**Q1)** 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.ArrayList;

import java.util.List;

// Observer interface

interface Observer {

    void update(float temperature, float humidity, float pressure);

}

// Subject interface

interface Subject {

    void registerObserver(Observer o);

    void removeObserver(Observer o);

    void notifyObservers();

}

// WeatherData class implementing Subject

class WeatherData implements Subject {

    private List<Observer> observers;

    private float temperature;

    private float humidity;

    private float pressure;

    public WeatherData() {

        observers = new ArrayList<>();

    }

    @Override

    public void registerObserver(Observer o) {

        observers.add(o);

    }

    @Override

    public void removeObserver(Observer o) {

        observers.remove(o);

    }

    @Override

    public void notifyObservers() {

        for (Observer observer : observers) {

            observer.update(temperature, humidity, pressure);

        }

    }

    public void setMeasurements(float temperature, float humidity, float pressure) {

        this.temperature = temperature;

        this.humidity = humidity;

        this.pressure = pressure;

        notifyObservers();

    }

}

// Display class implementing Observer

class CurrentConditionsDisplay implements Observer {

    private float temperature;

    private float humidity;

    public CurrentConditionsDisplay(Subject weatherData) {

        weatherData.registerObserver(this);

    }

    @Override

    public void update(float temperature, float humidity, float pressure) {

        this.temperature = temperature;

        this.humidity = humidity;

        display();

    }

    public void display() {

        System.out.println("Current conditions: " + temperature + "F degrees and " + humidity + "% humidity.");

    }

}

// Main class to run the Weather Station program

public class WeatherStation {

    public static void main(String[] args) {

        WeatherData weatherData = new WeatherData();

        CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(weatherData);

        // Simulate new weather measurements

        weatherData.setMeasurements(80, 65, 30.4f);

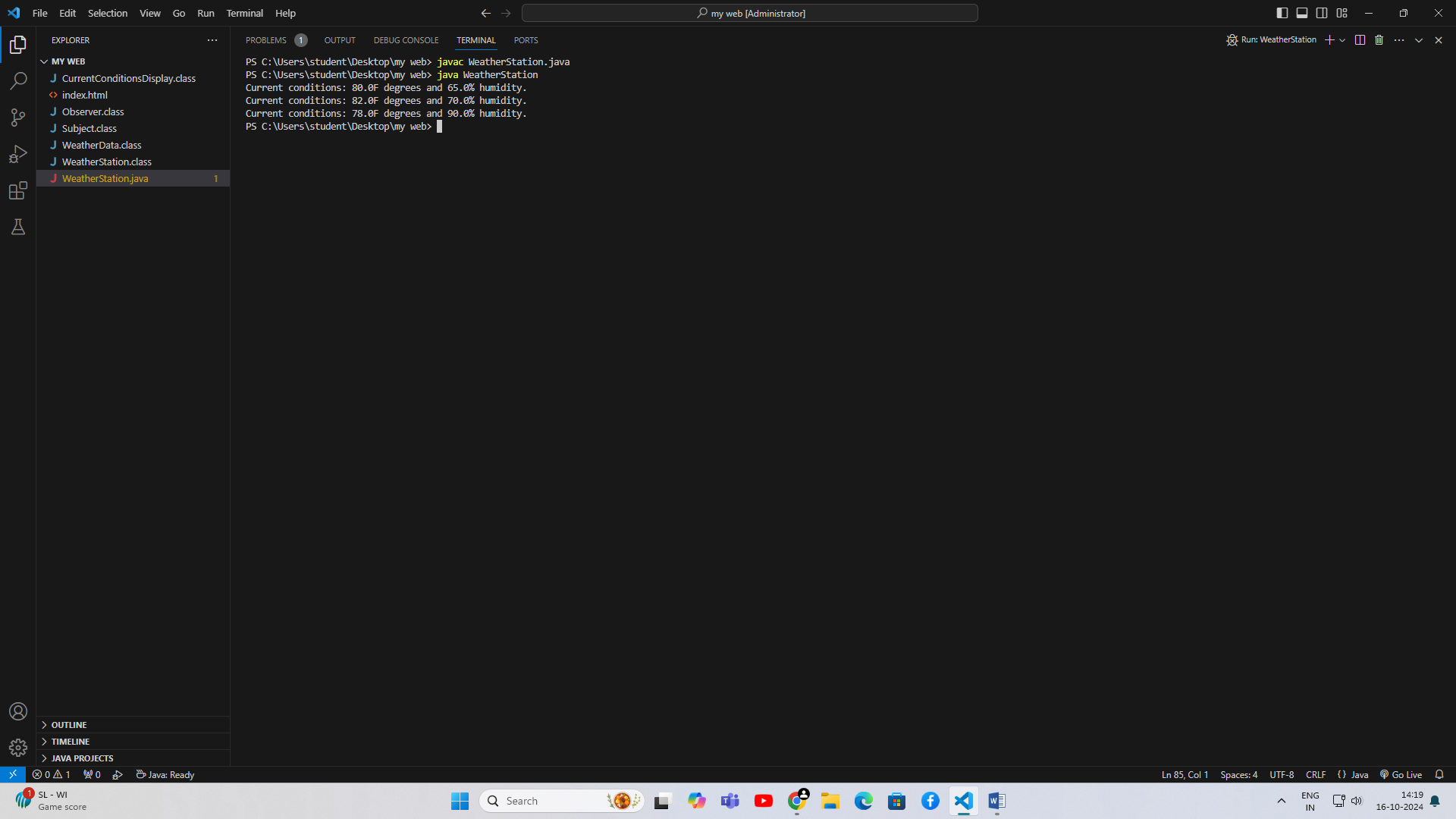
        weatherData.setMeasurements(82, 70, 29.2f);

        weatherData.setMeasurements(78, 90, 29.2f);

    }

}

Output:



**Q2)** Write a Java Program to implement I/O Decorator for converting uppercase letters to lower case letters.

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.io.Reader;

// BasicReader class to read text from a file

class BasicReader {

private BufferedReader reader;

public BasicReader(String filePath) throws IOException {

reader = new BufferedReader(new FileReader(filePath));

}

public String readLine() throws IOException {

return reader.readLine();

}

public void close() throws IOException {

reader.close();

}

}

// UpperToLowerDecorator class to convert uppercase letters to lowercase

class UpperToLowerDecorator {

private BasicReader basicReader;

public UpperToLowerDecorator(BasicReader basicReader) {

this.basicReader = basicReader;

}

public String readLine() throws IOException {

String line = basicReader.readLine();

return line != null ? line.toLowerCase() : null; // Convert to lowercase

}

public void close() throws IOException {

basicReader.close();

}

}

// Main class to demonstrate the functionality

public class IODecoratorExample {

public static void main(String[] args) {

String filePath = "input.txt"; // Specify your file path here

try {

BasicReader basicReader = new BasicReader(filePath);

UpperToLowerDecorator decorator = new UpperToLowerDecorator(basicReader);

String line;

while ((line = decorator.readLine()) != null) {

System.out.println(line); // Print the converted line

}

decorator.close(); // Close the decorator

} catch (IOException e) {

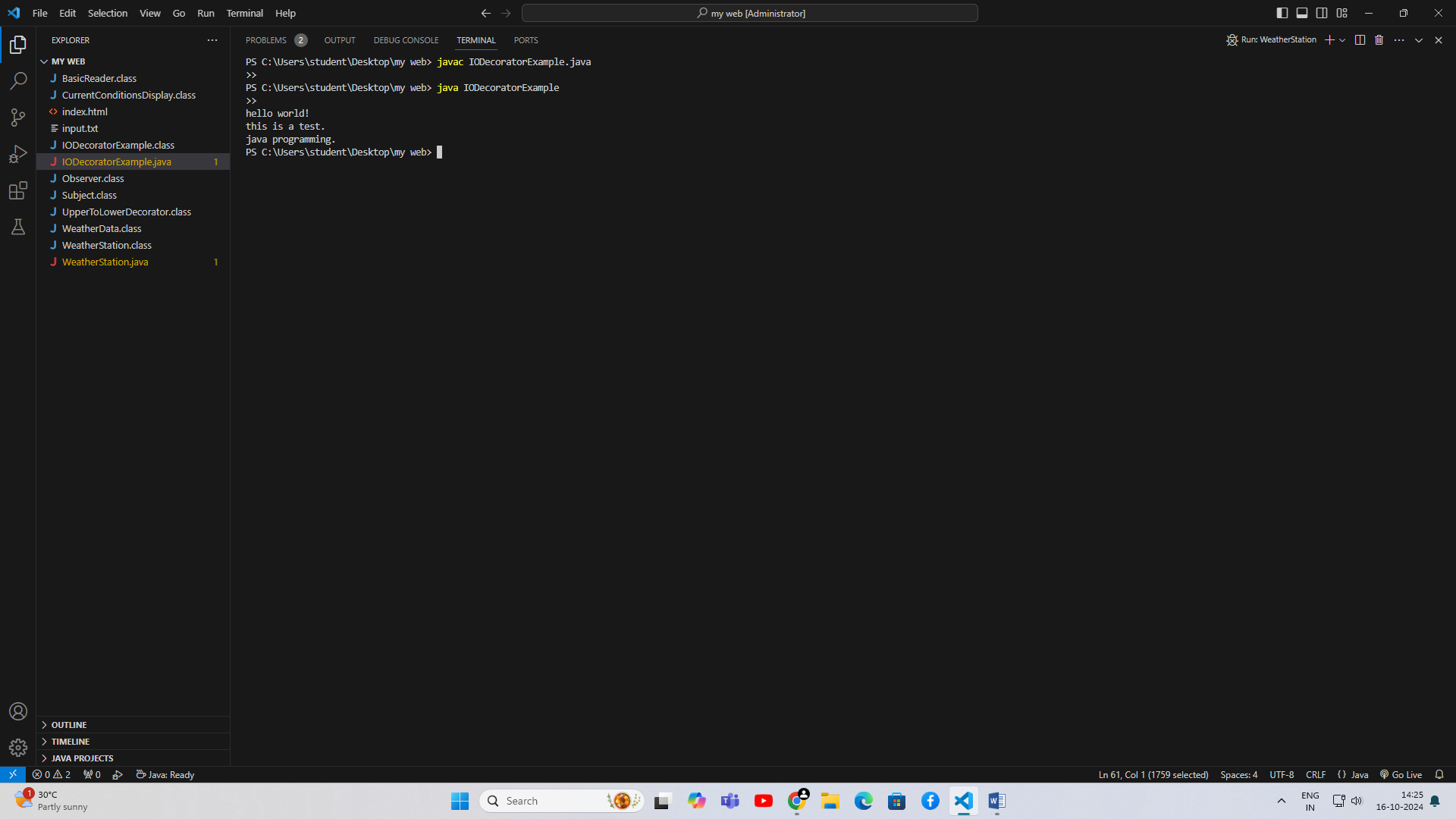
e.printStackTrace();

}

}

}

Output:



**Q3)** Write a Java Program to implement Factory method for Pizza Store with createPizza(), orderPizza(),prepare(), bake(), cut(), box(). Use this to create variety of pizza’s like NyStyleCheesePizza, ChicagoStyleCheesePizzaetc.

// Abstract Pizza class

abstract class Pizza {

String name;

String dough;

String sauce;

void prepare() {

System.out.println("Preparing " + name);

System.out.println("Tossing dough: " + dough);

System.out.println("Adding sauce: " + sauce);

}

void bake() {

System.out.println("Baking " + name);

}

void cut() {

System.out.println("Cutting " + name);

}

void box() {

System.out.println("Boxing " + name);

}

public String getName() {

return name;

}

}

// Concrete Pizza classes

class NYStyleCheesePizza extends Pizza {

public NYStyleCheesePizza() {

name = "NY Style Cheese Pizza";

dough = "Thin Crust Dough";

sauce = "Marinara Sauce";

}

}

class ChicagoStyleCheesePizza extends Pizza {

public ChicagoStyleCheesePizza() {

name = "Chicago Style Cheese Pizza";

dough = "Extra Thick Crust Dough";

sauce = "Plum Tomato Sauce";

}

}

// PizzaStore abstract class

abstract class PizzaStore {

public Pizza orderPizza(String type) {

Pizza pizza = createPizza(type);

pizza.prepare();

pizza.bake();

pizza.cut();

pizza.box();

return pizza;

}

protected abstract Pizza createPizza(String type);

}

// Concrete PizzaStore classes

class NYStylePizzaStore extends PizzaStore {

@Override

protected Pizza createPizza(String type) {

if (type.equals("cheese")) {

return new NYStyleCheesePizza();

} else {

return null; // You can add more pizza types here

}

}

}

class ChicagoStylePizzaStore extends PizzaStore {

@Override

protected Pizza createPizza(String type) {

if (type.equals("cheese")) {

return new ChicagoStyleCheesePizza();

} else {

return null; // You can add more pizza types here

}

}

}

// Main class to demonstrate the Factory Method pattern

public class PizzaStoreExample {

public static void main(String[] args) {

PizzaStore nyStore = new NYStylePizzaStore();

PizzaStore chicagoStore = new ChicagoStylePizzaStore();

System.out.println("Ordering a NY Style Cheese Pizza:");

nyStore.orderPizza("cheese");

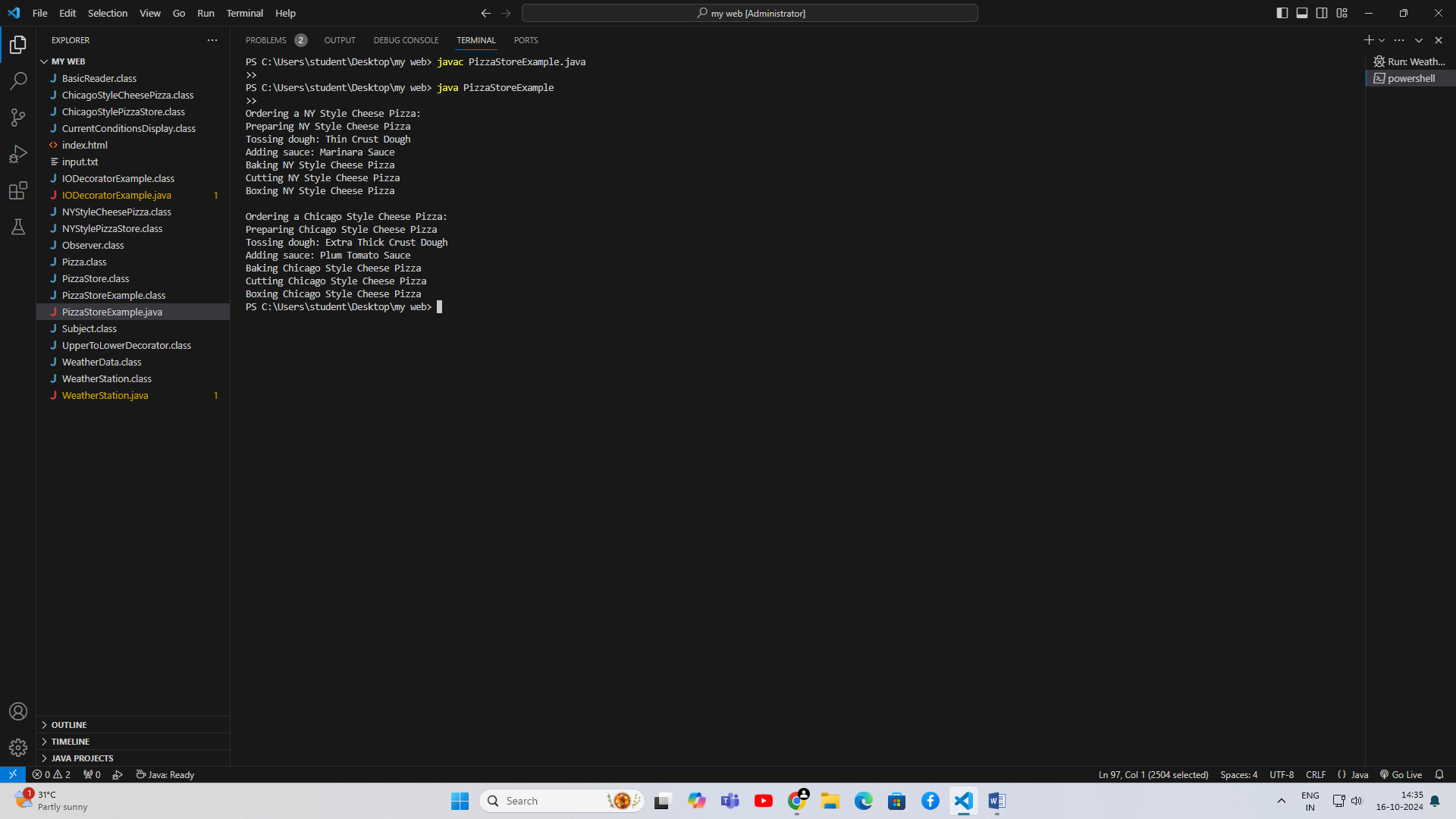
System.out.println("\nOrdering a Chicago Style Cheese Pizza:");

chicagoStore.orderPizza("cheese");

}

}

Output:



**Q4)** Write a Java Program to implement Singleton pattern for multithreading.

class Singleton {

// Volatile variable to ensure visibility of changes across threads

private static volatile Singleton instance;

// Private constructor to prevent instantiation

private Singleton() {

}

// Double-checked locking method to get the instance

public static Singleton getInstance() {

if (instance == null) { // First check (no locking)

synchronized (Singleton.class) {

if (instance == null) { // Second check (with locking)

instance = new Singleton();

}

}

}

return instance;

}

// Example method

public void showMessage() {

System.out.println("Hello from Singleton!");

}

}

// Runnable implementation to demonstrate multithreading

class SingletonThread implements Runnable {

@Override

public void run() {

Singleton singleton = Singleton.getInstance();

singleton.showMessage();

}

}

// Main class to test the Singleton implementation

public class SingletonExample {

public static void main(String[] args) {

// Create multiple threads that will access the Singleton

Thread thread1 = new Thread(new SingletonThread());

Thread thread2 = new Thread(new SingletonThread());

Thread thread3 = new Thread(new SingletonThread());

// Start the threads

thread1.start();

thread2.start();

thread3.start();

// Wait for threads to finish

try {

thread1.join();

thread2.join();

thread3.join();

} catch (InterruptedException e) {

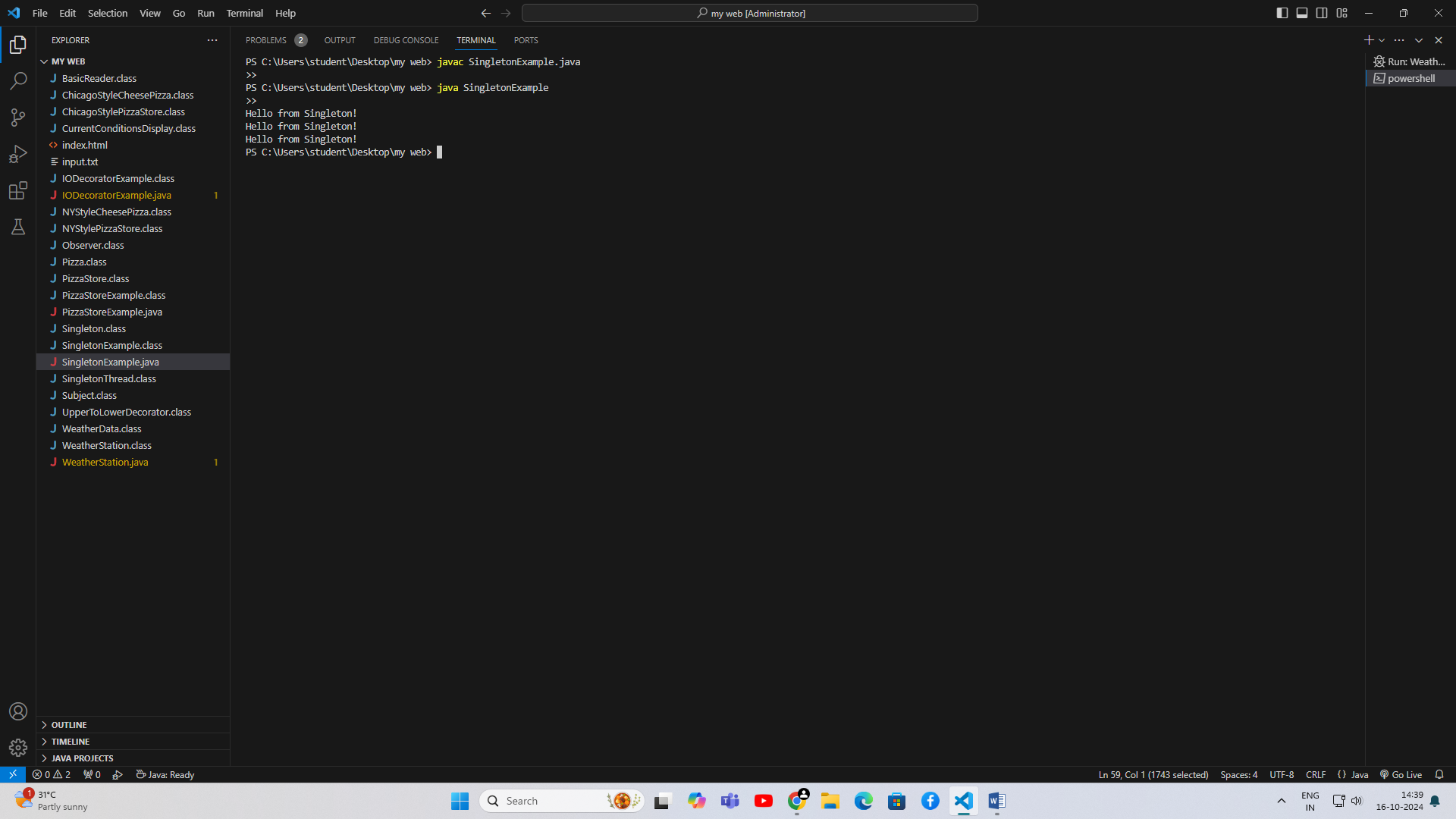
e.printStackTrace();

}

}

}

Output:



**Q5)** Write a Java Program to implement command pattern to test Remote Control. Book

package CommandPatternRemote;

public interface Command {

    void execute();

}

class Light {

    public void on() {

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

    }

    public void off() {

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

    }

}

class Fan {

    public void on() {

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

    }

    public void off() {

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

    }

}

class LightOnCommand implements Command {

    Light light;

    public LightOnCommand(Light light) {

        this.light = light;

    }

    public void execute() {

        light.on();

    }

}

class LightOffCommand implements Command {

    Light light;

    public LightOffCommand(Light light) {

        this.light = light;

    }

    public void execute() {

        light.off();

    }

}

class FanOnCommand implements Command {

    Fan fan;

    public FanOnCommand(Fan fan) {

        this.fan = fan;

    }

    public void execute() {

        fan.on();

    }

}

class FanOffCommand implements Command {

    Fan fan;

    public FanOffCommand(Fan fan) {

        this.fan = fan;

    }

    public void execute() {

        fan.off();

    }

}

// RemoteControl.java

package CommandPatternRemote;

class RemoteControl {

    Command[] onCommands;

    Command[] offCommands;

    public RemoteControl() {

        onCommands = new Command[2];

        offCommands = new Command[2];

    }

    public void setCommand(int slot, Command onCommand, Command offCommand) {

        onCommands[slot] = onCommand;

        offCommands[slot] = offCommand;

    }

    public void pressOnButton(int slot) {

        onCommands[slot].execute();

    }

    public void pressOffButton(int slot) {

        offCommands[slot].execute();

    }

}

package CommandPatternRemote;

// RemoteControlTest.java

public class RemoteControlTest {

    public static void main(String[] args) {

        RemoteControl remote = new RemoteControl();

        Light livingRoomLight = new Light();

        Fan livingRoomFan = new Fan();

        Command lightOn = new LightOnCommand(livingRoomLight);

        Command lightOff = new LightOffCommand(livingRoomLight);

        Command fanOn = new FanOnCommand(livingRoomFan);

        Command fanOff = new FanOffCommand(livingRoomFan);

        // Set commands in the remote control

        remote.setCommand(0, lightOn, lightOff);

        remote.setCommand(1, fanOn, fanOff);

        // Simulate button presses

        System.out.println("Pressing Light ON button...");

        remote.pressOnButton(0);

        System.out.println("Pressing Light OFF button...");

        remote.pressOffButton(0);

        System.out.println("Pressing Fan ON button...");

        remote.pressOnButton(1);

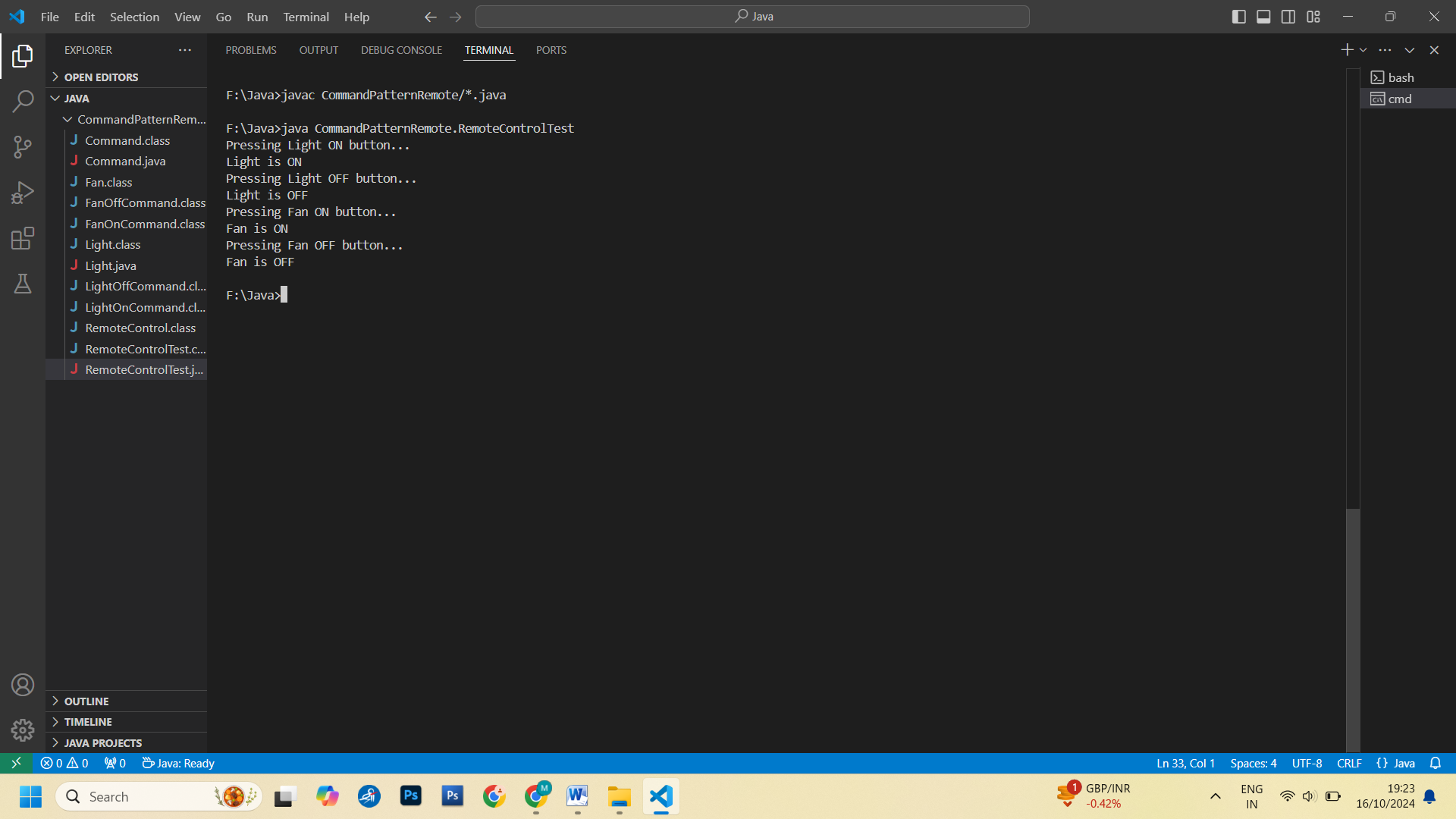
        System.out.println("Pressing Fan OFF button...");

        remote.pressOffButton(1);

    }

}

Output:



Q6. Write a Java Program to implement undo command to test Ceilingfan.

package CeilingFanUndo;

// Command.java

public interface Command {

    void execute();

    void undo();

}

package CeilingFanUndo;

// CeilingFan.java

class CeilingFan {

    public static final int HIGH = 3;

    public static final int MEDIUM = 2;

    public static final int LOW = 1;

    public static final int OFF = 0;

    private String location;

    private int speed;

    public CeilingFan(String location) {

        this.location = location;

        speed = OFF;  // Fan starts off

    }

    public void high() {

        speed = HIGH;

        System.out.println(location + " CeilingFan is on HIGH");

    }

    public void medium() {

        speed = MEDIUM;

        System.out.println(location + " CeilingFan is on MEDIUM");

    }

    public void low() {

        speed = LOW;

        System.out.println(location + " CeilingFan is on LOW");

    }

    public void off() {

        speed = OFF;

        System.out.println(location + " CeilingFan is OFF");

    }

    public int getSpeed() {

        return speed;

    }

}

package CeilingFanUndo;

// CeilingFanHighCommand.java

class CeilingFanHighCommand implements Command {

    CeilingFan ceilingFan;

    int prevSpeed;

    public CeilingFanHighCommand(CeilingFan ceilingFan) {

        this.ceilingFan = ceilingFan;

    }

    public void execute() {

        prevSpeed = ceilingFan.getSpeed();

        ceilingFan.high();

    }

    public void undo() {

        if (prevSpeed == CeilingFan.HIGH) {

            ceilingFan.high();

        } else if (prevSpeed == CeilingFan.MEDIUM) {

            ceilingFan.medium();

        } else if (prevSpeed == CeilingFan.LOW) {

            ceilingFan.low();

        } else {

            ceilingFan.off();

        }

    }

}

package CeilingFanUndo;

class CeilingFanMediumCommand implements Command {

    CeilingFan ceilingFan;

    int prevSpeed;

    public CeilingFanMediumCommand(CeilingFan ceilingFan) {

        this.ceilingFan = ceilingFan;

    }

    public void execute() {

        prevSpeed = ceilingFan.getSpeed();

        ceilingFan.medium();

    }

    public void undo() {

        if (prevSpeed == CeilingFan.HIGH) {

            ceilingFan.high();

        } else if (prevSpeed == CeilingFan.MEDIUM) {

            ceilingFan.medium();

        } else if (prevSpeed == CeilingFan.LOW) {

            ceilingFan.low();

        } else {

            ceilingFan.off();

        }

    }

}

package CeilingFanUndo;

// CeilingFanOffCommand.java

class CeilingFanOffCommand implements Command {

    CeilingFan ceilingFan;

    int prevSpeed;

    public CeilingFanOffCommand(CeilingFan ceilingFan) {

        this.ceilingFan = ceilingFan;

    }

    public void execute() {

        prevSpeed = ceilingFan.getSpeed();

        ceilingFan.off();

    }

    public void undo() {

        if (prevSpeed == CeilingFan.HIGH) {

            ceilingFan.high();

        } else if (prevSpeed == CeilingFan.MEDIUM) {

            ceilingFan.medium();

        } else if (prevSpeed == CeilingFan.LOW) {

            ceilingFan.low();

        } else {

            ceilingFan.off();

        }

    }

}

package CeilingFanUndo;

// RemoteControl.java

class RemoteControl {

    Command[] onCommands;

    Command[] offCommands;

    Command undoCommand;

    public RemoteControl() {

        onCommands = new Command[2];

        offCommands = new Command[2];

        undoCommand = null; // No command executed yet

    }

    public void setCommand(int slot, Command onCommand, Command offCommand) {

        onCommands[slot] = onCommand;

        offCommands[slot] = offCommand;

    }

    public void pressOnButton(int slot) {

        onCommands[slot].execute();

        undoCommand = onCommands[slot];

    }

    public void pressOffButton(int slot) {

        offCommands[slot].execute();

        undoCommand = offCommands[slot];

    }

    public void pressUndoButton() {

        if (undoCommand != null) {

            undoCommand.undo();

        } else {

            System.out.println("No command to undo");

        }

    }

}

package CeilingFanUndo;

// RemoteControlTest.java

public class RemoteControlTest {

    public static void main(String[] args) {

        RemoteControl remote = new RemoteControl();

        CeilingFan ceilingFan = new CeilingFan("Living Room");

        CeilingFanHighCommand ceilingFanHigh = new CeilingFanHighCommand(ceilingFan);

        CeilingFanMediumCommand ceilingFanMedium = new CeilingFanMediumCommand(ceilingFan);

        CeilingFanOffCommand ceilingFanOff = new CeilingFanOffCommand(ceilingFan);

        // Set commands for the remote control

        remote.setCommand(0, ceilingFanHigh, ceilingFanOff);

        remote.setCommand(1, ceilingFanMedium, ceilingFanOff);

        // Simulate button presses

        System.out.println("Turning ceiling fan to HIGH...");

        remote.pressOnButton(0);

        System.out.println("Undoing last command...");

        remote.pressUndoButton();

        System.out.println("Turning ceiling fan to MEDIUM...");

        remote.pressOnButton(1);

        System.out.println("Turning ceiling fan OFF...");

        remote.pressOffButton(1);

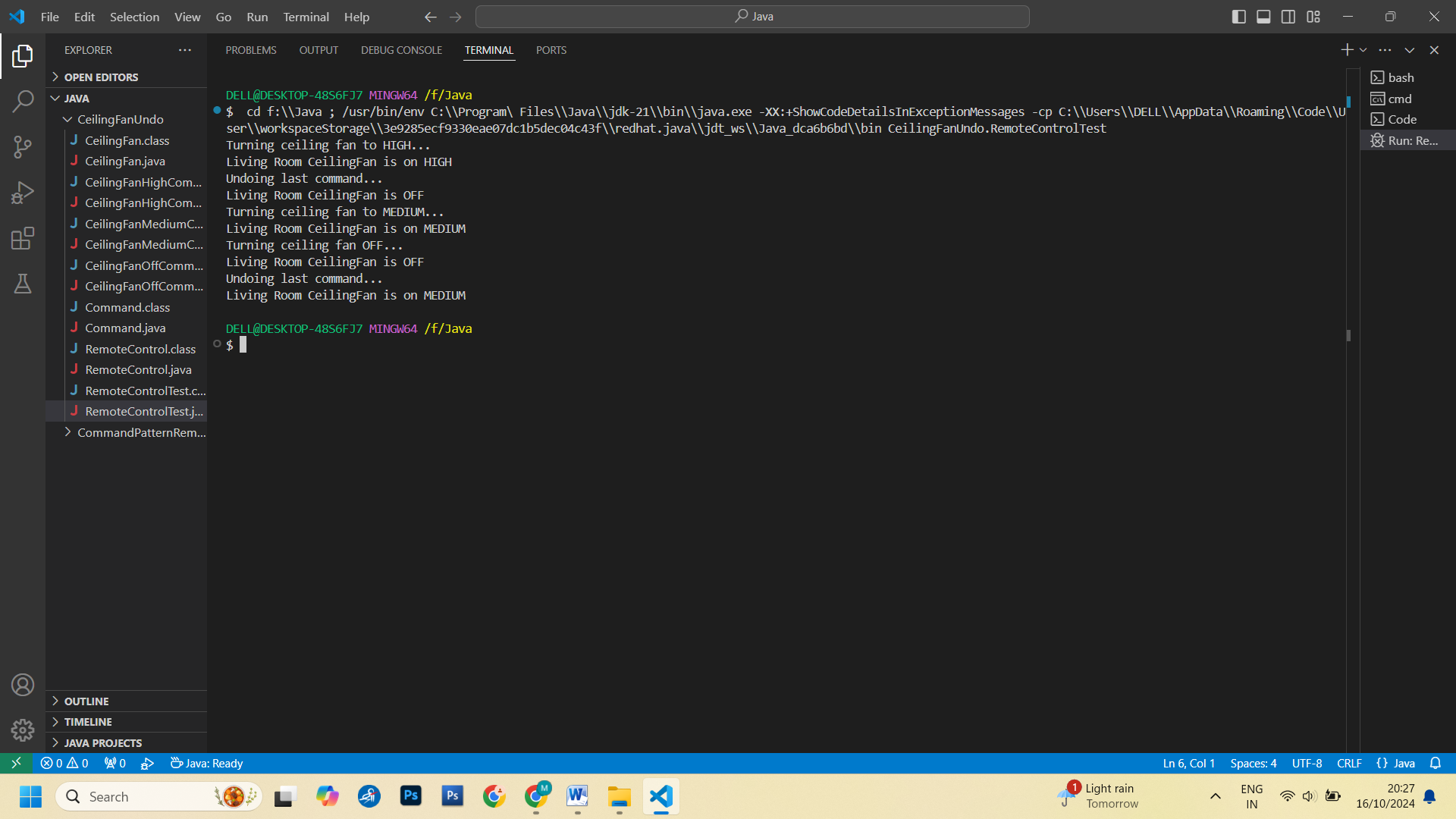
        System.out.println("Undoing last command...");

        remote.pressUndoButton();

    }

}

Output:



**Q7.** Write a Java Program to implement Iterator Pattern forDesigning Menu like Breakfast, Lunch or DinnerMenu.

package MenuIteratorPattern;

class MenuItem {

    private String name;

    private String description;

    private boolean vegetarian;

    private double price;

    public MenuItem(String name, String description, boolean vegetarian, double price) {

        this.name = name;

        this.description = description;

        this.vegetarian = vegetarian;

        this.price = price;

    }

    public String getName() {

        return name;

    }

    public String getDescription() {

        return description;

    }

    public boolean isVegetarian() {

        return vegetarian;

    }

    public double getPrice() {

        return price;

    }

}

package MenuIteratorPattern;

interface Menu {

    Iterator createIterator();

}

package MenuIteratorPattern;

interface Iterator {

    boolean hasNext();

    Object next();

}

package MenuIteratorPattern;

class BreakfastMenu implements Menu {

    private MenuItem[] menuItems;

    private int numberOfItems = 0;

    public BreakfastMenu() {

        menuItems = new MenuItem[5];

        addItem("Pancakes", "Fluffy pancakes with syrup", true, 2.99);

        addItem("Waffles", "Crispy waffles with fruit", true, 3.59);

        addItem("Omelette", "Cheese omelette with toast", false, 4.29);

    }

    public void addItem(String name, String description, boolean vegetarian, double price) {

        MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

        if (numberOfItems >= menuItems.length) {

            System.out.println("Menu is full! Can't add item: " + name);

        } else {

            menuItems[numberOfItems] = menuItem;

            numberOfItems++;

        }

    }

    @Override

    public Iterator createIterator() {

        return new BreakfastMenuIterator(menuItems);

    }

}

package MenuIteratorPattern;

class LunchMenu implements Menu {

    private MenuItem[] menuItems;

    private int numberOfItems = 0;

    public LunchMenu() {

        menuItems = new MenuItem[5];

        addItem("Sandwich", "Grilled chicken sandwich", false, 5.99);

        addItem("Salad", "Mixed salad with dressing", true, 4.99);

        addItem("Pizza", "Cheese pizza with toppings", false, 7.99);

    }

    public void addItem(String name, String description, boolean vegetarian, double price) {

        MenuItem menuItem = new MenuItem(name, description, vegetarian, price);

        if (numberOfItems >= menuItems.length) {

            System.out.println("Menu is full! Can't add item: " + name);

        } else {

            menuItems[numberOfItems] = menuItem;

            numberOfItems++;

        }

    }

    @Override

    public Iterator createIterator() {

        return new LunchMenuIterator(menuItems);

    }

}

package MenuIteratorPattern;

class BreakfastMenuIterator implements Iterator {

    private MenuItem[] items;

    private int position = 0;

    public BreakfastMenuIterator(MenuItem[] items) {

        this.items = items;

    }

    @Override

    public boolean hasNext() {

        return position < items.length && items[position] != null;

    }

    @Override

    public Object next() {

        MenuItem menuItem = items[position];

        position++;

        return menuItem;

    }

}

package MenuIteratorPattern;

class LunchMenuIterator implements Iterator {

    private MenuItem[] items;

    private int position = 0;

    public LunchMenuIterator(MenuItem[] items) {

        this.items = items;

    }

    @Override

    public boolean hasNext() {

        return position < items.length && items[position] != null;

    }

    @Override

    public Object next() {

        MenuItem menuItem = items[position];

        position++;

        return menuItem;

    }

}

package MenuIteratorPattern;

class Waitress {

    private Menu breakfastMenu;

    private Menu lunchMenu;

    public Waitress(Menu breakfastMenu, Menu lunchMenu) {

        this.breakfastMenu = breakfastMenu;

        this.lunchMenu = lunchMenu;

    }

    public void printMenu() {

        System.out.println("Breakfast Menu:");

        printMenu(breakfastMenu.createIterator());

        System.out.println("\nLunch Menu:");

        printMenu(lunchMenu.createIterator());

    }

    private void printMenu(Iterator iterator) {

        while (iterator.hasNext()) {

            MenuItem menuItem = (MenuItem) iterator.next();

            System.out.print(menuItem.getName() + ", ");

            System.out.print(menuItem.getPrice() + " -- ");

            System.out.println(menuItem.getDescription());

        }

    }

}

package MenuIteratorPattern;

public class IteratorPatternDemo {

    public static void main(String[] args) {

        Menu breakfastMenu = new BreakfastMenu();

        Menu lunchMenu = new LunchMenu();

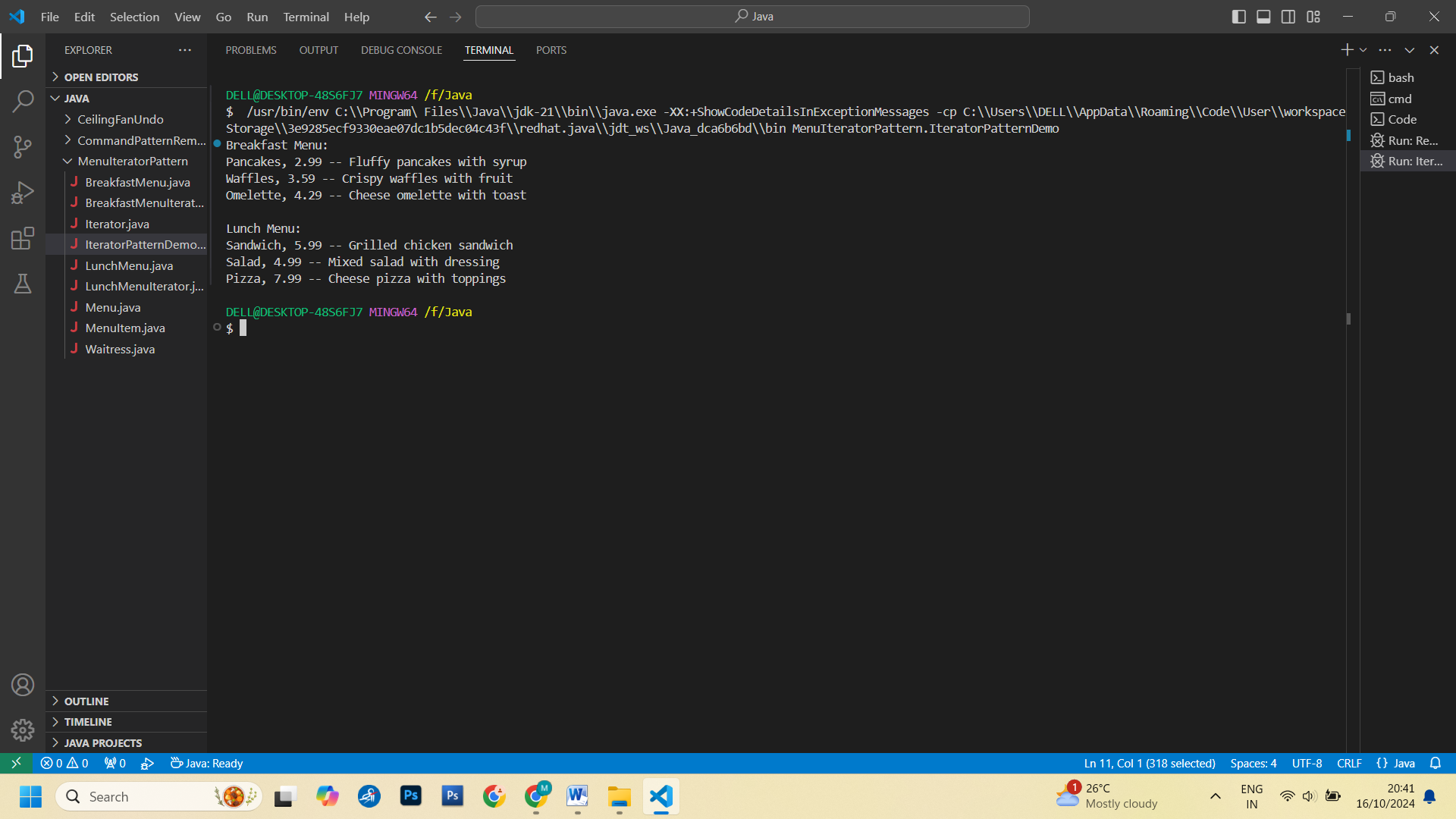
        Waitress waitress = new Waitress(breakfastMenu, lunchMenu);

        waitress.printMenu();

    }

}

Output:



**Assignments based on Internet of Things (IoT)**

Q1) To write a program to sense the available networks using Arduino

#include "WiFi.h"

void setup() {

Serial.begin(115200);

delay(100);

// Initialize WiFi in station mode

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

Serial.println("Starting Wi-Fi scan...");

// Scan for networks

int networks = WiFi.scanNetworks();

Serial.println("Scan complete.");

// Display the results

if (networks == 0) {

Serial.println("No networks found.");

} else {

Serial.print(networks);

Serial.println(" networks found:");

for (int i = 0; i < networks; ++i) {

// Display SSID and signal strength (RSSI) of each network found

Serial.print(i + 1);

Serial.print(": ");

Serial.print(WiFi.SSID(i));

Serial.print(" (Signal strength: ");

Serial.print(WiFi.RSSI(i));

Serial.println(" dBm)");

}

}

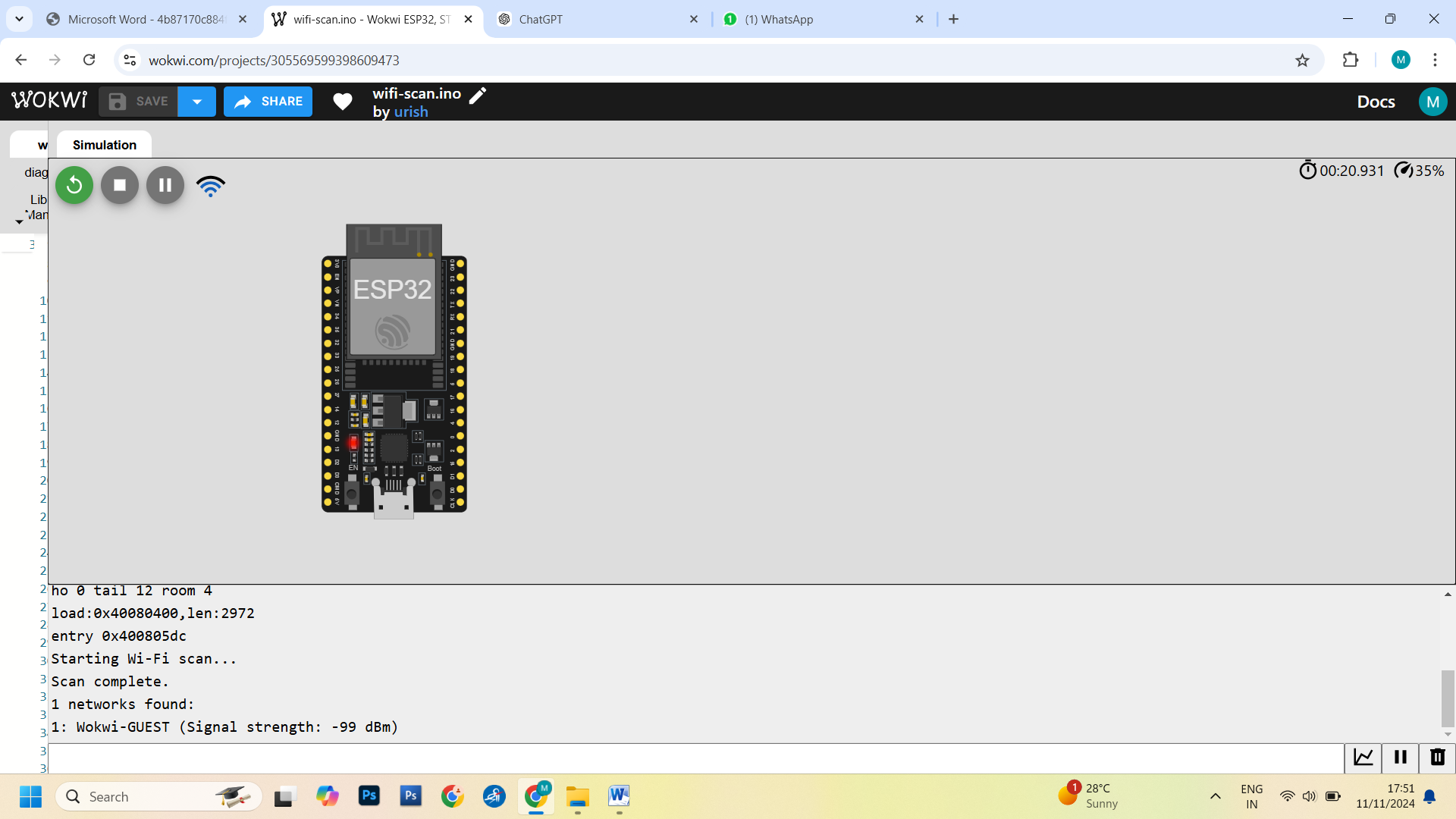
}

void loop() {

// Keep the loop empty

}

Output:



Q2) To write a program to measure the distance using ultrasonic sensor and make LED blink using Arduino.

// Define pins

const int trigPin = 9;

const int echoPin = 10;

const int ledPin = 13;

// Variables for duration and distance

long duration;

int distance;

void setup() {

  // Initialize serial communication

**Serial**.begin(9600);

  // Set the ultrasonic sensor and LED pins

  pinMode(trigPin, OUTPUT);

  pinMode(echoPin, INPUT);

  pinMode(ledPin, OUTPUT);

}

void loop() {

  // Clear the trigPin

  digitalWrite(trigPin, LOW);

  delayMicroseconds(2);

  // Set the trigPin HIGH for 10 microseconds

  digitalWrite(trigPin, HIGH);

  delayMicroseconds(10);

  digitalWrite(trigPin, LOW);

  // Read the echoPin and calculate the duration

  duration = pulseIn(echoPin, HIGH);

  // Calculate the distance in centimeters

  distance = duration \* 0.034 / 2;

  // Print the distance to the Serial Monitor

**Serial**.print("Distance: ");

**Serial**.print(distance);

**Serial**.println(" cm");

  // Make the LED blink if the distance is less than 10 cm

  if (distance < 400) {

    digitalWrite(ledPin, HIGH);  // Turn LED on

    delay(500);                  // Wait for 500 ms

    digitalWrite(ledPin, LOW);   // Turn LED off

    delay(500);                  // Wait for 500 ms

  } else {

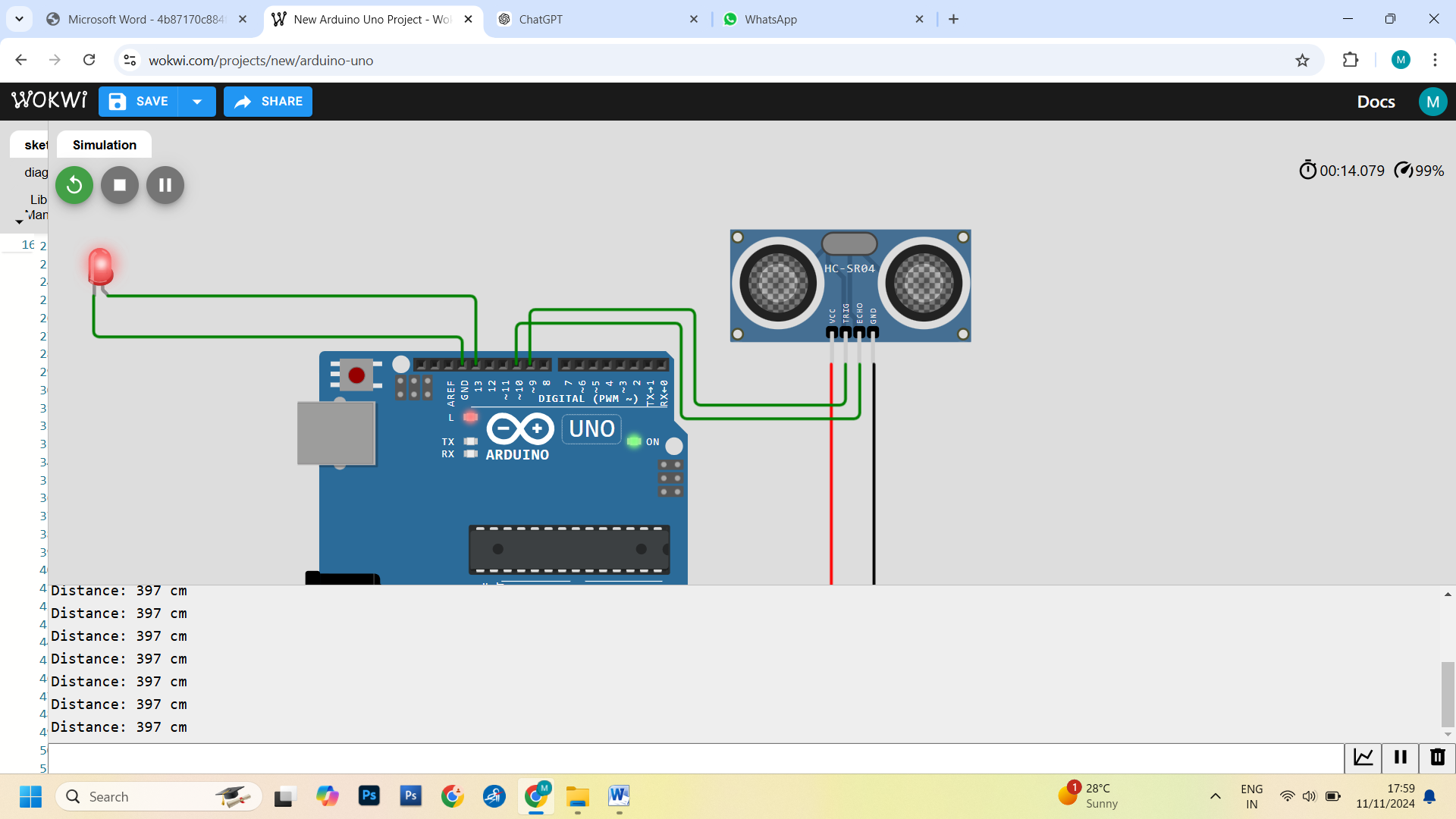
    digitalWrite(ledPin, LOW);   // Keep LED off if distance is more than 10 cm

  }

  delay(100); // Small delay for stability

}

OutPut:



Q3) To write a program to detects the vibration of an object with sensor using Arduino.

const int buttonPin = 1;  // Pin connected to the push button

const int ledPin = 13;    // Pin connected to LED

void setup() {

  pinMode(buttonPin, INPUT);     // Set button pin as input

  pinMode(ledPin, OUTPUT);       // Set LED pin as output

**Serial**.begin(9600);            // Initialize serial communication

}

void loop() {

  int buttonState = digitalRead(buttonPin);  // Read the button state

  if (buttonState == HIGH) {

    // If button is pressed (simulating vibration)

**Serial**.println("Vibration (button press) detected!");

    digitalWrite(ledPin, HIGH);  // Turn on the LED

  } else {

    // If button is not pressed

**Serial**.println("No vibration (button not pressed)");

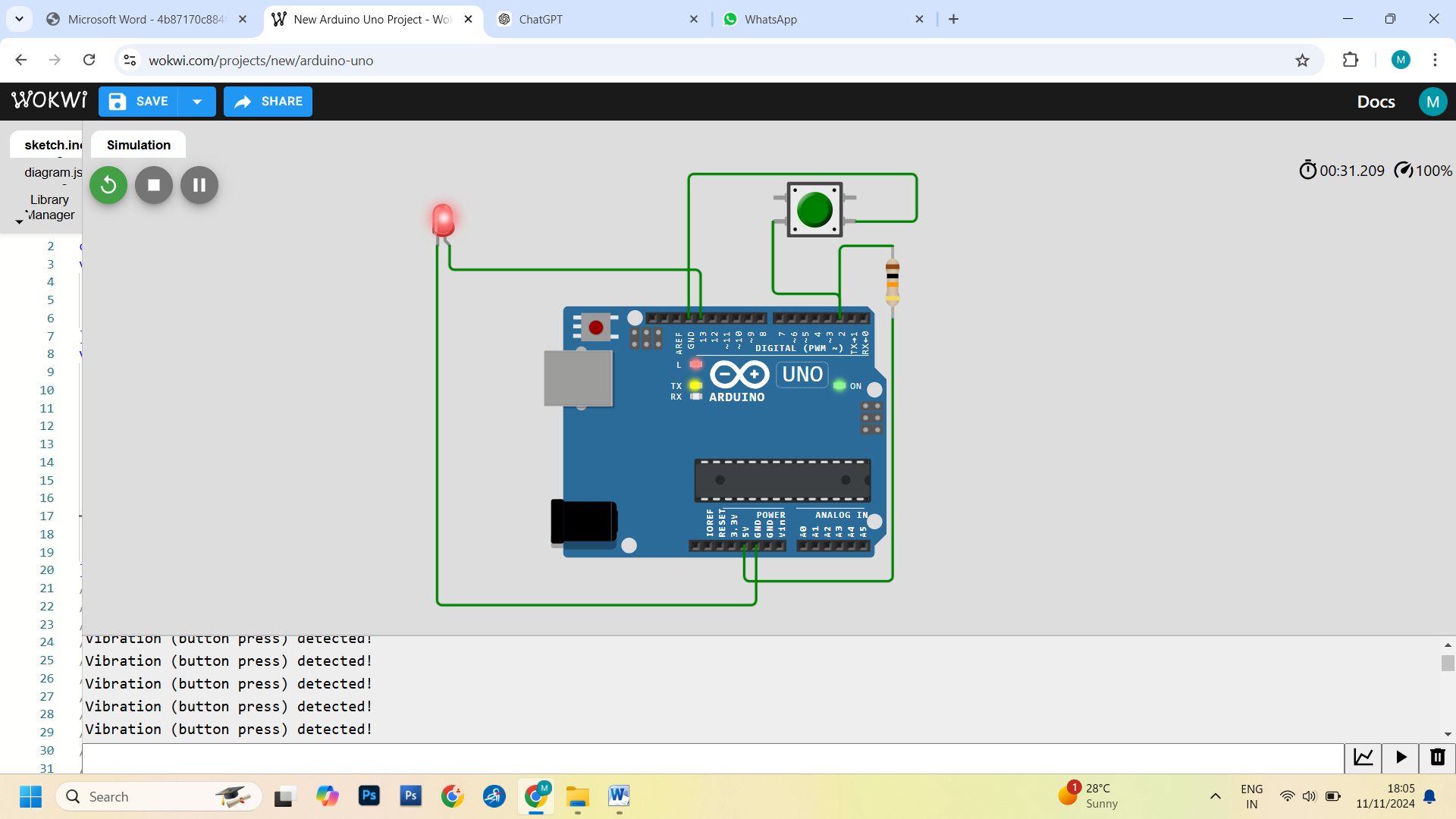
    digitalWrite(ledPin, LOW);   // Turn off the LED

  }

  delay(100); // Small delay for stability

}

Output:



Q4) To write a program to sense a finger when it is placed on the board Arduino.

const int buttonPin = 1;

void setup() {

  pinMode(buttonPin, INPUT);

**Serial**.begin(9600);

}

void loop() {

  int buttonState = digitalRead(buttonPin);

  if (buttonState == HIGH) {

**Serial**.println("Finger (Button) detected!");

  } else {

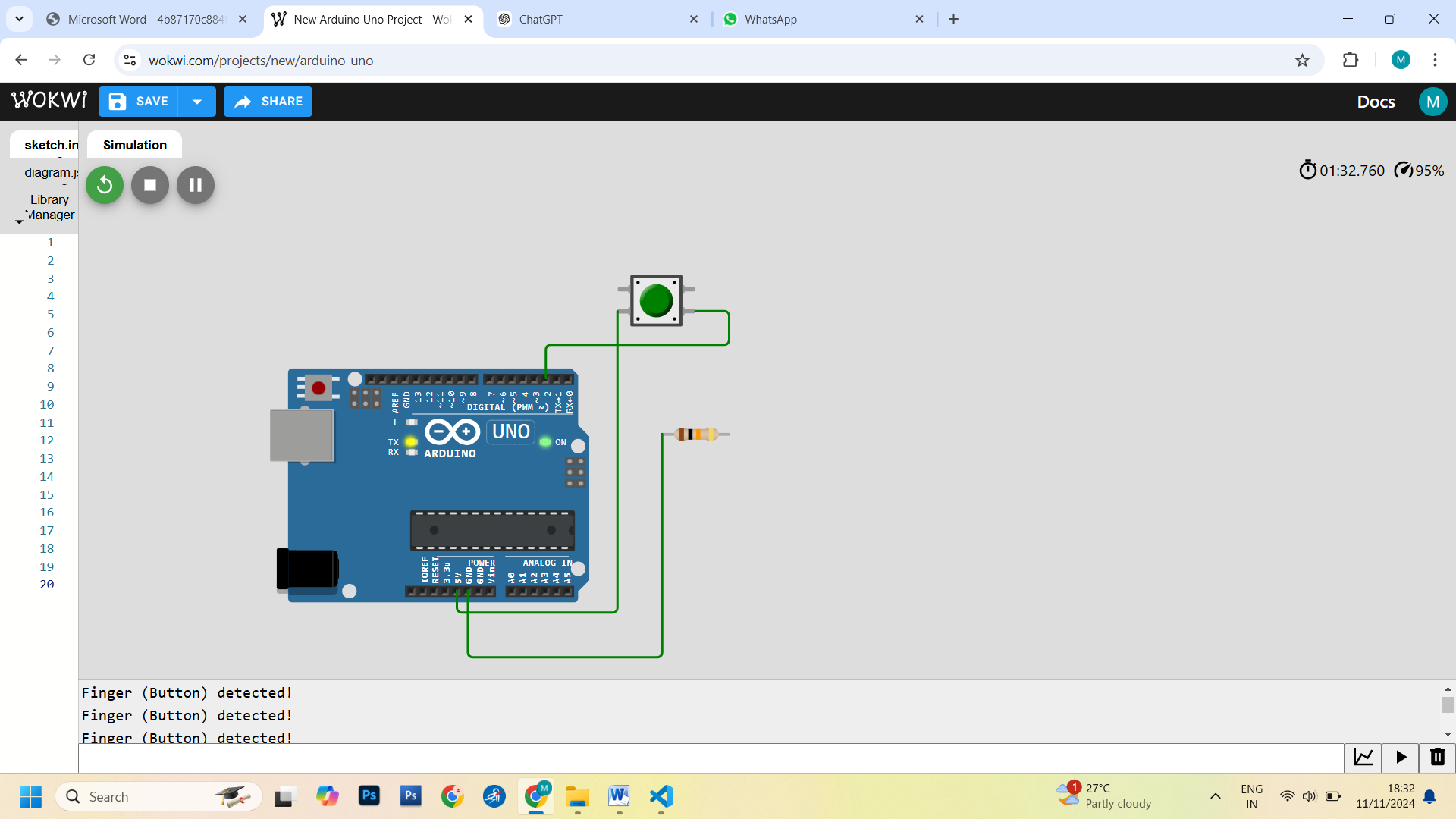
**Serial**.println("No touch detected.");

  }

  delay(100);

}

Output:



Q5) To write a program to connect with the available Wi-Fi using Arduino.

#include "WiFi.h"

// Default Wi-Fi credentials

const char\* ssid = "Wokwi-GUEST";

const char\* password = "";

void setup() {

Serial.begin(115200);

delay(100);

// Begin Wi-Fi connection

WiFi.begin(ssid, password);

Serial.print("Connecting to Wi-Fi");

// Check connection status

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("\nConnected to Wi-Fi!");

Serial.print("IP Address: ");

Serial.println(WiFi.localIP());

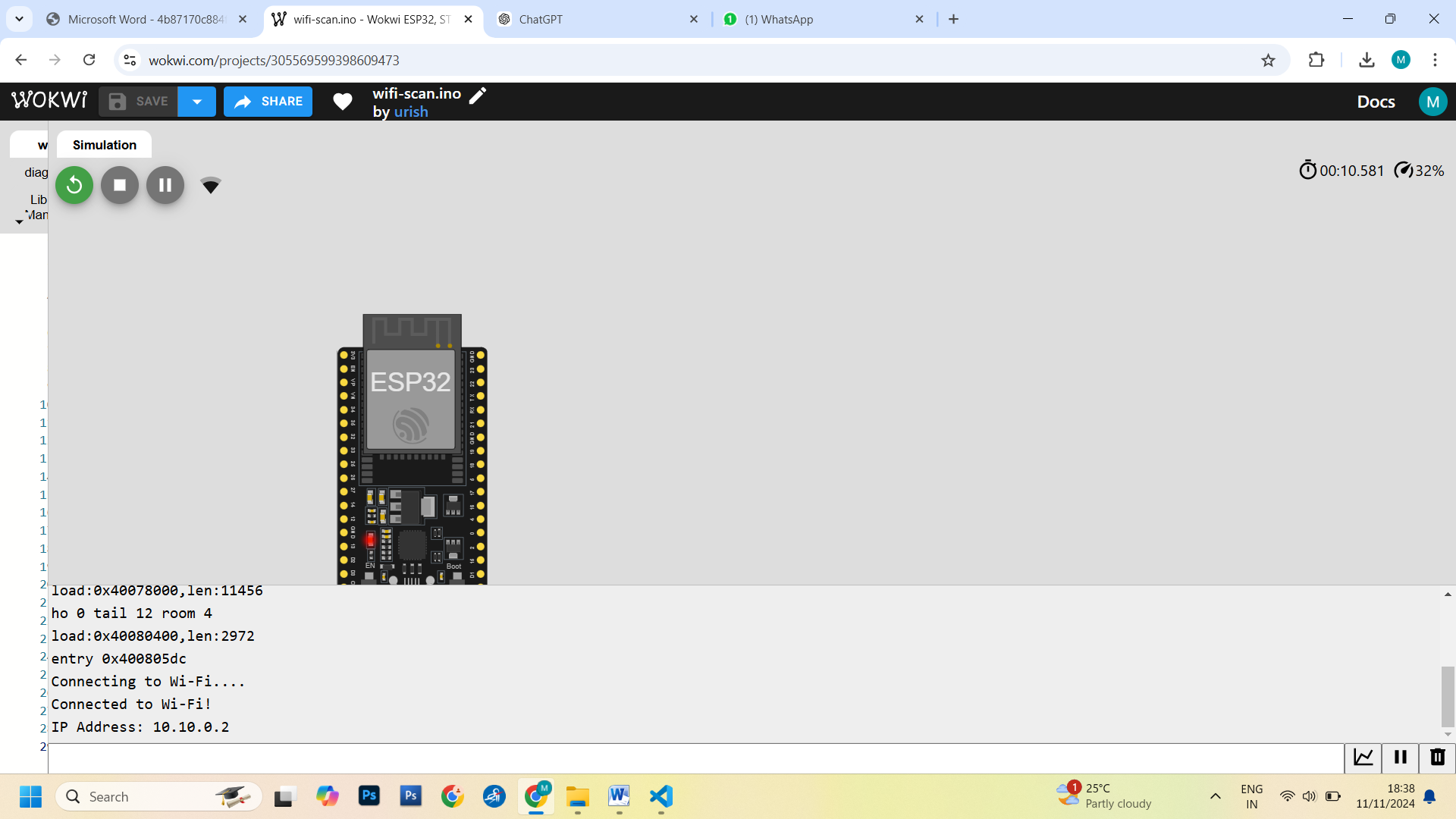
}

void loop() {

// The loop is empty in this case

}

Output:



Q6) To write a program to get temperature notification using Arduino.

#include <DHT.h>

// Define the pin connected to the DHT sensor

#define DHTPIN 7

// Define the sensor type (DHT11 or DHT22)

#define DHTTYPE DHT11

// Create an instance of the DHT class

DHT dht(DHTPIN, DHTTYPE);

const int ledPin = 13;  // Pin for LED notification (optional)

void setup() {

  // Start Serial communication

**Serial**.begin(9600);

  // Initialize the DHT sensor

  dht.begin();

  // Initialize the LED pin

  pinMode(ledPin, OUTPUT);

}

void loop() {

  // Wait a few seconds between measurements

  delay(2000);

  // Get the temperature in Celsius

  float temperature = dht.readTemperature();

  // Check if the reading is valid

  if (isnan(temperature)) {

**Serial**.println("Failed to read temperature!");

    return;

  }

  // Print the temperature to the Serial Monitor

**Serial**.print("Temperature: ");

**Serial**.print(temperature);

**Serial**.println(" \*C");

  // If the temperature is above 30°C, trigger a notification (LED blink)

  if (temperature > 30) {

    digitalWrite(ledPin, HIGH);  // Turn on LED

**Serial**.println("Temperature is too high! Turning on LED.");

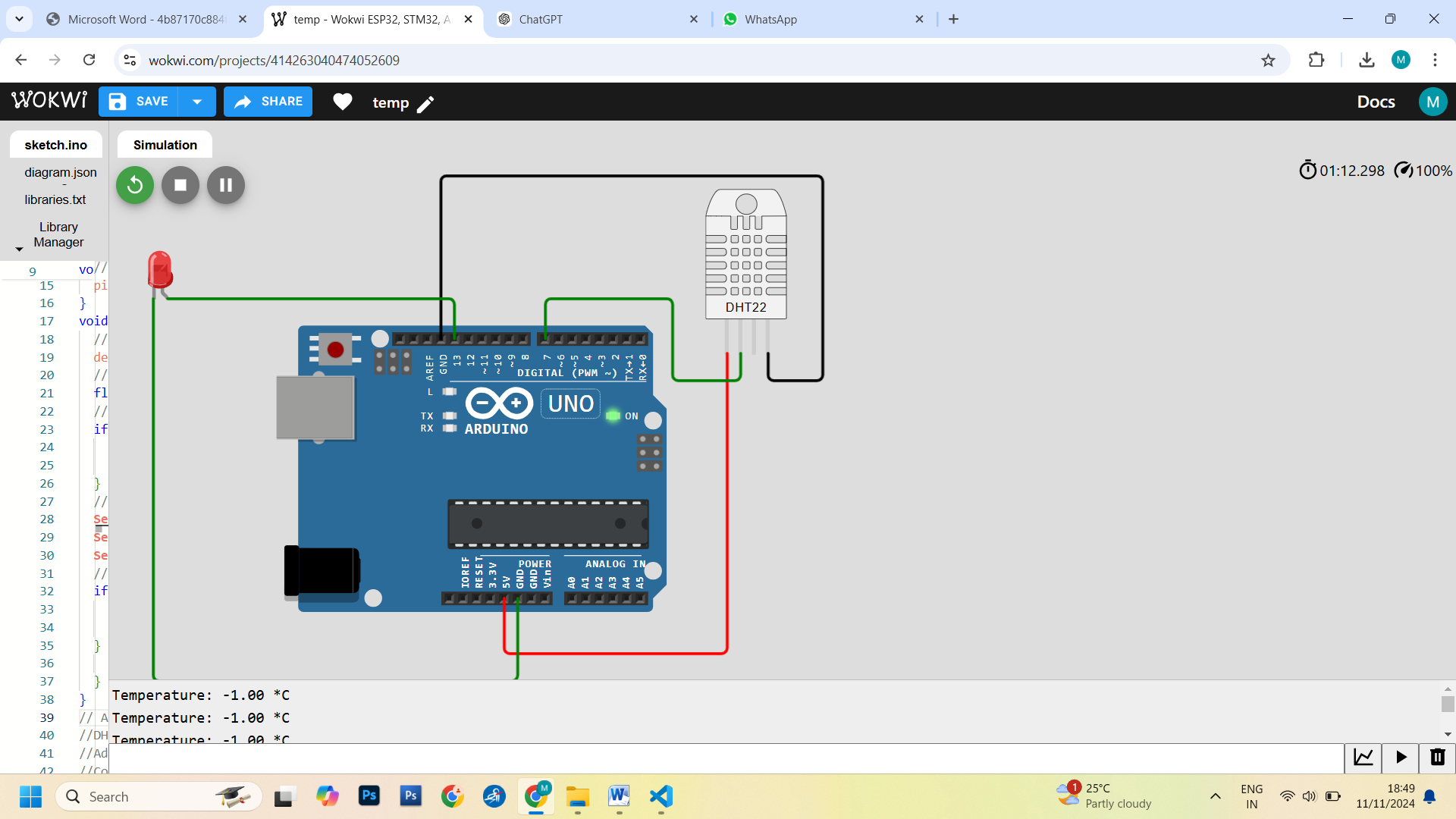
  } else {

    digitalWrite(ledPin, LOW);   // Turn off LED

  }

}

Output:



Q7) To write a program for LDR to vary the light intensity of LED using Arduino

const int ldrPin = A0;  // Pin for LDR

const int ledPin = 9;   // Pin for LED (PWM pin)

int ldrValue = 0;       // Variable to store the LDR value

void setup() {

  pinMode(ledPin, OUTPUT);  // Set LED pin as output

**Serial**.begin(9600);       // Start serial communication for debugging

}

void loop() {

  // Read the value from the LDR (0 to 1023)

  ldrValue = analogRead(ldrPin);

  // Map the LDR value to a range suitable for PWM (0 to 255)

  int ledBrightness = map(ldrValue, 0, 1023, 0, 255);

  // Write the mapped value to the LED (control brightness)

  analogWrite(ledPin, ledBrightness);

  // Output the LDR value and LED brightness for debugging

**Serial**.print("LDR Value: ");

**Serial**.print(ldrValue);

**Serial**.print("  LED Brightness: ");

**Serial**.println(ledBrightness);

  delay(100);  // Wait for a short time before the next reading

}

Output:

