1. Library Management System

This program allows users to manage books, borrowers, and transactions in a library. It includes classes for `Book`, `Member`, and `Library`.

Concepts Used: Classes, Encapsulation, Inheritance, Polymorphism

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

class Book {
    private String title;
    private String author;
    private boolean isAvailable;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.isAvailable = true;
    }

    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public boolean isAvailable() {
        return isAvailable;
    }

    public void borrowBook() {
        if (isAvailable) {
            isAvailable = false;
            System.out.println("You've borrowed " + title);
        } else {
            System.out.println(title + " is currently not available.");
        }
    }

    public void returnBook() {
        isAvailable = true;
        System.out.println("You've returned " + title);
    }
}

class Member {
    private String name;
    private List<Book> borrowedBooks;

    public Member(String name) {
        this.name = name;
        this.borrowedBooks = new ArrayList<>();
    }

    public void borrowBook(Book book) {
        if (book.isAvailable()) {
            book.borrowBook();
            borrowedBooks.add(book);
        } else {
            System.out.println(book.getTitle() + " is unavailable.");
        }
    }

    public void returnBook(Book book) {
        if (borrowedBooks.remove(book)) {
            book.returnBook();
        } else {
            System.out.println("You did not borrow this book.");
        }
    }

    public void showBorrowedBooks() {
        System.out.println(name + "'s Borrowed Books:");
        for (Book book : borrowedBooks) {
            System.out.println("- " + book.getTitle() + " by " +
book.getAuthor());
        }
    }
}

public class Library {
    public static void main(String[] args) {
        Book book1 = new Book("1984", "George Orwell");
        Book book2 = new Book("The Great Gatsby", "F. Scott Fitzgerald");

        Member member = new Member("Alice");

        member.borrowBook(book1);
        member.borrowBook(book2);

        member.showBorrowedBooks();

        member.returnBook(book1);
        member.showBorrowedBooks();
    }
}
```

---

2. Bank Account Management System

A simple system to manage multiple bank accounts, including deposit, withdrawal, and balance check features. The program includes classes for `BankAccount` and `Customer`.

Concepts Used: Classes, Encapsulation, Polymorphism

java
Copy code
```
class BankAccount {
    private String accountNumber;
    private double balance;

    public BankAccount(String accountNumber, double initialBalance) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public String getAccountNumber() {
        return accountNumber;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: $" + amount);
        } else {
            System.out.println("Invalid deposit amount.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew: $" + amount);
        } else {
            System.out.println("Invalid or insufficient balance.");
        }
    }
}

class Customer {
    private String name;
    private BankAccount bankAccount;

    public Customer(String name, String accountNumber, double initialBalance)
{
        this.name = name;
        this.bankAccount = new BankAccount(accountNumber, initialBalance);
    }

    public void showBalance() {
        System.out.println("Balance for " + name + ": $" +
bankAccount.getBalance());
    }

    public void deposit(double amount) {
        bankAccount.deposit(amount);
    }
    public void withdraw(double amount) {
        bankAccount.withdraw(amount);
    }
}

public class BankSystem {
    public static void main(String[] args) {
        Customer customer = new Customer("John Doe", "12345", 500.0);
        customer.showBalance();

        customer.deposit(200.0);
        customer.showBalance();

        customer.withdraw(100.0);
        customer.showBalance();
    }
}
```

---

3. Inventory Management System

This system keeps track of items in an inventory. It includes classes for `Item`, `PerishableItem`, and `Inventory`.

Concepts Used: Inheritance, Polymorphism, Encapsulation

```
java
Copy code
class Item {
    private String name;
    private int quantity;

    public Item(String name, int quantity) {
        this.name = name;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public int getQuantity() {
        return quantity;
    }

    public void addQuantity(int amount) {
        quantity += amount;
    }

    public void reduceQuantity(int amount) {
        if (quantity >= amount) {
            quantity -= amount;
        } else {
            System.out.println("Insufficient quantity for " + name);
        }
    }
}

class PerishableItem extends Item {
    private int expirationDays;

    public PerishableItem(String name, int quantity, int expirationDays) {
        super(name, quantity);
        this.expirationDays = expirationDays;
    }

    public int getExpirationDays() {
        return expirationDays;
    }
}

public class InventoryManagement {
    public static void main(String[] args) {
        Item item1 = new Item("Laptop", 50);
        PerishableItem item2 = new PerishableItem("Milk", 30, 7);

        System.out.println("Item: " + item1.getName() + ", Quantity: " +
item1.getQuantity());
        System.out.println("Item: " + item2.getName() + ", Quantity: " +
item2.getQuantity() + ", Expires in: " + item2.getExpirationDays() + "
days");

        item1.addQuantity(10);
        item2.reduceQuantity(5);

        System.out.println("Updated Quantity of " + item1.getName() + ": " +
item1.getQuantity());
        System.out.println("Updated Quantity of " + item2.getName() + ": " +
item2.getQuantity());
    }
}
```

---

4. Shopping Cart System

This project implements a simple shopping cart, where items can be added and removed. It uses `Product` and `ShoppingCart` classes to encapsulate functionality.

Concepts Used: Encapsulation, Aggregation, Loops

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

class Product {
    private String name;
    private double price;
    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

class ShoppingCart {
    private List<Product> products;

    public ShoppingCart() {
        products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        products.add(product);
        System.out.println(product.getName() + " added to cart.");
    }

    public void removeProduct(Product product) {
        if (products.remove(product)) {
            System.out.println(product.getName() + " removed from cart.");
        } else {
            System.out.println(product.getName() + " not found in cart.");
        }
    }

    public double calculateTotal() {
        double total = 0;
        for (Product product : products) {
            total += product.getPrice();
        }
        return total;
    }

    public void showCart() {
        System.out.println("Shopping Cart:");
        for (Product product : products) {
            System.out.println("- " + product.getName() + ": $" +
product.getPrice());
        }
        System.out.println("Total: $" + calculateTotal());
    }
}

public class ShoppingCartSystem {
    public static void main(String[] args) {
        Product product1 = new Product("Laptop", 999.99);
        Product product2 = new Product("Headphones", 199.99);
        ShoppingCart cart = new ShoppingCart();
        cart.addProduct(product1);
        cart.addProduct(product2);

        cart.showCart();

        cart.removeProduct(product1);
        cart.showCart();
    }
}
```

---

4. Shopping Cart System

This project implements a simple shopping cart, where items can be added and removed. It uses `Product` and `ShoppingCart` classes to encapsulate functionality.

Concepts Used: Encapsulation, Aggregation, Loops

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

class Product {
    private String name;
    private double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }
}

class ShoppingCart {
    private List<Product> products;

    public ShoppingCart() {
        products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        products.add(product);
        System.out.println(product.getName() + " added to cart.");
    }
    public void removeProduct(Product product) {
        if (products.remove(product)) {
            System.out.println(product.getName() + " removed from cart.");
        } else {
            System.out.println(product.getName() + " not found in cart.");
        }
    }

    public double calculateTotal() {
        double total = 0;
        for (Product product : products) {
            total += product.getPrice();
        }
        return total;
    }

    public void showCart() {
        System.out.println("Shopping Cart:");
        for (Product product : products) {
            System.out.println("- " + product.getName() + ": $" +
product.getPrice());
        }
        System.out.println("Total: $" + calculateTotal());
    }
}

public class ShoppingCartSystem {
    public static void main(String[] args) {
        Product product1 = new Product("Laptop", 999.99);
        Product product2 = new Product("Headphones", 199.99);

        ShoppingCart cart = new ShoppingCart();
        cart.addProduct(product1);
        cart.addProduct(product2);

        cart.showCart();

        cart.removeProduct(product1);
        cart.showCart();
    }
}
```

Here are more OOP projects in Java that cover various scenarios, using classes, encapsulation, inheritance, and polymorphism.

---

5. Student Management System

This project manages student information, including their grades and courses. It includes classes for `Student`, `Course`, and `Enrollment`.

Concepts Used: Classes, Encapsulation, Aggregation, Composition

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

class Course {
    private String courseName;
    private int credits;

    public Course(String courseName, int credits) {
        this.courseName = courseName;
        this.credits = credits;
    }

    public String getCourseName() {
        return courseName;
    }

    public int getCredits() {
        return credits;
    }
}

class Student {
    private String name;
    private List<Course> enrolledCourses;

    public Student(String name) {
        this.name = name;
        this.enrolledCourses = new ArrayList<>();
    }

    public void enrollCourse(Course course) {
        enrolledCourses.add(course);
        System.out.println(name + " enrolled in " + course.getCourseName());
    }

    public void showCourses() {
        System.out.println(name + "'s Courses:");
        for (Course course : enrolledCourses) {
            System.out.println("- " + course.getCourseName() + " (" +
course.getCredits() + " credits)");
```

```
        }
    }
}

public class StudentManagementSystem {
    public static void main(String[] args) {
        Student student = new Student("Alice");

        Course math = new Course("Mathematics", 3);
        Course science = new Course("Science", 4);

        student.enrollCourse(math);
        student.enrollCourse(science);

        student.showCourses();
    }
}
```

---

6. Employee Payroll System

This project calculates and manages the payroll for employees, using classes for `Employee`, `FullTimeEmployee`, and `PartTimeEmployee`.

Concepts Used: Inheritance, Polymorphism, Encapsulation

```
java
Copy code
abstract class Employee {
    private String name;
    private int id;

    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public abstract double calculateSalary();

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }
}

class FullTimeEmployee extends Employee {
    private double monthlySalary;

    public FullTimeEmployee(String name, int id, double monthlySalary) {
        super(name, id);
        this.monthlySalary = monthlySalary;
    }

    @Override
    public double calculateSalary() {
        return monthlySalary;
    }
}

class PartTimeEmployee extends Employee {
    private double hourlyWage;
    private int hoursWorked;

    public PartTimeEmployee(String name, int id, double hourlyWage, int
hoursWorked) {
        super(name, id);
        this.hourlyWage = hourlyWage;
        this.hoursWorked = hoursWorked;
    }

    @Override
    public double calculateSalary() {
        return hourlyWage * hoursWorked;
    }
}

public class PayrollSystem {
    public static void main(String[] args) {
        Employee fullTime = new FullTimeEmployee("John Doe", 1, 3000.0);
        Employee partTime = new PartTimeEmployee("Jane Smith", 2, 20.0, 120);

        System.out.println(fullTime.getName() + "'s Salary: $" +
fullTime.calculateSalary());
        System.out.println(partTime.getName() + "'s Salary: $" +
partTime.calculateSalary());
    }
}
```

---

7. Online Order Processing System

This project manages online orders, including order details and total amount calculations. It includes classes for `Product`, `Order`, and `Customer`.

Concepts Used: Aggregation, Encapsulation

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

class Product {
    private String productName;
    private double price;
    public Product(String productName, double price) {
        this.productName = productName;
        this.price = price;
    }

    public String getProductName() {
        return productName;
    }

    public double getPrice() {
        return price;
    }
}

class Order {
    private List<Product> products;

    public Order() {
        products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        products.add(product);
        System.out.println(product.getProductName() + " added to the
order.");
    }

    public double calculateTotal() {
        double total = 0;
        for (Product product : products) {
            total += product.getPrice();
        }
        return total;
    }

    public void showOrderDetails() {
        System.out.println("Order Details:");
        for (Product product : products) {
            System.out.println("- " + product.getProductName() + ": $" +
product.getPrice());
        }
        System.out.println("Total Amount: $" + calculateTotal());
    }
}

public class OrderSystem {
    public static void main(String[] args) {
        Product product1 = new Product("Laptop", 999.99);
        Product product2 = new Product("Phone", 499.99);

        Order order = new Order();
        order.addProduct(product1);
        order.addProduct(product2);

        order.showOrderDetails();
    }
}
```

8. Food Delivery System

This project manages food orders for a delivery service. It includes classes for `FoodItem`, `Order`, and `Customer`, with functionality for adding food items to orders.

Concepts Used: Interfaces, Enums, Abstraction, Composition

```
java
Copy code
import java.util.ArrayList;
import java.util.List;

enum Cuisine {
    ITALIAN,
    CHINESE,
    INDIAN
}

interface Orderable {
    double getPrice();
    String getName();
}

abstract class FoodItem implements Orderable {
    protected String name;
    protected double price;
    protected Cuisine cuisine;

    public FoodItem(String name, double price, Cuisine cuisine) {
        this.name = name;
        this.price = price;
        this.cuisine = cuisine;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public double getPrice() {
        return price;
    }

    public abstract void displayInfo();
}

class Pizza extends FoodItem {
    public Pizza(String name, double price) {
        super(name, price, Cuisine.ITALIAN);
    }

    @Override
    public void displayInfo() {
        System.out.println("Pizza: " + name + " - Price: $" + price);
    }
}

class Noodles extends FoodItem {
    public Noodles(String name, double price) {
        super(name, price, Cuisine.CHINESE);
    }

    @Override
    public void displayInfo() {
        System.out.println("Noodles: " + name + " - Price: $" + price);
    }
}

class Order {
    private List<Orderable> items;

    public Order() {
        items = new ArrayList<>();
    }

    public void addItem(Orderable item) {
        items.add(item);
        System.out.println(item.getName() + " added to the order.");
    }

    public void displayOrderDetails() {
        System.out.println("Order Details:");
        double total = 0;
        for (Orderable item : items) {
            item.displayInfo();
            total += item.getPrice();
        }
        System.out.println("Total Amount: $" + total);
    }
}

public class FoodDeliverySystem {
    public static void main(String[] args) {
        Order order = new Order();
        FoodItem pizza = new Pizza("Margherita Pizza", 12.00);
        FoodItem noodles = new Noodles("Chow Mein", 8.50);

        order.addItem(pizza);
        order.addItem(noodles);
        order.displayOrderDetails();
    }
}
```