Exercise Solutions-Design Patterns and Principles

Exercise 1: Implementing the Singleton Pattern

Code:

```
package Cognizant_Exercises;

class Logger {

private static Logger instance;

private Logger() {}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

System.out.println("Log: " + message);

}

}

public class SingletonTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("Hello from logger1");

logger2.log("Hello from logger2");

System.out.println(logger1 == logger2); // true

}
```

}

Output:

```
Console ×  Debug
<terminated> SingletonTest [Java Application] C:\Program Files\ec
Log: Hello from logger1
Log: Hello from logger2
true
```

Exercise 2: Implementing the Factory Method Pattern

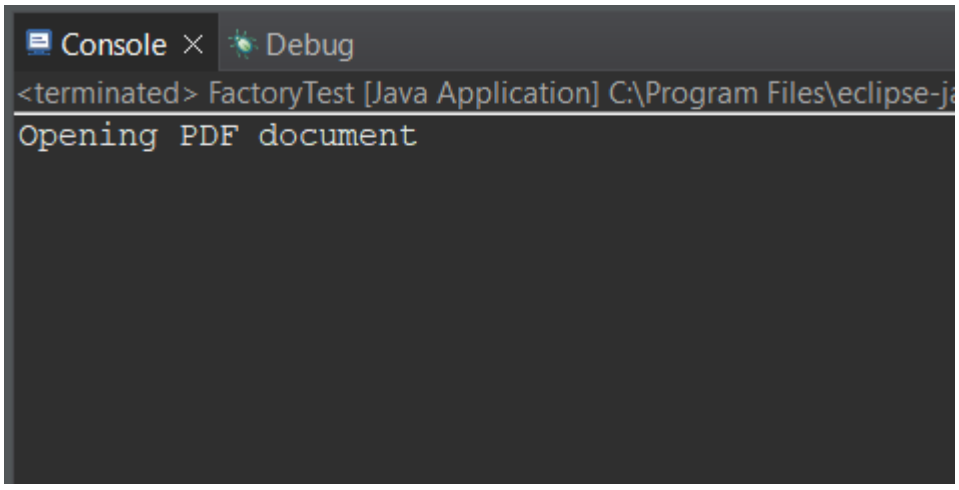
Code:

```
package cognizant_Exercises;

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

System.out.println("Opening Word document");

}

}

class PdfDocument implements Document {

public void open() {

System.out.println("Opening PDF document");

}

}

class ExcelDocument implements Document {
```

```
public void open() {
System.out.println("Opening Excel document");
}
}
abstract class DocumentFactory {
public abstract Document createDocument();
}
class WordFactory extends DocumentFactory {
public Document createDocument() {
return new WordDocument();
}
}
class PdfFactory extends DocumentFactory {
public Document createDocument() {
return new PdfDocument();
}
}
class ExcelFactory extends DocumentFactory {
public Document createDocument() {
return new ExcelDocument();
}
}
public class FactoryTest {
public static void main(String[] args) {
DocumentFactory factory = new PdfFactory();
```

```
Document doc = factory.createDocument();
doc.open();
}
}
```

Output:



Exercise 3: Implementing the Builder Pattern

Code:

```
package cognizant_Exercises;
class Computer {
private String CPU;
private String RAM;
private String storage;
private Computer(Builder builder) {
this.CPU = builder.CPU;
this.RAM = builder.RAM;
this.storage = builder.storage;
}
public static class Builder {
```

```
private String CPU;
private String RAM;
private String storage;
public Builder setCPU(String CPU) {
this.CPU = CPU;
return this;
}
public Builder setRAM(String RAM) {
this.RAM = RAM;
return this;
}
public Builder setStorage(String storage) {
this.storage = storage;
return this;
}
public Computer build() {
return new Computer(this);
}
}
public void showConfig() {
System.out.println("CPU: " + CPU + ", RAM: " + RAM + ", Storage: " +
storage);
}
}
public class BuilderTest {
```

```
public static void main(String[] args) {
Computer computer = new Computer.Builder()
.setCPU("AMD ryzen 5")
.setRAM("8GB")
.setStorage("512GB SSD")
.build();
computer.showConfig();
}
}
```

Output:

```
Console × Debug
<terminated> BuilderTest [Java Application] C:\Program Files\eclipse-java-2022-12-
CPU: AMD ryzen 5, RAM: 8GB, Storage: 512GB SSD
```

Exercise 4: Implementing the Adapter Pattern

Code:

```
package cognizant_Exercises;
interface PaymentProcessor {
void processPayment(double amount);
}
class RazorPayGateway {
public void makePayment(double amount) {
System.out.println("Paid " + amount + " using RazorPay");
}
```

```
}
class PayPalGateway {
public void sendPayment(double amount) {
System.out.println("Paid " + amount + " using PayPal");
}
}
class RazorPayAdapter implements PaymentProcessor {
private RazorPayGateway razorPay = new RazorPayGateway();
public void processPayment(double amount) {
razorPay.makePayment(amount);
}
}
class PayPalAdapter implements PaymentProcessor {
private PayPalGateway payPal = new PayPalGateway();
public void processPayment(double amount) {
payPal.sendPayment(amount);
}
}
public class AdapterTest {
public static void main(String[] args) {
PaymentProcessor processor = new PayPalAdapter();
processor.processPayment(1500.0);
}
}
```

Output:

```
Console × Debug
<terminated> AdapterTest [Java Application] C:\Program Files\ecl
Paid 1500.0 using PayPal
```

Exercise 5: Implementing the Decorator Pattern

Code:

```
package cognizant_Exercises;
interface Notifier {
void send(String message);
}
class EmailNotifier implements Notifier {
public void send(String message) {
System.out.println("Sending Email: " + message);
}
}
abstract class NotifierDecorator implements Notifier {
protected Notifier notifier;
public NotifierDecorator(Notifier notifier) {
this.notifier = notifier;
}
}
class SMSNotifier extends NotifierDecorator {
public SMSNotifier(Notifier notifier) {
super(notifier);
```

```
}
public void send(String message) {
notifier.send(message);
System.out.println("Sending SMS: " + message);
}
}
class SlackNotifier extends NotifierDecorator {
public SlackNotifier(Notifier notifier) {
super(notifier);
}
public void send(String message) {
notifier.send(message);
System.out.println("Sending Slack message: " + message);
}
}
public class DecoratorTest {
public static void main(String[] args) {
Notifier notifier = new SlackNotifier(new SMSNotifier(new
EmailNotifier()));
notifier.send("Design Pattern Alert!");
}
}
```

Output:

```
Console × Debug
<terminated> DecoratorTest [Java Application] C:\Program Files\eclipse-java-2
Sending Email: Design Pattern Alert!
Sending SMS: Design Pattern Alert!
Sending Slack message: Design Pattern Alert!
```

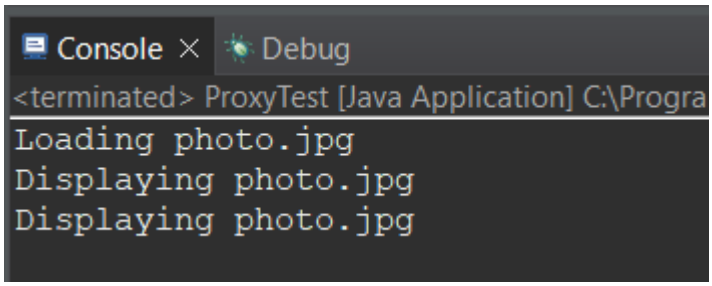
Exercise 6: Implementing the Proxy Pattern

Code:

```
package cognizant_Exercises;

interface Image {

void display();

}

class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromDisk();

}

private void loadFromDisk() {

System.out.println("Loading " + filename);

}

public void display() {

System.out.println("Displaying " + filename);

}

}
```

```
class ProxyImage implements Image {
private RealImage realImage;
private String filename;
public ProxyImage(String filename) {
this.filename = filename;
}
public void display() {
if (realImage == null) {
realImage = new RealImage(filename);
}
realImage.display();
}
}
public class ProxyTest {
public static void main(String[] args) {
Image image = new ProxyImage("photo.jpg");
image.display(); // loading and displaying
image.display(); // only displaying
}
}
```

Output:



Exercise 7: Implementing the Observer Pattern

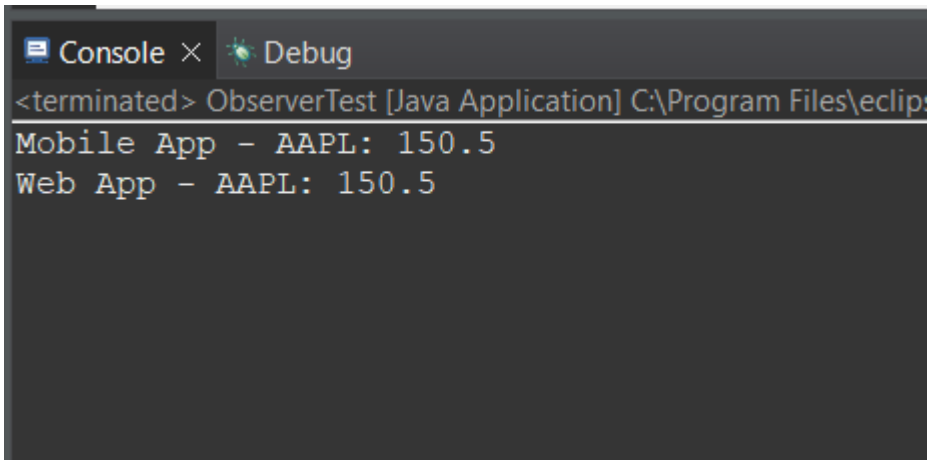
Code:

```
package cognizant_Exercises;
import java.util.ArrayList;
import java.util.List;
interface Observer {
void update(String stockName, double price);
}
interface Stock {
void registerObserver(Observer o);
void removeObserver(Observer o);
void notifyObservers();
}
class StockMarket implements Stock {
private List<Observer> observers = new ArrayList<>();
private String stockName;
private double price;
public void setStockData(String stockName, double price) {
this.stockName = stockName;
this.price = price;
notifyObservers();
}
public void registerObserver(Observer o) {
observers.add(o);
}
```

```
public void removeObserver(Observer o) {
observers.remove(o);
}
public void notifyObservers() {
for (Observer o : observers) {
o.update(stockName, price);
}
}
}
class MobileApp implements Observer {
public void update(String stockName, double price) {
System.out.println("Mobile App - " + stockName + ": " + price);
}
}
class WebApp implements Observer {
public void update(String stockName, double price) {
System.out.println("Web App - " + stockName + ": " + price);
}
}
public class ObserverTest {
public static void main(String[] args) {
StockMarket market = new StockMarket();
Observer mobile = new MobileApp();
Observer web = new WebApp();
market.registerObserver(mobile);
```

```
market.registerObserver(web);
market.setStockData("AAPL", 150.50);
}
}
```

Output:



Exercise 8: Implementing the Strategy Pattern

Code:

```
package cognizant_Exercises;
interface PaymentStrategy {
void pay(int amount);
}
class CreditCardPayment implements PaymentStrategy {
public void pay(int amount) {
System.out.println("Paid " + amount + " using Credit Card.");
}
}
class PayPalPayment implements PaymentStrategy {
public void pay(int amount) {
```

```
System.out.println("Paid " + amount + " using PayPal.");
}
}
class PaymentContext {
private PaymentStrategy strategy;
public PaymentContext(PaymentStrategy strategy) {
this.strategy = strategy;
}
public void executePayment(int amount) {
strategy.pay(amount);
}
}
public class StrategyTest {
public static void main(String[] args) {
PaymentContext context = new PaymentContext(new
CreditCardPayment());
context.executePayment(1000);
context = new PaymentContext(new PayPalPayment());
context.executePayment(2000);
}
}
```

Output:

```
Console × Debug
<terminated> StrategyTest [Java Application] C:\Program Files\eclipse-j
Paid 1000 using Credit Card.
Paid 2000 using PayPal.
```

Exercise 9: Implementing the Command Pattern

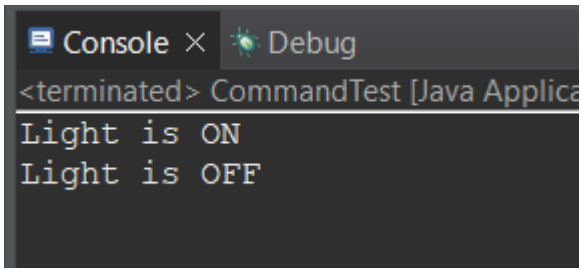
<u>Code:</u>

```
package cognizant_Exercises;

interface Command {

void execute();

}

class Light {

public void on() {

System.out.println("Light is ON");

}

public void off() {

System.out.println("Light is OFF");

}

}

class LightOnCommand implements Command {

private Light light;

public LightOnCommand(Light light) {

this.light = light;

}
```

```
public void execute() {
light.on();
}
}
class LightOffCommand implements Command {
private Light light;
public LightOffCommand(Light light) {
this.light = light;
}
public void execute() {
light.off();
}
}
class RemoteControl {
private Command command;
public void setCommand(Command command) {
this.command = command;
}
public void pressButton() {
command.execute();
}
}
public class CommandTest {
public static void main(String[] args) {
Light light = new Light();
```

```
Command on = new LightOnCommand(light);
Command off = new LightOffCommand(light);
RemoteControl remote = new RemoteControl();
remote.setCommand(on);
remote.pressButton();
remote.setCommand(off);
remote.pressButton();
}
}
```

Output:



Exercise 10: Implementing the MVC Pattern

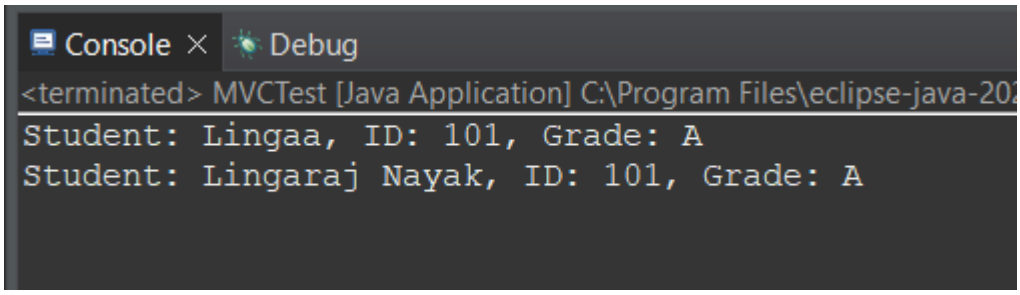
Code:

```
package cognizant_Exercises;
class Student {
private String name;
private String id;
private String grade;
public Student(String name, String id, String grade) {
this.name = name;
this.id = id;
this.grade = grade;
```

```
}
public String getName() { return name; }
public String getId() { return id; }
public String getGrade() { return grade; }
public void setName(String name) { this.name = name; }
public void setGrade(String grade) { this.grade = grade; }
}
class StudentView {
public void displayStudentDetails(String name, String id, String grade) {
System.out.println("Student: " + name + ", ID: " + id + ", Grade: " + grade);
}
}
class StudentController {
private Student model;
private StudentView view;
public StudentController(Student model, StudentView view) {
this.model = model;
this.view = view;
}
public void updateView() {
view.displayStudentDetails(model.getName(), model.getId(),
model.getGrade());
}
public void setStudentName(String name) {
model.setName(name);
```

```
}
}
public class MVCTest {
public static void main(String[] args) {
Student student = new Student("Lingaa", "101", "A");
StudentView view = new StudentView();
StudentController controller = new StudentController(student, view);
controller.updateView();
controller.setStudentName("Lingaraj Nayak");
controller.updateView();
}
}
```

Output:



Exercise 11: Implementing Dependency Injection

Code:

```
package cognizant_Exercises;
interface CustomerRepository {
String findCustomerById(String id);
}
class CustomerRepositoryImpl implements CustomerRepository {
public String findCustomerById(String id) {
```

```
return "Customer#" + id;
}
}
class CustomerService {
private CustomerRepository repository;
public CustomerService(CustomerRepository repository) {
this.repository = repository;
}
public void displayCustomer(String id) {
System.out.println("Found: " + repository.findCustomerById(id));
}
}
public class DIExample {
public static void main(String[] args) {
CustomerRepository repo = new CustomerRepositoryImpl();
CustomerService service = new CustomerService(repo);
service.displayCustomer("2025");
}
}
```

Output:

```
Console ×  Debug
<terminated> DIExample [Java Application] C:\Program Files\eclipse-java-2022-12
Found: Customer#2025
```

Submitted By:

Name : Lingaraj Nayak

Superset ID: 6387607