

FULL STACK WEB DEVELOPEMNT LABORATORY

Prepared by

**Dr. B. Padmaja**

Associate Professor

CAREER DEVELOPMENT CENTER

**INSTITUTE OF AERONAUTICAL ENGNEERING**

**FULL STACK WEB DEVELOPEMNT LABORATORY**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S No.** | **Topic Name** | **Page No.** |
|  | **Object Oriented Programming (OOPS)**   1. Class and Objects 2. Encapsulation/Abstraction 3. Inheritance 4. Overriding/Polymorphism 5. String/String Buffer 6. Abstract classes 7. Packages 8. Interfaces | **4** |
|  | **Wrapper Classes & Exception Handling**   * 1. Wrapper Classes   2. Exception Handling | **20** |
|  | **I/O Streams**   1. I/O Operations 2. Object Serialization | **24** |
|  | **Collections**   1. Lists 2. Sets 3. Maps | **28** |
|  | **JUnit**   1. Junit with Eclipse 2. Assert Methods and Annotations 3. Test Suites | **35** |
|  | **Multi-Threading**   1. Thread Creation 2. Thread Control and Priorities | **37** |
|  | **RDBMS/SQL/JDBC**   1. Select Statement 2. Restricting and Sorting 3. Data Modification Language(DML) 4. Data Defination Language(DDL) 5. Establishing Connections 6. Executing Query and Processing Results 7. Using PreparedStatement and MetaData objects 8. Using CallableStatement and transactions | **39** |
|  | **Hyper Text Markup Language (HTML)**   1. Layout Tags and Semantic Tags 2. Tables,Forms and Frames 3. Style and div tags 4. Introduction to HTML 5 | **49** |
|  | **Cascading Style Sheets (CSS) /JavaScript**   1. Styles and Style sheets 2. Formatting with CSS 3. Links and Lists 4. CSS Box Model 5. CSS3 6. Introduction to JavaScript and Basic Functions 7. JavaScript Objects 8. JavaScript Validations 9. Introduction to BootStrap 10. BootStrap Grid System 11. Front-End Framework 12. Bootstrap Grid System | **60** |
|  | **Servlet /** **Java Server Pages(JSP)**   1. Introduction to Servlets 2. Servlet-Get and Post Requests 3. Serlet Config and Servlet Context 4. Servlet-Cookies and Session Management 5. Introduction to JSP 6. Scripting Elements, Directive Elements, Standard Action Elements, Scope Variables, Implicit objects 7. JavaBeans in JSP | **92** |
|  | **Asynchronous JavaScript and XML(AJAX)**   1. Ajax Application 2. Ajax Database Application 3. XML Parsing 4. JSON Parsing | **97** |
|  | **Hibernate**   1. Introduction to Hibernate 2. Objects 3. Hibernate with Annotations 4. HQL 5. Mapping | **104** |
|  | **Spring**   1. Spring Basics 2. Spring Inversion of Control 3. Spring MVC 4. Spring MVC with JDBC & Hibernate | **132** |
|  | **Spring Boot**   1. Introduction 2. Spring Boot Dependency Injection And Auto Wiring 3. Spring Webapp MVC 4. Spring Boot MVC JPA H2 5. Spring Boot MVC JPA H2 Rest | **154** |
|  | **ReactJS**   * 1. React Overview, Installation   2. Components   3. State, Props, Props Validations   4. Component API, Component Lifecycle  1. Forms, Events 2. Routers | **164** |

**1. Object Oriented Programming**

**1.1 Class and Objects**

1. Create a class Box that uses a parameterized constructor to initialize the dimensions of a box. The dimensions of the Box are width, height, depth. The class should have a method that can returs the volume of the box. Create an object of the Box class and test the functionalities.

class Box {

write code here

}

public double volume() {

write code here

}

public double getWidth() {

DONE

write code here

}

public double getHeight() {

write code here

}

public double getDepth() {

write code here

}

public static void main(String[] args) {

write code here

}

}

1. Create a new class called calculator with the following methods:
   * + A static method called **powerInt(int num1, int num2).**
     + A static method called **powerDouble(double num1, int num2).**

This method should return num1 to the power num2. Invoke both the methods and test the functionalities.

**Hint:** Use Math.pow(double, double) to calculate the power.

public class Calculator {

public static int powerInt(int num1, int num2) {

return (int) Math.pow(num1, num2);

}

public static double powerDouble(double num1, int num2) {

return Math.pow(num1, num2);

DONE

}

public static void main(String[] args) {

write code here

}

}

**1.2 Encapsulation / Abstraction**

1. Create a class Author with the following information. Member variables: name (String), email (String), and gender (char) Parameterized Constructor: To initialize the variables

public class Author {

write code here

public String getName() {

return name;

}

public String getEmail() {

return email;

DONE

}

public char getGender() {

return gender;

}

@Override

public String toString() {

write code here

}

public static void main(String[] args) {

write code here

}

}

1. Create a class Book with the following information.

Member variables: name (String), author (of the class Author you have just created), price (double), and qtyInStock (int)

[Assumption: Each book will be written by exactly one Author]

**Parameterized Constructor:** To initialize the variables Getters and Setters for all the member variables. In the main method, create a book object and print all details of the book (including the author details)

public class Book {

private String name;

private Author author;

private double price;

private int qtyInStock;

public Book(String name, Author author, double price, int qtyInStock) {

write code here

}

public String getName() {

return name;

}

DONE

public void setName(String name) {

this.name = name;

}

public Author getAuthor() {

return author;

}

public void setAuthor(Author author) {

this.author = author;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public int getQtyInStock() {

return qtyInStock;

}

public void setQtyInStock(int qtyInStock) {

this.qtyInStock = qtyInStock;

}

@Override

public String toString() {

write code here

}

public static void main(String[] args) {

write code here

}

}

class Author {

write code here

}

public String getName() {

write code here

}

public String getEmail() {

write code here

}

DONE

public char getGender() {

write code here

}

@Override

public String toString() {

write code here

}

}

**1.3 Inheritance**

1. Create a class named 'Animal' which includes methods like eat() and sleep(). Create a child class of Animal named 'Bird' and override the parent class methods.  Add a new method named fly Create an instance of Animal class and invoke the eat and sleep methods using this object. Create an instance of Bird class and invoke the eat, sleep and fly methods using this object.

class Animal {

public void eat() {

System.out.println("Animal is eating");

}

public void sleep() {

System.out.println("Animal is sleeping");

}

DONE

}

class Bird extends Animal {

@Override

public void eat() {

System.out.println("Bird is eating");

}

@Override

public void sleep() {

System.out.println("Bird is sleeping");

}

public void fly() {

System.out.println("Bird is flying");

}

public static void main(String[] args) {

write code here

}

}

1. Create a class called Person with a member variable name. Save it in a file called Person.java. Create a class called Employee that will inherit the Person class. The other data members of the Employee class are annual salary (double),  the year the employee started to work, and the national insurance number which is a String. Save this in a file called Employee.java Your class should have the necessary constructors and getter / setter methods. Write another class called TestEmployee, containing a main method to fully test your class definition.

public class Person {

private String name;

public Person(String name) {

this.name = name;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

public class Employee extends Person {

write code here

DONE

}

public double getAnnualSalary() {

return annualSalary;

}

public void setAnnualSalary(double annualSalary) {

this.annualSalary = annualSalary;

}

public int getYearStarted() {

return yearStarted;

}

public void setYearStarted(int yearStarted) {

this.yearStarted = yearStarted;

}

public String getNationalInsuranceNumber() {

return nationalInsuranceNumber;

}

public void setNationalInsuranceNumber(String nationalInsuranceNumber) {

this.nationalInsuranceNumber = nationalInsuranceNumber;

}

@Override

public String toString() {

write code here

}

}

public class TestEmployee {

public static void main(String[] args) {

DONE

write code here

}

}

**1.4 Overriding / Polymorphism**

1. Create a base class Fruit with name, taste and size as its attributes. Create a method called eat() which describes the name of the fruit and its taste. Inherit the same in 2 other classes Apple and Orange and override the eat() method to represent each fruit taste.

public class Fruit {

private String name;

private String taste;

private String size;

public Fruit(String name, String taste, String size) {

write code here

DONE

}

public void eat() {

System.out.println("This is a " + name + " and it tastes " + taste + ".");

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getTaste() {

return taste;

}

public void setTaste(String taste) {

this.taste = taste;

}

public String getSize() {

return size;

}

public void setSize(String size) {

this.size = size;

}

}

public class Apple extends Fruit {

DONE

public Apple(String size) {

super("Apple", "sweet", size);

}

@Override

public void eat() {

System.out.println("This is an Apple and it tastes sweet.");

}

}

public class Orange extends Fruit {

public Orange(String size) {

super("Orange", "sour", size);

}

DONE

@Override

public void eat() {

System.out.println("This is an Orange and it tastes sour.");

}

}

public class TestFruits {

public static void main(String[] args) {

write code here

}

}

1. Write a program to create a class named shape. It should contain2 methods, draw() and erase() that prints  "Drawing Shape" and "Erasing Shape" respectively. For this class, create three sub classes, Circle, Triangle and Square and each class should override the parent classfunctions - draw() and erase. The draw() method should print "Drawing Circle", "Drawing Triangle" and "Drawing Square respectively. The erase() method should print "Erasing Circle", "Erasing Triangle" and "Erasing Square" respectively. Create objects of Circle, Triangle and Square in the following way and observe the polymorphic nature of the class by calling draw() and erase() method using each object.

Shape c = new Circle();

Shape t = new Triangle();

Shape s = new Square();

public class Shape {

public void draw() {

System.out.println("Drawing Shape");

}

public void erase() {

System.out.println("Erasing Shape");

}

}

DONE

public class Circle extends Shape {

write code here

}

public class Triangle extends Shape {

write code here

}

public class Square extends Shape {

write code here

}

public class TestShapes {

public static void main(String[] args) {

write code here

}

}

**1.5 String / String Buffer**

1. Write a program to check whether a given string is Palindrome or not.

public class PalindromeChecker {

public static boolean isPalindrome(String s) {

write code here

DONE

}

public static void main(String[] args) {

write code here

}

}

1. Write a Java program that will concatenate 2 strings and return the result. The result should be in lowercase.

Note: If the concatenation creates a double-char, then one of the characters need to be omitted.

**Input** Sachin, Tendulkar

**Output**: sachintendulkar

**Input**: Mark, kate

**Output**: markate

public class StringConcatenation {

public static String concatenateStrings(String str1, String str2) {

write code here

}

DONE

public static void main(String[] args) {

write code here

}

}

3. Given a string, return a new string made of 'n' copies of the first 2 chars of the original string where 'n' is the length of the string.

**Input**: Wipro

**Output**: WiWiWiWiWi

public class StringCopies {

public static String repeatFirstTwoChars(String str) {

write code here

DONE

}

public static void main(String[] args) {

write code here

}

}

1. Read a string and write a java program that will return the first half of the string, if the length of the string is even. It should return null for odd length string.

**Input**: TomCat

**Output**: Tom

**Input**: Apron

**Output**: null

public class StringHalf {

public static String getFirstHalf(String str) {

write code here

}

DONE

public static void main(String[] args) {

write code here

}

}

1. Read a string and returns a new string without the first and last character of the input string.

**Input**: Suman

**Output**: uma

public class RemoveFirstAndLast {

public static String removeFirstAndLast(String str) {

write code here

DONE

}

public static void main(String[] args) {

write code here

}

}

1. Given 2 strings, a and b, return a new string of the form short+long+short, with the shorter string on the outside and the longer string in the inside. The strings will not be the same length, but they may be empty (length 0). If input is "hi" and "hello", then output will be "hihellohi".

public class ShortLongShort {

public static String shortLongShort(String a, String b) {

write code here

}

DONE

public static void main(String[] args) {

write code here

}

}

1. Given a string, if the first or last chars are 'x', return the string without those 'x' chars, otherwise return the string unchanged. if the input is "xHix", then output is "Hi". if the input is "America", then the output is "America".

public class RemoveX {

public static String removeX(String str) {

write code here

}

DONE

public static void main(String[] args) {

write code here

}

}

8. Read a string (with \* in it). The program should return a new string in which the following characters are removed-\*, the characters that are to the left and right of \*

**Input**: ab\*cd

**Output**: ad

public class RemoveAdjacentChars {

public static String removeAdjacentChars(String str) {

write code here

}

public static void main(String[] args) {

write code here

}

}

9. Given two strings, a and b, print a new string which is made of the following combination-first character of a, the first character of b, second character of a, second character of b and so on. Any characters left, will go to the end of the result.

**Input**: Hello,World

**Output**: HWeolrllod

public class CombineStrings {

public static String combineStrings(String a, String b) {

write code here

}

}

10. Given a string and an integer n, print a new string made of n repetitions of the last n characters of the string. You may assume that n is between 0 and the length of the string, inclusive.

**Input**: Wipro,3

**Output**: propropro

public class RepeatLastNChars {

public static String repeatLastNChars(String str, int n) {

write code here

}

public static void main(String[] args) {

write code here

}

}

**1.6 Abstract Classes**

1. Create an abstract class Compartment to represent a rail coach. Provide an abstract function notice in this class.

public abstract String notice();

Derive FirstClass, Ladies, General, Luggage classes from the compartment class. Override the notice function in each of them to print notice message that is suitable to the specific type of compartment.

Create a class TestCompartment . Write main function to do the following:

* Declare an array of Compartment of size 10.
* Create a compartment of a type as decided by a randomly generated integer in the range 1 to 4.
* Check the polymorphic behavior of the notice method.
* [i.e based on the random number genererated, the first compartment can be Luggage, the second one could be Ladies and so on..]

**Compartment.java**

public abstract class Compartment {

public abstract String notice();

}

FirstClass.java

public class FirstClass extends Compartment {

@Override

DONE

public String notice() {

return "First Class Compartment";

}

}

**Ladies.java**

public class Ladies extends Compartment {

@Override

public String notice() {

DONE

return "Ladies Compartment";

}

}

**General.java**

public class General extends Compartment {

@Override

public String notice() {

DONE

return "General Compartment";

}

}

**Luggage.java**

public class Luggage extends Compartment {

@Override

public String notice() {

DONE

return "Luggage Compartment";

}

}

**TestCompartment.java**

import java.util.Random;

public class TestCompartment {

DONE

public static void main(String[] args) {

write code here

}

}

1. Create a package called test package. Define a class called foundation inside the test package. Inside the class, you need to define 4 integer variables:

* var1 with private access modifier
* var2 with default access modifier
* var3 with protected access modifier
* var4 with public access modifier

Import this class and packages in another class. Try to access all 4 variables of the foundation class and see what variables are accessible and what are not accessible.

**Step 1:** Create the Foundation Class in the testpackage Package

Foundation.java (inside testpackage)

package testpackage;

public class Foundation {

write code here

}

}

**Step 2:** Create Another Class to Access Foundation Class Variables

AccessFoundation.java (outside testpackage)

import testpackage.Foundation;

public class AccessFoundation {

public static void main(String[] args) {

write code here

}

}

1. Create a class called compartment which represents the ship compartments with attributes like height, width and breadth. Take care it should not conflict with the compartment class you have created in Abstract class exercise 2. To avoid conflict create this class in a new package called com.wipro.automobile.ship

import com.wipro.automobile.ship.Compartment;

public class TestShipCompartment {

public static void main(String[] args) {

write code here

}

}

**1.7 Packages**

1. Create a package called com.automobile. Define an abstract class called Vehicle. Vehicle class has the following abstract methods:

public String getModelName()

public String getRegistrationNumber()

public String getOwnerName()

Create twowheeler subpackage under automobile package Hero class extends automobile.vehicle class with the following methods

**public int getSpeed():** returns the current speed of the vehicle.

**public void radio():** provides facility to control the radio device

Honda class extends com.automobile.vehicle class with the following methods

**public int getSpeed():** Returns the current speed of the vehicle.

**public void cdplayer():** provides facility to control the cd player device which is available in the car. Create a test class to test the methods available in all these child class.

**Step 1:** Create the Vehicle Abstract Class

**Vehicle.java** (inside com/automobile package structure)

package com.automobile;

public abstract class Vehicle {

public abstract String getModelName();

public abstract String getRegistrationNumber();

public abstract String getOwnerName();

}

**Step 2:** Create the Hero Class (Twowheeler Subpackage)

**Hero.java** (inside com/automobile/twowheeler package structure)

package com.automobile.twowheeler;

import com.automobile.Vehicle;

public class Hero extends Vehicle {

write code here

}

@Override

public String getModelName() {

return modelName;

}

@Override

public String getRegistrationNumber() {

return registrationNumber;

}

@Override

public String getOwnerName() {

return ownerName;

}

public int getSpeed() {

return speed;

}

public void radio() {

System.out.println("Controlling the radio device");

}

}

**Step 3:** Create the Honda Class (Fourwheeler Subpackage)

**Honda.java** (inside com/automobile/fourwheeler package structure)

package com.automobile.fourwheeler;

import com.automobile.Vehicle;

public class Honda extends Vehicle {

private int speed;

public Honda(String modelName, String registrationNumber, String ownerName, int speed) {

write code here

}

@Override

public String getModelName() {

return modelName;

}

@Override

public String getRegistrationNumber() {

return registrationNumber;

}

@Override

public String getOwnerName() {

return ownerName;

}

public int getSpeed() {

return speed;

}

public void cdplayer() {

System.out.println("Controlling the CD player device");

}

}

**Step 4:** Create a Test Class to Test Methods

**TestVehicle**.java (outside com.automobile package)

import com.automobile.twowheeler.Hero;

import com.automobile.fourwheeler.Honda;

public class TestVehicle {

public static void main(String[] args) {

write code here

}

}

1. Add the following ideas to the previous hands on: Create FourWheeler subpackage under automobile package Logan class extends com.automobile.Vehicle class

**public int speed():** Returns the current speed of the vehicle.

**public int gps():** provides facility to control the gps device

Ford class extends com.automobile. Vehicle class

**public int speed()**: Returns the current speed of the vehicle.

**public int tempControl():** provides facility to control the air conditioning device which is available in the car Create objects of the relevant classes and test the various functionalities of the class.

**Step 1:** Define Logan Class

**Logan.java** (inside com/automobile/fourwheeler package structure)

package com.automobile.fourwheeler;

import com.automobile.Vehicle;

public class Logan extends Vehicle {

write code here

}

@Override

public String getModelName() {

return modelName;

}

@Override

public String getRegistrationNumber() {

return registrationNumber;

}

@Override

public String getOwnerName() {

return ownerName;

}

@Override

public int speed() {

return speed;

}

public int gps() {

write code here

}

}

**Step 2**: Define Ford Class

**Ford.java** (inside com/automobile/fourwheeler package structure)

package com.automobile.fourwheeler;

import com.automobile.Vehicle;

public class Ford extends Vehicle {

private int speed;

public Ford(String modelName, String registrationNumber, String ownerName, int speed) {

write code here

}

@Override

public String getModelName() {

return modelName;

}

@Override

public String getRegistrationNumber() {

return registrationNumber;

}

@Override

public String getOwnerName() {

return ownerName;

}

@Override

public int speed() {

return speed;

}

public int tempControl() {

write code here

}

}

**Step 3:** Update the TestVehicle Class to Test All Functionalities

**TestVehicle.java** (outside com.automobile package)

import com.automobile.twowheeler.Hero;

import com.automobile.fourwheeler.Honda;

import com.automobile.fourwheeler.Logan;

import com.automobile.fourwheeler.Ford;

public class TestVehicle {

public static void main(String[] args) {

write code here

}

}

1. Write an interface called Playable, with a method **void play();**

Let this interface be placed in a package called music. Write a class called Veena which implements Playable interface. Let this class be placed in a package **music.string.** Write a class called Saxophone which implements Playable interface. Let this class be placed in a package **music.wind**

Write another class Test in a package called live. Then,

1. Create an instance of Veena and call play() method
2. Create an instance of Saxophone and call play() method
3. Place the above instances in a variable of type Playable and then call play()

**Playable.java**

package music;

public interface Playable {

void play();

}

**Veena.java**

package music.string;

import music.Playable;

public class Veena implements Playable {

@Override

public void play() {

System.out.println("Veena is playing");

}

}

**Saxophone.java**

package music.wind;

import music.Playable;

public class Saxophone implements Playable {

@Override

public void play() {

System.out.println("Saxophone is playing");

}

}

**Test.java**

package live;

import music.Playable;

import music.string.Veena;

import music.wind.Saxophone;

public class Test {

public static void main(String[] args) {

write code here

}

}

1. Read the input the size of the array and the elements in the array. The program then asks the user to enter a particular index and prints the element at that index. Index starts from zero. This program may generate Array Index Out Of Bounds Exception or NumberFormatException. Use exception handling mechanisms to handle this exception.

**Sample Input and Output 1:**

Enter the number of elements in the array : 2

Enter the elements in the array: 50 80

Enter the index of the array element you want to access: 1

The array element at index 1 = 80

**Output:** The array element successfully accessed

**Sample Input and Output 2:**

Enter the number of elements in the array: 2

Enter the elements in the array: 50 80

Enter the index of the array element you want to access: 9

Output

java.lang.ArrayIndexOutOfBoundsException

**Sample Input and Output 3:**

Enter the number of elements in the array: 2

Enter the elements in the array: 30 j

Output:

java.lang.NumberFormatException

import java.util.Scanner;

public class ArrayElementAccess {

public static void main(String[] args) {

write code here

}

}

**1.8 Interfaces**

1. Write a class MathOperation which accepts 5 integers through command line. Create an array using these parameters. Loop through the array and obtain the sum and average of all the elements and display the result. Various exceptions that may arise like ArithmeticException, NumberFormatException, and so on should be handled.

public class MathOperation {

public static void main(String[] args) {

write code here

DONE

}

}

**2. Wrapper Classes & Exception Handling**

**2.1 Wrapper Classes**

1. Generates the minimum and maximum value for each of the Numeric Wrapper classes (Byte, Short, integer, Long, Float, Double)

**Sample Output:**

**Integer range:**

min: -2147483648

max: 2147483647

Double range:

min: 4.9E-324

max: 1.79769313486231578308

Long range:

min: -9223372036854775808

max: 9223372036854775887

**Short range:**

min: -32768

max: 32767

Byte range:

min: -128

max: 127

Float range:

min: 1.4E-45

max: 3.4028235638

public class WrapperMinMaxValues {

public static void main(String[] args) {

DONE

write code here

}

}

1. Receive an integer number as a command line argument, and print the binary, octal and hexadecimal equivalent of the given number.

**Output:**

java Test 20

Given Number :28

Binary equivalent: 10120

Octal equivalent :24

Hexadecimal equivalent :14

public class ConvertNumber {

public static void main(String[] args) {

DONE

write code here

}

}

1. Read an integer number (between 1 and 255) from the user and prints the binary representation of the number. The answer should be printed as a String.

**Note:** The output displayed should contain & digits and should be padded with leading Øs(zeros), in case the returned String contains less than 8 characters.

For example, if the user enters the value 16, then the output should be 80010000 and if the user enters the value 100, the output should be 01100100. You are expected to use Integer class conversion method/s described in the PDF file. Use Scanner class to accept user inputs.

**Hint:** You may use String.format() method for the expected output

import java.util.Scanner;

public class BinaryRepresentation {

DONE

public static void main(String[] args) {

write code here

}

}

1. Create an employee class with properties of your choice. Create an object of this class and also create a clone of the same. After making the clone, change the properties of the original employee object and print the properties of both the original and clone object and note down your observation.

class Employee implements Cloneable {

private String name;

private int age;

private double salary;

public Employee(String name, int age, double salary) {

write code here

DONE

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

@Override

protected Object clone() throws CloneNotSupportedException {

return super.clone();

}

@Override

public String toString() {

write code here

}

}

public class CloneTest {

DONE

public static void main(String[] args) {

write code here

}

}

**2.2 Exceptional Handling**

1. Write a Program to take care of Number Format Exception if user enters values other than integer for calculating average marks of 2 students. The name of the students and marks in 3 subjects are taken from the user while executing the program.

In the same Program write your own Exception classes to take care of Negative values and values out of range (i.e. other than in the range of 8-100)

**Student.java**

class NegativeValueException extends Exception {

public NegativeValueException(String message) {

super(message);

DONE

}

}

**OutOfRangeException.java**

class OutOfRangeException extends Exception {

public OutOfRangeException(String message) {

super(message);

DONE

}

}

**CalculateAverageMarks.java**

import java.util.Scanner;

public class CalculateAverageMarks {

public static void main(String[] args) {

write code here

DONE

}

public static void validateMarks(int marks) throws NegativeValueException, OutOfRangeException {

write code here

}

}

1. A student portal provides user to register their profile. During registration the system needs to validate the user should be located in India. If not the system should throw an exception.

* Create a user defined exception class named "InvalidCountryException".
* Overload the respective constructors.
* Create a main class "UserRegistration", add the following method, void registerUser(String username, String userCountry) with the below implementation if userCountry is not equal to "India" throw a InvalidCountryException with the message "User Outside India cannot be registered"

if userCountry is equal to "India", print the message "User registration done successfully". Invoke the method registerUser from the main method with the data specified and see how the program behaves.

**Input**: Mickey, US

**Output**: Invalid Country (Exception should be thrown.)

**Input**: Mini, India

**Output**: User registration done successfully

**Create Custom Exception Class**

public class InvalidCountryException extends Exception {

public InvalidCountryException() {

super("User Outside India cannot be registered");

}

public InvalidCountryException(String message) {

super(message);

DONE

}

}

**Create Main Class**

public class UserRegistration {

public static void main(String[] args) {

write code here

}

public static void registerUser(String username, String userCountry) throws InvalidCountryException {

DONE

write code here

}

}

1. Write a class MathOperation which accepts 5 integers through command line. Create an array using these parameters. Loop through the array and obtain the sum and average of all the elements and display the result. Various exceptions that may arise like ArithmeticException, NumberFormatException, and so on should be handled.

public class MathOperation {

public static void main(String[] args) {

if (args.length != 5) {

write code here

}

DONE

}

1. Write a Program to take care of Number Format Exception if user enters values other than integer for calculating average marks of 2 students. The name of the students and marks in 3 subjects are taken from the user while executing the program.

import java.util.Scanner;

public class CalculateAverageMarks {

public static void main(String[] args) {

write code here

DONE

}

1. In the same Program write your own Exception classes to take care of Negative values and values out of range (i.e. other than in the range of 0-100)

import java.util.Scanner;

public class CalculateAverageMarks {

public static void main(String[] args) {

write code here

DONE

}

private static int getValidMarks(String prompt, Scanner scanner) throws NumberFormatException, NegativeValueException, OutOfRangeException {

write code here

}

private static void validateMarks(int marks) throws NegativeValueException, OutOfRangeException {

write code here

}

private static double calculateAverage(int marks1, int marks2, int marks3)

{

write code here

}

}

1. Write a program to accept name and age of a person from the command prompt (passed as arguments when you execute the class) and ensure that the age entered is >=18 and < 60.

Display proper error messages. The program must exit gracefully after displaying the error message in case the arguments passed are not proper.

Hint : Create a user defined exception class for handling errors.

public class InvalidAgeException extends Exception {

public InvalidAgeException() {

super("Age must be between 18 and 59");

}

public InvalidAgeException(String message) {

super(message);

}

}

DONE

public class ValidatePerson {

public static void main(String[] args) {

write code here

}

public static void validateAge(int age) throws InvalidAgeException {

write code here

}

}

**3. Input Output (I/O) Streams**

**3.1 I\O Operations**

1. Count the number of times a character appears in a File.

**Note:** The character check is case insensitive... i.e., 'a' and 'A' are considered to be the same

Sample Input and Output:

Enter the file name

Input.txt

Enter the character to be counted

File Input.txt' has 99 instances of letter 'r'.

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

public class CharacterCounter {

public static void main(String[] args) {

write code here

DONE

}

}

private static int countCharacterInFile(String fileName, char charToCount) throws IOException {

write code here

}

}

1. Copy contents from one file to another and check the output.

**Sample Input and Output**:

Enter the input file name

Input.txt

Enter the output file name

Output.txt

File is copied.

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Scanner;

public class FileCopy {

public static void main(String[] args) {

write code here

}

DONE

private static void copyFileContents(String inputFileName, String outputFileName) throws IOException {

write code here

}

}

1. Count the occurrences of each word in an input file and write the word along with its corresponding count in an output file.

**Note:** The words should be sorted alphabetically in the output file

Hint: Consider using Map Collection

For Example, let's assume the following are the contents of inputFile.txt

Manoj works at IARE

Katari works at IARE

Sureka works at IARE

Harish works at IARE

Anitha works at IARE

Janani works at IARE

D:\>Java FilewordCount inputfile.txt outputFile.txt

After Execution of the program the contents of outputFile.txt should be as below

Anitha: 1

Harish: 1

Janani: 1

Katari: 1

Manoj: 1

Sureka: 1

Wipro: 6

at: 6

works: 6

import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Map;

import java.util.TreeMap;

public class WordCount {

public static void main(String[] args) {

write code here

}

private static Map<String, Integer> countWords(String inputFile) throws IOException {

write code here

}

private static void writeWordCounts(String outputFile, Map<String, Integer> wordCountMap) throws IOException {

write code here

}

}

**3.2 Object Serialization**

1. Create a class called Employee with properties name(String),dateOfBirth(java.util.Date), department (String), designation(String) and Salary (double). Create respective getter and setter methods and constructors (no-argument constructor and parameterized constructor) for the same. Create an object of the Employee class and save this object in a file called "data" using serialization. Later using deserialization read this object and print the properties of this object.

Define the Employee Class:

import java.io.\*;

import java.util.Date;

public class Employee implements Serializable {

private static final long serialVersionUID = 1L;

write code here

public Employee() {

}

public Employee(String name, Date dateOfBirth, String department, String designation, double salary) {

write code here

}

DONE

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Date getDateOfBirth() {

return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) {

this.dateOfBirth = dateOfBirth;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

public String getDesignation() {

return designation;

}

public void setDesignation(String designation) {

this.designation = designation;

DONE

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

@Override

public String toString() {

write code here

}

}

**Serialization:**

public class SerializationExample {

public static void main(String[] args) {

write code here

DONE

}

private static void serializeEmployee(Employee employee) {

write code here

}

}

**Deserialization:**

public class DeserializationExample {

public static void main(String[] args) {

write code here

DONE

}

private static Employee deserializeEmployee() {

write code here

}

}

**4. Collections**

**4.1 Lists**

1. Write a Java program to create an ArrayList, add all the months of a year and print the same.

import java.util.ArrayList;

public class MonthsOfYear {

DONE

public static void main(String[] args) {

write code here

}

}

1. Create an application for employee management with the following classes:

a) Create an Employee class with following attributes and behaviors:

* int empid
* String empName
* String email
* String gender
* float salary
* void GetEmployeeDetails() -> prints employee details

b) Create one more class EmployeeDB with the following attributes and behaviors.

* ArrayList list;
* boolean addEmployee (Employee e) -> adds the employee object to the collection
* boolean deleteEmployee(int empid) ->delete the employee object from the collection with the given empid
* String showPayslip(int empid) -> returns the payslip of the employee with the given empid

Provide implementation for all the methods and test your program.

import java.util.ArrayList;

class Employee {

int empid;

String empName;

String email;

String gender;

float salary;

DONE

public Employee(int empid, String empName, String email, String gender, float salary) {

write code here

}

void GetEmployeeDetails() {

write code here

}

}

class EmployeeDB {

DONE

write code here

}

public class EmployeeManagement {

public static void main(String[] args) {

write code here

}

}

1. Create an ArrayList that can store only Strings. Create a printAll method that will print all the elements of the ArrayList using an Iterator.

import java.util.ArrayList;

import java.util.Iterator;

public class StringArrayList {

public static void main(String[] args) {

DONE

write code here

}

public static void printAll(ArrayList<String> list) {

write code here

}

}

1. Create an ArrayList that can store only numbers like int, float, double, etc, but not any other data type.

import java.util.ArrayList;

public class NumberArrayList {

public static void main(String[] args) {

write code here

DONE

}

public static void printAllNumbers(ArrayList<Number> list) {

for (Number number : list) {

System.out.println(number);

}

}

}

1. Implement the assignment 1 using Linked List

import java.util.LinkedList;

public class MonthsOfYearLinkedList {

DONE

public static void main(String[] args) {

write code here

}

}

1. Implement the assignment 1 using Vector

import java.util.Vector;

public class MonthsOfYearVector {

DONE

public static void main(String[] args) {

write code here

}

}

1. Write a program that will have a Vector which is capable of storing Employee objects. Use an Iterator and enumeration to list all the elements of the Vector.

import java.util.Vector;

import java.util.Iterator;

import java.util.Enumeration;

class Employee {

int empid;

String empName;

String email;

String gender;

float salary;

DONE

public Employee(int empid, String empName, String email, String gender, float salary) {

write code here

}

void GetEmployeeDetails() {

write code here

}

}

public class EmployeeVector {

public static void main(String[] args) {

write code here

}

}

**4.2 Sets**

1. Develop a java class with a instance variable H1 (HashSet) add a method saveCountryNames (String CountryName), the method should add the passed country to a HashSet (HI) and return the added HashSet (H1). Develop a method getCountry(String CountryName) which iterates through the HashSet and returns the country if exist else return null.

**Note:** You can test the methods using a main method.

import java.util.HashSet;

import java.util.Iterator;

public class CountryManager {

private HashSet<String> H1;

public CountryManager() {

H1 = new HashSet<>();

}

public HashSet<String> saveCountryNames(String CountryName) {

H1.add(CountryName);

return H1;

}

DONE

public String getCou ntry(String CountryName) {

write code here

}

public static void main(String[] args) {

write code here

}

}

1. Write a program to store a group of employee names into a HashSet, retrieve the elements one by one using an Iterator.

import java.util.HashSet;

import java.util.Iterator;

public class EmployeeNames {

public static void main(String[] args) {

DONE

write code here

}

}

1. Create a Collection called TreeSet which is capable of storing String objects. Then try these following operations:
   * + Reverse the elements of the Collection.
     + Iterate the elements of the TreeSet using Iterator.
     + Check if a particular element exists or not.

import java.util.TreeSet;

import java.util.Iterator;

import java.util.NavigableSet;

DONE

public class TreeSetExample {

public static void main(String[] args) {

write code here

}

}

1. Implement the assignment 1 using TreeSet

import java.util.TreeSet;

import java.util.Iterator;

class Employee implements Comparable<Employee> {

int empid;

String empName;

String email;

String gender;

float salary;

public Employee(int empid, String empName, String email, String gender, float salary) {

write code here

}

void GetEmployeeDetails() {

System.out.println("Employee ID: " + empid);

System.out.println("Employee Name: " + empName);

System.out.println("Email: " + email);

System.out.println("Gender: " + gender);

System.out.println("Salary: " + salary);

}

@Override

public int compareTo(Employee other) {

return Integer.compare(this.empid, other.empid);

}

}

public class EmployeeTreeSet {

public static void main(String[] args) {

write code here

}

}

**4.3 Maps**

1. Develop a java class with a instance variable M1 (HashMap) create a method saveCountryCapital (String CountryName, String capital), the method should add the passed country and capital as key/value in the map M1 and return the Map (M1).

Key- Country India

Value Capital

Delhi

Japan

Tokyo

* Develop a method **getCapital (String CountryName)** which returns the capital for the country passed, from the Map MI created in step 1.
* Develop a method **getCountry (String capitalName)** which returns the country for the capital name, passed from the Map M1 created in step 1.
* Develop a method which iterates through the map Mi and creates another map M2 with Capital as the key and value as Country and returns the Map M2.

Key - Capital Delhi

Value Country India

Tokyo

Japan

Develop a method which iterates through the map M1 and creates an ArrayList with all the Country names stored as keys. This method should return the ArrayList.

**Note:** You can test the methods using a main method.

import java.util.HashMap;

import java.util.ArrayList;

import java.util.Map;

import java.util.Iterator;

public class CountryCapitalManager {

private HashMap<String, String> M1;

public CountryCapitalManager() {

M1 = new HashMap<>();

}

public HashMap<String, String> saveCountryCapital(String countryName, String capital) {

DONE

M1.put(countryName, capital);

return M1;

}

public String getCapital(String countryName) {

return M1.get(countryName);

}

public String getCountry(String capitalName) {

write code here

}

public HashMap<String, String> createCapitalCountryMap() {

write code here

}

public ArrayList<String> getAllCountries() {

write code here

}

public static void main(String[] args) {

write code here

}

}

1. Create a Collection called HashMap which is capable of storing String objects. The program should have the following abilities
   * Check if a particular key exists or not.
   * Check if a particular value exists or not.
   * Use Iterator to loop through the map.

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

DONE

public class HashMapOperations {

public static void main(String[] args) {

write code here

}

}

1. Write a program that will have a Properties class object which is capable of storing some States of India and their Capital. Use an Iterator to list all the elements stored in the Properties.

import java.util.Properties;

import java.util.Iterator;

import java.util.Set;

public class StateCapitalProperties {

public static void main(String[] args) {

write code here

}

}

1. Create a Collection "ContactList" using HashMap to store name and phone number of contacts added.

The program should use appropriate generics (String, Integer) and have the following abilities:

* + Check if a particular key exists or not.
  + Check if a particular value exists or not.
  + Use Iterator to loop through the map.

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

public class ContactListProgram {

public static void main(String[] args) {

write code here

}

}

1. Implement the assignment 1 using TreeMap

import java.util.TreeMap;

import java.util.ArrayList;

import java.util.Map;

import java.util.Iterator;

public class CountryCapitalManager {

private TreeMap<String, String> M1;

public CountryCapitalManager() {

M1 = new TreeMap<>();

}

public TreeMap<String, String> saveCountryCapital(String countryName, String capital) {

M1.put(countryName, capital);

return M1;

}

public String getCapital(String countryName) {

return M1.get(countryName);

}

public String getCountry(String capitalName) {

write code here

}

public TreeMap<String, String> createCapitalCountryMap() {

write code here

}

public ArrayList<String> getAllCountries() {

write code here

}

public static void main(String[] args) {

write code here

}

}

1. Implement the assignment 1 using Hash Table

import java.util.Hashtable;

import java.util.ArrayList;

import java.util.Enumeration;

public class CountryCapitalManager {

private Hashtable<String, String> M1;

public CountryCapitalManager() {

M1 = new Hashtable<>();

}

public Hashtable<String, String> saveCountryCapital(String countryName, String capital) {

M1.put(countryName, capital);

return M1;

}

public String getCapital(String countryName) {

return M1.get(countryName);

}

public String getCountry(String capitalName) {

write code here

}

public Hashtable<String, String> createCapitalCountryMap() {

write code here

}

public ArrayList<String> getAllCountries() {

ArrayList<String> countriesList = new ArrayList<>(M1.keySet());

return countriesList;

}

public static void main(String[] args) {

write code here

}

}

**5. JUnit**

**5.1 JUnit With Eclipse**

1. Create a Project named JunitLearning

* Create a package named com.wipro.task
* Copy the given class into the com.wipro.task package package com.wipro.task;

import java.util.Arrays;

public class DailyTasks {

public String doStringConcat (String s1, String s2) {

return s1+" "+52;

}

public int[] sortValues(int arr[]){

Arrays.sort(arr);

return arr;

}

public boolean checkPresence(String str, String a) {

return str.contains(a);

}

* Create a new package called com.wipro.test;
* Create a class named TestStringConcat to test the functionality of doStringConcat method [hint: use assertEquals method]

package com.wipro.task;

import java.util.Arrays;

public class DailyTasks {

public String doStringConcat(String s1, String s2) {

return s1 + " " + s2;

}

public int[] sortValues(int arr[]) {

Arrays.sort(arr);

return arr;

}

public boolean checkPresence(String str, String a) {

return str.contains(a);

}

}

package com.wipro.test;

import com.wipro.task.DailyTasks;

import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class TestStringConcat {

@Test

public void testDoStringConcat() {

DailyTasks tasks = new DailyTasks();

String result = tasks.doStringConcat("Hello", "World");

assertEquals("Hello World", result);

}

}

**5.2 Assert Methods and Annotations**

1. Create a class named TestSort to test the functionality of the sortValues method [hint: use assertArrayEquals method]

package com.wipro.test;

import com.wipro.task.DailyTasks;

import static org.junit.Assert.assertArrayEquals;

import org.junit.Test;

public class TestSort {

@Test

public void testSortValues() {

DailyTasks tasks = new DailyTasks();

int[] inputArray = {5, 3, 8, 1, 2};

int[] expectedArray = {1, 2, 3, 5, 8};

int[] resultArray = tasks.sortValues(inputArray);

assertArrayEquals(expectedArray, resultArray);

}

}

1. Create a class named TestCheckPresence to test the functionality of sortValues method [hint: use assertTrue and assertFalse methods]

package com.wipro.test;

import com.wipro.task.DailyTasks;

import static org.junit.Assert.assertTrue;

import static org.junit.Assert.assertFalse;

import org.junit.Test;

public class TestCheckPresence {

@Test

public void testCheckPresenceTrue() {

DailyTasks tasks = new DailyTasks();

boolean result = tasks.checkPresence("Hello World", "World");

assertTrue(result);

}

@Test

public void testCheckPresenceFalse() {

DailyTasks tasks = new DailyTasks();

boolean result = tasks.checkPresence("Hello World", "Universe");

assertFalse(result);

}

}

**5.3 Test Suite**

1. Create a test suite for all the 3 classes created under the com.wipro.test package and execute the same

package com.wipro.test;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

@SuiteClasses({

TestStringConcat.class,

TestSort.class,

TestCheckPresence.class

})

public class AllTests {

// This class remains empty. It is used only as a holder for the above annotations.

}

**6. Multi-Threading**

**6.1 Thread Creation**

1. Create two threads and assign names Scooby' and 'Shaggy to the two threads. Display both thread names.

**Step 1:** Create a Class that Implements Runnable

public class Task implements Runnable {

@Override

public void run() {

write code here

}

}

**Step 2:** Create and Start the Threads

public class Main {

public static void main(String[] args) {

write code here

}

}

1. store colours in the form of an array

String colours[]={"white", "blue", "black ", "green", "red", "yellow");

display all colours repeatedly by generating colour index from Random class. If the randos colour index matches to red stop display. Note: perform this task by implementing Runnabe interface

import java.util.Random;

public class ColourTask implements Runnable {

private String[] colours = {"white", "blue", "black", "green", "red", "yellow"};

private Random random = new Random();

@Override

public void run() {

write code here

}

}

public static void main(String[] args) {

write code here

}

}

**6.2 Thread Control and Priorities**

1. Create a thread which prints 1 to 10. After printing 5, there should be a delay of 5000 milliseconds before printing 6

public class NumberPrinter implements Runnable {

@Override

public void run() {

write code here

}

}

public static void main(String[] args) {

write code here

}

}

1. Create two threads, one thread to display all even numbers between 1 & 20, another to display odd numbers between 1 & 20.

Note: Display all even numbers followed by odd numbers

Hint: use join.

**EvenNumbers Class**

public class EvenNumbers implements Runnable {

@Override

public void run() {

write code here

}

}

**OddNumbers Class**

public class OddNumbers implements Runnable {

@Override

public void run() {

write code here

}

}

**Main Class**

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create three threads with different priorities MAX, MIN, NORM- and start the threads at the same time. Observe the completion of the threads.

class PriorityThread extends Thread {

public PriorityThread(String name) {

super(name);

}

public void run() {

write code here

}

}

public class ThreadPriorityDemo {

public static void main(String[] args) {

write code here

}

}

**7. RDBMS /SQL /JDBC**

**7.1 Select Statement**

1. Determine the structure of the DEPARTMENTS table and its contents.

Select \* from departments

1. Create a query to display the last\_name, job\_ID, hire\_date, and employee\_ID for each employee, with the employee\_ID appearing first. Provide an alias STARTDATE for the HIRE DATE column.

SELECT employee\_id AS "Employee ID",

last\_name AS "Last Name",

job\_id AS "Job ID",

hire\_date AS STARTDATE

FROM employees;

Select employeeId,lastname,jobid,hiredate as startdtae from employees;

1. Create a query to display all unique job IDs from the EMPLOYEES table.

Select distinct jobid from employees

1. Create a query to display employeeid, last\_name, jobid and hiredate from employee table with more describing column names. Name the column headings Emp# Employee, Job and Hire\_Date respectively.

Select employeeid Emp#, last\_name Employee, jobid job ,hiredate hire\_date from employees

**7.2 Restricting and Sorting Data**

1. Create a report that displays the lastname and salary of employees who earn more than $12,00.

Select lastname,salary from employees where salary>1200;

1. Create a report that displays the last name and department number for employee number 176.

Select lastname, deptnumber from employee where employeeid=176;

1. To find high-salary and low-salary employees. Create a query to display the last name and salary for any employee whose salary is not in the range of $5,000 το $12,000

Select max(salary) from employees;

Select min(salary) from employees;

Select lastname,salary from employees where salary not between 5000 and 12000;

1. Create a report to display the last name, job id, and hire date for employees with the last names of Matos and Taylor. Order the query in ascending order by the hire date.

Select last\_name,job\_id,hire\_date from employees where lastname=’matos’ or lastname=’taylor’ order by hire\_date asc;

1. Display the last name and department ID of all employees in departments 20 or 58 in ascending alphabetical order by name.

Select deptid,lastname from employees where deptid=20 or deptid=58

Order by lastname asc;

1. List employees who earn between $5,000 and $12,000, and are in department 20 or 50. Label the columns as Employee and Monthly Salary, respectively.

Select lastname employee,salary monthlysalary from employees where salary between 5000 and 12000 and departmentid in (20,50);

1. Create a report that displays the last name and hire date for all employees who were hired in 1994.

Select lastname ,hiredate from employees where to\_char(hiredate,’yyyy’)=1994

1. Create a report to display the last name and job title of all employees who do not have a manager.

Select e1. lastname,e1.job from employees e1 left join employees e2 on e1.mgrid=e2.id

Where e2.id is null;

1. Create a report to display the last name, salary, and commission of all employees who earn commissions.

Sort data in descending order based on salary and commissions. Use the column's numeric position in the ORDER BY clause.

Select lastname,salary,commission from employees where commission is not null

Order by 2 desc,3 desc;

1. Create a report that displays the last name and salary of employees who earn more than an amount

that the user specifies after a prompt. If you enter 12000, it should display all employees earning more than 12000,

Eg: Salary\_value: 12000.

Select last\_name,salary from employees where salary>&salary;

1. Create a query that prompts the user for a manager ID and generates the employee ID, last name,

salary and department for that manager's employees and prompts a column name by which result should be sorted.

Eg:

manager\_id103

sorted\_by last\_name

select employeeid,last\_name,salary dept from employees where mgrid=&managerid\_no order by &column\_name

1. Display all employee last names in which the third letter of the name is "a".

Select \* from employees where lastname like ‘\_\_\_a%’.

1. Display the last names of all employees who have both an "a" and an "e" in their last name.

Select \* from employees where lastname like ‘%[ae]%’;

1. Display the last name, job, and salary for all employees whose jobs are either those of a sales representative or of a stock clerk, and whose salaries are not equal to $2,500, $3,500, or $7,000.

Select last\_name,job,salary from employees where job in(‘sales representative’,’stock clerk’) and

Salary not in(2,500,3500,7000)

**7.3 Data Modification Language (DML)**

1. Create table MY\_EMPLOYEE as Select employee\_id, first\_name, last\_name, department\_id, e, last\_name, department\_id, salary from EMPLOYEES where 1=2;

It creates my\_employee table with employees columns names and data types

1. Test the table creation by viewing the structure using describe command

|  |  |  |
| --- | --- | --- |
| **Name** | **Null Type** | **Type** |
| EMPLOYEE ID  FIRST NAME  LAST NAME  DEPARTMENT\_ID  SALARY | NOT NULL | NUMBER (6)  VARCHAR2(28)  VARCHAR2(25)  NUMBER(4)  NUMBER (8,2) |

5 rows selected

Create table employees(employeeid NUMBER (6),first\_name VARCHAR2(28),last\_name VARCHAR2(25),

DEPARTMENT\_ID NUMBER(4),salary NUMBER (8,2)) Desc employees

1. Insert one record without listing the column names in the insert statement. Check whether data is inserted

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **employee\_id** | **first name** | **last\_name** | **department\_id** | **Salary** |
| 201 | Michael | Harstein | 20 | 13000 |

Insert into employees values(201,’machael’,’Harstein’,20,13000)

1. Insert one record without listing the column names in the insert statement where salary value remain undetermined. Check whether data is inserted

Eg.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **employee\_id** | **first\_name** | **last\_name** | **department\_id** | **salary** |
| 201 | Michael | Hartstein | 20 | 13000 |
| 202 | Pat | Fay | 20 | NULL |

Insert into employees values(202,’pat’,’Fay’,20,null)

1. Use the above Script to insert the below given records

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| employee\_id | first\_name | last\_name | department\_id | salary |
| 205 | Shelley | Higgins | 110 | 12000 |
| 100 | Steven | King | 90 | 24000 |
| 101 | Neena | Kochhar | 90 | 17000 |
| 102 | Lex De | Haan | 90 | 17000 |
| 111 | Ismael | Sciarra | 100 | 7700 |
| 112 | Jose | Manuel Urma | 100 | 7800 |
| 204 | Hermann | Baer | 70 | 10000 |

Insert into employees values(2005,’shelley’,’higgins’,110,12000), (100,’steven’,’king’,90,24000), (101,’Neena’,’kochhar’,90,17000), (102,’Lex Den’,’Haan’,90,17000), (111,’ismael’,’sciarra’,100,7700), (112,’jose’,’Manuel urma’,100,7800), (204,’Hermann’,’Baer’,70,10000),

1. Create a query to increase salary by 10% for all employees in dept 90.

Update employees set salary =salary+(salary/100)\*10 where deptno=90

1. Create a query to update Last\_name of emp 282 to Higgins.

Update employees set empid=282 where lastname=’Higgins’

1. Delete employees whose name either first or last name has char seq of "man"

Delete from employees where lastname like ‘%man%’ or firstname like ‘%man%’;

1. Create the DEPT table based on the following table instance chart. Save the statement in a script called lab\_10\_01.sql , and then execute the statement in the script to create the table. Confirm that the table is created.

Specification Values:

Column named Dept\_ID of Numeric 7 size and would be a primary key.

Column named Dept\_Name of varchar2 size 20.

Create table Dept(Dept\_ID number(7) primary key,Dept\_name varchar(20))

**7.4 Data Defination Language (DDL)**

1. Populate the DEPT table with data from the DEPARTMENTS table. Include only columns that you need.

Insert dept Id 10 and Name Accounts

Insert dept Id as null and Name as TT

Correct by giving 20 and TT

Insert A1 as Id and Accounts

Correct by giving 30 and Accounts

Insert into Dept values(10,’Accounts’)

Insert into dept values(null,’TT’);

Update set dept\_id=20 where Dept\_name=’TT’

Update dept set dept\_id=30 where dept\_name=’Accounts’

1. Create the EMP table based on the following table instance chart. Save the statement in a script called lab\_10\_03.sql , and then execute the statement in the script to create the table. Confirm that the table is created.

Specification- Values

Column Name: ID, LAST\_NAME, FIRST\_NAME, DEPT\_ID

Key Type: PK, -, -, FK

Nulls /Unique: -, Not null, -, -,

FK Table: -, -, -, Dept

FK Column: -, -, -, ID

Data type: NUMBER, VARCHAR2, VARCHAR2, NUMBER

Length: 7, 25, 25, 7

Insert 101,Sam,Sundar,10

Insert 101,Ram,Krishna,20

Insert 102,Gopi,null,40

Insert 103,null,ram,20

Create table emp(id number(7) primary key, last\_name varchar2(25) not null,

First\_name varchar2(25),dept\_id number(7))

Create table dept(id number(7) references emp(dept\_id))

Insert into emp values(101,’sam’,’sundar’,10);

Insert into emp values(101,’ram’,’krishna’,20); ---it will not accept because of primary key

Insert into emp  
  
values(102,’gopi’,null,40);

Insert into emp values(103,null,’ram’,20); ---it will not accept because of not null constraint on last\_name

1. Write a java program that establishes a connection to oracle database successfully. If the connection is successful, it should display a message “Connection Established successfully”. In case, it is not able to do so due to any exception, it should display the message “Connection could not be established “. If there is an exception, it should also display the description of the exception.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class OracleDBConnection {

public static void main(String[] args) {

write code here

}

}

**7.5 Establishing Connections**

1. After establishing the connection successfully, exclude the registration process(by commenting the line containing the code Class.forName(“..”)). Observe the result.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class OracleDBConnection {

public static void main(String[] args) {

write code here

}

}

1. Write a java program that connects to oracle database, queries the inbuilt table “emp” and displays the first two columns (empno using column index and ename using column name ) of all the rows.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import java.sql.SQLException;

public class OracleDBQuery {

public static void main(String[] args) {

write code here

}

}

**7.6 Executing Query and Processing Results**

1. Modify the above program to display all the rows where sal is greater than 1000 and less than 2000. Display the columns ename, job, sal and comm.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class DisplayEmployeeRecords {

public static void main(String[] args) {

write code here

}

}

1. Develop a jdbc program containing main method, which should instantiate a class called DAOClass, which should contain methods called insert, delete, modify and display. Description of what each of these methods are expected to do is given below. Necessary details required for executing these methods, are passed from command line argument. For e.g. If the name of the class containing the main method is JDBCCalls, then if you want to insert a record, you will execute this class as java **JDBCCalls 1 101 “Ajit” “IV” “20-Nov-2001” 4000** Where 1 is the option for inserting the record and all other details are the values for the columns in each row of the student table. The structure of student table is given below. Similarly, for deleting a record, you have to execute the code as java **JDBCCalls 2 101** where 2 is the option for deleting a record and 101 is the rollno of the student, whose record has to be deleted. For modifying a record, you will use java **JDBCCalls 3 101 4500**, where 3 is the option for modifying a record and the 4500 is the new fee which needs to replace the old fee value. For Displaying records, if the main class is executed as follows java JDBCCalls 4 101 it should display only one record, that of the student with roll no. 101. 4 option is for displaying the record. If the main class is executed as java JDBCCalls 4 (without specifying the rollno.), it means that details of all the students should be displayed.

**DAOClass.java**

import java.sql.\*;

import java.util.\*;

public class DAOClass {

private Connection connect() {

write code here

}

public void insert(int rollno, String name, String standard, String dob, int fee) {

write code here

}

public void delete(int rollno) {

write code here

}

public void modify(int rollno, int fee) {

write code here

}

public void display(int rollno) {

write code here

}

public void displayAll() {

write code here

}

}

**JDBCCalls.java**

public class JDBCCalls {

public static void main(String[] args) {

write code here

}

}

**7.7** [**Using PreparedStatement & MetaData objects**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T10)

1. **Inserting a record**

ABC International School wants to computerize students details. The school maintains a database of students in Oracle. The student table contains information related to students and is shown in the following student table structure.

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Type** | **Constraint** |
| Rollno | Number (4) | Primary Key |
| StudentName | Varchar (20) | Not Null |
| Standard | Varchar (2) | Not Null |
| Date\_Of\_Birth | Date |  |
| Fees | Number (9,2) |  |

When a new student joins the school, the student record is inserted in the student table.

The valid student details are as follows:

* + **Rollno:** Valid value is a 4-digit number
  + **StudentName:** Valid value can contain maximum 20 letters in uppercase
  + **Standard:** Valid values are Roman Letters representing I to X(I, II, III, IV….IX, X)

Write a Java program to insert some records to the table

**Step 1:** **Prepare Your Database**

Ensure your Oracle database is set up with a table named student that matches the structure you provided:

CREATE TABLE student (

Rollno NUMBER(4) PRIMARY KEY,

StudentName VARCHAR2(20) NOT NULL,

Standard VARCHAR2(2) NOT NULL,

Date\_Of\_Birth DATE,

Fees NUMBER(9,2)

);

**Step 2:** **JDBC Setup**

Include the Oracle JDBC driver (ojdbc.jar) in your project's classpath. If you're using Maven, you can add the dependency in your pom.xml:

<dependency>

<groupId>com.oracle.database.jdbc</groupId>

<artifactId>ojdbc8</artifactId>

<version>19.3.0.0</version>

</dependency>

**Step 3:** **Java Program to Insert Records**

Write a Java program that connects to your Oracle database using JDBC, inserts records into the student table, and handles valid student details:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class InsertStudentRecords {

public static void main(String[] args) {

write code here

}

private static void insertStudent(PreparedStatement stmt, int rollno, String name, String standard, String dob, double fees) throws SQLException {

write code here

}

}

1. **Deleting a Student’s record**

When a student leaves the school, the record related to that student needs to be deleted from the Student table. The student’s roll no, whose record has to be deleted, should be passed as a command line argument. Upon deletion, the Student details must be stored in another table named StudentLog which will maintain the details such as Rollno, StudentName, Standard and Leaving\_date.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.sql.ResultSet;

import java.sql.Timestamp;

import java.util.Date;

public class DeleteStudentRecord {

public static void main(String[] args) {

write code here

}

}

1. **Modification of Student record**

When there is a change in the fee to be paid by a student, the respective row should be appropriately updated. Pass the rollno from the command prompt along with the new fee amount and this amount should be reflected in the table for that particular student.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

public class UpdateStudentFee {

public static void main(String[] args) {

write code here

}

}

1. **Display Student details**

Write the code to display details of all the students, if no roll no. is passed, while executing the main program.

If while executing the main program, the roll no. is passed, then it should display the record of that particular student.

**Step 1:** **Define the Student Class**

Create a Student class with fields to represent student details like roll number, name, course, and marks.

public class Student {

private int rollNumber;

private String name;

private String course;

private double marks;

public Student(int rollNumber, String name, String course, double marks) {

this.rollNumber = rollNumber;

this.name = name;

this.course = course;

this.marks = marks;

}

public int getRollNumber() {

return rollNumber;

}

public String getName() {

return name;

}

public String getCourse() {

return course;

}

public double getMarks() {

return marks;

}

@Override

public String toString() {

write code here

}

}

**Step 2:** **Create a Data Source (Simulated)**

For the purpose of this example, we'll simulate a data source containing student records.

import java.util.ArrayList;

import java.util.List;

public class DataSource {

private static List<Student> students = new ArrayList<>();

write code here

}

public static List<Student> getAllStudents() {

return students;

}

public static Student getStudentByRollNumber(int rollNumber) {

write code here

}

}

**Step 3:** **Main Program to Display Student Details**

Write a Main class that handles the main logic to display student details based on the command-line arguments.

public class Main {

public static void main(String[] args) {

write code here

}

}

**7.8** [**Using CallableStatement and Transactions**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T11)

1. Create a stored procedure that calculates net salary of all the employees whose records are stored in table "emp".

The criteria for calculating net salary is as follows :

Gross salary = sal + comm.

Net Salary = gross salary - IT

If the employee's commission is null then IT is calculated as

IT = 10% of gross salary

else if the employees commission is less than 500, then IT is calculated as

IT = 15% of gross salary

else

IT = 20% of gross salary.

Develop a jdbc program that invokes this stored procedure by passing the empno. and in return gets the net salary of each employee. Display on screen the empno., ename and net salary of all the employees.

**Step 1:** Create the Stored Procedure

Here’s the SQL code to create the stored procedure in your database (assuming MySQL syntax):

CREATE PROCEDURE CalculateNetSalary(IN emp\_no INT)

BEGIN

DECLARE gross\_salary DECIMAL(10, 2);

DECLARE net\_salary DECIMAL(10, 2);

DECLARE IT DECIMAL(10, 2);

DECLARE commission DECIMAL(10, 2);

-- Get the salary and commission of the employee

SELECT sal, comm INTO gross\_salary, commission

FROM emp

WHERE empno = emp\_no;

-- Calculate gross salary

SET gross\_salary = IFNULL(gross\_salary, 0) + IFNULL(commission, 0);

-- Calculate IT based on commission

IF commission IS NULL THEN

SET IT = 0.10 \* gross\_salary; -- 10% of gross salary

ELSEIF commission < 500 THEN

SET IT = 0.15 \* gross\_salary; -- 15% of gross salary

ELSE

SET IT = 0.20 \* gross\_salary; -- 20% of gross salary

END IF;

-- Calculate net salary

SET net\_salary = gross\_salary - IT;

-- Output the result

SELECT empno, ename, net\_salary AS net\_salary

FROM emp

WHERE empno = emp\_no;

**Step 2:** Develop the JDBC Program

Now, let's write a Java program using JDBC to invoke this stored procedure and display the results:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.CallableStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class CalculateNetSalary {

public static void main(String[] args) {

write code here

}

}

**8. Hyper Text Markup Language (HTML)**

**8.1** [**Layout Tags and Semantic Tags**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T2)

1. Create a webpage to display your name with every letter having a different heading size.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Display Name with Different Heading Sizes</title>

<style>

body {

write code here

}

</style>

</head>

<body>

<div class="name-container">

<h1>A</h1>

<h2>B</h2>

<h3>C</h3>

<h4>D</h4>

<h5>E</h5>

<h6>F</h6>

</div>

</body>

</html>

1. Create a webpage that displays two lists with any information you want. One list should be an ordered list and the other list should be an unordered list.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Ordered and Unordered Lists</title>

<style>

body {

write code here

}

.container {

write code here

}

h2 {

text-align: center;

}

</style>

</head>

<body>

<div class="container">

<h2>My Favorite Movies</h2>

<ol>

write code here

</ol>

<h2>My Favorite Hobbies</h2>

<ul>

write code here

</ul>

</div>

</body>

</html>

1. We need to have a HTML containing three lists.

List 1 will contain the following :

1) One

a. Uno

b. Hanna

2) Two

a. Dos

b. Dool

3) Three

a. Tres

b. Set

List 2 will contain:

1) Chevy

1. Cavilier
2. Malibu

2) Ford

1. Escort
2. F100

3) Mazda

626

List 3 should be of two types : Fruits and Vegetables

A. Fruits

i. Apples,Oranges,Pears,Banana

B. Vegetables

i. Corn

a)On the cob

b)Cream

ii. Carrots

iii. Green Beans

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Three Lists</title>

<style>

body {

write code here

}

.container {

write code here

}

h2 {

text-align: center;

}

ul, ol {

margin-left: 20px;

}

ol ul, ul ul {

list-style-type: none;

}

</style>

</head>

<body>

<div class="container">

<h2>List 1</h2>

<ol>

write code here

</ol>

<h2>List 2</h2>

<ul>

write code here

</ul>

<h2>List 3</h2>

<ol>

write code here

</ol>

</div>

</body>

</html>

1. Create a page that will display an image that has a border of size 2, a width of 200, and a height of 200.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Image with Border</title>

<style>

body {

write code here

}

.image-container {

write code here

}

.image-container img {

write code here

}

</style>

</head>

<body>

<div class="image-container">

<img src="https://via.placeholder.com/200" alt="Sample Image">

</div>

</body>

</html>

1. Create a page which contains two links- one at the top and one at the bottom. On clicking the top link, the page has to scroll down to the bottom of the page where bottom link is present. On clicking the bottom link, the page has to scroll up to the top of the page.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Scroll Links</title>

<style>

body {

write code here

}

.top-link, .bottom-link {

write code here

}

.bottom-link {

margin-top: auto;

}

</style>

</head>

<body>

write code here

</body>

**8.2** [**Tables, Forms and Frames**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T3)

1. Write a HTML program to display the table having the columns(highlighted) as Decimal ,English, Spanish, Korean, Binary. Following are the data to be in each rows respective of their columns :

**Decimals:** {1,2,3,4,5}

**English:** {one,two,three,four,five}

**Spanish:** {uno,dos,tres,quarto,cinco}

**Korean:** {hanna,dool,set,net,daset}

**Binary:** {0001,0010,0011,0100,0101}

Each element is the data which will be our rows for the corresponding columns

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Multilingual Number Table</title>

<style>

body {

write code here

}

table {

write code here

}

th, td {

write code here

}

th {

write code here

}

tr:nth-child(even) {

background-color: #f9f9f9;

}

</style>

</head>

<body>

<h2>Number Translation Table</h2>

<table>

write code here

</table>

1. Write a HTML program to create visitor information capture. The page should have the heading “Visitors information” and should ask the following information from the user :

* Email address using a textbox.
* User’s interest web for research or purchase as check box.
* Occupation as drop down box and preselected value should be given as (select one).
* Check “want us to send you junk emails?” and the options given are “absolutely” and “no way man” using the radio button.
* A horizontal line divides the heading and the body content. Once all the inputs are given, the user should be able to submit or reset.
* This whole form should be tabulated using table tag.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Visitor Information</title>

<style>

body {

write code here

}

.container {

write code here

}

h2 {

text-align: center;

}

table {

write code here

}

td {

padding: 10px;

vertical-align: top;

}

th {

text-align: left;

padding: 10px;

}

hr {

margin: 20px 0;

}

.form-actions {

text-align: center;

}

</style>

</head>

<body>

<div class="container">

<h2>Visitor Information</h2>

<hr>

<form>

write code here

</form>

</div>

</body>

</html>

1. Write a HTML program to create an online registration form which captures the following data:

Username, Password, Confirm password, Name using text boxes.

* Gender by radio buttons.
* Address should be taken by multiple line inputs from the user.

Finally the Profession should be provided by selecting from the drop down box. Make sure the contents of the page is aligned in the middle.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Online Registration Form</title>

<style>

body {

write code here

}

.container {

write code here

}

h2 {

text-align: center;

}

table {

width: 100%;

border-collapse: collapse;

}

td {

padding: 10px;

vertical-align: top;

}

.form-actions {

text-align: center;

}

</style>

</head>

<body>

<div class="container">

<h2>Registration Form</h2>

<form>

write code here

</form>

</div>

</body>

</html>

1. Write a HTML program to display the HTML page as described below:

* Create a page which is divided in 3 parts. 30% horizontally on top and the rest 70 % bellow is split vertically in 2 parts again 30 % and 70% each
* The Top header frame should display an image
* Left side below frame should have a menu of “Font” and “Alignment”.
* If we click on “Font” link a different page which contains 5 different fonts with font’s names should be shown on the RHS
* If we click on “Alignment” link a different page which contains a different alignment type should be shown on the RHS frame.
* For this create 2 webpages named Font and Alignment.
* Font page should have 5 names of different fonts displayed like this

Agency FB

Cambria

Elephant

##### Alignment page should have sample text aligned to different alignments

**Main Page (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<frameset rows="30%,70%">

< write code here

</frameset>

</html>

**Header Page (header.html)**

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<body>

<img src="your-image-url.jpg" alt="Header Image" style="width:100%; height:100%;">

</body>

</html>

**Menu Page (menu.html)**

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<body>

write code here

</body>

</html>

**Font Page (font.html)**

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<body>

write code here

</body>

</html>

Alignment Page (alignment.html)

html

Copy code

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<body>

write code here

</body>

</html>

Modern Method (Using HTML and CSS)

**Main Page (index.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Three Part Layout</title>

<style>

body, html {

write code here

}

.header {

write code here

}

.content {

write code here

}

.menu {

write code here

}

.main-content {

grid-column: 2 / 3;

padding: 20px;

}

iframe {

width: 100%;

height: 100%;

border: none;

}

</style>

</head>

<body>

<div class="header">

<img src="your-image-url.jpg" alt="Header Image" style="width:100%; height:100%;">

</div>

<div class="content">

write code here

</div>

</body>

</html>

**Font Page (font.html)**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Font Page</title>

</head>

<body>

write code here

</body>

</html>

**Alignment Page (alignment.html)**

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<body>

write code here

</body>

</html>

**8.3** [**Style and div tags**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T4)

1. Apply the style, font: Times New Roman in the Hands-on Assignment 1 on Tables

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Number Table</title>

<style>

body {

write code here

}

table {

write code here

}

th, td {

write code here

}

th {

background-color: #f2f2f2;

font-weight: bold;

}

</style>

</head>

<body>

<table>

write code here

</table>

</body>

</html>

1. Group the Hands-on Assignment 2 and 3 on Forms to a single web page providing separate div tags for each forms.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Forms</title>

<style>

body {

write code here

}

.container {

write code here

}

h2 {

text-align: center;

}

table {

width: 100%;

border-collapse: collapse;

}

td {

padding: 10px;

vertical-align: top;

}

.form-actions {

text-align: center;

}

.separator {

margin: 20px 0;

border: none;

height: 2px;

background-color: #ccc;

}

</style>

</head>

<body>

<div class="container">

write code here

</div>

<div class="container">

write code here

</div>

</body>

</html>

**8.4** [**Introduction to HTML5**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T5)

1. Apply the HTML 5 attributes in the Hands-on Assignment 2 and 3 on Forms

Sample Attributes : email, placeholder, required, multiple.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Forms</title>

<style>

body {

write code here

}

.container {

write code here

}

h2 {

text-align: center;

}

table {

width: 100%;

border-collapse: collapse;

}

td {

padding: 10px;

vertical-align: top;

}

.form-actions {

text-align: center;

}

.separator {

write code here

}

</style>

</head>

<body>

<div class="container">

write code here

</div>

<div class="container">

write code here

</div>

</body>

</html>

**9. Cascading Style Sheets (CSS)/JAVASCRIPT**

**9.1** [**Styles and Style Sheets**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T2)

1. Create a webpage that displays the div element with the following properties.

* Alignment: Center
* Font Size : 27pixels
* Font Name: Arial

Note: Use CSS Id selector

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Centered Div</title>

<style>

#centered-div {

write code here

}

</style>

</head>

<body>

<div id="centered-div">

This is a centered div with 27px Arial font.

</div>

</body>

</html>

1. Create a webpage that displays the div element with the following properties

* Alignment: Center
* Font Size : 25pixels
* Font Name: Verdana

Note: Use CSS class selector

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Centered Div with CSS Class</title>

<style>

.centered-div {

write code here

}

</style>

</head>

<body>

<div class="centered-div">

This is a centered div with 25px Verdana font.

</div>

</body>

</html>

1. You should write an external CSS file with the following attributes for the body tag and you need to link this CSS file to your HTML file.

* Font size: 24pixels
* Body background: #00BFFF
* The text should be displayed in “Verdana” font.

**STEP 1:** Write CSS Code:

Open styles.css and add the following CSS rules:

body {

font-size: 24px;

font-family: Verdana, sans-serif;

background-color: #00BFFF;

}

CSS File to HTML File:

<!DOCTYPE html>

<html lang="en">

<head>

write code here

</head>

<body>

<!-- Content of your webpage goes here -->

</body>

</html>

1. Create a webpage that displays the div element with the following attributes.

* Font name: Verdana
* Font size: 20 pixels
* Color: red
* Align: center

Note: Use Inline style sheet

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Div with Inline CSS</title>

</head>

<body>

write code here

</body>

</html>

**9.2**[**Formatting with CSS**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T3)

1. Create a webpage that displays an image as the background of the page and set the background image position to right top.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Background Image Example</title>

<style>

body {

write code here

}

.content {

write code here

}

h1 {

text-align: center;

}

p {

text-align: justify;

}

</style>

</head>

<body>

<div class="content">

write code here

</div>

</body>

</html>

1. Create a webpage that has a background color and a background image.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Webpage with Background Color and Image</title>

<style>

body {

write code here

}

.content {

write code here

}

h1 {

text-align: center;

}

p {

text-align: justify;

}

</style>

</head>

<body>

<div class="content">

write code here

</div>

</body>

</html>

1. Create a webpage that displays different background color for different elements.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Webpage with Background Color and Image</title>

<style>

body {

write code here

}

.content {

write code here

}

h1 {

text-align: center;

}

p {

text-align: justify;

}

</style>

</head>

<body>

<div class="content">

write code here

</div>

</body>

</html>

1. Create a webpage that displays an image as the background and also repeats the background image only horizontally.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Background Image with Horizontal Repeat</title>

<style>

body {

write code here

}

.content {

write code here

}

h1 {

text-align: center;

}

p {

text-align: justify;

}

</style>

</head>

<body>

<div class="content">

write code here

</div>

</body>

</html>

1. Create a webpage that displays an image as the background and also repeats the background image only vertically.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Background Image with Vertical Repeat</title>

<style>

body {

write code here

}

.content {

write code here

}

h1 {

text-align: center;

}

p {

text-align: justify;

}

</style>

</head>

<body>

<div class="content">

write code here

</div>

</body>

</html>

1. Create a webpage that displays the text in different colors for each elements.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text in Different Colors</title>

<style>

body {

write code here

}

.red-text {

color: red;

}

.green-text {

color: green;

}

.blue-text {

color: blue;

}

.purple-text {

color: purple;

}

.orange-text {

color: orange;

}

</style>

</head>

<body>

< write code here

</body>

</html>

1. Create a webpage that displays the div element with the following text properties.

* Text color: Magenta
* Text alignment: right
* Text decoration: underline
* Text transform: uppercase
* Text indent: 60 pixels

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Div Element</title>

<style>

.styled-div {

write code here

}

</style>

</head>

<body>

<div class="styled-div">

This is a sample text inside a div element with specified text properties.

</div>

</body>

</html>

1. Create a webpage that displays the div element with the following font properties.

* Font family: Verdana
* Font style: italic
* Font size: 20pixels
* Text align: left
* Font weight: bold
* Text color: Red

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Div Element with Font Properties</title>

<style>

.styled-div {

write code here

}

</style>

</head>

<body>

<div class="styled-div">

This is a sample text inside a div element with specified font properties.

</div>

</body>

</html>

1. Create a web page that displays an image of size approximately 200 x 200 px, which gets itself repeated fully on the page. : (Use CSS)

HTML Code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Repeating Image Background</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

<div class="repeating-image"></div>

</body>

</html>

CSS Code (styles.css):

body {

margin: 0;

padding: 0;

background-color: #f0f0f0; /\* Fallback background color \*/

}

.repeating-image {

write code here

}

1. Using CSS, create a web page that displays text “Welcome to CSS World” with an image as its border,

a. Image repeats itself to fit the contents

b. Image should stretch to fit the contents

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>CSS Border with Repeating Image</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

<div class="border-repeating">

Welcome to CSS World

</div>

</body>

</html>

**CSS Code (styles.css):**

body {

margin: 0;

padding: 0;

background-color: #f0f0f0; /\* Fallback background color \*/

}

.border-repeating {

write code here

}

**9.3** [**Links and Lists**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T4)

1. Create a webpage that displays the hyperlink with the following color attributes.

• Set the color as %FF0000 for unvisited link

• Set the color as %FFFF00 for visited link

• Set the color as %FF00FF for mouse over link

• Set the color as %0000FF for selected link

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Hyperlinks</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

ul {

list-style-type: none;

padding: 0;

}

li {

margin-bottom: 10px;

}

a {

text-decoration: none;

padding: 5px 10px;

}

.unvisited-link {

color: #FF0000; /\* Red for unvisited link \*/

}

.visited-link:visited {

color: #FFFF00; /\* Yellow for visited link \*/

}

.mouse-over-link:hover {

color: #FF00FF; /\* Magenta for mouse-over link \*/

}

.selected-link:active {

color: #0000FF; /\* Blue for selected link \*/

}

1. Create a webpage that displays the hyperlink with the following properties.

* Set the color as red for unvisited link
* Set the text decoration as underline for unvisited link

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Hyperlinks</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

ul {

list-style-type: none;

padding: 0;

}

li {

margin-bottom: 10px;

}

a {

padding: 5px 10px;

text-decoration: none;

}

.unvisited-link {

color: red; /\* Red color for unvisited link \*/

text-decoration: underline; /\* Underline for unvisited link \*/

}

.visited-link:visited {

color: #000; /\* Default color for visited link \*/

}

.mouse-over-link:hover {

color: #00F; /\* Blue color for mouse-over link \*/

}

.selected-link:active {

color: #F00; /\* Red color for selected link \*/

}

1. Create a webpage that displays the hyperlink with the following background color properties.

* Set the background color as %FF0000 for unvisited link
* Set the background color as %FFFF00 for visited link
* Set the background color as %FF00FF for mouse over link
* Set the background color as %0000FF for selected link
* Create a webpage that displays a list as an ordered list with the following properties.
* List-style-type: lower-greek;
* Text align: left
* Color: red
* Text transform: uppercase
* Text decoration: underline
* Font name: Verdana
* Font size: 20pixels
* Font weight: bold

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Hyperlinks and Ordered List</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Verdana, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

ul, ol {

list-style-type: none;

padding: 0;

}

ul li, ol li {

margin-bottom: 10px;

}

a {

display: inline-block;

padding: 5px 10px;

text-decoration: none;

color: #fff;

}

.unvisited-link {

background-color: #FF0000;

}

.visited-link:visited {

background-color: #FFFF00;

}

.mouse-over-link:hover {

background-color: #FF00FF;

}

.selected-link:active {

background-color: #0000FF;

}

.styled-list {

write code here

}

1. Create a webpage that displays a list as an unordered list with the following properties.

* List-style-type: square;
* Text align: left
* Color: red
* Text transform: uppercase
* Text decoration: underline
* Font name: Arial
* Font size: 25pixels
* Font weight: bold

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Unordered List</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

ul {

list-style-type: none;

padding: 0;

}

ul.styled-list {

write code here

}

ul.styled-list li {

margin-bottom: 10px;

}

1. Create a webpage that displays an image as the list item marker.

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>List with Image Marker</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.image-list {

list-style-type: none;

padding-left: 20px;

}

.image-list li {

write code here

}

1. Create a webpage that displays the paragraph with a total width of 250pixels.

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Paragraph with Width</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.paragraph-width {

write code here

}

[**9.4 CSS Box Model**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T5)

1. Create a webpage that displays the paragraph with the following properties.

* Set the width of the border as 10 Pixels.
* Set the style of the border as dotted.
* Set the color of the border as #0000ff

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Paragraph with Border</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.bordered-paragraph {

write code here

}

1. Create a webpage that displays the paragraph with the following margin properties.

* Set the top margin as 2cm
* Set the bottom margin as 25%

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Paragraph with Margin Properties</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.margined-paragraph {

write code here

}

1. Create a webpage that displays three paragraphs with the following margin properties.

* Set the top margin as 3cm for first paragraph
* Set the bottom margin as 5cm for first paragraph
* Set the left margin as 2cm for second paragraph
* Set the top margin as 25% for third paragraph

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Paragraphs with Margin Properties</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.paragraph1 {

margin-top: 3cm; /\* Top margin set to 3 centimeters \*/

margin-bottom: 5cm; /\* Bottom margin set to 5 centimeters \*/

}

.paragraph2 {

margin-left: 2cm; /\* Left margin set to 2 centimeters \*/

}

.paragraph3 {

margin-top: 25%; /\* Top margin set to 25% of the parent element's height \*/

}

1. Create a webpage that displays four paragraphs with the following padding properties.

* Set the left padding as 3cm for first paragraph
* Set the right padding as 50% for second paragraph
* Set the bottom padding as 25% for third paragraph
* Set the top padding as 25% for fourth paragraph

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Paragraphs with Padding Properties</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.paragraph1 {

padding-left: 3cm;

}

.paragraph2 {

padding-right: 50%;

}

.paragraph3 {

padding-bottom: 25%;

}

.paragraph4 {

padding-top: 25%;

}

1. Create a webpage that displays the div element with the following CSS properties.

* Set the border as 2 pixels solid
* Set the padding as 20 pixels
* Set the background as #dddddd
* Set the border top left radius as 2em

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Styled Div Element</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.styled-div {

write code here

}

**9.5** [**CSS3**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T6)

1. Create a webpage that displays the div element with the following CSS property.

* Set the text shadow as h-shadow

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text Shadow Example</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

}

h2 {

text-align: center;

color: #333;

}

.text-shadow-div {

write code here

}

1. Create a webpage that displays a white text with black shadow.

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>White Text with Black Shadow</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f0f0f0; /\* Light gray background \*/

}

h2 {

text-align: center;

color: #333;

}

.text-with-shadow {

write code here

}

1. Create a webpage that displays an image as the border around an element with the following properties.

• Border: 15 pixels solid

• Padding: 20 pixels

• Border image: url (imagename.png) 30 30 stretch

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Image Border Example</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f0f0f0; /\* Light gray background \*/

}

h2 {

text-align: center;

color: #333;

}

.bordered-element {

write code here

}

1. Create a webpage that displays an image as the background around an element with the following properties.

* Border: 1pixels solid black
* Height: 150pixels
* Width: 180pixels
* Background : url (imagename.gif)
* Background-repeat: no-repeat
* Background-size: cover

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Image Background Example</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f0f0f0; /\* Light gray background \*/

}

h2 {

text-align: center;

color: #333;

}

.background-element {

write code here

}

1. Create a webpage that displays the text which is provided in div element and rotates the div element counter-clockwise with 20 degrees.

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Rotated Text Example</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f0f0f0;

}

h2 {

text-align: center;

color: #333;

}

.rotated-div {

write code here

}

1. Create a webpage that displays the div element with the following properties.

• Set the element to be two times of its original width, and three times of its original height:

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Scaled Div Element Example</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f0f0f0; /\* Light gray background \*/

}

h2 {

text-align: center;

color: #333;

}

.scaled-div {

write code here

}

1. Create a webpage that displays the div element with the following properties.

* Skews the element 20 degrees along the X-axis, and 10 degrees along the Y-axis
* Set the background color as yellow
* Set the border as 1pixels solid black
* Set the width as 200pixels
* Set the height as 100 pixels

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Skewed Div Element Example</title>

<link rel="stylesheet" href="styles.css"> <!-- Link to external CSS file -->

</head>

<body>

write code here

</body>

</html>

**CSS Code (styles.css):**

body {

font-family: Arial, sans-serif;

padding: 20px;

background-color: #f0f0f0; /\* Light gray background \*/

}

h2 {

text-align: center;

color: #333;

}

.skewed-div {

write code here

}

**9.6** [**Introduction to Javascript and Basics Functions**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T7)

1. Write a JavaScript program which will print ‘HelloWorld’ in the body of the HTML document

**HTML Code:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>HelloWorld Example</title>

<script src="script.js" defer></script> <!-- Link to external JavaScript file -->

</head>

<body>

</body>

</html>

**JavaScript Code (script.js):**

const body = document.body;

const paragraph = document.createElement('p');

paragraph.textContent = 'HelloWorld';

body.appendChild(paragraph);

1. Write a JavaScript function that reverses a number.

**INPUT:** 32243;

**OUTPUT** : 34223

function reverseNumber(num) {

let numStr = num.toString();

let reversedStr = numStr.split('').reverse().join('');

let reversedNum = parseInt(reversedStr);

return reversedNum;

}

let x = 32243;

let reversed = reverseNumber(x);

console.log('Reversed number:', reversed);

1. Write a JavaScript function that checks whether a passed string is palindrome or not?

function isPalindrome(str) {

write code here

}

let testStr1 = "A man, a plan, a canal, Panama!";

let testStr2 = "racecar";

let testStr3 = "hello";

console.log(isPalindrome(testStr1));

console.log(isPalindrome(testStr2));

console.log(isPalindrome(testStr3));

1. Write a JavaScript function that accepts a string as a parameter and converts the first letter of each word of the string in upper case.

function capitalizeWords(str) {

write code here

}

let inputString = "hello world";

let capitalizedString = capitalizeWords(inputString);

console.log(capitalizedString);

**9.7 JavaScript Objects**

1. Write a JavaScript program that displays the position of the first occurrence of "World" in the variable txt.

let txt = "Hello World! Welcome to the World.";

let position = txt.indexOf("World");

if (position !== -1) {

console.log(`The position of the first occurrence of "World" is: ${position}`);

} else {

console.log("The string 'World' was not found in the text.");

}

1. Write a JavaScript program that converts the text to upper case.

let uppercaseText = text.toUpperCase();

console.log("Original Text:", text);

console.log("Uppercase Text:", uppercaseText);

1. Write a JavaScript function to remove specified number of characters from a string.

function removeCharacters(str, numToRemove) {

write code here

}

let originalString = "Hello, world!";

let numToRemove = 5;

let modifiedString = removeCharacters(originalString, numToRemove);

console.log(`Original String: "${originalString}"`);

console.log(`Modified String (removed ${numToRemove} characters): "${modifiedString}"`);

1. Create an HTML page with two textboxes and a Calculate button:

When user enters the numbers and clicks the Calculate button, it has to alert the sum.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sum Calculator</title>

<style>

body {

write code here

}

.input-container {

margin-bottom: 20px;

}

</style>

</head>

<body>

write code here

</body>

</html>

1. Write a JavaScript program which will have one textbox to get input as string and after giving input when you leave the input textbox, a function is triggered which transforms the input text to uppercase.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Transform Input to Uppercase</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 50px;

}

.input-container {

margin-bottom: 20px;

}

</style>

</head>

<body>

write code here

</body>

</html>

1. Create a HTML page with 4 hyperlinks named white, red, blue and pink. The background colour of the HTML page will change based on the link which is clicked.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Change Background Color</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

margin-top: 50px;

}

.container {

margin-bottom: 20px;

}

.color-link {

margin: 0 10px;

text-decoration: none;

padding: 5px 10px;

border: 1px solid #ccc;

border-radius: 5px;

cursor: pointer;

}

</style>

</head>

<body>

write code here

</body>

</html>

1. Write a javascript function to implement it.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Change Background Color</title>

<style>

body {

write code here

}

.container {

margin-bottom: 20px;

}

.color-link {

write code here

}

</style>

</head>

<body>

write code here

</body>

</html>

1. Write a JavaScript program which will

a. give an alert message of today’s date.

b. prompt the user to give their name and gives an alert message ‘Hello’ and username.

c. prompt the user to give 2 numbers and alerts the sum of those 2 numbers.

**Part a:** Alert today's date

let today = new Date();

let formattedDate = `${today.getDate()}-${today.getMonth() + 1}-${today.getFullYear()}`;

alert(`Today's date is: ${formattedDate}`);

**Part b**: Prompt user for their name and greet them

let username = prompt('Please enter your name:');

alert(`Hello, ${username}! Welcome.`);

**Part c:** Prompt user for two numbers and alert their sum

let num1 = parseFloat(prompt('Enter the first number:'));

let num2 = parseFloat(prompt('Enter the second number:'));

write code here

1. Write a JavaScript program that prompts the user to enter a number and display whether the number given is odd or even.

let number = parseInt(prompt('Enter a number:'));

write code here

1. Create program to display current time in a textbox (HH:MM:SS) such that the value in the time textbox is dynamic (Time should get updated every second) and not static.[Hint: Use setTimeOut function]

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Dynamic Time Display</title>

</head>

<body>

write code here

</body>

</html>

**9.8 JavaScript Validations**

1. Write validation functions and modify the onSubmit event Handler in the form code to validate the following form fields:

1. Member number

* Must be entered
* Must be a number

2. Password

* Must be entered
* Must be longer than 4 characters

If an error occurs, use an alert box to display an error message for the first field in error and place the cursor in the first field in error.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Form Validation Example</title>

<script>

write code here

</script>

</head>

<body>

write code here

</body>

</html>

1. Write validation functions and modify the onSubmit event Handler in the form code to validate the following form fields:

Phone Number

* Must be entered
* Must be in below format

XXX-XXX-XXXX

XXX.XXX.XXXX

XXX XXX XXXX

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Phone Number Validation Example</title>

<script>

write code here

</script>

</head>

<body>

write code here

</body>

</html>

1. Write a validation function to check a password between 6 to 20 characters which contain at least one numeric digit, one uppercase and one lowercase letter

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Password Validation Example</title>

<script>

function validatePassword(password) {

write code here

}

function validateForm() {

write code here

}

</script>

</head>

<body>

write code here

</body>

</html>

**11.9 Introduction to BootStrap**

1. Download bootstrap files as source files from the given location <http://getbootstrap.com/>
2. Unzip the contents and add the right files in the correct location of your MyExercise dynamic web project in eclipse

**STEP 1:** Unzip the Contents: Ensure you have unzipped the contents of your project archive (e.g., .zip file).

**STEP 2:** Open Eclipse: Launch Eclipse IDE.

**STEP 3**: Open Your Dynamic Web Project: Locate your existing dynamic web project in Eclipse where you want to add the files. If the project is not already imported into Eclipse:

Go to File > Import.

Select General > Existing Projects into Workspace.

Click Next, then browse to the directory where your project is located, select it, and click Finish.

**STEP 4:** Locate Project Structure: In the Project Explorer or Package Explorer view in Eclipse, locate your dynamic web project. Typically, the structure includes folders like src, WebContent, and possibly others depending on your project setup.

**STEP 5:** Add Files to WebContent: Most web-related files (HTML, CSS, JavaScript, images, etc.) are typically placed in the WebContent directory (or equivalent). Right-click on the WebContent folder in Eclipse. Choose Import if you have files to import from your unzipped folder structure. Alternatively, you can directly copy and paste files from the unzipped folder into the WebContent directory in Eclipse.

**STEP 6**: Refresh the Project: After adding or importing files, right-click on the project name in Eclipse. Select Refresh to ensure Eclipse recognizes the newly added files. Check Deployment

**STEP 7**: Descriptors and Configuration: If your project requires deployment descriptors (web.xml for traditional servlet projects, or pom.xml for Maven projects), ensure they are correctly configured to include any new files you've added.

**STEP 8**: Run and Test: Once files are added and the project is configured, run your dynamic web project to ensure everything works as expected. Right-click on your project, then select Run As > Run on Server to deploy and test your web application.

1. Create a OldLook.html web page in the current project with the following content

|  |  |
| --- | --- |
| **Tags** | **Contents** |
| H1 | Welcome to my page |
| P | this is a trial for bootstrap |

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Old Look Page</title>

<!-- Optional: You can link to Bootstrap CSS if desired -->

<!-- <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"> -->

</head>

<body>

write code here

</body>

</html>

1. Create a NewLook.html web page in the current project with the contents of the previous exercise along with the stylesheet reference of css/bootstrap.min.css Observe the difference if any.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>New Look</title>

<link rel="stylesheet" href="css/bootstrap.min.css">

</head>

<body>

write code here

</body>

</html>

**11.10 Front-end Frameworks**

1. Create a webpage which displays different images on screen as you resize the browser. Take 4 images like img1.jpg , img2.jpg, img3.jpg , img4.jpg which should be displayed on the web page and only one should be visible at any time as you keep resizing the screen it should start changing to img2 and then to img3 and so on.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Responsive Images</title>

<style>

body {

margin: 0;

padding: 0;

}

.image {

display: none;

width: 100%;

height: auto;

}

.image-default {

display: block;

}

@media (min-width: 600px) and (max-width: 899px) {

.image-default {

display: none;

}

.image-medium {

display: block;

}

}

@media (min-width: 900px) and (max-width: 1199px) {

.image-default, .image-medium {

display: none;

}

.image-large {

display: block;

}

}

@media (min-width: 1200px) {

.image-default, .image-medium, .image-large {

display: none;

}

.image-xlarge {

display: block;

}

}

</style>

</head>

<body>

< write code here

</body>

</html>

1. Create a small table with 2 columns Name and contact no, Populate the table with 5 of your friends names and their contact nos. When the mouse is taken over the table it should highlight the row where the mouse is placed.(hint : use table hover)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Hover Highlight Table</title>

<style>

body {

font-family: Arial, sans-serif;

}

table {

width: 50%;

border-collapse: collapse;

margin: 20px auto;

}

th, td {

padding: 12px;

border: 1px solid #ddd;

text-align: left;

}

th {

background-color: #f2f2f2;

}

tr:hover {

background-color: #f1f1f1;

}

</style>

</head>

<body>

<table>

write code here

</table>

</body>

</html>

1. Create a table of food contents with its calorie value. Display the rows of food contents with calories

* <=300 in light green background,
* 301 to 700 in light blue,
* 700 - 1200 in light orange
* >1200 in light red.

Hint: use table table-striped

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Calorie Table</title>

<style>

body {

font-family: Arial, sans-serif;

}

table {

width: 70%;

border-collapse: collapse;

margin: 20px auto;

}

th, td {

padding: 12px;

border: 1px solid #ddd;

text-align: left;

}

th {

background-color: #f2f2f2;

}

.calories-low {

background-color: lightgreen;

}

.calories-medium {

background-color: lightblue;

}

.calories-high {

background-color: lightorange;

}

.calories-very-high {

background-color: lightcoral;

}

</style>

</head>

<body>

<table class="table table-striped">

write code here

</table>

</body>

</html>

1. Create a webpage to display Tulips.jpg from sample pictures of your machine. Ensure the image has soft round edges and as you resize the screen, the image should also resize itself and fit its display to full page size.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Responsive Tulips Image</title>

<style>

body, html {

write code here

}

.responsive-image {

write code here

}

</style>

</head>

<body>

<img src="path/to/your/image/Tulips.jpg" alt="Tulips" class="responsive-image">

</body>

</html>

1. Create a webpage called OnlineRead with give look and feel

Page should display Book Types as buttons with given background effect:

* For All – white
* Kids – bark blue
* Teens – Green
* Scientific – Light Blue
* Crime – OrangeHorror – red
* Feedback – shown

In next line display the comfort font size with buttons of varying sizes ranging from big to small (in 4 steps)

**Step 1:** HTML Structure

Create an HTML file named OnlineRead.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>OnlineRead</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

<div class="container">

write code here

</div>

<h2>Comfort Font Size</h2>

<div class="font-sizes">

write code here

</div>

<h2>Feedback</h2>

<form>

write code here

</form>

</div>

</body>

</html>

**Step 2:** CSS Styling

Create a CSS file named styles.css to style the page as per the requirements:

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

padding: 20px;

}

.container {

write code here

}

h1 {

text-align: center;

color: #333;

}

.book-types {

text-align: center;

margin-bottom: 20px;

}

.book-types .btn {

write code here

}

.book-types .all { background-color: white; color: #000; }

.book-types .kids { background-color: darkblue; }

.book-types .teens { background-color: green; }

.book-types .scientific { background-color: lightblue; color: #000; }

.book-types .crime { background-color: orange; }

.book-types .horror { background-color: red; }

h2 {

color: #333;

}

.font-sizes {

text-align: center;

margin-bottom: 20px;

}

.font-sizes .font-btn {

margin: 5px;

padding: 10px 20px;

border: none;

border-radius: 4px;

cursor: pointer;

color: #fff;

font-size: 16px;

}

.font-sizes .big { font-size: 24px; background-color: #4CAF50; }

.font-sizes .medium { font-size: 20px; background-color: #2196F3; }

.font-sizes .small { font-size: 16px; background-color: #FFC107; }

.font-sizes .smallest { font-size: 12px; background-color: #f44336; }

form {

text-align: center;

}

textarea {

write code here

}

button[type="submit"] {

write code here

}

button[type="submit"]:hover {

background-color: #45a049;

}

**11.11 Bootstrap Grid System**

1. Create a webpage with 4 grid sections containing Contents of some 10 lines. Ensure the grid alters itself as the browser is resized.

**Step 1:** HTML Structure

Create an HTML file named responsiveGrid.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Responsive Grid</title>

<link rel="stylesheet" href="styles.css">

</head>

<body>

write code here

</body>

</html>

**Step 2:** CSS Styling

Create a CSS file named styles.css to style the CSS Grid Layout:

body {

write code here

}

.container {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

gap: 20px;

padding: 20px;

}

.grid-item {

write code here

}

p {

margin: 0 0 10px;

}

**10. Servlets /** **Java Server Pages(JSP)**

**10.1 Introduction to Servlets**

1. Write a servlet to display “My First Servlet program” on browser.

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/first-servlet")

public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

}

**HeaderServlet.java**

import java.io.IOException;

import java.util.Enumeration;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/header-servlet")

public class HeaderServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

}

**10.2** [**Servlets-Get and Post Requests**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T2)

1. Create a servlet that prints all the request headers it receives, along with their associated values.

**ParametersServlet.java**

import java.io.IOException;

import java.util.Enumeration;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/parameters-servlet")

public class ParametersServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

processRequest(request, response);

}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

processRequest(request, response);

}

public void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

}

**form.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Test Parameters Servlet</title>

</head>

<body>

write code here

</body>

</html>

**web.xml (if needed)**

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd" version="3.1">

<servlet>

<servlet-name>ParametersServlet</servlet-name>

<servlet-class>ParametersServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ParametersServlet</servlet-name>

<url-pattern>/parameters-servlet</url-pattern>

</servlet-mapping>

</web-app>

1. Write a servlet to show all the parameters sent to the servlet via either GET or POST.

Note: consider all types of form fields.

**web.xml**

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

version="3.1">

<!-- Define context parameters -->

<context-param>

<param-name>contextParam1</param-name>

<param-value>Context Value 1</param-value>

</context-param>

<context-param>

<param-name>contextParam2</param-name>

<param-value>Context Value 2</param-value>

</context-param>

<!-- Define the servlet -->

<servlet>

write code here

</servlet>

<servlet-mapping>

write code here

</servlet-mapping>

</web-app>

**ConfigContextServlet.java**

import java.io.IOException;

import java.util.Enumeration;

import javax.servlet.ServletConfig;

import javax.servlet.ServletContext;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/config-context-servlet")

public class ConfigContextServlet extends HttpServlet {

@Override

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

@Override

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request, response);

}

}

**10.3** [**Servlet Config and Servlet Context**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T3)

1. Write a Servlet Program that prints the Servlet Config and Servlet Context Parameters

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/visitor-servlet")

public class VisitorServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

@Override

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request, response);

}

}

**10.4** [**Servlet - Cookies and Session Management**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T4)

1. Create a Servlet that recognizes the first time visitor to a web application and responds by saying "Welcome, you are visiting for the first time". When the page is visited for the second time, it should say 'Welcome Back'.

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.Cookie;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet("/display-cookies")

public class DisplayCookiesServlet extends HttpServlet {

write code here

}

@Override

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request, response);

}

}

1. Display all the cookies available for the given application in a tabular format. If there are no cookies available with the application, display “No Cookies”.

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

@WebServlet("/session-tracking")

public class SessionTrackingServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

}

1. Using session tracking, display how many times the particular client has visited the page. Display the session id and other details about the session object.

**10.5** [**Introduction to JSP**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T5)

1. Create a HTML Page, which asks the user to enter a number in a textbox. On clicking the submit button, it places the request to a Servlet. The Servlet generates all Prime numbers which are less than the given number and adds them to an ArrayList and forwards the control to a JSP page. The JSP page iterates through the ArrayList and prints them in a tabular format.
2. Write a JSP program which when invoked will print today’s date and time. [Hint: Use JSP expressions here]
3. Write a jsp program that prints a table with numbers and their corresponding factorial values.

**10.6** [**Scripting**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T6) **Elements, Standard Action Elements, Directive Elements, Scope Variables, Implicit Objects**

1. 4 Scripting Elements

<%-- --%> JSP Comments

<%! %> Declaration Element

<%= %> Expression Tag

<% %> Scriptlet

1. 9 Standard Action Elements

<jsp:include>

<jsp:forward>

<jsp:useBean>

<jsp:setProperty />

<jsp:getProperty />

</jsp:useBean>

<jsp:plugin>

<jsp:params>

<jsp:param/>

<jsp:fallback />

</jsp:params>

</jsp:plugin>

1. 3 Directive Elements

<%@ page %>

<%@ include %>

<%@ taglib %>

1. 4 Scope Variables

page scope

request scope

session scope

application scope

1. 9 Implicit Objects

page - this

request – HttpServletRequest

response - HttpServletResponse

out - JspWriter

application - ServletContext

session - HttpSession

config - ServletConfig

pageContext - PageContext

exception - Throwable

**10.7** [**JavaBeans in JSP**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T6)

1. Create a bean that represents information needed to calculate an employee's salary. Has String (employee name) and int (employee ID) properties. Create an application to demonstrate automatically filling in bean properties from request parameters.
2. Create a servlet that forwards the request to one of three different JSP pages, depending on the value of the operation request parameter. Say if input is <10 then page 1 or greater than 10 and less than 99 then page 2 otherwise error page
3. Create an emp table in the database with fields name, id and designation.

* Create a HTML file with fields name, id and designation. Make sure that the textfields have the same name as name, id and designation.
* Create a javabean with fields name, id and designation.
* On click of the submit button of the HTML, invoke a jsp page which will extract the values that were given by the HTML page and it invokes a servlet.
* The Servlet will make a connection to the database and store the value in the table.
* Connection to the database should have been established using a separate java class.
* You can enhance the program by including more features like deletion, updation and selecting all records.

**11. Asynchronous JavaScript and XML (AJAX)**

[**11.1 AJAX Application**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T3)

1. Develop a web application to display a list of important cities of different states displayed in a dropdown list on an html page. On selecting a state, the list of cities of that particular state should be displayed in a tabular format, just below the dropdown list. Create static list in the servlet for the application.

**CityServlet.java:**

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.HashMap;

import java.util.Map;

@WebServlet("/CityServlet")

public class CityServlet extends HttpServlet {

write code here

}

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

write code here

}

}

**index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Important Cities</title>

<script>

write code here

</script>

</head>

<body>

write code here

</body>

</html>

[**11.2 AJAX Database Application**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T4)

1. Develop a web application to dynamically check for availability of an email-id for registration. If email-id does not exist in the database, ‘Available!’, else ‘Not Available!’, should be displayed next to the email textbox. If email-id pattern does not conform to an email-id, display Invalid Email Id. Perform the check when the textbox loses focus.

Consider the following fields for the Profile table:

EMAIL, PASSWORD, NAME, DATEOFBIRTH, GENDER, CITY, PINCODE, MOBILE

Note\*: Check should be performed irrespective of uppercase or lowercase data.

**Step 1:** Set Up the Database and Server Environment

CREATE TABLE Profile (

EMAIL VARCHAR(255) PRIMARY KEY,

PASSWORD VARCHAR(255),

NAME VARCHAR(255),

DATEOFBIRTH DATE,

GENDER VARCHAR(10),

CITY VARCHAR(255),

PINCODE VARCHAR(10),

MOBILE VARCHAR(15)

);

**Backend Setup (Node.js with Express example):**

mkdir email-check-app

cd email-check-app

npm init -y

npm install express mysql body-parser cors

Create a file server.js:

javascript

Copy code

const express = require('express');

const mysql = require('mysql');

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express();

app.use(bodyParser.json());

app.use(cors());

const db = mysql.createConnection({

host: 'localhost',

user: 'root',

password: '',

database: 'your\_database\_name'

});

db.connect(err => {

if (err) throw err;

console.log('Database connected!');

});

app.post('/check-email', (req, res) => {

write code here

});

app.listen(3000, () => {

console.log('Server running on port 3000');

});

**index.html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Email Check</title>

<script>

write code here

</script>

</head>

<body>

<div>

write code here

</div>

</body>

</html>

**11.3** [**XML**](https://talentnext.wipro.com/PBLApp/tmodule.jsp#T1) **Parsing**

1. Create an XML document for the following data:

Student Registration Form

Name Rahul

Age 16

Gender Male

Course C Programming

<?xml version="1.0" encoding="UTF-8"?>

<StudentRegistrationForm>

<Student>

write code here

</Student>

</StudentRegistrationForm>

1. Check whether the following XML documents are well formed. If it is not well formed, correct the errors.

me@wipro.com

you@wipro.com

Re: XML

I'm working on XML now.

<Email>

write code here

</Email>

1. Develop DTD for the below given XML document:

Manju

Wipro

9988776655

<?xml version="1.0" encoding="UTF-8"?>

<Employee>

< write code here

</Employee>

1. Develop DTD file for the below given XML document:

QWZ5671

39.95

Red

Burgund

Red

Burgundy

RRX9856

42.50

Red

Navy

Burgundy

Red

Navy

Burgundy

Black

Navy

Black

Burgundy

Black

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE Products SYSTEM "products.dtd">

<Products>

<Product>

write code here

</Product>

<Product>

write code here

</Product>

</Products>

1. Develop an XML file following the DTD file given below:

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE Products [

<!ELEMENT Products (Product+)>

<!ELEMENT Product (ProductCode, Price, Colors, Sizes)>

<!ELEMENT ProductCode (#PCDATA)>

<!ELEMENT Price (#PCDATA)>

<!ELEMENT Colors (Color+)>

<!ELEMENT Sizes (Size+)>

<!ELEMENT Color (#PCDATA)>

<!ELEMENT Size (#PCDATA)>

]>

<Products>

<Product>

write code here

</Product>

<Product>

write code here

</Product>

</Products>

1. Create namespaces for the following set of tags.

Student- Root tag

StudentID

Theory Marks

PracticalMarks

TotalMarks

Grade

Student- Root tag

StudentID

Name

DOB

Gender

Course

<?xml version="1.0" encoding="UTF-8"?>

<Students xmlns:marks="http://example.com/marks" xmlns:info="http://example.com/info">

<marks:Student>

write code here

</marks:Student>

1. Avoid Naming conflict in the following XML document:

Moto E

123

Qualcomm Snapdragon 200

2390 mAh

<?xml version="1.0" encoding="UTF-8"?>

<Products xmlns:phone="http://example.com/phone">

<phone:Phone>

write code here

</phone:Phone>

</Products>

1. Create an XSD file for the XML document given in the DTD HandsOn section Exercise 1

Beetel M71 Landline

456

LCD Screen

Yes

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Define element types -->

<xs:element name="Products">

<xs:complexType>

write code here

</xs:complexType>

</xs:element>

</xs:schema>

1. Create an XSD file for the XML document given in the DTD HandsOn section Exercise 2

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- Define element types -->

<xs:element name="Students">

<xs:complexType>

write code here

</xs:complexType>

</xs:element>

</xs:schema>

* 1. **JSON Parsing**

1. **JSON Example**

JSON example can be created by object and array. Each object have different data such as text, number, boolean, etc.

JSON Object contains in the form of key/value pair. The keys are strings and values are JSON types. Keys and values are separated by : colon symbol. Each entry is separated by comma.

The curly braces represents JSON object

{

“employee”: {

“name”:”ABC”,

“salary”:100000.00,

“married”: true

}

}

JSON Array Example

[“Sunday”, “Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”, “Saturday”]

JSON Array having objects

[

{“name”:”ABC”, “email”:”abc@gmail.com},

{“name”:”XYZ”, “email”:”xyz@gmail.com}

]

1. JSON Object

JSON object having another object

{

“firstName”:”ABC”,

“lastName”:”XYZ”,

“age”:30,

“address”: {

“streetAddress”:”Street No: 1, Ameerpet”,

“city”:”Hyderabad”

“state”:”Telangana”,

“postalCode”:500018

}

}

1. Java JSON Example

The json.sample library allows us to read and write JSON data in Java. We can encode and decode JSON objects in java using json.simple library.

To install json.sample you need to set classpath of json-sample.jar in CLASSPATH setting.

Java JSON Encode

import org.json.simple.JSONObject;

public class JSONExample1 {

public static void main(String rags[]) {

JSONObject obj=new JSONOject();

obj.put(“name”,”ABC”);

obj.put(“age”,new Integer(30));

obj.put(“salary”, new Double(100000.00));

System.out.println(obj);

}

}

Java JSON Decode

import org.json.simple.JSONObject;

import org.json.simple.JSONValue;

public class JSONDecodeExample1 {

public static void main(String rags[]) {

String s=”{\”name\”:\”ABC\”,\”salary\”:100000.00,\”age\”:30}”;

Object obj=JSONValue.parse(s);

JSONObject jsonObject=(JSONObject)obj;

String name=(String)jsonObject.get(”name”);

Double salary=(Double)jsonObject.get(”salary”);

Integer age=(Integer)jsonObject.get(”age”);

System.out.println(name+” ”+salary.doubleValue()+” ”+age.intValue());

}

}

**12. Hibernate**

**12.1 INTRODUCTION TO HIBERNATE**

1. Connect to Oracle using Scott login

**SOLUTION:**

* Go to the command prompt and set environment variables according to your database server, e.g.:

set ORACLE\_HOME=C:\ora\db19

set PATH=%ORACLE\_HOME%\bin;%PATH%

* Start SQL\*Plus:

cd /d %ORACLE\_HOME%\rdbms\admin

sqlplus /nolog

* If the demo tables in the SCOTT schema have not been created, you can create them How?
* If the SCOTT user has been locked, you need to unlock it:
* Connect as SYS:

connect sys as sysdba

* Set password and unlock:

alter user scott identified by tiger account unlock;

* Connect as SCOTT, e.g. connect scott/tiger@db19

1. view the table structure of EMP table (desc Emp)

Create proper bean class for EMP table and insert a record into this table

Use Hibernate mapping file (.hbm.xml)

**STEP 1** Maven Dependencies (pom.xml)

Make sure you have the Hibernate dependencies in your pom.xml:

<dependencies>

<!-- Hibernate core dependency -->

<dependency>

write code here

</dependency>

<!-- H2 Database dependency for testing purposes -->

<dependency>

write code here

</dependency>

<!-- JPA API dependency -->

<dependency>

write code here

</dependency>

</dependencies>

**STEP 2.** Hibernate Configuration File (hibernate.cfg.xml)

Configure Hibernate settings in hibernate.cfg.xml:

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

write code here

</session-factory>

</hibernate-configuration>

**STEP 3**. Create the Entity Class Define the entity class Flower:

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "Flower")

public class Flower {

@Id

@Column(name = "FlowerID", nullable = false, length = 50)

private String flowerId;

@Column(name = "FlowerName", nullable = false, length = 50)

private String flowerName;

@Column(name = "Color", nullable = false, length = 50)

private String color;

@Column(name = "Price", nullable = false)

private double price;

public String getFlowerId() {

return flowerId;

}

public void setFlowerId(String flowerId) {

write code here

}

public String getFlowerName() {

return flowerName;

}

public void setFlowerName(String flowerName) {

write code here

}

public String getColor() {

return color;

}

public void setColor(String color) {

write code here

}

public double getPrice() {

write code here

}

public void setPrice(double price) {

write code here

}

}

**STEP 4:**. Running the Hibernate Application

Finally, you need to create a Hibernate utility class to manage the session factory and write a main class to run the application.

**HibernateUtil.java**

package com.example;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

private static final SessionFactory sessionFactory = buildSessionFactory();

private static SessionFactory buildSessionFactory() {

write code here

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

public static void shutdown() {

getSessionFactory().close();

}

}

**Main.java**

package com.example;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create a table with the following structure.

Column Name Datatypes

FlowerID Varchar2

FlowerName Varchar2

Color Varchar2

Price Number

**STEP 1** Maven Dependencies (pom.xml)

Ensure you have the necessary dependencies for Hibernate and the Oracle JDBC driver:

<dependencies>

<!-- Hibernate core dependency -->

<dependency>

write code here

</dependency>

<!-- Oracle JDBC driver -->

<dependency>

write code here

</dependency>

<!-- JPA API dependency -->

<dependency>

write code here

</dependency>

</dependencies>

**STEP 2** Hibernate Configuration File (hibernate.cfg.xml)

Configure Hibernate settings for the Oracle database in hibernate.cfg.xml:

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

write code here

</session-factory>

</hibernate-configuration>

**STEP 3** Create the Entity Class

Define the entity class Flower:

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "Flower")

public class Flower {

@Id

@Column(name = "FlowerID", nullable = false, length = 50)

private String flowerId;

@Column(name = "FlowerName", nullable = false, length = 50)

private String flowerName;

@Column(name = "Color", nullable = false, length = 50)

private String color;

@Column(name = "Price", nullable = false)

private double price;

public String getFlowerId() {

return flowerId;

}

public void setFlowerId(String flowerId) {

this.flowerId = flowerId;

}

public String getFlowerName() {

return flowerName;

}

public void setFlowerName(String flowerName) {

this.flowerName = flowerName;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

**STEP 4**. Running the Hibernate Application

You need to create a Hibernate utility class and a main class to run the application.

**HibernateUtil.java**

package com.example;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateUtil {

private static final SessionFactory sessionFactory = buildSessionFactory();

private static SessionFactory buildSessionFactory() {

write code here

}

public static SessionFactory getSessionFactory() {

return sessionFactory;

}

public static void shutdown() {

getSessionFactory().close();

}

}

**Main.java**

package com.example;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create proper bean class and insert a record into this table

**Step 1:** Create a Database Table

Let's assume we have a table named Employee with the following structure:

CREATE TABLE Employee (

id INT PRIMARY KEY,

name VARCHAR(100),

department VARCHAR(100),

salary DECIMAL(10, 2)

);

**Step 2:** Create the Bean Class

public class Employee {

private int id;

private String name;

private String department;

private BigDecimal salary;

public Employee() {}

public Employee(int id, String name, String department, BigDecimal salary)

{

write code here

}

public int getId() {

write code here

}

public void setId(int id) {

write code here

}

public String getName() {

write code here

}

public void setName(String name) {

write code here

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

write code here

}

public BigDecimal getSalary() {

write code here

}

public void setSalary(BigDecimal salary) {

write code here

}

}

**Step 3**: Insert a Record Using JDBC

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.math.BigDecimal;

public class EmployeeDAO {

write code here

}

public static void main(String[] args) {

write code here

}

}

1. Using Hibernate create an employee table in the backend and insert 5 employee objects into the database.

The following are the details of the employee table.

Name varchar2

Id Number

Designation varchar2

Salary Number

The Employee Id should be auto incremented

[ Hint : generator class="increment"]

Using Hibernate create a Product table in the backend and insert 5 Product objects into the database.

The following are the details of theProduct table.

ProductName varchar2

ProductId Number

Price Number

The value of the ProductId should be taken from a sequnce which is created at the back end

[ Hint : generator class=sequence ]

**Step 1:** Set up Hibernate Configuration

Dependencies

Include Hibernate and other necessary dependencies in your pom.xml if you are using Maven:

<dependencies>

<dependency>

write code here

</dependency>

<dependency>

write code here

</dependency>

<dependency>

write code here

</dependency>

</dependencies>

**Hibernate Configuration File (hibernate.cfg.xml)**

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

write code here

</session-factory>

</hibernate-configuration>

**Step 2:** Create Entity Classes

**Employee Entity**

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "Employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "Id")

private Long id;

@Column(name = "Name")

private String name;

@Column(name = "Designation")

private String designation;

@Column(name = "Salary")

private Double salary;

public Employee() {}

public Employee(String name, String designation, Double salary) {

write code here

}

}

**Product Entity**

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "Product")

@SequenceGenerator(name = "product\_seq", sequenceName = "product\_seq", allocationSize = 1)

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "product\_seq")

@Column(name = "ProductId")

private Long productId;

@Column(name = "ProductName")

private String productName;

@Column(name = "Price")

private Double price;

public Product() {}

public Product(String productName, Double price) {

write code here

}

}

**Step 3:** Insert Records

Main Class to Insert Records

package com.example;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

write code here

}

}

**12.2 OBJECTS**

1. Assume that the following data exists in the table that you have created earlier

**FlowerID FlowerName Color Price**

1 Rose Red 10

2 Carnation White 15

Retrieve and display the details of the id 2 - flower- ‘Carnation’.

**Step 1:** Hibernate Configuration and Dependencies

This has been covered in the previous steps. Ensure you have Hibernate and database dependencies in your project and a proper Hibernate configuration.

**Step 2:** Create the Flower Entity Class

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "Flower")

public class Flower {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "FlowerID")

private Long flowerId;

@Column(name = "FlowerName")

private String flowerName;

@Column(name = "Color")

private String color;

@Column(name = "Price")

private Double price;

public Flower() {}

public Flower(String flowerName, String color, Double price) {

write code here

}

public Long getFlowerId() {

return flowerId;

}

public void setFlowerId(Long flowerId) {

this.flowerId = flowerId;

}

public String getFlowerName() {

return flowerName;

}

public void setFlowerName(String flowerName) {

this.flowerName = flowerName;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public Double getPrice() {

return price;

}

public void setPrice(Double price) {

this.price = price;

}

}

**Step 3:** Retrieve and Display the Record

package com.example;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class RetrieveFlower {

public static void main(String[] args) {

write code here

}

}

1. Write a Hibernate Program to Delete a Flower with its Id from the table that you have created earlier. If there is no Flower with that id exists, then an appropriate error message needs to be stored to the user.

package com.example;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class DeleteFlower {

public static void main(String[] args) {

write code here

}

}

1. Write a Hibernate Program that will ask the user to enter an id and price. Using that the corresponding Flower's price need to be updated. After updating, show that the value is updated, by displaying that record.

package com.example;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.util.Scanner;

public class UpdateFlowerPrice {

public static void main(String[] args) {

write code here

}

}

**12.3 Hibernate with Annotations**

1. Write a Hibernate Program which makes using of all the annotations @Entity, @Table, @Id, @Column to create the mapping for the following table Car\_Details

RegNo char(5)

Model varchar2(20)

Color varchar2(10)

Manufacturer varchar2(20)

<dependency>

write code here

</dependency>

<dependency>

write code here

</dependency>

<dependency>

write code here

</dependency>

<dependency>

write code here

</dependency>

**build.gradle**

implementation 'org.hibernate:hibernate-core:5.4.32.Final'

implementation 'org.hibernate:hibernate-entitymanager:5.4.32.Final'

implementation 'javax.persistence:javax.persistence-api:2.2'

implementation 'org.postgresql:postgresql:42.2.19'

Here is the Hibernate entity class for Car\_Details:

import javax.persistence.\*;

@Entity

@Table(name = "Car\_Details")

public class CarDetails {

@Id

@Column(name = "RegNo", length = 5, nullable = false)

private String regNo;

@Column(name = "Model", length = 20)

private String model;

@Column(name = "Color", length = 10)

private String color;

@Column(name = "Manufacturer", length = 20)

private String manufacturer;

public String getRegNo() {

return regNo;

}

public void setRegNo(String regNo) {

this.regNo = regNo;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getManufacturer() {

return manufacturer;

}

public void setManufacturer(String manufacturer) {

this.manufacturer = manufacturer;

}

}

**hibernate.cfg.xml**

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

write code here

</session-factory>

</hibernate-configuration>

**Main Class to Test CRUD Operations**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Insert the following records into the Car-Detail table

**RegNo| Model| Color| Manufacturer**

KL-07| AB 123 Polo| White| Volkswagen

KL-07| AB 234 Vento| Black| Volkswagen

KL-07| AC 345 Corolla| Silver| Toyota

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

write code here

}

}

**12.4 HQL**

1. Display the details of all cars from the following table using HQL.

RegNo Model Color Manufacturer

KL-07 AB 123 Polo White Volkswagen

KL-07 AB234 Vento Black Volkswagen

KL-07 AC 345 Corolla Silver Toyota

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.query.Query;

import java.util.List;

public class Main {

public static void main(String[] args) {

write code here

}

private static void insertCarDetails(SessionFactory factory) {

write code here

}

private static void displayAllCarDetails(SessionFactory factory) {

write code here

}

}

1. Write a Hibernate program to display only the RegNo and the Manufacturer's name in the above program

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.query.Query;

import java.util.List;

public class Main {

public static void main(String[] args) {

write code here

}

private static void insertCarDetails(SessionFactory factory) {

write code here

}

private static void displayRegNoAndManufacturer(SessionFactory factory) {

write code here

}

}

1. Write a Hibernate program which will ask the user to enter a registration number and display the details of that Vehicle. Assign the value to the query using a label.

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.query.Query;

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

write code here

}

private static void insertCarDetails(SessionFactory factory) {

write code here

}

private static void displayVehicleDetails(SessionFactory factory, String regNo) {

write code here

}

}

1. Write a Hibernate program to display only the Manufacturer's names which are starting with ‘V’ from above table

import javax.persistence.\*;

@Entity

@Table(name = "Car\_Details")

public class CarDetails {

@Id

@Column(name = "RegNo", length = 10)

private String regNo;

@Column(name = "Model", length = 20)

private String model;

@Column(name = "Color", length = 10)

private String color;

@Column(name = "Manufacturer", length = 20)

private String manufacturer;

public String getRegNo() {

return regNo;

}

public void setRegNo(String regNo) {

this.regNo = regNo;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getManufacturer() {

return manufacturer;

}

public void setManufacturer(String manufacturer) {

this.manufacturer = manufacturer;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.query.Query;

import java.util.List;

public class Main {

public static void main(String[] args) {

write code here

}

private static void insertCarDetails(SessionFactory factory) {

write code here

}

private static void displayManufacturersStartingWithV(SessionFactory factory) {

write code here

}

}

1. Write a Hibernate program to display cars details based on color [Hint: Use Criteria Query ]

**CarDetails.java**

import javax.persistence.\*;

@Entity

@Table(name = "Car\_Details")

public class CarDetails {

@Id

@Column(name = "RegNo", length = 10)

private String regNo;

@Column(name = "Model", length = 20)

private String model;

@Column(name = "Color", length = 10)

private String color;

@Column(name = "Manufacturer", length = 20)

private String manufacturer;

public String getRegNo() {

return regNo;

}

public void setRegNo(String regNo) {

this.regNo = regNo;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getManufacturer() {

return manufacturer;

}

public void setManufacturer(String manufacturer) {

this.manufacturer = manufacturer;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.criterion.Restrictions;

import org.hibernate.Criteria;

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

write code here

}

private static void insertCarDetails(SessionFactory factory) {

write code here

}

private static void promptAndDisplayDetailsByColor(SessionFactory factory)

{

write code here

}

}

1. Write a Hibernate program to display total count of cars based on Reg No [Hint: Use Criteria Query ]

**CarDetails.java**

import javax.persistence.\*;

@Entity

@Table(name = "Car\_Details")

public class CarDetails {

@Id

@Column(name = "RegNo", length = 10)

private String regNo;

@Column(name = "Model", length = 20)

private String model;

@Column(name = "Color", length = 10)

private String color;

@Column(name = "Manufacturer", length = 20)

private String manufacturer;

public String getRegNo() {

return regNo;

}

public void setRegNo(String regNo) {

this.regNo = regNo;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getManufacturer() {

return manufacturer;

}

public void setManufacturer(String manufacturer) {

this.manufacturer = manufacturer;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import org.hibernate.criterion.Projections;

import org.hibernate.Criteria;

import java.util.List;

public class Main {

public static void main(String[] args) {

write code here

}

private static void insertCarDetails(SessionFactory factory) {

write code here

}

private static void displayTotalCountOfCarsByRegNo(SessionFactory factory)

{

write code here

}

}

**12.5 Mapping**

1. Create a program to persist the following class object using Association Mapping where car would be on OneToOne mapping:

public class CarOwner {

private int OnnerId;

private String OnerName;

private CarDetail car;

}

**Step 1:** Define the CarOwner class

**CarOwner.java**

import javax.persistence.\*;

@Entity

@Table(name = "Car\_Owner")

public class CarOwner {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "OwnerId")

private int ownerId;

@Column(name = "OwnerName", length = 50)

private String ownerName;

@OneToOne(cascade = CascadeType.ALL)

@JoinColumn(name = "RegNo")

private CarDetails car;

public int getOwnerId() {

return ownerId;

}

public void setOwnerId(int ownerId) {

this.ownerId = ownerId;

}

public String getOwnerName() {

return ownerName;

}

public void setOwnerName(String ownerName) {

this.ownerName = ownerName;

}

public CarDetails getCar() {

return car;

}

public void setCar(CarDetails car) {

this.car = car;

}

}

**Step 2**: Modify the CarDetails class

**CarDetails.java**

import javax.persistence.\*;

@Entity

@Table(name = "Car\_Details")

public class CarDetails {

@Id

@Column(name = "RegNo", length = 10)

private String regNo;

@Column(name = "Model", length = 20)

private String model;

@Column(name = "Color", length = 10)

private String color;

@Column(name = "Manufacturer", length = 20)

private String manufacturer;

public String getRegNo() {

return regNo;

}

public void setRegNo(String regNo) {

this.regNo = regNo;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public String getColor() {

return color;

}

public void setColor(String color) {

this.color = color;

}

public String getManufacturer() {

return manufacturer;

}

public void setManufacturer(String manufacturer) {

this.manufacturer = manufacturer;

}

}

**Step 3**: Create the Hibernate configuration and main class

hibernate.cfg.xml

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

write code here

</session-factory>

</hibernate-configuration>

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

write code here

}

1. Make another class called MultiCarOwner program to persist the following class object using Collection Mapping of one owner to many cars mapping:

public class CarOwner {

private int OnnerId;

private String OnerName;

private List cars;

}

**CarOwner.java**

import javax.persistence.\*;

import java.util.List;

@Entity

@Table(name = "CarOwner")

public class CarOwner {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "OwnerId")

private int ownerId;

@Column(name = "OwnerName")

private String ownerName;

@OneToMany(cascade = CascadeType.ALL)

@JoinColumn(name = "OwnerId")

private List<Car> cars;

public CarOwner() {

}

public CarOwner(String ownerName, List<Car> cars) {

this.ownerName = ownerName;

this.cars = cars;

}

public int getOwnerId() {

return ownerId;

}

public void setOwnerId(int ownerId) {

this.ownerId = ownerId;

}

public String getOwnerName() {

return ownerName;

}

public void setOwnerName(String ownerName) {

this.ownerName = ownerName;

}

public List<Car> getCars() {

return cars;

}

public void setCars(List<Car> cars) {

this.cars = cars;

}

}

**Car.java**

import javax.persistence.\*;

@Entity

@Table(name = "Car")

public class Car {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "CarId")

private int carId;

@Column(name = "RegNo")

private String regNo;

@Column(name = "Model")

private String model;

@ManyToOne

@JoinColumn(name = "OwnerId")

private CarOwner owner;

public Car() {

}

public Car(String regNo, String model) {

this.regNo = regNo;

this.model = model;

}

public int getCarId() {

return carId;

}

public void setCarId(int carId) {

this.carId = carId;

}

public String getRegNo() {

return regNo;

}

public void setRegNo(String regNo) {

this.regNo = regNo;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public CarOwner getOwner() {

return owner;

}

public void setOwner(CarOwner owner) {

this.owner = owner;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.util.ArrayList;

import java.util.List;

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration().configure().buildSessionFactory();

List<Car> cars = new ArrayList<>();

cars.add(new Car("KL-07AB123", "Polo"));

cars.add(new Car("KL-07AB234", "Vento"));

CarOwner carOwner = new CarOwner("John Doe", cars);

persistCarOwner(factory, carOwner);

int ownerId = carOwner.getOwnerId();

CarOwner retrievedOwner = retrieveCarOwner(factory, ownerId);

if (retrievedOwner != null) {

System.out.println("Retrieved Car Owner:");

System.out.println("Owner ID: " + retrievedOwner.getOwnerId());

System.out.println("Owner Name: " + retrievedOwner.getOwnerName());

System.out.println("Cars:");

for (Car car : retrievedOwner.getCars()) {

System.out.println("Car ID: " + car.getCarId());

System.out.println("Reg No: " + car.getRegNo());

System.out.println("Model: " + car.getModel());

System.out.println("-----------");

}

}

factory.close();

}

private static void persistCarOwner(SessionFactory factory, CarOwner carOwner) {

Session session = factory.openSession();

Transaction transaction = session.beginTransaction();

session.save(carOwner);

transaction.commit();

session.close();

}

private static CarOwner retrieveCarOwner(SessionFactory factory, int ownerId) {

Session session = factory.openSession();

CarOwner carOwner = session.get(CarOwner.class, ownerId);

session.close();

return carOwner;

}

}

1. Create a program to persist the following class 10 objects using Collection Mapping:

Public class Continent{

Private String cname;

Private Map countries;

}

Note : Countries property should be of type Map object with Country String class object as key and Capital String class object as value Hibernate

**Entity Classes:**

**Continent.java**

import javax.persistence.\*;

import java.util.\*;

@Entity

@Table(name = "Continent")

public class Continent {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "ContinentId")

private int continentId;

@Column(name = "CName")

private String cname;

@ElementCollection

@CollectionTable(name = "Country\_Capital", joinColumns = @JoinColumn(name = "ContinentId"))

@MapKeyColumn(name = "Country")

@Column(name = "Capital")

private Map<String, String> countries = new HashMap<>();

public Continent() {

}

public Continent(String cname, Map<String, String> countries) {

this.cname = cname;

this.countries = countries;

}

public int getContinentId() {

return continentId;

}

public void setContinentId(int continentId) {

this.continentId = continentId;

}

public String getCname() {

return cname;

}

public void setCname(String cname) {

this.cname = cname;

}

public Map<String, String> getCountries() {

return countries;

}

public void setCountries(Map<String, String> countries) {

this.countries = countries;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.util.HashMap;

import java.util.Map;

public class Main {

public static void main(String[] args) {

write code here

}

private static void persistContinent(SessionFactory factory, Continent continent) {

write code here

}

private static Continent retrieveContinent(SessionFactory factory, int continentId) {

write code here

}

}

1. Write an Employee class and Passport class such that, one employee object should hold only one passport object (one-to-one). Implement a client code such that when we save or delete Employee object, automatically passport object should be stored or deleted in a related table. Note: you can add any appropriate class members

Entity Classes:

**Employee.java**

import javax.persistence.\*;

@Entity

@Table(name = "Employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "EmployeeId")

private int employeeId;

@Column(name = "EmployeeName")

private String employeeName;

@OneToOne(mappedBy = "employee", cascade = CascadeType.ALL)

private Passport passport;

public Employee() {

}

public Employee(String employeeName, Passport passport) {

this.employeeName = employeeName;

this.passport = passport;

}

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

public String getEmployeeName() {

return employeeName;

}

public void setEmployeeName(String employeeName) {

this.employeeName = employeeName;

}

public Passport getPassport() {

return passport;

}

public void setPassport(Passport passport) {

this.passport = passport;

}

}

**Passport.java**

import javax.persistence.\*;

@Entity

@Table(name = "Passport")

public class Passport {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "PassportId")

private int passportId;

@Column(name = "PassportNumber")

private String passportNumber;

@OneToOne

@JoinColumn(name = "EmployeeId")

private Employee employee;

public Passport() {

}

public Passport(String passportNumber) {

this.passportNumber = passportNumber;

}

public int getPassportId() {

return passportId;

}

public void setPassportId(int passportId) {

this.passportId = passportId;

}

public String getPassportNumber() {

return passportNumber;

}

public void setPassportNumber(String passportNumber) {

this.passportNumber = passportNumber;

}

public Employee getEmployee() {

return employee;

}

public void setEmployee(Employee employee) {

this.employee = employee;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

write code here

}

private static void saveEmployee(SessionFactory factory, Employee employee) {

write code here

}

private static Employee retrieveEmployee(SessionFactory factory, int employeeId) {

write code here

}

private static void deleteEmployee(SessionFactory factory, Employee employee) {

write code here

}

}

1. Convert the Employee class with address class such that, one employee object should have only one address object (one-to-one). As a Component Mapping Such that there is only one employee table and the address details are added as columns to the Employee table

**Employee.java**

import javax.persistence.\*;

@Entity

@Table(name = "Employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "EmployeeId")

private int employeeId;

@Column(name = "EmployeeName")

private String employeeName;

@Embedded

private Address address;

public Employee() {

}

public Employee(String employeeName, Address address) {

this.employeeName = employeeName;

this.address = address;

}

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

public String getEmployeeName() {

return employeeName;

}

public void setEmployeeName(String employeeName) {

this.employeeName = employeeName;

}

public Address getAddress() {

return address;

}

public void setAddress(Address address) {

this.address = address;

}

}

**Address.java**

import javax.persistence.\*;

@Embeddable

public class Address {

@Column(name = "Street")

private String street;

@Column(name = "City")

private String city;

@Column(name = "State")

private String state;

@Column(name = "ZipCode")

private String zipCode;

public Address() {

}

public Address(String street, String city, String state, String zipCode) {

write code here

}

public String getStreet() {

return street;

}

public void setStreet(String street) {

this.street = street;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

public String getZipCode() {

return zipCode;

}

public void setZipCode(String zipCode) {

this.zipCode = zipCode;

}

}

**Main.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

write code here

}

private static void saveEmployee(SessionFactory factory, Employee employee) {

write code here

}

private static Employee retrieveEmployee(SessionFactory factory, int employeeId) {

write code here

}

private static void updateEmployeeAddress(SessionFactory factory, Employee employee, Address updatedAddress) {

write code here

}

private static void deleteEmployee(SessionFactory factory, Employee employee) {

write code here

}

}

**13. SPRING**

**13.1 Spring Basics**

1. A Model class “Movie” has the below properties

Sensitivity: Internal & Restricted

• movieId – String – E.g.: M001

• movieName – String – E.g.: The Firm

• movieActor – String – E.g.: Tom Cruise

• Create a spring xml configuration file to create a bean entry for the Movie object with some sample property values

The client program should be able to display all the details of the Movie object on the console.

Model Class:

**Movie.java**

public class Movie {

private String movieId;

private String movieName;

private String movieActor;

public Movie() {

}

public Movie(String movieId, String movieName, String movieActor) {

this.movieId = movieId;

this.movieName = movieName;

this.movieActor = movieActor;

}

public String getMovieId() {

return movieId;

}

public void setMovieId(String movieId) {

this.movieId = movieId;

}

public String getMovieName() {

return movieName;

}

public void setMovieName(String movieName) {

this.movieName = movieName;

}

public String getMovieActor() {

return movieActor;

}

public void setMovieActor(String movieActor) {

this.movieActor = movieActor;

}

}

Spring XML Configuration:

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Client Program:**

Main.java

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create a class called DefaultMessage with a single member variable called message which is initialized with a default value "Spring". Instantiate this class and test

Class Definition:

**DefaultMessage.java**

public class DefaultMessage {

private String message = "Spring";

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

}

Spring XML Configuration:

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

Client Program:

**Main.java**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

write code here

}

}

**13.2 SPRING INVERSION OF CONTROL**

1. Create an abstract parent class called Shape with an abstract draw method and create subclasses of this class like Rectangle, Triangle and Parallelogram. Define a simple java class called DrawShape, which will instantiate an object depending on the dependency injection and calls appropriate draw method.

Abstract Parent Class and Subclasses:

Shape.java

public abstract class Shape {

public abstract void draw();

}

**Rectangle.java**

public class Rectangle extends Shape {

@Override

public void draw() {

System.out.println("Drawing a Rectangle");

}

}

**Triangle.java**

public class Triangle extends Shape {

@Override

public void draw() {

System.out.println("Drawing a Triangle");

}

}

**Parallelogram.java**

public class Parallelogram extends Shape {

@Override

public void draw() {

System.out.println("Drawing a Parallelogram");

}

}

**DrawShape.java**

public class DrawShape {

private Shape shape;

public DrawShape(Shape shape) {

this.shape = shape;

}

public void drawShape() {

shape.draw();

}

}

Spring XML Configuration:

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Client Program:**

**Main.java**

.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create a class called ConstructorMessage with a single member variable called message which is initialized with null.The class will have a parameterized constructor which initializes the member variable called message. Instantiate this class using Inversion control and test. [Perform Constructor Injection]

Class Definition:

**ConstructorMessage.java**

public class ConstructorMessage {

private String message;

public ConstructorMessage(String message) {

this.message = message;

}

public String getMessage() {

return message;

}

}

Spring XML Configuration:

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Client Program:**

**Main.java**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create a class called SetterMessage with a single member variable called message which is initialized with null. Instantiate this class using Inversion control and test[ Perform Setter Injection ]

There are two Model classes as below:

Student

studentId -> String -> S001

studentName -> String -> Steve

studentTest -> Object of type Test

Test

testId -> String -> T001

testTitle -> String -> “Core Java Test”

testMarks -> int -> 80

**SetterMessage.java**

public class SetterMessage {

private String message;

public void setMessage(String message) {

this.message = message;

}

public String getMessage() {

return message;

}

}

**Student.java**

public class Student {

private String studentId;

private String studentName;

private Test studentTest;

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

write code here

}

public String getStudentName() {

write code here

}

public void setStudentName(String studentName) {

write code here

}

public Test getStudentTest() {

write code here

}

public void setStudentTest(Test studentTest) {

write code here

}

}

**Test.java**

public class Test {

private String testId;

private String testTitle;

private int testMarks;

public String getTestId() {

return testId;

}

public void setTestId(String testId) {

this.testId = testId;

}

public String getTestTitle() {

return testTitle;

}

public void setTestTitle(String testTitle) {

this.testTitle = testTitle;

}

public int getTestMarks() {

return testMarks;

}

public void setTestMarks(int testMarks) {

this.testMarks = testMarks;

}

}

**Spring XML Configuration:**

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Client Program:**

**Main.java**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

write code here

}

}

1. Create a spring program to create 2 student instances. First student has taken up the test with Core Java Test with 80 as score and the Second student has taken Oracle test with 83 as score

In the client program display all the details of both the students along with their test details as

well.

**Class Definitions:**

**// Student.java**

public class Student {

private String studentId;

private String studentName;

private Test studentTest;

public String getStudentId() {

return studentId;

}

public void setStudentId(String studentId) {

this.studentId = studentId;

}

public String getStudentName() {

return studentName;

}

public void setStudentName(String studentName) {

this.studentName = studentName;

}

public Test getStudentTest() {

return studentTest;

}

public void setStudentTest(Test studentTest) {

this.studentTest = studentTest;

}

}

**// Test.java**

public class Test {

private String testId;

private String testTitle;

private int testMarks;

public String getTestId() {

return testId;

}

public void setTestId(String testId) {

this.testId = testId;

}

public String getTestTitle() {

return testTitle;

}

public void setTestTitle(String testTitle) {

this.testTitle = testTitle;

}

public int getTestMarks() {

return testMarks;

}

public void setTestMarks(int testMarks) {

this.testMarks = testMarks;

}

}

**Spring XML Configuration:**

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Client Program:**

**Main.java**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

write code here

}

private static void displayStudentDetails(Student student) {

write code here

}

}

1. There are two Model classes as below:

Player

playerId -> String -> P001

playerName -> String -> Sachin

country -> Object -> of type ‘Country’

Country

countryId -> String -> C001

countryName -> String -> ‘India’

Every player belongs to a particular country. Create a spring program to create 5 players and 2

countries. 2 players belong to one country and the other 3 players belong to another country.

The client program should display the details of all players and their corresponding country

details. If given a country name, it should also display all the player names who belong to that

country.

**Class Definitions:**

**Player.java**

public class Player {

private String playerId;

private String playerName;

private Country country;

public String getPlayerId() {

return playerId;

}

public void setPlayerId(String playerId) {

this.playerId = playerId;

}

public String getPlayerName() {

return playerName;

}

public void setPlayerName(String playerName) {

this.playerName = playerName;

}

public Country getCountry() {

return country;

}

public void setCountry(Country country) {

this.country = country;

}

}

Country.java

public class Country {

private String countryId;

private String countryName;

public String getCountryId() {

return countryId;

}

public void setCountryId(String countryId) {

this.countryId = countryId;

}

public String getCountryName() {

return countryName;

}

public void setCountryName(String countryName) {

this.countryName = countryName;

}

}

**Spring XML Configuration:**

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Client Program:**

Main.java

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import java.util.ArrayList;

import java.util.List;

public class Main {

public static void main(String[] args) {

write code here

}

private static void displayPlayersByCountryName(List<Player> players, String countryName) {

write code here

}

}

**13.3 SPRING MVC**

1. Using Spring MVC write a HelloWorld Program

**Step 1:** Set Up the Project Structure

Create a Maven project with the following structure:

src/main/java/com/example/helloworld/controller/HelloWorldController.java

src/main/resources/spring-mvc-config.xml

src/main/webapp/WEB-INF/web.xml

src/main/webapp/WEB-INF/views/hello.jsp

pom.xml

**Step 2:** Add Spring MVC Dependencies

pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>helloworld</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

write code here

</dependencies>

<build>

write code here

</build>

</project>

**Step 3:** Configure DispatcherServlet in web.xml

**web.xml**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

write code here

</web-app>

**Step 4:** Create Spring Configuration File (spring-mvc-config.xml)

**spring-mvc-config.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Step 5**: Create the Controller

**HelloWorldController.java**

.example.helloworld.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView hello() {

write code here

}

}

**Step 6:** Create the View

**hello.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Hello World</title>

</head>

<body>

<h2>${message}</h2>

</body>

</html>

1. Create a Spring web application where index page has a link requesting to display top view series / movies On click the controller has to return a Model Object of ArrayList containing list of names of the series or movies (Minimum 10) which needs to be displayed in out result.jsp using Expression Language (EL)

**Step 1:** Set Up the Project Structure

Create a Maven project with the following structure:

src/main/java/com/example/movies/controller/MovieController.java

src/main/resources/spring-mvc-config.xml

src/main/webapp/WEB-INF/web.xml

src/main/webapp/WEB-INF/views/index.jsp

src/main/webapp/WEB-INF/views/result.jsp

pom.xml

**Step 2**: Add Spring MVC Dependencies

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>movies</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

write code here

</dependencies>

<build>

write code here

</build>

</project>

**Step 3:** Configure DispatcherServlet in web.xml

**web.xml**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

version="3.1">

<display-name>Spring MVC Movies</display-name>

<servlet>

write code here

</servlet>

<servlet-mapping>

write code here

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

**Step 4**: Create Spring Configuration File (spring-mvc-config.xml)

spring-mvc-config.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

write code here

</beans>

**Step 5:** Create the Controller

MovieController.java

package com.example.movies.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import java.util.ArrayList;

import java.util.List;

@Controller

public class MovieController {

@RequestMapping("/topmovies")

public String showTopMovies(Model model) {

write code here

}

}

**Step 6:** Create Views

**index.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Top Movies</title>

</head>

<body>

<h2>Welcome to the Top Movies Page</h2>

<a href="topmovies">Click here to see the top 10 movies</a>

</body>

</html>

**result.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Top Movies</title>

</head>

<body>

<h2>Top 10 Movies</h2>

<ul>

<c:forEach var="movie" items="${movies}">

<li>${movie}</li>

</c:forEach>

</ul>

<a href="index.jsp">Back to Home</a>

</body>

</html>

1. Create a form which takes three inputs from the User -> Science marks , Maths marks and English marks After entering the data, when the user clicks on the submit button, it has to calculate the sum of these numbers and show the total marks back to the user. Use Spring MVC for this solution and use XML configuration

**Step 1:** Set Up the Project Structure

Create a Maven project with the following structure:

src/main/java/com/example/marks/controller/MarksController.java

src/main/resources/spring-mvc-config.xml

src/main/webapp/WEB-INF/web.xml

src/main/webapp/WEB-INF/views/index.jsp

src/main/webapp/WEB-INF/views/result.jsp

pom.xml

**Step 2:** Add Spring MVC Dependencies

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

write code here

</project>

**Step 3:** Configure DispatcherServlet in web.xml

**web.xml**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

version="3.1">

<display-name>Spring MVC Marks</display-name>

<servlet>

write code here

</servlet>

<servlet-mapping>

write code here

</servlet-mapping>

<welcome-file-list>

write code here

</welcome-file-list>

</web-app>

**Step 4:** Create Spring Configuration File (spring-mvc-config.xml)

**spring-mvc-config.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<context:component-scan base-package="com.example.marks.controller"/>

<mvc:annotation-driven/>

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/"/>

<property name="suffix" value=".jsp"/>

</bean>

</beans>

**Step 5:** Create the Controller

MarksController.java

package com.example.marks.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

public class MarksController {

@RequestMapping(value = "/", method = RequestMethod.GET)

public String showForm(Model model) {

write code here

}

@RequestMapping(value = "/calculate", method = RequestMethod.POST)

public String calculateTotal(@ModelAttribute("marks") Marks marks, Model model) {

write code here

}

}

**Marks.java**

package com.example.marks.controller;

public class Marks {

private int scienceMarks;

private int mathsMarks;

private int englishMarks;

public int getScienceMarks() {

return scienceMarks;

}

public void setScienceMarks(int scienceMarks) {

this.scienceMarks = scienceMarks;

}

public int getMathsMarks() {

return mathsMarks;

}

public void setMathsMarks(int mathsMarks) {

this.mathsMarks = mathsMarks;

}

public int getEnglishMarks() {

return englishMarks;

}

public void setEnglishMarks(int englishMarks) {

this.englishMarks = englishMarks;

}

}

**Step 6**: Create Views

**index.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<html>

<head>

<title>Marks Form</title>

</head>

<body>

<h2>Enter your marks</h2>

<form:form method="post" action="calculate" modelAttribute="marks">

<table>

write code here

</table>

</form:form>

</body>

</html>

**result.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Total Marks</title>

</head>

<body>

write code here

</body>

</html>

1. Create a Spring web application where index page has a link requesting to display top view series / movies On click the controller has to return a Model Object of ArrayList containing list of names of the series or movies (Minimum 10) which needs to be displayed in out result.jsp using JSTL (HINT : forEach tag)

**Step 1**: Set Up the Project Structure

Create a Maven project with the following structure:

src/main/java/com/example/movies/controller/MovieController.java

src/main/resources/spring-mvc-config.xml

src/main/webapp/WEB-INF/web.xml

src/main/webapp/WEB-INF/views/index.jsp

src/main/webapp/WEB-INF/views/result.jsp

pom.xml

**Step 2**: Add Spring MVC Dependencies

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>movies</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

write code here

</dependencies>

<build>

write code here

</build>

</project>

**Step 3:** Configure DispatcherServlet in web.xml

**web.xml**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"

version="3.1">

<display-name>Spring MVC Movies</display-name>

<servlet>

write code here

</servlet>

<servlet-mapping>

write code here

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

**Step 4:** Create Spring Configuration File (spring-mvc-config.xml)

spring-mvc-config.xml

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd">

<context:component-scan base-package="com.example.movies.controller" />

<mvc:annotation-driven />

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp" />

</bean>

</beans>

**Step 5:** Create the Controller

**MovieController.java**

package com.example.movies.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import java.util.ArrayList;

import java.util.List;

@Controller

public class MovieController {

write code here

}

}

**Step 6:** Create Views

**index.jsp**

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<html>

<head>

<title>Top Movies</title>

</head>

<body>

write code here

</body>

</html>

**13.4 SPRINGMVC WITH Hibernate**

1. Create a Spring based MVC application which collects Student details from the user and inserts them into the database. To create a Spring MVC application that collects student details from the user and inserts them into a database, we'll need to set up the project, configure Spring MVC, create the necessary database schema, and implement the controller and views. Here’s a step-by-step guide to achieve this:

**Step 1:** Project Setup

Create a Maven Project: Use your preferred IDE or command-line tool to create a new Maven project.

Add Dependencies: Include Spring MVC, JDBC, MySQL (or your preferred database driver), and JSTL dependencies in your pom.xml.

Example pom.xml dependencies:

<dependencies>

<!-- Spring MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.9</version>

</dependency>

<!-- Spring JDBC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>5.3.9</version>

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.28</version>

</dependency>

<!-- JSTL for JSP -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

**Step 2:** Database Configuration

**SQL script:**

CREATE DATABASE IF NOT EXISTS school;

USE school;

CREATE TABLE IF NOT EXISTS students (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) NOT NULL,

age INT,

address VARCHAR(255)

);

Configure Database Connection:

Configure database connection properties in application.properties or dataSource bean in Spring XML configuration (if using XML configuration).

Example application.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/school

spring.datasource.username=root

spring.datasource.password=password

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

**Step 3:** Spring MVC Configuration

**web.xml (if using XML configuration):**

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<display-name>SpringMVCStudentExample</display-name>

<!-- Configure Spring MVC DispatcherServlet -->

<servlet>

write code here

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<!-- Character Encoding Filter -->

<filter>

write code here

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

</web-app>

Create Spring Configuration File:

Configure Spring beans, database connection, and transaction management.

Example spring-mvc-config.xml (if using XML configuration):

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx.xsd">

<context:component-scan base-package="com.example.students.controller" />

<!-- Enable Spring MVC annotation-driven -->

<mvc:annotation-driven />

<!-- DataSource Configuration -->

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />

<property name="url" value="jdbc:mysql://localhost:3306/school" />

<property name="username" value="root" />

<property name="password" value="password" />

</bean>

<!-- Hibernate Configuration -->

<bean id="sessionFactory"

class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

</props>

</property>

<property name="packagesToScan" value="com.example.students.model" />

</bean>

<!-- Transaction Management -->

<tx:annotation-driven />

<bean id="transactionManager"

class="org.springframework.orm.hibernate5.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

</beans>

**Step 4:** Create Entity and DAO

Create Entity Class:

Define Student entity class with annotations for ORM mapping.

**Example Student.java:**

package com.example.students.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Student {

write code here

}

**StudentDAO.java:**

package com.example.students.dao;

import com.example.students.model.Student;

public interface StudentDAO {

void save(Student student);

}

StudentDAOImpl.java:

package com.example.students.dao;

import com.example.students.model.Student;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

@Repository

public class StudentDAOImpl implements StudentDAO {

@Autowired

private SessionFactory sessionFactory;

@Override

public void save(Student student) {

Session session = sessionFactory.getCurrentSession();

session.save(student);

}

}

**Step 5:** Create Controller and Views

**Example StudentController.java:**

package com.example.students.controller;

import com.example.students.dao.StudentDAO;

import com.example.students.model.Student;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

@RequestMapping("/student")

public class StudentController {

@Autowired

private StudentDAO studentDAO;

@RequestMapping(value = "/form", method = RequestMethod.GET)

public String showForm(Model model) {

model.addAttribute("student", new Student());

return "studentForm";

}

1. Create a Spring based MVC application which displays all the Student details which were inserted earlier

Add a method to retrieve all students from the database.

package com.example.students.dao;

import com.example.students.model.Student;

import java.util.List;

public interface StudentDAO {

void save(Student student);

List<Student> getAllStudents();

}

Implement getAllStudents() Method in StudentDAOImpl.java:

package com.example.students.dao;

import com.example.students.model.Student;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public class StudentDAOImpl implements StudentDAO {

@Autowired

private SessionFactory sessionFactory;

@Override

public void save(Student student) {

Session session = sessionFactory.getCurrentSession();

session.save(student);

}

@Override

public List<Student> getAllStudents() {

Session session = sessionFactory.getCurrentSession();

return session.createQuery("from Student", Student.class).getResultList();

}

}

**Step 3:** Create a Controller to Display Student Details

Create StudentController.java:

package com.example.students.controller;

import com.example.students.dao.StudentDAO;

import com.example.students.model.Student;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import java.util.List;

@Controller

@RequestMapping("/student")

public class StudentController {

@Autowired

private StudentDAO studentDAO;

@RequestMapping(value = "/list", method = RequestMethod.GET)

public String listStudents(Model model) {

write code here

}

}

**Step 4:** Create JSP View to Display Student Details

Create studentList.jsp:

<%@ page contentType="text/html;charset=UTF-8" language="java" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>

<head>

<title>Student List</title>

</head>

<body>

write code here

</body>

</html>

**14.SPRINGBOOT**

**14.1 INTRODUCTION**

1. Create a Spring Boot application to display a message "Hello ".

**Step 1:** Set Up a Spring Boot Project

**pom.xml dependencies:**

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

**Step 2:** Create a Controller

HelloController.java:

package com.example.demo;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloController {

@GetMapping("/")

public String hello() {

return "Hello World";

}

}

**14.2 SPRINGBOOT Dependency Injection & Auto wiring**

1. Define three bean class called Laptop, Hard disk and Battery. And achieve dependency injection and auto wired by using Spring Boot.

**Step 1:** Set Up a Spring Boot Project

**pom.xml dependencies:**

<dependencies>

write code here

</dependencies>

**Step 2:** Define Bean Classes

**Laptop.java:**

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class Laptop {

private HardDisk hardDisk;

private Battery battery;

@Autowired

public Laptop(HardDisk hardDisk, Battery battery) {

write code here

}

public String getDetails() {

write code here

}

}

**HardDisk.java:**

package com.example.demo;

import org.springframework.stereotype.Component;

@Component

public class HardDisk {

public String getDetails() {

write code here

}

}

**Battery.java:**

package com.example.demo;

import org.springframework.stereotype.Component;

@Component

public class Battery {

public String getDetails() {

write code here

}

}

**DemoApplication.java:**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

write code here

}

}

**usage in a controller:**

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class LaptopController {

@Autowired

private Laptop laptop;

@GetMapping("/laptop")

public String getLaptopDetails() {

write code here

}

}

**14.3 SPRING WEBApp MVC**

1. Create a login page web application by using Spring Boot MVC. And if the user name is and the password is "Wipro@123" then redirect to success page else redirect to failure page.

**Step 1**: Set Up a Spring Boot Project

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

**Step 2**: Create Login Controller

Create Login Controller:

package com.example.demo;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class LoginController {

@GetMapping("/login")

public String showLoginForm() {

return "login";

}

@PostMapping("/login")

public ModelAndView login(@RequestParam String username, @RequestParam String password) {

write code here

}

@GetMapping("/success")

public String success() {

return "success";

}

@GetMapping("/failure")

public String failure() {

return "failure";

}

}

**Step 3:** Create HTML Pages

Create Login Form (login.html):

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login Page</title>

</head>

<body>

<h2>Login</h2>

<form action="/login" method="post">

<label>Username: <input type="text" name="username"></label><br>

<label>Password: <input type="password" name="password"></label><br>

<input type="submit" value="Login">

</form>

</body>

</html>

**Create Success Page (success.html):**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login Success</title>

</head>

<body>

<h2>Login Successful!</h2>

<p>Welcome to the application.</p>

</body>

</html>

**Create Failure Page (failure.html):**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Login Failure</title>

</head>

<body>

<h2>Login Failed!</h2>

<p>Invalid username or password. Please try again.</p>

</body>

</html>

**14.4 SPRINGBOOT MVC JPA H2**

1. Create a web application by using Spring Boot MVC for "Employee Management System". And perform all the CRUD operations by using JPA and H2.

**Step 1:** Set Up a Spring Boot Project

Create a new Spring Boot Project:

<dependencies>

write code here

</dependencies>

**Step 2:** Define Entity Class

**Example Employee.java:**

package com.example.demo.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String email;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**Step 3:** Create Repository Interface

**Example EmployeeRepository.java:**

package com.example.demo.repository;

import com.example.demo.model.Employee;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**Step 4:** Create Service Layer

**Example EmployeeService.java:**

package com.example.demo.service;

import com.example.demo.model.Employee;

import java.util.List;

public interface EmployeeService {

List<Employee> getAllEmployees();

Employee getEmployeeById(Long id);

void saveEmployee(Employee employee);

void deleteEmployee(Long id);

}

**EmployeeServiceImpl.java:**

package com.example.demo.service;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public Employee getEmployeeById(Long id) {

Optional<Employee> optionalEmployee = employeeRepository.findById(id);

return optionalEmployee.orElse(null);

}

@Override

public void saveEmployee(Employee employee) {

employeeRepository.save(employee);

}

@Override

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

**Step 5:** Create Controller

**EmployeeController.java:**

package com.example.demo.controller;

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@Controller

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@GetMapping("/list")

public String listEmployees(Model model) {

write code here

}

@GetMapping("/showFormForAdd")

public String showFormForAdd(Model model) {

write code here

}

@PostMapping("/saveEmployee")

public String saveEmployee(@ModelAttribute("employee") Employee employee) {

write code here

}

@GetMapping("/showFormForUpdate")

public String showFormForUpdate(@RequestParam("employeeId") Long id, Model model) {

write code here

}

@GetMapping("/delete")

public String deleteEmployee(@RequestParam("employeeId") Long id) {

write code here

}

}

**Step 6:** Create Views (HTML Templates)

**Example employee-list.html (under src/main/resources/templates):**

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Employee List</title>

</head>

<body>

write code here

</body>

</html>

**employee-form.html:**

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8">

<title>Employee Form</title>

</head>

<body>

write code here

</body>

</html>

**14.5 SPRINGBOOT MVC JPA H2 REST**

1. Create a web application by using Spring Boot MVC for "Employee Management System". And perform all the CRUD operations by using JPA, H2 and REST web service.

**Step 1:** Set Up a Spring Boot Project

**Example pom.xml dependencies:**

<dependencies>

write code here

</dependencies>

**Step 2:** Define Entity Class

**Example Employee.java:**

package com.example.demo.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String firstName;

private String lastName;

private String email;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**Step 3**: Create Repository Interface

**Example EmployeeRepository.java:**

package com.example.demo.repository;

import com.example.demo.model.Employee;

import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**Step 4:** Create Service Layer

**Example EmployeeService.java:**

package com.example.demo.service;

import com.example.demo.model.Employee;

import java.util.List;

public interface EmployeeService {

List<Employee> getAllEmployees();

Employee getEmployeeById(Long id);

void saveEmployee(Employee employee);

void deleteEmployee(Long id);

}

**EmployeeServiceImpl.java:**

package com.example.demo.service;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public Employee getEmployeeById(Long id) {

Optional<Employee> optionalEmployee = employeeRepository.findById(id);

return optionalEmployee.orElse(null);

}

@Override

public void saveEmployee(Employee employee) {

employeeRepository.save(employee);

}

@Override

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

**Step 5:** Create REST Controller

**Example EmployeeRestController.java:**

package com.example.demo.controller;

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeRestController {

@Autowired

private EmployeeService employeeService;

@GetMapping

public List<Employee> getAllEmployees() {

write code here

}

@GetMapping("/{id}")

public Employee getEmployeeById(@PathVariable Long id) {

write code here

}

@PostMapping

public Employee addEmployee(@RequestBody Employee employee) {

write code here

}

@PutMapping("/{id}")

public Employee updateEmployee(@PathVariable Long id, @RequestBody Employee employee) {

write code here

}

@DeleteMapping("/{id}")

public String deleteEmployee(@PathVariable Long id) {

write code here

}

}

**15.ReactJS**

* 1. **ReactJS overview and installation**

1. ReactJS is a JavaScript library used for building reusable components. We can build composable user interfaces. React is used in V in MVC. It uses JavaScript Virtual DOM that is faster than the regular DOM.
2. Installing ReacjJS

**Step 1:** Set Up ReactJS

**Install Node.js**

Download node.js from https://nodejs.org/en/download/package-manager

Install node-v20.16.0-x64.msi

Open command prompt and verify node installation

node –v

npm –v

**Install ReactJS**

On the same command prompt type the following command

npm install create-react-app

npx create-react-app my-app

cd my-app

npm start

**Step 2:** ReactJS directory structure

my-app

README.md

node\_modules

package.json

.gitignore

public

favicon.ico

index.html

logo192.png

logo512.png

manifest.json

robots.txt

src

App.css

App.js

App.test.js

index.css

index.js

logo.svg

serviceWorker.js

setupTests.js

**Step 3:** Some important file (index.html)

<!DOCTYPE html>

<html lang = "en">

<head>

<meta charset = "UTF-8">

<title>React App</title>

</head>

<body>

<div id = "app"></div>

<script src = 'index\_bundle.js'></script>

</body>

</html>

**Step 4:** Some important file (App.js)

import React, {Component} from ‘react’;

export default class App extends Component {

render() {

var myStyle = {

fontSize:100%,

color:’#FF0000’

}

return (

<div>

<h1 style={myStyle}>Hello World</h1>

<h3>Here we can add user defined components</h3>

{// single line comment }

{/\* multi line comment \*/}

</div>

);

}

}

* 1. **ReactJS Components**

1. ReactJS Develop a first React component (functional component / class component) as Stateless component and include that component in App.js

**App.js**

import React, {Component} from ‘react’;

export default class App extends Component {

render() {

return (

<div>

<Header/>

<Footer/>

</div>

);

}

}

**Writing Header component as Class component**

**Header.js**

export default class Header extends Component {

fill the content here….

}

Or

**Writing Header component as Function Component**

function Header() {

return <h1>Hi iam Heading Content</h1>;

}

Export default Header;

**Footer.js**

export default class Footer extends Component {

fill the content here….

}

1. Develop a ReactJS Stateful Component

**App.js**

import React from “react”;

class App extends React.Component {

constructor(props) {

super(props);

this.state = { change:true };

}

render() {

return (

<div>

<button

onClick={() => {

this.setState({ change: !this.state.change });

}}

>

Click Here!

</button>

{this.state.change ? (

<h1>Before state change</h1>

) : (

<h1>After state change </h1>

)}

</div>

);

}

}

export default App;

1. Develop another example on Stateful Component

Stateful components are those components which have a state. The state gets initialized in the constructor. It stores information about the component’s state change in memory. It may get changed depending upon the action of the component or child components.

* They can hold and manage local state.
* They have lifecycle methods (like componentDidMount, componentDidUpdate, etc.).
* Usually more complex than stateless components.

**Clock.js**

class Clock extends React.Component {

constructor(props) {

super(props);

this.state = {date: new Date()};

}

componentDidMount() {

this.timerID = setInterval(

() => this.tick(),

1000

);

}

componentWillUnmount() {

clearInterval(this.timerID);

}

tick() {

this.setState({

date: new Date()

});

}

render() {

return (

<div>

<h1>Hello, world!</h1>

<h2>It is {this.state.date.toLocaleTimeString()}.</h2>

</div>

);

}

}

Include Clock component in App.js and run the application

* 1. **State, Props and Props Validations**

1. How to pass props into Component and how can we access them in the Component

**// Header.js**

import React from 'react';

function Header() {

const Header = (props) => {

return(

<div>

<h1>{props.branding}</h1>

</div>

);

}

}

export default Header;

**// Contact.js**

import React, {Component} from 'react';

export class Contact extends Component {

render() {

return(

<div>

<h4>{this.props.name}</h4>

<ul>

<li>Email: {this.props.email}</li>

<li>Phone: {this.props.phone}</li>

</ul>

</div>

);

}

}

**// App.js**

import React, {Component} from 'react';

import Contact from './components/Contact';

import Header from './components/Header';

import './App.css';

export default class App extends Component {

render() {

return (

<div class="App">

<h1>The App Component</h1>

<Header branding="Contact Manager"/>

<Contact name="Surya" email="activesurya@gmail.com" phone="9848111288"/>

<Contact name="ABC" email="abc@gmail.com" phone="9848123456"/>

</div>

);

}

}

1. How to perform validations on Component props

**// Header.js**

import PropTypes from 'prop-types';

Header.propTypes = {

branding: PropTypes.object.isRequired

};

**// Contact.js**

import PropTypes from 'prop-types';

Contact.propTypes = {

name: PropTypes.string.isRequired,

email: PropTypes.string.isRequired,

phone: PropTypes.string.isRequired

};

* 1. **React Component API and Component Lifecycle**

1. ReactJS component is a top level API. It makes the code completely individual and reusable in the application. It includes various methods for:
   1. Creating elements
   2. Transforming elements
   3. Fragments

Here, we are going to explain the three most important methods available in the React component API.

1. setState()
2. forceUpdate()
3. findDOMNode()
4. setState() method

This method is used to update the state of the component. This method does not always replace the state immediately. Instead, it only adds changes to the original state. It is a primary method that is used to update the user interface(UI) in response to event handlers and server responses.

Note: In the ES6 classes, this.method.bind(this) is used to manually bind the setState() method.

**// App.js**

import React, { Component } from 'react';

import PropTypes from 'prop-types';

class App extends React.Component {

constructor() {

super();

this.state = {

msg: "Welcome to JavaTpoint"

};

this.updateSetState = this.updateSetState.bind(this);

}

updateSetState() {

this.setState({

msg:"Its a best ReactJS tutorial"

});

}

render() {

return (

<div>

<h1>{this.state.msg}</h1>

<button onClick = {this.updateSetState}>SET STATE</button>

</div>

);

}

}

export default App;

1. forceUpdate()

This method allows us to update the component manually.

**// App.js**

import React, { Component } from 'react';

class App extends React.Component {

constructor() {

super();

this.forceUpdateState = this.forceUpdateState.bind(this);

}

forceUpdateState() {

this.forceUpdate();

};

render() {

return (

<div>

<h1>Example to generate random number</h1>

<h3>Random number: {Math.random()}</h3>

<button onClick = {this.forceUpdateState}>ForceUpdate</button>

</div>

);

}

}

export default App;

1. findDOMNode()

For DOM manipulation you need to use ReactDOM.findDOMNode() method. This method allows us to find or access the underlying DOM node.

**// App.js**

import React, { Component } from 'react';

import ReactDOM from 'react-dom';

class App extends React.Component {

constructor() {

super();

this.findDomNodeHandler1 = this.findDomNodeHandler1.bind(this);

this.findDomNodeHandler2 = this.findDomNodeHandler2.bind(this);

};

findDomNodeHandler1() {

var myDiv = document.getElementById('myDivOne');

ReactDOM.findDOMNode(myDivOne).style.color = 'red';

}

findDomNodeHandler2() {

var myDiv = document.getElementById('myDivTwo');

ReactDOM.findDOMNode(myDivTwo).style.color = 'blue';

}

render() {

return (

<div>

<h1>ReactJS Find DOM Node Example</h1>

<button onClick = {this.findDomNodeHandler1}>FIND\_DOM\_NODE1</button>

<button onClick = {this.findDomNodeHandler2}>FIND\_DOM\_NODE2</button>

<h3 id = "myDivOne">JTP-NODE1</h3>

<h3 id = "myDivTwo">JTP-NODE2</h3>

</div>

);

}

}

export default App;

1. React Component Life Cycle

In ReactJS, every component creation process involves various lifecycle methods. These lifecycle methods are termed as component's lifecycle. These lifecycle methods are not very complicated and called at various points during a component's life. The lifecycle of the component is divided into **four phases**. They are:

1. Initial Phase
2. Mounting Phase
3. Updating Phase
4. Unmounting Phase

Each phase contains some lifecycle methods that are specific to the particular phase. Let us discuss each of these phases one by one.

## 1. Initial Phase

It is the **birth** phase of the lifecycle of a ReactJS component. Here, the component starts its journey on a way to the DOM. In this phase, a component contains the default Props and initial State. These default properties are done in the constructor of a component. The initial phase only occurs once and consists of the following methods.

* **getDefaultProps()**  
  It is used to specify the default value of this.props. It is invoked before the creation of the component or any props from the parent is passed into it.
* **getInitialState()**  
  It is used to specify the default value of this.state. It is invoked before the creation of the component.

## 2. Mounting Phase

In this phase, the instance of a component is created and inserted into the DOM. It consists of the following methods.

* **componentWillMount()**  
  This is invoked immediately before a component gets rendered into the DOM. In the case, when you call **setState()** inside this method, the component will not **re-render**.
* **componentDidMount()**  
  This is invoked immediately after a component gets rendered and placed on the DOM. Now, you can do any DOM querying operations.
* **render()**  
  This method is defined in each and every component. It is responsible for returning a single root **HTML node** element. If you don't want to render anything, you can return a **null** or **false** value.

## 3. Updating Phase

It is the next phase of the lifecycle of a react component. Here, we get new **Props** and change **State**. This phase also allows to handle user interaction and provide communication with the components hierarchy. The main aim of this phase is to ensure that the component is displaying the latest version of itself. Unlike the Birth or Death phase, this phase repeats again and again. This phase consists of the following methods.

* **componentWillRecieveProps()**  
  It is invoked when a component receives new props. If you want to update the state in response to prop changes, you should compare this.props and nextProps to perform state transition by using **this.setState()** method.
* **shouldComponentUpdate()**  
  It is invoked when a component decides any changes/updation to the DOM. It allows you to control the component's behavior of updating itself. If this method returns true, the component will update. Otherwise, the component will skip the updating.
* **componentWillUpdate()**  
  It is invoked just before the component updating occurs. Here, you can't change the component state by invoking **this.setState()** method. It will not be called, if **shouldComponentUpdate()** returns false.
* **render()**  
  It is invoked to examine **this.props** and **this.state** and return one of the following types: React elements, Arrays and fragments, Booleans or null, String and Number. If shouldComponentUpdate() returns false, the code inside render() will be invoked again to ensure that the component displays itself properly.
* **componentDidUpdate()**  
  It is invoked immediately after the component updating occurs. In this method, you can put any code inside this which you want to execute once the updating occurs. This method is not invoked for the initial render.

## 4. Unmounting Phase

It is the final phase of the react component lifecycle. It is called when a component instance is **destroyed** and **unmounted** from the DOM. This phase contains only one method and is given below.

* **componentWillUnmount()**  
  This method is invoked immediately before a component is destroyed and unmounted permanently. It performs any necessary **cleanup** related task such as invalidating timers, event listener, canceling network requests, or cleaning up DOM elements. If a component instance is unmounted, you cannot mount it again.

**// App.js**

import React, { Component } from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {hello: "ActiveNET"};

this.changeState = this.changeState.bind(this)

}

render() {

return (

<div>

<h1>ReactJS component's Lifecycle</h1>

<h3>Hello {this.state.hello}</h3>

<button onClick = {this.changeState}>Click Here!</button>

</div>

);

}

componentWillMount() {

console.log('Component Will MOUNT!')

}

componentDidMount() {

console.log('Component Did MOUNT!')

}

changeState(){

this.setState({hello:"Its a great reactjs tutorial."});

}

componentWillReceiveProps(newProps) {

console.log('Component Will Recieve Props!')

}

shouldComponentUpdate(newProps, newState) {

return true;

}

componentWillUpdate(nextProps, nextState) {

console.log('Component Will UPDATE!');

}

componentDidUpdate(prevProps, prevState) {

console.log('Component Did UPDATE!')

}

componentWillUnmount() {

console.log('Component Will UNMOUNT!')

}

}

export default App;

* 1. **Forms, Events**

Forms are integral part of any modern web application. It allows the users to interact with the application as well as gather information from the users. Forms can perform many tasks that depend on the nature of your business requirements and logic such as authentication of the user, adding user, searching, filtering, booking, ordering etc. A form can contain text fields, buttons, checkboxes, radio buttons etc.

In HTML, form elements typically maintain their own state and update it according to the user input. In the controlled component, the input form element is handled by the component rather than the DOM. Here, the mutable state is kept in the state property and will be updated only with **setState()** method.

Controlled components have functions that govern the data passing into them on every **onChange event**, rather than grabbing the data only once, e.g., when you click a **submit button**. This data is then saved to state and updated with setState() method. This makes component have better control over the form elements and data.

A controlled component takes its current value through **props** and notifies the changes through **callbacks** like an onChange event. A parent component "controls" this changes by handling the callback and managing its own state and then passing the new values as props to the controlled component. It is also called as a "dumb component."

**// App.js**

import React, { Component } from 'react';

class App extends React.Component {

constructor(props) {

super(props);

this.state = {value: ''};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

this.setState({value: event.target.value});

}

handleSubmit(event) {

alert('You have submitted the input successfully: ' + this.state.value);

event.preventDefault();

}

render() {

return (

<form onSubmit={this.handleSubmit}>

<h1>Controlled Form Example</h1>

<label>

Name:

<input type="text" value={this.state.value} onChange={this.handleChange} />

</label>

<input type="submit" value="Submit" />

</form>

);

}

}

export default App;

* 1. **React Routers**

Routing is a process in which a user is directed to different pages based on their action or request. ReactJS Router is mainly used for developing Single Page Web Applications. React Router is used to define multiple routes in the application. When a user types a specific URL into the browser, and if this URL path matches any 'route' inside the router file, the user will be redirected to that particular route.

React Router is a standard library system built on top of the React and used to create routing in the React application using React Router Package. It provides the synchronous URL on the browser with data that will be displayed on the web page. It maintains the standard structure and behavior of the application and mainly used for developing single page web applications.

**Need of React Router**

React Router plays an important role to display multiple views in a single page application. Without React Router, it is not possible to display multiple views in React applications. Most of the social media websites like Facebook, Instagram uses React Router for rendering multiple views.

**React Router installation**

React contains three different packages for routing. These are:

1. **react-router:** It provides the core routing components and functions for the React Router applications.
2. **react-router-native:** It is used for mobile applications.
3. **react-router-dom:** It is used for web applications design.

$ npm install react-router-dom --save

**// About.js**

import React from 'react'

class About extends React.Component {

render() {

return <h1>About</h1>

}

}

export default About

**// Contact.js**

import React from 'react'

class Contact extends React.Component {

render() {

return <h1>Contact</h1>

}

}

export default Contact

**// App.js**

import React from 'react'

class App extends React.Component {

render() {

return (

<div>

<h1>Home</h1>

</div>

)

}

}

export default App

**// index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import { Route, Link, BrowserRouter as Router } from 'react-router-dom'

import './index.css';

import App from './App';

import About from './about'

import Contact from './contact'

const routing = (

<Router>

<div>

<h1>React Router Example</h1>

<ul>

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/about">About</Link>

</li>

<li>

<Link to="/contact">Contact</Link>

</li>

</ul>

<Route exact path="/" component={App} />

<Route path="/about" component={About} />

<Route path="/contact" component={Contact} />

</div>

</Router>

)

ReactDOM.render(routing, document.getElementById('root'));