Part 1: The Singleton Pattern (Creational)

```java
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

public class ServiceLogger {
    private static ServiceLogger instance;
    private PrintWriter writer;

    private ServiceLogger() throws IOException{
        try {
            FileWriter fw = new FileWriter("E:\\Vanier\\Semester 5 fall
2025\\Programming Patterns\\Labs\\Lab7\\app_log.txt", true);
            //FileWriter fw = new FileWriter("app_log.txt", true); this also
works

            this.writer = new PrintWriter(fw);
            System.out.println("--- LOGGER INTIALIZED ---");
        } catch (IOException e) {
            throw new IOException(e.getMessage());
        }
    }

    public static synchronized ServiceLogger getInstance() throws IOException{
        if(instance == null){
            instance = new ServiceLogger();
        }

        return instance;
    }

    public void closeWriter(){
        writer.close();
    }

    public void log(String serviceName, String message){
        writer.println("[" + new java.util.Date() + "] [" + serviceName + "] " +
message);
        writer.flush();
    }
    public static void main(String[] args) throws IOException {
        ServiceLogger userLogger = ServiceLogger.getInstance();
        userLogger.log("UserService", "User with userid:student1 logged in.");

        ServiceLogger productLogger = ServiceLogger.getInstance();
        productLogger.log("ProductService", "Product 'B-102' viewed.");

        System.out.println("Same logger instance: " + (userLogger ==
productLogger));
        userLogger.closeWriter();
    }


// 4. Observe the Results:
    // [Fri Oct 31 10:02:49 EDT 2025] [UserService] User with userid:student1
logged in.
    // [Fri Oct 31 10:02:49 EDT 2025] [ProductService] Product 'B-102' viewed.
}
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

public class ServiceLogger {
    private static ServiceLogger instance;
    private PrintWriter writer;

    private ServiceLogger() throws IOException{
        try {
            FileWriter fw = new FileWriter("E:\\Vanier\\Semester 5 fall
2025\\Programming Patterns\\Labs\\Lab7\\app_log.txt", true);
            //FileWriter fw = new FileWriter("app_log.txt", true); this also
works

            this.writer = new PrintWriter(fw);
            System.out.println("--- LOGGER INTIALIZED ---");
        } catch (IOException e) {
            throw new IOException(e.getMessage());
        }
    }

    public static synchronized ServiceLogger getInstance() throws IOException{
        if(instance == null){
            instance = new ServiceLogger();
        }

        return instance;
    }

    public void closeWriter(){
        writer.close();
    }

    public void log(String serviceName, String message){
        writer.println("[" + new java.util.Date() + "] [" + serviceName + "] " +
message);
        writer.flush();
    }

    public static void main(String[] args) throws IOException {
        ServiceLogger userLogger = ServiceLogger.getInstance();
        userLogger.log("UserService", "User with userid:student1 logged in.");

        ServiceLogger productLogger = ServiceLogger.getInstance();
        productLogger.log("ProductService", "Product 'B-102' viewed.");

        System.out.println("Same logger instance: " + (userLogger ==
productLogger));
        userLogger.closeWriter();
    }


// 4. Observe the Results:
    // [Fri Oct 31 10:02:49 EDT 2025] [UserService] User with userid:student1
logged in.
    // [Fri Oct 31 10:02:49 EDT 2025] [ProductService] Product 'B-102' viewed.
}
```

Part 2: The Adapter Pattern (Structural)

Target: IProductService Interface

```
public interface IProductService {
    public Product getProductById(int id);
}
```

Data Object: Product Class

```
public class Product {
    private int id;
    private String name;
    private double price;

    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public double getPrice() {
        return price;
    }
    @Override
    public String toString() {
        return "Product [id=" + id + ", name=" + name + ", price=" + price + "]";
    }
}
```

Adaptee: NewProductApi Class

```
public class NewProductApi {
    public String fetchProductJSon(int id){
        return "{ \"productId\": " + id + ", \"productName\": \"SuperWidget\",
\"cost\": 45.99 }";
    }
}
```

Adapter: ProductApiAdapter Class Implements IProduct

```
public class ProductApiAdapter implements IProductService{
    private NewProductApi npaObject;

    public ProductApiAdapter(NewProductApi npaObject){
        this.npaObject = npaObject;
    }

    @Override
    public Product getProductById(int id) {
        String productString = npaObject.fetchProductJSon(id);
        //{ "productId": 101, "productName": "SuperWidget", "cost": 45.99 }
        String productId =
productString.split("\"productId\":")[1].split(",")[0].trim();
        String productName = productString.split("\"productName\":
\"")[1].split("\"")[0];
        String cost = productString.split("\"cost\":")[1].split("}")[0].trim();

        Product newProduct = new Product(Integer.parseInt(productId),
productName, Double.parseDouble(cost));
        return newProduct;
    }
```

Client: Main Method

```
    public static void main(String[] args) {
        NewProductApi instance = new NewProductApi();
        ProductApiAdapter adapter = new ProductApiAdapter(instance);

        Product product = adapter.getProductById(10);
        System.out.println(product);
    }
}
```

Part 3: The Model-View-Controller (MVC) Pattern

The Model (Data Object): Product Class

```
public 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;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        long temp;
        temp = Double.doubleToLongBits(price);
        result = prime * result + (int) (temp ^ (temp >>> 32));
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Product other = (Product) obj;
        if (name == null) {
            if (other.name != null)
                return false;
```

```java
        } else if (!name.equals(other.name))
            return false;
        if (Double.doubleToLongBits(price) !=
Double.doubleToLongBits(other.price))
            return false;
        return true;
    }
}
```

The Model (The Logic): CartModel Class

```java
import java.util.HashMap;
import java.util.Map;

public class CartModel {
    private Map<Product, Integer> items;

    public CartModel(){
        items = new HashMap<>();
    }

    public void addProduct(Product p){
        if(!items.containsKey(p)){
            items.put(p, 1);
        }
        else{
            items.put(p, items.get(p) + 1);
        }
    }

    public Map<Product, Integer> getItems(){
        return items;
    }

    public double getTotalPrice(){
        double totalPrice = 0;
        for(Map.Entry<Product, Integer> entry: items.entrySet()){
            double entryPrice;
            entryPrice = entry.getKey().getPrice() * entry.getValue();
            totalPrice += entryPrice;
        }
        return totalPrice;
    }
}
```

The View: CartView Class

```
import java.util.Map;

public class CartView {
    public void displayCart(Map<Product, Integer> items, double total){
        double totalPrice = 0;
        for(Map.Entry<Product, Integer> entry: items.entrySet()){
            int quantity = entry.getValue();
            double price = entry.getKey().getPrice() * quantity;
            System.out.println("Name: " + entry.getKey().getName() + " Quantity: 
" + quantity + " Price: " + price);
            totalPrice += price;
        }
        System.out.println("Total Price: " + totalPrice);
    }
}
```

The Controller: CartController Class

```
import java.util.Map;

public class CartController {
    private CartModel model;
    private CartView view;

    public CartController(CartModel model, CartView view){
        this.model = model;
        this.view = view;
    }

    public void handleAddProduct(Product p){
        model.addProduct(p);
    }

    public void handleDisplayCart(){
        Map<Product, Integer> items = model.getItems();
        double total = model.getTotalPrice();
        view.displayCart(items, total);
        for(Map.Entry<Product, Integer> entry : items.entrySet()){
            System.out.println(entry.getKey().getName() + " (" + entry.getValue()
+ ")");
        }
    }
}
```

Test: Main Class

```
public class Main {
    public static void main(String[] args) {
        Product laptop = new Product("Laptop", 1200.00);
        Product mouse = new Product("Mouse", 45.00);

        CartModel model = new CartModel();
        CartView view = new CartView();
        CartController controller = new CartController(model, view);

        System.out.println("Adding Laptop...");
        controller.handleAddProduct(laptop);
        controller.handleDisplayCart();
        System.out.println("\nAdding Mouse...");
        controller.handleAddProduct(mouse);
        System.out.println("Adding another Laptop...");
        controller.handleAddProduct(laptop);
        controller.handleDisplayCart();
    }
}
```