SLIP 1

Q.1 Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

File 1: `LowerCaseReader.java`

```java
import java.io.FilterReader;
import java.io.IOException;
import java.io.Reader;

public class LowerCaseReader extends FilterReader {
    protected LowerCaseReader(Reader in) {
        super(in);
    }

    @Override
    public int read() throws IOException {
        int c = super.read();
        if (c != -1) {
            c = Character.toLowerCase((char) c);
        }
        return c;
    }

    @Override
    public int read(char[] cbuf) throws IOException {
        int bytesRead = super.read(cbuf);
        for (int i = 0; i < bytesRead; i++) {
            cbuf[i] = Character.toLowerCase(cbuf[i]);
        }
        return bytesRead;
    }

    @Override
    public int read(char[] cbuf, int off, int len) throws IOException {
        int bytesRead = super.read(cbuf, off, len);
        for (int i = off; i < off + bytesRead; i++) {
            cbuf[i] = Character.toLowerCase(cbuf[i]);
        }
        return bytesRead;
    }
}
```

File 2: `Main.java`

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.StringReader;

public class Main {
    public static void main(String[] args) {
        String input = "HELLO WORLD! THIS IS A TEST STRING.";

        try (LowerCaseReader reader = new LowerCaseReader(new
StringReader(input));
             BufferedReader bufferedReader = new BufferedReader(reader)) {

            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Just run second file only.

Q.2 Write a program to sense the available networks using Arduino

Create a New ESP32 Project

- On the Wokwi simulator, select the ESP32 Dev Board when creating your new project.

2. Use the ESP32 WiFi Scan Code

Paste this code in the editor (it works for ESP32):

```
#include <WiFi.h>  // ESP32 uses WiFi.h

void setup() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
  Serial.println("Scanning for available networks...");
}

void loop() {
  int n = WiFi.scanNetworks();
  Serial.println("Scan complete.");
  if (n == 0) {
    Serial.println("No networks found.");
  } else {
    Serial.print(n);
    Serial.println(" network(s) found:");
    for (int i = 0; i < n; ++i) {
      Serial.print(i+1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (Signal: ");
      Serial.print(WiFi.RSSI(i));
      Serial.println(" dBm)");
      delay(10);
    }
  }
  Serial.println("");
  delay(10000);
}
```

O/p

```
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,w
p_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1156
load:0x40078000,len:11456
ho 0 tail 12 room 4
load:0x40080400,len:2972
entry 0x400805dc
Scanning for available networks...
Scan complete.
1 network(s) found:
```

```
1: Wokwi-GUEST (Signal: -74 dBm)

Scan complete.
1 network(s) found:
1: Wokwi-GUEST (Signal: -82 dBm)

Scan complete.
1 network(s) found:
1: Wokwi-GUEST (Signal: -86 dBm)
```

Slip 2

Q.1 Write a Java Program to implement Singleton pattern for multithreading

`Singleton.java`

```java
public class Singleton {
    private static Singleton instance;

    private Singleton() {
        // Private constructor
    }

    private static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    public static Singleton getInstanceSync() {
        return getInstance();
    }

    public void doSomething() {
        System.out.println("Doing something... Thread: " +
Thread.currentThread().getName());
    }
}
```

`MainThread.java`

```java
public class MainThread {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                Singleton singleton = Singleton.getInstanceSync();
                singleton.doSomething();
            }
        });

        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                Singleton singleton = Singleton.getInstanceSync();
                singleton.doSomething();
            }
        });

        t1.start();
        t2.start();
    }
}
```

o/p:

```
Doing something... Thread: Thread-1

Doing something... Thread: Thread-0

PS C:\Users\ADMIN\Desktop\SADP Slip Solution>
```

Q.2 Write a program to measure the distance using ultrasonic sensor and make LED blink using Arduino.

select the Arduino Uno board.

1. Add the HC-SR04 Ultrasonic Sensor

- In the Wokwi editor, click the "+" (add component) button.
- Search for "HC-SR04" and add it to your schematic.

2. Wire the Sensor Properly

- Connect the pins as follows:
- VCC → Arduino 5V
- GND → Arduino GND
- Trig → Arduino digital pin 9
- Echo → Arduino digital pin 10

Code

```
#define TRIG_PIN 9

#define ECHO_PIN 10

#define LED_PIN 13


void setup() {

  pinMode(TRIG_PIN, OUTPUT);

  pinMode(ECHO_PIN, INPUT);

  pinMode(LED_PIN, OUTPUT);

  Serial.begin(9600);

}


void loop() {

  long duration, distance;


  // Clear trigPin

  digitalWrite(TRIG_PIN, LOW);

  delayMicroseconds(2);


  // Set trigPin HIGH for 10 microseconds

  digitalWrite(TRIG_PIN, HIGH);

  delayMicroseconds(10);

  digitalWrite(TRIG_PIN, LOW);


  // Read echoPin
  duration = pulseIn(ECHO_PIN, HIGH);

  // Calculate distance in cm
  distance = duration * 0.034 / 2;
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  // Blink LED if distance < 15cm
  if (distance < 15) {
    digitalWrite(LED_PIN, HIGH);
    delay(250);
    digitalWrite(LED_PIN, LOW);
    delay(250);
  } else {
    digitalWrite(LED_PIN, LOW);
  }

  delay(500);
}
```

Slip 3

Q.1 Write a JAVA Program to implement built-in support (java.util.Observable) Weather station with members temperature, humidity, pressure and methods mesurmentsChanged(), setMesurment(), getTemperature(), getHumidity(), getPressure()

```
import java.util.Observable;
import java.util.Observer;
```

```java
@SuppressWarnings("deprecation")
class WeatherStation extends Observable {
    private float temperature;
    private float humidity;
 private float pressure;

    public void setMeasurements(float temperature, float humidity, float
pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }

    public void measurementsChanged() {
        setChanged();
        notifyObservers();
    }

    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }
}

@SuppressWarnings("deprecation")
class CurrentConditionsDisplay implements Observer, DisplayElement {
    private float temperature;
    private float humidity;
    public CurrentConditionsDisplay(Observable observable) {
        observable.addObserver(this);
    }

    public void update(Observable observable, Object arg) {
        if (observable instanceof WeatherStation) {
            WeatherStation weatherStation = (WeatherStation) observable;
            this.temperature = weatherStation.getTemperature();
            this.humidity = weatherStation.getHumidity();
            display();
        }
    }

    public void display() {
        System.out.println("Current conditions: " + temperature + "F degrees 
and " + humidity + 
"% humidity");
    }
}

interface DisplayElement {
    void display();
}

public class WeatherStationTest {
    public static void main(String[] args) {
        WeatherStation weatherStation = new WeatherStation();
        new CurrentConditionsDisplay(weatherStation);

        weatherStation.setMeasurements(80, 65, 30.4f);
        weatherStation.setMeasurements(82, 70, 29.2f);
        weatherStation.setMeasurements(78, 90, 29.2f);
    }
}
```

Q. 2 Write a program to detects the vibration of an object with sensor using Arduino.

- Board: Select Arduino Uno

2. Add Components

- Button: Connect one side to Arduino digital pin 2 and other side to GND
- LED: Connect anode (+) to pin 13 (through resistor to GND)

Code:

```
#define SENSOR_PIN 2  // Button simulates vibration sensor DO

#define LED_PIN 13    // LED pin


void setup() {

  pinMode(SENSOR_PIN, INPUT_PULLUP);  // Enable pull-up for button

  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int vibration = digitalRead(SENSOR_PIN); // LOW when button pressed

  if (vibration == LOW) {  // Button pressed = "vibration detected"
    Serial.println("Vibration detected!");
    digitalWrite(LED_PIN, HIGH); // Turn on LED
  } else {
    Serial.println("No vibration.");
    digitalWrite(LED_PIN, LOW);  // Turn off LED
  }

  delay(200);
}
```

Slip 4

Q.1 Write a Java Program to implement Factory method for Pizza Store with createPizza(), orederPizza(), prepare(), Bake(), cut(), box(). Use this to create variety of pizza's like NyStyleCheesePizza, ChicagoStyleCheesePizza etc.

PizzaStoreTest.java

```
// Abstract Product
abstract class Pizza {
    public abstract void prepare();
    public abstract void bake();
    public abstract void cut();
    public abstract void box();
}

// Concrete Products
class NYStyleCheesePizza extends Pizza {
    @Override
    public void prepare() { System.out.println("Preparing NY Style Cheese 
Pizza"); }
```

```
    @Override
    public void bake()    { System.out.println("Baking NY Style Cheese 
Pizza"); }
    @Override
    public void cut()     { System.out.println("Cutting NY Style Cheese 
Pizza"); }
    @Override
    public void box()     { System.out.println("Boxing NY Style Cheese 
Pizza"); }
}

class ChicagoStyleCheesePizza extends Pizza {
    @Override
    public void prepare() { System.out.println("Preparing Chicago Style Cheese 
Pizza"); }
    @Override
    public void bake()    { System.out.println("Baking Chicago Style Cheese 
Pizza"); }
    @Override
    public void cut()     { System.out.println("Cutting Chicago Style Cheese 
Pizza"); }
    @Override
    public void box()     { System.out.println("Boxing Chicago Style Cheese 
Pizza"); }
}

// Creator (Factory)
abstract class PizzaStore {
    // Factory method
    public abstract Pizza createPizza(String type);

    // Template method for ordering
    public void orderPizza(String type) {
        Pizza pizza = createPizza(type);
        if (pizza == null) {
            System.out.println("Unknown pizza type: " + type);
            return;
        }
        pizza.prepare();
        pizza.bake();
        pizza.cut();
        pizza.box();
    }
}

// Concrete Creators
class NYPizzaStore extends PizzaStore {
    @Override
```

```java
    public Pizza createPizza(String type) {
        if (type.equalsIgnoreCase("cheese")) {
            return new NYStyleCheesePizza();
        }
        // more NY styles can be added here
        return null;
    }
}

class ChicagoPizzaStore extends PizzaStore {
    @Override
    public Pizza createPizza(String type) {
        if (type.equalsIgnoreCase("cheese")) {
            return new ChicagoStyleCheesePizza();
        }
        // more Chicago styles can be added here
        return null;
    }
}

// Test main
public class PizzaStoreTest {
    public static void main(String[] args) {
        PizzaStore nyStore = new NYPizzaStore();
        PizzaStore chicagoStore = new ChicagoPizzaStore();

        System.out.println("-- NY Pizza Store Order --");
        nyStore.orderPizza("cheese");

        System.out.println("\n-- Chicago Pizza Store Order --");
        chicagoStore.orderPizza("cheese");
    }
}
```

Q.2 Write a program to sense a finger when it is placed on the board Arduino.

Instructions for Sensing a Finger (Touch Simulation) With Arduino in Wokwi

1. Select Board & Components

- Board: Select Arduino Uno
- Sensor: Add a Button (to simulate the finger press)
- LED (optional, for visual feedback)

2. Wiring Setup

- Button:
- One side → Arduino digital pin 2
- Other side → GND
- LED:
- Anode (+) → pin 13 (with 220Ω resistor to GND)
- Cathode (–) → GND

Code :

```
#define TOUCH_PIN 2  // Button pin simulating touch
#define LED_PIN 13   // LED pin


void setup() {
  pinMode(TOUCH_PIN, INPUT_PULLUP); // Enable internal pull-up resistor
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
}


void loop() {
  int fingerDetected = digitalRead(TOUCH_PIN); // LOW if button pressed
(finger "touched")

  if (fingerDetected == LOW) {

    Serial.println("Finger detected on board!");

    digitalWrite(LED_PIN, HIGH); // LED ON

  } else {

    Serial.println("No finger detected.");

    digitalWrite(LED_PIN, LOW);  // LED OFF

  }

  delay(200);

}
```

Slip 5

Q.1 Write a Java Program to implement Adapter pattern for Enumeration iterator

EnumerationAdapterTest.java

```java
import java.util.*;

// Adapter class: adapts Enumeration to Iterator
class EnumerationIteratorAdapter<T> implements Iterator<T> {
    private Enumeration<T> enumeration;

    public EnumerationIteratorAdapter(Enumeration<T> enumeration) {
        this.enumeration = enumeration;
    }

    @Override
    public boolean hasNext() {
        return enumeration.hasMoreElements();
    }

    @Override
    public T next() {
        return enumeration.nextElement();
    }

    @Override
    public void remove() {
        throw new UnsupportedOperationException("Remove not supported by 
Enumeration");
    }
}

// Demo/test
public class EnumerationAdapterTest {
    public static void main(String[] args) {
        // Create an old-style Vector
        Vector<String> vector = new Vector<>();
        vector.add("Apple");
        vector.add("Banana");
        vector.add("Cherry");

        // Get Enumeration
        Enumeration<String> enumeration = vector.elements();

        // Use adapter to turn Enumeration into Iterator
        Iterator<String> iterator = new 
EnumerationIteratorAdapter<>(enumeration);

        System.out.println("Iterating using Iterator (via Adapter):");
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

Q.2 Write a program to connect with the available Wi-Fi using Arduino.

Code :-

```
#include <WiFi.h>

const char* ssid     = "Wokwi-GUEST";  // WiFi network name (use simulated 
network in Wokwi)
const char* password = "";             // No password for Wokwi-GUEST

void setup() {
  Serial.begin(115200);
  Serial.println();
  Serial.print("Connecting to WiFi: ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  // Wait until connected
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.println("WiFi connected!");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Add your main code here
}
```

Slip 6

Q.1 Write a Java Program to implement command pattern to test Remote Control

RemoteControlTest.java

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

```java
// Receiver classes
class Light {
    public void on() { System.out.println("Light is ON"); }
    public void off() { System.out.println("Light is OFF"); }
}

class GarageDoor {
    public void up() { System.out.println("Garage Door is UP"); }
    public void down() { System.out.println("Garage Door is DOWN"); }
}

// Concrete Command classes
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 GarageDoorUpCommand implements Command {
    private GarageDoor garageDoor;
    public GarageDoorUpCommand(GarageDoor garageDoor) { this.garageDoor =
garageDoor; }
    public void execute() { garageDoor.up(); }
}

class GarageDoorDownCommand implements Command {
    private GarageDoor garageDoor;
    public GarageDoorDownCommand(GarageDoor garageDoor) { this.garageDoor =
garageDoor; }
    public void execute() { garageDoor.down(); }
}

// Invoker: RemoteControl
class RemoteControl {
    private Command slot; // Only one slot for demo
    public void setCommand(Command command) { this.slot = command; }
    public void pressButton() { slot.execute(); }
}

// Test main
public class RemoteControlTest {
    public static void main(String[] args) {
        RemoteControl remote = new RemoteControl();

        Light light = new Light();
        GarageDoor garageDoor = new GarageDoor();

        Command lightOn = new LightOnCommand(light);
        Command lightOff = new LightOffCommand(light);
        Command garageUp = new GarageDoorUpCommand(garageDoor);
        Command garageDown = new GarageDoorDownCommand(garageDoor);

        System.out.println("Testing Remote Control:");

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

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

        remote.setCommand(garageUp);
        remote.pressButton();

        remote.setCommand(garageDown);
        remote.pressButton();
    }
}
```

Q.2 Write a program to get temperature notification using Arduino.

Temperature Notification Using Arduino in Wokwi

1. Select Board & Components

- Board: Select Arduino Uno
- Sensor: Add DHT22 or DHT11 temperature and humidity sensor
  (Search “DHT22” or “DHT11” and add one)
- LED or Buzzer (for notification)

- 220Ω resistor (for LED)

---

2. Wiring Setup

- DHT22/DHT11 Sensor:
- VCC → Arduino 5V
- GND → Arduino GND
- Data → Arduino digital pin 2
- LED:
- Anode (+) → Arduino pin 13 (with resistor to GND)
- Cathode (–) → GND

Code

```
#include "DHT.h"


#define DHTPIN 2      // Pin connected to the data pin of DHT22

#define DHTTYPE DHT22 // DHT22 sensor type

#define LED_PIN 13    // Notification LED pin


DHT dht(DHTPIN, DHTTYPE);


void setup() {

  Serial.begin(9600);

  dht.begin();

  pinMode(LED_PIN, OUTPUT);

}


void loop() {

  float temp = dht.readTemperature();


  if (isnan(temp)) {
    Serial.println("Failed to read from DHT22 sensor!");
    digitalWrite(LED_PIN, LOW);
    return;
  }

  Serial.print("Temperature: ");
  Serial.print(temp);
  Serial.println(" °C");

  // Notification: If temperature > 30 °C
  if (temp > 30) {
    Serial.println("Temperature HIGH! Notification triggered.");
    digitalWrite(LED_PIN, HIGH); // Turn LED on
  } else {
    digitalWrite(LED_PIN, LOW);  // Turn LED off
  }

  delay(2000); // Wait 2 seconds before next reading
}
```

Slip 7

Q.1 Write a Java Program to implement undo command to test Ceiling fan.

`CeilingFanUndoTest.java`

```java
import java.util.Stack;

// Command interface
interface Command {
    void execute();
    void undo();
}

// Receiver: CeilingFan
class CeilingFan {
    private int speed; // 0 = off

    public void setSpeed(int speed) {
        this.speed = speed;
        System.out.println("Ceiling fan speed set to " + speed);
    }
    public int getSpeed() { return speed; }
}

// Concrete command to set speed
class CeilingFanSpeedCommand implements Command {
    private CeilingFan ceilingFan;
    private int prevSpeed;
    private int newSpeed;

    public CeilingFanSpeedCommand(CeilingFan ceilingFan, int newSpeed) {
        this.ceilingFan = ceilingFan;
        this.newSpeed = newSpeed;
    }

    public void execute() {
        prevSpeed = ceilingFan.getSpeed();
        ceilingFan.setSpeed(newSpeed);
    }

    public void undo() {
        ceilingFan.setSpeed(prevSpeed);
        System.out.println("(undo to speed " + prevSpeed + ")");
    }
}

// Undo manager (Invoker)
class UndoManager {
    private Stack<Command> history = new Stack<>();

    public void executeCommand(Command cmd) {
        cmd.execute();
        history.push(cmd);
    }

    public void undo() {
        if(!history.isEmpty()) {
            Command cmd = history.pop();
            cmd.undo();
        } else {
            System.out.println("Nothing to undo!");
        }
    }
}

// Test class
public class CeilingFanUndoTest {
    public static void main(String[] args) {
        CeilingFan fan = new CeilingFan();
        UndoManager manager = new UndoManager();

        manager.executeCommand(new CeilingFanSpeedCommand(fan, 1));
        manager.executeCommand(new CeilingFanSpeedCommand(fan, 2));
        manager.executeCommand(new CeilingFanSpeedCommand(fan, 3));

        manager.undo();
        manager.undo();
        manager.undo();
        manager.undo(); // nothing left to undo
    }
}
```

Q.2 Write a program for LDR to vary the light intensity of LED using Arduino.

LDR-controlled LED Brightness (Arduino Uno & Wokwi)

1. Select Board & Components

- Arduino Uno (board)
- LDR (Light Dependent Resistor, search "LDR" or "photoresistor")
- 10kΩ Resistor (for voltage divider)
- LED
- 220Ω Resistor (for LED)

---

2. Wiring Setup

- LDR and 10kΩ resistor form a voltage divider:
- One end of LDR → 5V
- Other end of LDR → A0 (Analog input on Arduino)

- Connect A0 to one end of the 10kΩ resistor
- Other end of 10kΩ resistor → GND
    - LED:
        - Anode (+) → Arduino pin 9 (PWM output)
        - Cathode (–) → GND (through 220Ω resistor)

Code

```
#define LDR_PIN    A0   // Analog pin connected to LDR
#define LED_PIN    9    // PWM pin for LED

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  int ldrValue = analogRead(LDR_PIN);  // Read LDR value (0-1023)
  int ledBrightness = map(ldrValue, 0, 1023, 0, 255);  // Scale for PWM

  analogWrite(LED_PIN, ledBrightness);            // Set LED brightness

  Serial.print("LDR Value: ");
  Serial.print(ldrValue);
  Serial.print(" | LED Brightness: ");
  Serial.println(ledBrightness);

  delay(200);
}
```

Slip 8

Q. 1 Write a Java Program to implement State Pattern for Gumball Machine. Create instance variable that holds current state from there, we just need to handle all actions, behaviors and state transition that can happen

GumballMachine.java

```
// State interface
interface State {
    void insertQuarter();
    void ejectQuarter();
    void turnCrank();
    void dispense();
}

// Concrete States
class NoQuarterState implements State {
    private GumballMachine gumballMachine;
    public NoQuarterState(GumballMachine gm) { this.gumballMachine = gm; }
    public void insertQuarter() {
        System.out.println("You inserted a quarter");
        gumballMachine.setState(gumballMachine.getHasQuarterState());
    }
    public void ejectQuarter() { System.out.println("No quarter to eject"); }
    public void turnCrank() { System.out.println("You turned, but there's no
quarter"); }
    public void dispense() { System.out.println("Pay first!"); }
}

class HasQuarterState implements State {
    private GumballMachine gumballMachine;
    public HasQuarterState(GumballMachine gm) { this.gumballMachine = gm; }
    public void insertQuarter() { System.out.println("Can't insert another
quarter"); }
    public void ejectQuarter() {
        System.out.println("Quarter returned");
        gumballMachine.setState(gumballMachine.getNoQuarterState());
    }
    public void turnCrank() {
        System.out.println("You turned the crank...");
        gumballMachine.setState(gumballMachine.getSoldState());
    }
    public void dispense() { System.out.println("Turn crank first"); }
```

```java
}

class SoldState implements State {
    private GumballMachine gumballMachine;
    public SoldState(GumballMachine gm) { this.gumballMachine = gm; }
    public void insertQuarter() { System.out.println("Please wait, dispensing 
gumball"); }
    public void ejectQuarter() { System.out.println("Already turned crank"); }
    public void turnCrank() { System.out.println("Turning twice doesn't get 
you another gumball!"); }
    public void dispense() {
        gumballMachine.releaseBall();
        if (gumballMachine.getCount() > 0) {
            gumballMachine.setState(gumballMachine.getNoQuarterState());
        } else {
            System.out.println("Oops, out of gumballs!");
            gumballMachine.setState(gumballMachine.getSoldOutState());
        }
    }
}

class SoldOutState implements State {
    private GumballMachine gumballMachine;
    public SoldOutState(GumballMachine gm) { this.gumballMachine = gm; }
    public void insertQuarter() { System.out.println("Sold out!"); }
    public void ejectQuarter() { System.out.println("Sold out, no quarter!"); 
}
    public void turnCrank() { System.out.println("No gumballs"); }
    public void dispense() { System.out.println("No gumballs dispensed"); }
}

// Context class
public class GumballMachine {
    private State soldOutState;
    private State noQuarterState;
    private State hasQuarterState;
    private State soldState;

    private State state;
    private int count = 0;

    public GumballMachine(int count) {
        soldOutState = new SoldOutState(this);
        noQuarterState = new NoQuarterState(this);
        hasQuarterState = new HasQuarterState(this);
        soldState = new SoldState(this);
        this.count = count;
        state = (count > 0) ? noQuarterState : soldOutState;
    }

    public void insertQuarter()   { state.insertQuarter();    }
    public void ejectQuarter()    { state.ejectQuarter();     }
    public void turnCrank()       { state.turnCrank(); state.dispense(); }

    void setState(State s)        { state = s; }
    void releaseBall() {
        if(count > 0) {
            System.out.println("A gumball comes rolling out the slot...");
            count -= 1;
        }
    }
    int getCount()               { return count; }
    State getSoldOutState()      { return soldOutState; }
    State getNoQuarterState()    { return noQuarterState; }
    State getHasQuarterState()   { return hasQuarterState; }
    State getSoldState()         { return soldState; }

    public String toString() {
        return "GumballMachine [gumballs left: " + count + "]";
    }

    // Test driver
    public static void main(String[] args) {
        GumballMachine gumballMachine = new GumballMachine(2);

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.ejectQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);

        gumballMachine.insertQuarter();
        gumballMachine.turnCrank();

        System.out.println(gumballMachine);
    }
}
```

Q.2 Start Raspberry Pi and execute various Linux commands in command terminal window: ls, cd, touch, mv, rm, man, mkdir, rmdir, tar, gzip, cat, more, less, ps, sudo, cron, chown, chgrp, pingetc.

o/p

Sign up on Webminal.org → log in → start typing Linux commands in the web terminal.

Slip 9

Q.1 Design simple HR Application using Spring Framework

Q.2 Write python programs on Pi : a) Read your name and print Hello message with name b) Read two numbers and print their sum, difference, product and division. c) Word and character count of a given string. [15 M] d) Area of a given shape (rectangle, triangle and circle) reading shape and appropriate values from standard input.

a) Read your name and print Hello message

python
```python
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

---

b) Read two numbers and print their sum, difference, product, and division

python
```python
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))

print("Sum =", a + b)
print("Difference =", a - b)
print("Product =", a * b)
if b != 0:
    print("Division =", a / b)
else:
    print("Division = Error (cannot divide by zero)")
```

---

c) Word and character count of a given string

python
```
text = input("Enter a string: ")
words = text.split()
word_count = len(words)
char_count = len(text)

print("Word count =", word_count)
print("Character count =", char_count)
```

---

d) Area of a given shape (rectangle, triangle, circle)

python
```
import math

shape = input("Enter shape (rectangle, triangle, circle): ").strip().lower()

if shape == "rectangle":
    length = float(input("Enter length: "))
    width = float(input("Enter width: "))
    area = length * width
    print("Area of rectangle =", area)

elif shape == "triangle":
    base = float(input("Enter base: "))
    height = float(input("Enter height: "))
    area = 0.5 * base * height
    print("Area of triangle =", area)

elif shape == "circle":
    radius = float(input("Enter radius: "))
    area = math.pi * radius ** 2
    print("Area of circle =", area)

else:
    print("Unknown shape!")
```

slip 10

Q.1 Write a Java Program to implement Strategy Pattern for Duck Behavior. Create instance variable that holds current state of Duck from there, we just need to handle all Flying Behaviors and Quack Behavior

FlyBehavior.java

```java
public interface FlyBehavior {
    void fly();
}
```

FlyWithWings.java

```java
public class FlyWithWings implements FlyBehavior {
    @Override
    public void fly() {
        System.out.println("I'm flying with wings!");
    }
}
```

FlyNoWay.java

```java
public class FlyNoWay implements FlyBehavior {
    @Override
    public void fly() {
        System.out.println("I can't fly.");
    }
}
```

FlyRocketPowered.java

```java
public class FlyRocketPowered implements FlyBehavior {
    @Override
    public void fly() {
        System.out.println("I'm flying with a rocket!");
    }
}
```

QuackBehavior.java

```java
public interface QuackBehavior {
    void quack();
}
```

Quack.java

```java
public class Quack implements QuackBehavior {
    @Override
    public void quack() {
        System.out.println("Quack!");
    }
}
```

MuteQuack.java

```java
public class MuteQuack implements QuackBehavior {
    @Override
    public void quack() {
        System.out.println("<< Silence >>");
    }
}
```

Squeak.java

```java
public class Squeak implements QuackBehavior {
    @Override
    public void quack() {
        System.out.println("Squeak!");
    }
}
```

Duck.java

```java
public abstract class Duck {
    protected FlyBehavior flyBehavior;
    protected QuackBehavior quackBehavior;

    public void setFlyBehavior(FlyBehavior fb) {
        flyBehavior = fb;
    }

    public void setQuackBehavior(QuackBehavior qb) {
        quackBehavior = qb;
    }
    public void performFly() {
        flyBehavior.fly();
    }

    public void performQuack() {
        quackBehavior.quack();
    }

    public abstract void display();
}
```

MallardDuck.java

```java
public class MallardDuck extends Duck {
    public MallardDuck() {
        flyBehavior = new FlyWithWings();
        quackBehavior = new Quack();
    }

    @Override
    public void display() {
        System.out.println("I'm a Mallard Duck.");
    }
}
```

ModelDuck.java

```java
public class ModelDuck extends Duck {
    public ModelDuck() {
        flyBehavior = new FlyNoWay();
        quackBehavior = new Quack();
    }

    @Override
    public void display() {
        System.out.println("I'm a Model Duck.");
    }
}
```

StrategyPatternTest.java

```java
public class StrategyPatternTest {
    public static void main(String[] args) {
        Duck mallard = new MallardDuck();
        mallard.display();
        mallard.performFly();
        mallard.performQuack();

        System.out.println();

        Duck model = new ModelDuck();
        model.display();
        model.performFly();
        model.performQuack();

        System.out.println("Changing model duck's fly behavior...");
        model.setFlyBehavior(new FlyRocketPowered());
        model.performFly();
    }
}
```

How to Run in VS Code

1. Create a folder (e.g., `DuckStrategyPattern`)
2. Place all `.java` files inside:
   FlyBehavior.java, FlyWithWings.java, FlyNoWay.java, FlyRocketPowered.java, QuackBehavior.java, Quack.java, MuteQuack.java, Squeak.java, Duck.java, MallardDuck.java, ModelDuck.java, StrategyPatternTest.java
3. Open the folder in VS Code
4. Open terminal (Ctrl + `)
5. Compile and run:

```bash
javac *.java
java StrategyPatternTest
```

Q.2 Write python programs on Pi like: a) Print a name 'n' times, where name and n are read from standard input, using for and while loops. b) Handle Divided by Zero Exception. c) Print current time for 10 times with an interval of10seconds. d) Read a fileline byline and print the word count of each line

a) Print a name 'n' times (using for and while loops)

Using for loop:

```
python
name = input("Enter your name: ")
n = int(input("Enter how many times to print: "))

for i in range(n):
    print(name)
```

Using while loop:

```
python
name = input("Enter your name: ")
n = int(input("Enter how many times to print: "))

count = 0
while count < n:
    print(name)
    count += 1
```

---

b) Handle Divided by Zero Exception

```
python
try:
    num = float(input("Enter numerator: "))
    denom = float(input("Enter denominator: "))
    result = num / denom
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Division by zero is not allowed.")
```

---

c) Print current time for 10 times with an interval of 10 seconds

```
python
import time

for i in range(10):
    print("Current Time:", time.strftime("%Y-%m-%d %H:%M:%S"))
    time.sleep(10)  # sleep for 10 seconds
```

---

d) Read a file line by line and print the word count of each line

```
python
filename = input("Enter file name: ")
```

```
try:
    with open(filename, 'r') as f:
        for line_num, line in enumerate(f, 1):
            word_count = len(line.split())
            print(f"Line {line_num}: {word_count} words")
except FileNotFoundError:
    print("File not found.")
```

slip 11

Q.1 Write a java program to implement Adapter pattern to design Heart Model to Beat Model

HeartModel.java

```
public class HeartModel {
    private int heartRate;

    public HeartModel(int heartRate) {
        this.heartRate = heartRate;
    }

    public int getHeartRate() {
        return heartRate;
    }

    public void setHeartRate(int heartRate) {
        this.heartRate = heartRate;
    }
}
```

BeatModel.java

```
public interface BeatModel {
    int getBeat();
}
```

HeartToBeatAdapter.java

```
public class HeartToBeatAdapter implements BeatModel {
    private HeartModel heartModel;

    public HeartToBeatAdapter(HeartModel heartModel) {
        this.heartModel = heartModel;
    }

    @Override
    public int getBeat() {
```

```java
        // Adapting heartRate to beat (could add conversion logic if needed)
        return heartModel.getHeartRate();
    }
}
```

AdapterPatternTest.java

```java
public class AdapterPatternTest {
    public static void main(String[] args) {
        // Create a HeartModel with initial heart rate
        HeartModel heartModel = new HeartModel(72);
        System.out.println("HeartModel heart rate: " +
heartModel.getHeartRate());

        // Use adapter to make HeartModel usable as BeatModel
        BeatModel beatModel = new HeartToBeatAdapter(heartModel);
        System.out.println("BeatModel beat: " + beatModel.getBeat());

        // Change heart rate, observe reflected beat
        heartModel.setHeartRate(88);
        System.out.println("Updated HeartModel heart rate: " +
heartModel.getHeartRate());
        System.out.println("BeatModel beat after update: " +
beatModel.getBeat());
    }
}
```

How to Run in VS Code

1. Create a folder (e.g., `AdapterHeartToBeat`)
2. Place all `.java` files inside: HeartModel.java, BeatModel.java, HeartToBeatAdapter.java, AdapterPatternTest.java
3. Open the folder in VS Code
4. Open terminal (Ctrl + `)
5. Run:

```bash
javac *.java
java AdapterPatternTest
```

Q.2 Run some python programs on Pi like a) Light an LED through Python program b) Get input from two switches and switch on corresponding LEDs c) Flash an LED at a given on time and off time cycle, where the two times are taken from a file

Slip 12

Q.1 Write a Java Program to implement Decorator Pattern for interface Car to define the assemble() method and then decorate it to Sports car and Luxury Car

Car.java

```
public interface Car {
    void assemble();
}
```

BasicCar.java

```
public class BasicCar implements Car {
    @Override
    public void assemble() {
        System.out.print("Basic Car.");
    }
}
```

CarDecorator.java

```
public class CarDecorator implements Car {
    protected Car decoratedCar;

    public CarDecorator(Car c) {
        this.decoratedCar = c;
    }

    @Override
    public void assemble() {
        decoratedCar.assemble();
    }
}
```

SportsCar.java

```java
public class SportsCar extends CarDecorator {
    public SportsCar(Car c) {
        super(c);
    }

    @Override
    public void assemble() {
        super.assemble();
        System.out.print(" Adding features of Sports Car.");
    }
}
```

LuxuryCar.java

```java
public class LuxuryCar extends CarDecorator {
    public LuxuryCar(Car c) {
        super(c);
    }

    @Override
    public void assemble() {
        super.assemble();
        System.out.print(" Adding features of Luxury Car.");
    }
}
```

DecoratorPatternTest.java

```java
public class DecoratorPatternTest {
    public static void main(String[] args) {
        Car sportsCar = new SportsCar(new BasicCar());
        System.out.print("Sports Car: ");
        sportsCar.assemble();
        System.out.println("\n");

        Car luxuryCar = new LuxuryCar(new BasicCar());
        System.out.print("Luxury Car: ");
        luxuryCar.assemble();
        System.out.println("\n");

        Car sportsLuxuryCar = new LuxuryCar(new SportsCar(new BasicCar()));
        System.out.print("Sports Luxury Car: ");
        sportsLuxuryCar.assemble();
        System.out.println();
    }
}
```

7. How to Run

1. Create a folder (e.g., `DecoratorCarExample`)
2. Place all the `.java` files above in that folder (Car.java, BasicCar.java, CarDecorator.java, SportsCar.java, LuxuryCar.java, DecoratorPatternTest.java)
3. Open the folder in VS Code
4. Open Terminal in VS Code: (Ctrl + `)
   Run the following commands—

```bash
javac *.java
java DecoratorPatternTest
```

slip 13

Q.1 Write a Java Program to implement an Adapter design pattern in mobile charger. Define two classes – Volt (to measure volts) and Socket (producing constant volts of 120V). Build an adapter that can produce 3 volts, 12 volts and default 120 volts. Implements Adapter pattern using Class Adapter

Volt.java

```
public class Volt {
    private int volts;

    public Volt(int volts) {
        this.volts = volts;
    }

    public int getVolts() {
        return volts;
    }

    public void setVolts(int volts) {
        this.volts = volts;
    }
}
```

Socket.java

```
public class Socket {
    public Volt getVolt() {
        return new Volt(120);
    }
}
```

SocketAdapter.java

```java
public interface SocketAdapter {
    Volt get3Volt();
    Volt get12Volt();
    Volt get120Volt();
}
```

SocketClassAdapterImpl.java

```java
public class SocketClassAdapterImpl extends Socket implements SocketAdapter {
    @Override
    public Volt get3Volt() {
        return convertVolt(getVolt(), 40);
    }

    @Override
    public Volt get12Volt() {
        return convertVolt(getVolt(), 10);
    }

    @Override
    public Volt get120Volt() {
        return getVolt();
    }

    private Volt convertVolt(Volt v, int divisor) {
        return new Volt(v.getVolts() / divisor);
    }
}
```

AdapterPatternTest.java

```java
public class AdapterPatternTest {
    public static void main(String[] args) {
        SocketAdapter adapter = new SocketClassAdapterImpl();

        Volt v3 = adapter.get3Volt();
        Volt v12 = adapter.get12Volt();
        Volt v120 = adapter.get120Volt();

        System.out.println("Mobile needs 3V : " + v3.getVolts() + "V");
        System.out.println("Needs 12V : " + v12.getVolts() + "V");
        System.out.println("Needs 120V : " + v120.getVolts() + "V");
    }
}
```

6. Compile and Run in VS Code

1. Create a folder (e.g., `AdapterMobileCharger`)
2. Add all five `.java` files (Volt.java, Socket.java, SocketAdapter.java, SocketClassAdapterImpl.java, AdapterPatternTest.java) in the folder
3. Open terminal in VS Code
Compile and run:

```bash
javac *.java
java AdapterPatternTest
```

slip 14

Q.1 Write a Java Program to implement Command Design Pattern for Command Interface with execute() . Use this to create variety of commands for LightOnCommand, LightOffCommand, GarageDoorUpCommand, StereoOnWithCDComman.

Command.java

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

Light.java

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

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

GarageDoor.java

```java
public class GarageDoor {
    public void up() {
        System.out.println("Garage Door is UP");
    }
}
```

Stereo.java

```java
public class Stereo {
    public void on() {
        System.out.println("Stereo is ON");
    }

    public void setCD() {
        System.out.println("Stereo is set for CD input");
    }

    public void setVolume(int volume) {
        System.out.println("Stereo volume set to " + volume);
    }
}
```

LightOnCommand.java

```java
public class LightOnCommand implements Command {
    private Light light;
    public LightOnCommand(Light light) {
        this.light = light;
    }
    @Override
    public void execute() {
        light.on();
    }
}
```

LightOffCommand.java

```java
public class LightOffCommand implements Command {
    private Light light;
    public LightOffCommand(Light light) {
        this.light = light;
    }
    @Override
    public void execute() {
        light.off();
    }
}
```

GarageDoorUpCommand.java

```java
public class GarageDoorUpCommand implements Command {
    private GarageDoor garageDoor;
    public GarageDoorUpCommand(GarageDoor garageDoor) {
        this.garageDoor = garageDoor;
    }
    @Override
    public void execute() {
        garageDoor.up();
    }
}
```

StereoOnWithCDCommand.java

```java
public class StereoOnWithCDCommand implements Command {
    private Stereo stereo;
    public StereoOnWithCDCommand(Stereo stereo) {
        this.stereo = stereo;
    }
    @Override
    public void execute() {
        stereo.on();
        stereo.setCD();
        stereo.setVolume(11);
    }
}
```

RemoteControl.java

```java
public class RemoteControl {
    private Command command;
    public void setCommand(Command command) {
        this.command = command;
    }
    public void pressButton() {
        command.execute();
    }
}
```

CommandPatternTest.java

```java
public class CommandPatternTest {
    public static void main(String[] args) {
        Light light = new Light();
        GarageDoor garageDoor = new GarageDoor();
        Stereo stereo = new Stereo();

        Command lightOn = new LightOnCommand(light);
        Command lightOff = new LightOffCommand(light);
        Command garageUp = new GarageDoorUpCommand(garageDoor);
        Command stereoOnWithCD = new StereoOnWithCDCommand(stereo);

        RemoteControl remote = new RemoteControl();

        System.out.print("Press Light ON: ");
        remote.setCommand(lightOn);
        remote.pressButton();

        System.out.print("Press Light OFF: ");
        remote.setCommand(lightOff);
        remote.pressButton();

        System.out.print("Press Garage Door UP: ");
        remote.setCommand(garageUp);
        remote.pressButton();

        System.out.print("Press Stereo ON with CD: ");
        remote.setCommand(stereoOnWithCD);
        remote.pressButton();
    }
}
```

How to Run in VS Code

1. Create a folder (e.g., `CommandPatternDemo`)
2. Add all `.java` files above (Command.java, Light.java, LightOnCommand.java, LightOffCommand.java, GarageDoor.java, GarageDoorUpCommand.java, Stereo.java, StereoOnWithCDCommand.java, RemoteControl.java, CommandPatternTest.java)
3. Open the folder in VS Code
4. Open Terminal (Ctrl + `)
5. Compile and run:

```
bash
javac *.java
java CommandPatternTest
```

slip 15

Q.1 Write a Java Program to implement Facade Design Pattern for Home Theater

Amplifier.java

```
public class Amplifier {
    public void on() {
        System.out.println("Amplifier ON");
    }

    public void off() {
        System.out.println("Amplifier OFF");
    }

    public void setVolume(int level) {
        System.out.println("Amplifier setting volume to " + level);
    }
}
```

DVDPlayer.java

```
public class DVDPlayer {
    public void on() {
        System.out.println("DVD Player ON");
    }

    public void off() {
        System.out.println("DVD Player OFF");
    }

    public void play(String movie) {
        System.out.println("DVD Player playing movie: " + movie);
    }

    public void stop() {
        System.out.println("DVD Player stopped");
    }
}
```

Projector.java

```java
public class Projector {
    public void on() {
        System.out.println("Projector ON");
    }

    public void off() {
        System.out.println("Projector OFF");
    }

    public void wideScreenMode() {
        System.out.println("Projector in widescreen mode");
    }
}
```

Screen.java

```java
public class Screen {
    public void down() {
        System.out.println("Screen going DOWN");
    }

    public void up() {
        System.out.println("Screen going UP");
    }
}
```

Lights.java

```java
public class Lights {
    public void dim(int level) {
        System.out.println("Lights dimmed to " + level + "%");
    }
    public void on() {
        System.out.println("Lights ON");
    }
}
```

HomeTheaterFacade.java

```java
public class HomeTheaterFacade {
    private Amplifier amp;
    private DVDPlayer dvd;
    private Projector projector;
    private Screen screen;
    private Lights lights;

    public HomeTheaterFacade(Amplifier amp, DVDPlayer dvd, Projector
projector, Screen screen, Lights lights) {
        this.amp = amp;
        this.dvd = dvd;
        this.projector = projector;
        this.screen = screen;
        this.lights = lights;
    }

    public void watchMovie(String movie) {
        System.out.println("Get ready to watch a movie...");
        lights.dim(10);
        screen.down();
        projector.on();
        projector.wideScreenMode();
        amp.on();
        amp.setVolume(15);
        dvd.on();
        dvd.play(movie);
    }

    public void endMovie() {
        System.out.println("Shutting movie theater down...");
        dvd.stop();
        dvd.off();
        amp.off();
        projector.off();
        screen.up();
        lights.on();
    }
}
```

HomeTheaterTest.java

```java
public class HomeTheaterTest {
    public static void main(String[] args) {
        Amplifier amp = new Amplifier();
        DVDPlayer dvd = new DVDPlayer();
        Projector projector = new Projector();
        Screen screen = new Screen();
        Lights lights = new Lights();

        HomeTheaterFacade homeTheater = new HomeTheaterFacade(
            amp, dvd, projector, screen, lights
        );

        homeTheater.watchMovie("Inception");
        System.out.println();
        homeTheater.endMovie();
    }
}
```

How to Run in VS Code

1. Create a folder (e.g., `FacadeHomeTheater`)
2. Add all `.java` files:
   Amplifier.java, DVDPlayer.java, Projector.java, Screen.java, Lights.java, HomeTheaterFacade.java, HomeTheaterTest.java
3. Open folder in VS Code
4. Open Terminal (Ctrl + `)
5. Compile and execute:

```bash
javac *.java
java HomeTheaterTest
```

Slip 16

Write a Java Program to implement Observer Design Pattern for number conversion. Accept a number in Decimal form and represent it in Hexadecimal, Octal and Binary. Change the Number and it reflects in other forms also

Subject.java

```java
import java.util.*;
```

```java
public class Subject {
    private List<Observer> observers = new ArrayList<>();
    private int number;

    public void attach(Observer obs) {
        observers.add(obs);
    }

    public void setNumber(int number) {
        this.number = number;
        notifyAllObservers();
    }

    public int getNumber() {
        return number;
    }

    private void notifyAllObservers() {
        for (Observer obs : observers) {
            obs.update();
        }
    }
}
```

Observer.java

```java
public abstract class Observer {
    protected Subject subject;
    public abstract void update();
}
```

BinaryObserver.java

```java
public class BinaryObserver extends Observer {
    public BinaryObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }
    @Override
    public void update() {
        System.out.println("Binary: " +
Integer.toBinaryString(subject.getNumber()));
    }
}
```

OctalObserver.java

```
public class OctalObserver extends Observer {
    public OctalObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }
    @Override
    public void update() {
        System.out.println("Octal: " +
Integer.toOctalString(subject.getNumber()));
    }
}
```

HexObserver.java

```
public class HexObserver extends Observer {
    public HexObserver(Subject subject) {
        this.subject = subject;
        this.subject.attach(this);
    }
    @Override
    public void update() {
        System.out.println("Hexadecimal: " +
Integer.toHexString(subject.getNumber()).toUpperCase());
    }
}
```

ObserverPatternTest.java

```
import java.util.Scanner;

public class ObserverPatternTest {
    public static void main(String[] args) {
        Subject numberSubject = new Subject();

        new BinaryObserver(numberSubject);
        new OctalObserver(numberSubject);
        new HexObserver(numberSubject);
        Scanner scanner = new Scanner(System.in);

        while(true) {
            System.out.print("Enter a decimal number (or -1 to exit): ");
            int num = scanner.nextInt();
            if (num == -1) break;
            numberSubject.setNumber(num);
            System.out.println();
        }
        scanner.close();
    }
}
```

How to Run in VS Code

1. Create a folder (e.g., `ObserverNumberConversion`)
2. Add all `.java` files:
   Subject.java, Observer.java, BinaryObserver.java, OctalObserver.java, HexObserver.java, ObserverPatternTest.java
3. Open the folder in VS Code
4. Open terminal (Ctrl+`)
5. Compile and run:

```bash
javac *.java
java ObserverPatternTest
```

slip 17

Q.1 Write a Java Program to implement Abstract Factory Pattern for Shape interface.

Shape.java

```java
public interface Shape {
    void draw();
}
```

RoundedRectangle.java

```java
public class RoundedRectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside RoundedRectangle::draw() method.");
    }
}
```

RoundedSquare.java

```java
public class RoundedSquare implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside RoundedSquare::draw() method.");
    }
}
```

Rectangle.java

```java
public class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside Rectangle::draw() method.");
    }
}
```

Square.java

```java
public class Square implements Shape {
    @Override
    public void draw() {
        System.out.println("Inside Square::draw() method.");
    }
}
```

AbstractFactory.java

```java
public abstract class AbstractFactory {
    abstract Shape getShape(String shapeType);
}
```

RoundedShapeFactory.java

```java
public class RoundedShapeFactory extends AbstractFactory {
    @Override
    Shape getShape(String shapeType) {
        if (shapeType.equalsIgnoreCase("RECTANGLE")) {
            return new RoundedRectangle();
        } else if (shapeType.equalsIgnoreCase("SQUARE")) {
            return new RoundedSquare();
        }
        return null;
    }
}
```

NormalShapeFactory.java

```java
public class NormalShapeFactory extends AbstractFactory {
    @Override
    Shape getShape(String shapeType) {
        if (shapeType.equalsIgnoreCase("RECTANGLE")) {
            return new Rectangle();
        } else if (shapeType.equalsIgnoreCase("SQUARE")) {
            return new Square();
        }
        return null;
    }
}
```

FactoryProducer.java

```java
public class FactoryProducer {
    public static AbstractFactory getFactory(boolean rounded) {
        if (rounded) {
            return new RoundedShapeFactory();
        } else {
            return new NormalShapeFactory();
        }
    }
}
```

AbstractFactoryPatternTest.java

```java
public class AbstractFactoryPatternTest {
    public static void main(String[] args) {
        AbstractFactory shapeFactory = FactoryProducer.getFactory(false);
        Shape shape1 = shapeFactory.getShape("RECTANGLE");
        shape1.draw();

        Shape shape2 = shapeFactory.getShape("SQUARE");
        shape2.draw();

        AbstractFactory roundedShapeFactory =
FactoryProducer.getFactory(true);
        Shape shape3 = roundedShapeFactory.getShape("RECTANGLE");
        shape3.draw();

        Shape shape4 = roundedShapeFactory.getShape("SQUARE");
        shape4.draw();
    }
}
```