Rayat Shikshan Sanstha's

# R.B. Narayanrao Borawake College, Shrirampur

**(Autonomous)**

**National Education Policy**

(Affiliated to Savitribai Phule Pune University)

**Two Years Degree Program in Computer Science(Faculty of Science and Technology)**

**Syllabus under Autonomy and National Education Policy**

**M. Sc. (Computer Science) Part- II**

# Practical Assignment Lab Book

**Choice Based Credit System [CBCS] Syllabus for National Education Policy to be implemented from Academic Year 2024-2025**

# Software Architecture & Design Pattern (CS-MJ-631T)

## Assignment Completion Sheet

Name of the Student :

Roll No :

| Sr.No | Assignment Title | Marks Obtained | Signature of Instructor |
|---|---|---|---|
| 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() | | |
| 2 | Write a Java Program to implement 1/0 Decorator for converting uppercase letters to lower case letters. | | |
| 3 | 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 NyStyleCheese Pizza, ChicagoStyleCheesePizza etc. | | |
| 4 | Write a Java Program to implement Singleton pattern for multithreading. | | |
| 5 | Write a Java Program to implement command pattern to test Remote Control. | | |
| 6 | Write a Java Program to implement undo command to test Ceiling fan. | | |
| 7 | Write a Java Program to implement Adapter pattern for Enumeration iterator. | | |
| 8 | Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu. | | |

| | | | |
|---|---|---|---|
| 9 | 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. For actions we need to implement methods to insert a quarter, remove a quarter, turning the crank and display gumball | | |
| 10 | Write a java program to implement Adapter pattern to design Heart Model to Beat | | |
| 11 | Design simple HR Application using Spring Framework | | |

**Total Marks :**

**Converted into 10 Marks :**

**Date:**

**Signature of In Charge** **Head**

**Internal Examiner** **External Examiner**

# Assignment No 1

**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;

   public 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();

   }

private void measurementsChanged() {

      setChanged();

      notifyObservers();

   }  public float getTemperature() {

      return temperature;

   }  public float getHumidity() {

      return humidity;

   }

public float getPressure() {
 return pressure;
    }
public static void main(String[] args) {
        WeatherStation weatherStation = new WeatherStation();
       Observer display = (o, arg) -> {
            if (o instanceof WeatherStation) {
                WeatherStation ws = (WeatherStation) o;
                System.out.println("Current weather conditions:");
                System.out.printf("Temperature: %.2f°F%n", ws.getTemperature());
                System.out.printf("Humidity: %.2f%%%n", ws.getHumidity());
                System.out.printf("Pressure: %.2f hPa%n", ws.getPressure());
            }
        };
      weatherStation.addObserver(display);
        weatherStation.setMeasurements(80, 65, 30.4f);
        weatherStation.setMeasurements(82, 70, 29.2f);
        weatherStation.setMeasurements(78, 90, 29.2f);
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No .2

**Q.2) Write a Java Program to implement 1/0 Decorator for converting uppercase letters to lower case letters.**

```
interface Text {
   String getContent();
}
class PlainText implements Text {
   private String content;
   public PlainText(String content) {
      this.content = content;
   }
@Override
   public String getContent() {
      return content;
   }
}
class LowerCaseDecorator implements Text {
   private Text text;
   public LowerCaseDecorator(Text text) {
      this.text = text;
   }
@Override
   public String getContent() {
      return text.getContent().toLowerCase();
```

```
    }
}
public class DecoratorPatternExample {
    public static void main(String[] args) {
        Text plainText = new PlainText("Hello World!");
        Text lowerCaseText = new LowerCaseDecorator(plainText);

        System.out.println("Original: " + plainText.getContent());
        System.out.println("Lowercase: " + lowerCaseText.getContent());
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No. 3

**Q.3) 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 NyStyleCheese Pizza, ChicagoStyleCheesePizza etc**

```
abstract class Pizza {
  String name;
 void prepare() {
       System.out.println("Preparing " + name);
    }   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;
    }
}
class NYStyleCheesePizza extends Pizza {
    public NYStyleCheesePizza() {
       name = "New York Style Cheese Pizza";
    }
}
```

```
class ChicagoStyleCheesePizza extends Pizza {
   public ChicagoStyleCheesePizza() {
      name = "Chicago Style Cheese Pizza";
   }
}
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);
}
class NYStylePizzaStore extends PizzaStore {
   protected Pizza createPizza(String type) {
      if (type.equals("cheese")) {
         return new NYStyleCheesePizza();
      }
      return null;
   }
}
class ChicagoStylePizzaStore extends PizzaStore {
    protected Pizza createPizza(String type) {
        if (type.equals("cheese")) {
            return new ChicagoStyleCheesePizza();
        }return null;
    }
}
public class PizzaFactoryMethod {
    public static void main(String[] args) {
        PizzaStore nyStore = new NYStylePizzaStore();
        PizzaStore chicagoStore = new ChicagoStylePizzaStore();
        Pizza pizza1 = nyStore.orderPizza("cheese");
        System.out.println("Ethan ordered a " + pizza1.getName() + "\n");
         Pizza pizza2 = chicagoStore.orderPizza("cheese");
        System.out.println("Joel ordered a " + pizza2.getName() + "\n");
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No .4

**Q.4) Write a Java Program to implement Singleton pattern for multithreading**

```
class Singleton {
   private static volatile Singleton instance;
    private Singleton() {
   }
public static Singleton getInstance() {
       if (instance == null) {
           synchronized (Singleton.class) {
               if (instance == null) {
                   instance = new Singleton();
               }
           }
       } return instance;
   }
public void showMessage() {
       System.out.println("Hello from Singleton!");
   }
}
public class SingletonPatternExample {
   public static void main(String[] args) {
       Runnable task = () -> {
           Singleton singleton = Singleton.getInstance();
 singleton.showMessage();
```

```
        };
        Thread thread1 = new Thread(task);
        Thread thread2 = new Thread(task);
        Thread thread3 = new Thread(task);
        thread1.start();
        thread2.start();
        thread3.start();
            try {
          thread1.join();
          thread2.join();
          thread3.join();
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No .5

**Q.5) Write a Java Program to implement command pattern to test Remote Control.**

```
interface Command {
    void execute();
}
class Light {
    public void on() {
        System.out.println("The light is ON");
    }
 public void off() {
        System.out.println("The light is OFF");
    }
}
class LightOnCommand implements Command {
    private Light light;
   public LightOnCommand(Light light) {
        this.light = light;
    }
  @Override
    public void execute() {
        light.on();
    }
}
class LightOffCommand implements Command {
    private Light light;
public LightOffCommand(Light light) {
        this.light = light;
    }
@Override
    public void execute() {
        light.off();
    }
}
class RemoteControl {
    private Command command;
    public void setCommand(Command command) {
        this.command = command;
    }
 public void pressButton() {
        command.execute();
    }
}
public class CommandPatternDemo {
    public static void main(String[] args) {
        Light livingRoomLight = new Light();
        Command lightOn = new LightOnCommand(livingRoomLight);
        Command lightOff = new LightOffCommand(livingRoomLight);

        RemoteControl remote = new RemoteControl();
        remote.setCommand(lightOn);
        remote.pressButton();
        remote.setCommand(lightOff);
        remote.pressButton();
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No.6

**Q.6) Write a Java Program to implement undo command to test Ceiling fan.**

```
interface Command {
   void execute();
   void undo();
}
class CeilingFan {
   private String location;
   private String speed;
  public CeilingFan(String location) {
      this.location = location;
      this.speed = "OFF";
   }
public void setSpeed(String speed) {
      this.speed = speed;
      System.out.println(location + " Ceiling Fan is " + speed);
   }
public String getSpeed() {
      return speed;
   }
}
class CeilingFanCommand implements Command {
   private CeilingFan fan;
   private String prevSpeed;
```

```
public CeilingFanCommand(CeilingFan fan) {
        this.fan = fan;
    }
 public void execute() {
        prevSpeed = fan.getSpeed();
        fan.setSpeed("ON");
    }

    public void undo() {
        fan.setSpeed(prevSpeed);
    }
}
class RemoteControl {
    private Command command;

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

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

    public void pressUndo() {
        command.undo();
    }
}
public class UndoCommandDemo {
    public static void main(String[] args) {
        CeilingFan fan = new CeilingFan("Living Room");
        CeilingFanCommand fanCommand = new CeilingFanCommand(fan);
        RemoteControl remote = new RemoteControl();
remote.setCommand(fanCommand);
        remote.pressButton();  // Turn fan ON
        remote.pressUndo();    // Undo (turn fan OFF)
}}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No 7

**Q.7) Write a Java Program to implement Adapter pattern for Enumeration iterator.**

```
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Vector;
class EnumerationIterator<T> implements Iterator<T> {
    private Enumeration<T> enumeration;

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

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

    @Override
    public T next() {
        return enumeration.nextElement();
    }
}
public class AdapterPatternDemo {
    public static void main(String[] args) {
        Vector<String> vector = new Vector<>();
        vector.add("Apple");
        vector.add("Banana");
        vector.add("Cherry");

        Enumeration<String> enumeration = vector.elements();
        Iterator<String> iterator = new EnumerationIterator<>(enumeration);

        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No.8

**Q.8) Write a Java Program to implement Iterator Pattern for Designing Menu like Breakfast, Lunch or Dinner Menu.**

```
import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

class MenuItem {

   private String name, description;

 public MenuItem(String name, String description) {

      this.name = name; this.description = description;

   } public String getName() { return name; }

   public String getDescription() { return description; }

}

interface Menu {

   Iterator<MenuItem> createIterator();

}

class BreakfastMenu implements Menu {

   private List<MenuItem> items = new ArrayList<>();

   public BreakfastMenu() { items.add(new MenuItem("Pancakes", "Fluffy
pancakes")); }

   public Iterator<MenuItem> createIterator() { return items.iterator(); }

}class LunchMenu implements Menu {

   private List<MenuItem> items = new ArrayList<>();

   public LunchMenu() { items.add(new MenuItem("Burger", "Juicy beef burger")); }

   public Iterator<MenuItem> createIterator() { return items.iterator(); }

}class DinnerMenu implements Menu {

   private List<MenuItem> items = new ArrayList<>();
```

```
    public DinnerMenu() { items.add(new MenuItem("Steak", "Grilled steak")); }
    public Iterator<MenuItem> createIterator() { return items.iterator(); }
}class Waitress {
    private Menu[] menus;
    public Waitress(Menu[] menus) { this.menus = menus; }
    public void printMenus() {
        for (Menu menu : menus) {
            Iterator<MenuItem> iterator = menu.createIterator();
            while (iterator.hasNext()) {
                MenuItem item = iterator.next();
                System.out.println(item.getName() + ": " + item.getDescription());
            }
            System.out.println();
        }
    }
}
public class IteratorPatternDemo {
    public static void main(String[] args) {
        Menu[] menus = { new BreakfastMenu(), new LunchMenu(), new DinnerMenu()
};
        new Waitress(menus).printMenus();
    }}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No .9

**Q.9) 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. For actions we need to implement methods to insert a quarter, remove a quarter, turning the crank and display gumball.**

```
class GumballMachine {
   State state;
   int count;
interface State {
      void insertQuarter();
      void ejectQuarter();
      void turnCrank();
      void dispense();
   }
class NoQuarterState implements State {
      public void insertQuarter() { System.out.println("Quarter inserted."); state
= hasQuarterState; }
      public void ejectQuarter() { System.out.println("No quarter to eject."); }
      public void turnCrank() { System.out.println("Insert a quarter first."); }
      public void dispense() { System.out.println("Pay first."); }
   }class HasQuarterState implements State {
      public void insertQuarter() { System.out.println("Already has a quarter.");
}
      public void ejectQuarter() { System.out.println("Quarter ejected."); state =
noQuarterState; }
      public void turnCrank() { System.out.println("Crank turned."); state =
soldState; }
public void dispense() { System.out.println("No gumball dispensed."); }
  }
class SoldState implements State {
      public void insertQuarter() { System.out.println("Please wait."); }
      public void ejectQuarter() { System.out.println("Too late!"); }
      public void turnCrank() { System.out.println("Turning again does
nothing."); }
      public void dispense() {
        if (count > 0) {
          count--;
          System.out.println("Gumball dispensed.");
          state = count > 0 ? noQuarterState : soldOutState;
        } else {
          System.out.println("Out of gumballs!");
          state = soldOutState;
        }
      }
  }
class SoldOutState implements State {
      public void insertQuarter() { System.out.println("Sold out!"); }
      public void ejectQuarter() { System.out.println("No quarter to eject."); }
      public void turnCrank() { System.out.println("Sold out!"); }
      public void dispense() { System.out.println("No gumball dispensed."); }
  }
NoQuarterState noQuarterState = new NoQuarterState();
  HasQuarterState hasQuarterState = new HasQuarterState();
```

```
    SoldState soldState = new SoldState();
    SoldOutState soldOutState = new SoldOutState();
     public GumballMachine(int count) {
        this.count = count > 0 ? count : 0;
        state = count > 0 ? noQuarterState : soldOutState;
    }
void insertQuarter() { state.insertQuarter(); }
    void ejectQuarter() { state.ejectQuarter(); }
    void turnCrank() { state.turnCrank(); state.dispense(); }
public static void main(String[] args) {
        GumballMachine machine = new GumballMachine(5);
        machine.insertQuarter();
        machine.turnCrank();
        machine.insertQuarter();
        machine.ejectQuarter();
        machine.turnCrank();
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No 10

**Q.10) Write a java program to implement Adapter pattern to design Heart Model to Beat**

```
interface Heart {
    void beat();
}

// Adaptee class
class HeartModel {
    public void startBeat() {
        System.out.println("Heart is beating...");
    }

    public void stopBeat() {
        System.out.println("Heart has stopped beating.");
    }
}

// Adapter class
class HeartAdapter implements Heart {
    private HeartModel heartModel;

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

    @Override
    public void beat() {
        heartModel.startBeat();
    }
}
public class AdapterPatternDemo {
    public static void main(String[] args) {
        HeartModel heartModel = new HeartModel();
        Heart heart = new HeartAdapter(heartModel);
         heart.beat(); // Simulating heart beat
        heartModel.stopBeat(); // Stopping the heart beat
    }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No 11

**Q.11) Design simple HR Application using Spring Framework**

```
package com.example.hr;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.*;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Service;
import javax.persistence.*;
import java.util.List;

@SpringBootApplication
public class HrApplication {
    public static void main(String[] args) {
        SpringApplication.run(HrApplication.class, args);
    }
}

// Employee Model
@Entity
class Employee {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String department;
    // Getters and Setters
}

// Repository
interface EmployeeRepository extends JpaRepository<Employee, Long> {}

// Service
@Service
class EmployeeService {
    private final EmployeeRepository repository;
    EmployeeService(EmployeeRepository repository) { this.repository =
repository; }
    List<Employee> getAll() { return repository.findAll(); }
    Employee getById(Long id) { return repository.findById(id).orElse(null); }
    Employee save(Employee employee) { return repository.save(employee); }
    void delete(Long id) { repository.deleteById(id); }
}

// Controller
@RestController
@RequestMapping("/api/employees")
class EmployeeController {
    private final EmployeeService service;
    EmployeeController(EmployeeService service) { this.service = service; }
    @GetMapping
    List<Employee> getAll() { return service.getAll(); }
    @GetMapping("/{id}")
    Employee getById(@PathVariable Long id) { return service.getById(id); }
    @PostMapping
    Employee create(@RequestBody Employee employee) { return
service.save(employee); }
    @DeleteMapping("/{id}")
    void delete(@PathVariable Long id) { service.delete(id); }
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Internet of Things(CS-MJ-633T)

## Assignment Completion Sheet

Name of the Student :

Roll No :

| Sr.No | Assignment Title | Marks Obtained | Signature of Instructor |
|---|---|---|---|
| 1 | To write a program to sense the available networks using Arduino | | |
| 2 | To write a program to measure the distance using ultrasonic sensor and make LED blinkusing Arduino. | | |
| 3 | To write a program to detects the vibration of an object with sensor using Arduino. | | |
| 4 | To write a program to sense a finger when it is placed on the bourd Arduino | | |
| 5 | To write a program to connect with the available Wi-Fi asing Arduino. | | |
| 6 | To write a program to get temperature notification using Arduino. | | |
| 7 | To write a program for LDR to vary the light intensity of LED using Arduino. | | |
| 8 | Start Raspberry Pi and try various Linux commands in command terminal window: Is, od | | |

|  |  |  |  |
|---|---|---|---|
|  | toach, mv, m man mkdir, mdir tarzio, cat, more, lesa, ps, sudo, cron, chown, chury ningetc |  |  |

**Total Marks :**

**Converted into 10 Marks :**

**Date:**

**Signature of In Charge** **Head**

**Internal Examiner** **External Examiner**

# Assignment No.1

## Q.1) To write a program to sense the available networks using Arduino

```
#include <SPI.h>
#include <WiFi.h>
void setup() {
// initialize serial and wait for the port to open:
Serial.begin(9600);
while (!Serial)
;
// attempt to connect using WEP encryption:
Serial.println("Initializing Wifi...");
printMacAddress();
// scan for existing networks:
Serial.println("Scanning available networks...");
listNetworks();
}
void loop() {
delay(10000);
// scan for existing networks:
Serial.println("Scanning available networks...");
listNetworks();
}
void printMacAddress() {
// the MAC address of your Wifi shield
byte mac[6];
// print your MAC address:
WiFi.macAddress(mac);
Serial.print("MAC: ");
Serial.print(mac[5], HEX);
Serial.print(":");
Serial.print(mac[4], HEX);
Serial.print(":");
Serial.print(mac[3], HEX);
Serial.print(":");
Serial.print(mac[2], HEX);
Serial.print(":");
Serial.print(mac[1], HEX);
Serial.print(":");
Serial.println(mac[0], HEX);
}
void listNetworks() {
// scan for nearby networks:
Serial.println("** Scan Networks **");
byte numSsid = WiFi.scanNetworks();
// print the list of networks seen:
Serial.print("number of available networks:");
Serial.println(numSsid);
// print the network number and name for each network found:
for (int thisNet = 0; thisNet < numSsid; thisNet++) {
Serial.print(thisNet);
Serial.print(") ");
Serial.print(WiFi.SSID(thisNet));
Serial.print("\tSignal: ");
Serial.print(WiFi.RSSI(thisNet));
Serial.print(" dBm");
Serial.print("\tEncryption: ");
Serial.println(WiFi.encryptionType(thisNet));
}
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No. 2

**Q.2) To write a program to measure the distance using ultrasonic sensor and make LED blinkusing Arduino.**

```
const int TRIG_PIN = 6; // Arduino pin connected to Ultrasonic
Sensor's TRIG pin
const int ECHO_PIN = 7; // Arduino pin connected to Ultrasonic
Sensor's ECHO pin
const int LED_PIN = 3; // Arduino pin connected to LED's pin
const int DISTANCE_THRESHOLD = 50; // centimeters
float duration_us, distance_cm;
void setup() {
Serial.begin(9600); // initialize serial port
pinMode(TRIG_PIN, OUTPUT); // set arduino pin to output mode
pinMode(ECHO_PIN, INPUT); // set arduino pin to input mode
pinMode(LED_PIN, OUTPUT); // set arduino pin to output mode
}
void loop() {
digitalWrite(TRIG_PIN, HIGH);
delayMicroseconds(10);
digitalWrite(TRIG_PIN, LOW);
duration_us = pulseIn(ECHO_PIN, HIGH);
distance_cm = 0.017 * duration_us;
if (distance_cm < DISTANCE_THRESHOLD)
digitalWrite(LED_PIN, HIGH); // turn on LED
else
digitalWrite(LED_PIN, LOW); // turn off LED
Serial.print("distance: ");
Serial.print(distance_cm);
Serial.println(" cm");
delay(500);

}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No 3

**Q.3) To write a program to detects the vibration of an object with sensor using Arduino.**

```
int vib_pin=7;
int led_pin=13;
void setup() {
pinMode(vib_pin,INPUT);
pinMode(led_pin,OUTPUT);
}
void loop() {
int val;
val=digitalRead(vib_pin);
if(val==1)
{
digitalWrite(led_pin,HIGH);
delay(1000);
digitalWrite(led_pin,LOW);
delay(1000);
}
else
digitalWrite(led_pin,LOW);
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No .4

**Q.4) To  write a program to sense a finger when it is placed on the bourd Arduino**

```
#include <CapacitiveSensor.h>
const int touchPin = 2;
const int touchThreshold = 50;
CapacitiveSensor touchSensor = CapacitiveSensor(0, touchPin);
void setup() {
Serial.begin(9600);
}
void loop() {
long touchValue = touchSensor.capacitiveSensor(30);
if (touchValue > touchThreshold) {
Serial.println("Finger detected!");
delay(1000); // wait for 1 second to prevent multiple detections
}
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No.5

## Q.5) To write a program to connect with the available Wi-Fi asing Arduino

```
   #include <SPI.h>
#include <WiFiNINA.h>
void setup() {
char ssid[] = ”Wifi name”;
char pass[] = ”secret password”;
Serial.begin(9600);
while (!Serial);
int status = WL_IDLE_STATUS;
while (status != WL_CONNECTED) {
Serial.print("Connecting to ");
Serial.println(ssid);
status = WiFi.begin(ssid, pass);
delay(5000);
}
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
}
Void(){}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

# Assignment No.6

## Q.6) To write a program to get temperature notification using Arduino.

```
#define ADC_VREF_mV 5000.0 // in millivolt
#define ADC_RESOLUTION 1024.0
#define PIN_LM35 A0
void setup() { Serial.begin(9600); }
void loop() {
// get the ADC value from the temperature sensor
int adcVal = analogRead(PIN_LM35);
// convert the ADC value to voltage in millivolt
float milliVolt = adcVal * (ADC_VREF_mV / ADC_RESOLUTION);
// convert the voltage to the temperature in Celsius
float tempC = milliVolt / 10;
// convert the Celsius to Fahrenheit
float tempF = tempC * 9 / 5 + 32;
// print the temperature in the Serial Monitor:
Serial.print("Temperature: ");
Serial.print(tempC); // print the temperature in Celsius
Serial.print("°C");
Serial.print(" ~ "); // separator between Celsius and Fahrenheit
Serial.print(tempF); // print the temperature in Fahrenheit
Serial.println("°F");
delay(1000);
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**

## Assignment No.7

**Q.7) To write a program for LDR to vary the light intensity of LED using Arduino.**

```
int sensor=A0;
int output=9;
void setup()
{
pinMode(output, OUTPUT);
}
void loop()
{
int reading=analogRead(sensor);
int bright=reading/4;
delay(500);
analogWrite(output, bright);
}
```

**Assignment Evaluation**

| | | | | | |
|---|---|---|---|---|---|
| 0: Not Done | | 1: Incomplete | | 2: Late Complete | |
| 3: Need Improvement | | 4: Completed | | 5: Well Done | |

**Date:** **Practical In-charge**