S1 Q1

```
import java.io.*;

// Custom decorator to convert input to lowercase
class LowerCaseInputStream extends FilterInputStream {

   protected LowerCaseInputStream(InputStream in) {
      super(in);
   }

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

public class s1 {

   public static void main(String[] args) {
      try {
         System.out.println("Enter text (uppercase will be converted to lowercase):");

         InputStream in = new LowerCaseInputStream(System.in);
         int ch;
         while ((ch = in.read()) != -1 && ch != '\n') {
            System.out.print((char) ch);
         }

      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

S1 Q2

Go to WOKWI -> Profile -> new Project -> ESP32 -> Starting Template Arduino ->

```
#include "WiFi.h"
void setup() {
  Serial.begin(115200);
```

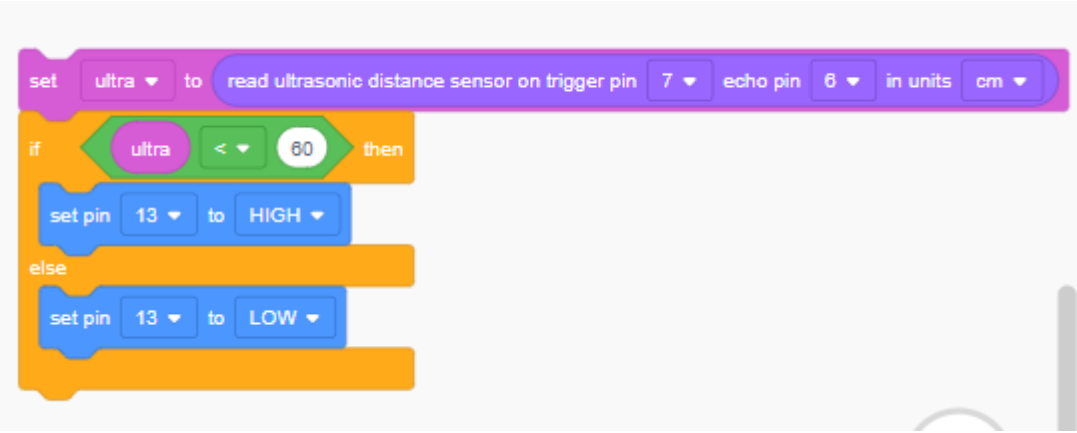
```
  Serial.println("Initializing WiFi...");
  WiFi.mode(WIFI_STA);
  Serial.println("Setup done!");
}

void loop() {
  Serial.println("Scanning...");

  // WiFi.scanNetworks will return the number of networks found
  int n = WiFi.scanNetworks();
  Serial.println("Scan done!");
  if (n == 0) {
    Serial.println("No networks found.");
  } else {
    Serial.println();
    Serial.print(n);
    Serial.println(" networks found");
    for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? " " : "*");
      delay(10);
    }
  }
  Serial.println("");

  // Wait a bit before scanning again
  delay(5000);
}
```

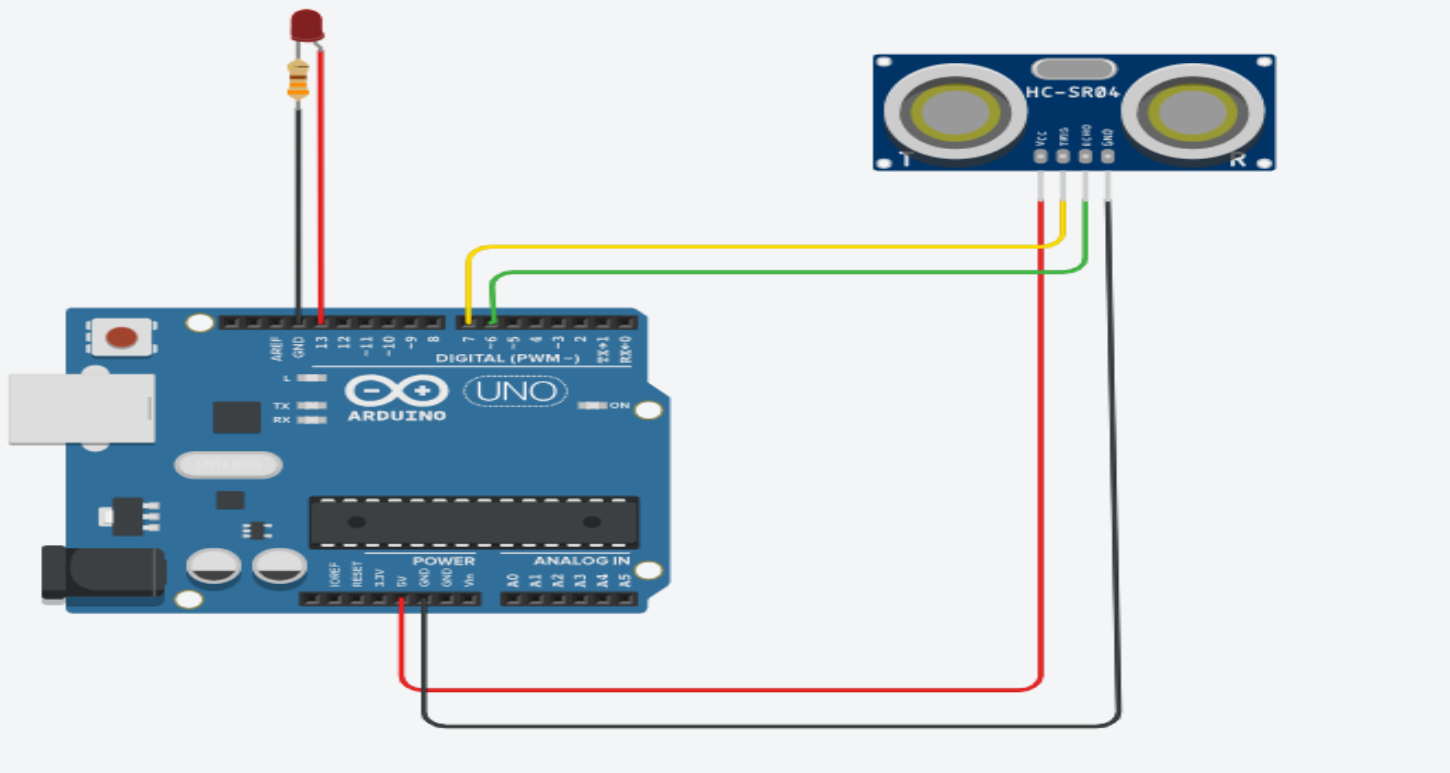
S2 Q1

```
// Simple thread-safe Singleton

class Singleton {

    private static Singleton instance;
```

```
    private Singleton() {
        System.out.println("Singleton instance created!");
    }

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

    public void showMessage(String threadName) {
        System.out.println("Hello from Singleton! Called by " + threadName);
    }
}

// Main class
public class s2 {

    public static void main(String[] args) {
        Runnable task = () -> {
            Singleton obj = Singleton.getInstance();
            obj.showMessage(Thread.currentThread().getName());
        };

        new Thread(task, "Thread-1").start();
        new Thread(task, "Thread-2").start();
        new Thread(task, "Thread-3").start();
    }
}
```

S2 Q2



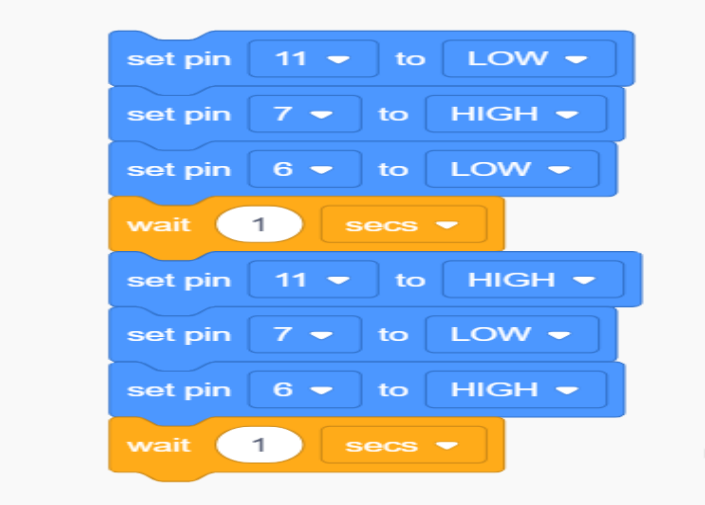


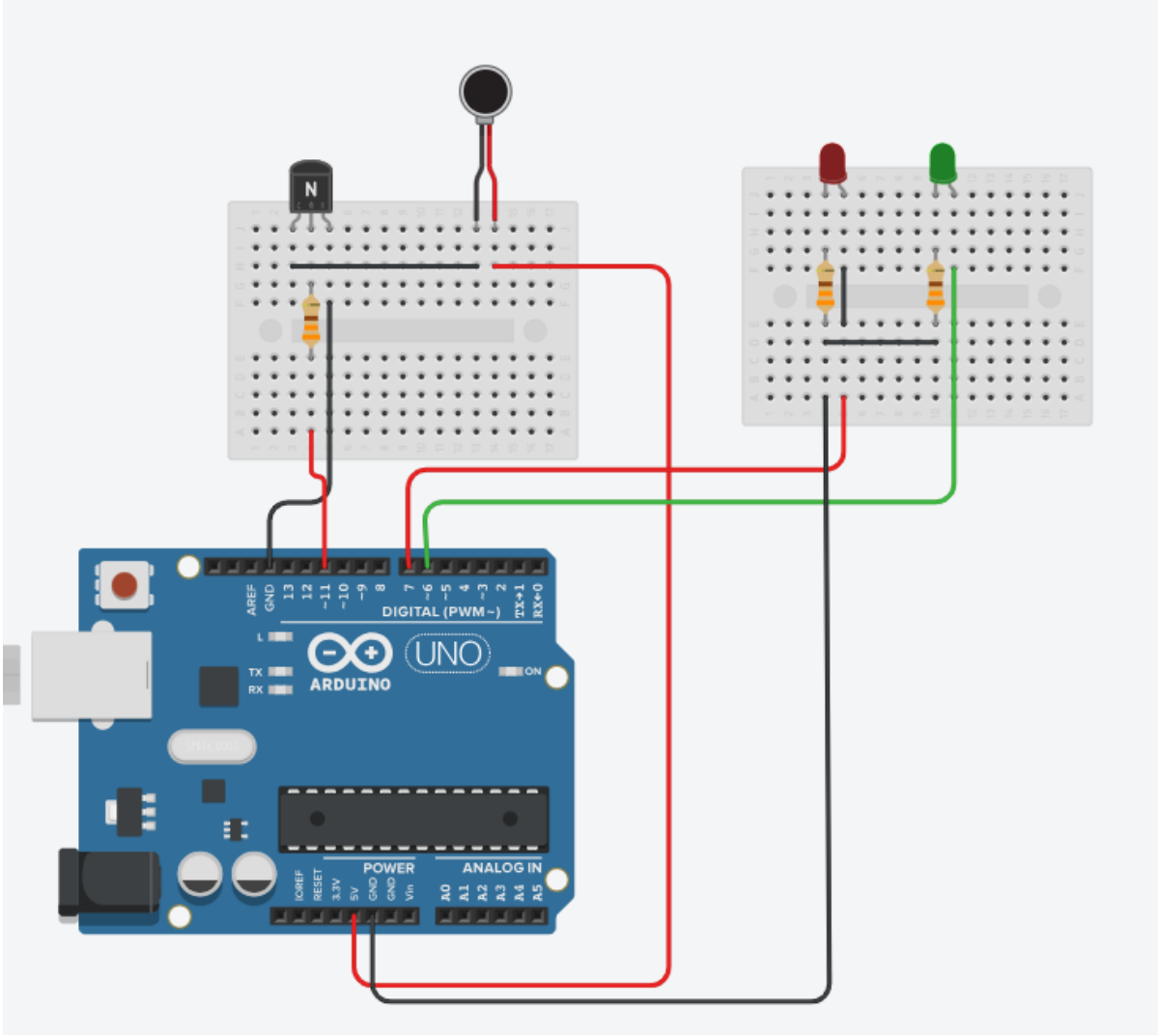
S3 Q1

```
// Observable class

class WeatherData {

   private float temperature, humidity, pressure;
   private CurrentConditionsDisplay observer; // single observer

   public void addObserver(CurrentConditionsDisplay o) {
      observer = o;
   }

   public void setMeasurements(float t, float h, float p) {
      this.temperature = t;
      this.humidity = h;
      this.pressure = p;
      notifyObserver();
   }

   private void notifyObserver() {
      if (observer != null) {
         observer.update(temperature, humidity, pressure);
      }
   }
```

```
}

// Observer class
class CurrentConditionsDisplay {

    public void update(float temperature, float humidity, float pressure) {
        System.out.println("Temp: " + temperature + "°C, Humidity: " + humidity + "%, Pressure: " + pressure + "
hPa");
    }
}

// Client
public class s3 {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
        CurrentConditionsDisplay display = new CurrentConditionsDisplay();

        weatherData.addObserver(display);

        // Simulate measurements
        weatherData.setMeasurements(25.5f, 65f, 1013f);
        weatherData.setMeasurements(26.2f, 70f, 1012f);
        weatherData.setMeasurements(24.8f, 90f, 1009f);
    }
}
```

S3 Q2





S5 Q1

```
import java.util.*;

// Adapter class: converts Enumeration to Iterator
class EnumerationIteratorAdapter<E> implements Iterator<E> {

   private Enumeration<E> enumeration;

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

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

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

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

// Main class
public class s5 {

    public static void main(String[] args) {
        Vector<String> v = new Vector<>();
        v.add("Apple");
        v.add("Banana");
        v.add("Cherry");

        Enumeration<String> enumeration = v.elements();

        // Use adapter to get an Iterator
        Iterator<String> iterator = new EnumerationIteratorAdapter<>(enumeration);

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

S5 Q2

NA

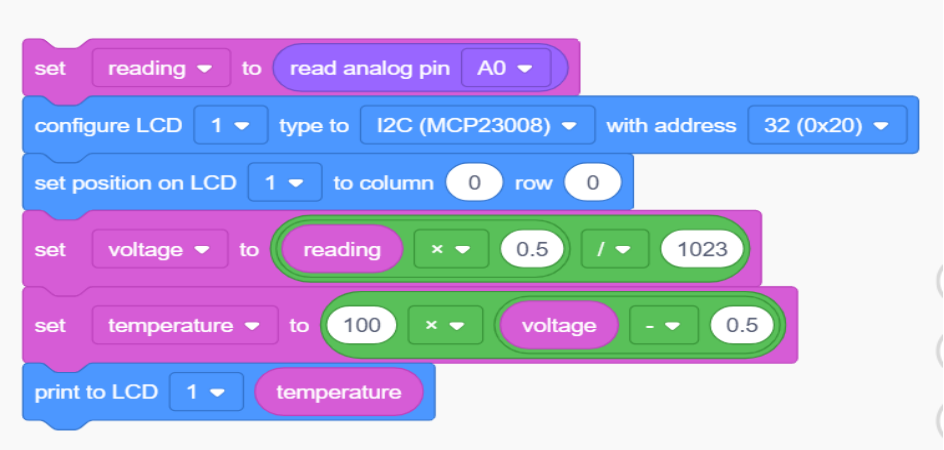
S6 Q1

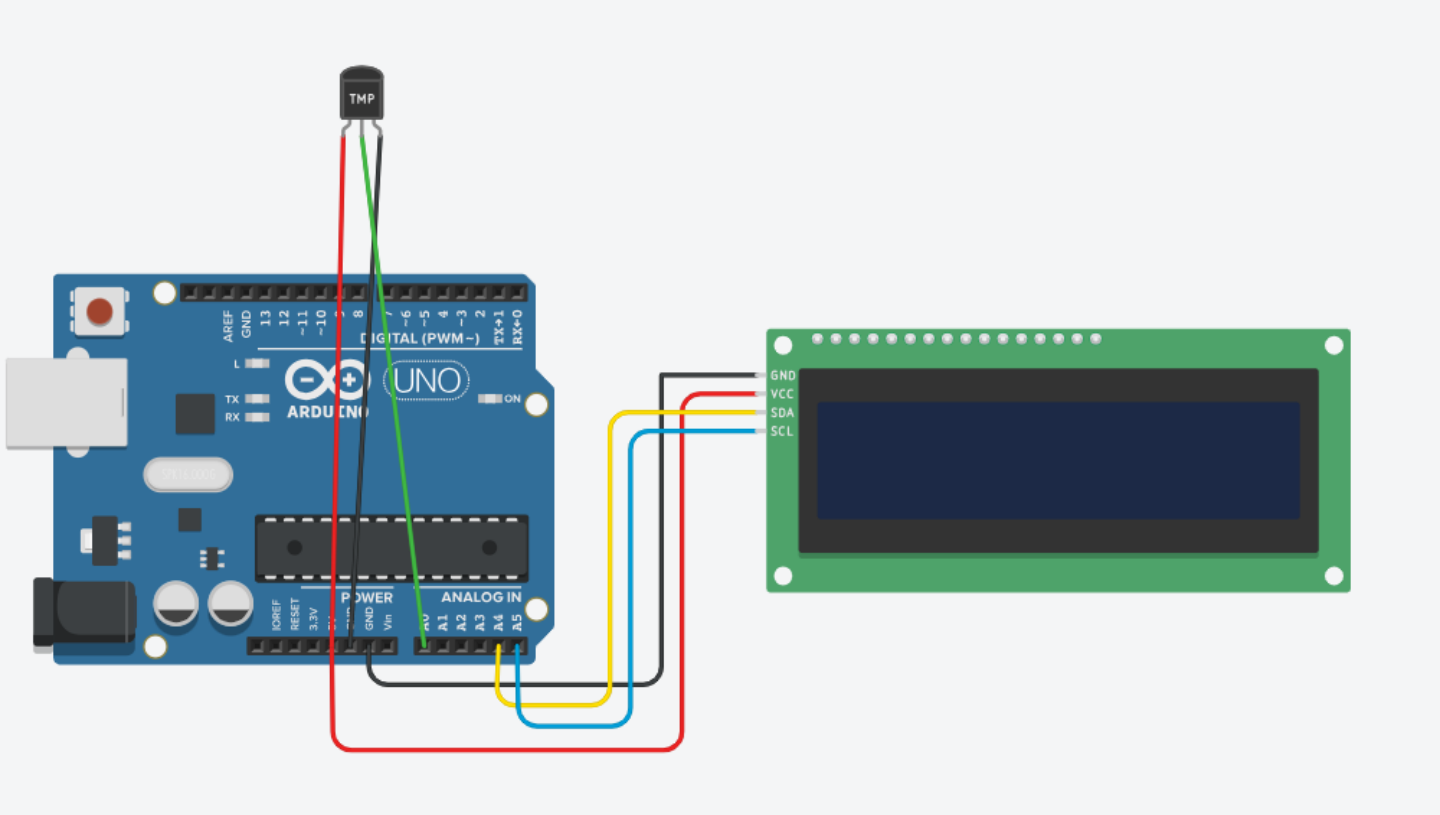
```
// Command Interface

interface Command {

    void execute();
}
```

```
// Receiver class
class Light {

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

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

// Concrete Commands
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();
    }
}

// Invoker class
class RemoteControl {
```

```
    private Command command;

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

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

// Client
public class s6 {

    public static void main(String[] args) {
        RemoteControl remote = new RemoteControl();
        Light livingRoomLight = new Light();

        // Turn ON the light
        remote.setCommand(new LightOnCommand(livingRoomLight));
        remote.pressButton();

        // Turn OFF the light
        remote.setCommand(new LightOffCommand(livingRoomLight));
        remote.pressButton();
    }
}
```

S6 Q2






```
#include <Adafruit_LiquidCrystal.h>

int reading = 0;
float voltage = 0;
float temperature = 0;

Adafruit_LiquidCrystal lcd_1(0);

void setup()
{
  pinMode(A0, INPUT);
  lcd_1.begin(16, 2);
}

void loop()
{
  reading = analogRead(A0);

  // Convert ADC to voltage
  voltage = reading * (5.0 / 1023.0);

  // Convert voltage to temperature (TMP36)
  temperature = (voltage - 0.5) * 100.0;
```

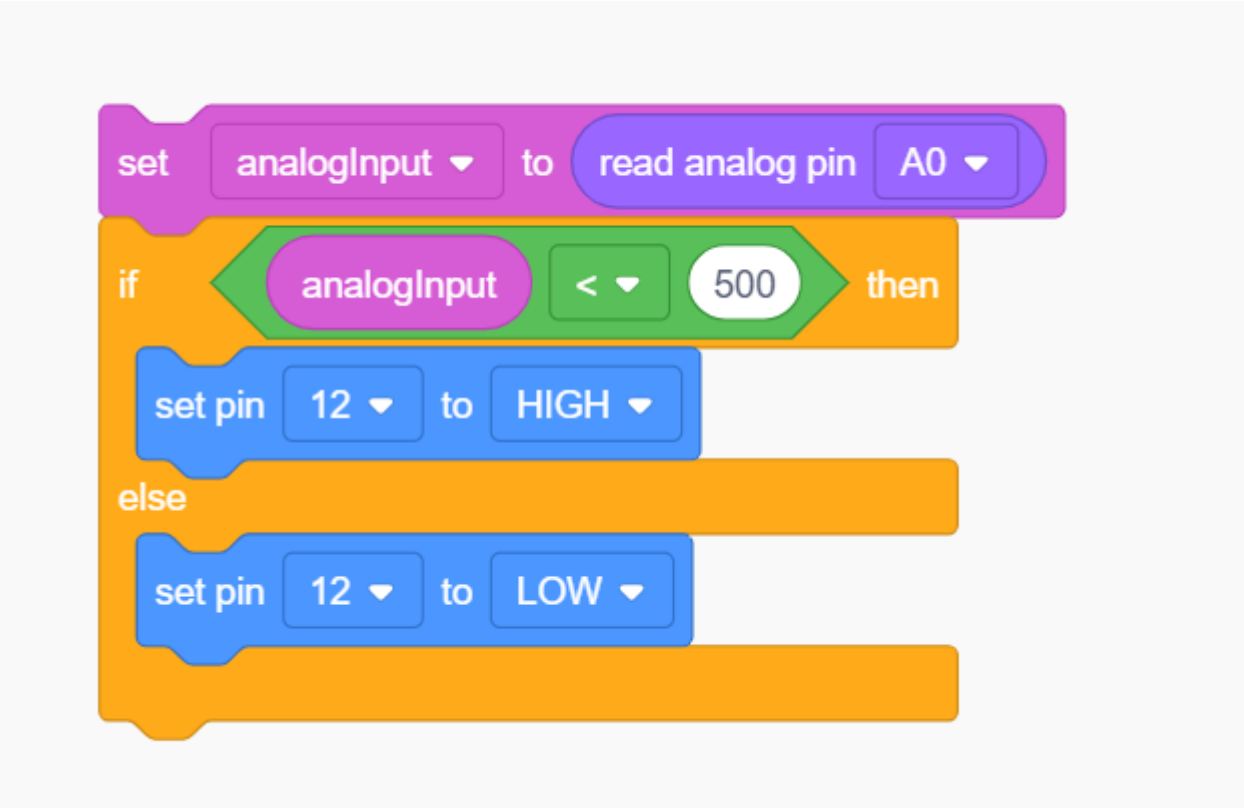
```
  lcd_1.setCursor(0, 1);
  lcd_1.print("Temp: ");
  lcd_1.print(temperature);
  lcd_1.print(" C   "); // extra spaces prevent ghost digits

  delay(250);
}
```

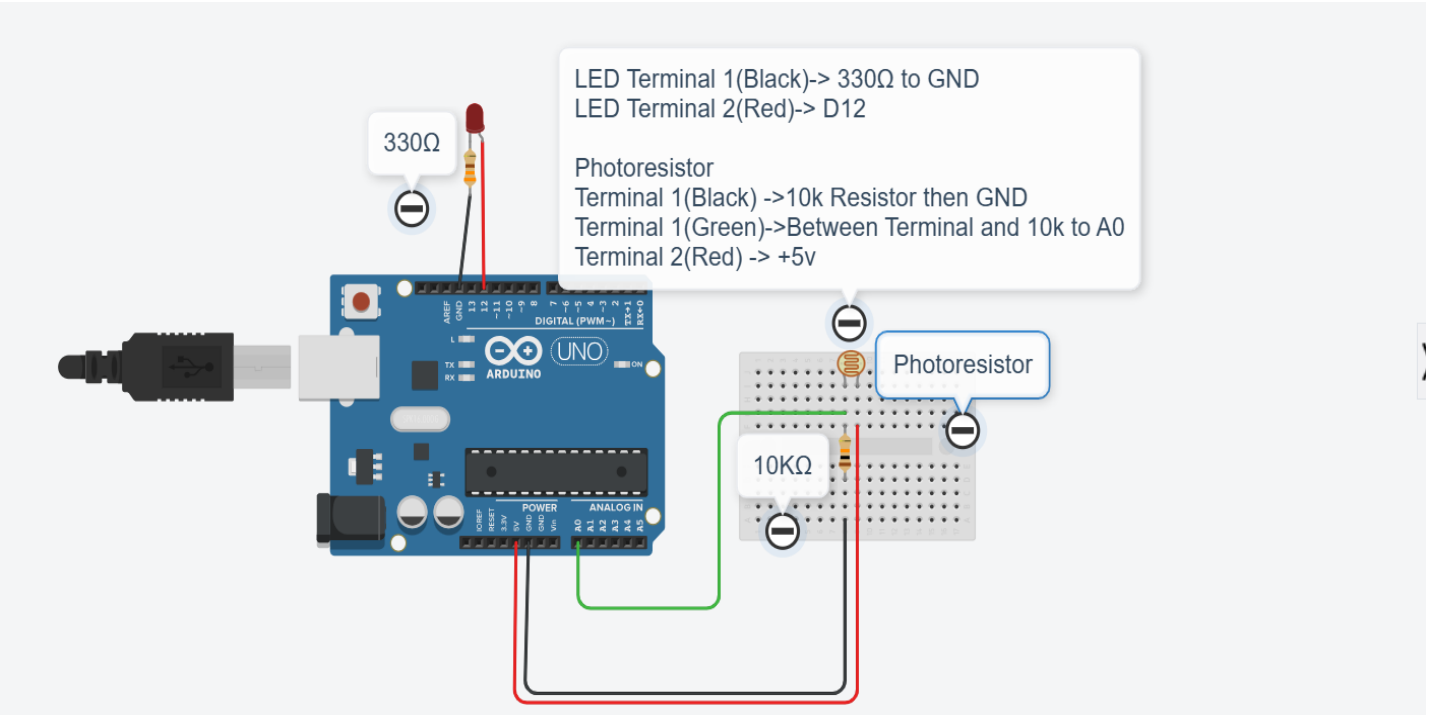
S7 Q1

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

// Receiver
class CeilingFan {
   void on() { System.out.println("Ceiling Fan ON"); }
   void off() { System.out.println("Ceiling Fan OFF"); }
}

// Client / Main
public class s7 {
   public static void main(String[] args) {
      CeilingFan fan = new CeilingFan();

      // Commands as simple anonymous classes
      Command fanOn = new Command() {
         public void execute() { fan.on(); }
         public void undo() { fan.off(); }
      };

      Command fanOff = new Command() {
         public void execute() { fan.off(); }
         public void undo() { fan.on(); }
      };

      // Use commands
      fanOn.execute();   // Turn fan ON
      fanOn.undo();      // Undo → Turn fan OFF
```

```
        fanOff.execute();  // Turn fan OFF
        fanOff.undo();     // Undo → Turn fan ON
    }
}
```

S7 Q2





S10 Q1

```
// Behavior interfaces

interface FlyBehavior {

   void fly();
}

interface QuackBehavior {

   void quack();
}

// Duck class
class Duck {

   FlyBehavior flyBehavior;
   QuackBehavior quackBehavior;

   Duck(FlyBehavior fly, QuackBehavior quack) {
      this.flyBehavior = fly;
      this.quackBehavior = quack;
   }

   void performFly() {
      flyBehavior.fly();
   }

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

   void setFlyBehavior(FlyBehavior fb) {
      this.flyBehavior = fb;
   }

   void setQuackBehavior(QuackBehavior qb) {
      this.quackBehavior = qb;
   }
}

// Main class
```

```
public class s10 {

   public static void main(String[] args) {
      // Mallard Duck
      Duck mallard = new Duck(
            () -> System.out.println("Flying with wings!"),
            () -> System.out.println("Quack!")
      );
      System.out.println("Mallard Duck:");
      mallard.performFly();
      mallard.performQuack();

      // Rubber Duck
      Duck rubberDuck = new Duck(
            () -> System.out.println("I cannot fly!"),
            () -> System.out.println("Squeak!")
      );
      System.out.println("\nRubber Duck:");
      rubberDuck.performFly();
      rubberDuck.performQuack();

      // Change behavior at runtime
      System.out.println("\nRubber Duck learns to fly!");
      rubberDuck.setFlyBehavior(() -> System.out.println("Flying with wings now!"));
      rubberDuck.performFly();
   }
}
```

S10 Q2

a) Using For Loop :-

```
name = input("Enter name: ")
n = int(input("Enter n: "))

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

Using While Loop:-

```
name = input("Enter name: ")
n = int(input("Enter n: "))

i = 0
while i < n:
```

```
    print(name)
    i += 1

b)
try:
    a = int(input("Enter numerator: "))
    b = int(input("Enter denominator: "))
    result = a / b
    print("Result:", result)
except ZeroDivisionError:
    print("Error: Cannot divide by zero!")

c)
import time
from datetime import datetime

for i in range(10):
    print("Current Time:", datetime.now())
    time.sleep(10)   # wait 10 seconds

d)
file = open("data.txt", "r")

for line in file:
    words = line.split()
    print("Line:", line.strip())
    print("Word Count:", len(words))

file.close()

S12 Q1
// Component interface
interface Car {
    void assemble();
}

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

```
}

// Concrete Decorator - Sports Car
class SportsCar implements Car {
   private Car car;

   public SportsCar(Car car) {
      this.car = car;
   }

   @Override
   public void assemble() {
      car.assemble();
      System.out.print(" + Sports Car features");
   }
}

// Concrete Decorator - Luxury Car
class LuxuryCar implements Car {
   private Car car;

   public LuxuryCar(Car car) {
      this.car = car;
   }

   @Override
   public void assemble() {
      car.assemble();
      System.out.print(" + Luxury Car features");
   }
}

// Client code
public class s12 {
   public static void main(String[] args) {
      Car sportsCar = new SportsCar(new BasicCar());
      System.out.print("SportsCar: ");
      sportsCar.assemble();
      System.out.println();

      Car luxuryCar = new LuxuryCar(new BasicCar());
      System.out.print("LuxuryCar: ");
```

```
        luxuryCar.assemble();
        System.out.println();

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

S12 Q2

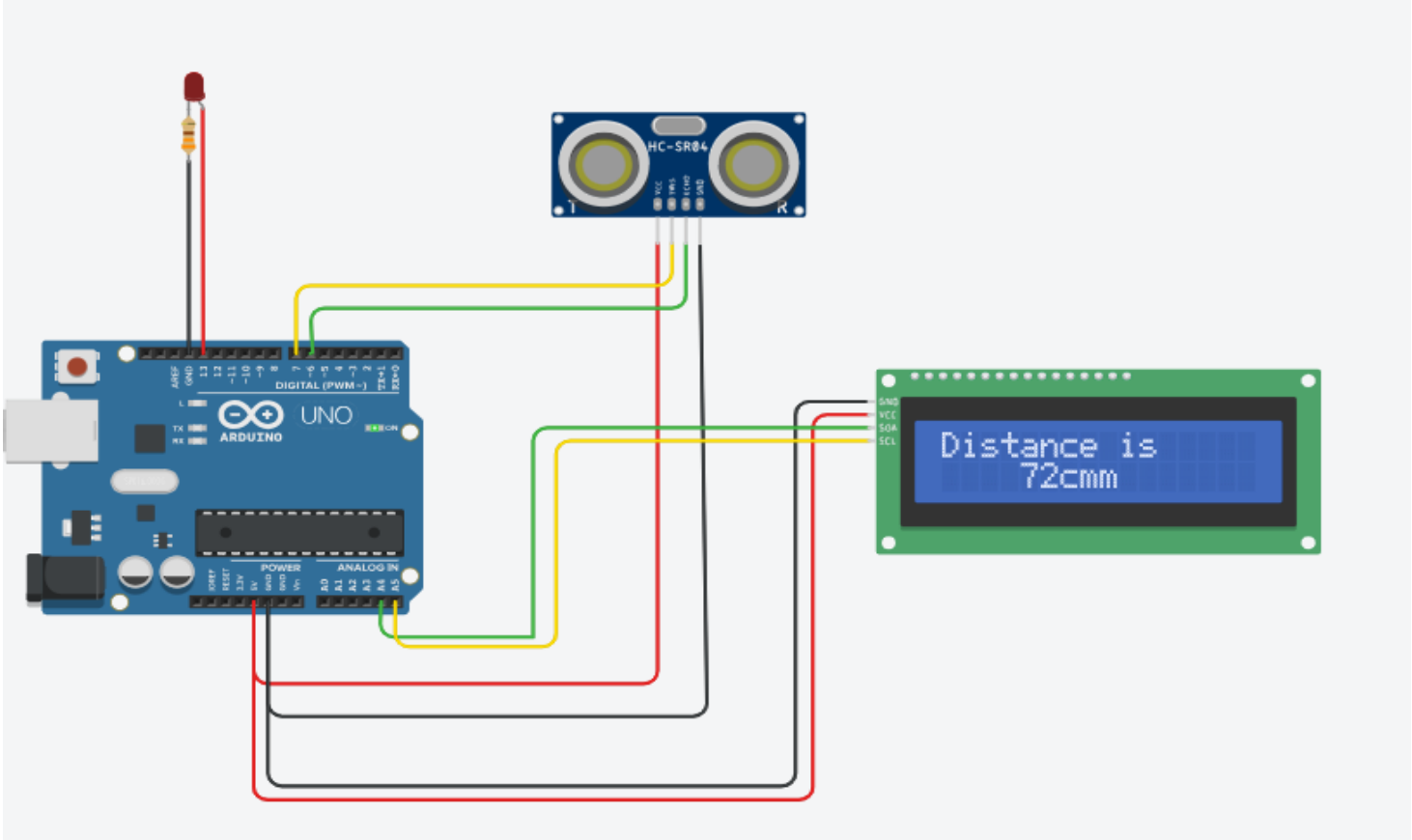
Same as S1 Q1

S13 Q1

```
// Volt class
class Volt {
    private int volts;

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

    public int getVolts() {
        return volts;
    }
}

// Socket class (gives 120V)
class Socket {
    public Volt getVolt() {
        return new Volt(120);
    }
}

// Adapter interface
interface SocketAdapter {
    Volt get120Volt();
    Volt get12Volt();
    Volt get3Volt();
}
```

```
// Class Adapter (using inheritance)
class SocketClassAdapterImpl extends Socket implements SocketAdapter {

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

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

    @Override
    public Volt get3Volt() {
        return convertVolt(getVolt(), 40);
    }

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

// Client code
public class s13 {
    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("Laptop needs 12V: " + v12.getVolts() + "V");
        System.out.println("Home appliance needs 120V: " + v120.getVolts() + "V");
    }
}
```

S13 Q2



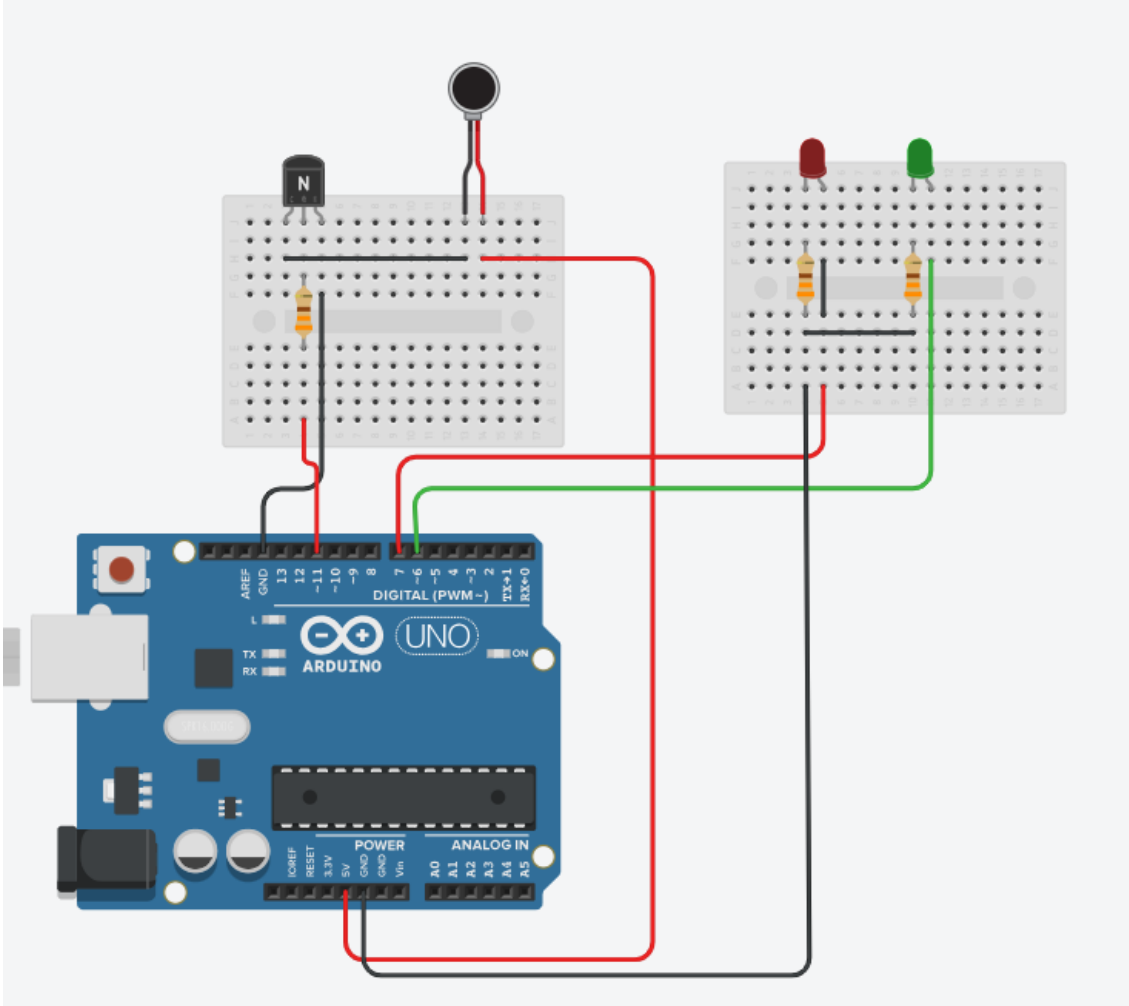

```
set ultra to read ultrasonic distance sensor on trigger pin 7 echo pin 6 in units cm
print to serial monitor hello world with newline
configure LCD 1 type to I2C (MCP23008) with address 32 (0x20)
set position on LCD 1 to column 0 row 0
print to LCD 1 Distance is
configure LCD 1 type to I2C (MCP23008) with address 32 (0x20)
set position on LCD 1 to column 0 row 1
print to LCD 1 ultra
set position on LCD 1 to column 6 row 1
print to LCD 1 cm
wait 0.5 secs
if ultra < 60 then
  set pin 13 to HIGH
else
  set pin 13 to LOW
```

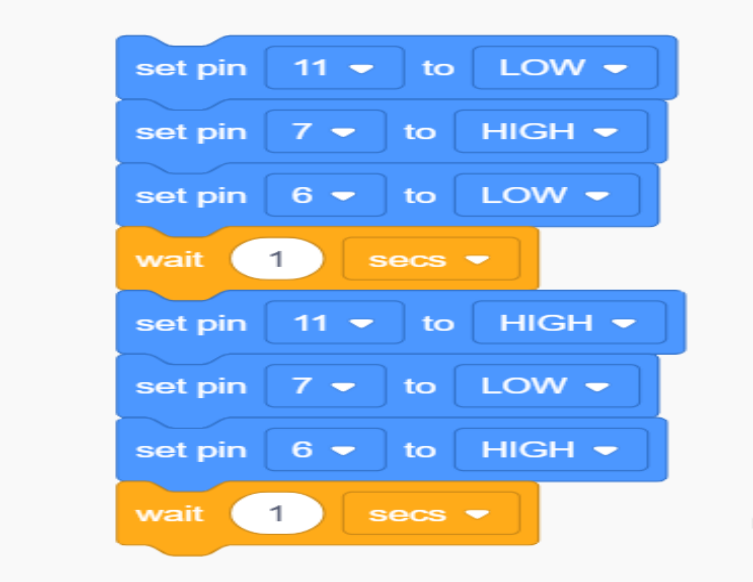
S14 Q1

```
public class S14 {

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

   public static void main(String[] args) {

      // Receivers
      class Light {
         void on() { System.out.println("Light ON"); }
         void off() { System.out.println("Light OFF"); }
      }

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

      class Stereo {
         void on() { System.out.println("Stereo ON"); }
         void cd() { System.out.println("Stereo CD"); }
         void volume(int v) { System.out.println("Stereo Volume " + v); }
      }

      Light light = new Light();
      GarageDoor garage = new GarageDoor();
      Stereo stereo = new Stereo();

      // Commands executed directly
      Command lightOn = () -> light.on();
      Command lightOff = () -> light.off();
      Command garageUp = () -> garage.up();
      Command stereoOn = () -> {
         stereo.on();
         stereo.cd();
         stereo.volume(11);
      };

      // Run commands
      lightOn.execute();
```

```
        lightOff.execute();
        garageUp.execute();
        stereoOn.execute();
    }
}
```

S14 Q2





S16 Q1

```
import java.util.*;

public class s16 {

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      // List of observers (each observer is just a lambda function)
      List<java.util.function.IntConsumer> observers = new ArrayList<>();
      observers.add(n -> System.out.println("Binary: " + Integer.toBinaryString(n)));
      observers.add(n -> System.out.println("Octal: " + Integer.toOctalString(n)));
      observers.add(n -> System.out.println("Hexadecimal: " + Integer.toHexString(n).toUpperCase()));

      while (true) {
         System.out.print("\nEnter a decimal number (-1 to exit): ");
         int num = sc.nextInt();
         if (num == -1) {
            break;
         }

         // Notify all observers
         for (var obs : observers) {
            obs.accept(num);
         }
      }
      sc.close();
   }
}
```

S16 Q2

NA

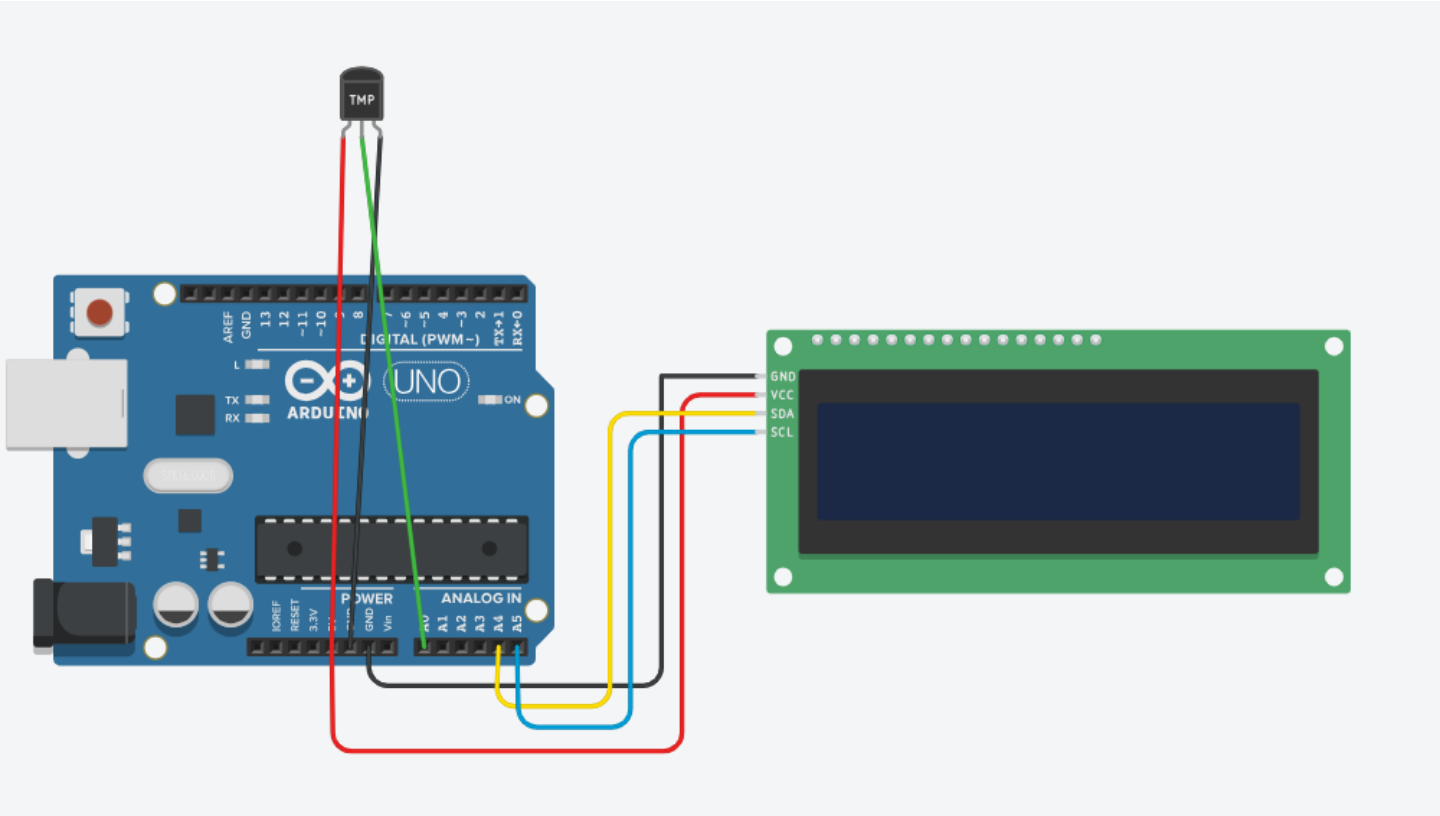
S17 Q1

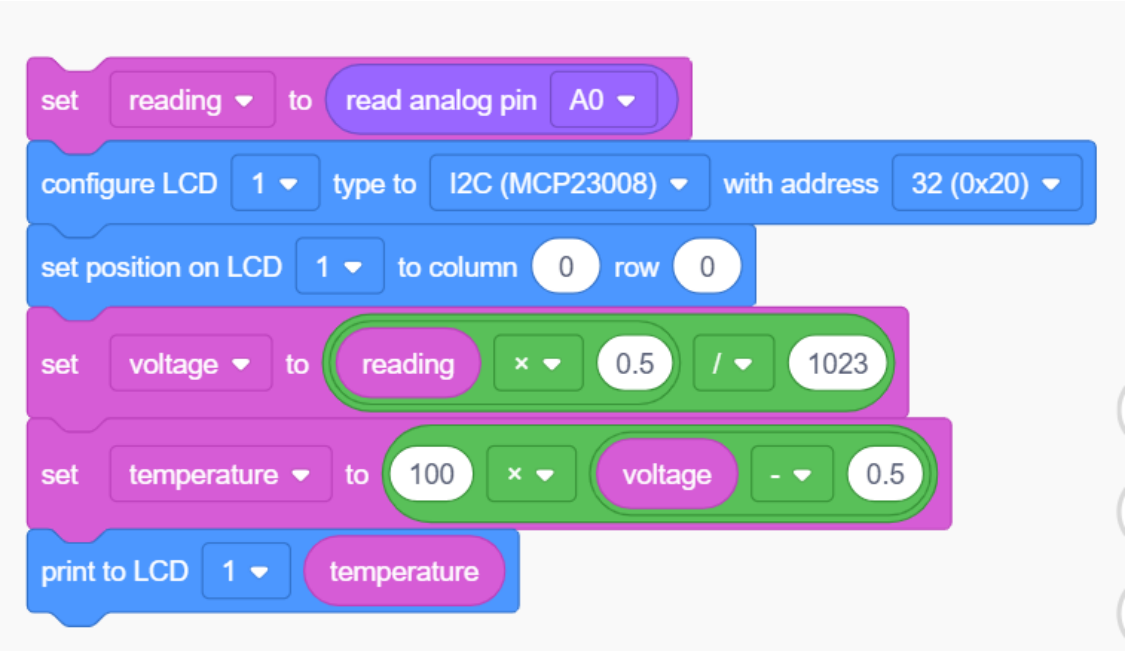
```
// Shape interface
interface Shape {
   void draw();
}

// Concrete shapes
class Circle implements Shape {
   public void draw() {
```

```
        System.out.println("Drawing a Circle");
    }
}

class Rectangle implements Shape {
    public void draw() {
        System.out.println("Drawing a Rectangle");
    }
}

// Abstract Factory
abstract class AbstractFactory {
    abstract Shape getShape(String shapeType);
}

// ShapeFactory
class ShapeFactory extends AbstractFactory {
    Shape getShape(String shapeType) {
        if (shapeType == null) return null;
        if (shapeType.equalsIgnoreCase("CIRCLE")) return new Circle();
        if (shapeType.equalsIgnoreCase("RECTANGLE")) return new Rectangle();
        return null;
    }
}

// Factory producer
class FactoryProducer {
    static AbstractFactory getFactory(String choice) {
        if (choice.equalsIgnoreCase("SHAPE")) return new ShapeFactory();
        return null;
    }
}

// Client code
public class s17 {
    public static void main(String[] args) {
        AbstractFactory shapeFactory = FactoryProducer.getFactory("SHAPE");

        // Get a Circle and draw
        Shape shape1 = shapeFactory.getShape("CIRCLE");
        shape1.draw();
```

```
        // Get a Rectangle and draw
        Shape shape2 = shapeFactory.getShape("RECTANGLE");
        shape2.draw();
    }
}
```

S17 Q2





S18 Q1

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

// Observable class
class WeatherData extends Observable {
   private float temperature, humidity, pressure;

   public void setMeasurements(float t, float h, float p) {
      this.temperature = t;
      this.humidity = h;
      this.pressure = p;
      setChanged();
      notifyObservers();
   }

   public float getTemperature() { return temperature; }
   public float getHumidity() { return humidity; }
   public float getPressure() { return pressure; }
}

// Observer class
class CurrentConditionsDisplay implements Observer {
   @Override
   public void update(Observable o, Object arg) {
      if (o instanceof WeatherData) {
         WeatherData wd = (WeatherData) o;
         System.out.println("Temp: " + wd.getTemperature() + "°C, Humidity: " + wd.getHumidity() + "%,
Pressure: " + wd.getPressure() + " hPa");
      }
   }
}

// Client
public class s18 {
   public static void main(String[] args) {
      WeatherData weatherData = new WeatherData();
      weatherData.addObserver(new CurrentConditionsDisplay());

      // simulate measurements
      weatherData.setMeasurements(25.5f, 65f, 1013f);
      weatherData.setMeasurements(26.2f, 70f, 1012f);
      weatherData.setMeasurements(24.8f, 90f, 1009f);
```

```
    }
}
```

S18 Q2
Same as S7 q2

S20 Q1
Same as S1 Q1
S20 Q2
Same as S1 Q2