulation to restrict access to sensitive details like insurance policy numbers.
  - Demonstrate polymorphism by iterating over a list of vehicles and calculating rental and insurance costs for each.

```
abstract class Vehicle{
    private String vehicleNumber;
    private String type;
    private int rentalRate;

    Vehicle(String vehicleNumber, String type, int rentalRate){
        this.vehicleNumber = vehicleNumber;
        this.type = type;
        this.rentalRate = rentalRate;
```

```
    }

    abstract void calculateRentalCost(int days);

    public void displayDetails(){
        System.out.println("Vehicle Number is : "+vehicleNumber);
        System.out.println("Vehicle Type is : "+type);
        System.out.println("Vehicle rental rate is : "+rentalRate);
    }

    void setVehicleNumber(String vehicleNumber){
        this.vehicleNumber = vehicleNumber;
    }

    void setType(String type){
        this.type = type;
    }

    void setRentalRate(int rentalRate){
        this.rentalRate = rentalRate;
    }

    String getVehicleNumber(){
        return vehicleNumber;
    }

    String getType(){
        return type;
    }

    int getRentalRate(){
        return rentalRate;
    }

}
```

```
interface Insurable{

    public void calculateInsurance();

    public void getInsuranceDetails();
```

```
}
```

```
class Bike extends Vehicle implements Insurable{

    private int rent;
    private int insurance;

    Bike(String vehicleNumber, String type, int rentalRate){
        super(vehicleNumber, type, rentalRate);
    }

    public void displayDetails(){
        System.out.println("Bike Number is : "+getVehicleNumber());
        System.out.println("Bike Type is : "+getType());
        System.out.println("Bike rental rate is : "+getRentalRate());
    }

    public void calculateRentalCost(int days){
        setRent(days*getRentalRate());
    }

    public void calculateInsurance(){
        setInsurance(getRentalRate()*250);
    }

    public void getRentalCost(){
        System.out.println("Rental Cost for this bike is : "+getRent());
    }

    public void getInsuranceDetails(){
        System.out.println("Insurance amount is : "+getInsurance());
    }

    void setRent(int rent){
        this.rent = rent;
    }

    void setInsurance(int insurance){
        this.insurance = insurance;
    }
```

```
    int getRent(){
        return this.rent;
    }

    int getInsurance(){
        return this.insurance;
    }

}
```

```
class Car extends Vehicle implements Insurable{

    private int rent;
    private int insurance;

    Car(String vehicleNumber, String type, int rentalRate){
        super(vehicleNumber, type, rentalRate);
    }

    public void displayDetails(){
        System.out.println("Car Number is : "+getVehicleNumber());
        System.out.println("Car Type is : "+getType());
        System.out.println("Car rental rate is : "+getRentalRate());
    }

    public void calculateRentalCost(int days){
        setRent(days*getRentalRate());
    }

    public void calculateInsurance(){
        setInsurance(getRentalRate()*500);
    }

    public void getRentalCost(){
        System.out.println("Rental Cost for this car is : "+getRent());
    }

    public void getInsuranceDetails(){
        System.out.println("Insurance amount is : "+getInsurance());
    }
```

```
    void setRent(int rent){
        this.rent = rent;
    }

    void setInsurance(int insurance){
        this.insurance = insurance;
    }

    int getRent(){
        return this.rent;
    }

    public int getInsurance(){
        return this.insurance;
    }

}
```

```
class Truck extends Vehicle implements Insurable{

    private int rent;
    private int insurance;

    Truck(String vehicleNumber, String type, int rentalRate){
        super(vehicleNumber, type, rentalRate);
    }

    public void displayDetails(){
        System.out.println("Truck Number is : "+getVehicleNumber());
        System.out.println("Truck Type is : "+getType());
        System.out.println("Truck rental rate is : "+getRentalRate());
    }

    public void calculateRentalCost(int days){
        setRent(days*getRentalRate());
    }

    public void calculateInsurance(){
        setInsurance(getRentalRate()*400);
    }
```

```
    public void getRentalCost(){
        System.out.println("Rental Cost for this truck is : "+getRent());
    }

    public void getInsuranceDetails(){
        System.out.println("Insurance amount is : "+getInsurance());
    }

    void setRent(int rent){
        this.rent = rent;
    }

    void setInsurance(int insurance){
        this.insurance = insurance;
    }

    int getRent(){
        return this.rent;
    }

    int getInsurance(){
        return this.insurance;
    }

}
```

```
import java.util.*;
class Main{
    public static void main(String[] args){
        Vehicle c1 = new Car("UP80DC2315","Hybrid", 500);

        Vehicle b1 = new Bike("HR50BR4526", "Electirc", 250);

        Vehicle t1 = new Truck("DL60HG5829", "Fuel Cell", 400);

        List<Vehicle> l1 = new ArrayList<>();

        l1.add(c1);
        l1.add(b1);
        l1.add(t1);
```

```
        for(int i=0;i<l1.size();i++){
            l1.get(i).displayDetails();

            if(l1.get(i) instanceof Car){
                ((Car)l1.get(i)).calculateRentalCost(5);
                ((Car)l1.get(i)).calculateInsurance();
                ((Car)l1.get(i)).getRentalCost();
                ((Car)l1.get(i)).getInsuranceDetails();
            }
            if(l1.get(i) instanceof Bike){
                ((Bike)l1.get(i)).calculateRentalCost(5);
                ((Bike)l1.get(i)).calculateInsurance();
                ((Bike)l1.get(i)).getRentalCost();
                ((Bike)l1.get(i)).getInsuranceDetails();
            }
            if(l1.get(i) instanceof Truck){
                ((Truck)l1.get(i)).calculateRentalCost(5);
                ((Truck)l1.get(i)).calculateInsurance();
                ((Truck)l1.get(i)).getRentalCost();
                ((Truck)l1.get(i)).getInsuranceDetails();
            }
        }


    }

}
```

---

4. Banking System

- **Description**: Create a banking system with different account types:
  - Define an abstract class `BankAccount` with fields like `accountNumber`, `holderName`, and `balance`.
  - Add methods like `deposit(double amount)` and `withdraw(double amount)` (concrete) and `calculateInterest()` (abstract).
  - Implement subclasses `SavingsAccount` and `CurrentAccount` with unique interest calculations.

- Create an interface `Loanable` with methods `applyForLoan()` and `calculateLoanEligibility()`.
- Use encapsulation to secure account details and restrict unauthorized access.
- Demonstrate polymorphism by processing different account types and calculating interest dynamically.

```
abstract class BankAccount{
    private int accountNumber;
    private String holderName;
    private double balance;

    BankAccount(int accountNumber, String holderName, double balance){
        this.accountNumber = accountNumber;
        this.holderName = holderName;
        this.balance = balance;
    }

    public void deposit(double amount){
        if(amount < 0)
            System.out.println("Invalid Amount");
        else
            this.balance += amount;
    }

    public void withdraw(double amount){
        if(amount > balance)
            System.out.println("Invalid Amount");
        else
            this.balance -= amount;
    }

    abstract void calculateInterest();

    void setHolderName(String name){
        this.holderName = name;
    }

    int getAccountNumber(){
        return this.accountNumber;
    }

    String getHolderName(){
        return this.holderName;
```

```
    }

    double getBalance(){
        return this.balance;
    }

    public void displayDetails(){
        System.out.println("Account number is : "+this.accountNumber);
        System.out.println("Account Holder Name is : "+this.holderName);
        System.out.println("Account balance is : "+this.balance);

    }
}
```

```
interface Loanable{
    public void applyForLoan(int amount);

    public boolean calculateLoanEligibility(int amount);
}
```

```
class CurrentAccount extends BankAccount implements Loanable{

    CurrentAccount(int accountNumber, String holderName, double balance){
        super(accountNumber, holderName, balance);
    }

    public void calculateInterest(){
        System.out.println("Interest amount for this bank account is :
"+(0.025*getBalance()));
    }

    public void applyForLoan(int amount){
        if(calculateLoanEligibility(amount))
            System.out.println("Applied for Loan");
        else
            System.out.println("Do not meet the Loan Eligibility criteria");
    }
```

```
    public boolean calculateLoanEligibility(int amount){
        if(amount <= 3*getBalance())
            return true;
        else
            return false;
    }
}
```

```
class SavingsAccount extends BankAccount implements Loanable{

    SavingsAccount(int accountNumber, String holderName, double balance){
        super(accountNumber, holderName, balance);
    }

    public void calculateInterest(){
        System.out.println("Interest amount for this bank account is :
"+(0.05*getBalance()));
    }

    public void applyForLoan(int amount){
        if(calculateLoanEligibility(amount))
            System.out.println("Applied for Loan");
        else
            System.out.println("Do not meet the Loan Eligibility criteria");
    }

    public boolean calculateLoanEligibility(int amount){
        if(amount <= 4*getBalance())
            return true;
        else
            return false;
    }
}
```

```
import java.util.*;
class Main{
    public static void main(String[] args){
        BankAccount s1 = new SavingsAccount(23892229, "Kushagra Sharma",
```

```
10000000);

        BankAccount c1 = new CurrentAccount(84720019, "Naman Agarwal",
5000000);

        List<BankAccount> l1 = new ArrayList<>();
        l1.add(s1);
        l1.add(c1);

        for(int i=0;i<l1.size();i++){
            l1.get(i).displayDetails();

            if(l1.get(i) instanceof SavingsAccount){
                ((SavingsAccount)l1.get(i)).calculateInterest();
                ((SavingsAccount)l1.get(i)).applyForLoan(30000000);

            }
            if(l1.get(i) instanceof CurrentAccount){
                ((CurrentAccount)l1.get(i)).calculateInterest();
                ((CurrentAccount)l1.get(i)).applyForLoan(1000000000);

            }
            l1.get(i).displayDetails();


        }

    }


}
```

---

5. Library Management System

- **Description**: Develop a library management system:
    - Use an abstract class `LibraryItem` with fields like `itemId`, `title`, and `author`.

- Add an abstract method `getLoanDuration()` and a concrete method `getItemDetails()`.
- Create subclasses `Book`, `Magazine`, and `DVD`, overriding `getLoanDuration()` with specific logic.
- Implement an interface `Reservable` with methods `reserveItem()` and `checkAvailability()`.
- Apply encapsulation to secure details like the borrower's personal data.
- Use polymorphism to allow a general `LibraryItem` reference to manage all items, regardless of type.

```
abstract class LibraryItem{
    private int itemId;
    private String title;
    private String author;

    LibraryItem(int itemId, String title, String author){
        this.itemId = itemId;
        this.title = title;
        this.author = author;
    }

    abstract String getLoanDuration();

    public void getItemDetails(){
        System.out.println("Item Id is : "+itemId);
        System.out.println("Title of this LibraryItem is : "+title);
        System.out.println("Author of this LibraryItem is : "+author);
    }

    void setTitle(String title){
        this.title = title;
    }

    void setAuthor(String author){
        this.author = author;
    }

    int getItemId(){
        return this.itemId;
    }

    String getTitle(){
```

```
        return this.title;
    }

    String getAuthor(){
        return this.author;
    }

}
```

```
interface Reservable{

    public void reserveItem();

    public boolean checkAvailability();

}
```

```
class Book extends LibraryItem implements Reservable{
    private boolean reserved = true;

    Book(int itemId, String title, String author){
        super(itemId, title, author);
    }

    public String getLoanDuration(){
            return "3 days";
    }

    public void reserveItem(){
        if(checkAvailability()){
            System.out.println("Item reserved for "+getLoanDuration());
            this.reserved = false;
        }
        else
            System.out.println("Item is not Available yet");
    }

    public boolean checkAvailability(){
```

```
        return this.reserved;
    }

    public void getItemDetails(){
        System.out.println("Book Id is : "+getItemId());
        System.out.println("Title of this Book is : "+getTitle());
        System.out.println("Author of this Book is : "+getAuthor());
    }

}
class DVD extends LibraryItem implements Reservable{

    private boolean reserved = true;

    DVD(int itemId, String title, String author){
        super(itemId, title, author);
    }

    public String getLoanDuration(){
            return "8 days";
    }

    public void reserveItem(){
        if(checkAvailability()){
            System.out.println("Item reserved for "+getLoanDuration());
            this.reserved = false;
        }
        else
            System.out.println("Item is not Available yet");
    }

    public boolean checkAvailability(){
        return this.reserved;
    }

    public void getItemDetails(){
        System.out.println("DVD Id is : "+getItemId());
        System.out.println("Title of this DVD is : "+getTitle());
        System.out.println("Author of this DVD is : "+getAuthor());
    }

}
```

```
class Magazine extends LibraryItem implements Reservable{
    private boolean reserved = true;

    Magazine(int itemId, String title, String author){
        super(itemId, title, author);
    }

    public String getLoanDuration(){
            return "7 days";
    }

    public void reserveItem(){
        if(checkAvailability()){
            System.out.println("Item reserved for "+getLoanDuration());
            this.reserved = false;
        }
        else
            System.out.println("Item is not Available yet");
    }

    public boolean checkAvailability(){
        return this.reserved;
    }

    public void getItemDetails(){
        System.out.println("Magazine Id is : "+getItemId());
        System.out.println("Title of this Magazine is : "+getTitle());
        System.out.println("Author of this Magazine is : "+getAuthor());
    }


}
```

```
import java.util.*;
class Main{
    public static void main(String[] args){
        LibraryItem b1 = new Book(2839, "Lord Of The Rings", "John Ronald");

        LibraryItem m1 = new Magazine(8593, "New York Fashion", "Mark
```

```
Clifton");

        LibraryItem d1 = new DVD(4891, "The Man Of Steel", "Howard Christan");

        List<LibraryItem> l1 = new ArrayList<>();

        l1.add(b1);
        l1.add(m1);
        l1.add(d1);

        for(int i=0;i<l1.size();i++){

            if(l1.get(i) instanceof Book){
                ((Book)l1.get(i)).getItemDetails();
                ((Book)l1.get(i)).getLoanDuration();
                ((Book)l1.get(i)).reserveItem();
            }
            if(l1.get(i) instanceof Magazine){
                ((Magazine)l1.get(i)).getItemDetails();
                ((Magazine)l1.get(i)).getLoanDuration();
                ((Magazine)l1.get(i)).reserveItem();
            }
            if(l1.get(i) instanceof DVD){
                ((DVD)l1.get(i)).getItemDetails();
                ((DVD)l1.get(i)).getLoanDuration();
                ((DVD)l1.get(i)).reserveItem();
            }
        }
    }

}
```

---

6. Online Food Delivery System

- **Description**: Create an online food delivery system:
    - Define an abstract class `FoodItem` with fields like `itemName`, `price`, and `quantity`.
    - Add abstract methods `calculateTotalPrice()` and concrete methods like `getItemDetails()`.

- Extend it into classes `VegItem` and `NonVegItem`, overriding `calculateTotalPrice()` to include additional charges (e.g., for non-veg items).
- Use an interface `Discountable` with methods `applyDiscount()` and `getDiscountDetails()`.
- Demonstrate encapsulation to restrict modifications to order details and use polymorphism to handle different types of food items in a single order-processing method.

```
abstract class FoodItem{
    private String itemName;
    private int price;
    private int quantity;

    FoodItem(String itemName, int price, int quantity){
        this.itemName = itemName;
        this.price = price;
        this.quantity = quantity;
    }

    abstract void calculateTotalPrice();

    public void getItemDetails(){
        System.out.println("Food item name is : "+this.itemName);
        System.out.println("Food item price is : "+this.price);
        System.out.println("Food item quantity is : "+this.quantity);

    }

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

    public void setPrice(int price){
        this.price = price;
    }

    public void setQuantity(int quantity){
        this.quantity = quantity;
    }

    public String getItemName(){
```

```
        return this.itemName;
    }

    public int getPrice(){
        return this.price;
    }

    public int getQuantity(){
        return this.quantity;
    }


}
```

```
interface Discountable{

    public void applyDiscount();

    public void getDiscountDetails();

}
```

```
class NonVegItem extends FoodItem implements Discountable{

    double discount;

    NonVegItem(String itemName, int price, int quantity){
        super(itemName, price, quantity);
    }

    public void calculateTotalPrice(){
        System.out.println("Total Price is :
"+(getPrice()-discount+0.05*getPrice()));

    }

    public void applyDiscount(){
        this.discount = 0.10*getPrice();
```

```
        System.out.println("Discount Applied");
    }

    public void getDiscountDetails(){
        System.out.println("Discount applied on this dish is :
"+this.discount);
    }

    public void getItemDetails(){
        System.out.println("Non Veg Item name is : "+getItemName());
        System.out.println("Non Veg Item price is : "+getPrice());
        System.out.println("Non Veg Item quantity is : "+getQuantity());

    }
}
```

```
class VegItem extends FoodItem implements Discountable{

    double discount;

    VegItem(String itemName, int price, int quantity){
        super(itemName, price, quantity);
    }

    public void calculateTotalPrice(){
        System.out.println("Total Price is : "+(getPrice()-discount));

    }

    public void applyDiscount(){
        this.discount = 0.20*getPrice();
        System.out.println("Discount Applied");
    }

    public void getDiscountDetails(){
        System.out.println("Discount applied on this dish is :
"+this.discount);
    }

    public void getItemDetails(){
        System.out.println("Veg Item name is : "+getItemName());
```

```
        System.out.println("Veg Item price is : "+getPrice());
        System.out.println("Veg Item quantity is : "+getQuantity());

    }
}
```

```
import java.util.*;
class Main{
    public static void main(String[] args){
        FoodItem vg1 = new VegItem("Matar Paneer", 250, 450);
        FoodItem nvg1 = new NonVegItem("Butter Chicken", 500, 500);

        List<FoodItem> l1 = new ArrayList<>();

        l1.add(vg1);
        l1.add(nvg1);

        for(int i=0;i<l1.size();i++){
            if(l1.get(i) instanceof VegItem){

                ((VegItem)l1.get(i)).getItemDetails();
                ((VegItem)l1.get(i)).applyDiscount();
                ((VegItem)l1.get(i)).getDiscountDetails();
                ((VegItem)l1.get(i)).calculateTotalPrice();
            }
            if(l1.get(i) instanceof NonVegItem){

                ((NonVegItem)l1.get(i)).getItemDetails();
                ((NonVegItem)l1.get(i)).applyDiscount();
                ((NonVegItem)l1.get(i)).getDiscountDetails();
                ((NonVegItem)l1.get(i)).calculateTotalPrice();
            }

        }

    }

}
```

7. Hospital Patient Management

- **Description**: Design a system to manage patients in a hospital:
  - Create an abstract class `Patient` with fields like `patientId`, `name`, and `age`.
  - Add an abstract method `calculateBill()` and a concrete method `getPatientDetails()`.
  - Extend it into subclasses `InPatient` and `OutPatient`, implementing `calculateBill()` with different billing logic.
  - Implement an interface `MedicalRecord` with methods `addRecord()` and `viewRecords()`.
  - Use encapsulation to protect sensitive patient data like diagnosis and medical history.
  - Use polymorphism to handle different patient types and display their billing details dynamically.

```
abstract class Patient{
    private int patientId;
    private String name;
    private int age;

    Patient(int patientId, String name, int age){
        this.patientId = patientId;
        this.name = name;
        this.age = age;
    }

    abstract public void calculateBill(int days);

    public void getPatientDetails(){
        System.out.println("Patient Name is : "+this.name);
        System.out.println("Patient id is : "+this.patientId);
        System.out.println("patient age is : "+this.age);

    }

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

    public void setAge(int age){
```

```
        this.age = age;
    }

    public int getPatientId(){
        return this.patientId;
    }

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

    public int getAge(){
        return this.age;
    }

}
```

```
interface MedicalRecord{

    public void addRecord();

    public void viewRecords();

}
```

```
class InPatient extends Patient implements MedicalRecord{

    private double bill;
    private String disease;

    InPatient(int patientId, String name, int age){
        super(patientId, name, age);
    }

    public void calculateBill(int days){
        bill = days*1500;
    }
```

```
    public void viewBill(){
        System.out.println("Bill generated is : "+this.bill);
    }

    public void addRecord(String disease){
        this.disease = disease;
        System.out.println("Record of the InPatient has been added");
    }

    public void viewRecords(){
        System.out.println("Patient Name is : "+getName());
        System.out.println("Patient id is : "+getPatientId);
        System.out.println("Patient age is : "+getAge());
        System.out.println("Patient disease is :"+this.disease);
    }
}
```

```
class OutPatient extends Patient implements MedicalRecord{

    private double bill;
    private String disease;

    OutPatient(int patientId, String name, int age){
        super(patientId, name, age);
    }

    public void calculateBill(int days){
        bill = days*1000;
    }

    public void viewBill(){
        System.out.println("Bill generated is : "+this.bill);
    }

    public void addRecord(String disease){
        this.disease = disease;
        System.out.println("Record of the InPatient has been added");
    }

    public void viewRecords(){
        System.out.println("Patient Name is : "+getName());
```

```
        System.out.println("Patient id is : "+getPatientId);
        System.out.println("Patient age is : "+getAge());
        System.out.println("Patient disease is :"+this.disease);
    }
}
```

```
import java.util.*;
class Main{
    Patient ip1 = new InPatient(4322, "Vedansh Gautam", 23);

    Patient op1 = new OutPatient(3232, "Nikhil Agrawal", 21);

    List<Patient> l1 = new ArrayList<>();

    l1.add(ip1);
    l1.add(op1);

    for(int i=0;i<l1.size();i++){
        if(l1.get(i) instanceof InPatient){
            ((InPatient)l1.get(i)).calculateBill();
            ((InPatient)l1.get(i)).viewBill();
            ((InPatient)l1.get(i)).addRecord("Alzhiemer");
            ((InPatient)l1.get(i)).viewRecors();

        }
        if(l1.get(i) instanceof OutPatient){
            ((OutPatient)l1.get(i)).calculateBill();
            ((OutPatient)l1.get(i)).viewBill();
            ((OutPatient)l1.get(i)).addRecord("Alzhiemer");
            ((OutPatient)l1.get(i)).viewRecors();

        }

    }
}
```

8. Ride-Hailing Application

- **Description**: Develop a ride-hailing application:
  - Define an abstract class `Vehicle` with fields like `vehicleId`, `driverName`, and `ratePerKm`.
  - Add abstract methods `calculateFare(double distance)` and a concrete method `getVehicleDetails()`.
  - Create subclasses `Car`, `Bike`, and `Auto`, overriding `calculateFare()` based on type-specific rates.
  - Use an interface `GPS` with methods `getCurrentLocation()` and `updateLocation()`.
  - Secure driver and vehicle details using encapsulation.
  - Demonstrate polymorphism by creating a method to calculate fares for different vehicle types dynamically.

```
abstract class Vehicle {
    private String vehicleId;
    private String driverName;
    private double ratePerKm;

    public Vehicle(String vehicleId, String driverName, double ratePerKm) {
        this.vehicleId = vehicleId;
        this.driverName = driverName;
        this.ratePerKm = ratePerKm;
    }

    public abstract double calculateFare(double distance);

    public String getVehicleDetails() {
        return "Vehicle ID: " + vehicleId + ", Driver: " + driverName + ",
Rate/km: " + ratePerKm;
    }

    public String getVehicleId() {
        return vehicleId;
    }

    public String getDriverName() {
        return driverName;
    }

    public double getRatePerKm() {
```

```
        return ratePerKm;
    }
}
```

```
interface GPS {
    String getCurrentLocation();
    void updateLocation(String newLocation);
}
```

```
class Auto extends Vehicle implements GPS {
    private String location;

    public Auto(String vehicleId, String driverName, double ratePerKm) {
        super(vehicleId, driverName, ratePerKm);
        this.location = "Unknown";
    }

    public double calculateFare(double distance) {
        return distance * getRatePerKm();
    }

    public String getCurrentLocation() {
        return "Auto Location: " + location;
    }

    public void updateLocation(String newLocation) {
        this.location = newLocation;
    }
}
```

```
class Bike extends Vehicle implements GPS {
    private String location;

    public Bike(String vehicleId, String driverName, double ratePerKm) {
        super(vehicleId, driverName, ratePerKm);
        this.location = "Unknown";
    }
```

```
    public double calculateFare(double distance) {
        return distance * getRatePerKm();
    }

    public String getCurrentLocation() {
        return "Bike Location: " + location;
    }

    public void updateLocation(String newLocation) {
        this.location = newLocation;
    }
}
```

```
class Car extends Vehicle implements GPS {
    private String location;

    public Car(String vehicleId, String driverName, double ratePerKm) {
        super(vehicleId, driverName, ratePerKm);
        this.location = "Unknown";
    }

    public double calculateFare(double distance) {
        return distance * getRatePerKm();
    }

    public String getCurrentLocation() {
        return "Car Location: " + location;
    }

    public void updateLocation(String newLocation) {
        this.location = newLocation;
    }
}
```

```
import java.util.*;

class Main {
    public static void main(String[] args) {
        Vehicle car1 = new Car("C101", "John Doe", 10.5);
        Vehicle bike1 = new Bike("B202", "Alice Smith", 5.0);
```

```
        Vehicle auto1 = new Auto("A303", "Bob Johnson", 8.0);

        List<Vehicle> vehicles = new ArrayList<>();
        vehicles.add(car1);
        vehicles.add(bike1);
        vehicles.add(auto1);

        double distance = 12.5;

        for (Vehicle vehicle : vehicles) {
            System.out.println(vehicle.getVehicleDetails());
            System.out.println("Fare for " + distance + " km: $" +
vehicle.calculateFare(distance));


            if (vehicle instanceof GPS) {
                GPS gpsVehicle = (GPS) vehicle;
                System.out.println(gpsVehicle.getCurrentLocation());
                gpsVehicle.updateLocation("Downtown");
                System.out.println("Updated: " +
gpsVehicle.getCurrentLocation());
            }

        }
    }
}
```