Refresher Assignment

Name : Vajid Kagdi

Student Id: 012528971

JavaScript Variables

Introdution:

- JS variables are containers to store values and have unique names.
- JS variable identifiers are case sensitive.
- Creating a variable in JS is called "declaring" a variable. ‘var’ keyword is used to declare variables.
    - Syntax : var Variable_name = Variable_value;
- 3 basic data types are : text, number and boolean.
- Two types of data : variables and constant. Vaariables data can change but constants data is fixed.
- Constants are created just like initialized variables, but you use the const keyword instead of var.
    - Syntax : const Constant_name = Constant_value;
- Data types are established when variable’s and constant’s values are set.

Scenario:

Newly open sandwiches shop called Sandwich Work'z Sandwiches near San Jose State University want to have a portal where customers can place order. The portal should take the name and number of sandwiches from the customers and should calculate the amount and amount including tax and display it to the customer.

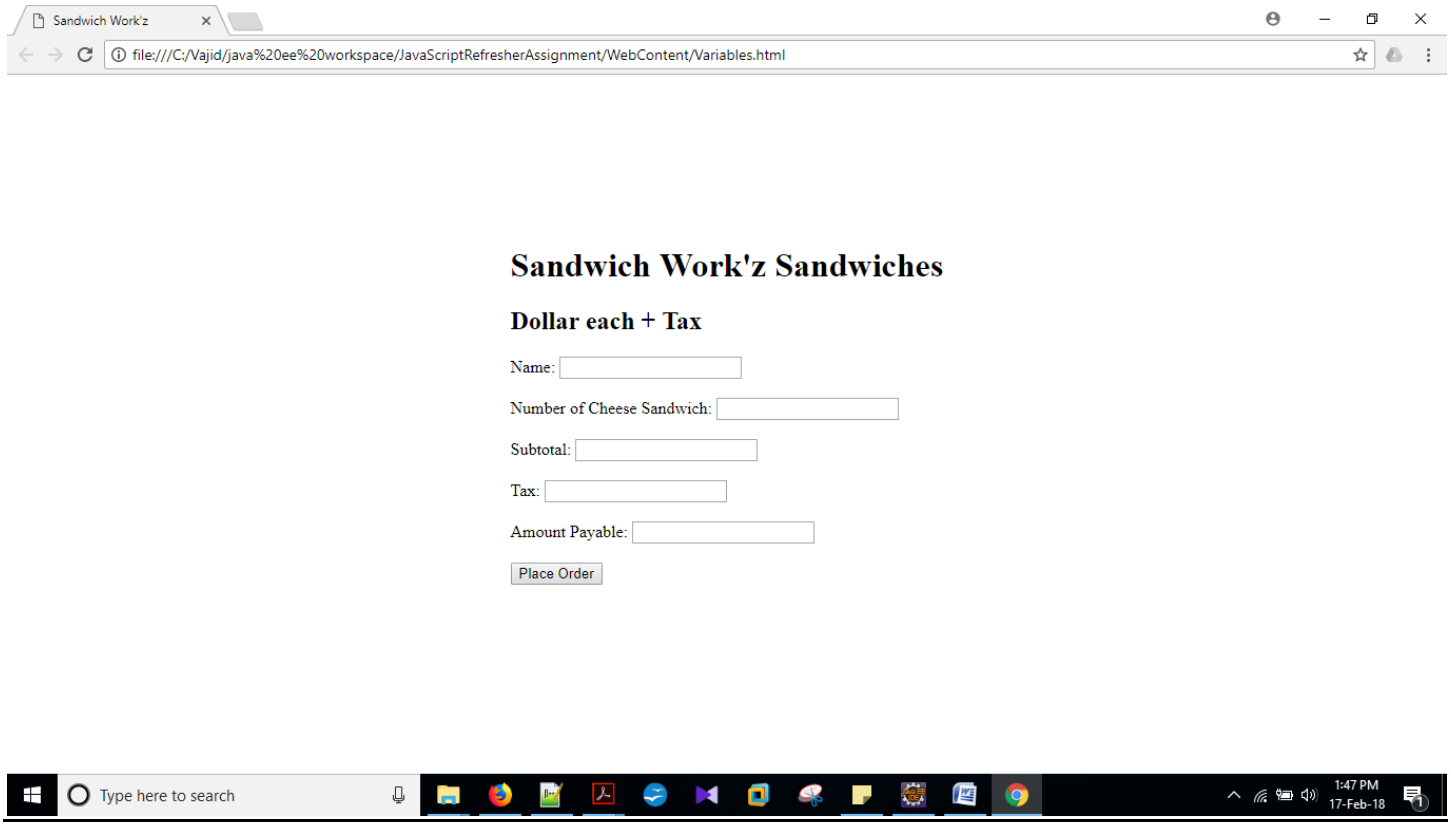
Your task is to create this portal for above requirement using java script variables and constants.

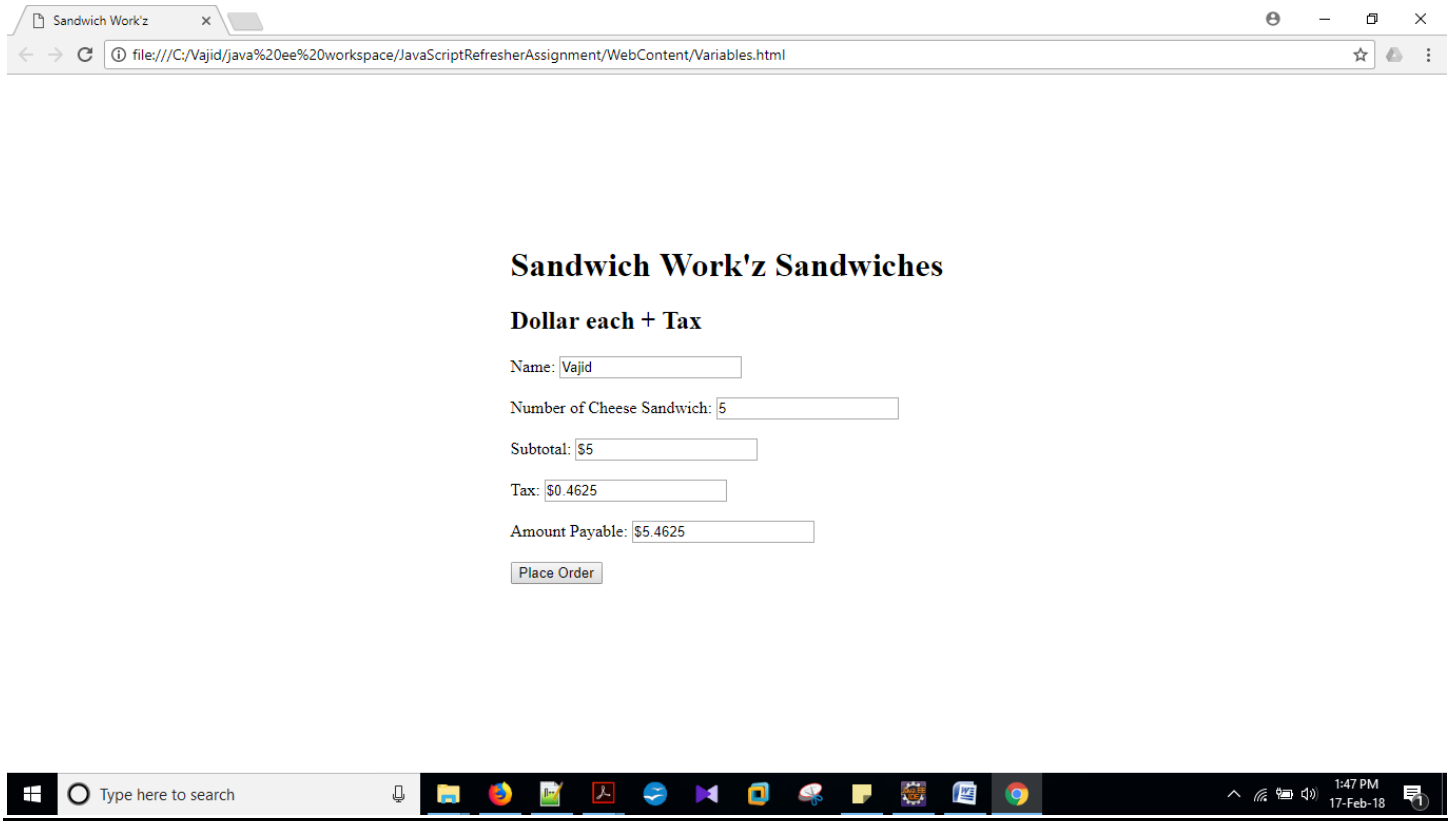
HTML Source Code:

```
<html>
<head>
<title>Sandwich Work'z</title>
<script type="text/javascript">
      function updateOrder() {

            //Using Constant for Tax Rate and  Sandwich Price
            const
            TAX_RATE = 0.0925;
            const
            SANDWICHPRICE = 1.0;

            //Using Variable for storing value.
            var numSandwich = document.getElementById("sandwich").value;
            var subTotal = numSandwich * SANDWICHPRICE;
            var tax = subTotal * TAX_RATE;

            var finalAmount = subTotal + tax;
            document.getElementById("subtotal").value = "$" + subTotal;
            document.getElementById("tax").value = "$" + tax;
            document.getElementById("total").value = "$" + finalAmount;
      }
</script>
<style type="text/css">
html, body {
      height: 100%;
}
```

```
html {
      display: table;
      margin: auto;
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>
</head>

<body>
      <h1>Sandwich Work'z Sandwiches</h1>
      <h2>Dollar each + Tax</h2>
      <form>
            <div>
                  Name: <input type="text" id="name" value="" />
            </div>
            <br>
            <div>
                  Number of Cheese Sandwich: <input type="text" id="sandwich"
                        onchange="updateOrder();" />
            </div>
            <br>
            <div>
                  Subtotal: <input type="text" id="subtotal" readonly="readonly" />
            </div>
            <br>
            <div>
                  Tax: <input type="text" id="tax" readonly="readonly" />
            </div>
            <br>
            <div>
                  Amount Payable: <input type="text" id="total" readonly="readonly" />
            </div>
            <div>
                  <br> <input type="button" value="Place Order" />
            </div>
      </form>
</body>
</html>
```

Source Code Run:





Objects

Introdution:

- Objects in javascript are variables too. But they can contain many values in the form of multiple name value pairs.
- For Eg.: A car is an object and it has properties like weight and color
    - Syntax : var car = {type:"Fiat", model:"500", color:"white"}
- These name:values pairs are called properties of class.
- To access the object properties there are two ways:
    - objectName.propertyName
    - objectName[“propertyName”]
- Functions can also be defined within objects. They are called methods. You access an object method with the following syntax:
    - objectName.methodName()

Scenario:

Professor Shim wants to create a webpage where the schedule of each lecture in the current Semester for subject CMPE273 is posted and students can come pre-prepare with the topic. The schedule should have Date, Topic and the short introduction of the topic on the page.

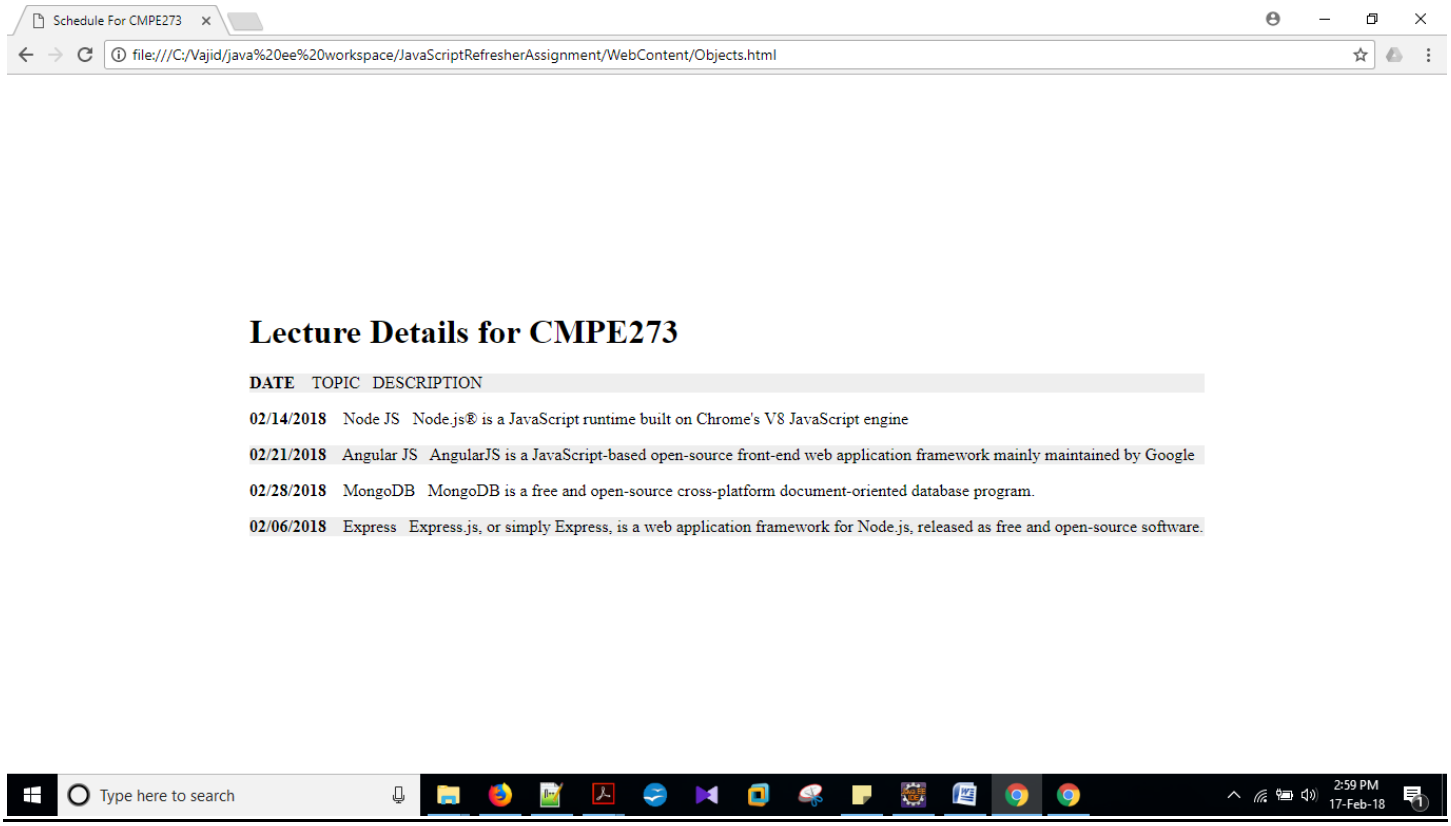
Your task is to create the webpage with above requirements using Javascript objects.

HTML Source Code:

```
<html>
<head>
<title>Schedule For CMPE273</title>
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>
<script type="text/javascript">
	// Lecture object constructor
	function Lecture(topic, date, description) {
		this.topic = topic;
		this.date = date;
		this.description = description;
	}
	// Array of Lecture entries and adding lecture objects to show together on screen.
	var lecture = [
			new Lecture("TOPIC", "DATE", "DESCRIPTION"),
```

```
                  new Lecture("Node JS", "02/14/2018",
                              "Node.js® is a JavaScript runtime built on Chrome's V8
JavaScript engine"),
                  new Lecture(
                              "Angular JS",
                              "02/21/2018",
                              "AngularJS is a JavaScript-based open-source front-end web
application framework mainly maintained by Google"),
                  new Lecture(
                              "MongoDB",
                              "02/28/2018",
                              "MongoDB is a free and open-source cross-platform document-
oriented database program."),
                  new Lecture(
                              "Express",
                              "02/06/2018",
                              "Express.js, or simply Express, is a web application
framework for Node.js, released as free and open-source software.") ];

      // Show Lecture entries
      function showLecture() {

            // Show the lecture entries
            var i = 0, lectureText = "";
            while (i < lecture.length) {

                  if (i % 2 == 0)
                        lectureText += "<p style='background-color:#EEEEEE'>";
                  else
                        lectureText += "<p>";

                  // Generate the formatted lecture HTML code
                  lectureText += "<strong>" + lecture[i].date
                              + "</strong>    " + lecture[i].topic
                              + "   " + lecture[i].description + "</p>";
                  i++;
            }
            // Set the lecture HTML code on the page
            document.getElementById("lecture").innerHTML = lectureText;
      }
</script>
</head>
<body onload="showLecture()">
      <h1>Lecture Details for CMPE273</h1>
      <div id="lecture"></div>
</body>
</html>
```

Source Code Run:



Functions

Introdution:

- Function is a subprogram that can be called by code external to the functions. This eliminates the need of writing the same code again and again.
- A function is composed of a sequence of statements.
- Values can be passed to a function, and the function will return a value. The return statement causes a function to stop executing at that point.
- The parameters of a function call are the function's *arguments*.
- There are several ways to define functions:
    - Syntax: function function_name(params) { //Statements of Code }
- To invoke a function you would simply need to write the name of that function.
- It is possible to create functions inside other functions. Simply declare another function inside the code of an existing function.
- We can also create a function expression by assigning a function to a variable.

Scenario:

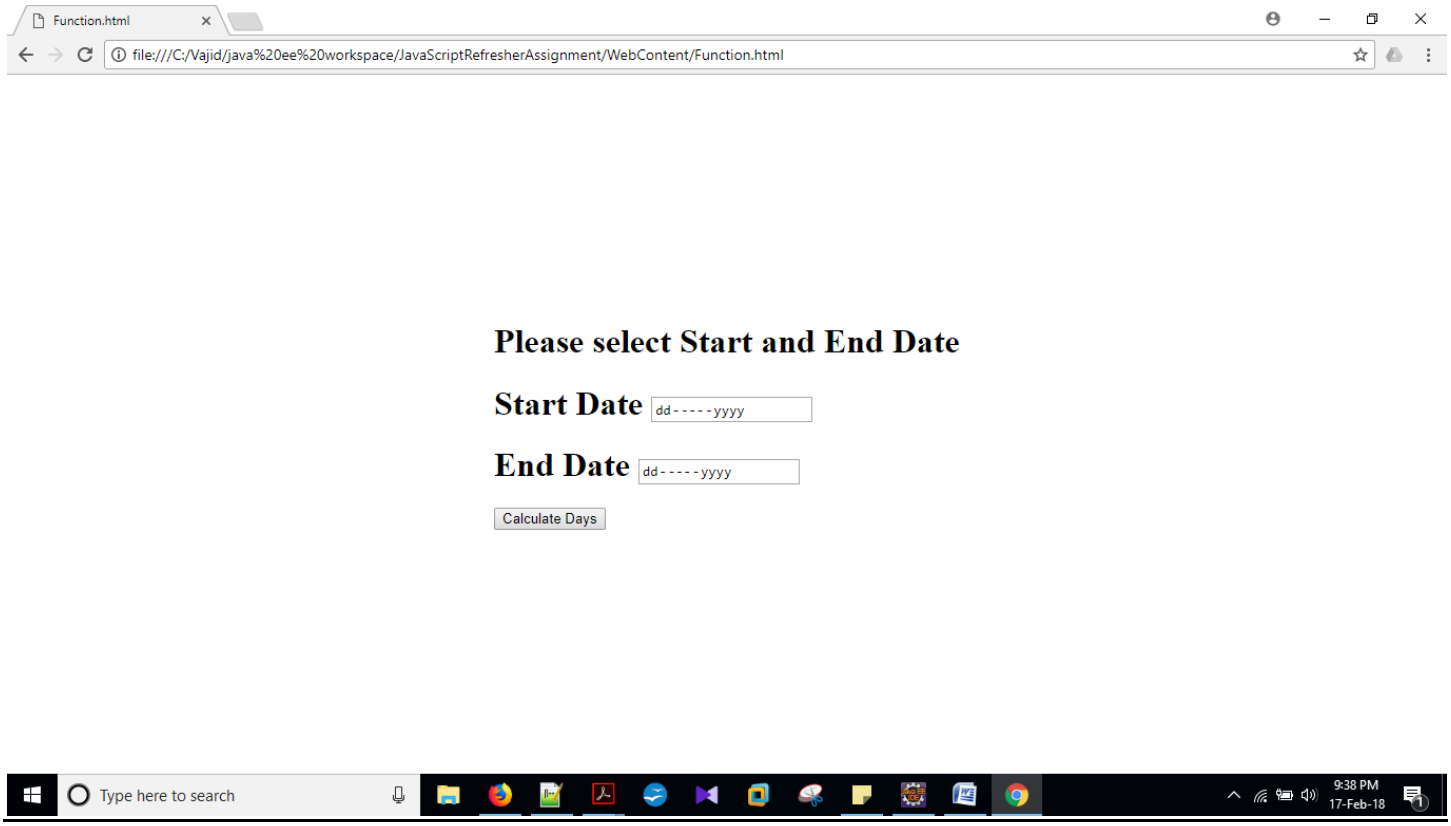
The calculation between two dates is always a tedious task. In order to be always accurate create a portal which will provide the difference between two dates. The portal should take two dates and on click of a button the function should be called which will calculate the days between two dates.

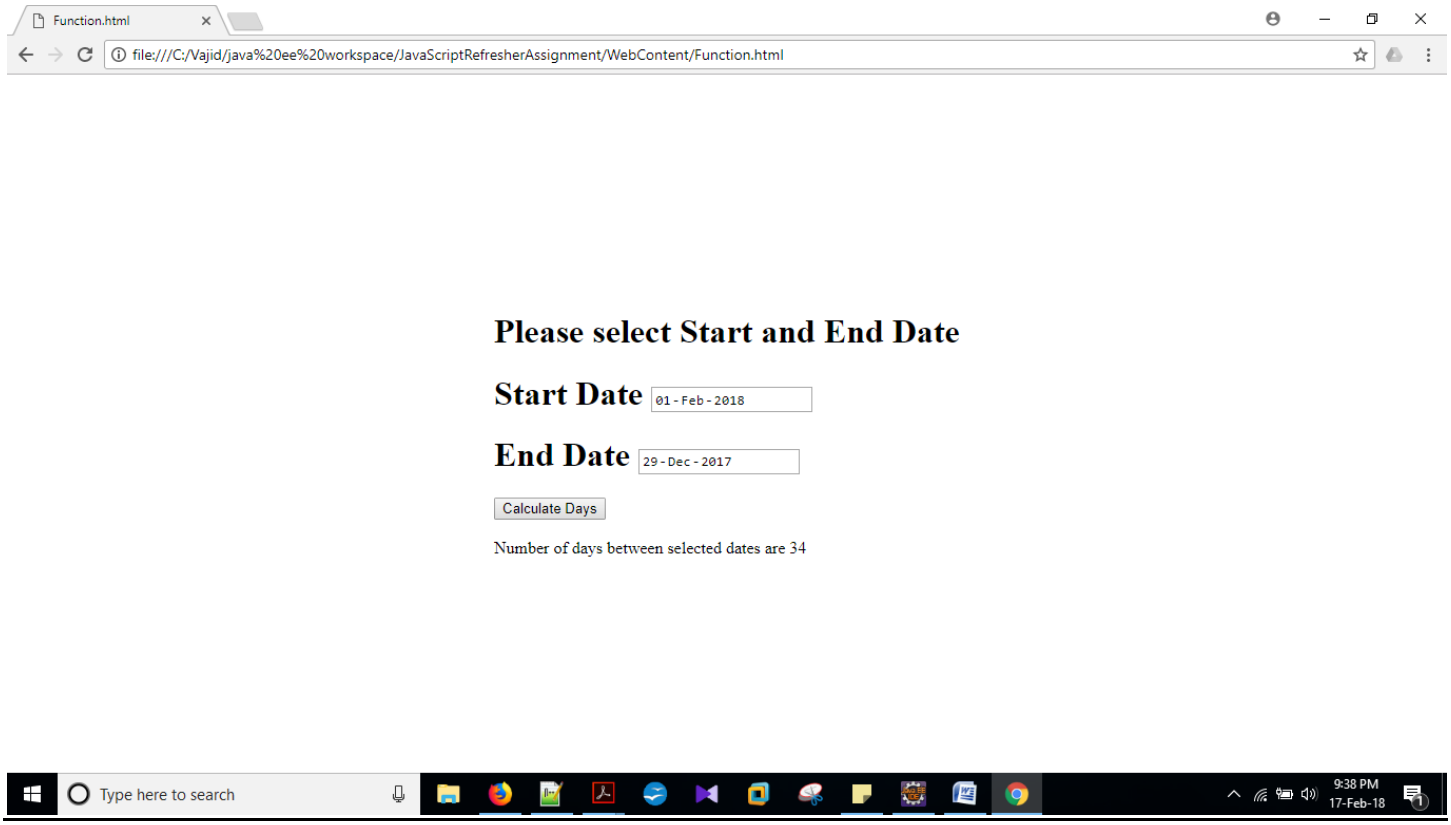
HTML Source Code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>
<script type="text/javascript">
	function calculate() {
		var start = new Date(document.getElementById("start").value);
		var end = new Date(document.getElementById("end").value);

		var ONEDAY = 1000 * 60 * 60 * 24;

		var date1_ms = start.getTime();
		var date2_ms = end.getTime();
```

```
            var difference_ms = Math.abs(date1_ms - date2_ms);

            document.getElementById("daysCount").innerHTML = " Number of days between
selected dates are "
                        + Math.round(difference_ms / ONEDAY);
      }
</script>
</head>
<body>

      <h1>Please select Start and End Date</h1>
      <form>

            <div>
                  <label><h1>
                              Start Date <input required="" type="date"
id="start"></label>
            </div>

            <div>
                  <label><h1>
                              End Date <input required="" type="date" id="end"></label>
            </div>

            <div>
                  <input type="button" value="Calculate Days" onclick="calculate();">
            </div>
            <br>
            <div id="daysCount"></div>
      </form>
</body>
</html>
```

Source Code Run:





Events

Introdution:

- JavaScript's interacts with HTML events that occur.
- Examples of HTML events:
  - With click of mouse.
  - When a web page has loaded.
  - When an image has been loaded.
  - When the mouse moves over an element.
  - When an input field is changed.
- Events are used with functions, and the function will not be executed before the event occurs.
- These functions are called event handlers.
- Main events used are:
  - Mouse Events : onClick, onmouseover, onmouseout
  - Keyboard Events : onkeyup, onkeydown, onkeypress
  - Form Events : onchange, onblur, onfocus.

Scenario:

A property dealer wants to create a portal where the buyers can calculate the price affordable to them by providing the number of bedrooms requirement and their annual salary. The portal should take two inputs : salary and the number of bedrooms.

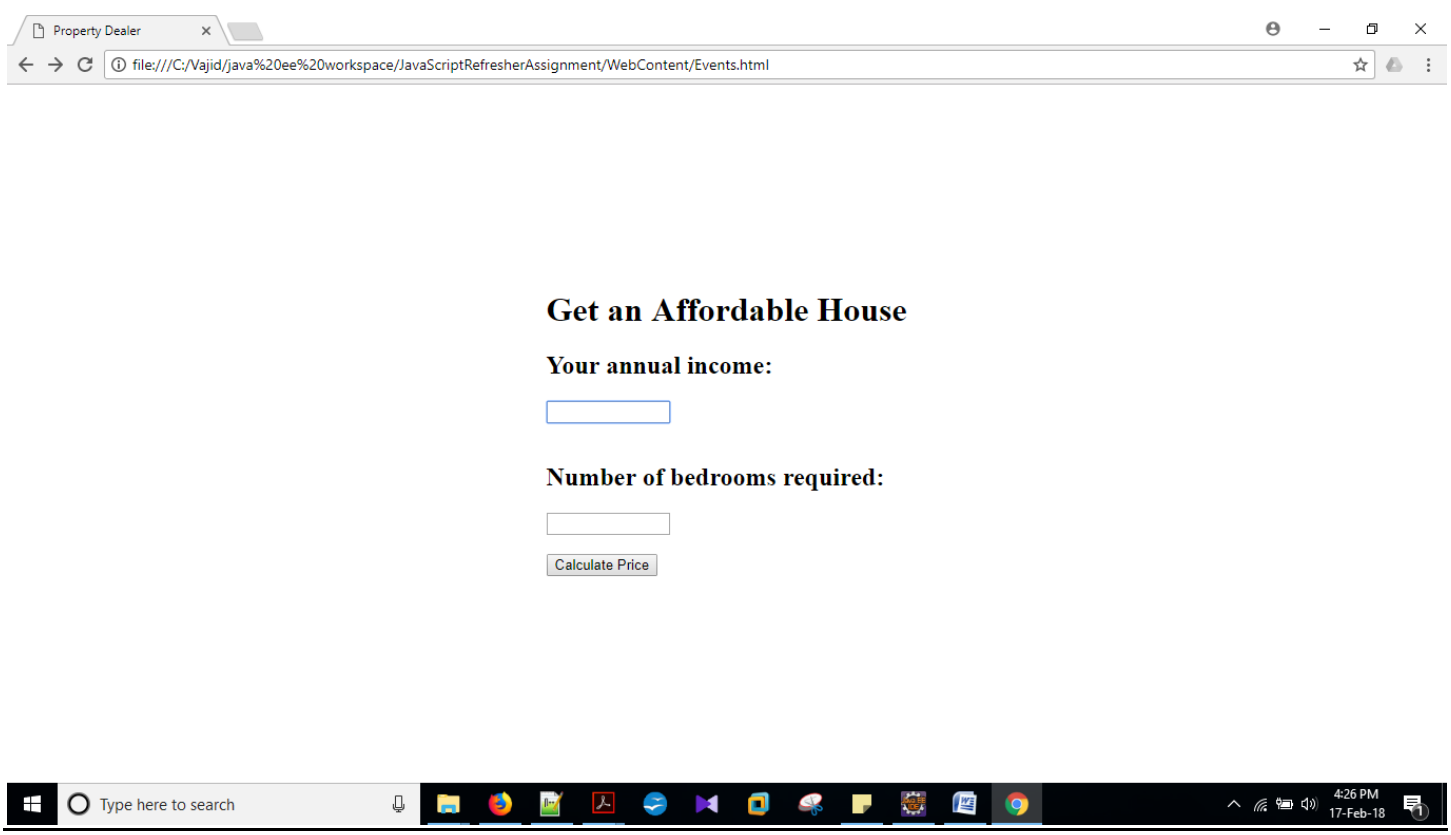
Your task is to create the portal to satisfy the requirement of property dealer using Javascript events.

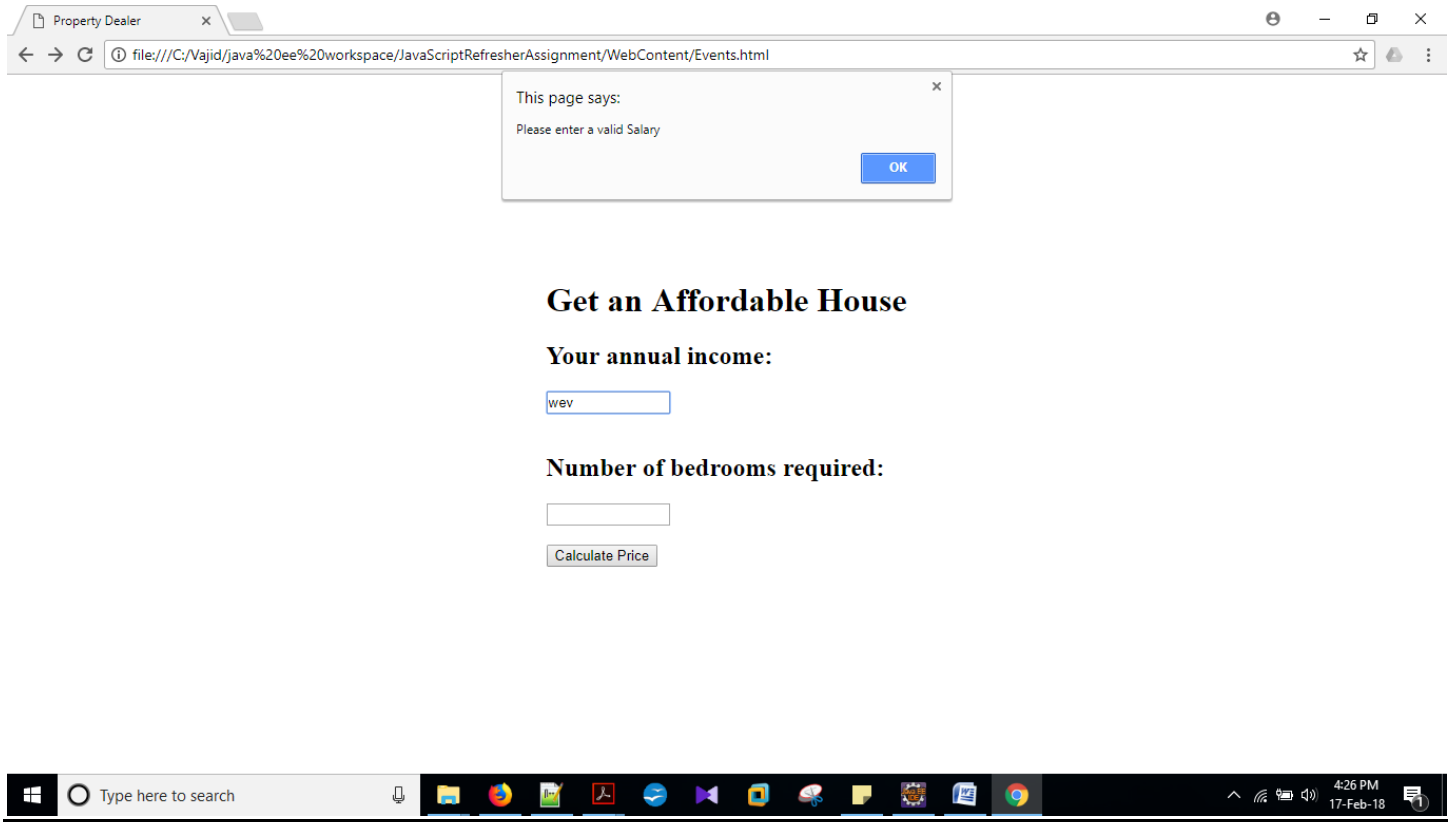
HTML Source Code:

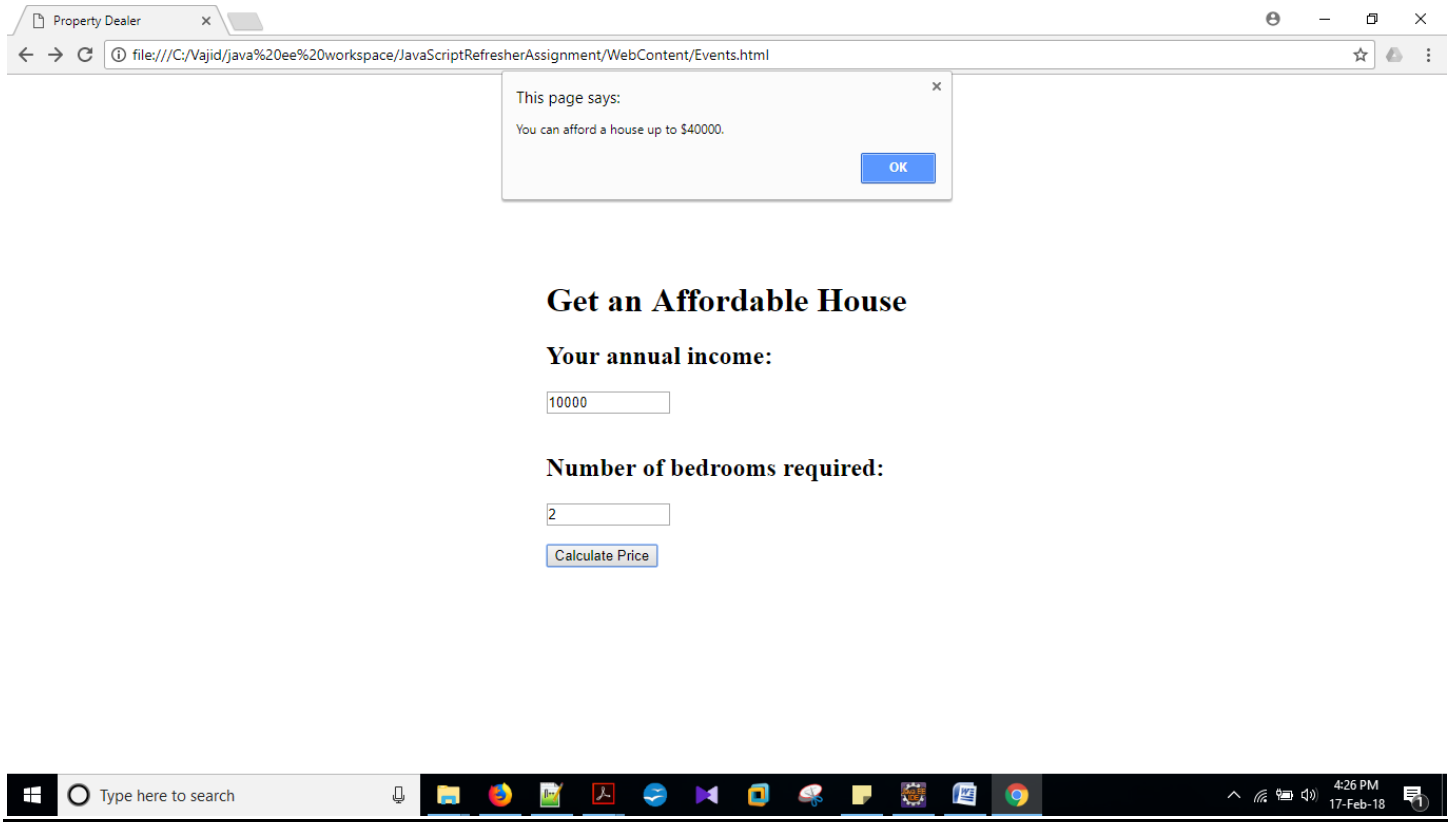
```
<html>
<head>
<title>Property Dealer</title>
<style type="text/css">
html, body {
      height: 100%;
}

html {
      display: table;
      margin: auto;
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>
<script type="text/javascript">
      function validateNumber(value) {
            // Validate the number
            if (isNaN(value))
                  alert("Please enter a valid Salary");
      }
```

```
        function calcPrice() {
                var maxPrice = document.getElementById("income").value * 4;
                alert("You can afford a house up to $" + maxPrice + ".");
        }
</script>
</head>

<body>
        <div>
                <h1>Get an Affordable House</h1>
                <div>
                        <h2>Your annual income:</h2>
                        <input id="income" type="text" size="12" required=""
                                onblur="validateNumber(this.value);" />
                </div>
                <br>
                <div>

                        <h2>Number of bedrooms required:</h2>
                        <input id="bedrooms" type="text" required="" size="12"
                                onchange="validateNumber(this.value);" />
                </div>
                <br> <input type="button" value="Calculate Price"
                        onclick="calcPrice();" />
                </form>
        </div>
</body>
</html>
```

Source Code Run:




Property Dealer
file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/Events.html
This page says:
Please enter a valid Salary
OK
Get an Affordable House
Your annual income:
wev
Number of bedrooms required:
Calculate Price
Type here to search
4:26 PM
17-Feb-18


Property Dealer
file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/Events.html
This page says:
You can afford a house up to $40000.
OK
Get an Affordable House
Your annual income:
10000
Number of bedrooms required:
2
Calculate Price
Type here to search
4:26 PM
17-Feb-18

Array

Introdution:

- Arrays are used to store multiple values in a single variable having same name for elements.
- Values can be accessed by referring to an index number. Array indexes start with 0.
    - Syntax : var array_name = [item1, item2, ...];
- The length property of array returns the number of elements stored in array.
- The sort() method sorts arrays
- New element can be added to an array is using the push method.
- Array.isArray(arrayVariable) : this returns true if the arrayVariable is of array type.

Scenario:

Professor Shim wants to assign enrollment number to all the students in the class and will take the assignment submission in that order. The portal should take the names of all the students and provide the numbers in the alphabetical order of names as entered.

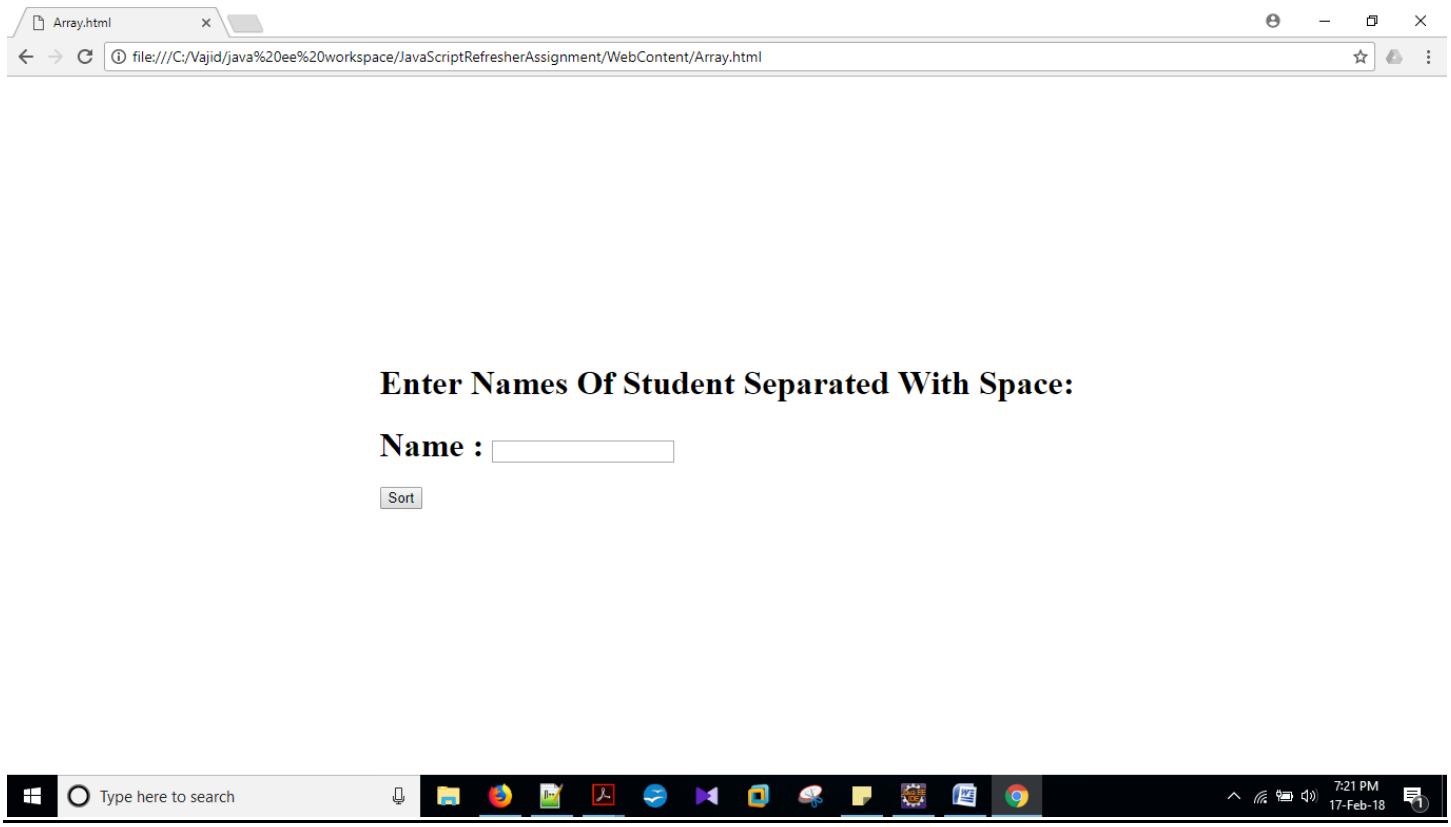
Your task is to create the portal which using Javascript Arrays.

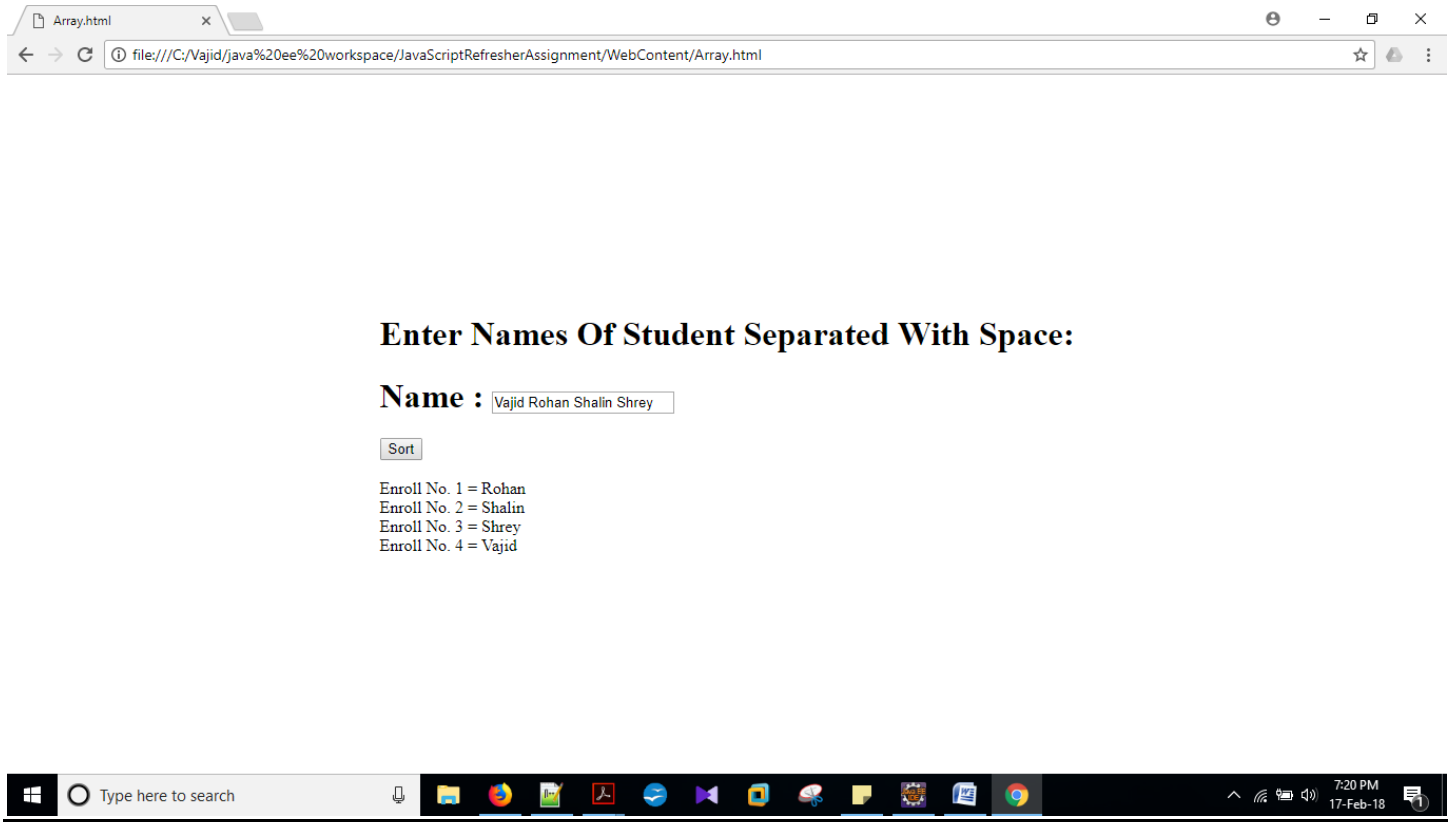
HTML Source Code:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>

<script type="text/javascript">
	function sortNames() {

		var names = document.getElementById("name").value;
		var nameArray = names.split(" ");

		nameArray.sort();
		var display = document.getElementById("sortedNames");
		var output = "";
		for (i = 0; i < nameArray.length; i++) {
			output = output + "<br/> Enroll No. " + (i + 1) + " = "
				+ nameArray[i];
		}
		display.innerHTML = output;
```

```
        }
</script>
</head>
<body>
        <h1>Enter Names Of Student Separated With Space:</h1>

        <form>
                <div>
                        <h1>
                                Name : <input type="text" id="name" />
                </div>
                <input type="button" onclick="sortNames();" value="Sort" />
                <div id="sortedNames"></div>

        </form>
</body>
</html>
```

Source Code Run:




Array.html
file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/Array.html
Enter Names Of Student Separated With Space:
Name :
Vajid Rohan Shalin Shrey
Sort
Enroll No. 1 = Rohan
Enroll No. 2 = Shalin
Enroll No. 3 = Shrey
Enroll No. 4 = Vajid
Type here to search
7:20 PM
17-Feb-18

Inheritance

Introdution:

- Javascript uses the concept of prototypes and prototype chaining for inheritance.
- When a function is created in JavaScript, JavaScript engine adds a prototype property to the function. This prototype property is an object
- We can access the function's prototype property using the syntax functionName.prototype.
- Prototype object of the constructor function is shared among all the objects created using the constructor function.
- New property can be added to the constructor function's prototype property.
- If an object A modifies property of the prototype having primitive value, other objects will not be effected.

Scenario:

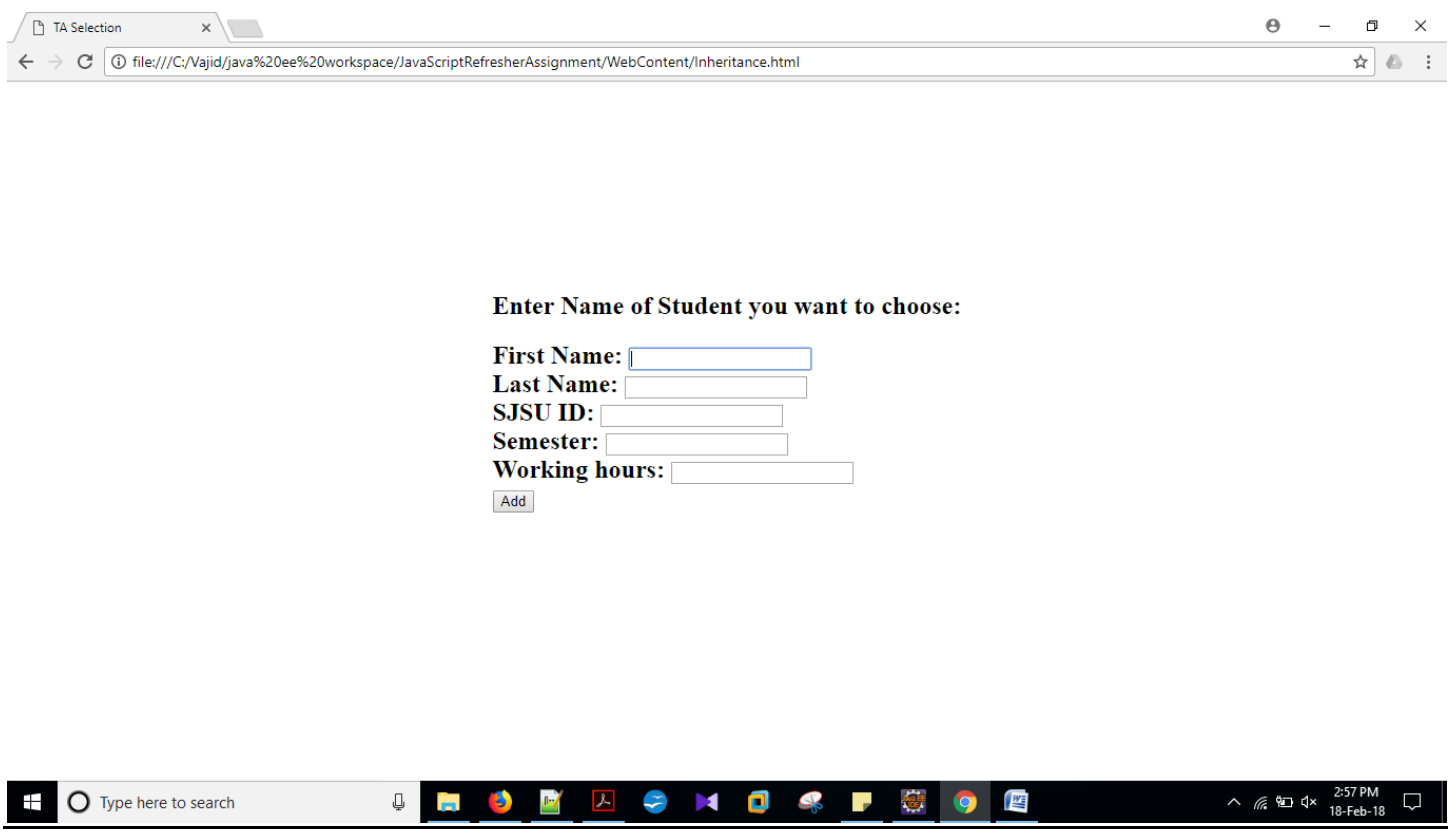
Professor Shim has hired 5 TA's this semester and want a system so that he can add all the TAs to that system along with the hors they will be working in a Week. Your task is to create the portal which will take information such as First name, Last name, Student id, Semester and hours using JavaScript Inheritance and prototype use case.

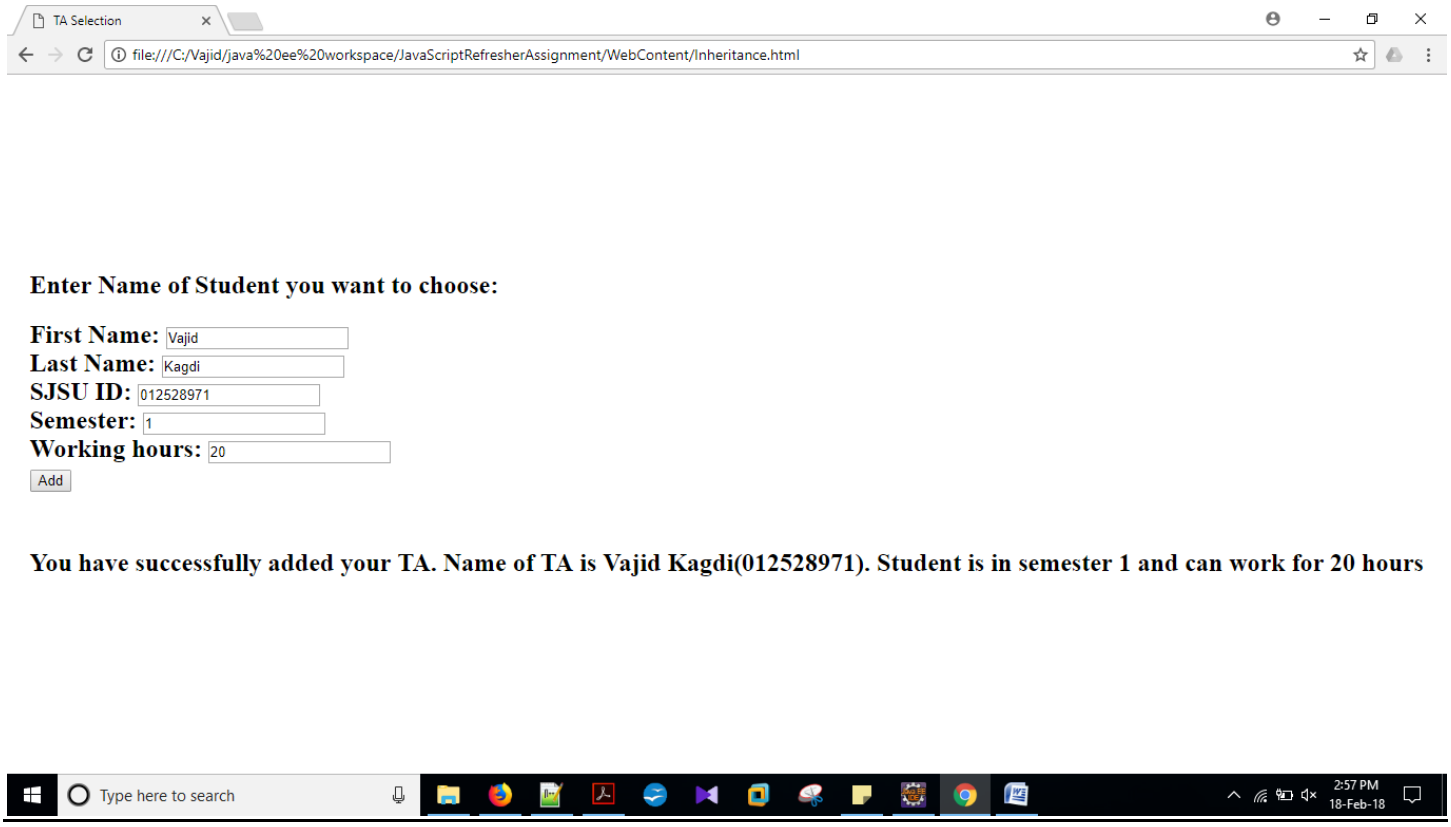
HTML Source Code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>TA Selection</title>
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>
<script>
	function User(firstname, lastname, sjsuid, sem, number_of_hours) {
		var final_result = "You have successfully added your TA. Name of TA is "
				+ firstname
				+ " "
				+ lastname
				+ "("
				+ sjsuid
				+ ")."
				+ " Student is in semester  "
				+ sem
				+ " and can work for "
```

```
                        + number_of_hours + " hours";
            document.getElementById("result").innerHTML = final_result;
      }
      function setDetails() {
            var first_name = document.getElementById("first_name").value;
            var last_name = document.getElementById("last_name").value;
            var sjsu_id = document.getElementById("sjsu_id").value;
            var sem = document.getElementById("semester").value;
            var hours = document.getElementById("hours").value;
            var ta_details = new User(first_name, last_name, sjsu_id, sem, hours);
      }
</script>
</head>
<body>
      <h2>
            Enter Name of Student you want to choose:
            <h2>

                  First Name: <input type="text" id="first_name" /> <br /> Last Name:
<input
                        type="text" id="last_name"> <br /> SJSU ID: <input
                        type="number" id="sjsu_id"> <br /> Semester: <input
                        type="number" id="semester"> <br /> Working hours: <input
                        type="number" id="hours"> <br />
                  <button id="button" onClick="setDetails()">Add</button>
                  <br /> <br /> <br />
                  <div id="result"></div>
</body>
</html>
```

Source Code Run:




TA Selection
file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/Inheritance.html
Enter Name of Student you want to choose:
First Name: Vajid
Last Name: Kagdi
SJSU ID: 012528971
Semester: 1
Working hours: 20
Add
You have successfully added your TA. Name of TA is Vajid Kagdi(012528971). Student is in semester 1 and can work for 20 hours
Type here to search
2:57 PM
18-Feb-18

Conditions

Introdution:

- Javascript provides many decision making conditions like if, else, elseif and switch these statements are used in making decisions.
- if..else statements is used where you want to execute a set of code when a condition is true and another if the condition is not true.
- elseif statements is used where you want to execute a some code if one of several conditions are true.
- switch statements is used where you want to execute one of the block of code from many.

Scenario:

Pizza hub is new store opening near San Jose State university. This store wants to come up with the portal where customers can place an order online. The portal should ask for name of the customer, no. of pizzas and the toppings required. It should calculate the final price using the toppings selected, number of pizza and the base price of the pizza.

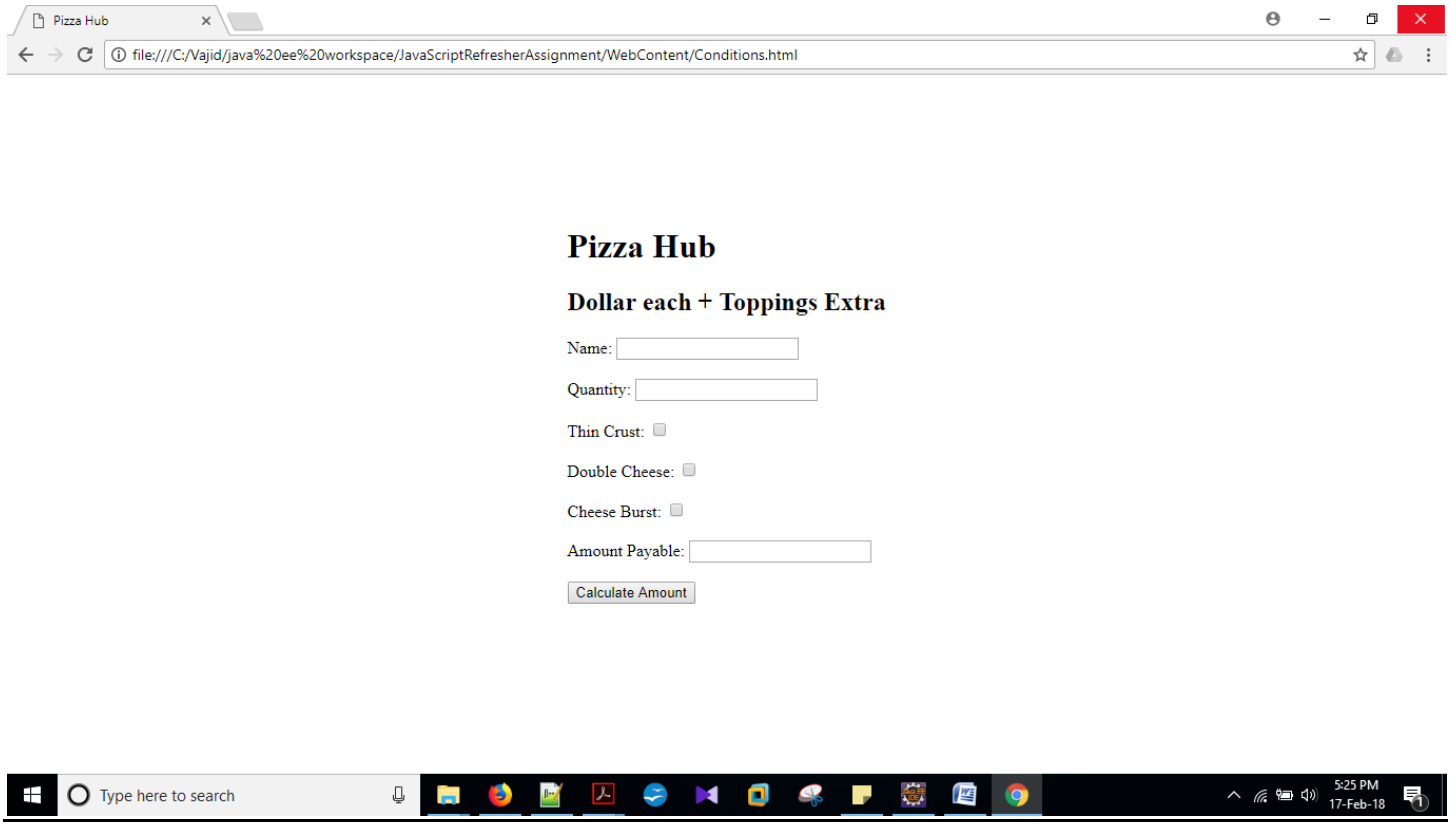
Your task is to create the portal and use conditions in Javascript to calculate the amount payable as per toppings selected.

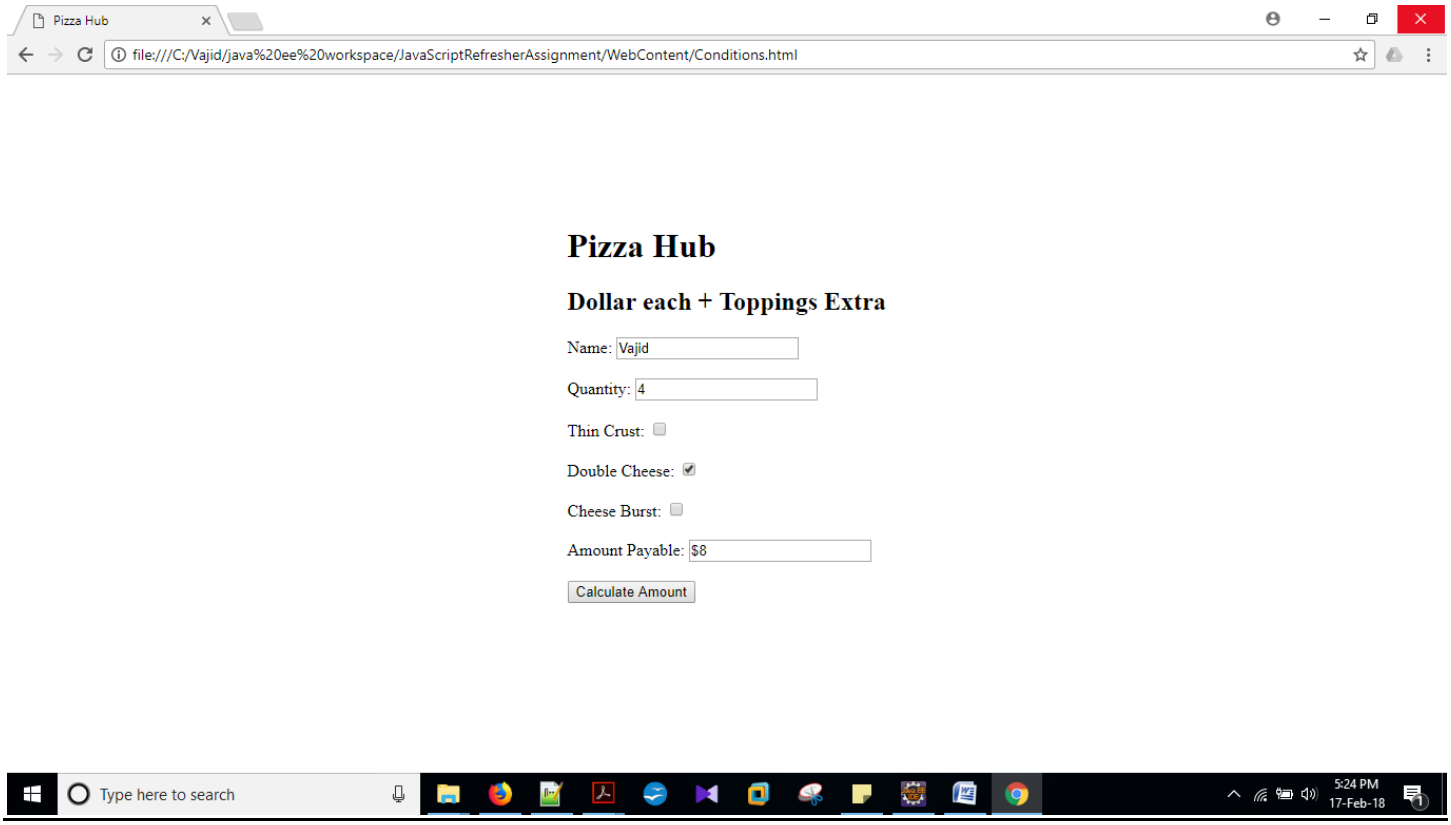
HTML Source Code:

```
<html>
<head>
<title>Pizza Hub</title>


<script type="text/javascript">
      function calculateOrder() {

            var simplePizzaPrice = 1.0;

            //Using Variable for storing value.
            var numPizza = document.getElementById("pizza").value;

            var thinCrust = document.getElementById("thinCrust");
            var cheeseBurst = document.getElementById("cheeseBurst");
            var doubleCheese = document.getElementById("doubleCheese");

            if (thinCrust.checked == true) {
                  simplePizzaPrice = simplePizzaPrice + 1;
            }
            if (cheeseBurst.checked == true) {
                  simplePizzaPrice = simplePizzaPrice + 1;
            }
            if (doubleCheese.checked == true) {
                  simplePizzaPrice = simplePizzaPrice + 1;
            }
```

```
            var subTotal = numPizza * simplePizzaPrice;
            document.getElementById("total").value = "$" + subTotal;
      }
</script>
<style type="text/css">
html, body {
      height: 100%;
}

html {
      display: table;
      margin: auto;
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>
</head>

<body>
      <h1>Pizza Hub</h1>
      <h2>Dollar each + Toppings Extra</h2>
      <form>
            <div>
                  Name: <input type="text" id="name" value="" />
            </div>
            <br>
            <div>
                  Quantity: <input type="text" id="pizza" />
            </div>
            <br>
            <div>
                  Thin Crust: <input type="checkbox" id="thinCrust">
            </div>
            <br>

            <div>
                  Double Cheese: <input type="checkbox" id="doubleCheese" />
            </div>
            <br>

            <div>
                  Cheese Burst: <input type="checkbox" id="cheeseBurst" />
            </div>
            <br>
            <div>
                  Amount Payable: <input type="text" id="total" readonly="readonly" />
            </div>
            <div>
                  <br> <input type="button" value="Calculate Amount"
                        onClick="calculateOrder()" />
            </div>
      </form>
</body>
</html>
```

Source Code Run:





Regular Expressions

Introdution:

- Regular expressions are patterns used to match character combinations in strings.
- In Javascript they are objects.
- In Javascript regular expression can be constructed in two ways:
    - var re = /ab+c/ : pattern enclosed between slashes.
    - var re = new RegExp('ab+c') : calling the constructor function of the RegExp
- Simple patterns are constructed of characters to find a direct match.
- When the search for a match requires something more than a direct match the pattern includes special characters like *, \, ^ etc.
- Regular expressions also have five optional flags that allow for global and case insensitive searching.

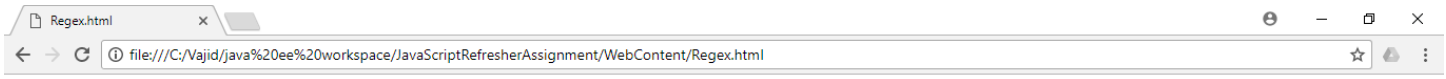
Scenario:

San Jose State University wants to take the emergency contact information of all the students. For this they require a portal where students can go and update the contact. It should take the Name and Number of the contact.

Your task is to create the portal and validate the mobile number using javascript regular expression.

HTML Source Code:

Source Code Run:



Enter Emergency Contact Info:

Name : Vajid

Phone : 669-265-5287

Update

Regex.html

file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/Regex.html

This page says:

664d isn't a correct format!

OK

Enter Emergency Contact Info:

Name : Vajid

Phone : 664d

Update

Type here to search

6:51 PM
17-Feb-18

Strict Mode

Introdution:

- ‘use strict’ Defines that JavaScript code should be executed in "strict mode".
- The purpose of "use strict" is that the code should be executed in "strict mode".
- Strict mode is declared to the beginning of a script or a function.
- Strict mode makes it easier to write "secure" JavaScript.
- Strict mode changes previously accepted "bad syntax" into real errors.
- In strict mode:
    - Using a variable, without declaring it, is not allowed
    - Deleting a variable/function is not allowed.
    - Duplicating a parameter name is not allowed.

Scenario:

Professor Shim wants to take a class on Javascript lecture on Strict Mode. He want the Student Assistant to create a program with two functions. One of the function should run in strict mode and other in normal mode in order to show error.

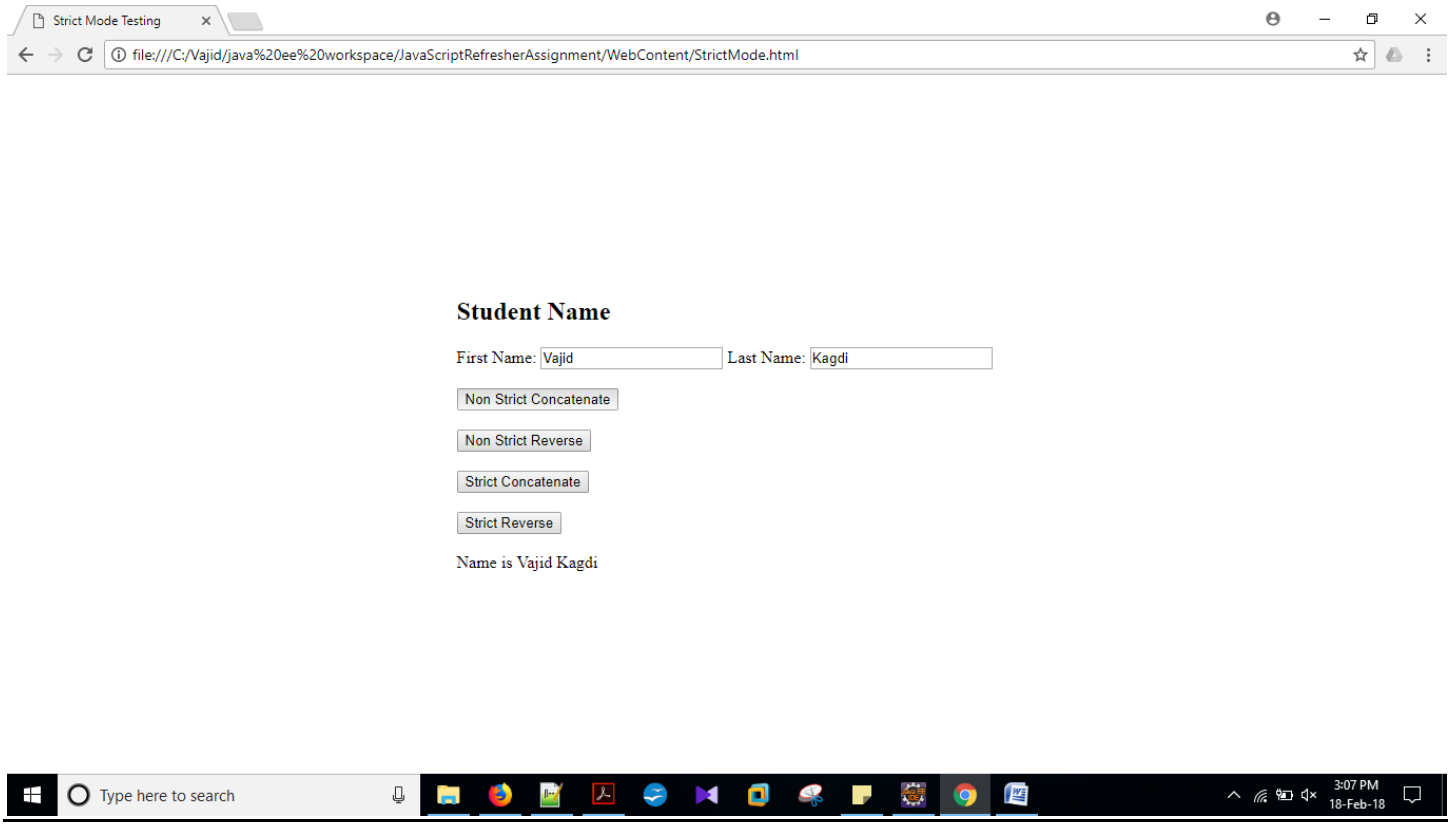
Your task is to create a HTML program using above requirement.

HTML Source Code:

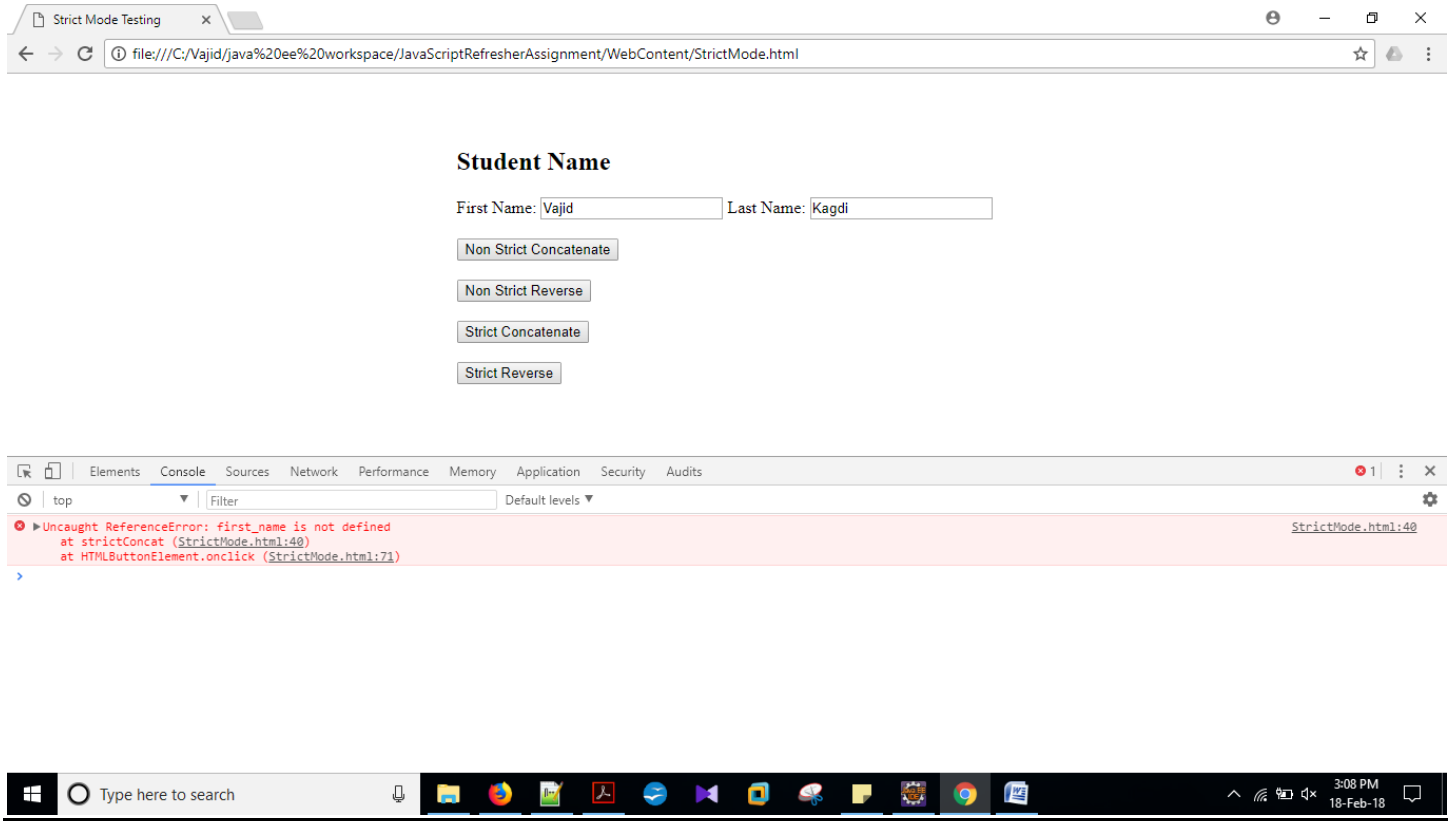
```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Strict Mode Testing</title>
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>
<script>
	function nonStrictConcat() {
		document.getElementById("result").innerHTML = "";
		first_name = document.getElementById("first_name").value;
		last_name = document.getElementById("last_name").value;
		document.getElementById("result").innerHTML = "Name is " + first_name
				+ " " + last_name;
	}

	function nonStrictReverse() {
```

```
            document.getElementById("result").innerHTML = "";
            first_name = document.getElementById("first_name").value;
            last_name = document.getElementById("last_name").value;
            document.getElementById("result").innerHTML = "Name is " + last_name
                        + " " + first_name;
      }
      function strictConcat() {
            "use strict";
            document.getElementById("result").innerHTML = "";
            first_name = document.getElementById("first_name").value;
            last_name = document.getElementById("last_name").value;
            document.getElementById("result").innerHTML = "Name is " + first_name
                        + " " + last_name;
      }

      function strictReverse() {
            "use strict";
            document.getElementById("result").innerHTML = "";
            first_name = document.getElementById("first_name").value;
            last_name = document.getElementById("last_name").value;
            document.getElementById("result").innerHTML = "Name is " + last_name
                        + " " + first_name;
      }
</script>
</head>
<body>
      <h2>Student Name</h2>
      First Name:
      <input type="text" id="first_name" /> Last Name:
      <input type="text" id="last_name" />
      <br />
      <br />
      <button id="button" onClick="nonStrictConcat()">Non Strict
            Concatenate</button>
      <br />
      <br />
      <button id="button" onClick="nonStrictReverse()">Non Strict
            Reverse</button>
      <br />
      <br />
      <button id="button" onClick="strictConcat()">Strict
            Concatenate</button>
      <br />
      <br />
      <button id="button" onClick="strictReverse()">Strict Reverse</button>
      <br />
      <br />
      <div id="result"></div>
</body>
</html>
```

Source Code Run:



Error With Strict Mode



Errors

Introdution:

- In Javascript Error is an object and it provides error information hen error occurs.
- Runtime errors result in new Error objects being created and thrown.
- Error Object Properties:
    - name : error name
    - message : error message
- When executing JavaScript code, different errors can occur.
- try statement lets you test a block of code for errors. catch statement lets you handle the error.
- throw statement lets you create custom errors

Scenario:

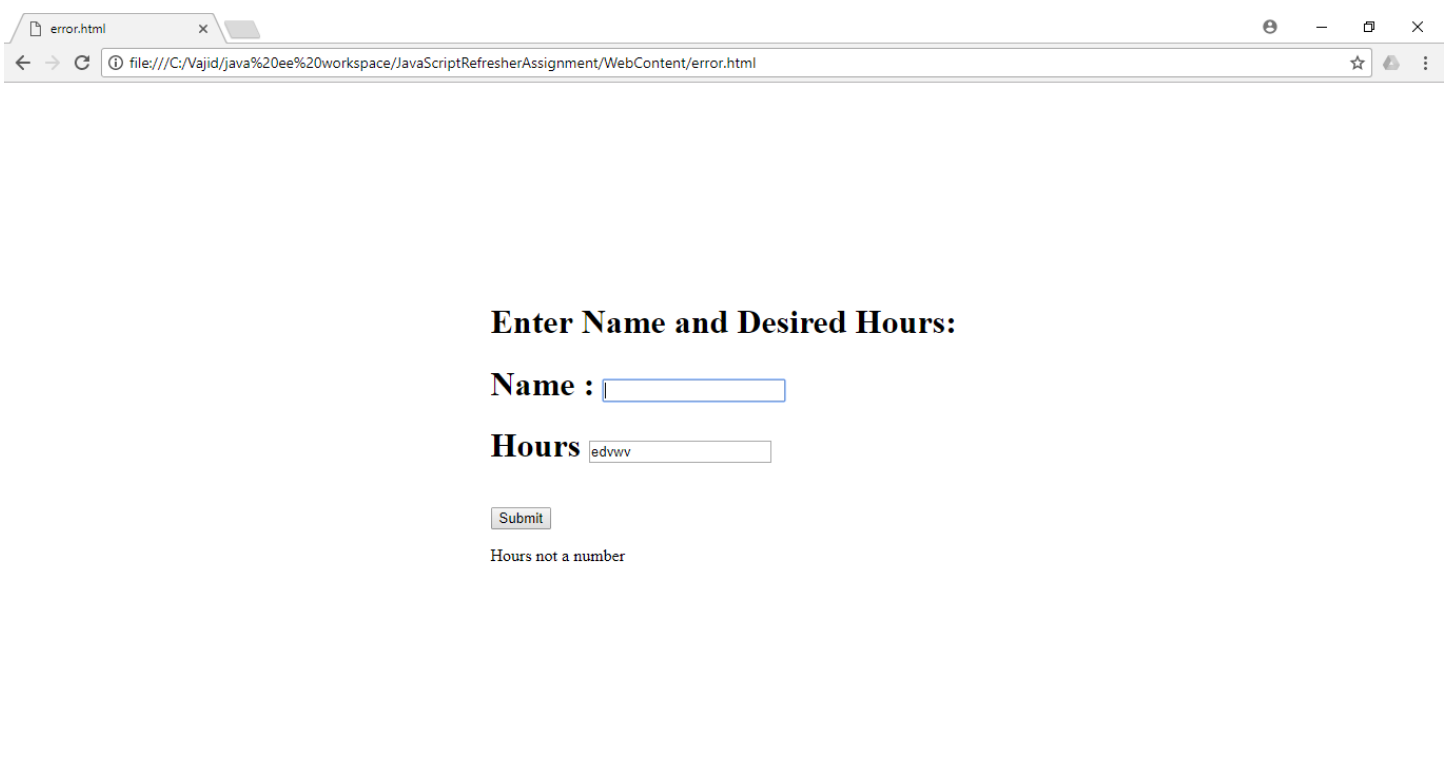
Spartan Shops in SJSU has started rolling out jobs for all open positions to students. The students who are interested to work can provide their name and number of hours they are available. Your task is to create the portal which will take above information. If the hours are not between 1 to 5, portal should display error.

HTML Source Code:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>
</head>
<body>


	<h1>Enter Name and Desired Hours:</h1>
	<form>
		<div>
			<h1>
				Name : <input type="text" id="name" />
		</div>
		<div>
			<h1>
				Hours <input id="hour" type="text" placeholder="Between 1-5"
					onchange="checkHours()">
		</div>
```

```
            <br>
            <button type="button">Submit</button>
            <p id="message"></p>
      </form>
      <script>
            function checkHours() {
                  var message, x;
                  message = document.getElementById("message");
                  message.innerHTML = "";
                  x = document.getElementById("hour").value;

                  //Using try with catch to handle the errors thrown
                  //Using throw keyword for throwing the error
                  try {
                        if (x == "")
                              throw "empty";
                        if (isNaN(x))
                              throw "not a number";
                        x = Number(x);
                        if (x < 1)
                              throw "too low";
                        if (x > 5)
                              throw "too high";
                  } catch (err) {
                        message.innerHTML = "Hours " + err;
                  }
            }
      </script>
</body>
</html>
```

Source Code Run:



error.html

file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/error.html

Enter Name and Desired Hours:

Name : Vajid Kagdi

Hours 6

Submit

Hours too high

Type here to search

8:40 PM
17-Feb-18

error.html

file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/error.html

Enter Name and Desired Hours:

Name : Vajid Kagdi

Hours 0

Submit

Hours too low

Type here to search

8:40 PM
17-Feb-18

Type Conversions

Introdution:

- JavaScript variables can be converted to a new variable of another data type
- This can happen in two ways:
    - By the use of a JavaScript function called explicit conversion.
    - Automatically by JavaScript called implicit conversion.
- Method **String()** can convert numbers to strings and it can be used on any type of numbers, literals, variables.
- method **Number()** can convert strings to numbers

Scenario:

Pan Cake Factory is a new store opening near San Jose State university. This store wants to come up with the portal where customers can place an order online. The portal should ask for name of the customer, no. of pan cakes. It should calculate the final price using number of pan cakes and the base price of the pan cake.

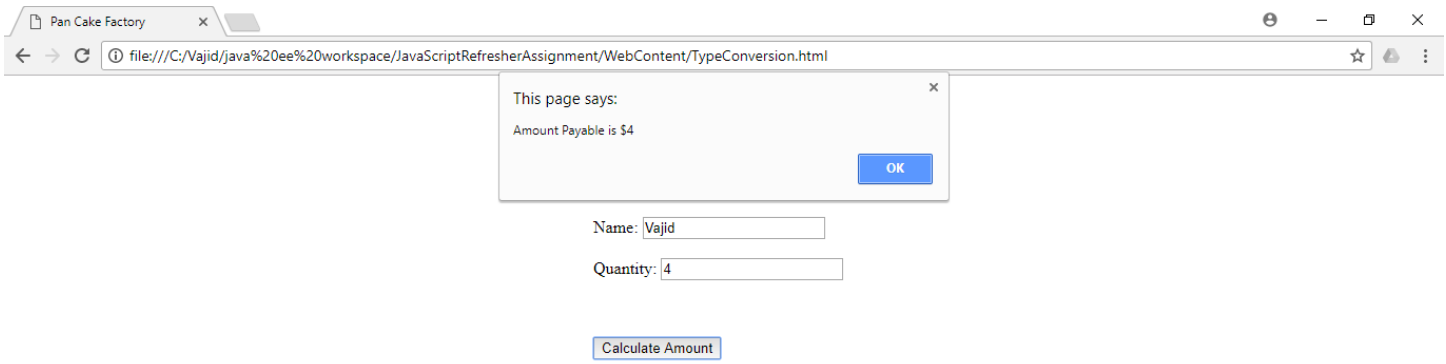
Your task is to create the portal and use conditions in Javascript to calculate the amount payable.

HTML Source Code:

```
<html>
<head>
<title>Pan Cake Factory</title>


<script type="text/javascript">
      function calculateOrder() {

            var simplePrice = 1.0;

            //Using Variable for storing value.
            var numCake = document.getElementById("cake").value;

            var thinCrust = document.getElementById("thinCrust");
            var cheeseBurst = document.getElementById("cheeseBurst");
            var doubleCheese = document.getElementById("doubleCheese");

            //Example of type conversion of String to Number
            var subTotal = numCake * simplePrice;

            //Example of type conversion of Number to String
            alert("Amount Payable is $" + subTotal);
      }
</script>
<style type="text/css">
html, body {
      height: 100%;
}

html {
      display: table;
      margin: auto;
```

```
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>
</head>

<body>
      <h1>Pan Cake Factory</h1>
      <form>
            <div>
                  Name: <input type="text" id="name" value="" />
            </div>
            <br>
            <div>
                  Quantity: <input type="text" id="cake" />
            </div>
            <br> <br>

            <div>
                  <br> <input type="button" value="Calculate Amount"
                        onClick="calculateOrder()" />
            </div>
      </form>
</body>
</html>
```

Source Code Run:



JSON

Introdution:

- JSON is a format for storing and transporting data and it stands for **J**ava**S**cript **O**bject **N**otation.
- JSON is a is lightweight data interchange format and is language independent.
- Code for reading and generating JSON data can be written in any programming language.
- JSON data is written as name/value pairs
- A common use of JSON is to read data from a web server, and display the data in a web page.
- JavaScript function JSON.parse() is used to convert JSON text into a JavaScript object.

Scenario:

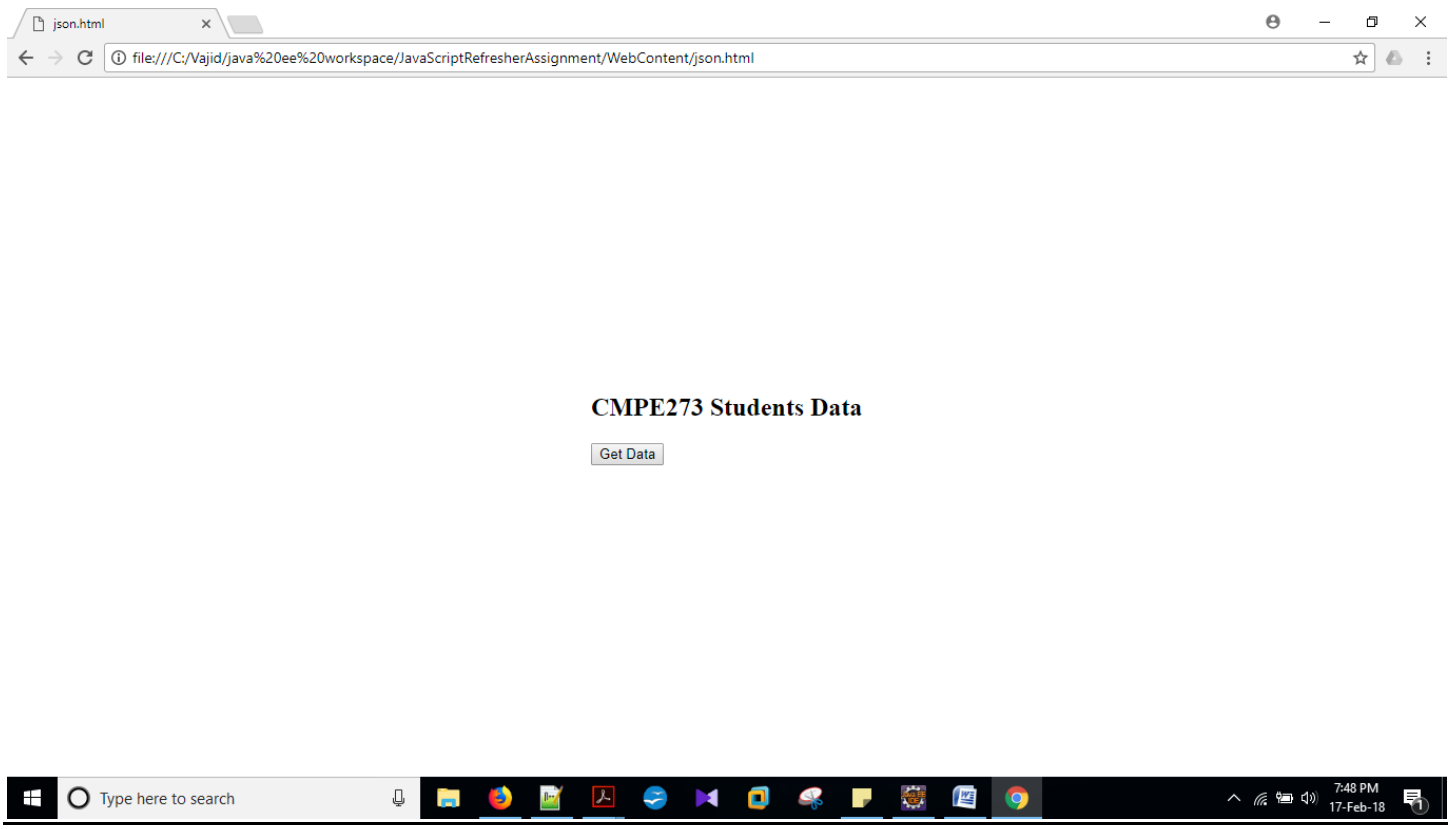
You are working on a team project and your team mate has done the backend part where he has written the API to bring Students data from the table and bring the data to front end in JSON format. Now your task is to create the front end and display student data.

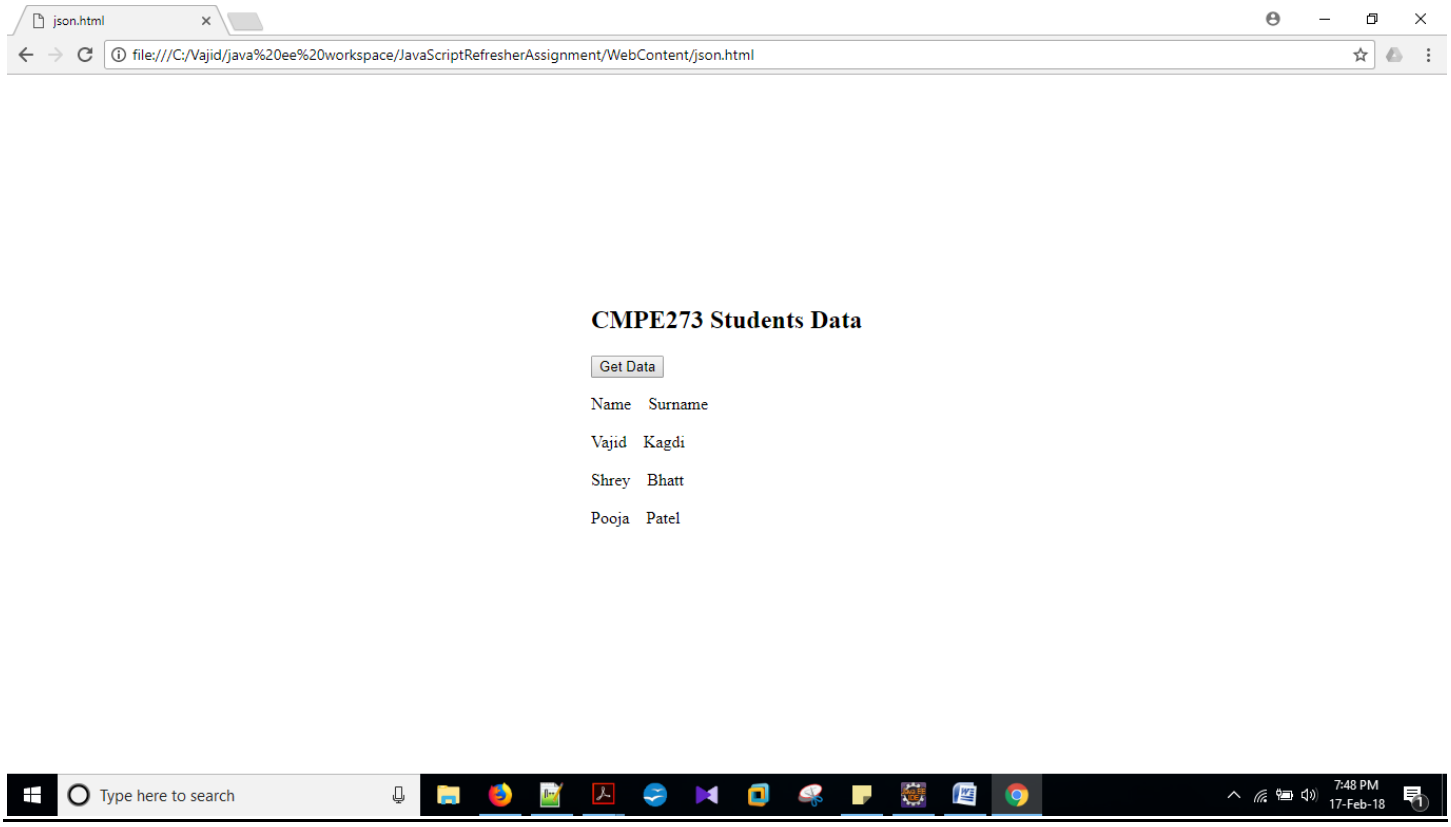
HTML Source Code:

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
html, body {
      height: 100%;
}

html {
      display: table;
      margin: auto;
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>
</head>
<body>

      <h2>CMPE273 Students Data</h2>
      <button onclick="getData()">Get Data</button>
      <p id="demo"></p>

      <script>
            function getData() {
                  var text = '{"students":['
                              + '{"firstName":"Name","lastName":"Surname" },'
                              + '{"firstName":"Vajid","lastName":"Kagdi" },'
                              + '{"firstName":"Shrey","lastName":"Bhatt" },'
                              + '{"firstName":"Pooja","lastName":"Patel" }]}';

                  //Parsing the JSON
```

```
                obj = JSON.parse(text);

                String
                output = "";

                for (i = 0; i < obj.students.length; i++) {
                      output = output + obj.students[i].firstName
                                  + "    " + obj.students[i].lastName
                                  + "<br><br>";

                }
                document.getElementById("demo").innerHTML = output;

            }
            //JSON Object:
        </script>

</body>
</html>
```

Source Code Run:




json.html
file:///C:/Vajid/java%20ee%20workspace/JavaScriptRefresherAssignment/WebContent/json.html
CMPE273 Students Data
Get Data
Name Surname
Vajid Kagdi
Shrey Bhatt
Pooja Patel
Type here to search
7:48 PM
17-Feb-18

HTML5

Local Storage

Introdution:

- Also called as web storage and it allows web applications can store data locally within the user's browser.
- Data is inserted in the form of name and value pair.
- Before HTML5, application data had to be stored in cookies, but web storage is more secure, and large amounts of data can be stored locally, without affecting performance.
- The storage limit is far larger than cookies and information is never transferred to the server.
- All pages, from one origin, can store and access the same data.
- Provides two objects for storing data:
    - window.localStorage - stores data with no expiration date
    - window.sessionStorage - stores data for one session and lost when browser closed

Scenario:

In order to remember the day to day tasks one of the professors has asked to create a In Browser Sticky Note application and it should have following features:

1. Add note
2. All notes should be visible even when browser is closed.
3. All notes should have checkbox which indicate that tasks are done or not.

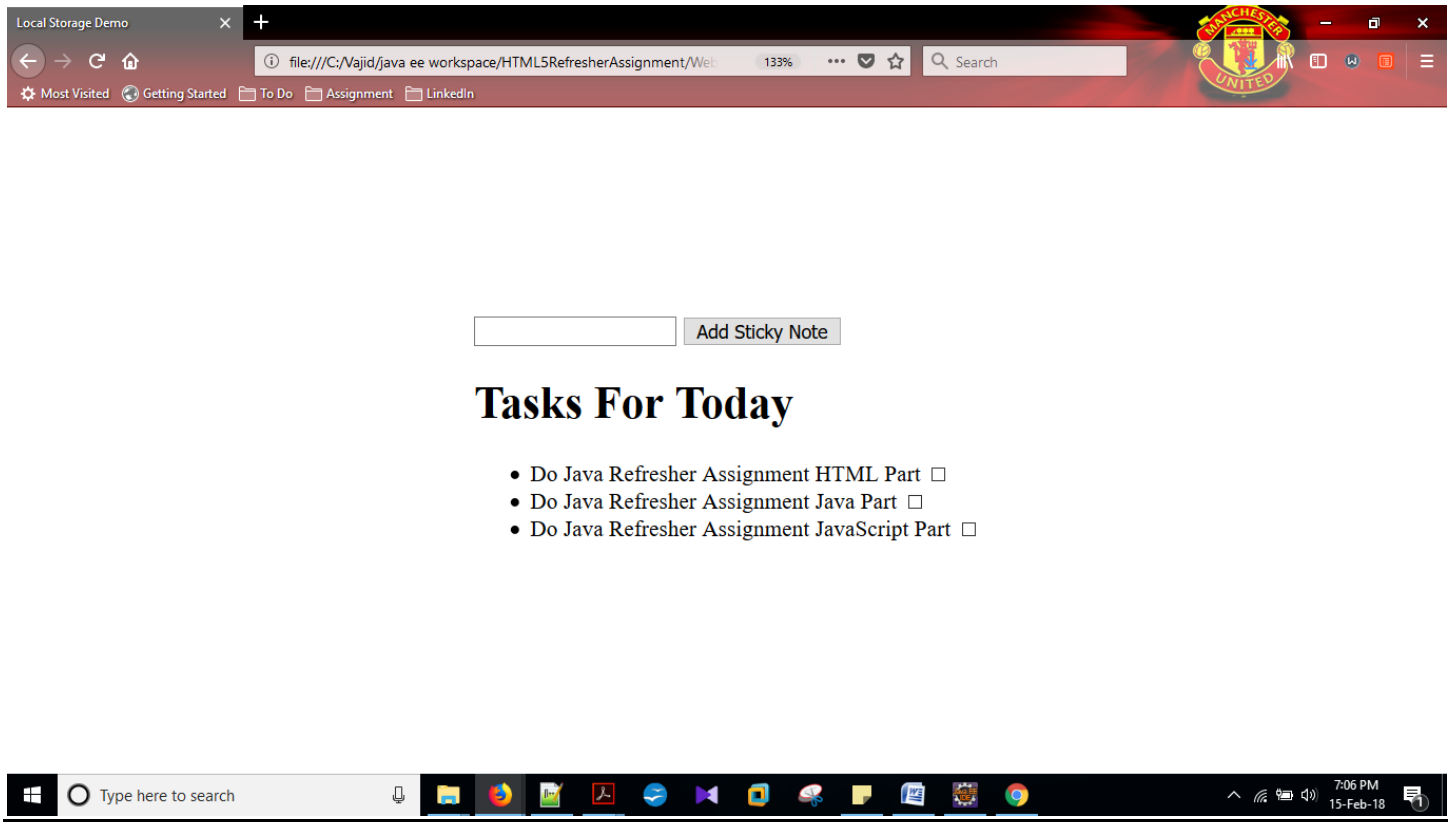
Your job is to create application using HTML5 local storage that provides these features.

HTML5 Source Code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Local Storage Demo</title>
<script type="text/javascript">
	window.onload = init;

	function init() {
		var button = document.getElementById("add_note");
		button.onclick = createNote;

		for (var i = 0; i < localStorage.length; i++) {
			var key = localStorage.key(i);
			if (key.substring(0, 6) == "sticky") {
				var value = localStorage.getItem(key);
				addNoteToDOM(value);
			}
		}
	}
	function addNoteToDOM(value) {
		var stickies = document.getElementById("listOfNote");
		var sticky = document.createElement("li");
```

```
            var span = document.createElement("span");
            var cb = document.createElement('input');
            cb.type = 'checkbox';
            span.innerHTML = value + "  ";
            sticky.appendChild(span)
            sticky.appendChild(cb);
            stickies.appendChild(sticky);
      }

      function createNote() {
            var value = document.getElementById("sticky_note").value;
            var key = "sticky_" + localStorage.length;
            localStorage.setItem(key, value);
            addNoteToDOM(value);
      }
</script>
<style type="text/css">
html, body {
      height: 100%;
}

html {
      display: table;
      margin: auto;
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>

</head>
<body>
      <form>
            <input type="text" id="sticky_note"> <input type="button"
                  id="add_note" value="Add Sticky Note">

            <h1>Tasks For Today</h1>
            <ul id="listOfNote">
      </form>

</body>
</html>
```

Source Run:


Local Storage Demo
file:///C:/Vajid/java ee workspace/HTML5RefresherAssignment/Web
133%
Search
Most Visited
Getting Started
To Do
Assignment
LinkedIn
Add Sticky Note
Tasks For Today
Do Java Refresher Assignment HTML Part
Do Java Refresher Assignment Java Part
Do Java Refresher Assignment JavaScript Part
Type here to search
7:06 PM
15-Feb-18

Media(Audio/Video)

Introdution:

- The <audio> tag is used to add sound to webpage.
- 3 supported file formats for the <audio> element: MP3, Wav, and Ogg.
- Attributes of <audio> tag : src, autoplay, controls, loop, muted and preload.
- Any text inside the between <audio> and </audio> will be displayed in browsers that do not support the <audio> tag.
- The <video> tag is used to add video to webpage.
- 3 supported video formats for the <video> element: MP4, WebM, and Ogg
- Any text between the <video> and </video> tags will be displayed in browsers that do not support the <video> element.
- Attributes of <video> tag : src, autoplay, controls, loop, muted, width, height and preload.

Scenario:

Professor Shim wants to upload the video and audio of lecture online so that students who cannot make it to the class with valid reasons can have access to the lecture. Following requirements are given by the professor:

- It should have link for each lecture taken till now.
- It should load the video and audio of lecture when clicking on the link
- The audio and video should have a play pause button.
- The video should have button to convert to big or small size when required.

Your task is to create the website having above requirements using audio and video tags of HTML5.

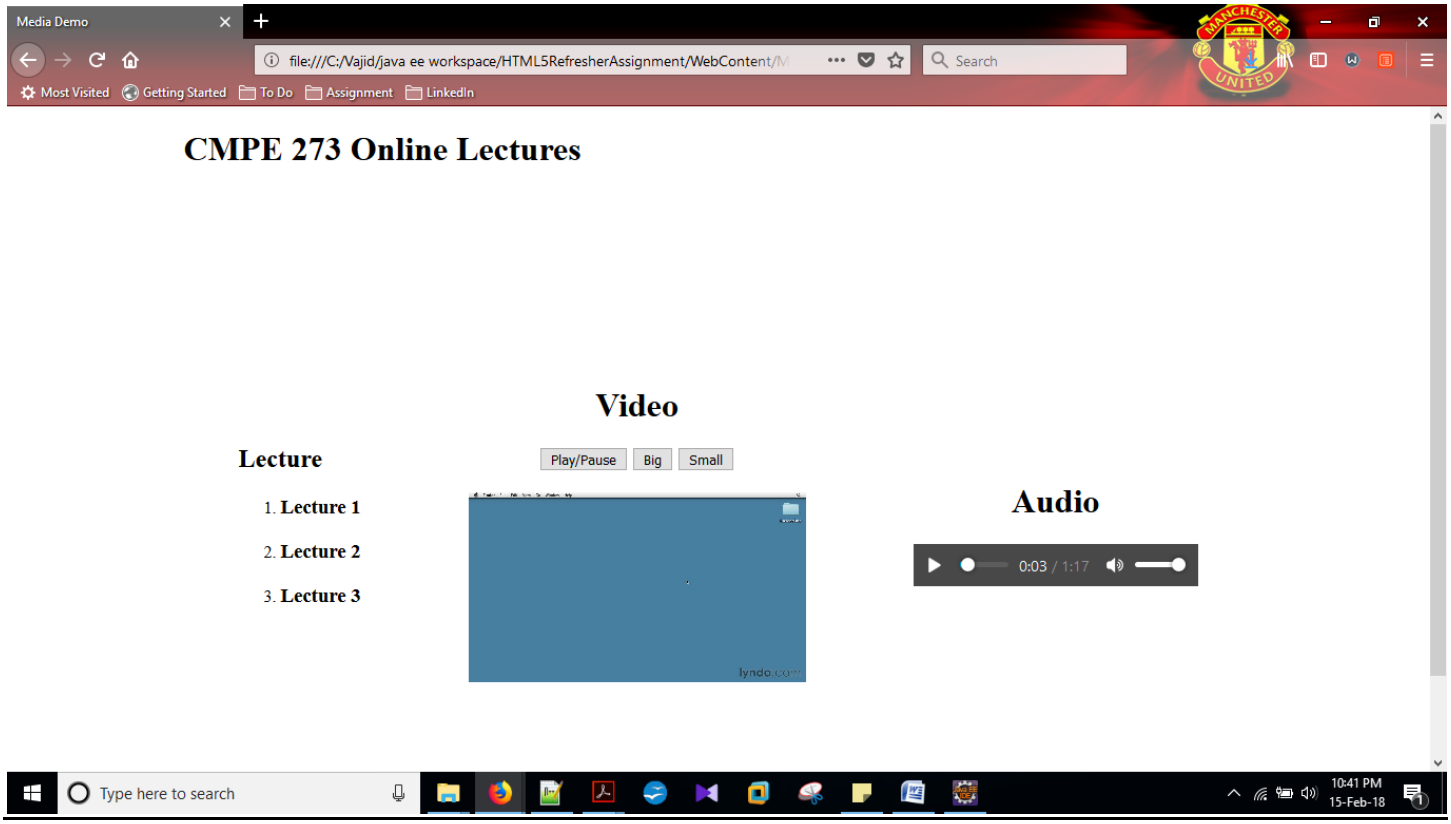
HTML5 Source Code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Media Demo</title>
<style type="text/css">
html, body {
	height: 100%;
}

html {
	display: table;
	margin: auto;
}

body {
	display: table-cell;
	vertical-align: middle;
}
</style>
<script>
	function getVideo(str) {
		//alert(str);
		var video = document.getElementById('video1');
```

```
            var sources = video.getElementsByTagName('source');
            sources[0].src = str + ".mp4";
            video.load();

            var audio = document.getElementById('audio1');
            var audioSources = audio.getElementsByTagName('source');
            audioSources[0].src = str + ".mp3";
            audio.load();

      }

      function playPause() {
            var myVideo = document.getElementById("video1");

            if (myVideo.paused)
                  myVideo.play();
            else
                  myVideo.pause();
      }

      function makeBig() {
            var myVideo = document.getElementById("video1");

            myVideo.width = 560;
      }

      function makeSmall() {
            var myVideo = document.getElementById("video1");

            myVideo.width = 320;
      }
</script>
</head>
<body>


      <table width=100% height=100% cellpadding="50">
            <h1>CMPE 273 Online Lectures</h1>
            <tr>
                  <td width=30%>
                        <h2>Lecture</h2>

                        <ol>
                              <li onclick="getVideo('Welcome')" id="1"><h3>Lecture 1</li>
                              <li onclick="getVideo('Using the exercise files')" id="1">
                                    <h3>Lecture 2
                              </li>
                              <li onclick="getVideo('Prerequisites')" id="1">
                                    <h3>Lecture 3
                              </li>
                        </ol>
                  </td>
                  <td width=40%><div style="text-align: center">
                              <h1>Video</h1>
                              <button onclick="playPause()">Play/Pause</button>
                              <button onclick="makeBig()">Big</button>
                              <button onclick="makeSmall()">Small</button>
                              <br> <br>
                              <video id="video1" width="420">
                                    <source id="source" src="Welcome.mp4"
type="video/mp4">
                              </video>
                        </div></td>
```

```
                <td width=30%><div style="text-align: center">
                            <h1>Audio</h1>
                            <audio controls id="audio1">
                                  <source src="welcome.mp3" type="audio/mpeg">
                            </audio>
                        </div></td>
            </tr>
      </table>
</body>
</html>
```

Source Code Run:



Input Type

Introdution:

- HTML5 introduced a number of new input types.
  - url : For entering a URL.
  - tel : For entering phone numbers.
  - email : For entering email addresses
  - number : For numeric input
  - range : For number input
  - date : For entering a date
- HTML5 introduced a number of new constraint attributes.
  - 'pattern' attribute specifies a regular expression used to validate an input field.
  - required attribute specifies that field must contain a value before the form can be submitted.
  - For numeric input types like number or range, you can specify the minimum and maximum values
  - maxlength attribute can be used to specify the maximum length of an input
- Datalist element : a list of suggested input values to associated with a form field.
- autofocus attribute : focus to immediately jump to a specific input
- placeholder attribute provides a hint to the user.

Scenario:

International Engineering Graduate Student Advisor is planning to keep a workshop designed for international graduate students for Overcoming Cultural Barriers in the US Interview. This workshop requires registration.

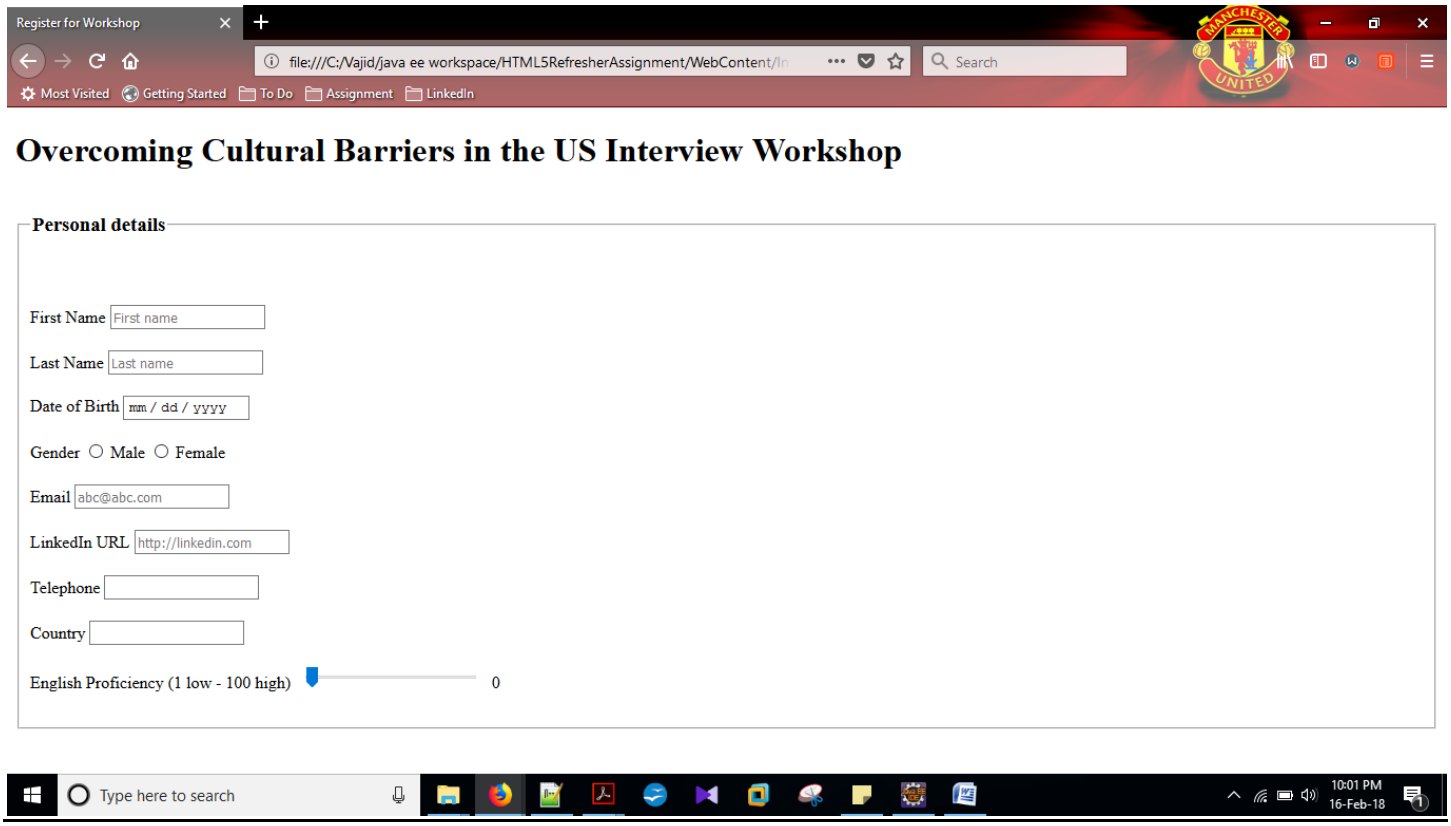
Your task is to create the form using HTML5 elements which will be used by students for registering for the event.

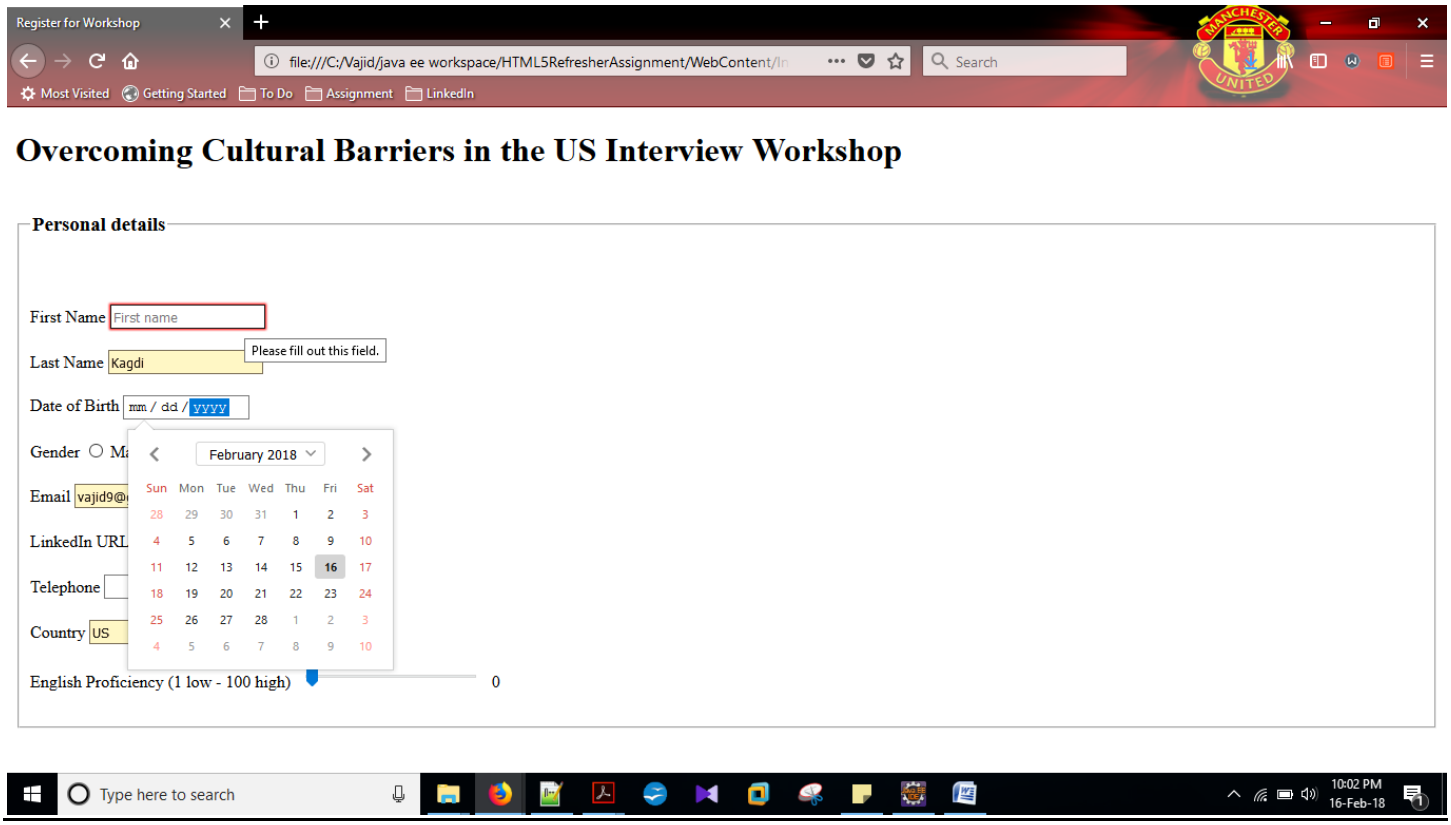
HTML5 Source Code:

```
<!DOCTYPE html>
<html>
<head>
<title>Register for Workshop</title>
</head>
<body>
      <form>
            <h1>Overcoming Cultural Barriers in the US Interview Workshop</h1>
            <fieldset>
                  <legend>
                        <h3>Personal details</h3>
                  </legend>
                  <br> <br>
                  <div>
                        <label>First Name <input placeholder="First name"
                              required="" autofocus="" type="text">
                        </label>
                  </div>
                  <br>
                  <div>
                        <label>Last Name <input placeholder="Last name" required=""
                              autofocus="" type="text">
                        </label>
                  </div>
                  <br>
```

```
                    <div>
                          <label>Date of Birth <input required="" type="date">
                          </label>
                    </div>
                    <br>
                    <div class="gender">
                          <label>Gender</label> <input id="male" type="radio">
<label>Male</label>
                          <input type="radio"> <label>Female</label>
                    </div>
                    <br>
                    <div>
                          <label>Email <input placeholder="abc@abc.com" required=""
                                type="email">
                          </label>
                    </div>
                    <br>
                    <div>
                          <label>LinkedIn URL <input placeholder="http://linkedin.com"
                                type="url">
                          </label>
                    </div>
                    <br>
                    <div>
                          <label>Telephone <input required="" type="tel">
                          </label>
                    </div>
                    <br>
                    <div>
                          <label>Country <input list="country" required="" type="text">
                                <datalist id="country">
                                      <option label="India" value="India"></option>
                                      <option label="China" value="China"></option>
                                </datalist>
                          </label>
                    </div>
                    <br>
                    <div>
                          <label>English Proficiency (1 low - 100 high) <input min="1"
                                max="100" value="0" type="range"> <output name="output"
                                      onforminput="value=a.value">0</output>
                          </label>
                    </div>
                    <br>
              </fieldset>
      </form>

</body>
</html>
```

Source Code Run:





Geolocation

Introdution:

- This is used to get geographical position of a user.
- The getCurrentPosition() method is used to return the user's position.
- If the getCurrentPosition() method is successful, it returns a coordinates object to the function specified in the parameter (showPosition)
- The showPosition() function outputs the Latitude and Longitude
- T he second parameter of the getCurrentPosition() method is used to handle errors.
- To display the results in Map, Google Map services can be used.
- Geolocation object has one method watchPosition() which returns the current position of the user and continues to return updated position as the user moves.

Scenario:

San Jose State University wants all the students to know the distance between their off campus current location to the San Jose State University location. Following requirements are given by the university:

- It should have a Button to know the location and distance
- On clicking the button it should display the current location in Map as well as the Latitude and Longitutde
- On clicking the button it should display the distance between the SJSU and current location of user.

Your task is to create the website having above requirements using 'geolocation' feature of HTML5.

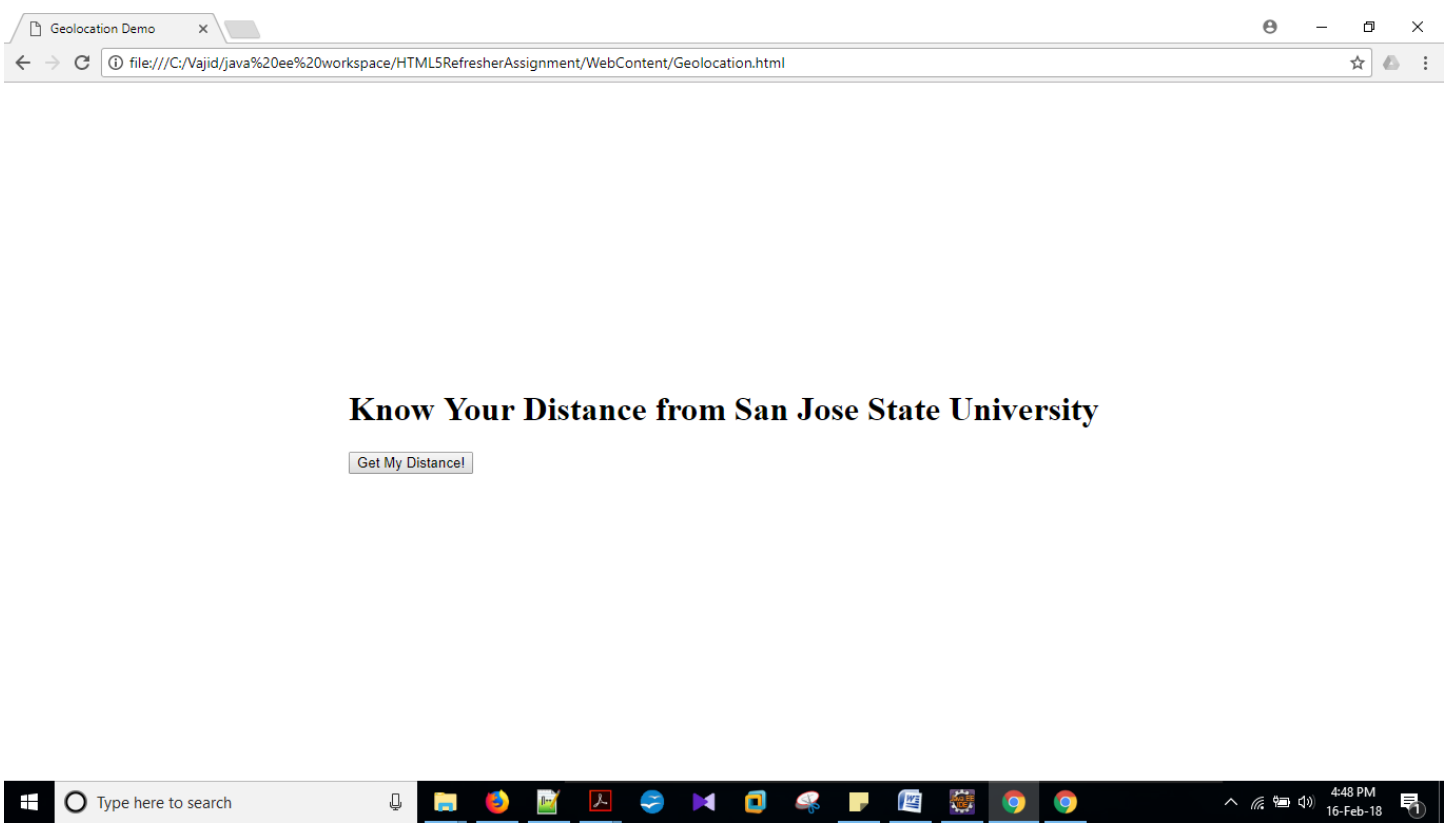
HTML5 Source Code:

```
<!DOCTYPE html>
<html>
<head>
<title>Geolocation Demo</title>
<style type="text/css">
html, body {
      height: 100%;
}

html {
      display: table;
      margin: auto;
}

body {
      display: table-cell;
      vertical-align: middle;
}
</style>
<script src="https://maps.google.com/maps/api/js?sensor=true"></script>
<script>
      var x = document.getElementById("error");
```

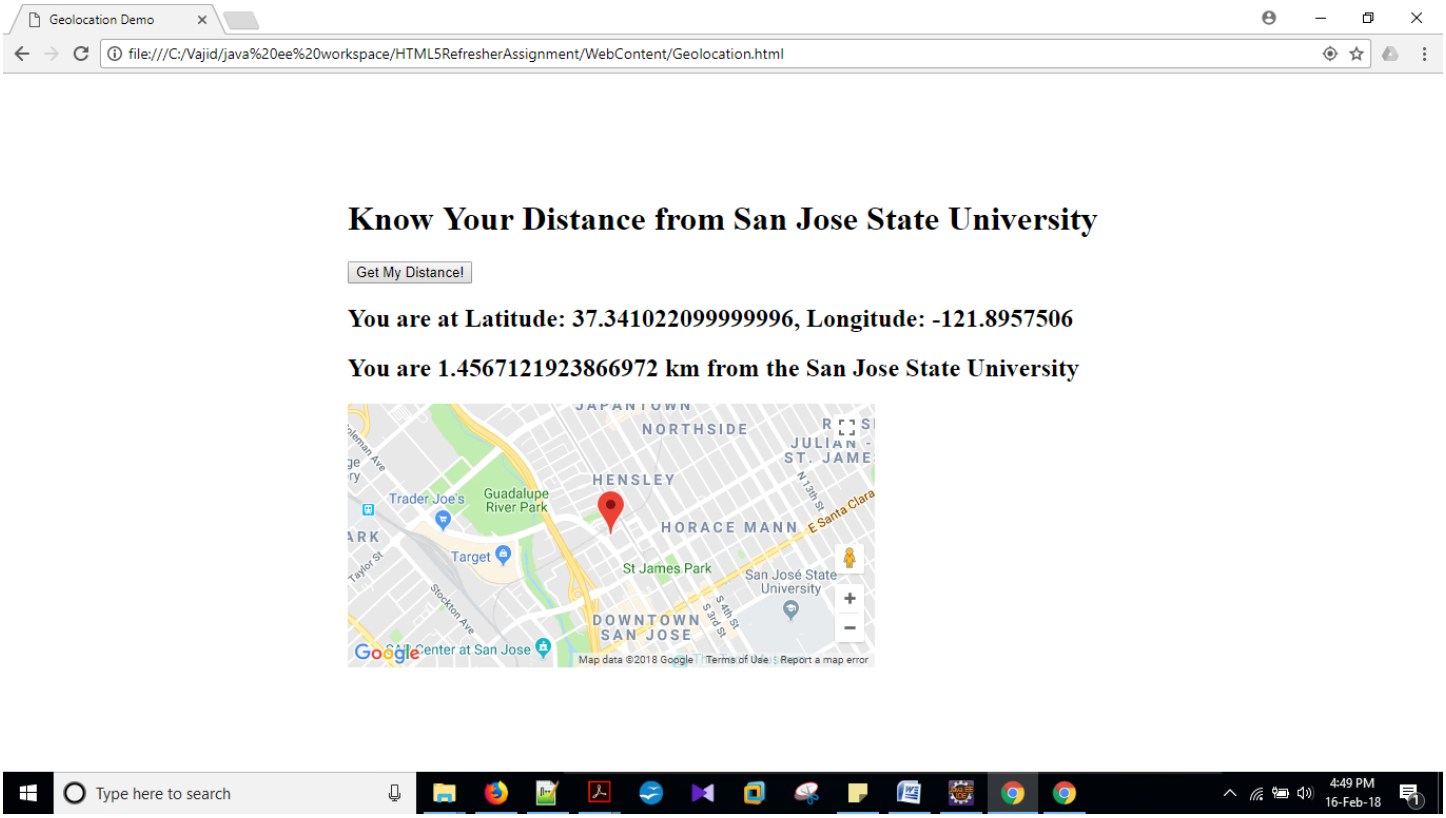
```
function getLocation() {
	if (navigator.geolocation) {
		navigator.geolocation.getCurrentPosition(showPosition);
	} else {
		x.innerHTML = "Geolocation is not supported by this browser.";
	}
}

function showPosition(position) {
	var lat = position.coords.latitude;
	var lon = position.coords.longitude;

	var div = document.getElementById("location");
	div.innerHTML = "You are at Latitude: " + lat + ", Longitude: " + lon;

	var latlon = new google.maps.LatLng(lat, lon)
	var mapholder = document.getElementById('mapholder')
	mapholder.style.height = '250px';
	mapholder.style.width = '500px';

	var myOptions = {
		center : latlon,
		zoom : 14,
		mapTypeId : google.maps.MapTypeId.ROADMAP,
		mapTypeControl : false,
		navigationControlOptions : {
			style : google.maps.NavigationControlStyle.SMALL
		}
	}

	var map = new google.maps.Map(document.getElementById("mapholder"),
			myOptions);
	var marker = new google.maps.Marker({
		position : latlon,
		map : map,
		title : "You are here!"
	});

	var sjsuCoords = {
		latitude : 37.3352,
		longitude : -121.88099
	};

	var km = computeDistance(position.coords, sjsuCoords);
	var div = document.getElementById("distance");
	distance.innerHTML = "You are " + km
			+ " km from the San Jose State University";
	showMap(position.coords);
}

function computeDistance(startCoords, destCoords) {
	var startLatRads = degreesToRadians(startCoords.latitude);
	var startLongRads = degreesToRadians(startCoords.longitude);
	var destLatRads = degreesToRadians(destCoords.latitude);
	var destLongRads = degreesToRadians(destCoords.longitude);

	//Taken help online for calculating distance using coordinates.
	var Radius = 6371; // radius of the Earth in km
	var distance = Math.acos(Math.sin(startLatRads) * Math.sin(destLatRads)
			+ Math.cos(startLatRads) * Math.cos(destLatRads)
			* Math.cos(startLongRads - destLongRads))
			* Radius;
	return distance;
```

```
        }
        function degreesToRadians(degrees) {
                var radians = (degrees * Math.PI) / 180;
                return radians;
        }
</script>

</head>

<body>

        <h1>Know Your Distance from San Jose State University</h1>

        <button onclick="getLocation()">Get My Distance!</button>


        <div id="error"></div>
        <h2>
                <div id="location"></div>
        </h2>
        <h2>
                <div id="distance"></div>
        </h2>
        <div id="mapholder"></div>


</body>
</html>
```

Source Code Run:



After Click on Button:



Java

Queues

Introdution:

- Abstract data structure which keeps objects in sequence and it is open at both ends
- One end is used for inserting and other end is used for removing data from queue.
- It follows First In – First Out(FIFO) method.
- Operations are called:
  - **enqueue()** – store item to the queue.
  - **dequeue()** – remove item to the queue.
- The time needed to add or delete an item is constant and independent of the number of items in the queue.
- Used to handle transaction processing in applications.

Scenario:

MLK Library has asked Student Assistant to keep the returned books of same author in specific order back to shelf. The order is decided by the priority of the books. The priority is decided on the condition of the book and the its date of purchase.

Your task is to create a program which will take input the books to be kept on shelf in the form of List and will return the least priority book to be kept on the shelf first using Priority Queue.

Source Class:

Book Class

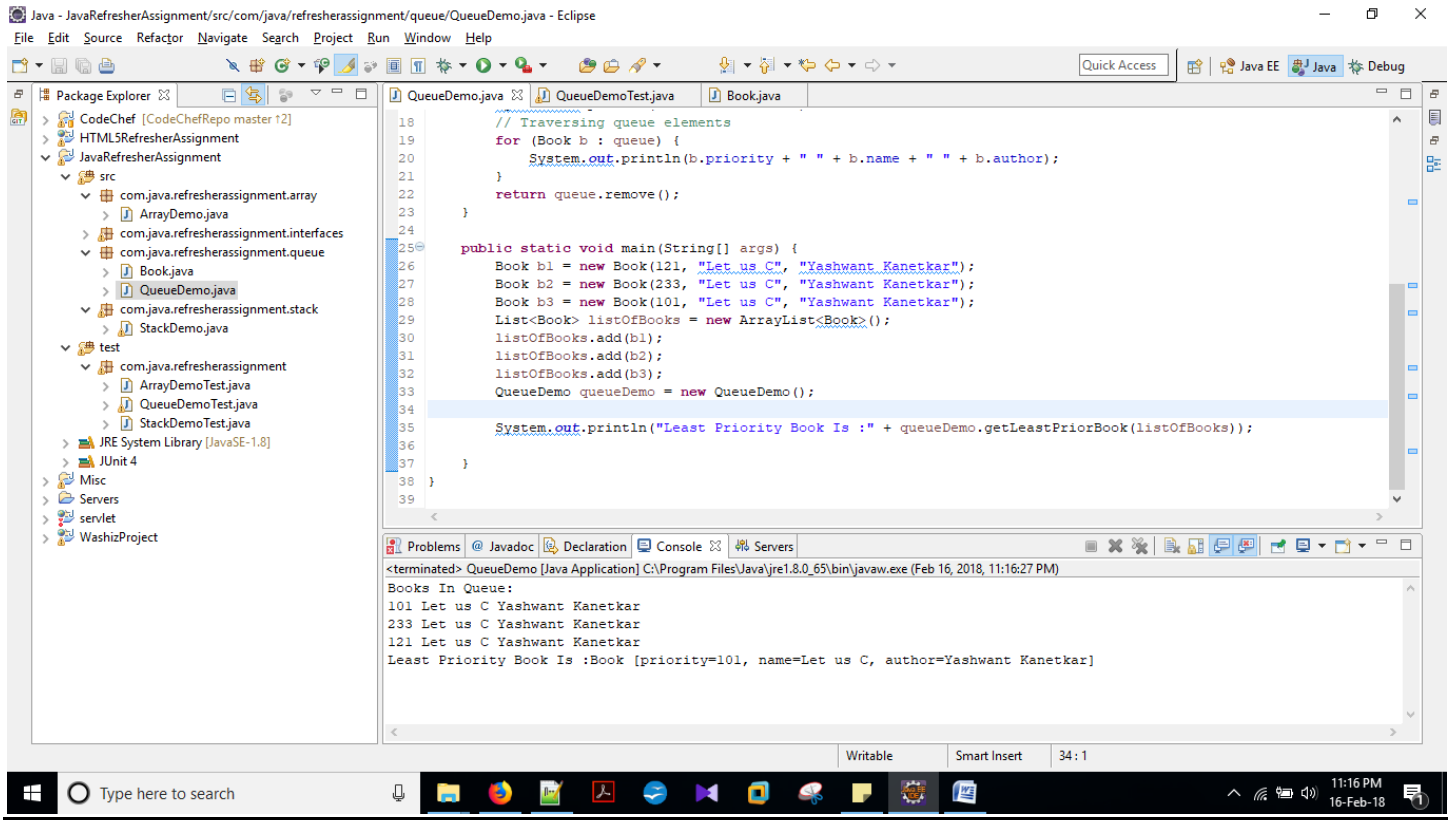
```
package com.java.refresherassignment.queue;

public class Book implements Comparable<Book> {
	public int priority;
	String name, author;

	public Book(int priority, String name, String author) {
		this.priority = priority;
		this.name = name;
		this.author = author;
	}

	@Override
	public String toString() {
		return "Book [priority=" + priority + ", name=" + name + ", author="
			+ author + "]";
	}

	// This method will be used by PriorityQueue to sort
	public int compareTo(Book b) {
		if (priority > b.priority) {
			return 1;
```

```
            } else if (priority < b.priority) {
                  return -1;
            } else {
                  return 0;
            }
      }
}
```

QueueDemo Class

```
package com.java.refresherassignment.queue;

import java.util.*;

public class QueueDemo {
      public Book getLeastPriorBook(List<Book> listOfBooks) {

            // Using Priority Queue
            Queue<Book> queue = new PriorityQueue<Book>();

            // Add Books to the queue using list
            for (Book b : listOfBooks) {
                  queue.add(b);

            }

            System.out.println("Books In Queue:");
            // Traversing queue elements
            for (Book b : queue) {
                  System.out.println(b.priority + " " + b.name + " " + b.author);
            }
            return queue.remove();
      }

      public static void main(String[] args) {
            Book b1 = new Book(121, "Let us C", "Yashwant Kanetkar");
            Book b2 = new Book(233, "Let us C", "Yashwant Kanetkar");
            Book b3 = new Book(101, "Let us C", "Yashwant Kanetkar");
            List<Book> listOfBooks = new ArrayList<Book>();
            listOfBooks.add(b1);
            listOfBooks.add(b2);
            listOfBooks.add(b3);
            QueueDemo queueDemo = new QueueDemo();

            System.out.println("Least Priority Book Is :" +
queueDemo.getLeastPriorBook(listOfBooks));

      }
}
```

Source Class Run:



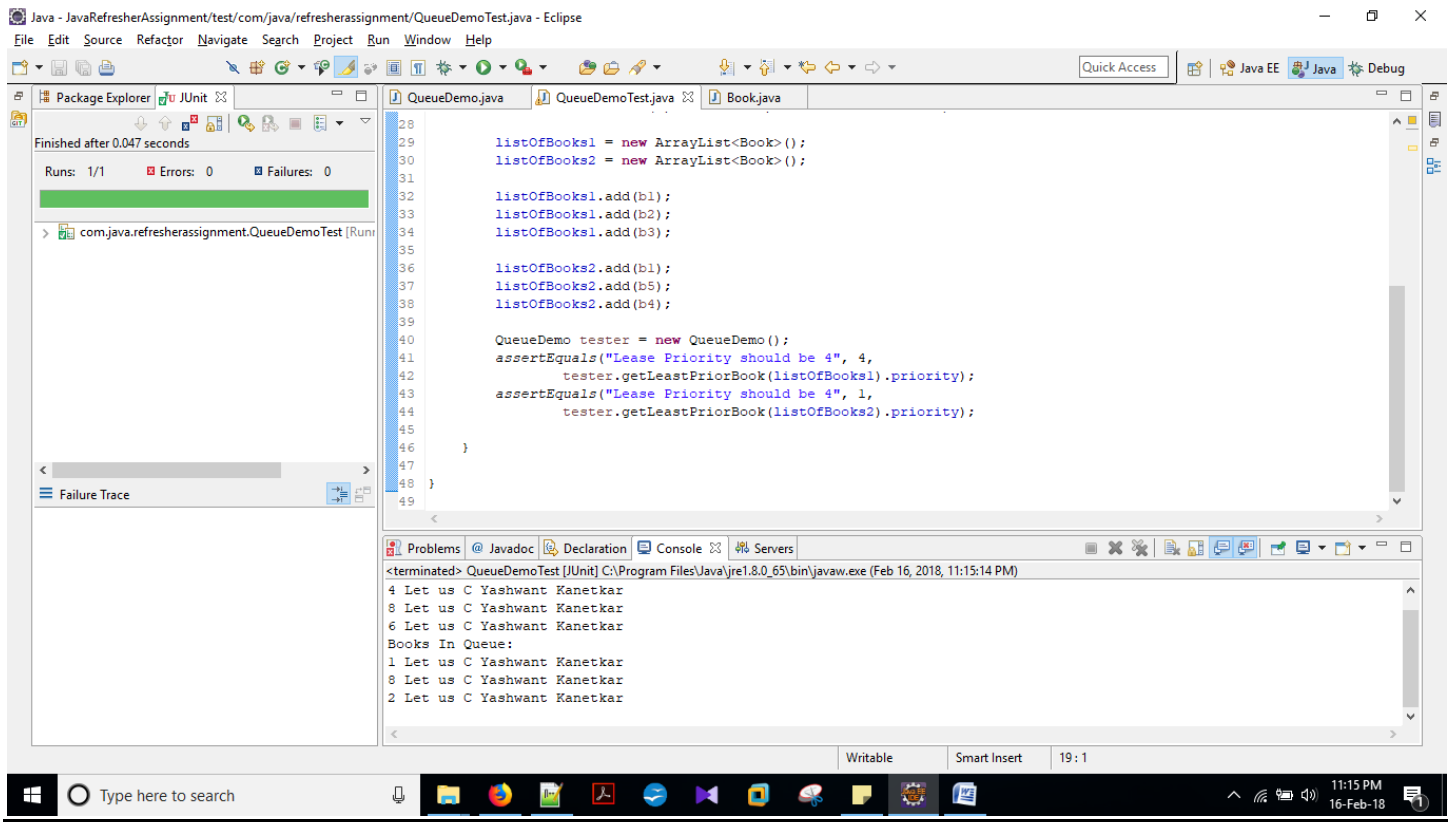
Test Class:

```
package com.java.refresherassignment;


import static org.junit.Assert.assertEquals;

import static org.junit.Assert.fail;


import java.util.ArrayList;

import java.util.List;


import org.junit.BeforeClass;

import org.junit.Test;


import com.java.refresherassignment.queue.Book;
```

```
import com.java.refresherassignment.queue.QueueDemo;

public class QueueDemoTest {

	List<Book> listOfBooks1;
	List<Book> listOfBooks2;

	@Test
	public void testGetLeastPriorBook() {

		Book b1 = new Book(8, "Let us C", "Yashwant Kanetkar");
		Book b2 = new Book(6, "Let us C", "Yashwant Kanetkar");
		Book b3 = new Book(4, "Let us C", "Yashwant Kanetkar");
		Book b4 = new Book(2, "Let us C", "Yashwant Kanetkar");
		Book b5 = new Book(1, "Let us C", "Yashwant Kanetkar");

		listOfBooks1 = new ArrayList<Book>();
		listOfBooks2 = new ArrayList<Book>();

		listOfBooks1.add(b1);
		listOfBooks1.add(b2);
		listOfBooks1.add(b3);

		listOfBooks2.add(b1);
		listOfBooks2.add(b5);
		listOfBooks2.add(b4);
```

```
        QueueDemo tester = new QueueDemo();

        assertEquals("Least Priority should be 4", 4,

                tester.getLeastPriorBook(listOfBooks1).priority);

        assertEquals("Least Priority should be 1", 1,

                tester.getLeastPriorBook(listOfBooks2).priority);


    }


}
```

Test Class Run:



Stacks

Introdution:

- A linear data structure represented as a pile of books or plates, where insertion and deletion of elements can take place at only one end that is top of the stack.
- Also called as Last In First Out(LIFO) Structure.
- Two basic operations that can be performed :
  - **Push() :** add element to stack
  - **Pop() :** remove element to stack
- Stack internally has a pointer: TOP, which refers to the top of the Stack element.
- Advantages: Easy to implement. Memory is saved as pointers are not involved.
- Disadvantages : It is not dynamic. It doesn't grow and shrink depending on needs at runtime.

Scenario:

In order to prepare for emergency situation, SJSU library decides to keep a count of student at all time in SJSU library. Whenever a student enters the library, an entry is made in the system and count is incremented by one. Whenever a student leaves the library, count is decremented by one. In this way, the database maintains count.

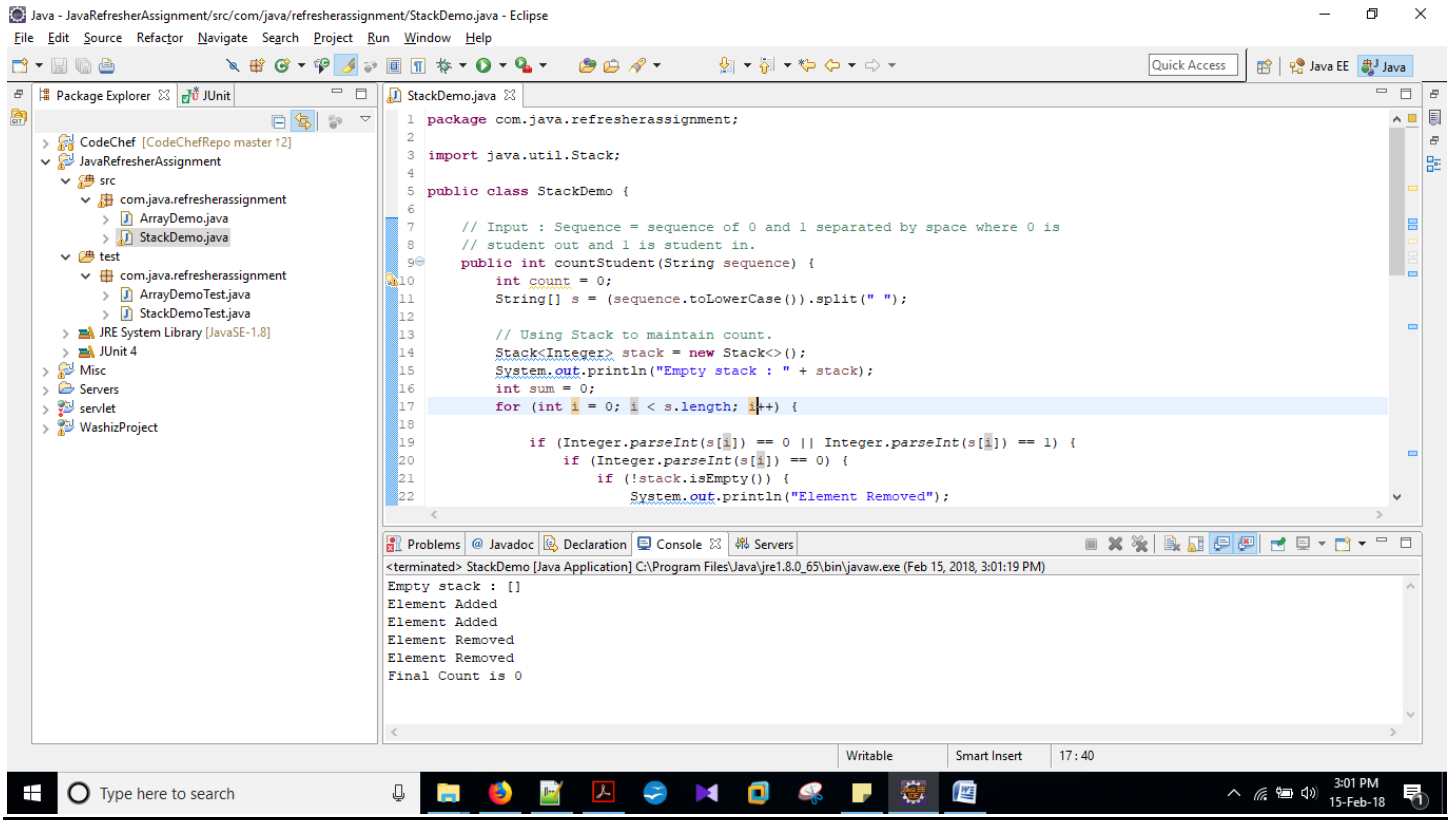
Your job is to study the create program that maintains count and give the number of students that are "still inside" the library.
Pre Conditions : Max Count can be : 100

Source Class:

```
package com.java.refresherassignment;

import java.util.Stack;

public class StackDemo {

	// Input : Sequence = sequence of 0 and 1 separated by space where 0 is
	// student out and 1 is student in.
	public int countStudent(String sequence) {
		int count = 0;
		String[] s = (sequence.toLowerCase()).split(" ");

		// Using Stack to maintain count.
		Stack<Integer> stack = new Stack<>();
		System.out.println("Empty stack : " + stack);
		int sum = 0;
		for (int i = 0; i < s.length; i++) {

			if (Integer.parseInt(s[i]) == 0 || Integer.parseInt(s[i]) == 1) {
				if (Integer.parseInt(s[i]) == 0) {
					if (!stack.isEmpty()) {
						System.out.println("Element Removed");
						sum -= 1;
						stack.pop();
					} else {
						throw new IllegalArgumentException();
```

```
                        }

                    } else {
                        System.out.println("Element Added");
                        sum += 1;
                        stack.push(sum);
                    }

                } else {
                    throw new IllegalArgumentException();
                }
            }

            if (stack.isEmpty()) {
                return 0;
            }
            return stack.pop();

        }

        public static void main(String[] args) {
            StackDemo stackDemo = new StackDemo();
            System.out.println("Final Count is "
                    + stackDemo.countStudent("1 1 0 0"));

        }
}
```

Source Class Run:



Test Class:

```
package com.java.refresherassignment;


import static org.junit.Assert.*;


import org.junit.Test;


public class StackDemoTest {


	@Test
	public void testStudentCount() {
		StackDemo tester = new StackDemo();
		assertEquals("Student Count Should be 0", 0,  tester.countStudent("1 0 1 1 0 0"));
		assertEquals("Student Count Should be 2", 2,  tester.countStudent("1 1 0 0 1 1"));
		assertEquals("Student Count Should be 10", 10,  tester.countStudent("1 1 1 1 0 1 1 1 1 1 1 1"));
	}


	 @Test(expected = IllegalArgumentException.class)
	  public void testExceptionIsThrown() {
	    StackDemo tester = new StackDemo();
	    tester.countStudent("0 0 1");
	    tester.countStudent("1 0 0");
	    tester.countStudent("1 1 1 0 0 0 0");


	    }
}
```

Test Class Run:

Java - JavaRefresherAssignment/test/com/java/refresherassignment/StackDemoTest.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access | Java EE | Java

Package Explorer | JUnit

Finished after 0.047 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.java.refresherassignment.StackDemoTest [Runne

testStudentCount (0.000 s)

testExceptionIsThrown (0.000 s)

Failure Trace

StackDemo.java | StackDemoTest.java

```
3⊕ import static org.junit.Assert.*;
6
7  public class StackDemoTest {
8
9⊖     @Test
10     public void testStudentCount() {
11         StackDemo tester = new StackDemo();
12         assertEquals("Student Count Should be 0", 0,  tester.countStudent("1 0 1 1 0 0"));
13         assertEquals("Student Count Should be 2", 2,  tester.countStudent("1 1 0 0 1 1"));
14         assertEquals("Student Count Should be 10", 10,  tester.countStudent("1 1 1 1 0 1 1 1 1 1 1 1"));
15     }
16
17⊖     @Test(expected = IllegalArgumentException.class)
18      public void testExceptionIsThrown() {
19         StackDemo tester = new StackDemo();
20         tester.countStudent("0 0 1");
21         tester.countStudent("1 0 0");
22         tester.countStudent("1 1 1 0 0 0 0");
23
24         }
25 }
26
```

Problems | Javadoc | Declaration | Console | Servers

<terminated> StackDemoTest [JUnit] C:\Program Files\Java\jre1.8.0_65\bin\javaw.exe (Feb 15, 2018, 3:02:32 PM)

```
Element Added
Element Added
Element Added
Element Added
Element Added
Element Added
Empty stack : []
```

Type here to search

3:02 PM
15-Feb-18

Arrays

Introdution:

- Collection of item of same type stored in continuous memory locations
- Each element can be uniquely identified with their index, where in index start from 0.
- Allows random access of element with the help of index and it is fast. Also have better cache locality.
- An array declaration has two components: the data type which can also be primitive data type and the name through which we can reference array.
- Size of an array once fixed cannot be changed at runtime.
- Two types of arrays in Java
    - Single Dimensional Array Syntax : dataType arr[] = new dataType[size];
    - Multi Dimensional Array Syntax : dataType arr[][] = new dataType[size1] [size2];
- Advantages: Its elements can be accessed directly and simply when the index value of an element is known
- Disadvantages : Arrays are fixed size and elements cannot be inserted or deleted easily. To insert or delete an element, other elements following the element have to be moved.

Scenario:

Wallmart has come up with the Valentines Day lottery system in which only couples with following condition can take part:
A couple in which a man with name M and a woman with name W can take part in lottery if and only if M is anagram of W or W is anagram of M.
One name is an anagram of another if the second is simply a rearrangement of the first.
For example, 'heart' and 'earth' are anagrams.

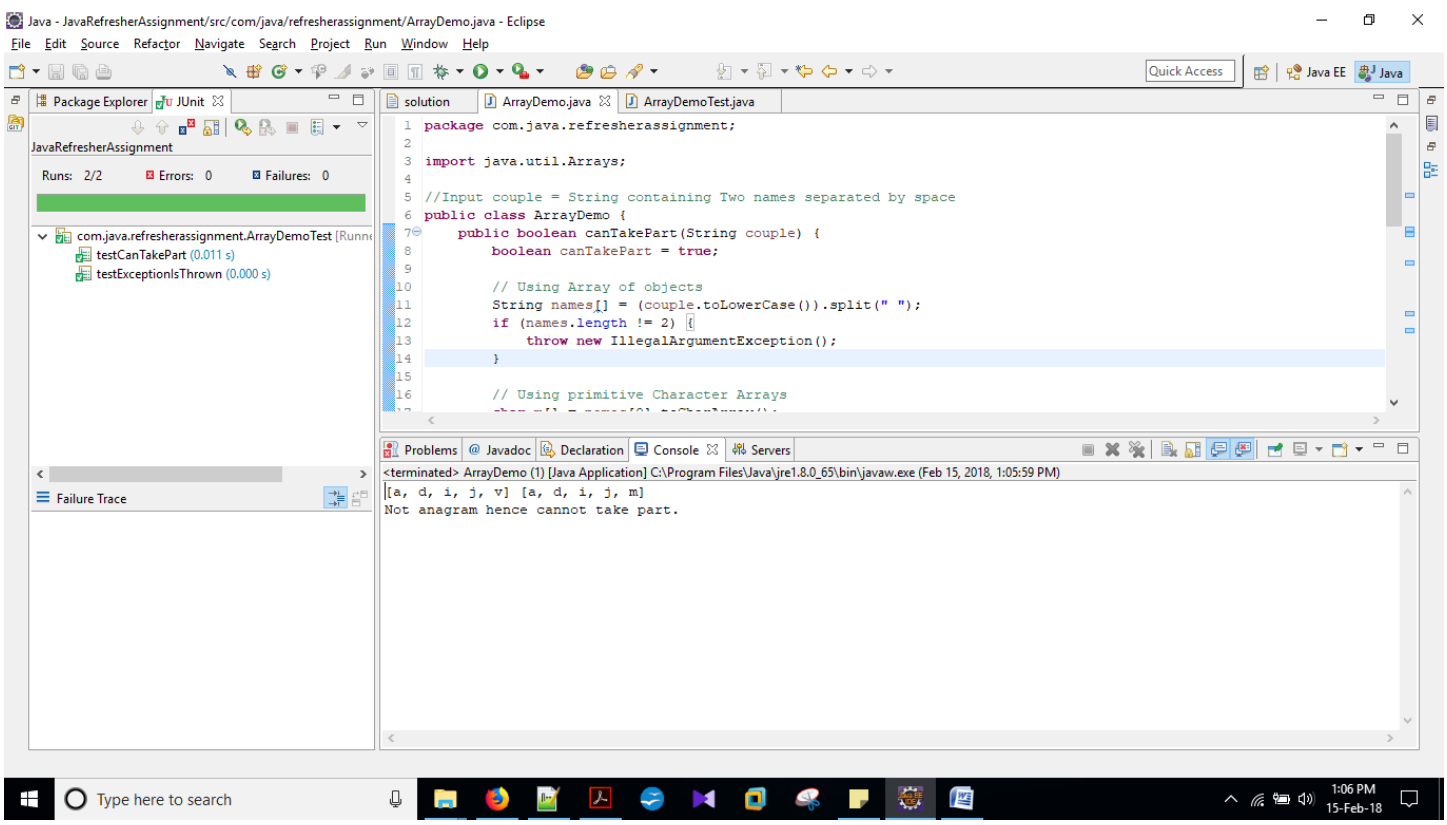
Below is the program to determine whether a couple is allowed to take part in lottery or not.

Source Class:

```
package com.java.refresherassignment;

import java.util.Arrays;

//Input couple = String containing Two names separated by space
public class ArrayDemo {
      public boolean canTakePart(String couple) {
            boolean canTakePart = true;

            // Using Array of objects
            String names[] = (couple.toLowerCase()).split(" ");
            if (names.length != 2) {
                  throw new IllegalArgumentException();
            }

            // Using primitive Character Arrays
            char m[] = names[0].toCharArray();
            char f[] = names[1].toCharArray();
```

```java
        // Using Sort method of Array
        Arrays.sort(m);
        Arrays.sort(f);
        System.out.println(Arrays.toString(m) + " " + Arrays.toString(f));

        // Comparing characters of Array
        for (int i = 0; i < m.length; i++) {
            if (m[i] != f[i]) {
                canTakePart = false;
            }
        }
        if (canTakePart) {
            System.out.println("Are anagram hence can take part.");

        } else {
            System.out.println("Not anagram hence cannot take part.");
        }
        return canTakePart;

    }

    public static void main(String[] args) {
        ArrayDemo arrayDemo = new ArrayDemo();
        arrayDemo.canTakePart("Vajid Majid");
    }
}
```

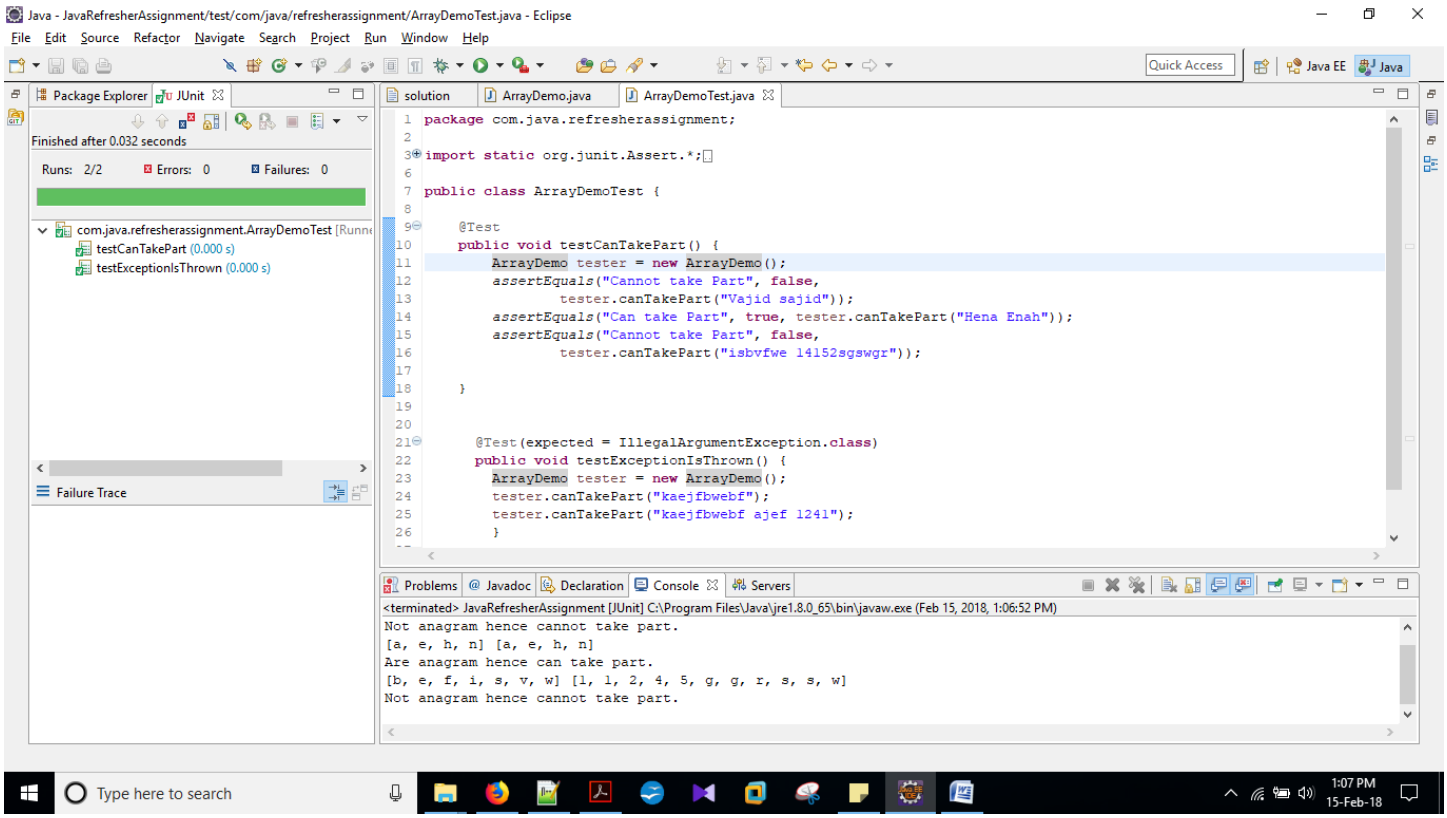
Source Class Run:



Test Class:

```
package com.java.refresherassignment;

import static org.junit.Assert.*;

import org.junit.Test;

public class ArrayDemoTest {

	@Test
	public void testCanTakePart() {
		ArrayDemo tester = new ArrayDemo();
		assertEquals("Cannot take Part", false,
				tester.canTakePart("Vajid sajid"));
		assertEquals("Can take Part", true, tester.canTakePart("Hena Enah"));
		assertEquals("Cannot take Part", false,
				tester.canTakePart("isbvfwe 14152sgswgr"));

	}


	 @Test(expected = IllegalArgumentException.class)
	 public void testExceptionIsThrown() {
	  ArrayDemo tester = new ArrayDemo();
	  tester.canTakePart("kaejfbwebf");
	  tester.canTakePart("kaejfbwebf ajef 1241");
	  }

}
```

Test Class Run:



Interfaces

Introdution:

- They are like 100% abstract class. Can be considered as a contract, it defines what class has to do but it does not provide how it can be done. For eg. Animal interface might declare that all Animal implementation classes have an eat() method, but the Animal interface doesn't supply any logic for the eat() method.
- Any class that implements the interface has to provide the logic for the same.
- “interface” keyword is used to declare an interface.
- All interface methods are implicitly public and abstract and all variables defined in an interface must be public, static, and final
- An interface can extend one or more other interfacesbut cannot extend or implement other class.
- Interface types can be used polymorphically
- “implements” keyword is used by class implementing any interface and one class can implement multiple interfaces.

Scenario:

Indian Government wants to build an API so that other developers can write their own display pages to show Indian cricket scores to the users and use this APIs. We will now create an application that displays 3 elements : runs, wickets and overs. All these will be updated in Scoreboard object in real time as match progress and APIs will send latest data.

Interface ThirdParty : which all 3rd Party need to implement if they want a score update.

Interface Cricket : all score stations need to implement this interface.

Class CricketData : class implementing Cricket Interface

Class ThirdPartyDisplay : class implementing ThirdParty Interface

Class CricketSimulation : Simulating the API experience

Source Class:

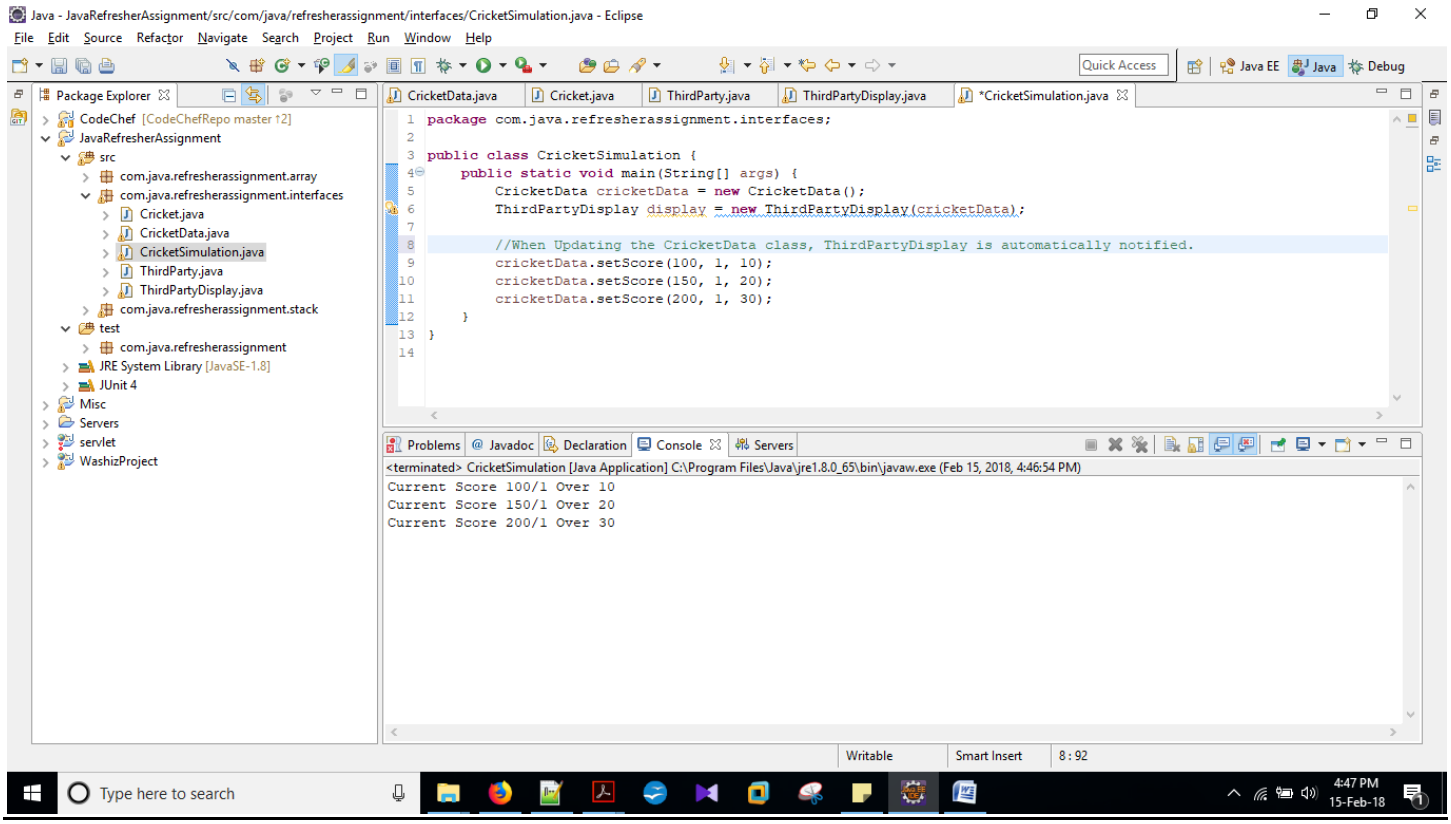
```
package com.java.refresherassignment.interfaces;

public interface Cricket {
	public void regiserThirdParty(ThirdParty o);

	public void removeThirdParty(ThirdParty o);

	public void updateThirdParty();
}


package com.java.refresherassignment.interfaces;

public interface ThirdParty {
```

```java
	public void update(int run, int wicket, int overs);
}


package com.java.refresherassignment.interfaces;

import java.util.ArrayList;

public class CricketData implements Cricket {

	private ArrayList thirdParty;
	private int run, wicket, overs;

	public int getRun() {
		return run;
	}

	public void setScore(int run, int wicket, int overs) {
		this.run = run;
		this.wicket = wicket;
		this.overs = overs;
		scoreChanged();
	}

	public int getWicket() {
		return wicket;
	}

	public int getOvers() {
		return overs;
	}

	public CricketData() {
		thirdParty = new ArrayList();
	}

	@Override
	public void regiserThirdParty(ThirdParty o) {
		thirdParty.add(o);

	}

	@Override
	public void removeThirdParty(ThirdParty o) {
		int i = thirdParty.indexOf(o);
		if (i >= 0) {
			thirdParty.remove(i);
		}
	}

	@Override
	public void updateThirdParty() {
		for (int i = 0; i < thirdParty.size(); i++) {
			ThirdParty party = (ThirdParty) thirdParty.get(i);
			party.update(run, wicket, overs);
		}
	}

	public void scoreChanged() {
		updateThirdParty();
	}

}
```

```java
package com.java.refresherassignment.interfaces;

public class ThirdPartyDisplay implements ThirdParty {

	private int run, wicket, overs;
	private Subject cricketData;

	public ThirdPartyDisplay(Cricket cricketData) {
		this.cricketData = cricketData;
		cricketData.regiserThirdParty(this);
	}

	@Override
	public void update(int run, int wicket, int overs) {
		this.run = run;
		this.wicket = wicket;
		this.overs = overs;
		display();
	}

	private void display() {
		System.out.println("Current Score " + run + "/" + wicket + " Over "
				+ overs);
	}

}
```

```java
package com.java.refresherassignment.interfaces;

public class CricketSimulation {
	public static void main(String[] args) {
		CricketData cricketData = new CricketData();
		ThirdPartyDisplay display = new ThirdPartyDisplay(cricketData);

		cricketData.setScore(100, 1, 10);
		cricketData.setScore(150, 1, 20);
		cricketData.setScore(200, 1, 30);
	}
}
```

Source Class Run:

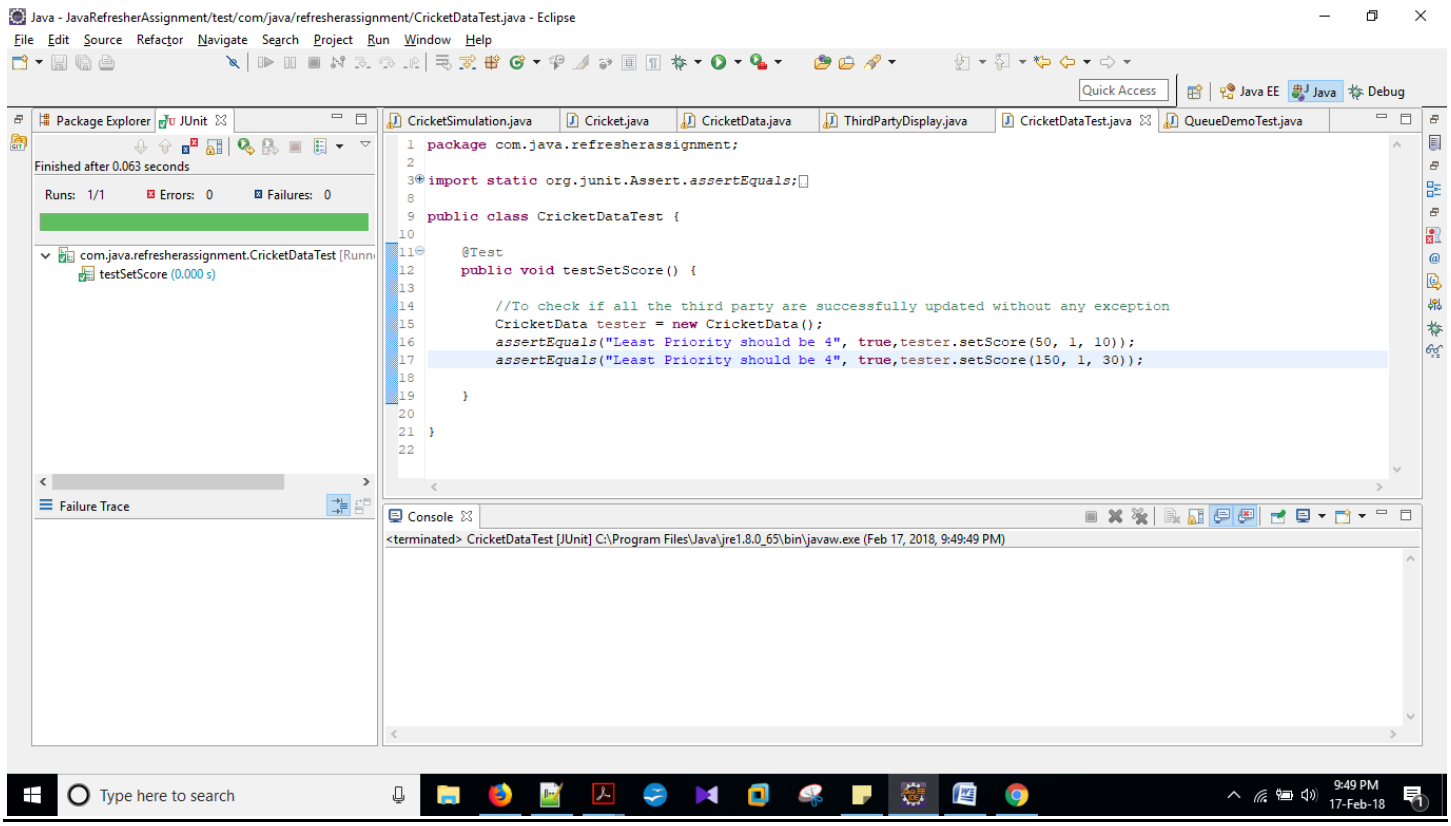


Test Class:

```
package com.java.refresherassignment;


import static org.junit.Assert.assertEquals;


import org.junit.Test;


import com.java.refresherassignment.interfaces.CricketData;


public class CricketDataTest {


	@Test

	public void testSetScore() {
```

```
        //To check if all the third party are successfully updated without any exception
        CricketData tester = new CricketData();
        assertEquals("Least Priority should be 4", true,tester.setScore(50, 1, 10));
        assertEquals("Least Priority should be 4", true,tester.setScore(150, 1, 30));

    }

}
```

Test Class Run:



Collections

Introdution:

- Collection represents a group of objects
- It is a framework that provides ways to store and manipulate group of objects.
- Operations like searching, sorting, insertion, manipulation, deletion can be performed.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc)
- The java.util package contains all the classes and interfaces for Collection framework.
- It also contains an Iterator interface which provides facility of iterating the elements.
- Collection is the root of the collection hierarchy.
- `Set` — a collection that cannot contain duplicate elements.
- `List` — an ordered collection (sometimes called a *sequence*). `List`s can contain duplicate elements.
- `Map` — an object that maps keys to values.
- Following are advantages of Collections Framework:
    - Programming effort reduced.
    - Speed and quality of program increased.
    - Reduces effort to write new structure.
    - Provides reusability.

Scenario:

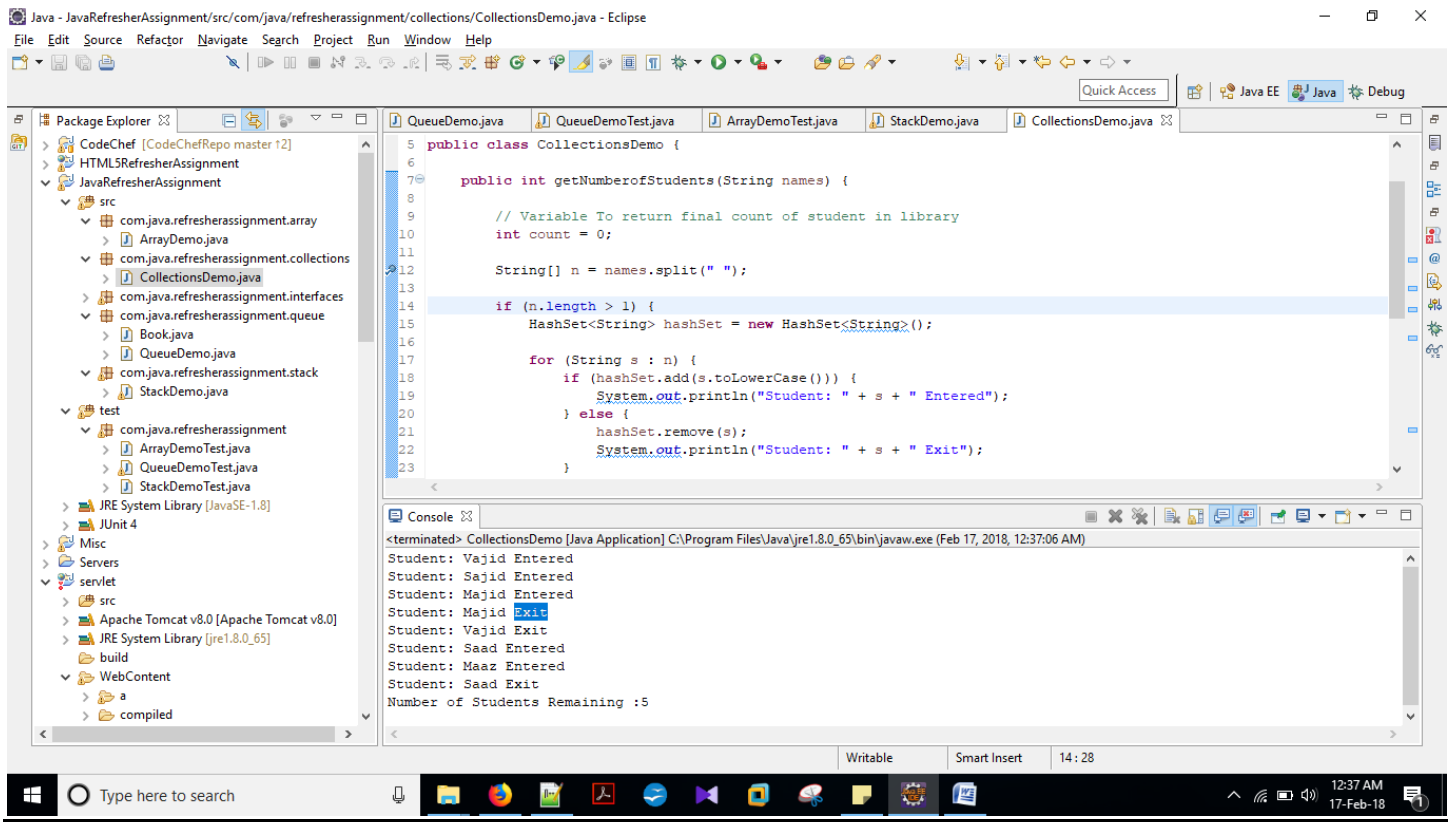
One of the highly used buildings in the premises of SJSU is the library. The library has a unique database management system. The description of the system is as follows:
Whenever a student enters the library, an entry is made in the system containing the student's name. Whenever a student leaves the library, again an entry is made in the system containing the student's name. In this way, the database entries are made. As an input, you are given this database containing various entries. Your job is to study the database and give the number of students that are "still inside" the library.

Source Class:

```
package com.java.refresherassignment.collections;

import java.util.HashSet;

public class CollectionsDemo {

	public int getNumberofStudents(String names) {

		// Variable To return final count of student in library
		int count = 0;

		String[] n = names.split(" ");

		if (n.length > 1) {
			HashSet<String> hashSet = new HashSet<String>();

			for (String s : n) {
				if (hashSet.add(s.toLowerCase())) {
					System.out.println("Student: " + s + " Entered");
				} else {
					hashSet.remove(s);
```

```
                        System.out.println("Student: " + s + " Exit");
                    }
                }
                count = hashSet.size();
                System.out.println("Number of Students Remaining :" + count);
            } else {
                throw new IllegalArgumentException();
            }

            return count;
        }

        public static void main(String[] args) {
            CollectionsDemo demo = new CollectionsDemo();
            // Normal Execution
            demo.getNumberofStudents("Vajid Sajid Majid Majid Vajid Saad Maaz Saad");
            // Exception
            // demo.getNumberofStudents("");

        }
}
```

Source Class Run:

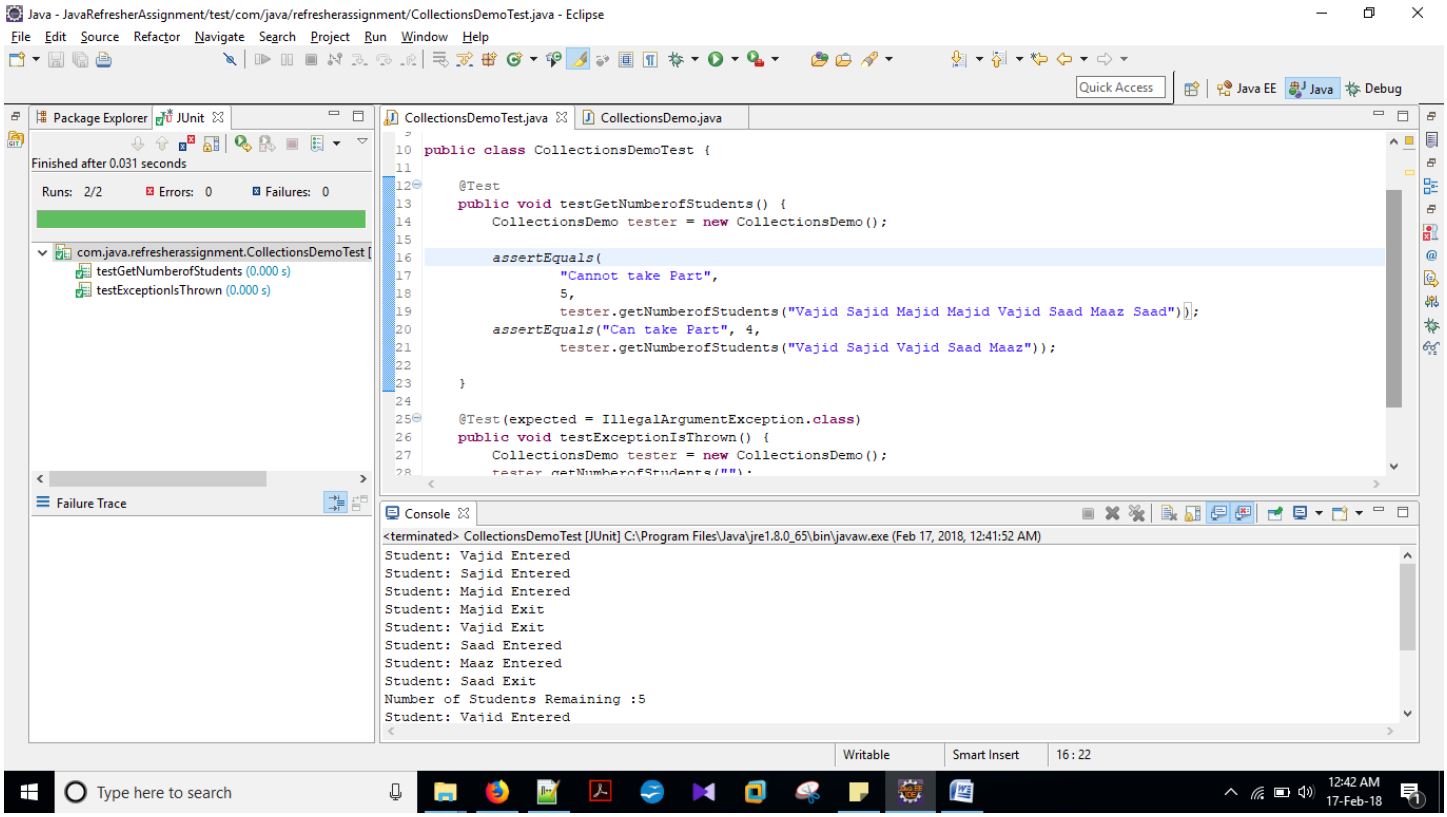


Test Class:

```
package com.java.refresherassignment;


import static org.junit.Assert.assertEquals;


import org.junit.Test;


import com.java.refresherassignment.array.ArrayDemo;
import com.java.refresherassignment.collections.CollectionsDemo;


public class CollectionsDemoTest {


	@Test
	public void testGetNumberofStudents() {
		CollectionsDemo tester = new CollectionsDemo();


		assertEquals(
				"Cannot take Part",
				5,
				tester.getNumberofStudents("Vajid Sajid Majid Majid Vajid Saad Maaz Saad"));
		assertEquals("Can take Part", 4,
				tester.getNumberofStudents("Vajid Sajid Vajid Saad Maaz"));


	}


	@Test(expected = IllegalArgumentException.class)
	public void testExceptionIsThrown() {
```

```
        CollectionsDemo tester = new CollectionsDemo();

        tester.getNumberofStudents("");

    }


}
```

Test Class Run:



Generics

Introdution:

- It represents abstraction over types. It allows developing abstract code and provide concrete types to operate on later.
- Added to Java after Release 5. It was added to provide compile- time type checking and removing risk of Class Cast Exception.
- Collections interface is the best example of generics use in Java.
- We can create our own classes with generics type. A generic type can be a class or interface that is parameterized over types.
    - Syntax: use angle brackets (<>) to specify the type parameter
- Sometimes we don't want whole class to be parameterized, in that case we can create java generics method.
- The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:
    - T - Type
    - E - Element
    - K - Key
    - N - Number
    - V – Value

Scenario:

Professor Shim wants to takes submission in different order at different times. Sometimes he takes in alphabetical order of names and some other time with the ascending order of student id.

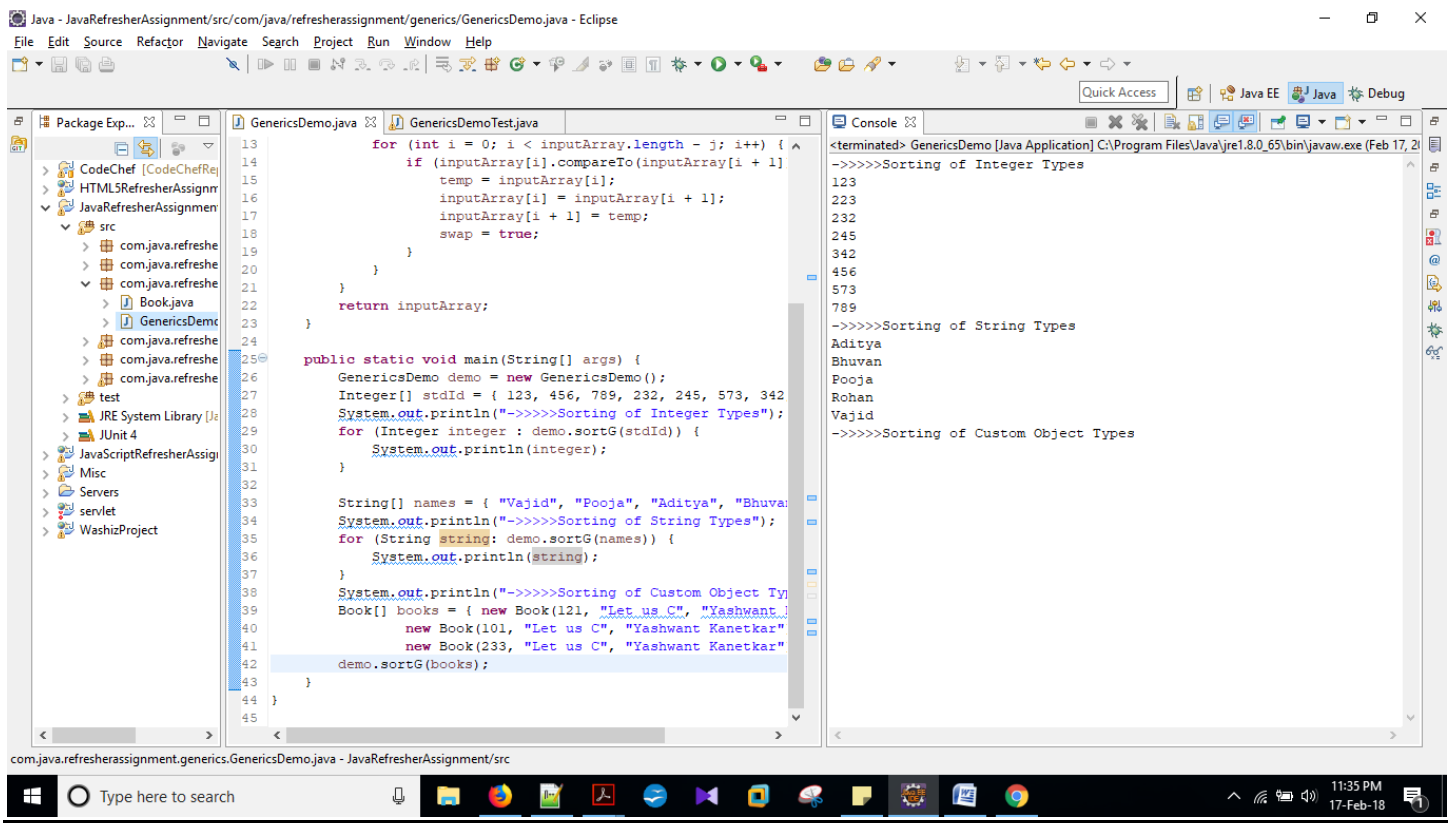
Your task is to create program that sorts the input which can be name or number and provide the order using Generics.

Source Class:

```
package com.java.refresherassignment.generics;

import com.java.refresherassignment.queue.Book;

public class GenericsDemo {

	// Use of generics method to sort Array of any type Including Object Type.
	public <E extends Comparable<E>> E[] sortG(E[] inputArray) {
		E temp;
		boolean swap = true;
		for (int j = 1; j < inputArray.length & swap; j++) {
			swap = false;
			for (int i = 0; i < inputArray.length - j; i++) {
				if (inputArray[i].compareTo(inputArray[i + 1]) > 0) {
					temp = inputArray[i];
					inputArray[i] = inputArray[i + 1];
					inputArray[i + 1] = temp;
					swap = true;
				}
			}
		}
		return inputArray;
	}
```

```java
        public static void main(String[] args) {
                GenericsDemo demo = new GenericsDemo();
                Integer[] stdId = { 123, 456, 789, 232, 245, 573, 342, 223 };
                System.out.println("->>>>>Sorting of Integer Types");
                for (Integer integer : demo.sortG(stdId)) {
                        System.out.println(integer);
                }

                String[] names = { "Vajid", "Pooja", "Aditya", "Bhuvan", "Rohan" };
                System.out.println("->>>>>Sorting of String Types");
                for (String string: demo.sortG(names)) {
                        System.out.println(string);
                }
                System.out.println("->>>>>Sorting of Custom Object Types");
                Book[] books = { new Book(121, "Let us C", "Yashwant Kanetkar"),
                                 new Book(101, "Let us C", "Yashwant Kanetkar"),
                                 new Book(233, "Let us C", "Yashwant Kanetkar") };
                demo.sortG(books);
        }
}
```

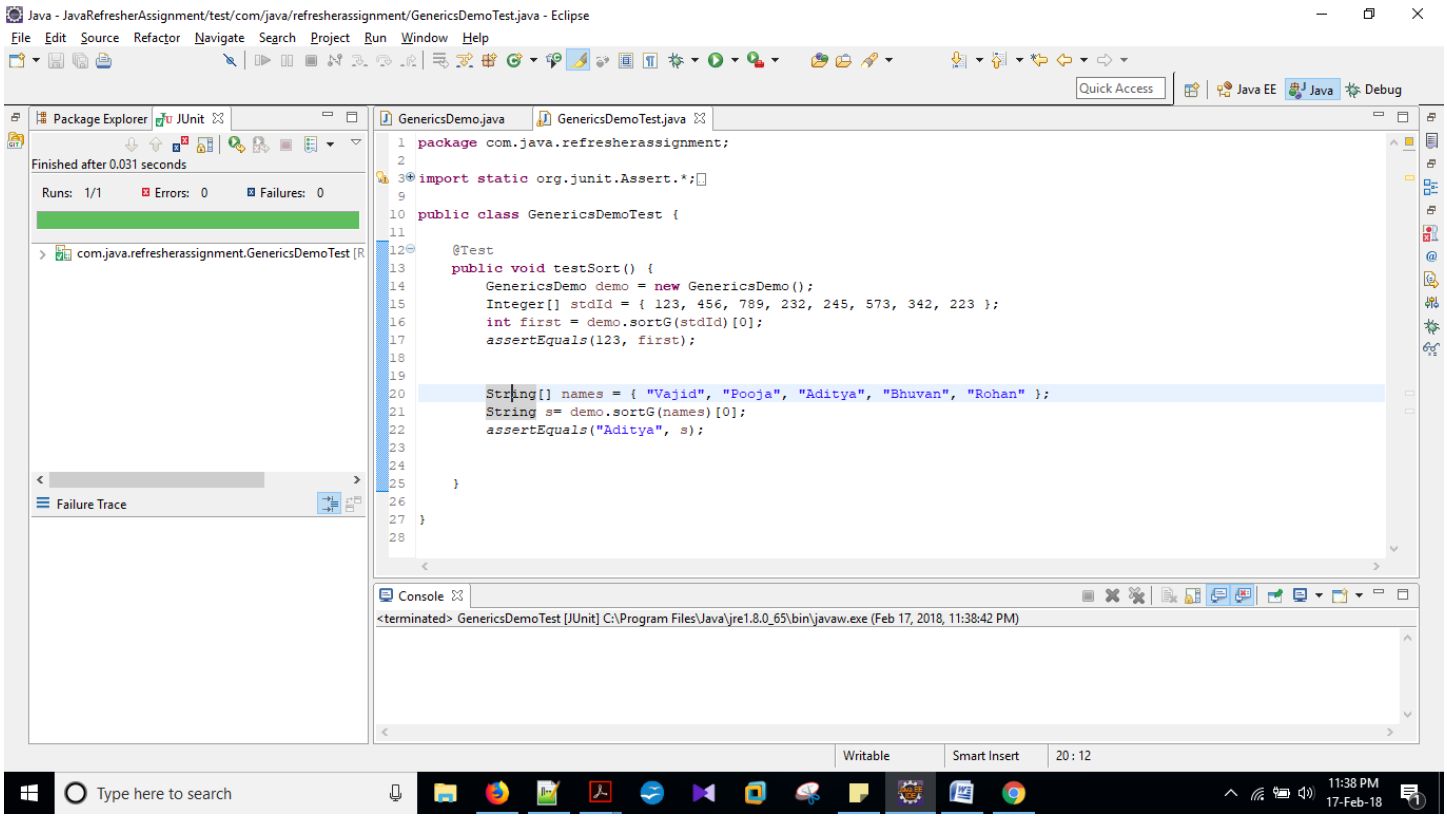
Source Class Run:



Test Class:

```
package com.java.refresherassignment;


import static org.junit.Assert.*;


import org.junit.Test;


import com.java.refresherassignment.generics.GenericsDemo;
import com.java.refresherassignment.queue.Book;


public class GenericsDemoTest {


	@Test
	public void testSort() {
		GenericsDemo demo = new GenericsDemo();
		Integer[] stdId = { 123, 456, 789, 232, 245, 573, 342, 223 };
		int first = demo.sortG(stdId)[0];
		assertEquals(123, first);


		String[] names = { "Vajid", "Pooja", "Aditya", "Bhuvan", "Rohan" };
		String s = demo.sortG(names)[0];
		assertEquals("Aditya", s);


	}


}
```

Test Class Run:



Multithreading

Introdution:

- It is a process of executing multiple threads at a time.
- A thread is a light weight sub process.
- Each thread is independent of other thread. If exception occurs in one thread it doesn't not affect other running threads.
- All threads share a common memory area.
    - The life cycle of the thread in java is controlled by JVM. Thread states can be : New, Runnable, Running, Terminated, Blocked.
- There are two ways to create threads in Java:
    - By extending Thread class
    - B y implementing Runnable interface.
- Thread class provide constructors and methods to create and perform operations on a thread.
    - **run():** is used to perform action for a thread.
    - **start()** method of Thread class is used to start a newly created thread
    - **sleep()** method of Thread class is used to sleep a thread
    - **join()** method waits for a thread to die
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Scenario:

Bank of the West allows to create a joint account for couples and friends. For this they want to design a system such that two persons can operate the account at the same time. Your task is to create the system which is capable of processing multiple transaction at time and transactions should be correctly maintained.

Design a system using thread and synchronization which will satisfy above requirement.

Source Class:

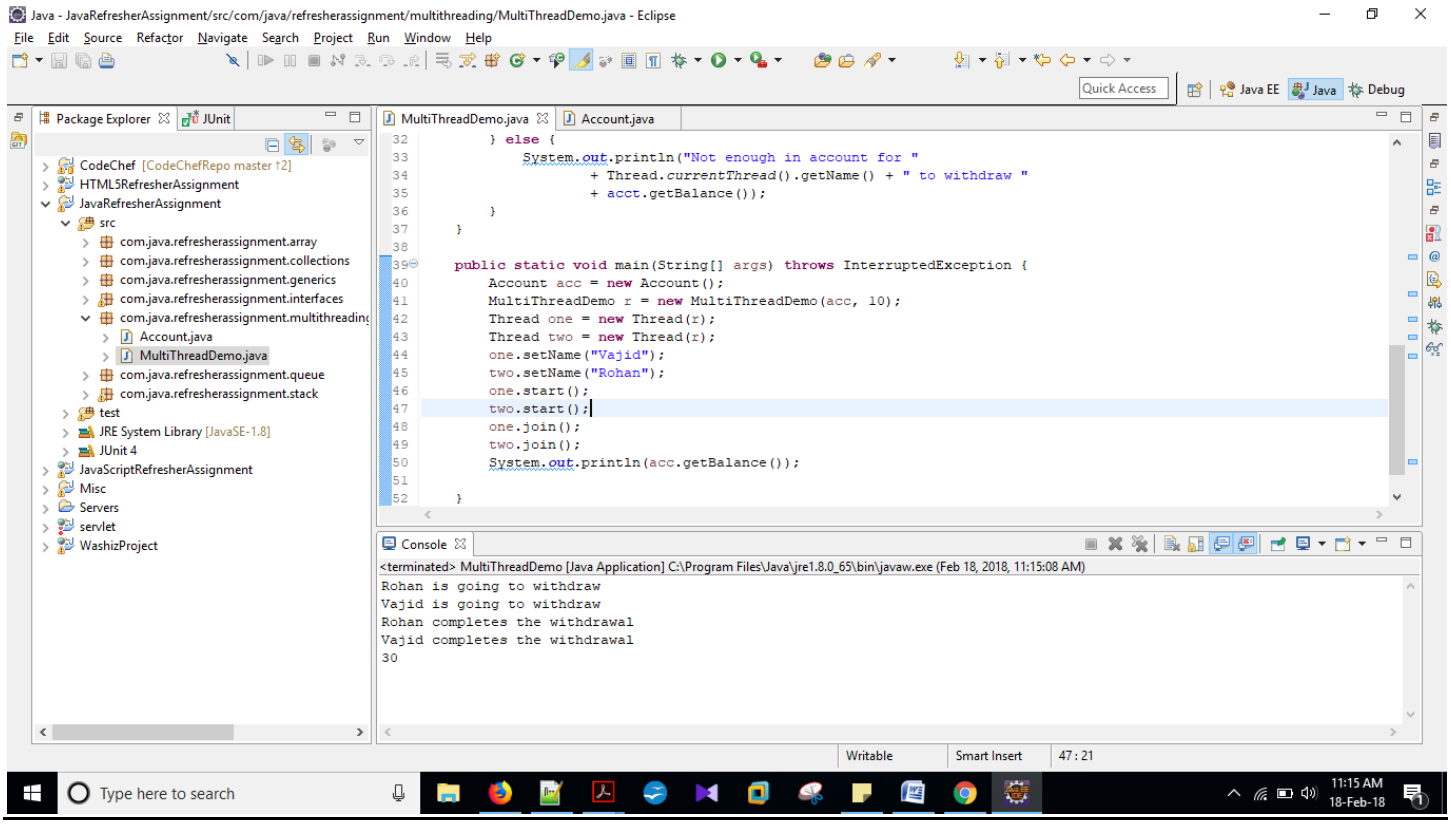
Account.Java

```
package com.java.refresherassignment.multithreading;

public class Account {
	private int balance = 50;

	public int getBalance() {
		return balance;
	}

	public void withdraw(int amountToBeWithdrawn2) {
		balance = balance - amountToBeWithdrawn2;
	}
}
```

MultiThreadDemo.java

```java
package com.java.refresherassignment.multithreading;

public class MultiThreadDemo implements Runnable {
	private Account acct;
	private int amountToBeWithdrawn;

	public MultiThreadDemo(Account acct, int amountToBeWithdrawn) {
		super();
		this.acct = acct;
		this.amountToBeWithdrawn = amountToBeWithdrawn;
	}

	@Override
	public void run() {
		makeWithdrawal(amountToBeWithdrawn);
		if (acct.getBalance() < 0) {
			System.out.println("Not sufficient Funds!");
		}
	}

	private void makeWithdrawal(int amountToBeWithdrawn2) {
		if (acct.getBalance() >= amountToBeWithdrawn2) {
			System.out.println(Thread.currentThread().getName()
				+ " is going to withdraw");
			try {
				Thread.sleep(100);
			} catch (InterruptedException ex) {
			}
			acct.withdraw(amountToBeWithdrawn2);
			System.out.println(Thread.currentThread().getName()
				+ " completes the withdrawal");
		} else {
			System.out.println("Not enough in account for "
				+ Thread.currentThread().getName() + " to withdraw "
				+ acct.getBalance());
		}
	}

	public static void main(String[] args) throws InterruptedException {
		Account acc = new Account();
		MultiThreadDemo r = new MultiThreadDemo(acc, 10);
		Thread one = new Thread(r);
		Thread two = new Thread(r);
		one.setName("Vajid");
		two.setName("Rohan");
		one.start();
		two.start();
		one.join();
		two.join();
		System.out.println(acc.getBalance());

	}
}
```

Source Class Run:



Test Class:

```
package com.java.refresherassignment;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

import com.java.refresherassignment.multithreading.Account;
import com.java.refresherassignment.multithreading.MultiThreadDemo;

public class MultiThreadDemoTest {

	@SuppressWarnings("deprecation")
	@Test
	public void testMultiThreadDemo() throws InterruptedException {

		Account acc = new Account();
		MultiThreadDemo r = new MultiThreadDemo(acc, 10);
		Thread one = new Thread(r);
		Thread two = new Thread(r);
		one.setName("Vajid");
		two.setName("Rohan");
		one.start();
		two.start();
		one.join();
		two.join();
		int expexted = 30;
		assertEquals("After 2 Withdrawel balance should be 30", expexted,
				acc.getBalance());
```

```
        Thread three = new Thread(r);
        Thread four = new Thread(r);
        three.setName("Shrey");
        four.setName("Pooja");
        three.start();
        four.start();
        three.join();
        four.join();
        expexted = 10;
        assertEquals("After 4 Withdrawel balance should be 10", expexted,
                acc.getBalance());

    }

}
```

Test Class Run: