CmpE 273 – Enterprise Ditributed Systems

**REFRESHER ASSIGNMENT**

**Submitted by:** Shreyam Kela (013775411)

<[shreyamkela@gmail.com](mailto:shreyamkela@gmail.com)>

**CONTENTS:**

|  |  |
| --- | --- |
| **Concept** | **Page Number** |
|  | |
| **JAVASCRIPT** | |
| Functions | 3 |
| Events | 4 |
| Arrays | 6 |
| Regular Expressions | 7 |
| Strict mode | 9 |
| Errors | 10 |
| Default Params | 11 |
| Includes and typeof | 12 |
| Use of import and export | 13 |
| Type Conversions | 14 |
| JSON | 15 |
| Object and Classes | 16 |
| Object.assign | 18 |
| Static method | 19 |
| Inheritance using sub-classes | 21 |
| Method overriding | 23 |
| Use of get | 24 |
| fetch() | 25 |
|  | |
| **HTML5** | |
| Local Storage | 26 |
| Media | 28 |
| Input Type | 29 |
| Geolocation | 31 |
|  | |
| **JAVA** | |
| Queues | 35 |
| Stacks | 37 |
| Arrays | 39 |
| Interfaces | 41 |
| Collections | 44 |
| Generics | 46 |
| Multithreading | 49 |

**JAVASCRIPT**

**Functions:**

* **Introduction:** A Function contains a set of instructions inside a block of code. According to the requirement, the function can be run arbitrarily to achieve a result. Functions are generally used as helpers for larger blocks of code in the program, which provides a set of functionalities, wherein the function is used and reused.
* **Programming Question:** Write a program of a function for a Travel Agency which lets the user enter a vehicle name and query whether the vehicle is available or not.
* **Code:**

let vehicles = {

"SUV" : 10,

"Sedan" : 27,

"Luxury" : 5

};

function travelAgency(vehicleType) {

if (vehicles[vehicleType] != undefined) {

console.log(`We provide ${vehicles[vehicleType]} vehicles of the type ${vehicleType}.`);

} else {

console.log('Sorry. We only provide the following types of vehicle:');

for (var key in vehicles) {

console.log(key);

}

}

}

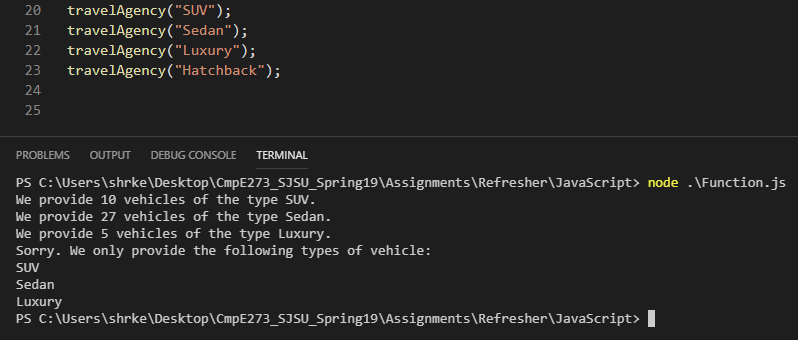
travelAgency("SUV");

travelAgency("Sedan");

travelAgency("Luxury");

travelAgency("Hatchback");

* **Output:**

****

**Events:**

* **Introduction:** Event is an occurrence that is the result of a user interacting with a webpage. The event could be a mouse click on a button or a key press at a particular place on the webpage. Interactions of the user are captured and a specific code block/function is triggered which handles that event.
* **Programming Question:** Write a program for a button that changes its own colour and alerts the user, when the user clicks on it. The colour should change to random colours on every click.
* **Code:**

<!DOCTYPE html>

<html>

<head>

<script>

function decimalToHex(decimal) {

hex = decimal.toString(16);

if (hex.length % 2) {

hex = '0' + hex;

}

return hex;

}

function randomColourHexValue() {

var decimalValue = Math.random() \* 255;

decimalValue = Math.floor(decimalValue);

var hexValue = decimalToHex(decimalValue);

return hexValue;

}

function setColor(button) {

var red = randomColourHexValue();

var green = randomColourHexValue();

var blue = randomColourHexValue();

var property = document.getElementById(button);

property.style.backgroundColor = `#${red}${blue}${green}`;

alert('Colour changed!');

}

</script>

</head>

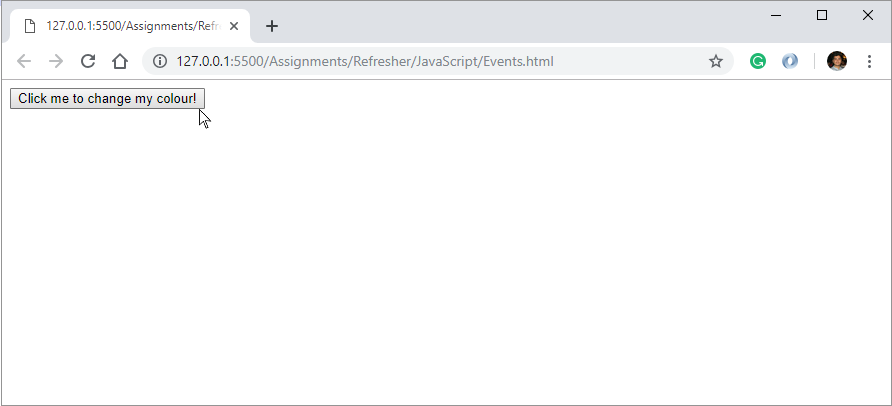
<body>

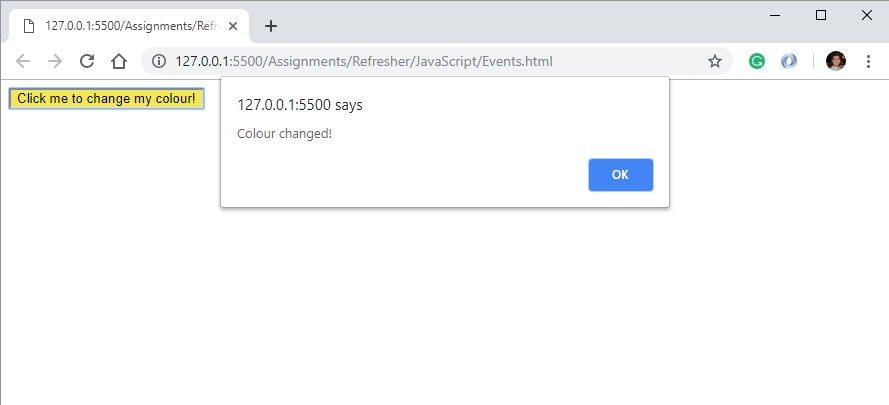
<input type="button" id="button" value = "Click me to change my colour!" style= "color : black" onclick="setColor('button')";/>

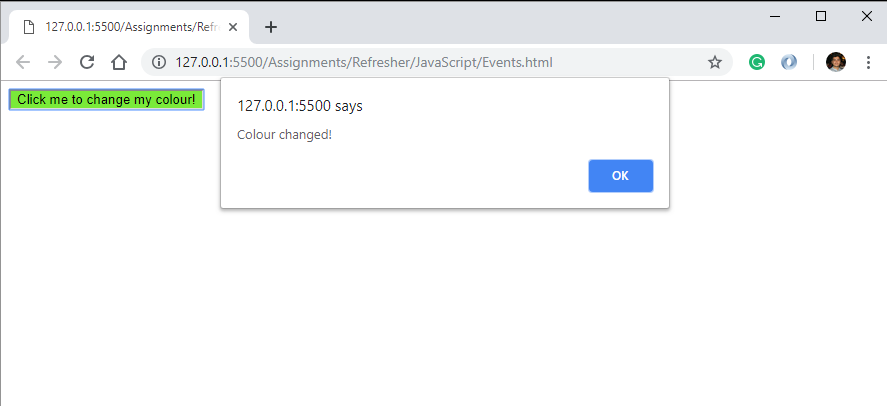
</body>

</html>

* **Output:**

****

****

****

**Arrays:**

* **Introduction:** Arrays are a type of data structure that store collections of data in a sequence. The data could be of the same type or different types. The data can be iterated through using indices and operated on, according to the requirement.
* **Programming Question:** Write a program to extract and separate the male and female candidates in an object “candidates” that contains the mixed data of the names and sex of the various candidates taking an exam. Store males and females in their respective arrays and output.
* **Code:**

let candidates = {

"John" : "M",

"Alexa" : "F",

"Bryan" : "M",

"Cindy" : "F",

"Roy" : "M",

"Evan" : "M"

};

let male = [];

let female = [];

for (key in candidates) {

if(candidates[key] == "M") {

male.push(key);

} else if (candidates[key] == "F") {

female.push(key);

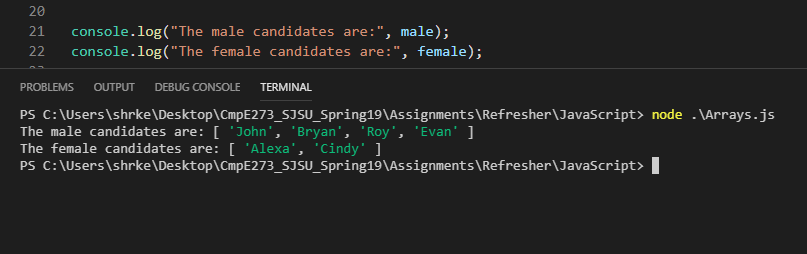
}

}

console.log("The male candidates are:", male);

console.log("The female candidates are:", female);

* **Output:**

****

**Regular Expressions:**

* **Introduction:** Regular expressions describe a set of strings based on patterns shared by each and every string in that set. The pattern can be used so as to determine whether another set of strings contains this pattern or not. This can be used to search or manipulate the data. There is a specific syntax to create regular expressions.
* **Programming Question:** Write a program that provides the user with an SJSU Edu email id when the user enters their personal email id. The program should also check whether the input email id is valid or not. Valid input email format – [dummy@gmail.com](mailto:dummy@gmail.com).

Example:

Input – [frank.junior@gmail.com](mailto:frank.junior@gmail.com)

Replace - @gmail.com => @sjsu.edu

Output – [frank.junior@sjsu.edu](mailto:frank.junior@sjsu.edu)

* **Code:**

function emailValidity(emailId) {

let checkEmail = emailId.match(/\S+@\S+/);

let validity = checkEmail? 'valid' : 'invalid';

console.log(`The email id is ${validity}`);

return validity;

}

function convertToEdu(emailId = 'xyz') {

if(emailValidity(emailId) == 'valid') {

let regex = '[^@]\*$';

let convertThis = emailId.match(regex);

let eduId = emailId.replace(convertThis[0],'sjsu.edu');

console.log(`New SJSU Edu ID for the student is ${eduId}`);

} else {

console.log('Please enter a valid email id.');

}

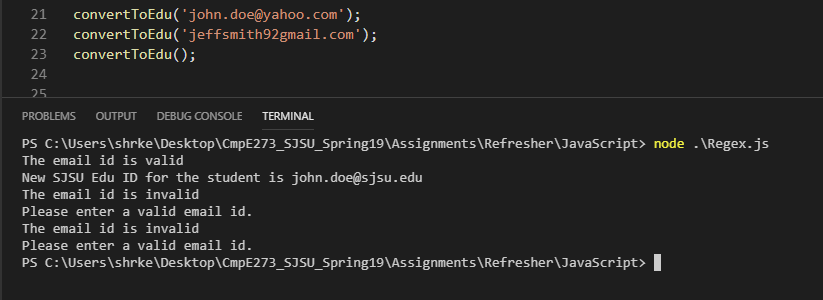
}

convertToEdu('john.doe@yahoo.com');

convertToEdu('jeffsmith92gmail.com');

convertToEdu();

* **Output:**

****

**Strict mode:**

* **Introduction:** Strict mode is a mode which restricts the use of a few functionalities in the program that would have been perfectly fine when used without the strict mode. It is an ES5 convention and it makes the code more secure. It also makes the programmer check for any unhandled exceptions or errors
* **Programming Question:** Write a program that tries changing the First Name linked to a particular Driving License Number. First Names and License numbers are stored in an object in the program. Run the program with and without the strict mode.
* **Code:**

"use strict";

let drivingLicense = {};

Object.defineProperty(drivingLicense, "license\_number", {value: 'A5e6U78p', writable: true});

Object.defineProperty(drivingLicense, "first\_name", {value: 'Harper', writable: false});

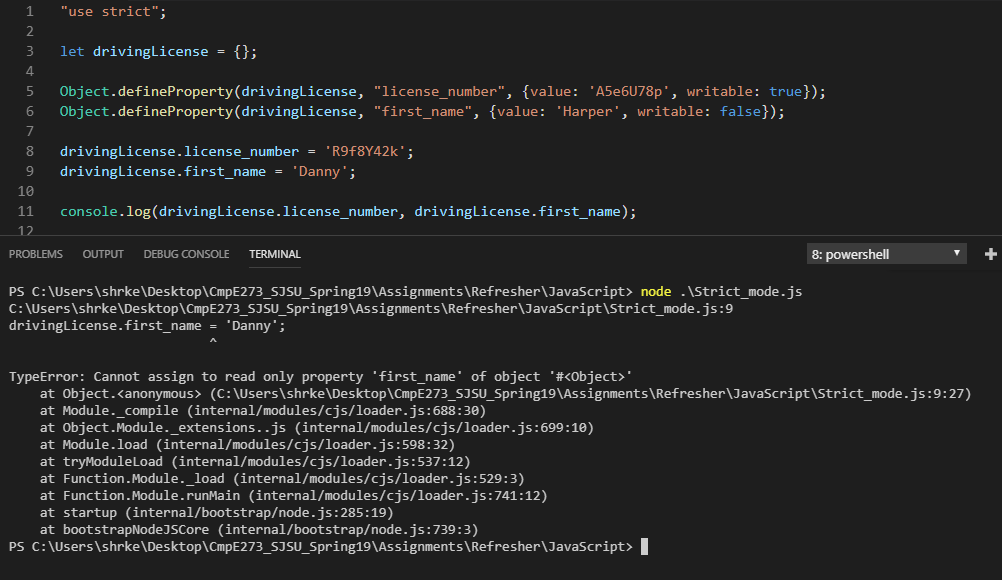
drivingLicense.license\_number = 'R9f8Y42k';

drivingLicense.first\_name = 'Danny';

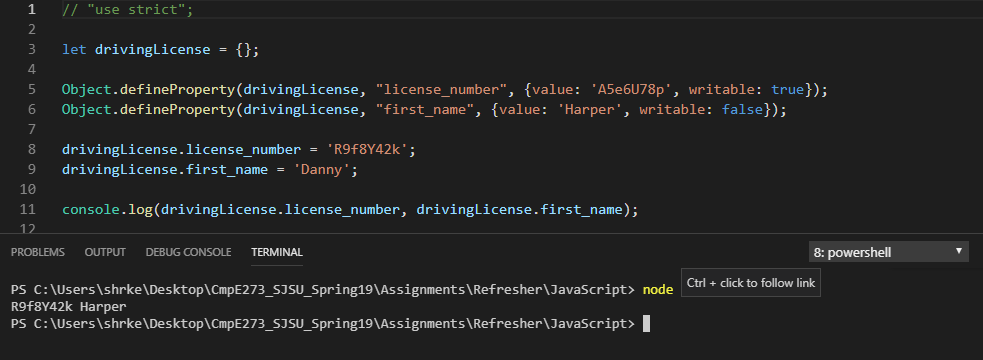
console.log(drivingLicense.license\_number, drivingLicense.first\_name);

* **Output:**

1. Strict Mode: ON (First Name unchanged and error is thrown)

****

1. Strict Mode: OFF (First Name still unchanged but no error is thrown – Dangerous, can cause future errors)

****

**Errors:**

* **Introduction:** Errors in JavaScript provide information about an error whenever the error occurs. Code has to be structured in a specific way so as to provide the corresponding error message correctly.
* **Programming Question:** Write a program that checks whether a particular username entered by the user is available or not. Error is thrown if the username is already taken.
* **Code:**

function checkAvailability(username) {

try {

if (usernameDatabase.includes(username)) {

throw "Sorry. The username already taken.";

} else {

usernameDatabase.push(username);

console.log(`Congratulations! The username is available.`);

}

}

catch (err) {

console.log(err);

}

finally {

console.log(`You have entered '${username}' as the username.`);

}

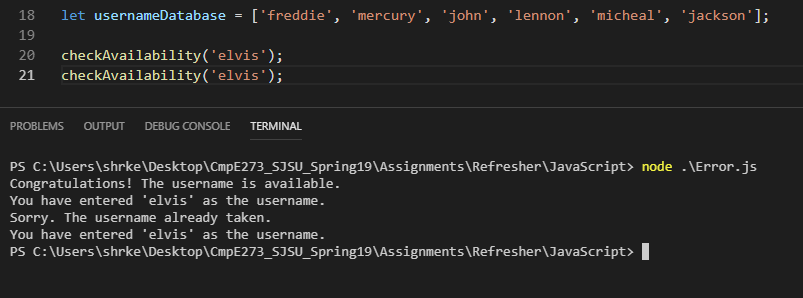
}

let usernameDatabase = ['freddie', 'mercury', 'john', 'lennon', 'micheal', 'jackson'];

checkAvailability('elvis');

checkAvailability('elvis');

* **Output:**

****

**Default Params:**

* **Introduction:** For JavaScript, when no value is passed as an argument into a function that takes parameters, the argument takes the value ‘undefined’ when inside the function. To check for this, we use default parameters and set default values to parameters, so that when no value is passed, the parameter takes the default value.
* **Programming Question:** Write a program that inputs employee name and position for a company, and stores it in an “employee” object. If user enters only the name input and not the position, use “Employee” as the default position.
* **Code:**

let employeeDatabase = [];

function company(employeeName, position = 'Employee') {

let employee = {

name : employeeName,

position : position

};

employeeDatabase.push(employee);

}

let printEmployees = () => {

for (index in employeeDatabase) {

let thisName = employeeDatabase[index].name;

let thisPosition = employeeDatabase[index].position;

console.log(`${thisName} - ${thisPosition}`);

}

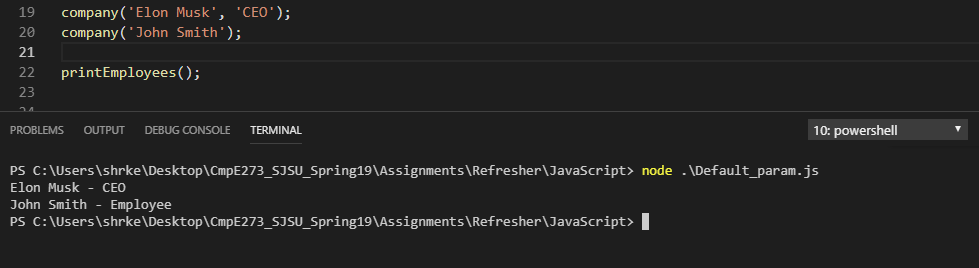
};

company('Elon Musk', 'CEO');

company('John Smith');

printEmployees();

* **Output:**

****

**Includes and typeof:**

* **Introduction:** Includes method checks whether a data structure includes a particular data or not. It also checks whether a given string contains a particular substring.

Typeof operator provides us with the data type of the variable queried with typeof.

* **Programming Question:** Write a program to that lets the user register a license number into the License Registration database. The program checks the data type of input and only lets “number” licenses to be registered. The program also checks whether a particular license number is already taken.
* **Code:**

let licenseNumbers = [];

let licenseRegistration = (number) => {

if(typeof(number) != "number") {

console.log(`The entered value ${number} cannot be registered as a valid license number.`);

} else {

if(licenseNumbers.includes(number)) {

console.log(`Sorry. The license number ${number} is already taken. Choose a new license number.`);

} else {

licenseNumbers.push(number);

console.log(`Congratulations! License number ${number} registered.`);

}

}

}

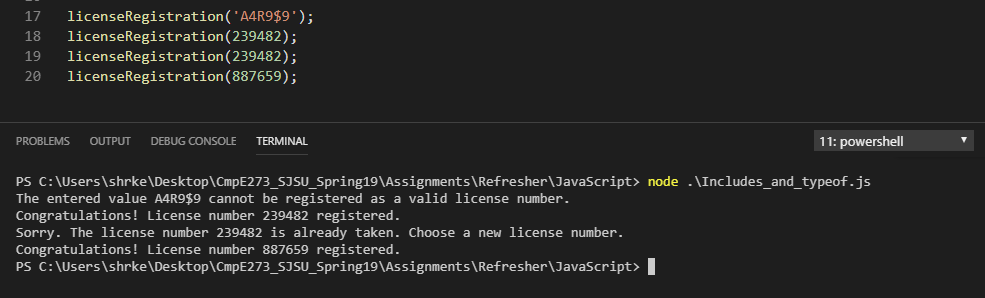
licenseRegistration('A4R9$9');

licenseRegistration(239482);

licenseRegistration(239482);

licenseRegistration(887659);

* **Output:**

****

**Use of require and export:**

* **Introduction:** Export lets us export functions to another file or program. The exported functions can be used in the other program with the help of require. Modules exported from another file are linked to this file using require.
* **Programming Question:** Write a program that calculates the final GPA according the course grades and credits in 3 courses.
* **Code:**

1. Require – This code module imports the calculateGPA method and passes in the courseGrades and courseCredits as arguments to the method.

var GPA = require('./export.js');

let courseGrades = [4.0, 3.33, 3.66];

let courseCredits = [3.0, 3.0, 3.0];

console.log('GPA: ', GPA.calculateGPA(courseGrades, courseCredits));

1. Export - This code module defines the calculateGPA method and exports it.

module.exports.calculateGPA = (courseGrades, courseCredits) => {

var totalGradePoints = 0;

var totalCredits = 0;

for(index in courseGrades) {

let thisCourseGrade = courseGrades[index];

let thisCourseCredit = courseCredits[index];

let courseGradePoint = thisCourseGrade\*thisCourseCredit;

totalGradePoints += courseGradePoint;

totalCredits += thisCourseCredit;

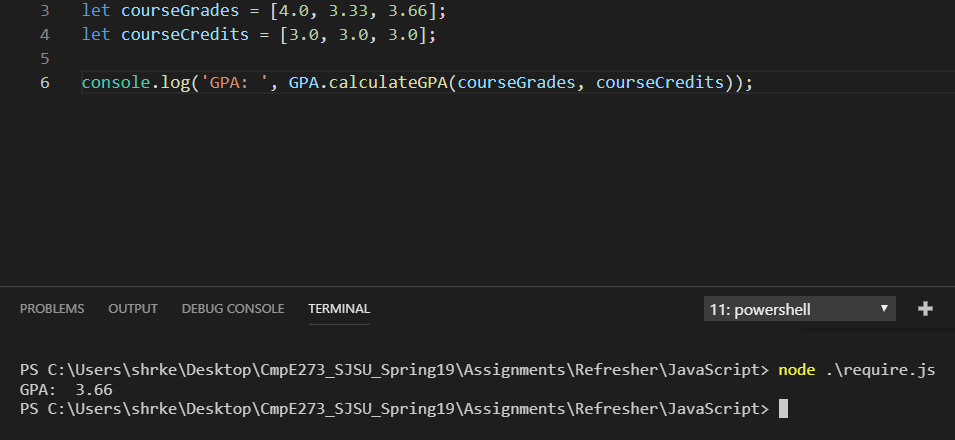
}

var finalGPA = totalGradePoints/totalCredits;

return Math.round(finalGPA \* 100) / 100;

};

* **Output:**

****

**Type Conversions:**

* **Introduction:** Type conversion functionality converts the data type of a particular variable to another data type, as required.
* **Programming Question:** Write a program to input First name, Last name, and birth year and generate a username for the user. Input first and last names as “string” and birth year as an “integer”. Generate username as (First name) + (Initial of Last name) + (Last 2 digits of the birth year).

Example:

Input: “Bruce”, “Wayne”, 1964

Output: BruceW64

* **Code:**

let generateUsername = (firstName, lastName, birthYear) => {

let birthYearString = String(birthYear);

let lastNameInitial = lastName[0];

let birthYearLastTwoDigit = birthYearString.substr(birthYearString.length-2);

let username = firstName + lastNameInitial + birthYearLastTwoDigit;

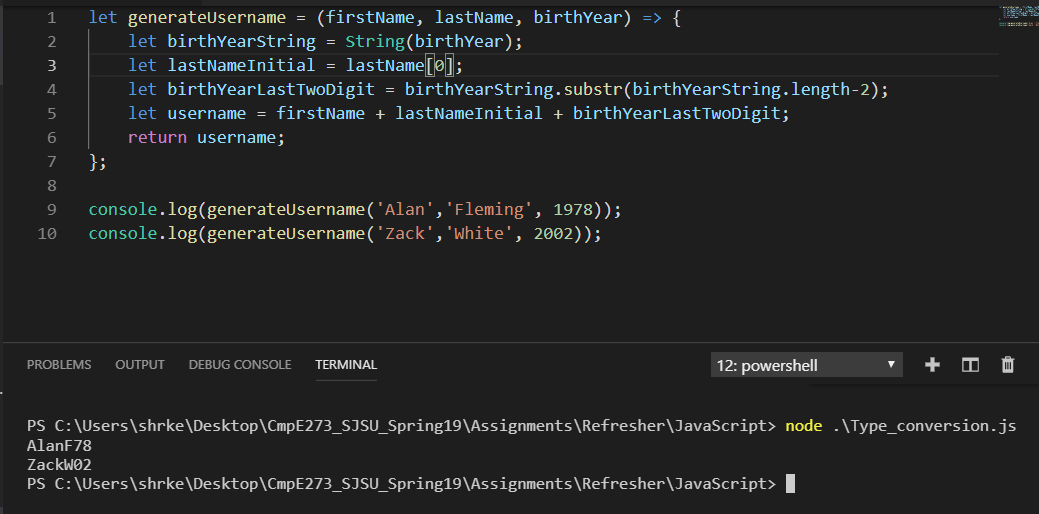
return username;

};

console.log(generateUsername('Alan','Fleming', 1978));

console.log(generateUsername('Zack','White', 2002));

* **Output:**

****

**JSON:**

* **Introduction:** JSON orJavaScript Object Notation is used for exchanging data JSON. It is not dependent on the language and its format is textual.

It has 2 main functions:

a) JSON.stringify() – JSON Object to JSON text/string.

b) JSON.parse() – JSON String to JSON object.

* **Programming Question:**

Write a program to send an object containing SJSU courses class strength as a JSON string, and parse it to back to an object. Then print the parsed object.

* **Code:**

var classStrength = [

{"Cmpe273" : 86},

{"Cmpe275" : 64},

{"Cmpe285" : 66}

];

console.log("Stringifying the class strength object");

var jsonString = JSON.stringify(classStrength);

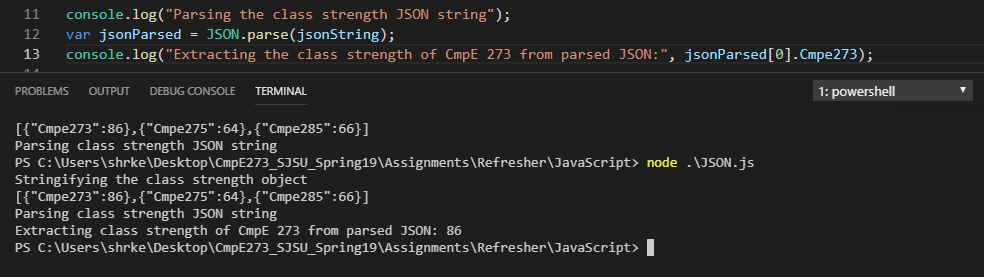
console.log(jsonString);

console.log("Parsing the class strength JSON string");

var jsonParsed = JSON.parse(jsonString);

console.log("Extracting the class strength of CmpE 273 from parsed JSON:", jsonParsed[0].Cmpe273);

* **Output:**

****

**Object and Classes:**

* **Introduction**: A JavaScript object is different from an object in other languages. An object in JavaScript is a variable which has sets of name-value pairs. Classes are blueprint data structures from which instances can be generated with the same properties.
* **Programming Question:**

1. Class: Write a program to make a Transport class which is a class for the different modes of transport. Instantiate it to create different instances as a Car and an Aeroplane.
2. Object: Write a program to make an organism object and add a property to it on the fly.

* **Code:**

1. Class:

class Transport {

constructor(type, name, seatingCapacity) {

this.type = type;

this.name = name;

this.seatingCapacity = seatingCapacity;

}

get myType() {

return this.type;

}

get myName() {

return this.name;

}

get myCapacity() {

return this.seatingCapacity;

}

}

const Car = new Transport('Car', 'Audi Q7', '5') ;

console.log(`Type: ${Car.myType}, Name: ${Car.myName}, Capacity: ${Car.myCapacity}`);

Transport.prototype.flyingAltitude = function (altitude) {

this.altitude = altitude;

return this.altitude;

}

const Aeroplane = new Transport('Aeroplane', 'Boeing 777', '400');

console.log(`Type: ${Aeroplane.myType}, Name: ${Aeroplane.myName}, Capacity: ${Aeroplane.myCapacity}, Altitude: ${Aeroplane.flyingAltitude('45000 ft')}`);

1. Object:

let organism = {

type: "Amphibian",

name: "Frog",

colour: "Green",

height: "3 inches",

weight: "1 pound",

count: 8456489

}

console.log(organism);

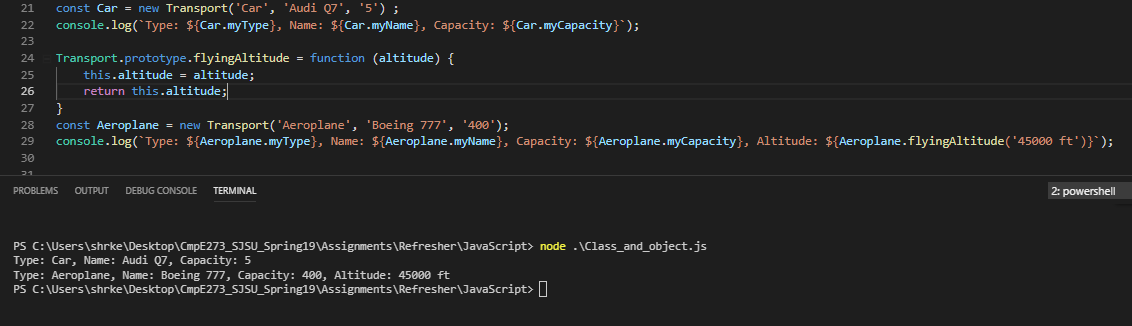
console.log('\nAdding a property to the object.\n');

organism.sound = "ribbit";

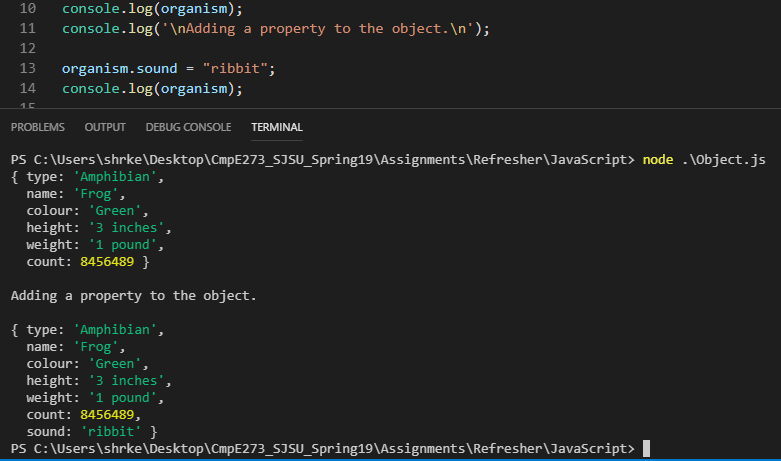
console.log(organism);

* **Output:**

1. Class:

****

1. Object:

****

**Object.assign:**

* **Introduction:**

Object.assign can be used to:

a) Copy an object.

b) Add a property to the object.

c) Merging multiple objects.

* **Programming Question:** Write a program to make objects for the rocky bodies Earth and Saturn. Use object.assign to write properties of Earth and Saturn to 2 other rocky bodies Mount Everest and Halley’s Comet.
* **Code:**

let saturn = {

type: "rocky body without life",

water: false,

};

let earth = {

type: "rocky body with life",

water: true,

starSystem: "Sun"

};

let mountEverest = Object.assign({height: "29000 ft"}, earth);

let halleysComet = Object.assign({water: false, tail: true}, earth, saturn);

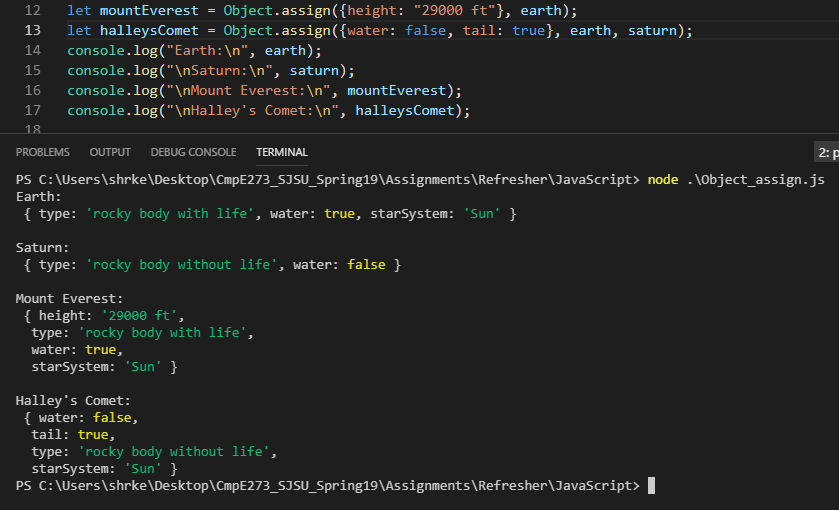
console.log("Earth:\n", earth);

console.log("\nSaturn:\n", saturn);

console.log("\nMount Everest:\n", mountEverest);

console.log("\nHalley's Comet:\n", halleysComet);

* **Output:**

****

**Static method:**

* **Introduction:** Static methods are class methods that can only be used as utility functions by the class itself and cannot be used by any instances of the class. To create a static method, append the method name with the keyword ‘static’.
* **Programming Question:** Write a program to create a student class. The class should have a static method that returns the count of total courses taken by a student. The class should have a non static method that prints all the courses taken.
* **Code:**

class student {

constructor(name, courseGrades) {

this.name = name;

this.courseGrades = courseGrades;

}

get myName() {

return this.name;

}

static totalCourses(courseGrades) {

var count = Object.keys(courseGrades).length;

return count;

}

get printTotalCourses() {

var courseCount = student.totalCourses(this.courseGrades);

return courseCount;

}

}

let courseGrades = {

"CmpE273" : 3.66,

"CmpE275" : 3.33,

"CmpE285" : 4.00

};

let alex = new student('Alex', courseGrades);

console.log(alex.myName);

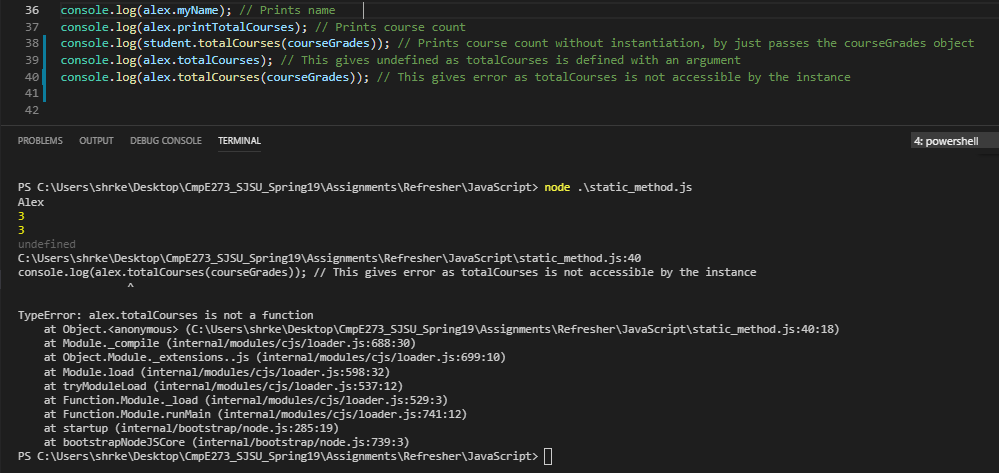
console.log(alex.printTotalCourses);

console.log(student.totalCourses(courseGrades));

console.log(alex.totalCourses);

console.log(alex.totalCourses(courseGrades));

* **Output:**

****

**Inheritance using sub-classes in JavaScript.:**

* **Introduction:** Inheritance is a programming concept with the help of which classes can inherit properties from other classes so we don’t have to write the code again. This also increases the efficiency and the readability of the code. To inherit a class, child classes use keywords “extends” or “super”.
* **Programming Question:** Write a program to make a refrigerator sub class that extends an appliance super class. The refrigerator class inherits all the properties and method of the appliance class and also adds a few other methods.
* **Code:**

class appliance {

constructor (name, price, manufacturedIn, size, type) {

this.type = type;

this.name = name;

this.price = price;

this.manufacturedIn = manufacturedIn;

this.size = size;

}

get myName() {

console.log(this.name);

}

get myType() {

return this.type;

}

totalCost(quantity) {

return this.price\*quantity;

}

}

class refrigerator extends appliance {

constructor(name, price, manufacturedIn, doorCount, capacity) {

var undef = undefined;

super(name, price, manufacturedIn);

this.doorCount = doorCount;

this.capacity = capacity;

this.item = [];

}

get myName() {

return (this.name).toUpperCase();

}

freeze(item) {

(this.item).push(item);

}

get fridgeItems() {

return this.item;

}

}

let fridgeSuperClass = new appliance("Refrigerator", "Whirlpool", 400, "Finland", "250 litre");

console.log("Super Class instance \"Refrigerator\":");

console.log("Type:", fridgeSuperClass.myType);

fridgeSuperClass.myName;

console.log("Size:", fridgeSuperClass.size);

console.log("Total Cost for 5 units:", fridgeSuperClass.totalCost(5));

let fridgeSubClass = new refrigerator("Bosch", 650, "Korea", 2, "400 litre");

fridgeSubClass.freeze("Apples"); // Freeze an item in the fridgeSubClass

fridgeSubClass.freeze("Cilantro");

console.log("\nSub Class instance \"Refrigerator\":");

console.log("Type:", fridgeSubClass.myType); // undefined as myType not defined for sub class

console.log("Name:", fridgeSubClass.myName);

console.log("Price:", fridgeSubClass.price);

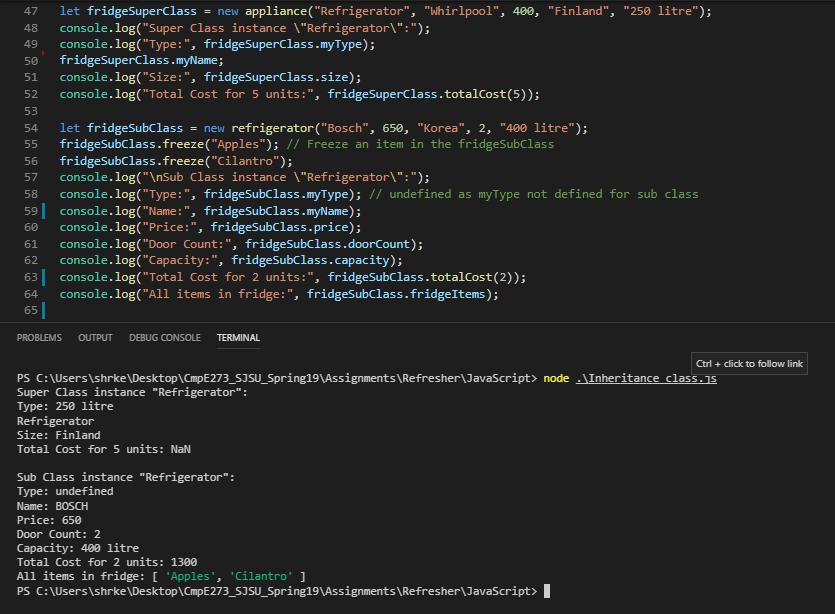
console.log("Door Count:", fridgeSubClass.doorCount);

console.log("Capacity:", fridgeSubClass.capacity);

console.log("Total Cost for 2 units:", fridgeSubClass.totalCost(2));

console.log("All items in fridge:", fridgeSubClass.fridgeItems);

* **Output:**

****

**Method overriding:**

* **Introduction:** A function can use a method of another function by overriding it. The name of the method remains the same but the functionality can change. Methods can be overridden using the prototype property.
* **Programming Question:** Write a function to make a give() and a take() functions that give and take money, respectively. The give function prototypes a transfer method which is responsible for giving the money. The take functions prototypes the same function by overriding it, and it takes money.
* **Code:**

function give() {

}

function take() {

}

give.prototype.transfer = function () { // Add methods to give() using prototype

console.log("I'm giving the money!");

};

take.prototype = Object.create(give.prototype);

take.prototype.transfer = function () {

console.log("I'm taking the money!");

};

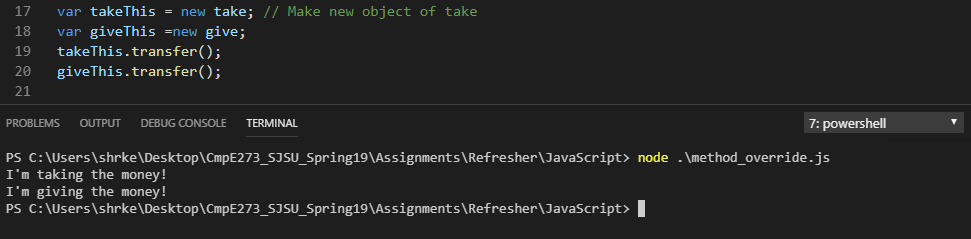
var takeThis = new take; // Make new object of take

var giveThis =new give;

takeThis.transfer();

giveThis.transfer();

* **Output:**

****

**Use of get (In Classes):**

* **Introduction:** Methods inside a class that are used to retrieve an internal property or an attribute of that class are called getter methods. To make a getter method, we append at the start of the function name, the word “get”.
* **Programming Question:** Write a program to make a class “cloth”. Instantiate the class with a kind of cloth and get its properties.
* **Code:**

class cloth {

constructor (texture, company, cost) {

this.texture = texture;

this.company = company;

this.cost = cost;

}

get myTexture() {

return this.texture;

}

get myCompany() {

return this.company;

}

get myCost() {

return this.cost;

}

}

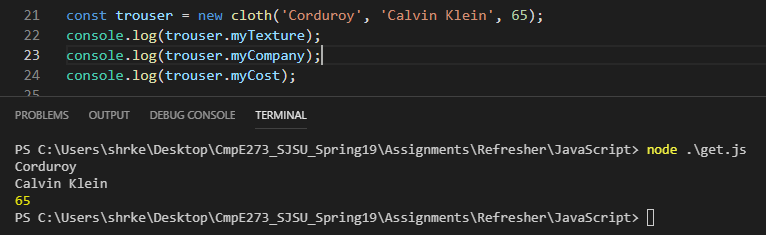
const trouser = new cloth('Corduroy', 'Calvin Klein', 65);

console.log(trouser.myTexture);

console.log(trouser.myCompany);

console.log(trouser.myCost);

* **Output:**

****

**fetch() ( Use any open-source API for fetching data):**

* **Introduction:** Fetch is a method that provides an interface to get network data. Fetch is mostly used in web development. Fetch can be clubbed with promises to form asynchronous calls to APIs.
* **Programming Question:** Write a program to fetch the name of a book from an open source books API, when queried with the ISBN of the book.
* **Code:**

const fetch = require('node-fetch');

// Open Source Books API - www.googleapis.com/books/ -> query ISBN 0747532699 (Harry Potter and the Philosopher's Stone)

function book() {

return fetch('https://www.googleapis.com/books/v1/volumes?q=isbn:0747532699')

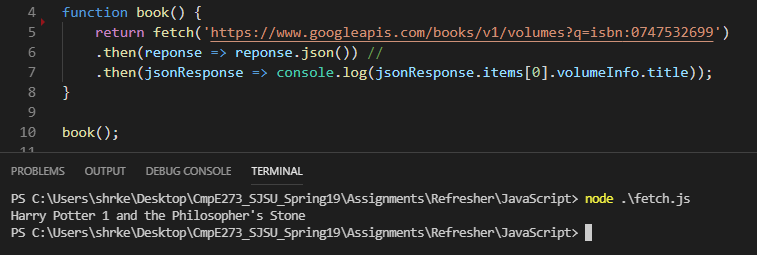
.then(reponse => reponse.json()) //

.then(jsonResponse => console.log(jsonResponse.items[0].volumeInfo.title));

}

book();

* **Output:**

****

**HTML5**

**Local Storage:**

* **Introduction:** All the modern browsers have local storage inside them, to store data on the client side. The data is not deleted even when the page is refreshed or the session changes. Methods can be used to set, manipulate, and delete data according to the requirement.
* **Programming Question:** Write a program to input First Name and Last Name of the user through a webpage, and store it into the local storage.
* **Code:**

<!DOCTYPE html>

<html>

<body>

First Name: <input type="text" id="firstname"><br>

Last Name: <input type="text" id="lastname"><br><br>

<button onclick="store()" type="button">Store</button><br><br>

<div id="result">Enter the details and click on Store.</div>

<script type="text/javascript">

localStorage.clear();

function store(){

if (typeof(Storage) !== "undefined") {

var inputFirstName = document.getElementById("firstname");

var inputLastName = document.getElementById("lastname");

localStorage.setItem("First name", inputFirstName.value);

localStorage.setItem("Last name", inputLastName.value);

document.getElementById("result").innerHTML = "Details stored!";

} else {

document.getElementById("result").innerHTML = "Sorry, your browser does not support local storage.";

}

}

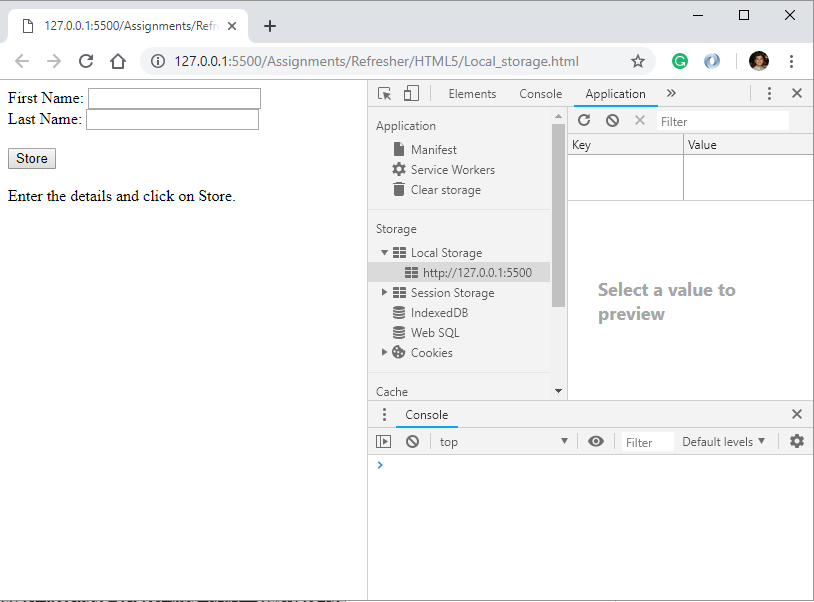
</script>

</body>

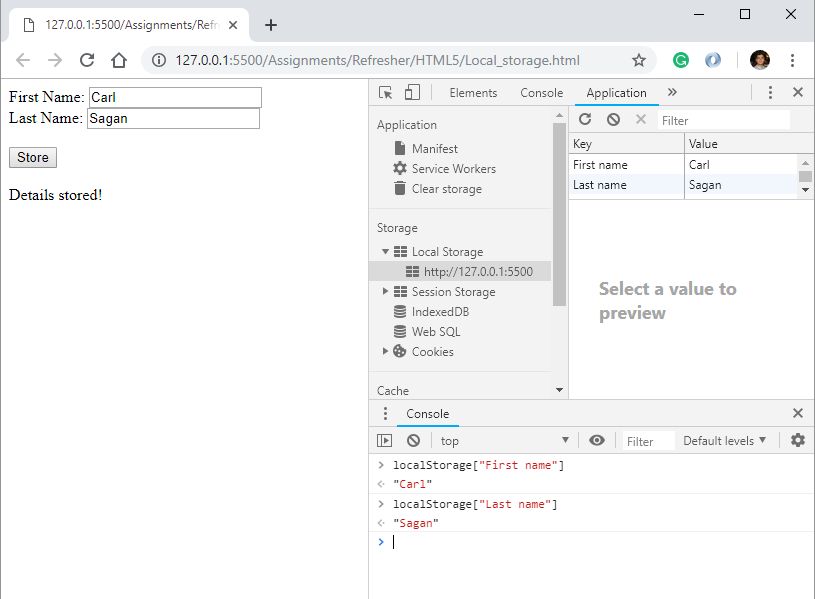
</html>

* **Output:**

1. Input the details

****

1. Store the details into the local storage.

****

**Media (Video and Audio):**

* **Introduction:** Modern browsers can display several types of media on the browser using predefined keywords of HTML5.
* **Programming Question:** Write a program to create a webpage that displays Windows logo video and also plays its audio.
* **Code:**

<!DOCTYPEhtml>

<html lang="en">

<head>

<title> Audio and Video </title>

</head>

<body>

<audio controls>

<source src = "WindowsAudio.mp3" type = "audio/mp3">

<source src = "WindowsAudio.ogg" type = "audio/ogg">

</audio>

<video width = "200" height = "200" controls>

<source src = "WindowsVideo.mp4" type = "video/mp4">

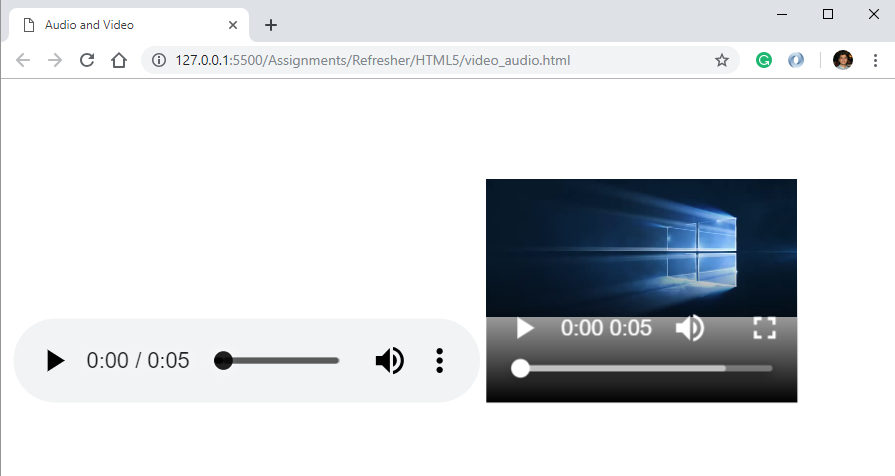
<source src = "WindowsVideo.ogg" type = "video/ogg">

</video>

</body>

</html>

* **Output:**

****

**Input Type** (make use of different input property options in HTML5 like patterns, autofocus, required, email etc. Place types you want, mention the properties used in your Introduction to Topic section):

* **Introduction:** HTML5 provides various different input type functionalities for adding standard fields and classifications to the webpage.

1. **’text’ input type:** To create a 20 character limit text box.
2. **required**: The required fields make the field mandatory.
3. **autofocus:** When the page is loaded, cursor is at this field.
4. **Pattern**: To mandate a pattern for a string using regular expressions.
5. **checked**: Keeps the field already checked.
6. **title**: Set the title to the prompt.
7. **autocomplete**: Auto completes the field when the user starts typing.
8. **email**: Validate the string entered for an email.
9. **checkbox**: Create checkboxes in the form.
10. **radio**: Create radio buttons in the form.
11. **reset**: Reset all the fields in the form
12. **submit**: Submit the form

* **Programming Question:** Write a program to create a form to input the University and student details from a user. The program should use the various input types.
* **Code:**

<!DOCTYPEhtml>

<html>

<head>

<title>University Details</title>

</head>

<body>

<form>

<h3>Current Attending University</h3>

<h4>Enter University Details</h4>

University Name:

<input type = "text" name = "uniname" autofocus required>

<br><br>

College Name:

<input type = "text" name = "colname">

<br><br>

Edu Id:

<input type = "email" name = "email" autocomplete = "on" required>

<br><br>

Address:

<input type = "text" name = "address">

<br><br>

Pin Code :

<input type = "text" name = "pincode" pattern = "[0-9]{5}" title = "Pincode should be of 5 digits">

<br><br><br>

Degree pursuing: <br>

<input type = "radio" name = "degree" value = "masters" checked> Masters <br>

<input type = "radio" name = "degree" value = "bachelors"> Bachelors <br>

<input type = "radio" name = "degree" value = "others"> Others

<br><br><br>

Current Job:<br>

<input type = "checkbox" name = "job1" value = "isa">Instructional Student Assistant<br>

<input type = "checkbox" name = "job2" value = "ga">Graduate Assistant<br>

<input type = "checkbox" name = "job3" value = "ra">Research Assistant<br>

<input type = "checkbox" name = "job4" value = "on">Other On-campus<br>

<br><br>

<input type = "submit"><br><br>

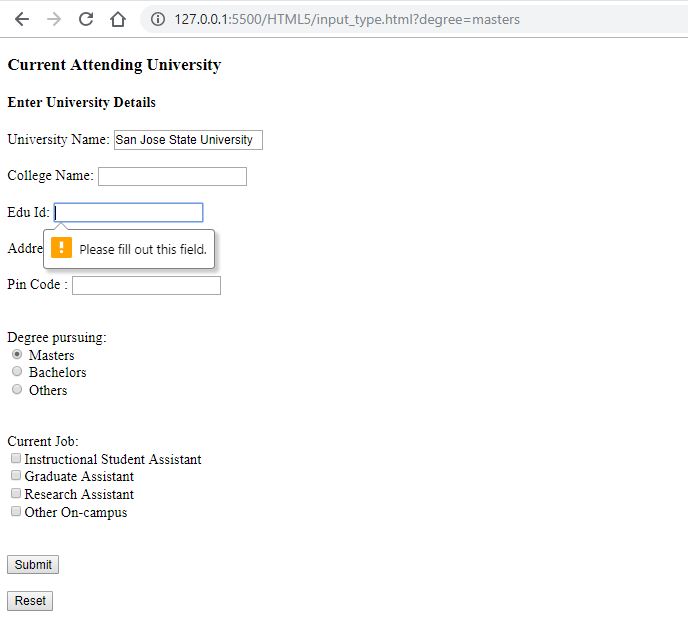
<input type = "reset">

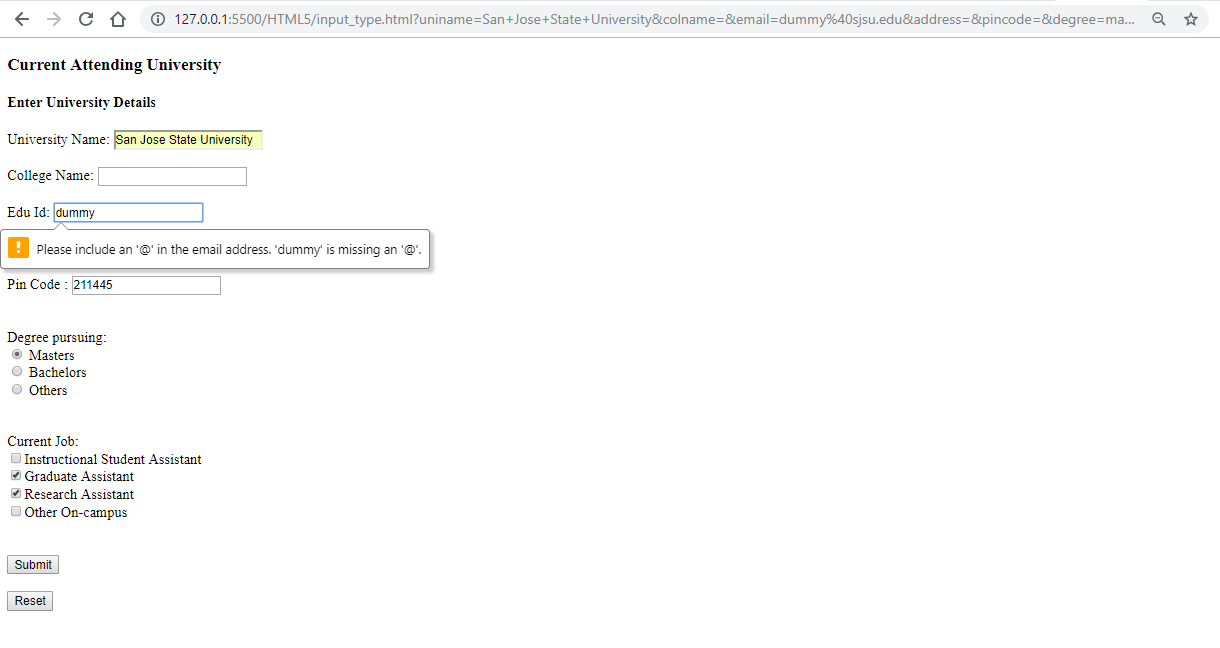
</form>

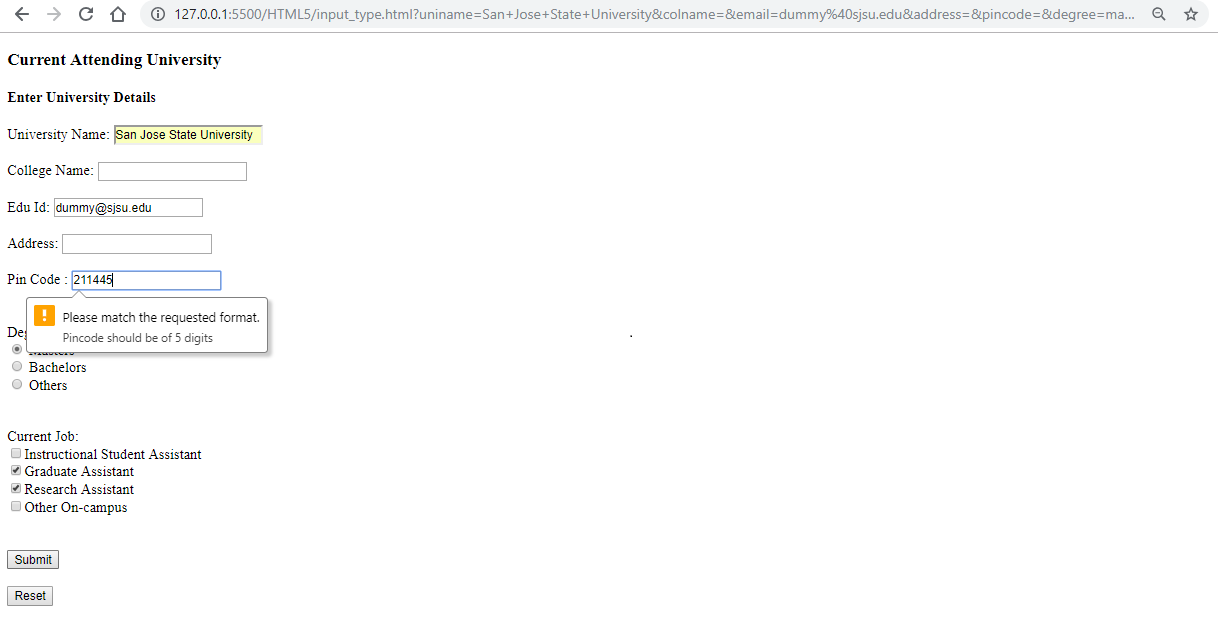
</body>

</html>

* **Output:**

****

****

****

**Geolocation:**

* **Introduction:** Geolocation is an API that is used to access the location of a user. Any program can use the functionality of geolocation to add the tracking feature in the webpages.
* **Programming Question:** Write a program to create a webpage that prompts the user for their location and if allowed, it prints the location and also stores it into the local storage.
* **Code:**

<!DOCTYPE html>

<html>

<body>

<p>Click to get the current coordinates.</p>

<button onclick = "getLocation()"> Click me! </button>

<p id="location">Location will print here...</p>

<p id="stored">Location will also be stored in the Local Storage.</p>

<script>

localStorage.clear();

var myLocation = document.getElementById("location");

function getLocation() {

if (!navigator.geolocation) {

myLocation.innerHTML = "Sorry, geolocation is not supported by your browser.";

} else {

navigator.geolocation.getCurrentPosition(position);

}

}

function position(myPosition) {

myLocation.innerHTML = "Latitude: " + myPosition.coords.latitude +

"<br>Longitude: " + myPosition.coords.longitude;

store(myPosition.coords.latitude, myPosition.coords.longitude);

}

function store(latitude, longitude) {

if (typeof(Storage) !== "undefined") {

localStorage.setItem("Latitude", latitude);

localStorage.setItem("Longitude", longitude);

document.getElementById("stored").innerHTML = "Location stored!";

} else {

document.getElementById("stored").innerHTML = "Sorry, your browser does not support local storage.";

}

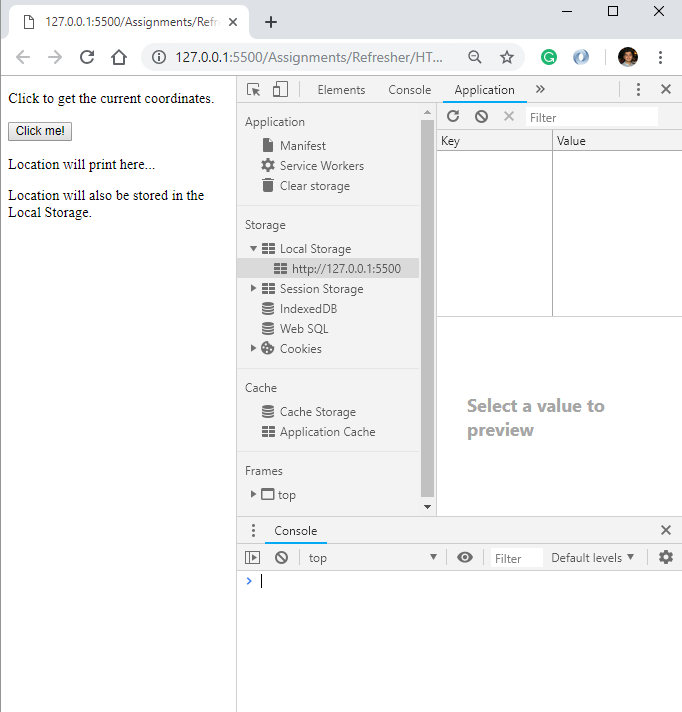
}

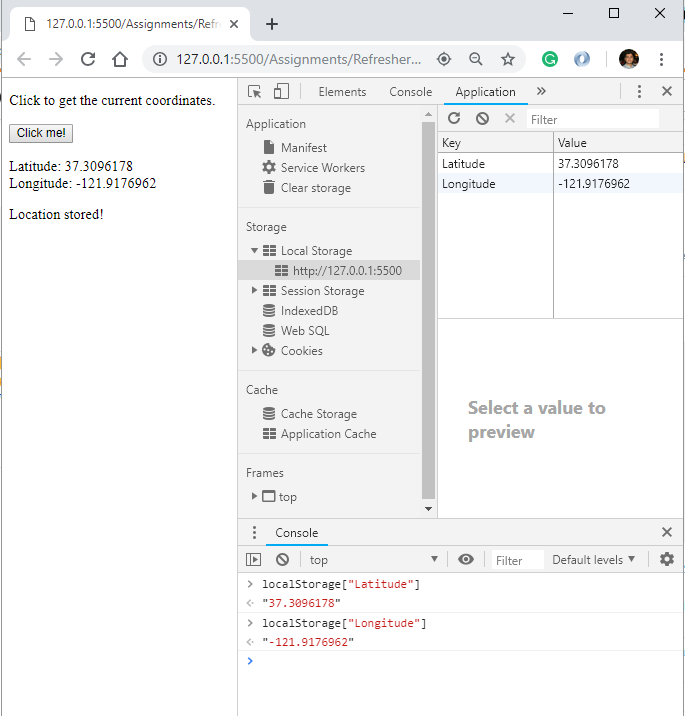
</script>

</body>

</html>

* **Output:**



****

**JAVA:**

(Use JUnit Framework for testing the application)

**Queues:**

* **Introduction:** Queue is a kind of data structure that is based on FIFO. The first entity in is the entity that gets out of the queue first. Queues can also be used when priority is being considered.
* **Programming Question:** Write a program to create a scheduler that schedules 4 tasks using a queue. Print the first task to be executed.
* **Code:**

**import** java.util.\*;

**public** **class** queues {

**public** **static** String schedular() {

Queue<String> taskQueue = **new** LinkedList<String>();

taskQueue.add("Google Chrome");

taskQueue.add("VLC player");

taskQueue.add("MS Word");

taskQueue.add("Visual Studio Code");

System.***out***.println("The tasks are: " + taskQueue);

System.***out***.println("Task with the most priority: " + taskQueue.peek());

**return** taskQueue.remove();

}

**public** **static** **void** main(String args[]) {

String priorityTask = queues.*schedular*();

System.***out***.println("Task to be executed first: " + priorityTask);

}

}

**JUnit Test:**

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**class** queuesTest {

@Test

**void** testSchedular() {

String testValue = queues.*schedular*();

String expectedValue = "Google Chrome";

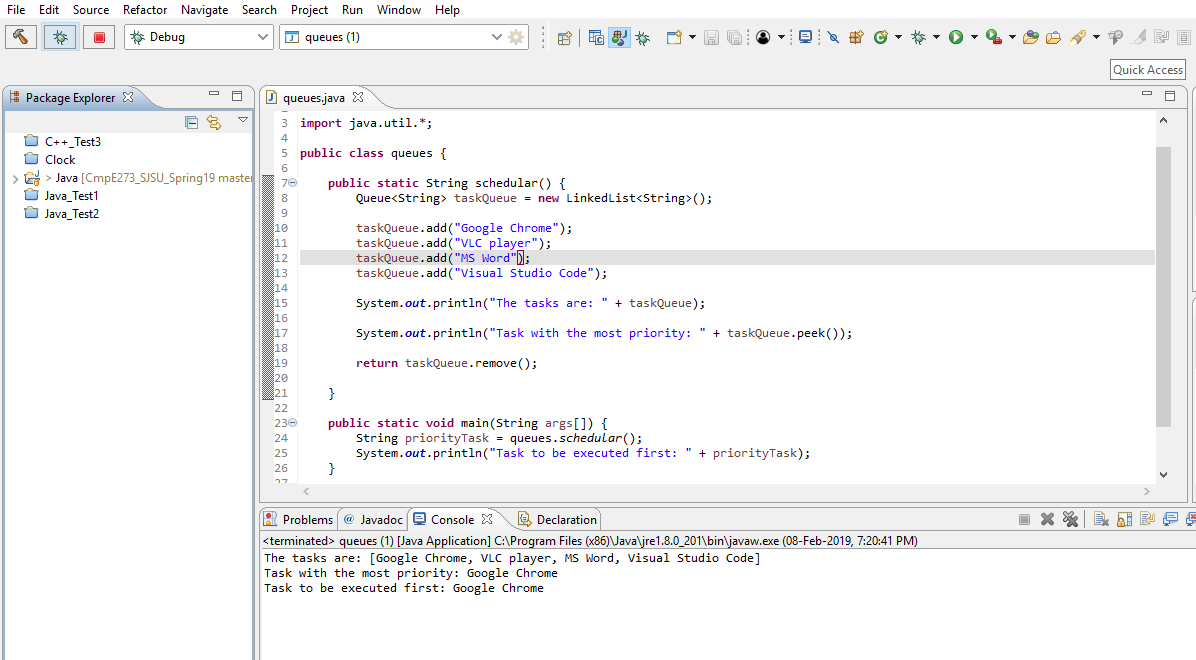
*assertEquals*(expectedValue, testValue);

}

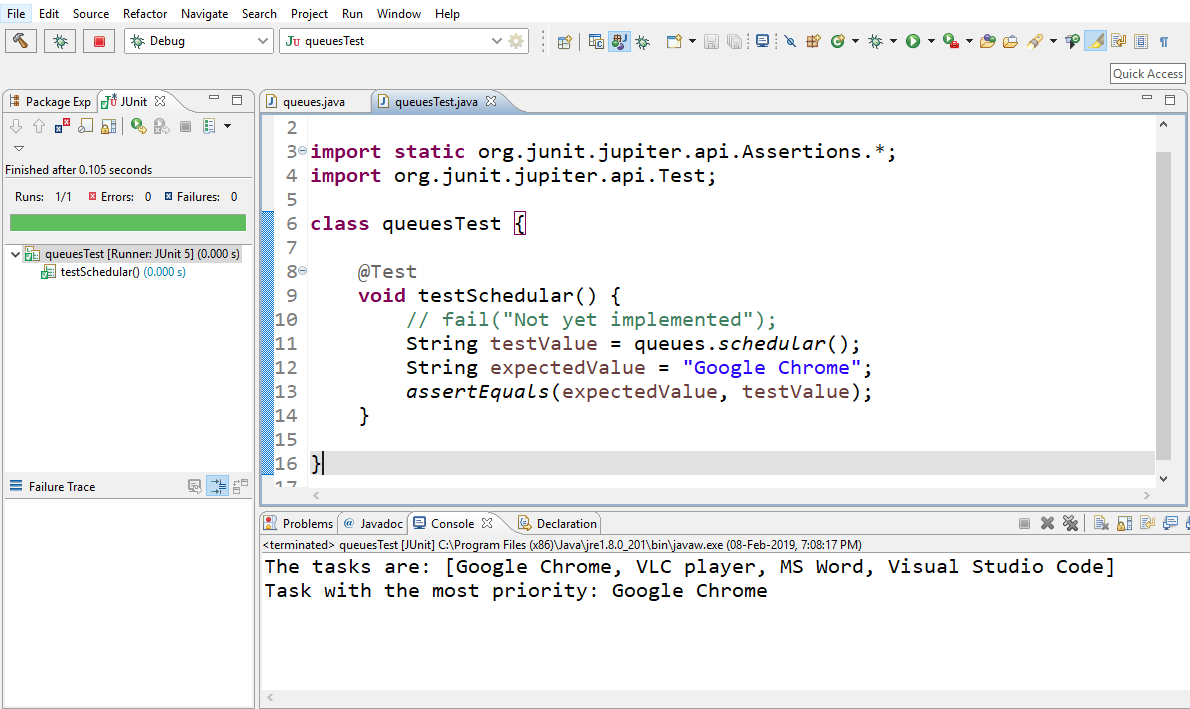
}

* **Output:**

1. Implementation:

****

1. JUnit Test:

****

**Stacks:**

* **Introduction:** Stack is a kind of data structure that is based on LIFO. The first entity in is the entity that gets out of the queue last.
* **Programming Question:** Write a program to create a deck containing 4 cards using stacks, and determine which card is at the bottom of the deck.
* **Code:**

**import** java.util.\*;

**public** **class** stacks {

**public** String deck() {

Stack<String> cardStack = **new** Stack<String>();

cardStack.push("Queen of Hearts");

cardStack.push("Jack of Clubs");

cardStack.push("Ace of Diamonds");

cardStack.push("Seven of Spades");

System.***out***.println("List of Cards: " + cardStack);

**return** cardStack.firstElement();

}

**public** **static** **void** main(String args[]) {

stacks myStack = **new** stacks();

System.***out***.println("First card pushed into the deck: " + myStack.deck());

}

}

**JUnit Test:**

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

class stacksTest {

private stacks classUnderTest;

@BeforeEach

void setUp() throws Exception {

classUnderTest = new stacks();

}

@Test

void test() {

String expectedValue = "Queen of Hearts";

String result = classUnderTest.deck();

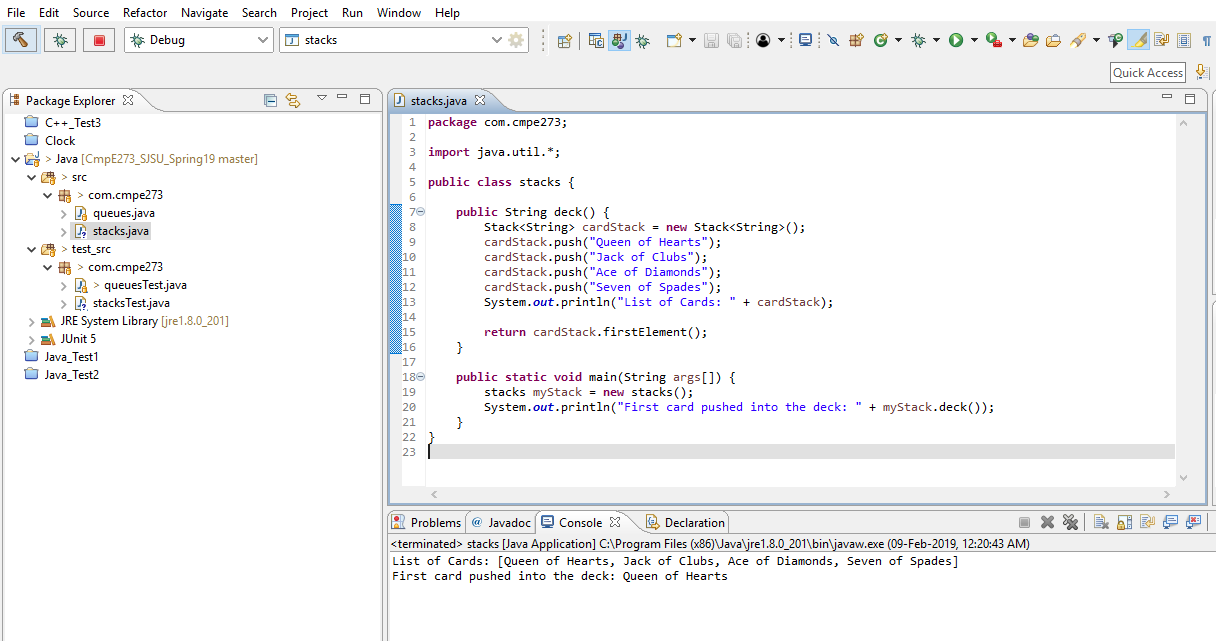
assertEquals(expectedValue, result);

}

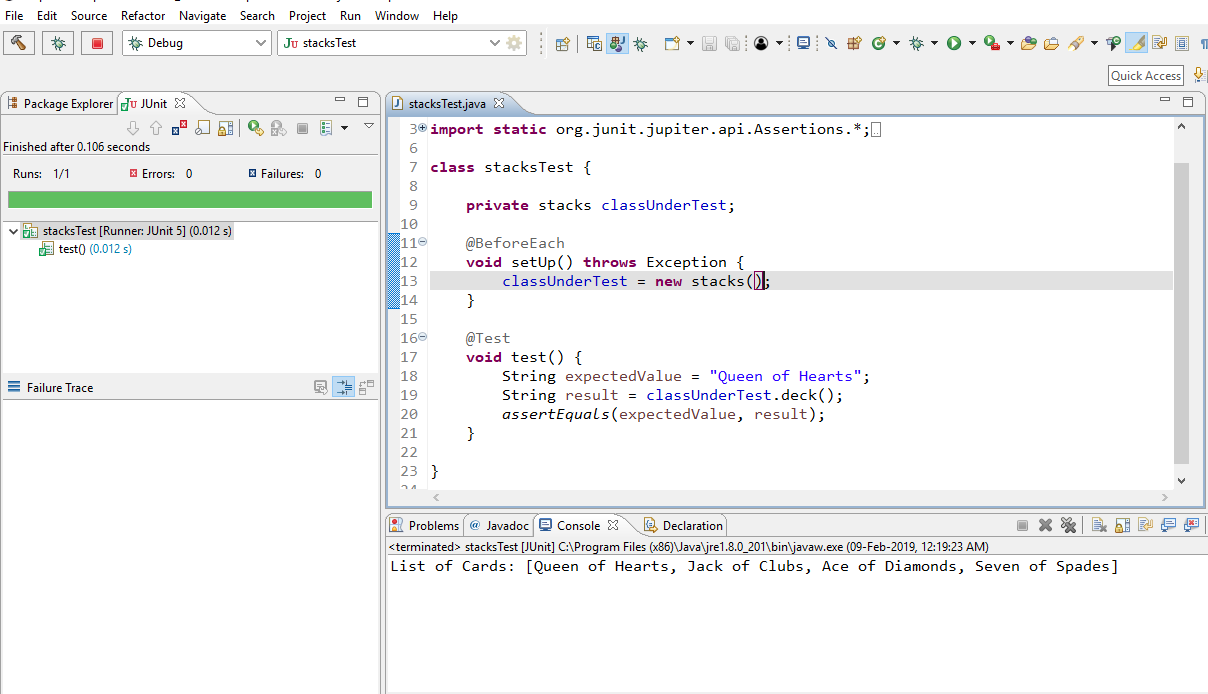
}

* **Output:**

1. Implementation:

****

1. JUnit Test:

****

**Arrays:**

* **Introduction:** Arrays are a type of data structure that store collections of data in a sequence. The data could be of the same type or different types. The data can be iterated through using indices and operated on, according to the requirement.
* **Programming Question:** Write a program that stores the courses taken by a student in an array and then outputs the 2nd course taken.
* **Code:**

**public** **class** arrays {

**public** String getCourses() {

String courses[] = {"CmpE 273", "CmpE 275", "CmpE 285"};

**return** courses[1];

}

**public** **static** **void** main(String[] args) {

arrays myCourses = **new** arrays();

String secondCourse = myCourses.getCourses();

System.***out***.println("The 2nd Course taken is: " + secondCourse);

}

}

**JUnit Test:**

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

class arraysTest {

private arrays classUnderTest;

@BeforeEach

void setUp() throws Exception {

classUnderTest = new arrays();

}

@Test

void test() {

String expectedValue = "CmpE 275";

String result = classUnderTest.getCourses();

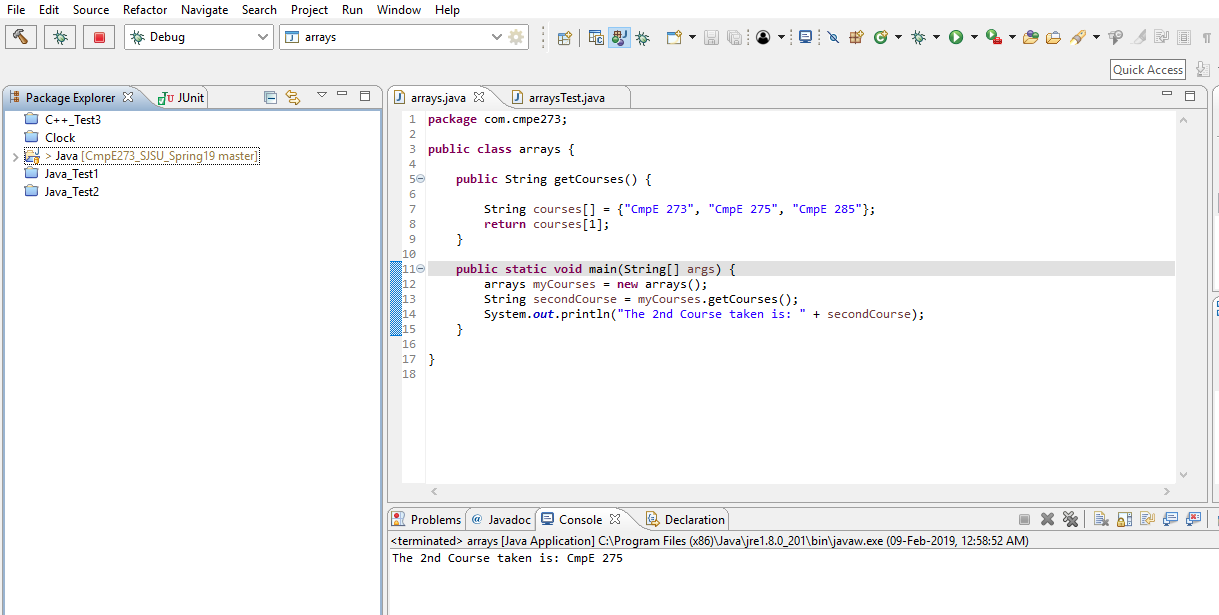
assertEquals(expectedValue, result);

}

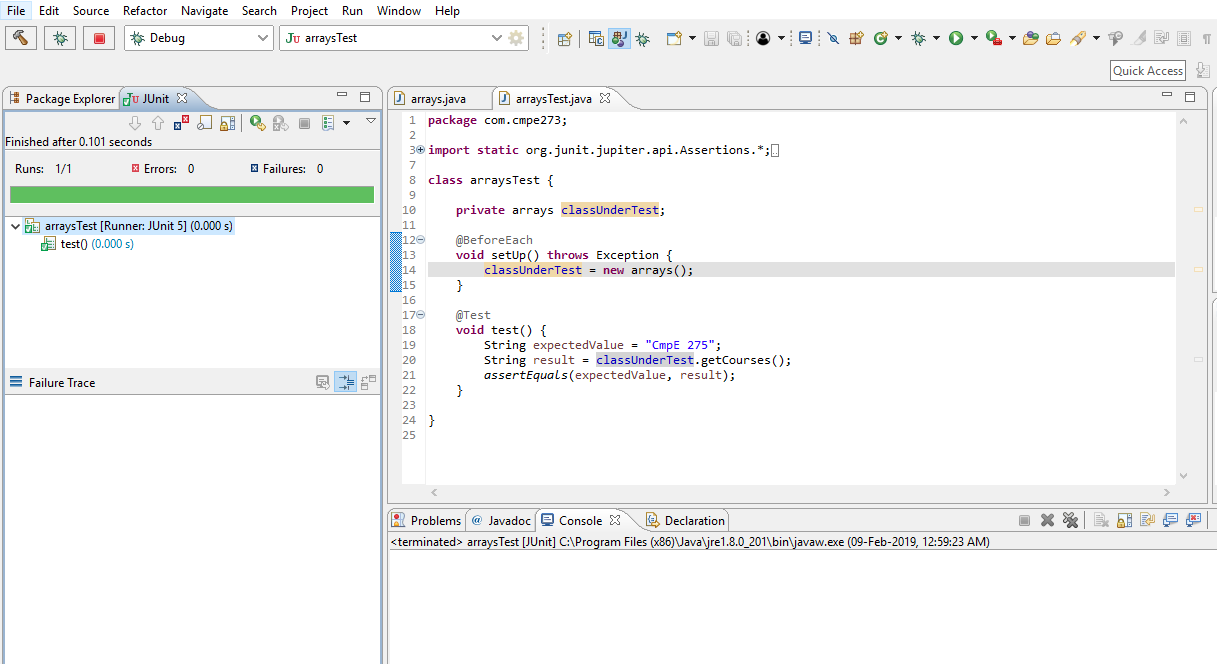
}

* **Output:**

1. Implementation:

****

1. JUnit Test:

****

**Interfaces:**

* **Introduction:** Interfaces are abstract classes that have abstract methods and have to be implemented by the classes that use this interface. Interfaces are used for multiple inheritances and abstraction.
* **Programming Question:** Write a program that has a deposit class and a deduct class that implement an abstract transact method. Transaction is of deposit or deduct depending on the class using the transact method.
* **Code:**

**public** **class** interfaces {

**public** **static** **void** main(String args[]) {

interfaces newTransaction = **new** interfaces();

**double** finalAmount = newTransaction.calculate();

System.***out***.println("Final Amount: " + finalAmount);

}

**public** **double** calculate() {

**double** initialAmount = 20000.0;

**double** depositAmount = 500.0;

**int** depositCount = 4;

**double** deductAmount = 200.0;

**int** deductCount = 10;

transact newDeposit = **new** deposit();

**double** totalAfterDeposit = newDeposit.transaction(depositCount, depositAmount, initialAmount);

transact newDeduct = **new** deduct();

**double** totalAfterDeduct = newDeduct.transaction(deductCount, deductAmount, totalAfterDeposit);

System.***out***.println("Initial Amount: " + initialAmount);

System.***out***.println("Amount after deposit: " + totalAfterDeposit);

System.***out***.println("Amount after deduct: " + totalAfterDeduct);

**return** totalAfterDeduct;

}

}

**interface** transact {

**double** transaction(**int** transactionCount, **double** transactionAmount, **double** totalAmount);

}

**class** deposit **implements** transact {

**public** **double** transaction(**int** transactionCount, **double** transactionAmount, **double** totalAmount) {

**double** totalTransaction = transactionAmount\*transactionCount;

totalAmount = totalAmount + totalTransaction;

**return** totalAmount;

}

}

**class** deduct **implements** transact {

**public** **double** transaction(**int** transactionCount, **double** transactionAmount, **double** totalAmount) {

**double** totalTransaction = transactionAmount\*transactionCount;

totalAmount = totalAmount - totalTransaction;

**return** totalAmount;

}

}

**JUnit Test:**

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**class** interfacesTest {

**private** interfaces classUnderTest;

@BeforeEach

**void** setUp() **throws** Exception {

classUnderTest = **new** interfaces();

}

@Test

**void** test() {

**double** expectedValue = 20000.0;

**double** result = classUnderTest.calculate();

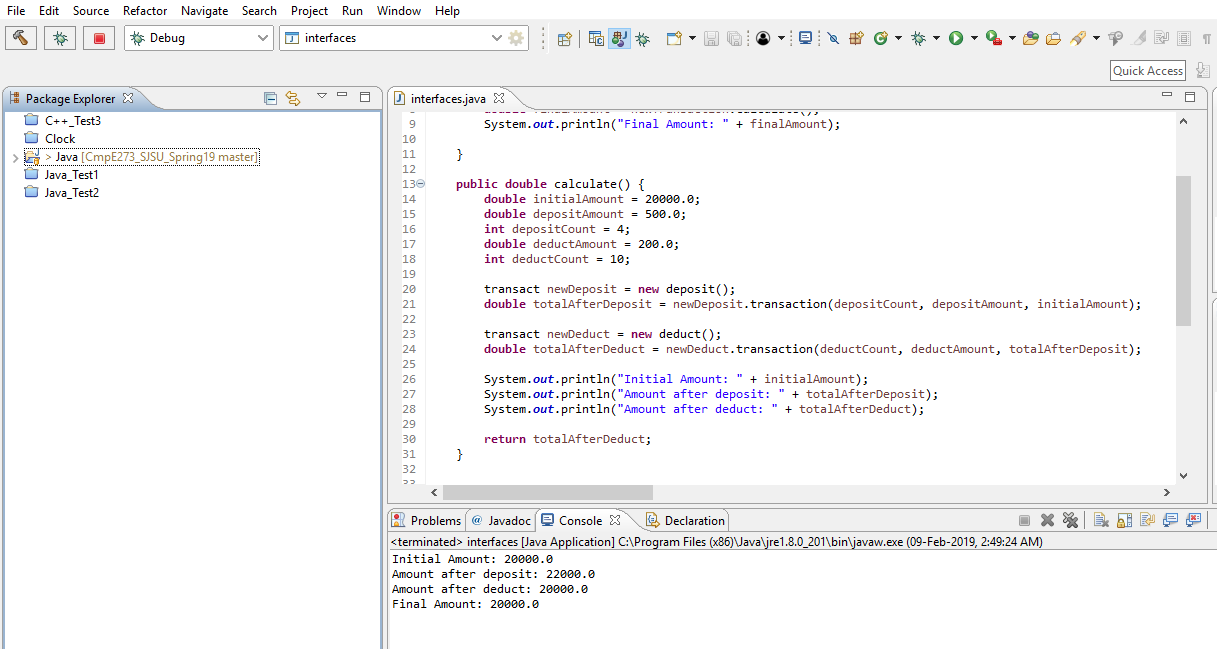
*assertEquals*(expectedValue, result);

}

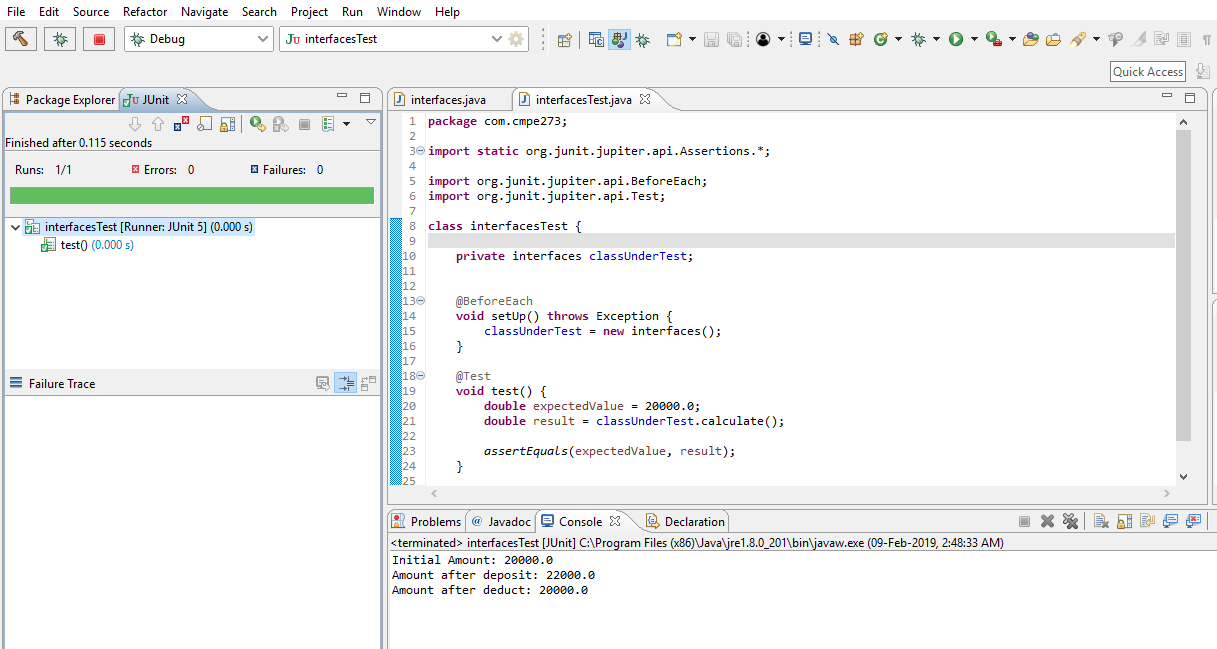
}

* **Output:**

1. Implementation:

****

1. JUnit Test:

****

**Collections:**

* **Introduction:** Collections is a data structure that provides the functionality to group similar entities together.
* **Programming Question:** Write a program to store all the courses taken by a student into a HashSet and then query the HashSet to see whether a user entered course is taken or not.
* **Code:**

**import** java.util.HashSet;

**public** **class** Collections {

HashSet<String> matchCourses = **new** HashSet<String>();

**public** **boolean** match(String matchThis) {

matchCourses.add("CmpE 285");

matchCourses.add("CmpE 273");

matchCourses.add("CmpE 275");

**return** matchCourses.contains(matchThis);

}

**public** **static** **void** main(String args[]) {

Collections newMatch = **new** Collections();

**boolean** isMatch\_1 = newMatch.match("CmpE 273");

**boolean** isMatch\_2 = newMatch.match("CmpE 281");

System.***out***.println("Match: " + isMatch\_1);

System.***out***.println("Match: " + isMatch\_2);

}

}

**JUint**:

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

class CollectionsTest {

private Collections classUnderTest;

@BeforeEach

void setUp() throws Exception {

classUnderTest = new Collections();

}

@Test

void test() {

boolean result = classUnderTest.match("CmpE 273");

boolean expectedValue = true;

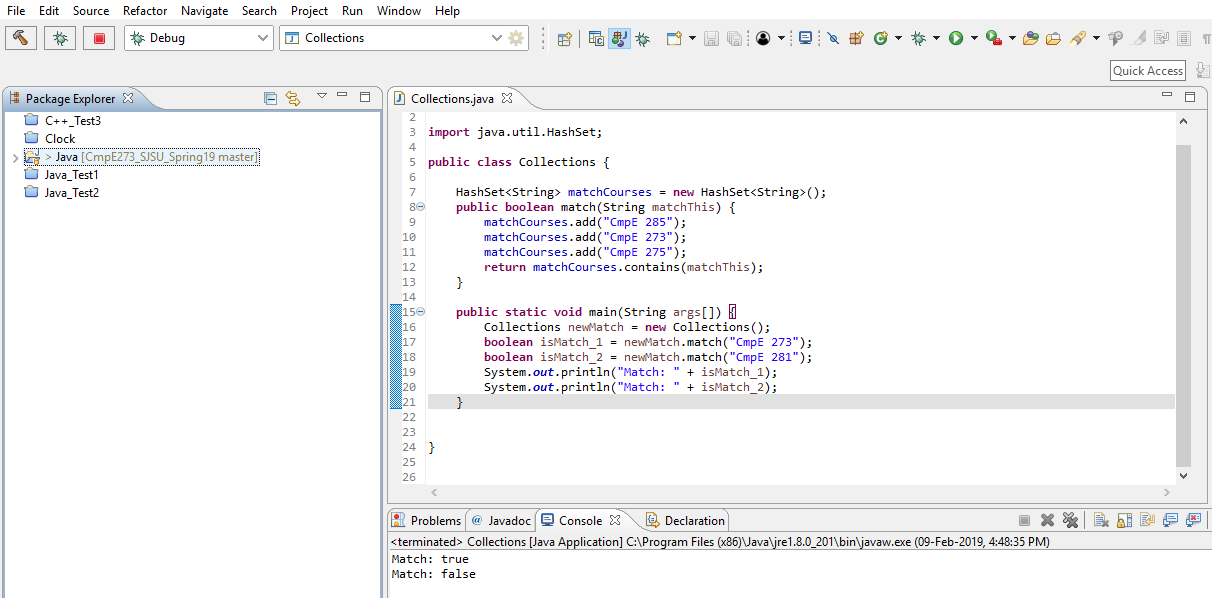
assertEquals(expectedValue, result);

}

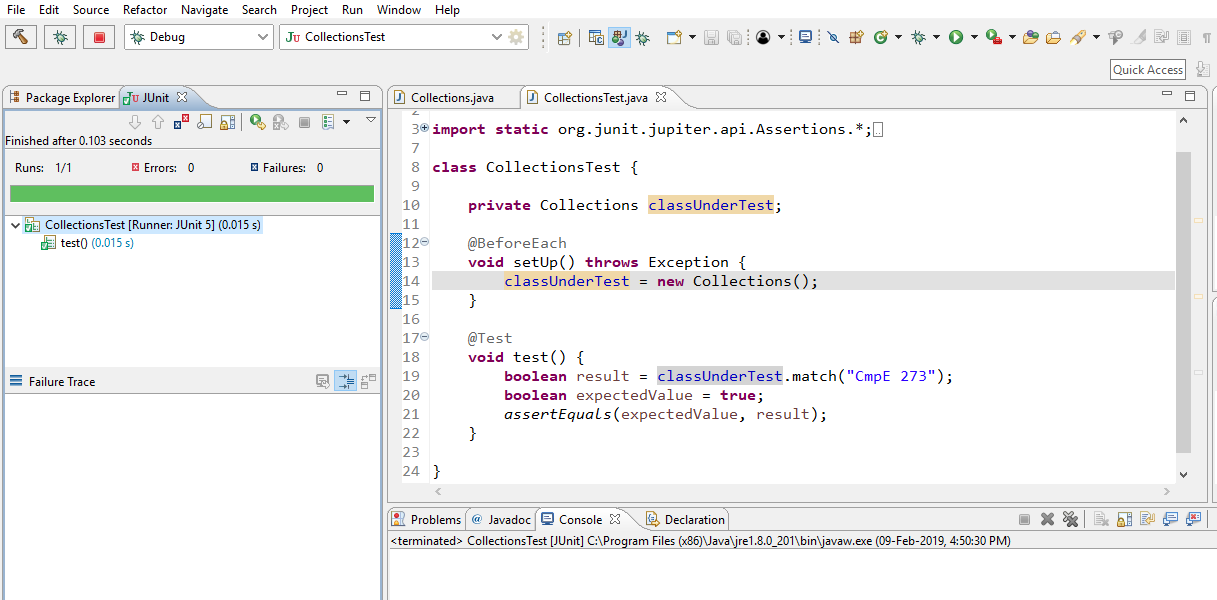
}

* **Output:**

1) Implementation:

****

2) JUnit Test:

****

**Generics:**

* **Introduction:** Generics in Java are similar to templates in C++. Generics help us use the same code for different data types without the use of type casting. Generics help us detect type errors and provide the code with type safety.
* **Programming Question:** Write a program that stores Driving License Numbers of users. The user can enter the driving license as a string or as an integer. The program also checks whether a queried driving license is already taken or not.
* **Code:**

**import** java.util.HashSet;

**public** **class** generics {

HashSet<String> stringLicenses = **new** HashSet<String>();

HashSet<Integer> intLicenses = **new** HashSet<Integer>();

**public** <E> **boolean** checkLicense(E check) {

stringLicenses.add("A739GH");

stringLicenses.add("Y7E3UI");

intLicenses.add(898746);

intLicenses.add(135957);

intLicenses.add(423162);

**if**(check **instanceof** String && stringLicenses.contains(check)) {

**return** **true**;

} **else** **if**(check **instanceof** Integer && intLicenses.contains(check)) {

**return** **true**;

} **else** {

**return** **false**;

}

}

**public** **static** **void** main(String args[]) {

generics license = **new** generics();

**boolean** checkThis\_1 = license.checkLicense("Y7E3UI");

**boolean** checkThis\_2 = license.checkLicense("R983YC");

**boolean** checkThis\_3 = license.checkLicense(135957);

**boolean** checkThis\_4 = license.checkLicense(841135);

System.***out***.println(checkThis\_1);

System.***out***.println(checkThis\_2);

System.***out***.println(checkThis\_3);

System.***out***.println(checkThis\_4);

}

}

**JUnit Test:**

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.AfterAll;

**import** org.junit.jupiter.api.AfterEach;

**import** org.junit.jupiter.api.BeforeAll;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**class** genericsTest {

**private** generics classUnderTest;

@BeforeAll

**static** **void** setup() {

System.***out***.println("@BeforeAll - executes once before all test methods in this class");

}

@BeforeEach

**void** init() {

classUnderTest = **new** generics();

System.***out***.println("@BeforeEach - executes before each test method in this class");

}

@Test

**void** test() {

**boolean** testValue = classUnderTest.checkLicense(135957);

**boolean** expectedValue = **true**;

*assertEquals*(expectedValue, testValue);

}

@AfterEach

**void** tearDown() {

System.***out***.println("@AfterEach - executed after each test method.");

}

@AfterAll

**static** **void** done() {

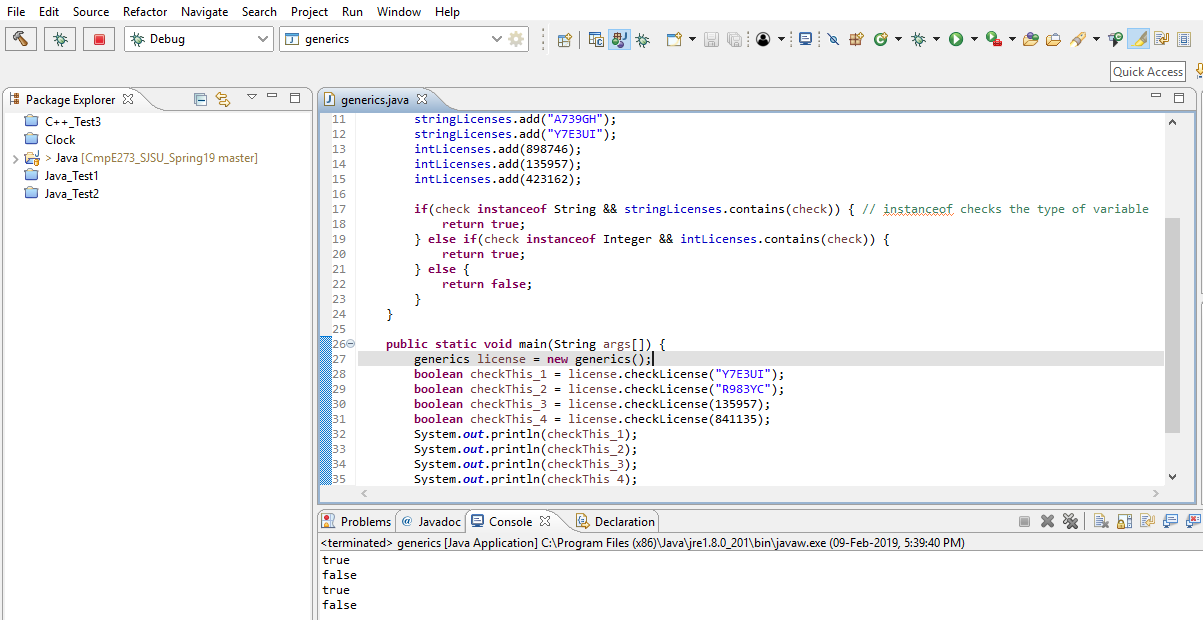
System.***out***.println("@AfterAll - executed after all test methods.");

}

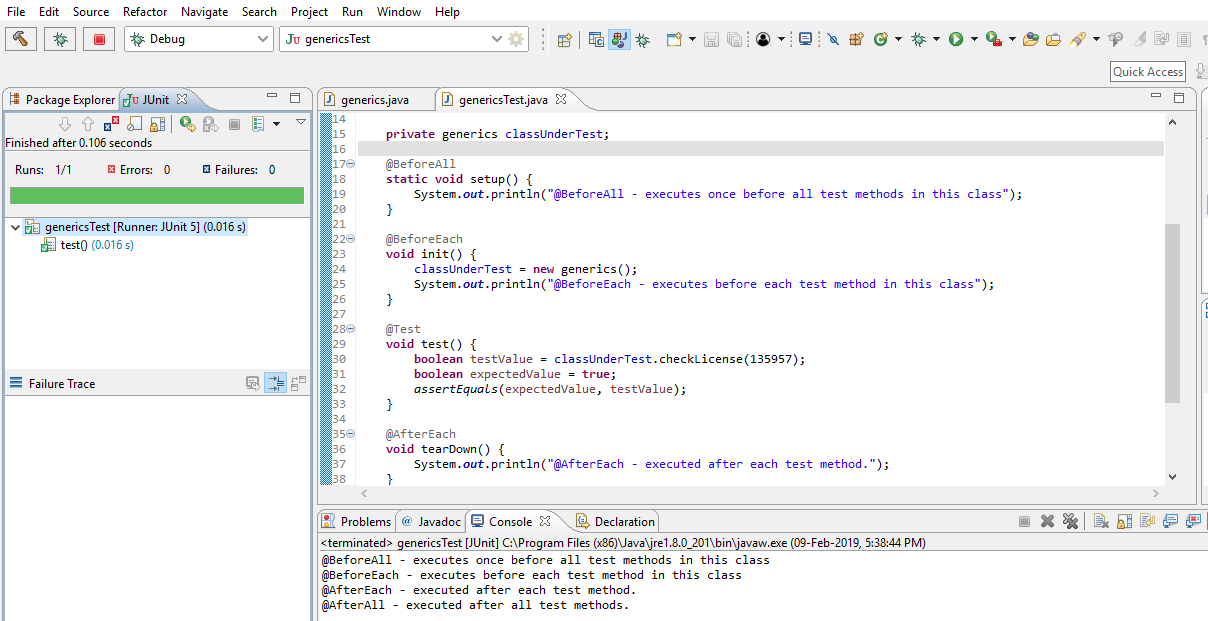
}

* **Output:**

1. Implementation:

****

1. JUnit Test:

****

**Multithreading:**

* **Introduction:** Multithreading provides the use of threads in our programs. We can assign different blocks of code to different threads that run simultaneously so that the whole code executes faster, with the help of the multi core processors.
* **Programming Question:** Write a program to use multithreading to use 2 different ports simultaneously for network access. The program should listen at both the ports simultaneously and then print the output.
* **Code:**

**public** **class** multithreading {

**public** **static** **void** main(String args[]) {

*portMultithreading*();

}

**public** **static** String portMultithreading() {

Port\_1 port1 = **new** Port\_1();

Port\_2 port2 = **new** Port\_2();

port1.start();

**try**{Thread.*sleep*(20);} **catch**(Exception e){System.***out***.println(e);}

port2.start();

**return** "Last port listened to: 2";

}

}

**class** Port\_1 **extends** Thread {

**public** **void** run() {

**for**(**int** i=0;i<5;i++) {

System.***out***.println("Listening at Port 1");

**try** { // Raise exception if sleep is not activated

Thread.*sleep*(500);

}

**catch**(Exception e) {

System.***out***.println(e);

}

}

}

}

**class** Port\_2 **extends** Thread {

**public** **void** run() {

**for**(**int** i=0;i<5;i++) {

System.***out***.println("Listening at Port 2");

**try** {

Thread.*sleep*(500);

}

**catch**(Exception e) {

System.***out***.println(e);

}

}

}

}

**JUnit Test:**

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**class** multithreadingTest {

@Test

**void** test() {

String testValue = multithreading.*portMultithreading*();

String expectedValue = "Last port listened to: 2";

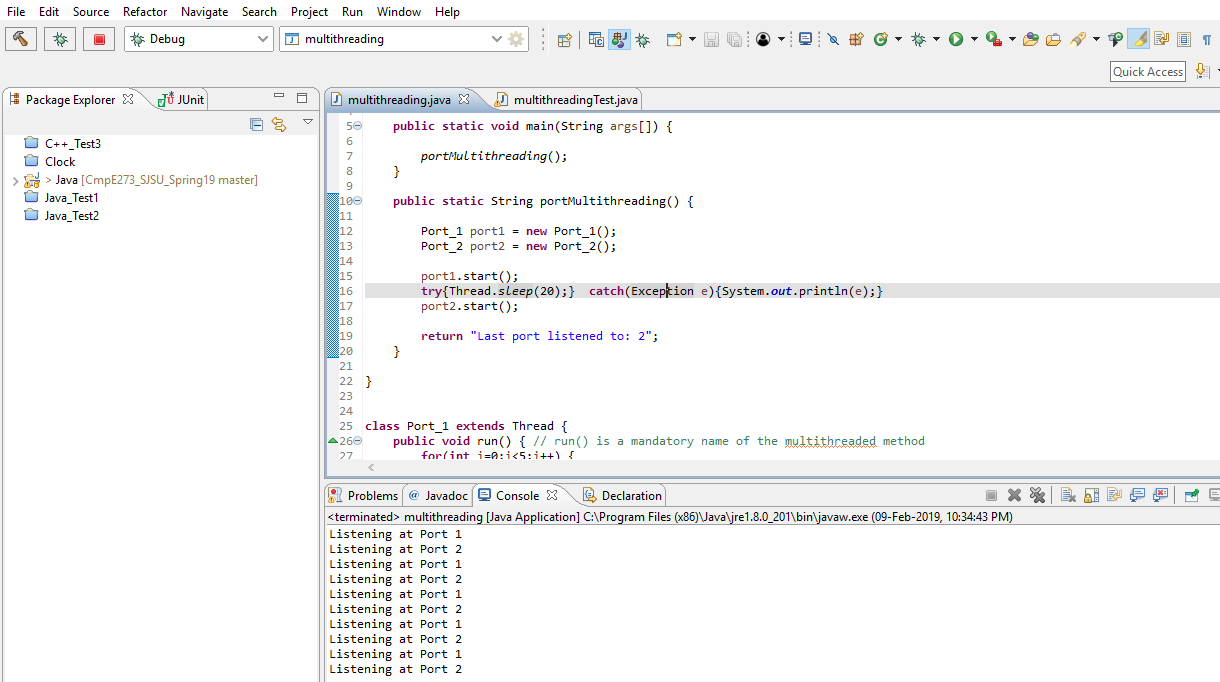
*assertEquals*(expectedValue, testValue);

}

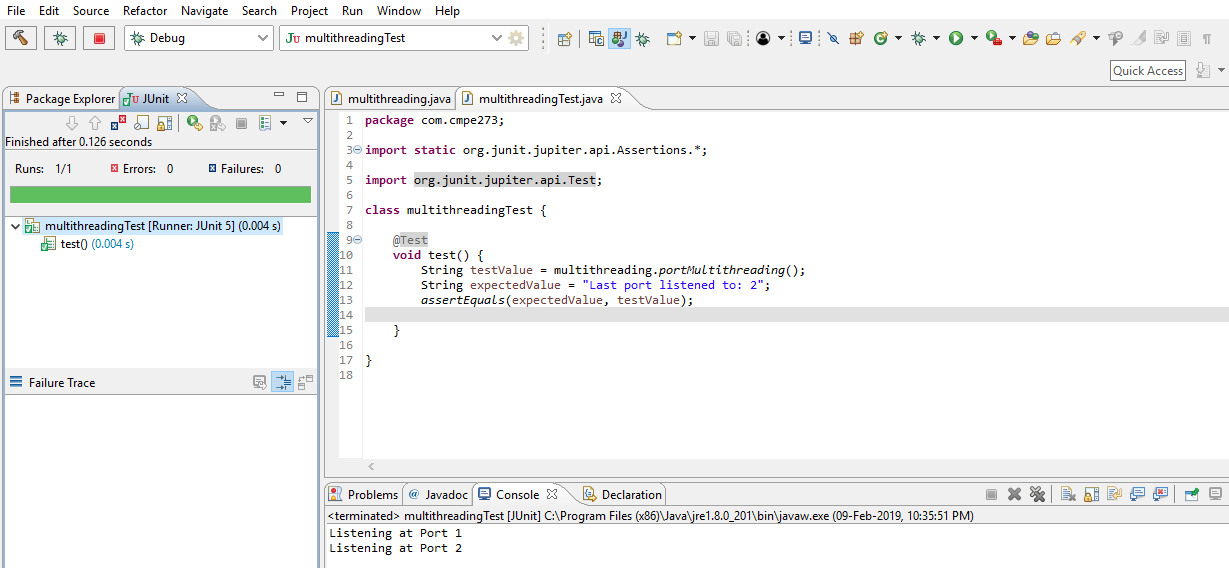
}

* **Output:**

1. Implementation:

****

1. JUnit Test:

****