1 a. Design an animation program to show the running clock using ClockAnimation extends StillClock. Use a Timer object that serves as the source of an ActionEvent, the listener must be an instance of ActionListener and registered with a Timer object. The Timer object uses its own constructor with a delay and a listener, where delay specifies the number of milliseconds between two action events. Display a clock using StillClock to show the current time, the clock does not tick after it is displayed but it should display a new current time for every second.

Introduction:

Here, we create a ClockAnimation class that extends StillClock, which is a class that displays a clock with the current time. We then create a Timer object that fires an ActionEvent every second and registers the ClockAnimation instance as its listener. When the Timer fires an ActionEvent, we call the setCurrentTime() method of StillClock to update the current time and repaint the clock.

In the paintComponent() method, we call the superclass's paintComponent() method to draw the clock face and hands, then we draw the current time in the center of the clock face.

Finally, in the main() method, we create a JFrame, add a ClockAnimation panel to it, and display the frame.

Code:

Main class:

```
import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.Timer;

import java.awt.Color;

import java.awt.Font;

import java.awt.Graphics;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


public class Main extends StillClock implements ActionListener {
```

```
    private Timer timer;
    public Main() {
        timer = new Timer(1000, this);
        timer.start();
    }
    public void actionPerformed(ActionEvent e) {
        setCurrentTime();
        repaint();
    }
    public static void main(String[] args) {
        JFrame frame = new JFrame("Clock Animation");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(300, 300);
        Main clock = new Main();
        frame.add(clock);
        frame.setVisible(true);
    }
}
```

Still clock class:

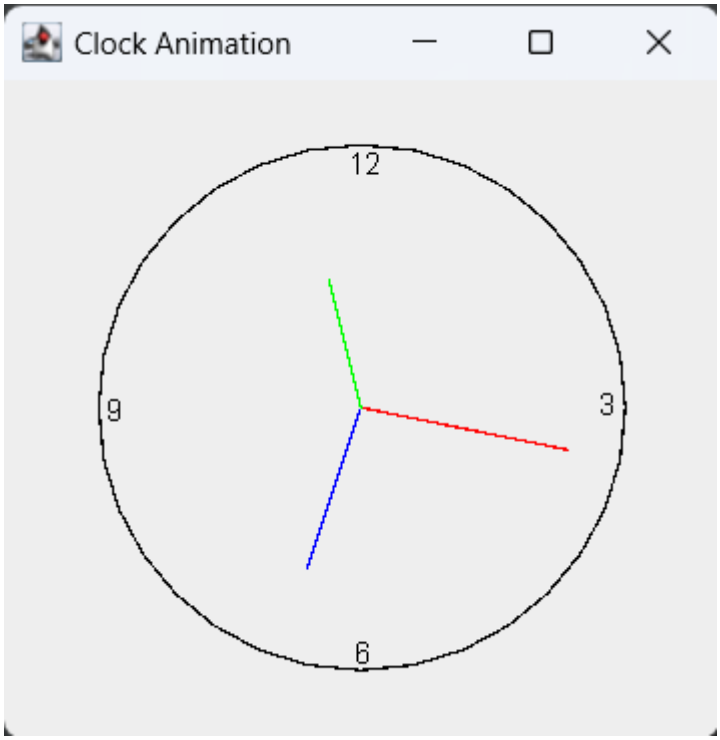
```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.util.Calendar;
import javax.swing.JComponent;
```

```
public class StillClock extends JComponent {


  protected int hour;

  protected int minute;

  protected int second;


  public StillClock() {

    setCurrentTime();

  }


  public void setCurrentTime() {

    // Get the current time from the system clock

    Calendar now = Calendar.getInstance();

    hour = now.get(Calendar.HOUR_OF_DAY);

    minute = now.get(Calendar.MINUTE);

    second = now.get(Calendar.SECOND);

  }


  @Override

  public Dimension getPreferredSize() {

    return new Dimension(200, 200);

  }


  @Override

  protected void paintComponent(Graphics g) {
```

```
super.paintComponent(g);


// Get clock dimensions
int clockRadius = (int)(Math.min(getWidth(), getHeight()) * 0.8 * 0.5);
int xCenter = getWidth()/2;
int yCenter = getHeight()/2;


// Draw clock face
g.setColor(Color.BLACK);
g.drawOval(xCenter - clockRadius, yCenter - clockRadius, 2 * clockRadius, 2 * clockRadius);
g.drawString("12", xCenter - 5, yCenter - clockRadius + 12);
g.drawString("9", xCenter - clockRadius + 3, yCenter + 5);
g.drawString("3", xCenter + clockRadius - 10, yCenter + 3);
g.drawString("6", xCenter - 3, yCenter + clockRadius - 3);


// Draw clock hands
int sLength = (int)(clockRadius * 0.8);
int xm = (int)(xCenter + sLength * Math.sin(second * (2 * Math.PI / 60)));
int ym = (int)(yCenter - sLength * Math.cos(second * (2 * Math.PI / 60)));
g.setColor(Color.RED);
g.drawLine(xCenter, yCenter, xm, ym);


int mLength = (int)(clockRadius * 0.65);
int xh = (int)(xCenter + mLength * Math.sin(minute * (2 * Math.PI / 60)));
int yh = (int)(yCenter - mLength * Math.cos(minute * (2 * Math.PI / 60)));
```

```
        g.setColor(Color.BLUE);
        g.drawLine(xCenter, yCenter, xh, yh);

        int hLength = (int)(clockRadius * 0.5);
        int xh2 = (int)(xCenter + hLength * Math.sin((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
        int yh2 = (int)(yCenter - hLength * Math.cos((hour % 12 + minute / 60.0) * (2 * Math.PI / 12)));
        g.setColor(Color.GREEN);
        g.drawLine(xCenter, yCenter, xh2, yh2);
    }
}
```

Output:



1 b. Develop any personalized application using NodeJS

Introduction:

This is a Node.js application that uses the Express framework and Mongoose library to connect to a MongoDB database. The application listens on port 4000 and has one route that accepts a POST request to insert data into the "Naruto" collection in the database. The route expects a JSON payload in the request body, which is then inserted into the collection using the insertOne() method. If the insertion is successful, a response with status code 202 and "success" message is sent back to the client. The application also uses the CORS middleware to allow cross-origin requests. The database connection is established using the MongoDB Atlas service and the connection string is passed as an argument to the connect() method of the Mongoose library.

This React component creates a form with four input fields (Name, Place, PC Link, and Date) and a submit button. It uses the useState hook to manage the state of the form data and a message to display the status of the data insertion. When the user submits the form, it sends a POST request to the Node.js server with the form data using Axios library. The server then inserts the data into a MongoDB database. If the insertion is successful, it sends back a "success" message, and the component displays an alert and updates the message state with "Data inserted successfully". If the insertion fails, the component displays an alert and updates the message state with "Data not inserted".

Code:

app.css:

```
body {
  font-family: Arial, sans-serif;
  margin-top: 250px;
  margin-bottom:250px;
  padding: 0;
  background-color: #555;
  background-image: url(https://htmlcolorcodes.com/assets/images/colors/neon-green-color-solid-background-1920x1080.png);
  background-size: auto;
}
```

```
/* Header styles */
header {
  background-color: #333;
  color: white;
  display: flex;
  align-items: center;
  height: 60px;
  padding: 0 20px;
}
header h1 {
  font-size: 24px;
  margin: 0;
}
/* Form styles */
form {
  display: flex;
  flex-direction: column;
  max-width: 500px;
  margin: 20px auto;
  padding: 20px;
  background-color: #f7f7f7;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
label {
  font-size: 16px;
```

```
  margin-bottom: 5px;
}
input {
  font-size: 16px;
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  margin-bottom: 20px;
}
button {
  font-size: 16px;
  padding: 10px 20px;
  border: none;
  background-color: rgb(0, 191, 255);
  color: white;
  border-radius: 4px;
  cursor: pointer;
}
button:hover {
  background-color: rgb(5, 235, 248);
}
button:disabled {
  background-color: #ccc;
  cursor: not-allowed;
}
```

```
/* Success message styles */

.success {

  color: green;

  margin-top: 20px;

}

/* Error message styles */

.error {

  color: red;

  margin-top: 20px;

}
```

server.js:

```
const express = require("express");

const db = require("mongoose");

var cors = require("cors");

var ObjectId = require("mongodb").ObjectId;

const app = express();``

app.use(express.json());

db.set("strictQuery", false);

app.use(cors());

//************mongodb connect */

db.connect("mongodb+srv://Vijayendra:vijayendra@cluster0.bfcm1qq.mongodb.net/test", {

  useNewURLParser: true,

  useUnifiedTopology: true,

})

  .then(() => console.log("MongoDb is connected"))
```

```
  .catch((err) => console.log(err));
var conn = db.connection;
//**************insert data to database */
app.post("/ins", function (req, res) {
  const fdata = req.body.fdata;
  conn.collection("Naruto").insertOne(fdata, (err, result) => {
    if (err) {
      console.log(err);
    } else {
      console.log("inserted");
      res.status(202).send("success");
    }
  });
});
app.listen(4000, () => {
  console.log("server running at 4000");
});
```

app.js:

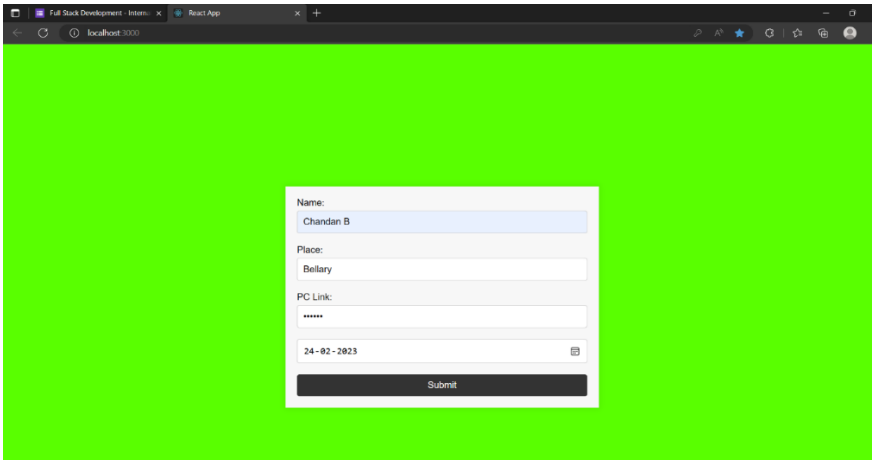
```
import React, { useState } from "react";
import Axios from "axios";
import "./App.css";
function App() {
  const [fdata, setFdata] = useState({
    Name: "",
```

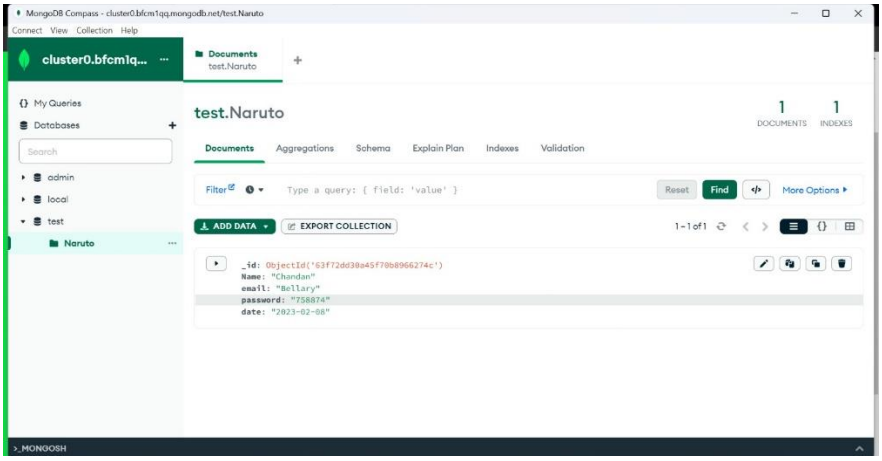
```
  email: "",
  password: "",
  date:""
});
const [msg, setMsg] = useState();
const changeHandler = (e) => {
  let name1 = e.target.name;
  let val = e.target.value;
  setFdata({ ...fdata, [name1]: val });
};
const submitHandler = (e) => {
  e.preventDefault();
  Axios.post("http://localhost:4000/ins", { fdata }).then((res) => {
    let ack = res.data;
    if (ack === "success") {
      setMsg("Data inserted successfully");
      console.log(msg);
      alert("Data inserted successfully");
    } else {
      setMsg("Data not inserted");
      console.log(msg);
      alert("Data not inserted");
    }
  });
};
```

```
return (
 <div className="form">
  <form onSubmit={submitHandler}>
   <label>Name:</label>
   <input
    type="text"
    name="Name"
    value={fdata.Name}
    onChange={changeHandler}
   />
   <label>Place:</label>
   <input
    type="text"
    name="email"
    value={fdata.email}
    onChange={changeHandler}
   />
   <label>PC Link:</label>
   <input
    type="password"
    name="password"
    value={fdata.password}
    onChange={changeHandler}
   />
   <input
```

```
        type="date"
        name="date"
        value={fdata.date}
        onChange={changeHandler}
      />
      <button type="submit">Submit</button>
    </form>
  </div>
 );
}
export default App;
```

Output:





2 a. Create your own application to add, delete and update the students' academic details using Java API.

Introduction:

This Java program uses an ArrayList to implement a student database application that allows users to add, delete, update and display student records.

The Student class represents a student with properties such as name, id, department, and gpa. The StudentDatabase class manages the collection of students using an ArrayList and provides methods to add, delete, and update students.

The main method uses a while loop and a switch statement to repeatedly prompt the user for a choice of action, such as adding a student, deleting a student, updating a student, displaying all students, or exiting the program. The program takes user input from the console and uses the appropriate StudentDatabase method to perform the chosen action on the student records.

Code:

```
import java.util.ArrayList;

import java.util.Scanner;

class Student {

   String name;

   int id;

   String department;

   double gpa;

   Student(String name, int id, String department, double gpa) {

      this.name = name;

      this.id = id;

      this.department = department;

      this.gpa = gpa;

   }
```

```
    @Override
    public String toString() {
        return "Name: " + name + "\nID: " + id + "\nDepartment: " + department + "\nGPA: " + gpa + "\n";
    }
}
class StudentDatabase {
    ArrayList<Student> students;

    StudentDatabase() {
        students = new ArrayList<Student>();
    }

    void addStudent(Student s) {
        students.add(s);
    }
    void deleteStudent(int id) {
        for (int i = 0; i < students.size(); i++) {
            if (students.get(i).id == id) {
                students.remove(i);
                break;
            }
        }
    }
    void updateStudent(int id, String department, double gpa) {
        for (int i = 0; i < students.size(); i++) {
```

```
            if (students.get(i).id == id) {
                students.get(i).department = department;
                students.get(i).gpa = gpa;
                break;
            }
        }
    }
    void displayAllStudents() {
        System.out.println("List of all students:");
        for (int i = 0; i < students.size(); i++) {
            System.out.println(students.get(i).toString());
        }
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StudentDatabase db = new StudentDatabase();
        while (true) {
            System.out.println("Select an option:");
            System.out.println("1. Add student");
            System.out.println("2. Delete student");
            System.out.println("3. Update student");
//          System.out.println("4. Get student");
            System.out.println("4. Display all students");
```

```
System.out.println("5. Exit");
int choice = sc.nextInt();
switch (choice) {
   case 1:
      System.out.print("Enter name: ");
      String name = sc.next();
      System.out.print("Enter id: ");
      int id = sc.nextInt();
      System.out.print("Enter department: ");
      String department = sc.next();
      System.out.print("Enter GPA: ");
      double gpa = sc.nextDouble();
      db.addStudent(new Student(name, id, department, gpa));
      break;
   case 2:
      System.out.print("Enter id of student to delete: ");
      int deleteId = sc.nextInt();
      db.deleteStudent(deleteId);
      break;
   case 3:
      System.out.print("Enter id of student to update: ");
      int updateId = sc.nextInt();
      System.out.print("Enter new department: ");
      String newDepartment = sc.next();
      System.out.print("Enter new GPA: ");
```

```
            double newGpa = sc.nextDouble();

            db.updateStudent(updateId, newDepartment, newGpa);

            break;

          case 4:

            db.displayAllStudents();

            break;

            case 5:System.exit(0);

        }

     }

   }

}
```

Output:

```
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
1
Enter name: hemanth
Enter id: 1
Enter department: ise
Enter CGPA: 10.0
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
1
Enter name: vijayendra
Enter id: 2
Enter department: cse
Enter CGPA: 9.9
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
```

```
4
List of all students:
Name: hemanth
ID: 1
Department: ise
GPA: 10.0

Name: vijayendra
ID: 2
Department: cse
GPA: 9.9

Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
1
Enter name: keerthi
Enter id: 3
Enter department: ec
Enter CGPA: 9.9
```

```
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
1
Enter name: chandan
Enter id: 4
Enter department: aiml
Enter CGPA: 9.8
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
4
List of all students:
Name: hemanth
ID: 1
Department: ise
GPA: 10.0
```

```
Name: vijayendra
ID: 2
Department: cse
GPA: 9.9

Name: keerthi
ID: 3
Department: ec
GPA: 9.9

Name: chandan
ID: 4
Department: aiml
GPA: 9.8

Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
2
Enter id of student to delete: 3
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
```

```
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
4
List of all students:
Name: hemanth
ID: 1
Department: ise
GPA: 10.0

Name: vijayendra
ID: 2
Department: cse
GPA: 9.9

Name: chandan
ID: 4
Department: aiml
GPA: 9.8
```

```
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
3
Enter id of student to update: 2
Enter new department: aiml
Enter new GPA: 10.0
Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
4
List of all students:
Name: hemanth
ID: 1
Department: ise
GPA: 10.0
```

```
Name: vijayendra
ID: 2
Department: aiml
GPA: 10.0

Name: chandan
ID: 4
Department: aiml
GPA: 9.8

Select an option:
1. Add student
2. Delete student
3. Update student
4. Display all students
5. Exit
5

Process finished with exit code 0
```

2 b. Create your own application to manage Food orders In Food Court using Java API.

Introduction:

In this program, we create a HashMap called menu to store the menu items and their prices. We then display the menu to the user using a for-each loop, and prompt the user to enter an item to order.

As the user enters each item, we check if it exists in the menu using the containsKey() method. If it does, we add its price to the totalPrice variable and display a message confirming the item was added. If it doesn't, we display an error message.

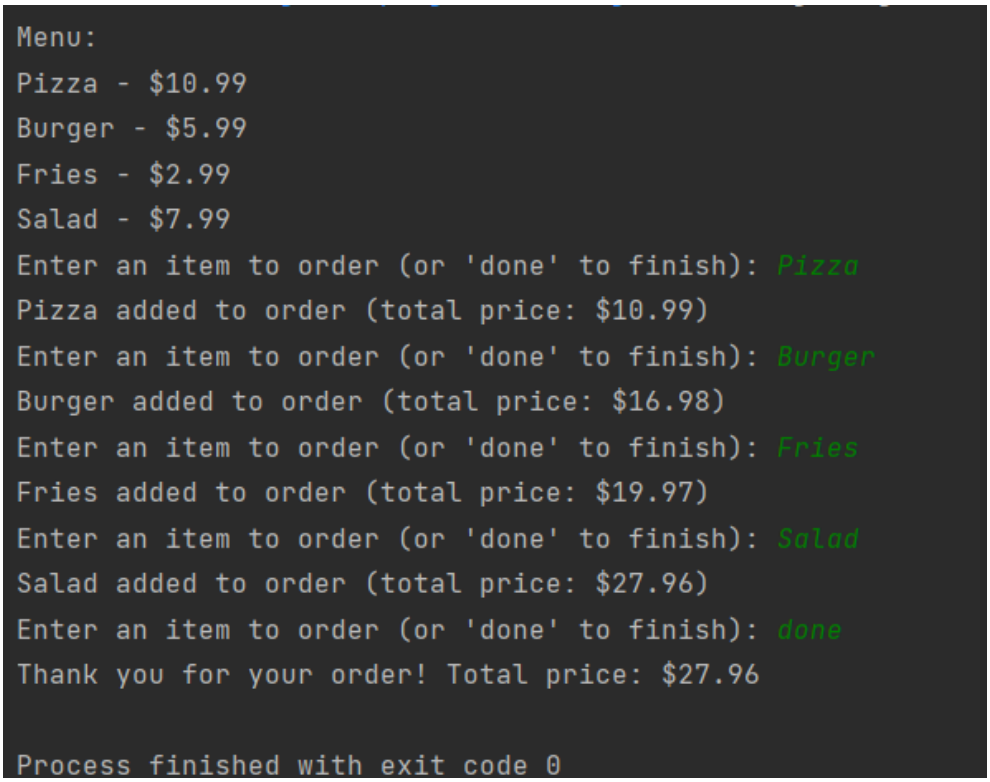
The user can continue placing orders until they enter "done". Once they're finished, we display the total price of the order using the total price variable.

Code:

```
import java.util.HashMap;
import java.util.Scanner;


public class Main {


  public static void main(String[] args) {


    // Creating a HashMap to store menu items and their prices
    HashMap<String, Double> menu = new HashMap<>();
    menu.put("Pizza", 10.99);
    menu.put("Burger", 5.99);
    menu.put("Fries", 2.99);
    menu.put("Salad", 7.99);
    Scanner input = new Scanner(System.in);
    String order = "";
    double totalPrice = 0.0;
    // Displaying the menu
    System.out.println("Menu:");
    for (String item : menu.keySet()) {
      System.out.println(item + " - $" + menu.get(item));
```

```
        }
        // Prompting the user to place an order
        while (!order.equalsIgnoreCase("done")) {
            System.out.print("Enter an item to order (or 'done' to finish): ");
            order = input.nextLine();
            if (menu.containsKey(order)) {
                totalPrice += menu.get(order);
                System.out.println(order + " added to order (total price: $" + totalPrice + ")");
            } else if (!order.equalsIgnoreCase("done")) {
                System.out.println("Sorry, we don't have that item on the menu.");
            }
        }
        // Displaying the total price of the order
        System.out.println("Thank you for your order! Total price: $" + totalPrice);
    }
}
```

Output:



Impact:

- The animation program designed to show the running clock using the ClockAnimation class extends the StillClock class, which displays a clock showing the current time. The program uses a Timer object to generate ActionEvents at regular intervals, which are then handled by an ActionListener registered with the Timer object. The ActionListener updates the clock with the current time every second, giving the appearance of a running clock.
- Developing personalized applications using NodeJS provides developers with a highly scalable and efficient framework for building robust, fast, and highly interactive web applications. NodeJS provides developers with a vast range of libraries and tools that they can use to build complex applications quickly and easily.
- The application to manage student academic details using Java API provides an easy-to-use interface for adding, deleting, and updating student records. This application can be used by schools, colleges, and other educational institutions to manage student data efficiently.
- The application to manage food orders in a food court using Java API allows customers to place food orders quickly and efficiently. The application also allows restaurant owners to manage their menus, track orders, and monitor their business's performance in real-time.
- Overall, the impact of these programs is significant as they provide efficient and convenient solutions to common problems, saving time and effort for users while also enhancing their overall experience.