Week 1 :

A) Singleton pattern :

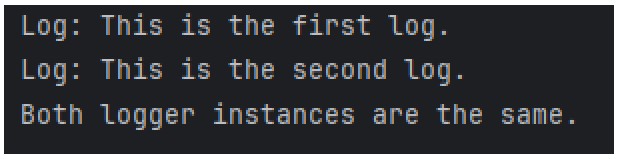
main.class :

```
public class Main {
  public static void main(String[] args) {
    Logger logger1 = Logger.getInstance();
    Logger logger2 = Logger.getInstance();
    logger1.log("This is the first log.");
    logger2.log("This is the second log.");
    if (logger1 == logger2) {
      System.out.println("Both logger instances are the same.");
    } else {
      System.out.println("Different instances exist!");}}}
```

Logger.class :

```
public class Logger {
  private static Logger instance;

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

OUTPUT:



```
Log: This is the first log.
Log: This is the second log.
Both logger instances are the same.
```

B) Factory Method Pattern :

Main class :

```
    public class Main {
    public static void main(String[] args) {
    DocumentFactory wordFactory = new WordDocumentFactory();
    Document word = wordFactory.createDocument();
    word.open();

    DocumentFactory pdfFactory = new PdfDocumentFactory();
    Document pdf = pdfFactory.createDocument();
    pdf.open();

    DocumentFactory excelFactory = new ExcelDocumentFactory();
    Document excel = excelFactory.createDocument();
    excel.open();
  }
}
```

Document.java:

```
public interface Document {
  void open();
}
```

WordDocument.java :

```
public class WordDocument implements Document {
  @Override
  public void open() {
    System.out.println("Opening Word document...");
  }
}
```

PdfDocument.java :

```
public class PdfDocument implements Document {
  @Override
  public void open() {
    System.out.println("Opening PDF document...");
  }
}
```

ExcelDocument.java:

```
public class ExcelDocument implements Document {
  @Override
  public void open() {
    System.out.println("Opening Excel document...");
  }
}
```

DocumentFactory.java:

```
public abstract class DocumentFactory {
  public abstract Document createDocument();
}
```

WordDocumentFactory.java:

```
public class WordDocumentFactory extends DocumentFactory {
  @Override
  public Document createDocument() {
    return new WordDocument();  // returns a WordDocument object
  }
}
```

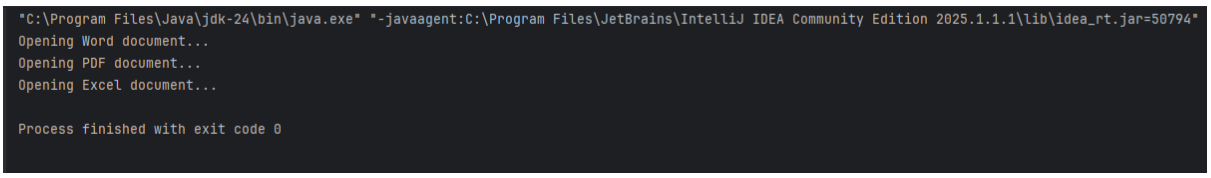
PdfDocumentFactory.java :

```
public class PdfDocumentFactory extends DocumentFactory {
  @Override
  public Document createDocument() {
    return new PdfDocument();  // returns a PdfDocument object
  }
}
```

ExcelDocumentFactory.java:

```
public class ExcelDocumentFactory extends DocumentFactory {
  @Override
  public Document createDocument() {
    return new ExcelDocument();  // returns an ExcelDocument object
  }
}
```

OUTPUT :



```
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.1.1\lib\idea_rt.jar=50794"
Opening Word document...
Opening PDF document...
Opening Excel document...

Process finished with exit code 0
```

C) Builder Pattern :

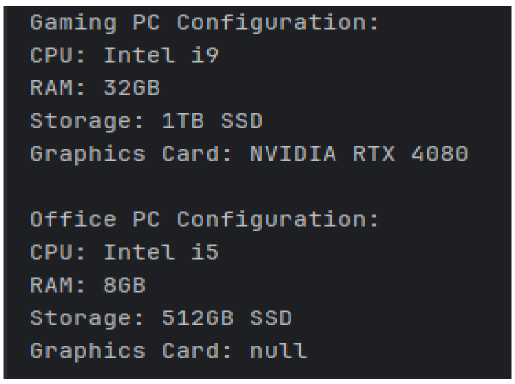
Main.java:

```
public class Main {

public static void main(String[] args) {

Computer gamingPC = new Computer.Builder()

.setCPU("Intel i9")

.setRAM("32GB")

.setStorage("1TB SSD")

.setGraphicsCard("NVIDIA RTX 4080")

.build();


System.out.println("Gaming PC Configuration:");

gamingPC.showSpecs();

 Computer officePC = new Computer.Builder()

.setCPU("Intel i5")

.setRAM("8GB")

.setStorage("512GB SSD")

.build();

System.out.println("\nOffice PC Configuration:");

officePC.showSpecs();}}
```

Computer.java:

```
public class Computer {

   private final String CPU;
   private final String RAM;
   private final String storage;
   private final String graphicsCard;

   private Computer(Builder builder) {
      this.CPU = builder.CPU;
      this.RAM = builder.RAM;
      this.storage = builder.storage;
      this.graphicsCard = builder.graphicsCard;   }
public void showSpecs() {
  System.out.println("CPU: " + CPU);
  System.out.println("RAM: " + RAM);
  System.out.println("Storage: " + storage);
  System.out.println("Graphics Card: " + graphicsCard);
}

  public static class Builder {
  private String CPU;
  private String RAM;
  private String storage;
  private String graphicsCard;

  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 Builder setGraphicsCard(String graphicsCard) {
    this.graphicsCard = graphicsCard;
    return this;
  }
  public Computer build() {
    return new Computer(this);}}}
```

OUTPUT:



```
Gaming PC Configuration:
CPU: Intel i9
RAM: 32GB
Storage: 1TB SSD
Graphics Card: NVIDIA RTX 4080

Office PC Configuration:
CPU: Intel i5
RAM: 8GB
Storage: 512GB SSD
Graphics Card: null
```

D) Adapter Pattern :

Paymentprocessor.java :

```
package adapter;

public interface PaymentProcessor {
  void processPayment(double amount);
}
```

Razorpay.java:

```
package adapter;

public class Razorpay {
  public void payViaRazor(double amt) {
    System.out.println("Paid ₹" + amt + " using Razorpay.");
  }
}
```

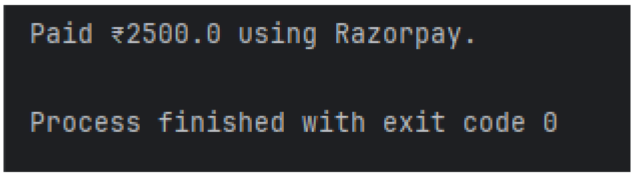
RazorpayAdapter.java:

```
package adapter;

public class RazorpayAdapter implements PaymentProcessor {
  private Razorpay razorpay = new Razorpay();

  @Override
  public void processPayment(double amount) {
    razorpay.payViaRazor(amount);
  }
}
```

AdapterTest.java:

```
package adapter;

public class AdapterTest {
  public static void main(String[] args) {
    PaymentProcessor processor = new RazorpayAdapter();
    processor.processPayment(2500.0);
    }
}
```

OUTPUT:



```
Paid ₹2500.0 using Razorpay.

Process finished with exit code 0
```

E) Decorator Pattern :

DecoratorTest.java:

```
package decorator;

public class DecoratorTest {
    public static void main(String[] args) {
        // Step-by-step decorator wrapping
        Notifier notifier = new EmailNotifier();
        notifier = new SMSNotifierDecorator(notifier);
        notifier = new SlackNotifierDecorator(notifier);

        notifier.send("System alert: CPU usage high!");
    }
}
```

SlackNotifierDecorator.java:

```
package decorator;

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

    @Override
    public void send(String message) {
        super.send(message);
        sendSlack(message);     }

    private void sendSlack(String message) {
        System.out.println("Slack: " + message);  }
}
```

SMSNotifierDecorator.java:

```
package decorator;

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

    @Override
    public void send(String message) {
        super.send(message);
        sendSMS(message);
    }

    private void sendSMS(String message) {
        System.out.println("SMS: " + message);
    }
}
```

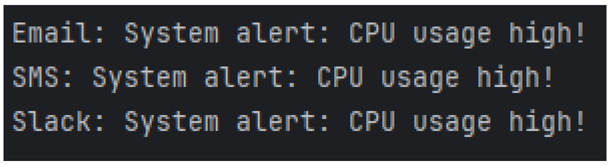
NotifierDecorator.java:

```
package decorator;

public abstract class NotifierDecorator implements Notifier {
    protected Notifier wrappee;

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

    @Override
    public void send(String message) {
        wrappee.send(message);    }
}
```

EmailNotifier.java:

```
package decorator;

public class EmailNotifier implements Notifier {
    @Override
    public void send(String message) {
        System.out.println("Email: " + message);  }
}
```

OUTPUT:



```
Email: System alert: CPU usage high!
SMS: System alert: CPU usage high!
Slack: System alert: CPU usage high!
```

F) Proxy Pattern:

ProxyPatternTest.java:

```
package proxy;

public class ProxyPatternTest {
    public static void main(String[] args) {
        Image img1 = new ProxyImage("sunset.jpg");

        System.out.println("First display:");
        img1.display();  // Loads and displays

        System.out.println("\nSecond display:");
        img1.display();  // Uses cached RealImage
    }
}
```

Image.java:

```
package proxy;

public interface Image {
    void display();
}
```

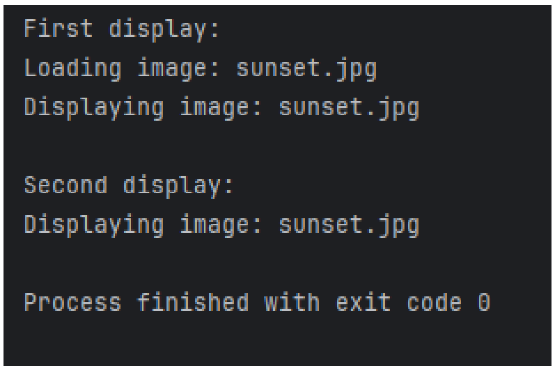
RealImage.java:

```
package proxy;

public class RealImage implements Image {
    private String filename;

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

    private void loadFromDisk() {
        System.out.println("Loading image: " + filename);
    }

    @Override
    public void display() {
        System.out.println("Displaying image: " + filename);
    }
}
```

ProxyImage.java:

```
package proxy;

public class ProxyImage implements Image {
    private RealImage realImage;
    private String filename;

    public ProxyImage(String filename) {
        this.filename = filename;
    }

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

OUTPUT:



```
First display:
Loading image: sunset.jpg
Displaying image: sunset.jpg

Second display:
Displaying image: sunset.jpg

Process finished with exit code 0
```

G) Observer Pattern:

Stock.java:

```
package observer;

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

StockMarket.java:

```
package observer;

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

public class StockMarket implements Stock {
   private List<Observer> observers = new ArrayList<>();
   private String stockName;
   private double price;

   public StockMarket(String stockName) {
      this.stockName = stockName;
   }

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

   public double getPrice() {
      return price;
   }

   public String getStockName() {
      return stockName;
   }

   @Override
   public void registerObserver(Observer o) {
      observers.add(o);
```

```
  }

  @Override
  public void removeObserver(Observer o) {
    observers.remove(o);
  }

  @Override
  public void notifyObservers() {
    for (Observer o : observers) {
      o.update(stockName, price);
    }
  }
}
```

Observer.java:

```
package observer;

public interface Observer {
  void update(String stockName, double price);
}
```

MobileApp.java:

```
package observer;

public class MobileApp implements Observer {
  private String user;

  public MobileApp(String user) {
    this.user = user;
  }

  @Override
  public void update(String stockName, double price) {
    System.out.println("Mobile App [" + user + "] - " + stockName + " price
updated to ₹" + price);
  }
}
```

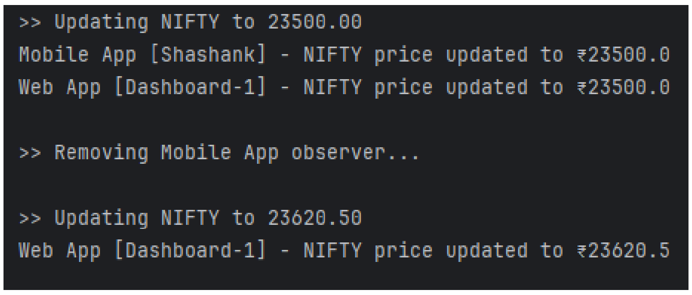
WebApp.java:

```
package observer;

public class WebApp implements Observer {
  private String dashboard;
  public WebApp(String dashboard) {
    this.dashboard = dashboard;
  }

  @Override
  public void update(String stockName, double price) {
    System.out.println("Web App [" + dashboard + "] - " + stockName + " price
updated to ₹" + price);
  }
}
```

ObserverPatternTest.java:

```
package observer;

public class ObserverPatternTest {
  public static void main(String[] args) {
    StockMarket niftyStock = new StockMarket("NIFTY");

    Observer mobileUser = new MobileApp("Shashank");
    Observer webUser = new WebApp("Dashboard-1");

    niftyStock.registerObserver(mobileUser);
    niftyStock.registerObserver(webUser);

    System.out.println(">> Updating NIFTY to 23500.00");
    niftyStock.setPrice(23500.00);

    System.out.println("\n>> Removing Mobile App observer...");
    niftyStock.removeObserver(mobileUser);

    System.out.println("\n>> Updating NIFTY to 23620.50");
    niftyStock.setPrice(23620.50);
  }
}
```

OUTPUT:



```
>> Updating NIFTY to 23500.00
Mobile App [Shashank] - NIFTY price updated to ₹23500.0
Web App [Dashboard-1] - NIFTY price updated to ₹23500.0

>> Removing Mobile App observer...

>> Updating NIFTY to 23620.50
Web App [Dashboard-1] - NIFTY price updated to ₹23620.5
```

H) Strategy Pattern:

PaymentStrategy.java:

```
package strategy;

public interface PaymentStrategy {
  void pay(double amount);
}
```

CreditCardPayment.java:

```
package strategy;

public class CreditCardPayment implements PaymentStrategy {
  private String cardNumber;
  private String cardHolder;

  public CreditCardPayment(String cardNumber, String cardHolder) {
    this.cardNumber = cardNumber;
    this.cardHolder = cardHolder;
  }

  @Override
  public void pay(double amount) {
    System.out.println("Paid ₹" + amount + " using Credit Card [Holder: " +
cardHolder + "]");
  }
}
```

PayPalPayment.java:

```
package strategy;

public class PayPalPayment implements PaymentStrategy {
  private String email;

  public PayPalPayment(String email) {
    this.email = email;
  }

  @Override
  public void pay(double amount) {
    System.out.println("Paid ₹" + amount + " using PayPal [Email: " + email + "]");
    }
}
```

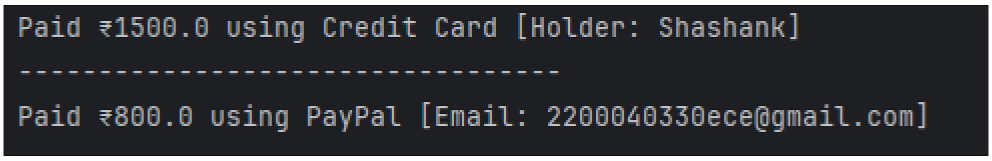
PaymentContext.java:

```
package strategy;

public class PaymentContext {
    private PaymentStrategy paymentStrategy;


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

    public void processPayment(double amount) {
        if (paymentStrategy == null) {
            System.out.println("No payment method selected.");
        } else {
            paymentStrategy.pay(amount);
        }
    }
}
```

StrategyPatternTest.java:

```
package strategy;

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



        context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-
3456", "Shashank"));
        context.processPayment(1500.00);

        System.out.println("----------------------------------");


        context.setPaymentStrategy(new
PayPalPayment("2200040330ece@gmail.com"));
        context.processPayment(800.00);
    }
```

OUTPUT:



```
Paid ₹1500.0 using Credit Card [Holder: Shashank]
----------------------------------
Paid ₹800.0 using PayPal [Email: 2200040330ece@gmail.com]
```

I) Command Pattern:

Command.java:

```
package command;

public interface Command {
  void execute();
}
```

LightOnCommand.java:

```
package command;

public class LightOnCommand implements Command {
  private Light light;

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

  @Override
  public void execute() {
    light.turnOn();
  }
}
```

LightOffCommand.java:

```
package command;

public class LightOffCommand implements Command {
  private Light light;

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

  @Override
  public void execute() {
        light.turnOff();
    }
}
```

RemoteControl.java:

```
package command;

public class RemoteControl {
    private Command command;

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

    public void pressButton() {
        command.execute();
    }
}
```

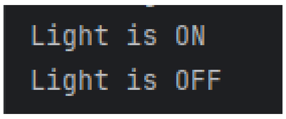
Light.java:

```
package command;

public class Light {
    public void turnOn() {
        System.out.println("Light is ON");
    }

    public void turnOff() {
        System.out.println("Light is OFF");
    }
}
```

CommandPatternTest.java:

```
package command;

public class CommandPatternTest {
    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();

    }
}
```

OUTPUT:



```
Light is ON
Light is OFF
```

J) Dependency Injection;

CustomerRepository.java:

```
package di;

public interface CustomerRepository {
    String findCustomerById(int id);
}
```

CustomerRepositoryImpl.java:

```
package di;

public class CustomerRepositoryImpl implements CustomerRepository {

    @Override
    public String findCustomerById(int id) {
        // Simulate customer data
        if (id == 1) {
            return "Customer[id=1, name=Shashank]";
        } else {
            return "Customer not found";
        }
    }
}
```

CustomerService.java:

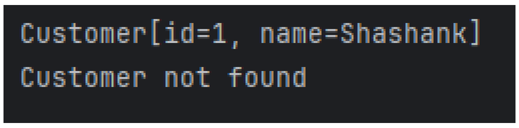
```
package di;
```

```java
public class CustomerService {

   private CustomerRepository repository;

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

   public void displayCustomer(int id) {
      String customer = repository.findCustomerById(id);
      System.out.println(customer);
   }
}
```

DependencyInjectionTest.java:

```java
package di;

public class DependencyInjectionTest {
   public static void main(String[] args) {
      // Inject dependency
      CustomerRepository repository = new CustomerRepositoryImpl();
      CustomerService service = new CustomerService(repository);

      service.displayCustomer(1);
      service.displayCustomer(2);
   }
}
```

OUTPUT:



```
Customer[id=1, name=Shashank]
Customer not found
```