Week 1 Assignment Submission

Exercise 1: Implementing the Singleton Pattern

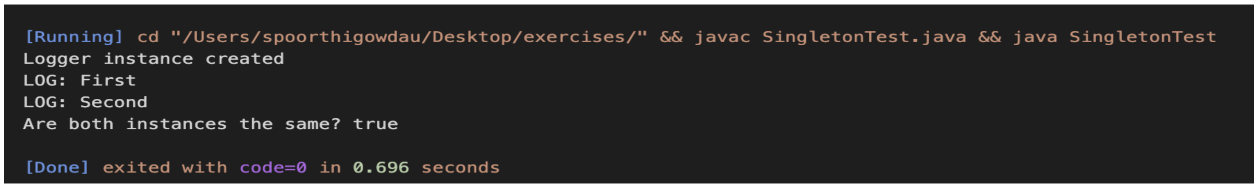
Logger.java

```java
public class Logger {
   private static Logger instance;
   private Logger() {
       System.out.println("Logger instance created");
   }
   public static Logger getInstance() {
       if (instance == null) {
           instance = new Logger();
       }
       return instance;
   }
   public void log(String message) {
       System.out.println("LOG: " + message);
   }
}
```

SingletonTest.java

```java
public class SingletonTest {
   public static void main(String[] args) {
       Logger logger1 = Logger.getInstance();
       logger1.log("First");
       Logger logger2 = Logger.getInstance();
       logger2.log("Second");
       System.out.println("Are both instances the same? " + (logger1 == logger2));
   }
}
```

Output:



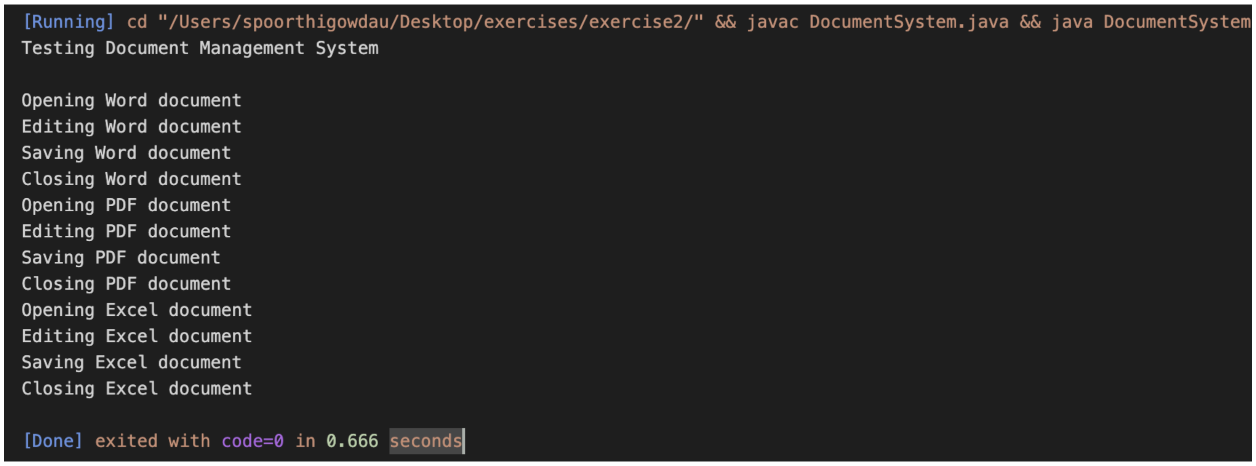
```
[Running] cd "/Users/spoorthigowdau/Desktop/exercises/" && javac SingletonTest.java && java SingletonTest
Logger instance created
LOG: First
LOG: Second
Are both instances the same? true

[Done] exited with code=0 in 0.696 seconds
```

Exercise 2: Implementing the Factory Method Pattern

```java
public class DocumentSystem {
   public interface Document {
       void open();
       void edit();
       void save();
       void close();
   }
   public static class WordDocument implements Document {
       public void open() { System.out.println("Opening Word document"); }
       public void edit() { System.out.println("Editing Word document"); }
       public void save() { System.out.println("Saving Word document"); }
       public void close() { System.out.println("Closing Word document"); }
   }
   public static class PdfDocument implements Document {
       public void open() { System.out.println("Opening PDF document"); }
       public void edit() { System.out.println("Editing PDF document"); }
       public void save() { System.out.println("Saving PDF document"); }
       public void close() { System.out.println("Closing PDF document"); }
   }
 public static class ExcelDocument implements Document {
       public void open() { System.out.println("Opening Excel document"); }
       public void edit() { System.out.println("Editing Excel document"); }
       public void save() { System.out.println("Saving Excel document"); }
       public void close() { System.out.println("Closing Excel document"); }
   }
   public abstract static class DocumentFactory {
       public abstract Document createDocument();
       public void processDocument() {
           Document doc = createDocument();
           doc.open();
           doc.edit();
           doc.save();
           doc.close();
       }
   }
public static class WordDocumentFactory extends DocumentFactory {
       public Document createDocument() { return new WordDocument(); }
```

```
    }
public static class PdfDocumentFactory extends DocumentFactory {
        public Document createDocument() { return new PdfDocument(); }
    }
public static class ExcelDocumentFactory extends DocumentFactory {
        public Document createDocument() { return new ExcelDocument(); }
    }
public static void main(String[] args) {
        System.out.println("Testing Document Management System\n");
DocumentFactory wordFactory = new WordDocumentFactory();
        DocumentFactory pdfFactory = new PdfDocumentFactory();
        DocumentFactory excelFactory = new ExcelDocumentFactory();
       wordFactory.processDocument();
        pdfFactory.processDocument();
        excelFactory.processDocument();
    }
}
```

Output:



```
[Running] cd "/Users/spoorthigowdau/Desktop/exercises/exercise2/" && javac DocumentSystem.java && java DocumentSystem
Testing Document Management System

Opening Word document
Editing Word document
Saving Word document
Closing Word document
Opening PDF document
Editing PDF document
Saving PDF document
Closing PDF document
Opening Excel document
Editing Excel document
Saving Excel document
Closing Excel document

[Done] exited with code=0 in 0.666 seconds
```

Exercise 3: Implementing the Builder Pattern

Computer.java

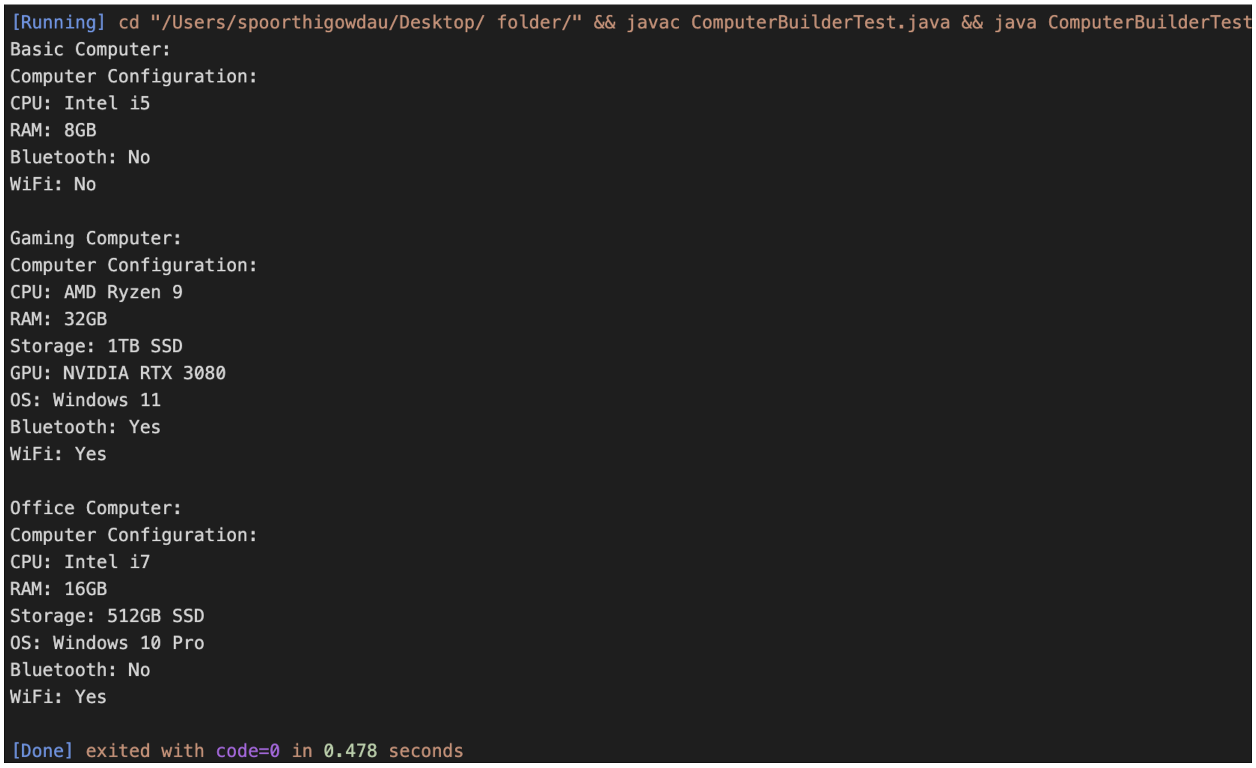
```java
public class Computer {
   private final String cpu;
   private final String ram;
   private final String storage;
   private final String gpu;
   private final String os;
   private final boolean hasBluetooth;
   private final boolean hasWiFi;
   private Computer(Builder builder) {
       this.cpu = builder.cpu;
       this.ram = builder.ram;
       this.storage = builder.storage;
       this.gpu = builder.gpu;
       this.os = builder.os;
       this.hasBluetooth = builder.hasBluetooth;
       this.hasWiFi = builder.hasWiFi;}
   public String getCpu() { return cpu; }
   public String getRam() { return ram; }
   public String getStorage() { return storage; }
   public String getGpu() { return gpu; }
   public String getOs() { return os; }
   public boolean hasBluetooth() { return hasBluetooth; }
   public boolean hasWiFi() { return hasWiFi; }
   @Override
   public String toString() {
       return "Computer Configuration:\n" +
              "CPU: " + cpu + "\n" +
              "RAM: " + ram + "\n" +
              (storage != null ? "Storage: " + storage + "\n" : "") +
              (gpu != null ? "GPU: " + gpu + "\n" : "") +
              (os != null ? "OS: " + os + "\n" : "") +
              "Bluetooth: " + (hasBluetooth ? "Yes" : "No") + "\n" +
              "WiFi: " + (hasWiFi ? "Yes" : "No");}
   public static class Builder {
       private final String cpu;
       private final String ram;
       private String storage = null;
       private String gpu = null;
       private String os = null;
       private boolean hasBluetooth = false;
       private boolean hasWiFi = false;
       public Builder(String cpu, String ram) {
           this.cpu = cpu;
           this.ram = ram;}
       public Builder storage(String storage) {
           this.storage = storage;
           return this;}
       public Builder gpu(String gpu) {
           this.gpu = gpu;
           return this;}
       public Builder os(String os) {
           this.os = os;
           return this;}
       public Builder hasBluetooth(boolean hasBluetooth) {
           this.hasBluetooth = hasBluetooth;
           return this;}
```

```java
        public Builder hasWiFi(boolean hasWiFi) {
            this.hasWiFi = hasWiFi;
            return this;}
        public Computer build() {
            return new Computer(this);
        }}}
```

ComputerBuilderTest.java

```java
public class ComputerBuilderTest {
   public static void main(String[] args) {
       Computer basicComputer = new Computer.Builder("Intel i5", "8GB")
               .build();
       System.out.println("Basic Computer:\n" + basicComputer + "\n");
       Computer gamingComputer = new Computer.Builder("AMD Ryzen 9", "32GB")
               .storage("1TB SSD")
               .gpu("NVIDIA RTX 3080")
               .os("Windows 11")
               .hasBluetooth(true)
               .hasWiFi(true)
               .build();
       System.out.println("Gaming Computer:\n" + gamingComputer + "\n");
       Computer officeComputer = new Computer.Builder("Intel i7", "16GB")
               .storage("512GB SSD")
               .os("Windows 10 Pro")
               .hasWiFi(true)
               .build();
       System.out.println("Office Computer:\n" + officeComputer);
   }
}
```

OUTPUT:



```
[Running] cd "/Users/spoorthigowdau/Desktop/ folder/" && javac ComputerBuilderTest.java && java ComputerBuilderTest
Basic Computer:
Computer Configuration:
CPU: Intel i5
RAM: 8GB
Bluetooth: No
WiFi: No

Gaming Computer:
Computer Configuration:
CPU: AMD Ryzen 9
RAM: 32GB
Storage: 1TB SSD
GPU: NVIDIA RTX 3080
OS: Windows 11
Bluetooth: Yes
WiFi: Yes

Office Computer:
Computer Configuration:
CPU: Intel i7
RAM: 16GB
Storage: 512GB SSD
OS: Windows 10 Pro
Bluetooth: No
WiFi: Yes

[Done] exited with code=0 in 0.478 seconds
```

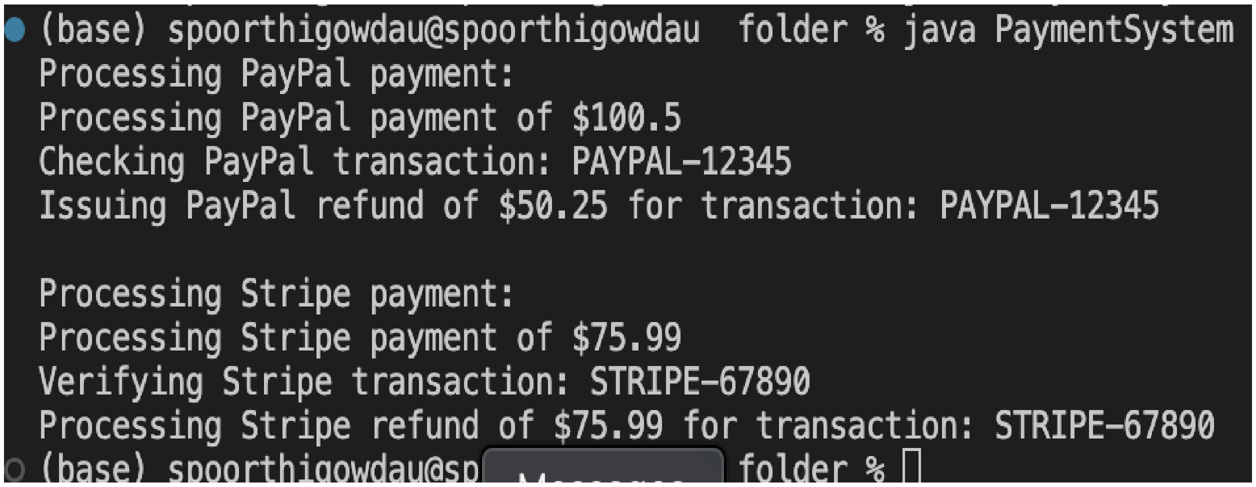
Exercise 4: Implementing the Adapter Pattern

PaymentSystem.java

```java
public class PaymentSystem {
    public static void main(String[] args) {
        PaymentProcessor payPalProcessor = new PayPalAdapter(new PayPalGateway());
        PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());
        System.out.println("Processing PayPal payment:");
        payPalProcessor.processPayment(100.50);
        payPalProcessor.verifyTransaction("PAYPAL-12345");
        payPalProcessor.refundPayment("PAYPAL-12345", 50.25);
        System.out.println("\nProcessing Stripe payment:");
        stripeProcessor.processPayment(75.99);
        stripeProcessor.verifyTransaction("STRIPE-67890");
        stripeProcessor.refundPayment("STRIPE-67890", 75.99);
    }}
interface PaymentProcessor {
    void processPayment(double amount);
    boolean verifyTransaction(String transactionId);
    void refundPayment(String transactionId, double amount);
}
class PayPalGateway {
    public void sendPayment(double amount) {
        System.out.println("Processing PayPal payment of $" + amount);
    }
    public boolean checkPaymentStatus(String transactionId) {
        System.out.println("Checking PayPal transaction: " + transactionId);
        return true;
    }
    public void issueRefund(String transactionId, double amount) {
        System.out.println("Issuing PayPal refund of $" + amount + " for transaction: "
+ transactionId);
    }
}

class StripeGateway {
    public void makePayment(double amount) {
        System.out.println("Processing Stripe payment of $" + amount);
    }

    public boolean verifyPayment(String transactionId) {
        System.out.println("Verifying Stripe transaction: " + transactionId);
        return true;
    }

    public void refund(String transactionId, double amount) {
        System.out.println("Processing Stripe refund of $" + amount + " for
transaction: " + transactionId);
    }
}

class PayPalAdapter implements PaymentProcessor {
    private final PayPalGateway payPalGateway;

    public PayPalAdapter(PayPalGateway payPalGateway) {
        this.payPalGateway = payPalGateway;
    }

    public void processPayment(double amount) {
        payPalGateway.sendPayment(amount);
    }

    public boolean verifyTransaction(String transactionId) {
        return payPalGateway.checkPaymentStatus(transactionId);
    }

    public void refundPayment(String transactionId, double amount) {
        payPalGateway.issueRefund(transactionId, amount);
    }
}

class StripeAdapter implements PaymentProcessor {
    private final StripeGateway stripeGateway;

    public StripeAdapter(StripeGateway stripeGateway) {
        this.stripeGateway = stripeGateway;
    }

    public void processPayment(double amount) {
        stripeGateway.makePayment(amount);
    }
```

```java
    public boolean verifyTransaction(String transactionId) {
        return stripeGateway.verifyPayment(transactionId);
    }

    public void refundPayment(String transactionId, double amount) {
        stripeGateway.refund(transactionId, amount);
    }
}
```

Output:

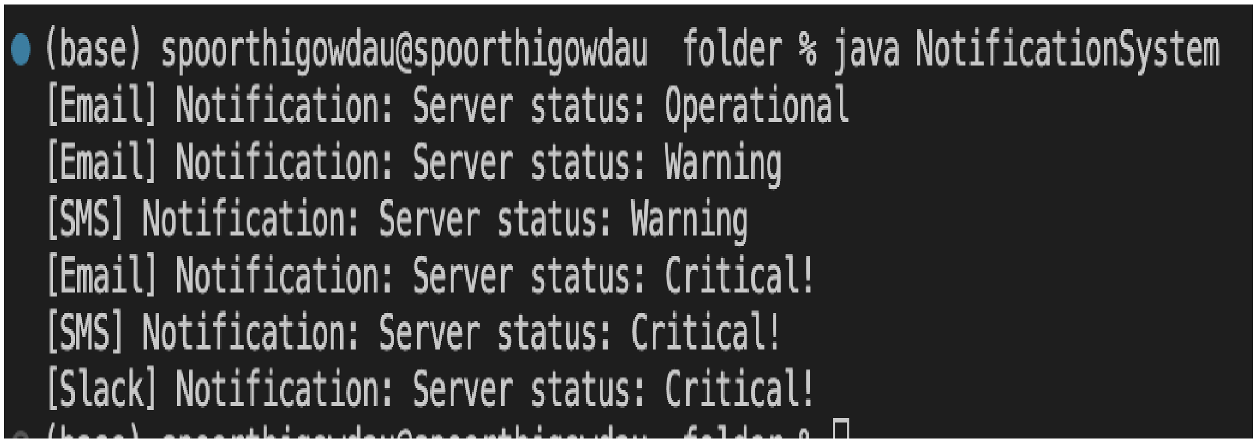


```
(base) spoorthigowdau@spoorthigowdau  folder % java PaymentSystem
Processing PayPal payment:
Processing PayPal payment of $100.5
Checking PayPal transaction: PAYPAL-12345
Issuing PayPal refund of $50.25 for transaction: PAYPAL-12345

Processing Stripe payment:
Processing Stripe payment of $75.99
Verifying Stripe transaction: STRIPE-67890
Processing Stripe refund of $75.99 for transaction: STRIPE-67890
(base) spoorthigowdau@sp              folder %
```

Exercise 5: Implementing the Decorator Pattern

```java
public class NotificationSystem {
   public static void main(String[] args) {
       Notifier emailOnly = new EmailNotifier();
       emailOnly.send("Server status: Operational");

       Notifier emailAndSms = new SMSNotifierDecorator(new EmailNotifier());
       emailAndSms.send("Server status: Warning");

       Notifier allChannels = new SlackNotifierDecorator(
                              new SMSNotifierDecorator(
                              new EmailNotifier()));
       allChannels.send("Server status: Critical!");
   }
}

interface Notifier {
   void send(String message);
}

class EmailNotifier implements Notifier {
   public void send(String message) {
       System.out.println("[Email] Notification: " + message);
   }
}

abstract class NotifierDecorator implements Notifier {
   protected Notifier wrappedNotifier;

   public NotifierDecorator(Notifier notifier) {
       this.wrappedNotifier = notifier;
   }

   public void send(String message) {
       wrappedNotifier.send(message);
   }
}

class SMSNotifierDecorator extends NotifierDecorator {
   public SMSNotifierDecorator(Notifier notifier) {
       super(notifier);
   }

   public void send(String message) {
       super.send(message);
       System.out.println("[SMS] Notification: " + message);
   }
}

class SlackNotifierDecorator extends NotifierDecorator {
   public SlackNotifierDecorator(Notifier notifier) {
       super(notifier);
   }

   public void send(String message) {
       super.send(message);
       System.out.println("[Slack] Notification: " + message);
   }
}
```

Output:



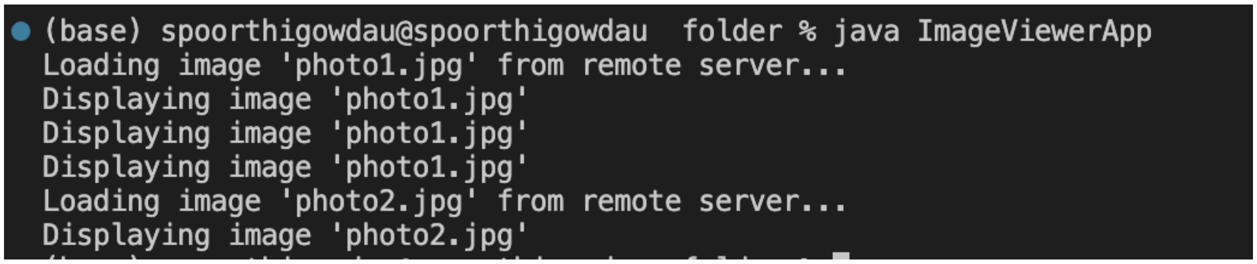
```
(base) spoorthigowdau@spoorthigowdau  folder % java NotificationSystem
[Email] Notification: Server status: Operational
[Email] Notification: Server status: Warning
[SMS] Notification: Server status: Warning
[Email] Notification: Server status: Critical!
[SMS] Notification: Server status: Critical!
[Slack] Notification: Server status: Critical!
```

Exercise 6: Implementing the Proxy Pattern

ImageViewerApp.java

```java
public class ImageViewerApp {
    public static void main(String[] args) {
        Image image1 = new ProxyImage("photo1.jpg");
        Image image2 = new ProxyImage("photo2.jpg");

        image1.display();
        image1.display();
        image1.display();
        image2.display();
    }
}

interface Image {
    void display();
}

class RealImage implements Image {
    private String filename;

    public RealImage(String filename) {
        this.filename = filename;
        loadFromServer();
    }

    private void loadFromServer() {
        System.out.println("Loading image '" + filename + "' from remote server...");
    }

    public void display() {
        System.out.println("Displaying image '" + filename + "'");
    }
}

class ProxyImage implements Image {
    private String filename;
    private RealImage realImage;
    public ProxyImage(String filename) {
        this.filename = filename;
    }

    public void display() {
        if (realImage == null) {
            realImage = new RealImage(filename);
        }
        realImage.display();
    }
}
```

Output:



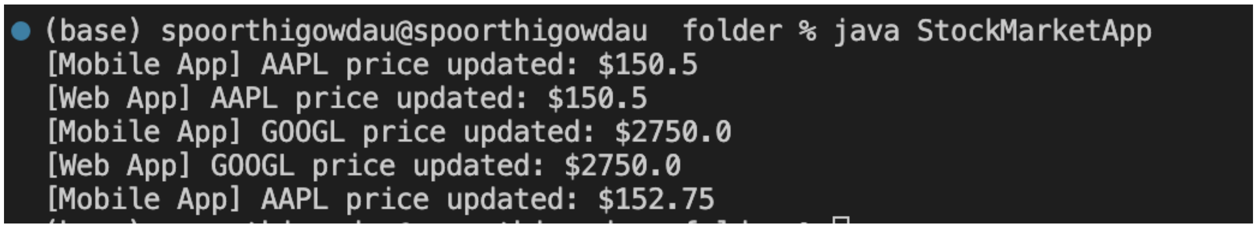
```
(base) spoorthigowdau@spoorthigowdau  folder % java ImageViewerApp
Loading image 'photo1.jpg' from remote server...
Displaying image 'photo1.jpg'
Displaying image 'photo1.jpg'
Displaying image 'photo1.jpg'
Loading image 'photo2.jpg' from remote server...
Displaying image 'photo2.jpg'
```

Exercise 7: Implementing the Observer Pattern

StockMarketApp.java

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

public class StockMarketApp {
    public static void main(String[] args) {
        StockMarket market = new StockMarket();

        Observer mobileApp = new MobileApp();
        Observer webApp = new WebApp();

        market.registerObserver(mobileApp);
        market.registerObserver(webApp);

        market.setStockPrice("AAPL", 150.50);
        market.setStockPrice("GOOGL", 2750.00);

        market.removeObserver(webApp);

        market.setStockPrice("AAPL", 152.75);
    }
}

interface Stock {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();
}

interface Observer {
    void update(String stockSymbol, double price);
}

class StockMarket implements Stock {
    private List<Observer> observers = new ArrayList<>();
    private String stockSymbol;
    private double price;
    public void setStockPrice(String stockSymbol, double price) {
        this.stockSymbol = stockSymbol;
        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 observer : observers) {
            observer.update(stockSymbol, price);
        }
    }
}
class MobileApp implements Observer {
    public void update(String stockSymbol, double price) {
        System.out.println("[Mobile App] " + stockSymbol + " price updated: $" +
price);
    }
}
class WebApp implements Observer {
    public void update(String stockSymbol, double price) {
        System.out.println("[Web App] " + stockSymbol + " price updated: $" + price);
    }
}
```

Output:



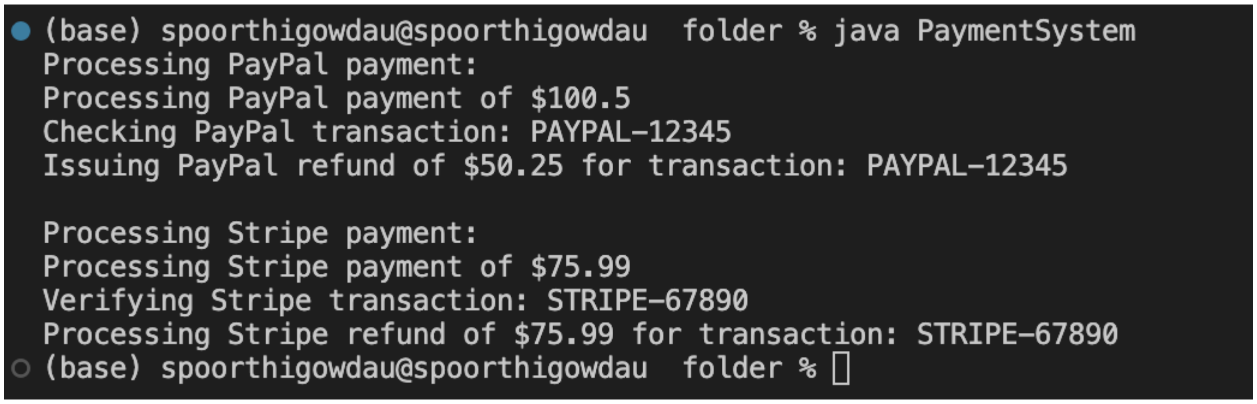
```
(base) spoorthigowdau@spoorthigowdau  folder % java StockMarketApp
[Mobile App] AAPL price updated: $150.5
[Web App] AAPL price updated: $150.5
[Mobile App] GOOGL price updated: $2750.0
[Web App] GOOGL price updated: $2750.0
[Mobile App] AAPL price updated: $152.75
```

Exercise 8: Implementing the Strategy Pattern

PaymentSystem.java

```
public class PaymentSystem {
    public static void main(String[] args) {
        PaymentContext context = new PaymentContext();

        context.setPaymentStrategy(new CreditCardPayment());
        context.executePayment(100.50);

        context.setPaymentStrategy(new PayPalPayment());
        context.executePayment(75.99);
    }
}

interface PaymentStrategy {
    void pay(double amount);
}

class CreditCardPayment implements PaymentStrategy {
    public void pay(double amount) {
        System.out.println("Paying $" + amount + " via Credit Card");
    }
}

class PayPalPayment implements PaymentStrategy {
    public void pay(double amount) {
        System.out.println("Paying $" + amount + " via PayPal");
    }
}

class PaymentContext {
    private PaymentStrategy paymentStrategy;

    public void setPaymentStrategy(PaymentStrategy strategy) {
        this.paymentStrategy = strategy;
    }

    public void executePayment(double amount) {
        paymentStrategy.pay(amount);
    }
}
```

Output:



```
● (base) spoorthigowdau@spoorthigowdau  folder % java PaymentSystem
  Processing PayPal payment:
  Processing PayPal payment of $100.5
  Checking PayPal transaction: PAYPAL-12345
  Issuing PayPal refund of $50.25 for transaction: PAYPAL-12345

  Processing Stripe payment:
  Processing Stripe payment of $75.99
  Verifying Stripe transaction: STRIPE-67890
  Processing Stripe refund of $75.99 for transaction: STRIPE-67890
○ (base) spoorthigowdau@spoorthigowdau  folder % 
```

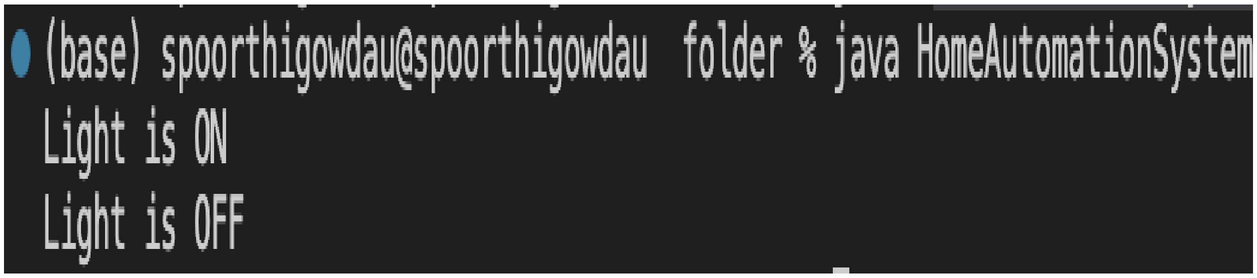
Exercise 9: Implementing the Command Pattern

HomeAutomationSystem.java

```java
public class HomeAutomationSystem {
   public static void main(String[] args) {
       Light livingRoomLight = new Light();

       Command lightOn = new LightOnCommand(livingRoomLight);
       Command lightOff = new LightOffCommand(livingRoomLight);

       RemoteControl remote = new RemoteControl();

       remote.setCommand(lightOn);
       remote.pressButton();

       remote.setCommand(lightOff);
       remote.pressButton();
   }
}

interface Command {
   void execute();
}

class LightOnCommand implements Command {
   private Light light;

   public LightOnCommand(Light light) {
       this.light = light;
   }

   public void execute() {
       light.turnOn();
   }
}

class LightOffCommand implements Command {
   private Light light;
```

```java
    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.turnOff();
    }
}
class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
}
class Light {
    public void turnOn() {
        System.out.println("Light is ON");
    }
    public void turnOff() {
        System.out.println("Light is OFF");
    }
}
```

Output:



```
(base) spoorthigowdau@spoorthigowdau  folder % java HomeAutomationSystem
Light is ON
Light is OFF
```

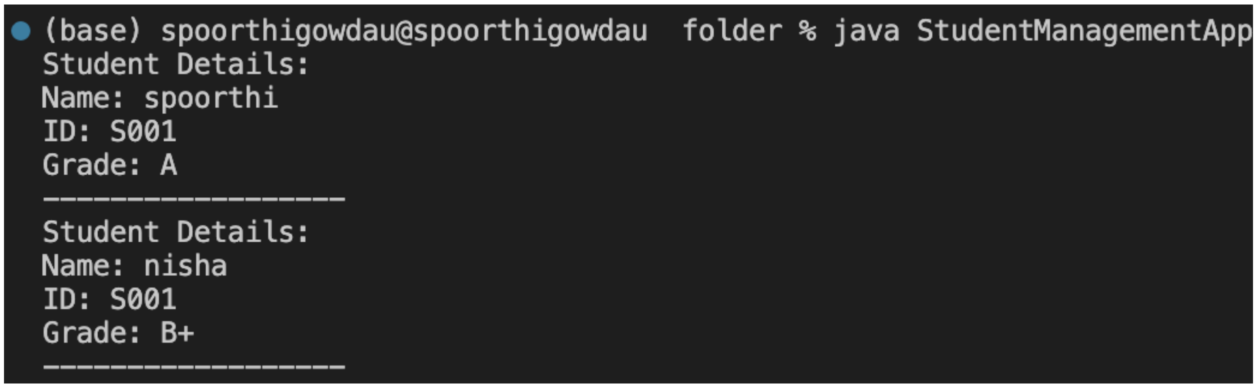
Exercise 10: Implementing the MVC Pattern

StudentManagementApp.java

```
public class StudentManagementApp {
   public static void main(String[] args) {
       Student model = new Student("spoorthi", "S001", "A");
       StudentView view = new StudentView();
       StudentController controller = new StudentController(model, view);

       controller.updateView();
       controller.setStudentName("nisha");
       controller.setStudentGrade("B+");
       controller.updateView();
   }
}

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 void setName(String name) { this.name = name; }
   public String getId() { return id; }
   public String getGrade() { return grade; }
   public void setGrade(String grade) { this.grade = grade; }
}

class StudentView {
   public void displayStudentDetails(String name, String id, String grade) {
       System.out.println("Student Details:");
       System.out.println("Name: " + name);
       System.out.println("ID: " + id);
```

```java
        System.out.println("Grade: " + grade);
        System.out.println("-------------------");
    }
}

class StudentController {
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }

    public void setStudentName(String name) { model.setName(name); }
    public void setStudentGrade(String grade) { model.setGrade(grade); }
    public void updateView() {
        view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());
    }
}
```

Output:



```
(base) spoorthigowdau@spoorthigowdau  folder % java StudentManagementApp
Student Details:
Name: spoorthi
ID: S001
Grade: A
-------------------
Student Details:
Name: nisha
ID: S001
Grade: B+
-------------------
```

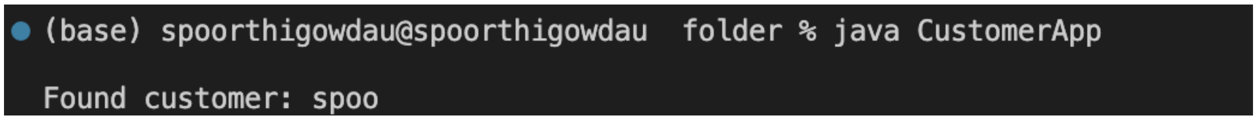
Exercise 11: Implementing Dependency Injection

CustomerApp.java

```java
public class CustomerApp {
   public static void main(String[] args) {
       CustomerRepository repository = new CustomerRepositoryImpl();
       CustomerService service = new CustomerService(repository);

       Customer customer = service.findCustomerById(123);
       System.out.println("Found customer: " + customer.getName());
   }
}

interface CustomerRepository {
   Customer findCustomerById(int id);
}

class CustomerRepositoryImpl implements CustomerRepository {
   public Customer findCustomerById(int id) {
       return new Customer(id, "spoo", "spoo@example.com");
   }
}

class CustomerService {
   private final CustomerRepository repository;

   public CustomerService(CustomerRepository repository) {
       this.repository = repository;
   }

   public Customer findCustomerById(int id) {
       return repository.findCustomerById(id);
   }
}

class Customer {
   private int id;
   private String name;
   private String email;
```

```java
    public Customer(int id, String name, String email) {
        this.id = id;
        this.name = name;
        this.email = email;
    }

    public int getId() { return id; }
    public String getName() { return name; }
    public String getEmail() { return email; }
}
```

Output:



```
● (base) spoorthigowdau@spoorthigowdau  folder % java CustomerApp
  Found customer: spoo
```