Capital University of Science and Technology, Islamabad

LAB MANUAL

V1.2 Fall 2021

ENTERPRISE APPLICATION DEVELOPMENT (CS4181 / SE4181)

DEPARTMENT OF COMPUTER SCIENCE, SOFTWARE ENGINEERING

Capital University of Science & Technology
Islamabad
Department of Computer Science,
Software Engineering
Faculty of Computing

Lab Course Development Team

Supervision and Coordination

Mr. Muhammad Sulaiman
Lecturer
Department of Computer Science
Faculty of Computing

Lab Designers

Mr. Shoaib Ali
Jr. Lecturer
Department of Computer Science
Faculty of Computing

Lab Reviewers

Mr. Qaisar Manzoor
Jr. Lecturer
Department of Software Engineering
Faculty of Computing

Table of Contents

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 01

Environment Setup / Web Technology Basics

Lab 01: HTML Environment Setup / Basic Web technology basics

1. Introduction

Using the Java 2 Platform, Standard Edition (J2SE) as a basis, Java 2 Platform, Enterprise Edition (J2EE) builds on top of this to provide the types of services that are necessary to build large scale, distributed, component based, multi-tier applications. Essentially, J2EE is a collection of APIs that can be used to build such systems, although this is only half of the picture. J2EE is also a standard for building and deploying enterprise applications, held together by the specifications of the APIs that it defines and the services that J2EE provides. In other words, this means that the "write once, run anywhere" promises of Java apply for enterprise applications too:

1. Enterprise applications can be run on different platforms supporting the Java 2 platform.
2. Enterprise applications are portable between application servers supporting the J2EE specification.

2. Relevant Lecture Reading

https://www.w3schools.com/html/
https://www.w3schools.com/css/default.asp
https://www.w3schools.com/js/default.asp

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	30 min	30 min
8.1	Walkthrough Task	65 min	60 min
9	Practice Task	75 min	60 min
10	Evaluation Task (Unseen)	30 min	30 min

4. Objective of the Experiment

1. To understand basic concept of what is J2EE.
2. To have a complete understanding of how-to setup J2EE environment.
3. To revise basic knowledge of web technologies like html, CSS and JavaScript.

5. Concept Map

5.1. What is J2EE?

J2EE means Java 2 Enterprise Edition. The functionality of J2EE is developing multitier web-based applications. The J2EE platform is consists of a set of services, application programming interfaces (APIs), and protocols.

5.2. What is HTML?

1. HTML is a **markup** language for **describing** web documents (web pages).
2. HTML stands for Hyper Text Markup Language.
3. A markup language is a set of markup tags.
4. HTML documents are described by HTML tags.
5. Each HTML tag describes different document content.

Example

```
<! DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>My First Heading</h1>
<p>My first paragraph. </p>
</body>
</html>
```

5.2.1. HTML Tags

HTML tags are **keywords** (tag names) surrounded by **angle brackets**:
<tagname>content goes here...</tagname>

1. HTML tags normally come **in pairs** like and
2. The first tag in a pair is the **start tag,** the second tag is the **end tag**
3. The end tag is written like the start tag, but with a **forward slash** inserted before the tag name

The start tag is also called the **opening tag**, and the end tag the **closing tag**.
Revise html from following link: https://www.w3schools.com/html/

5.2.2. What is CSS?

CSS stands for **C**ascading **S**tyle **S**heets

1. CSS describes how HTML elements are to be displayed on screen, paper, or in other media
2. CSS **saves a lot of work**. It can control the layout of multiple web pages all at once
3. External style sheets are stored in **CSS files.**

Syntax:
A CSS ruleset consists of a selector and a declaration block.

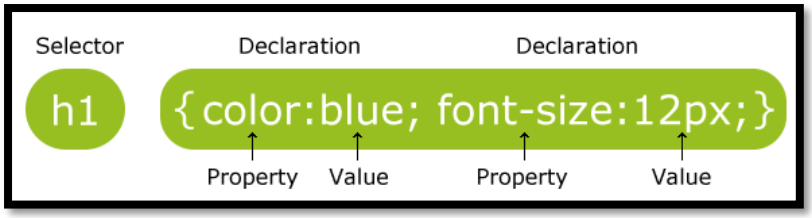


Figure 1: CSS Rule Syntax

Example

```
p {
   color: red;
   text-align: center;
}
```

5.2.3. What is JavaScript?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.
It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.
The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.
JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Syntax:
JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.
You can place the **<script>** tags, containing your JavaScript, anywhere within you web page.
The <script> tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
   JavaScript code
</script>
```

Your First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document. Write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
  <body>
    <script language="javascript" type="text/javascript">
      <!--
        document.write ("Hello World!")
      //-->
    </script>
  </body>
</html>
```

This code will produce the following result –

```
Hello World!
```

6. Homework before Lab

6.1. Homework 1

Install all the soft wares by getting help from Walkthrough task of the manual. As you have to do Task 2 on the same tools.

6.2. Homework 2

You have to make an HTML page having 3 Text Fields and Labels of ***First Name, Last Name*** and ***Password*** (mode of the password field should be Password) and a Submit Button. The Title of the html page must be "***My Web Page***"

Moreover, Study about three types of CSS styling Types and then implement following CSS on HTML using external style sheet.

1. Width of the Text fields should be 100px.
2. Height of the Text fields should be 40px.
3. Font should be Cambria and its size must be 14px.
4. Border color should be black.
5. Background of Submit button should be Grey and its Fore Color should be Black.

When the user clicks on the Submit button, validate using JavaScript. If any field is empty display a message "Fields should not be Empty" in Red Color.

7. Tools and Procedure.

7.1. Tools

1. JDK
2. IntelliJ Idea 2020.1. (For Practice Tasks of this Lab)
3. Xampp and Tomcat Server

7.2. Setting up JDK & Environment Variable.

Click **Next** as shown in Figure 2.

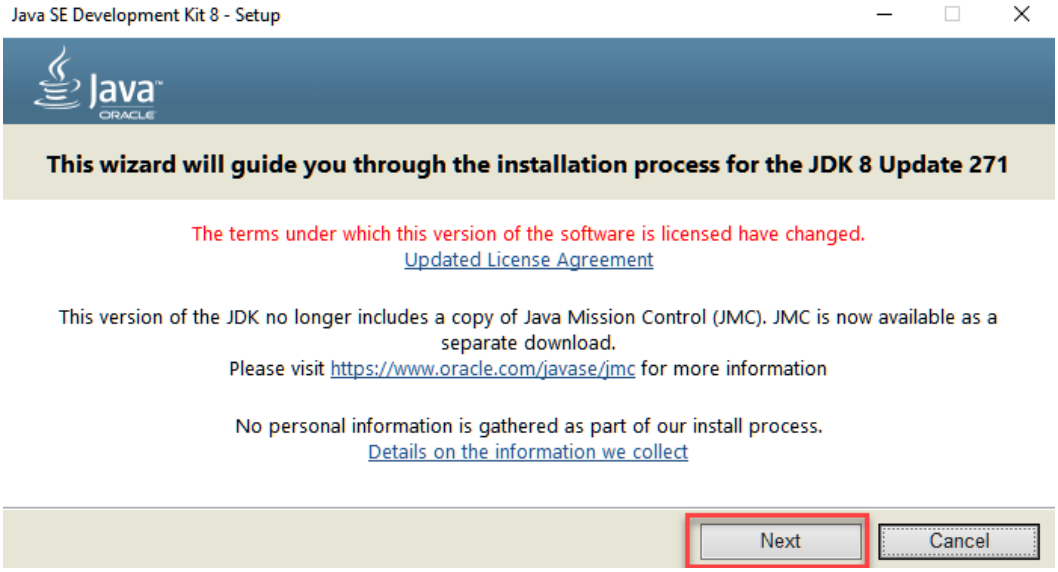


Figure 2: JDK installation wizard

Click **Next** as shown in Figure 3.

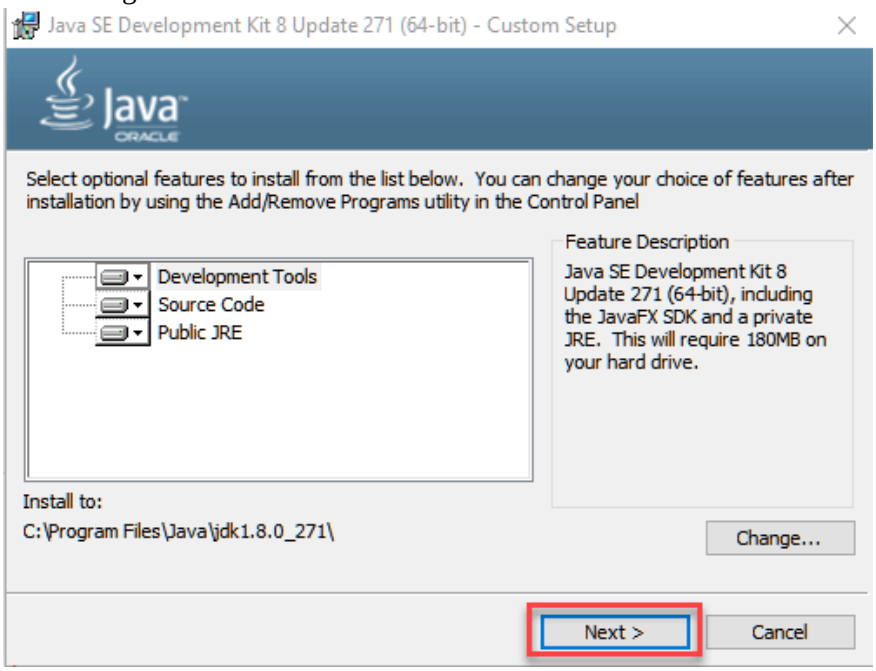


Figure 3: Installation Step

When jdk installed successfully then click **Close** to finish installation wizard.

Figure 4: Installation finished

Copy the path of bin folder which is installed in JDK folder.

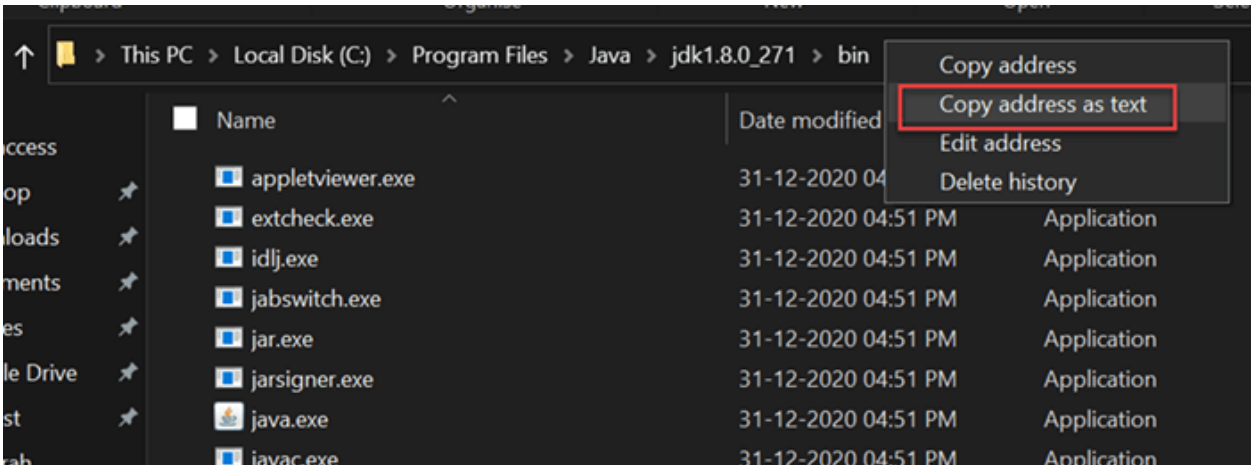


Figure 5: jdk bin path for environment variable

Next step is to set Environment Variables in Java. Press windows button, write env and select below mentioned option.

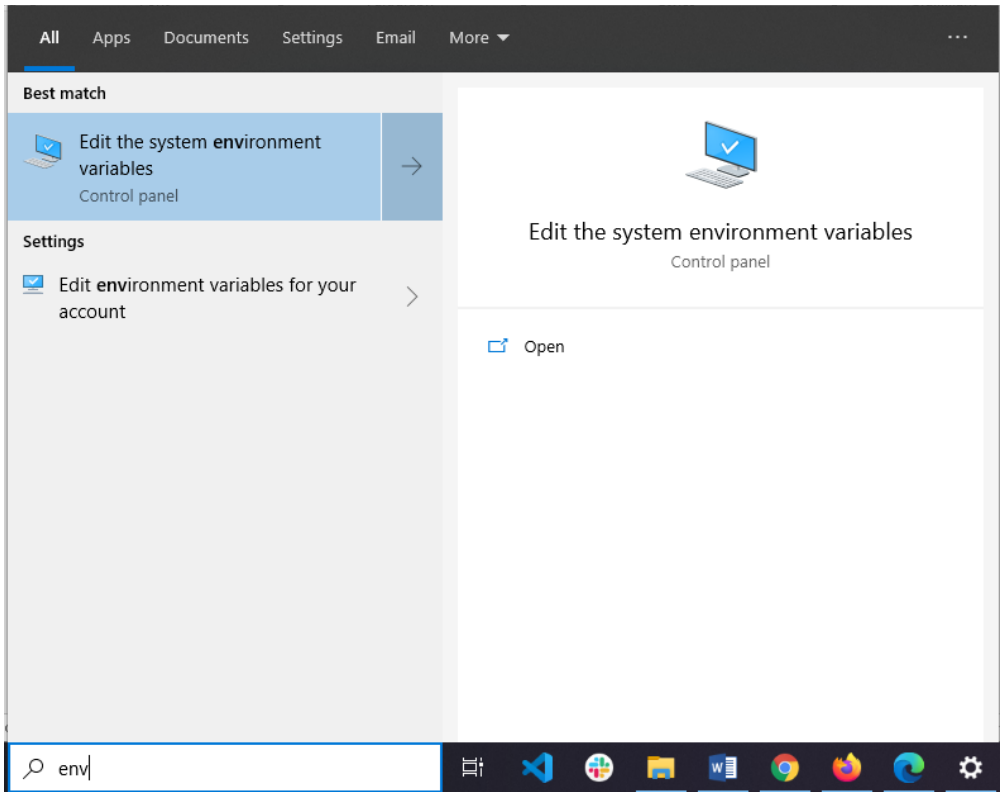


Figure 6: Open environment variable step

Click on Environment Variables.

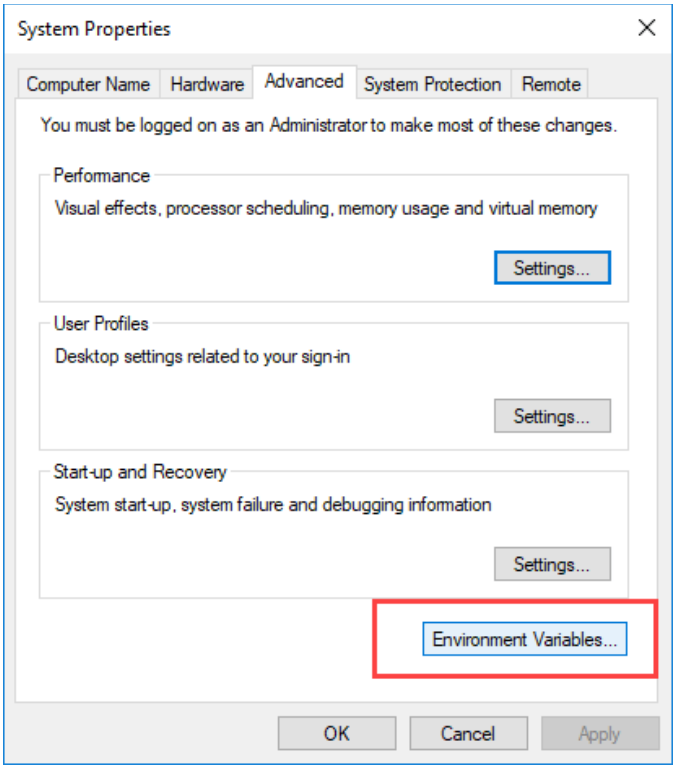


Figure 7: System Properties

Select Path and click on EDIT button.

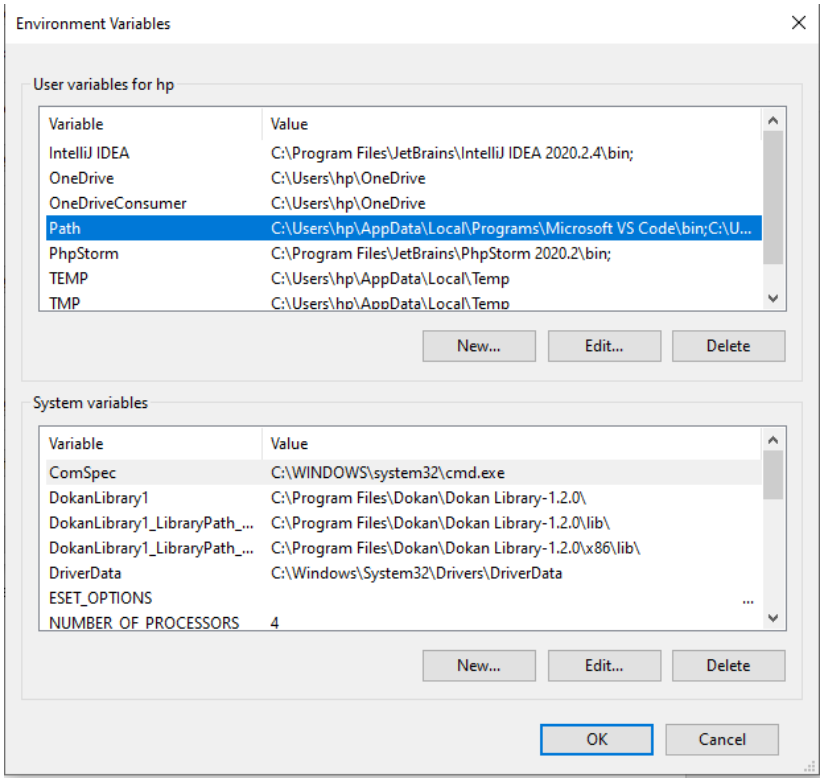


Figure 8: Setting Path Variable for jdk

Click on New button & paste folder address. Click on Ok

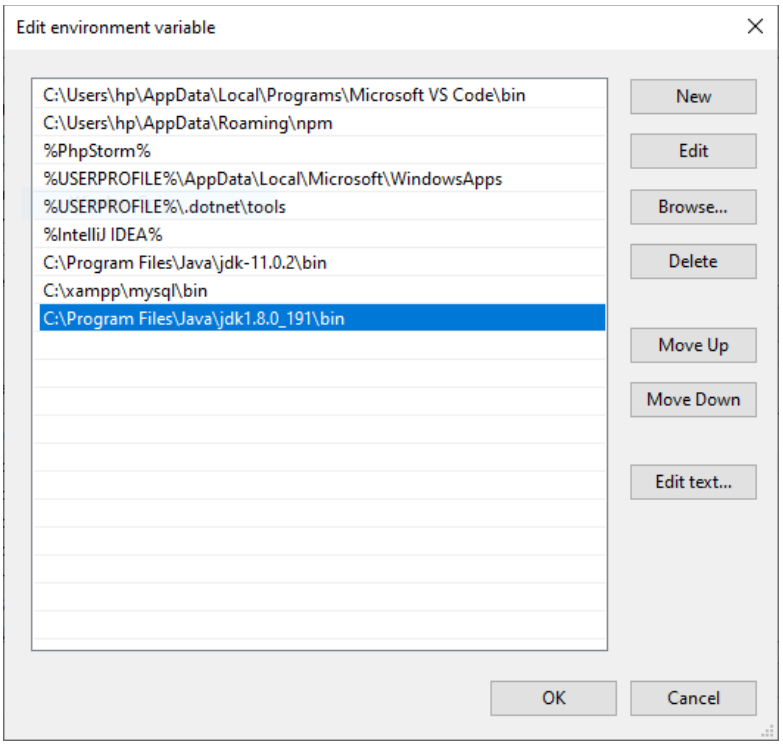


Figure 9: Add jdk bin path as new variable

To verify the successful installation of jdk, open cmd and write command “javac”. If you get something as shown in Figure then you installation is successful.

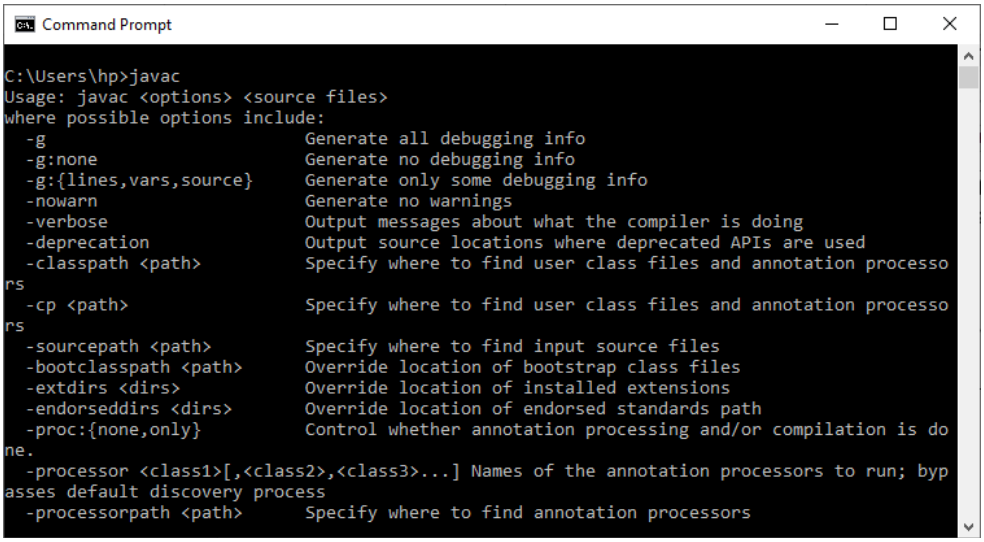
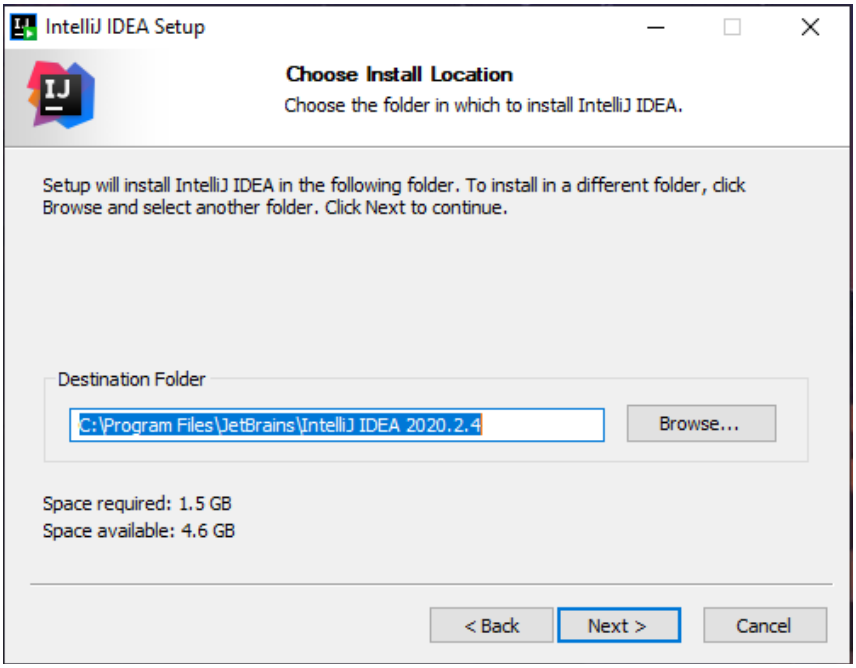


Figure 10: Jdk successful installation verification

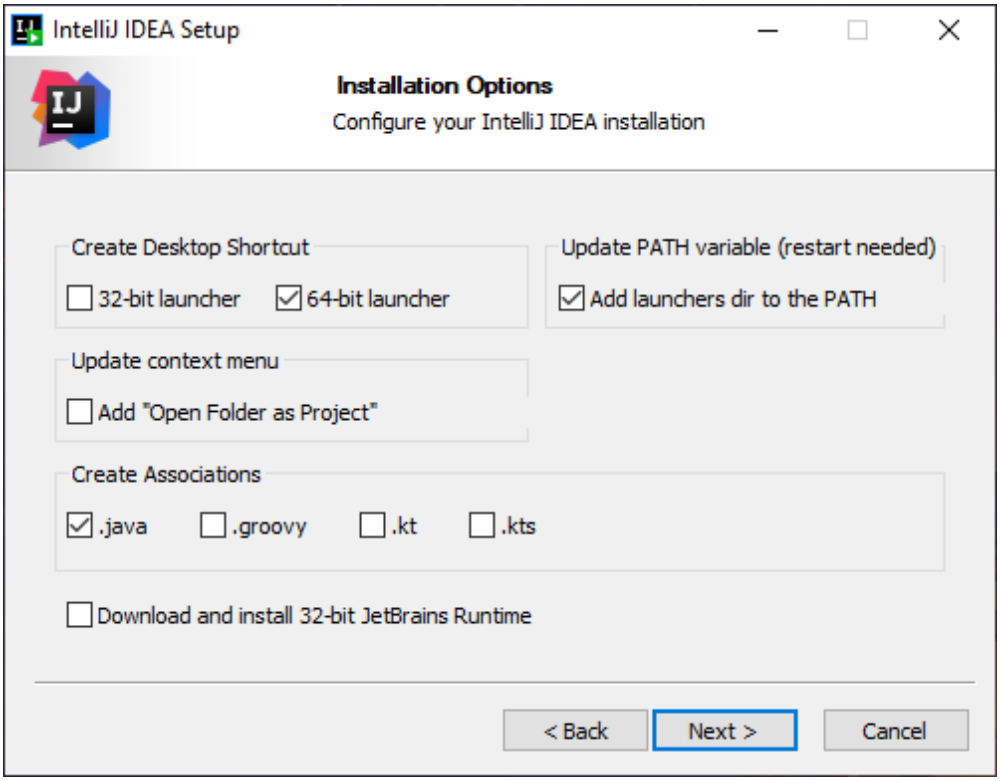
7.3. Setting up IntelliJ Idea and Tomcat Server.

Download IntelliJ Idea. Click on installation program and follow the prompts.

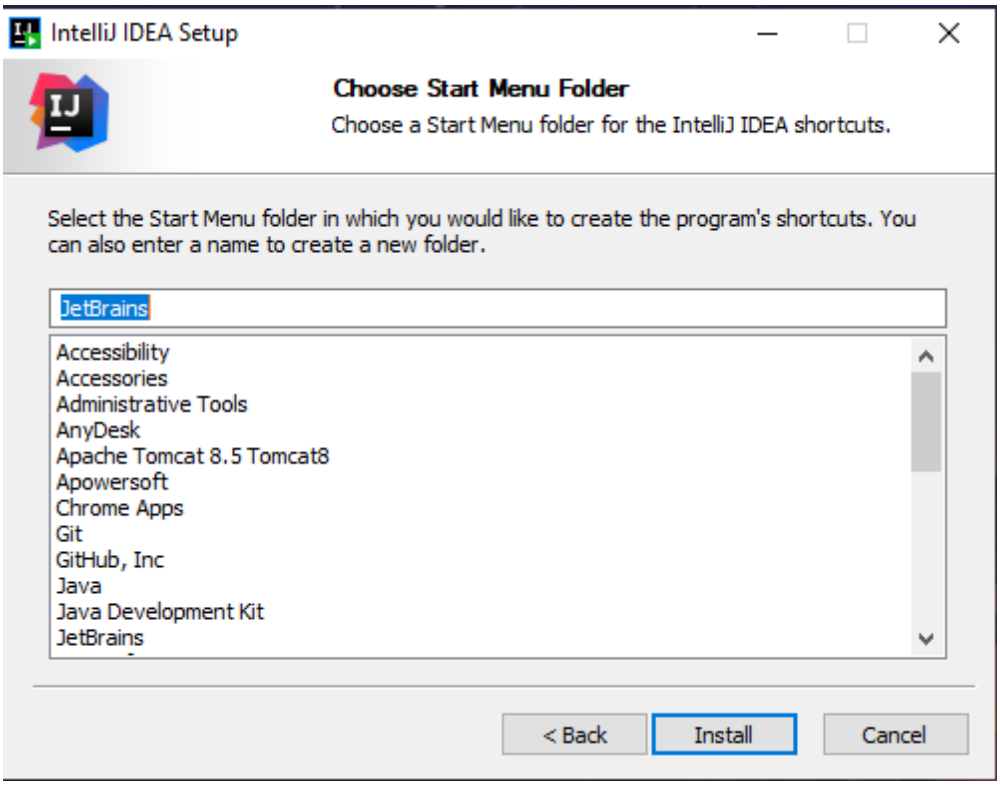
The **Choose Install Location** window will prompt you to either except the default installation directory or select the location where you would like to install IntelliJ. If the installation directory is ok, select the **Next** button to continue with the installation.



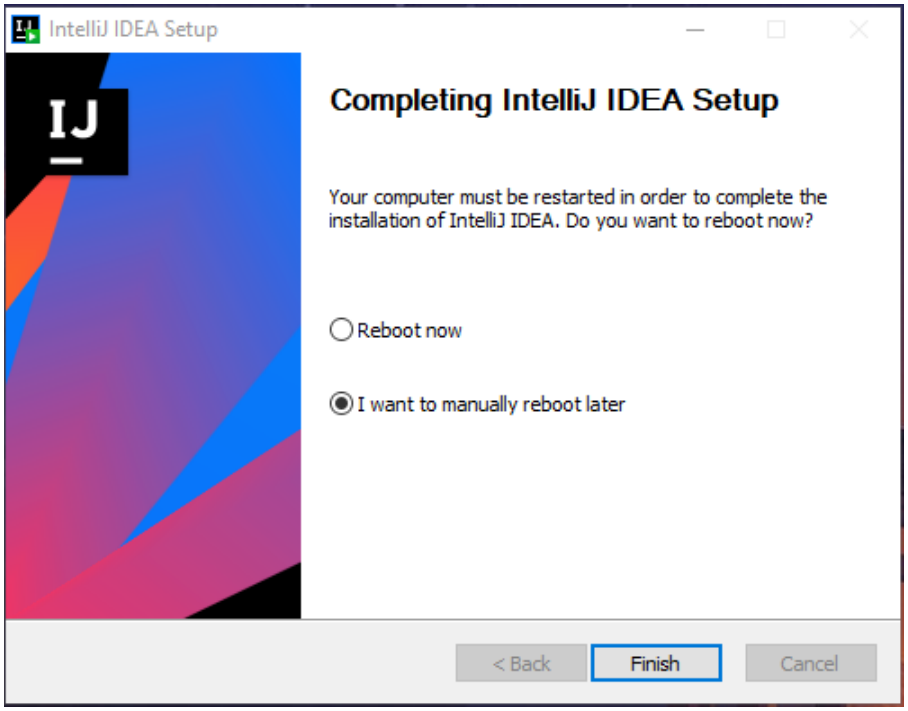
Check below mentioned check boxes and click **Next**



Click **Install** to start intelliJ installation

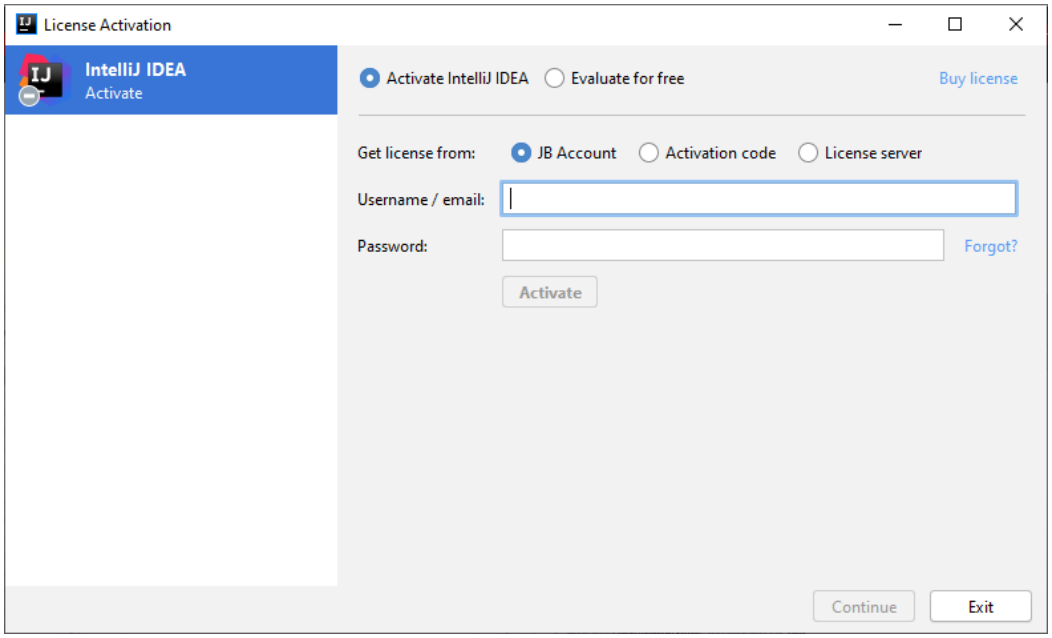


Choose reboot now to restart system for completion of setup.

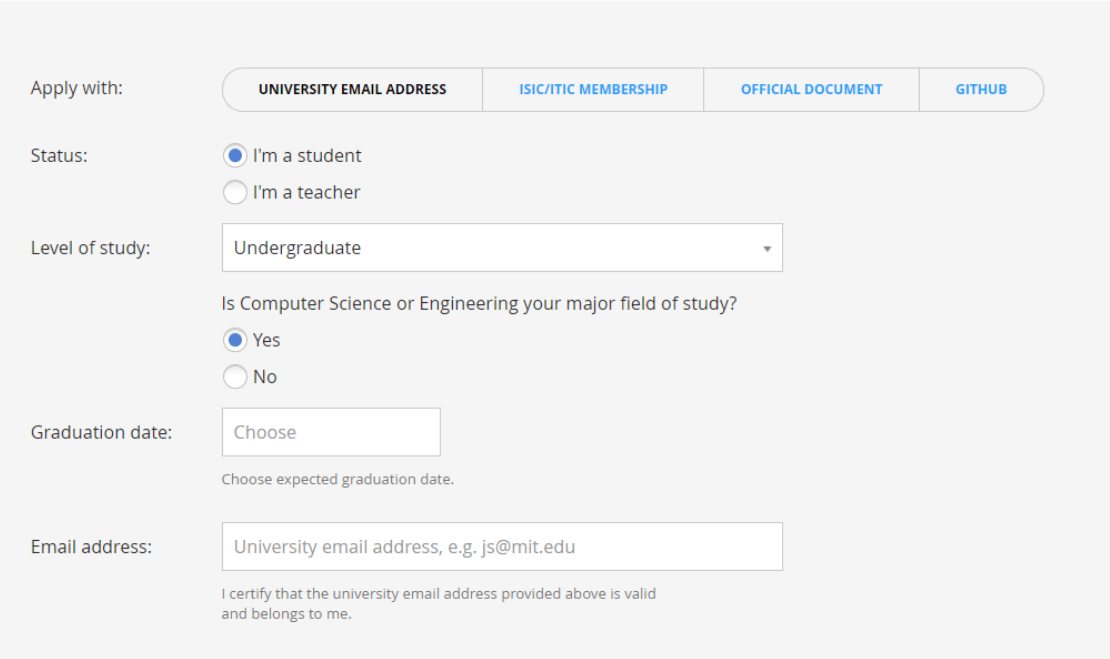


Click on the **Finish** button to complete the installation.

For students IntelliJ Idea offers free license. To get this free license & to activate IntelliJ Idea follow below mentioned instructions.

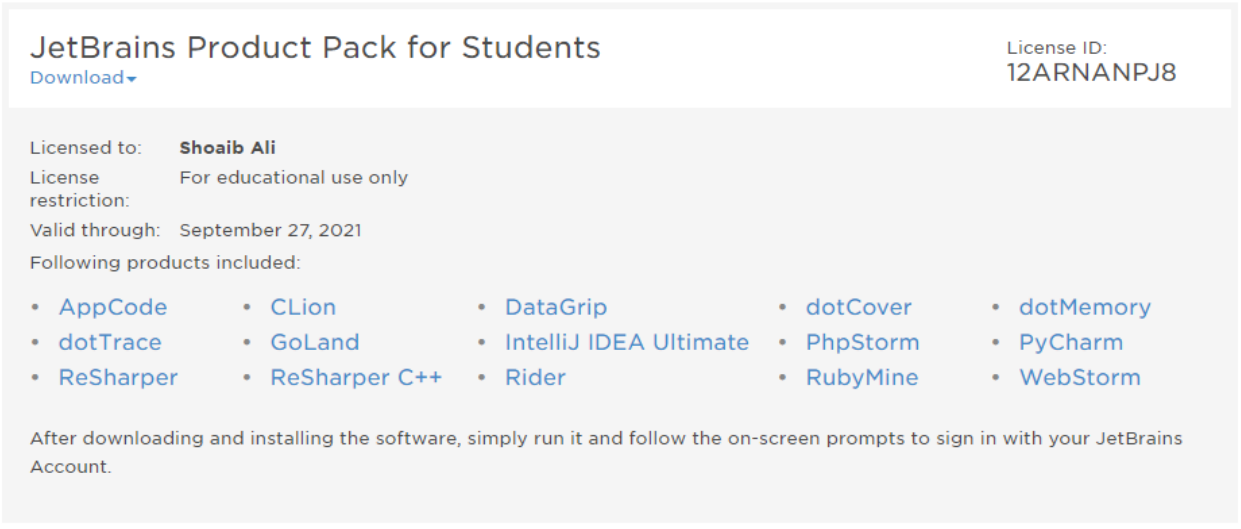


Visit following link and fill out form to get free access to jet Brains products for students.
https://www.jetbrains.com/shop/eform/students
Note: Use official student email address. For example: BCS203xxx@cust.pk



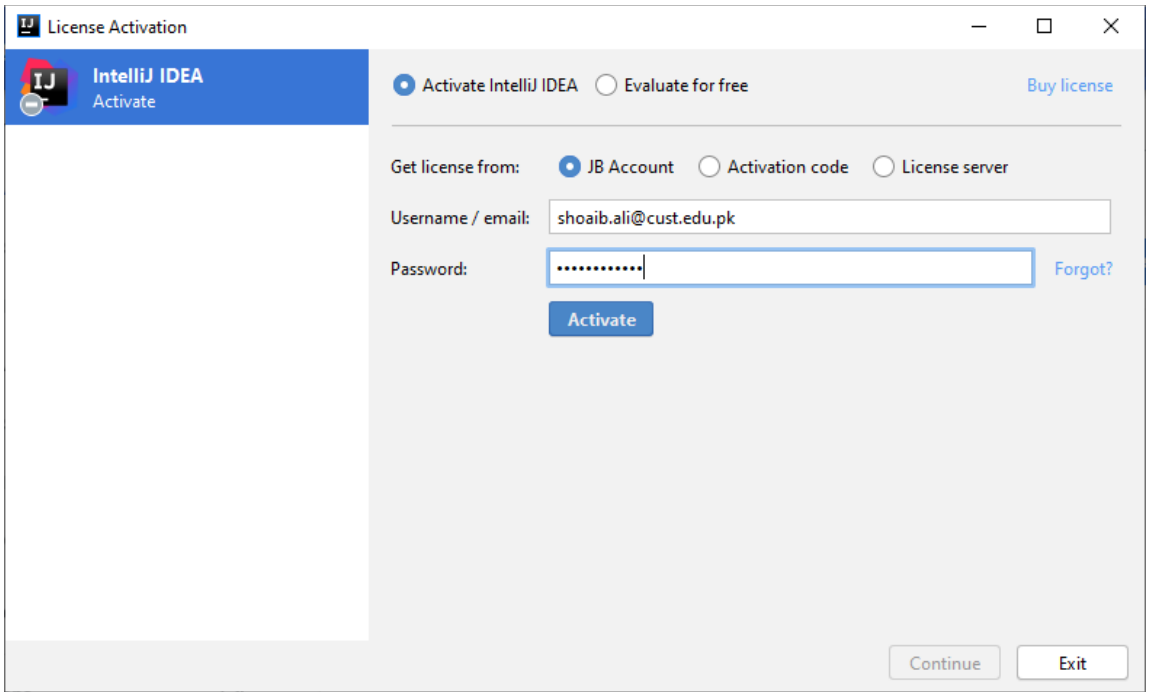
After successful submission you will get email in your official email inbox regarding free products. Follow those instructions to get access to free products. You will be asked to signup for JetBrains account. Follow this link to register your account (Use official email address given by university).
https://account.jetbrains.com/login

Your next step is to login to your account by following the same above-mentioned link.



Can't find your license here? Link your past purchases to your JetBrains Account by providing a license key or domain .

Now you are successfully registered just use this account username password to activate IntelliJ Idea license. Press Activate button to activate.

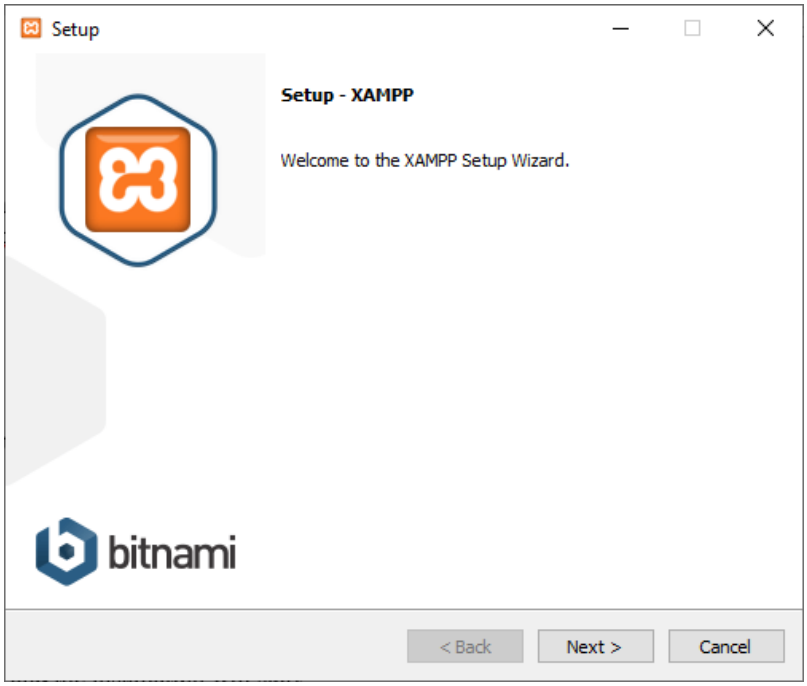


Now you are ready to create your first project in IDEA.

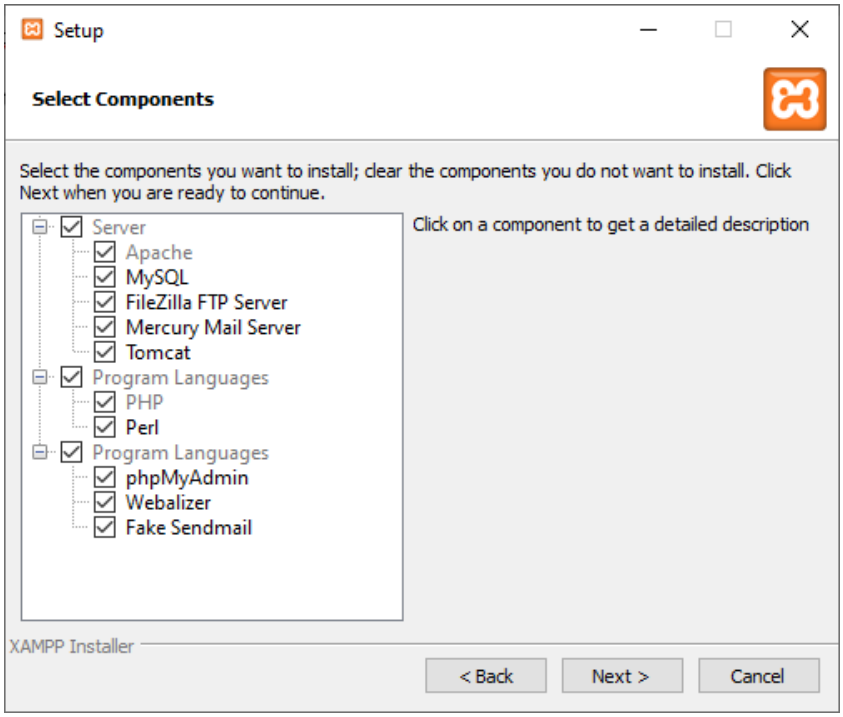
7.3. Installing Xampp

Open Xampp Installation File and you will be shown with the following window.

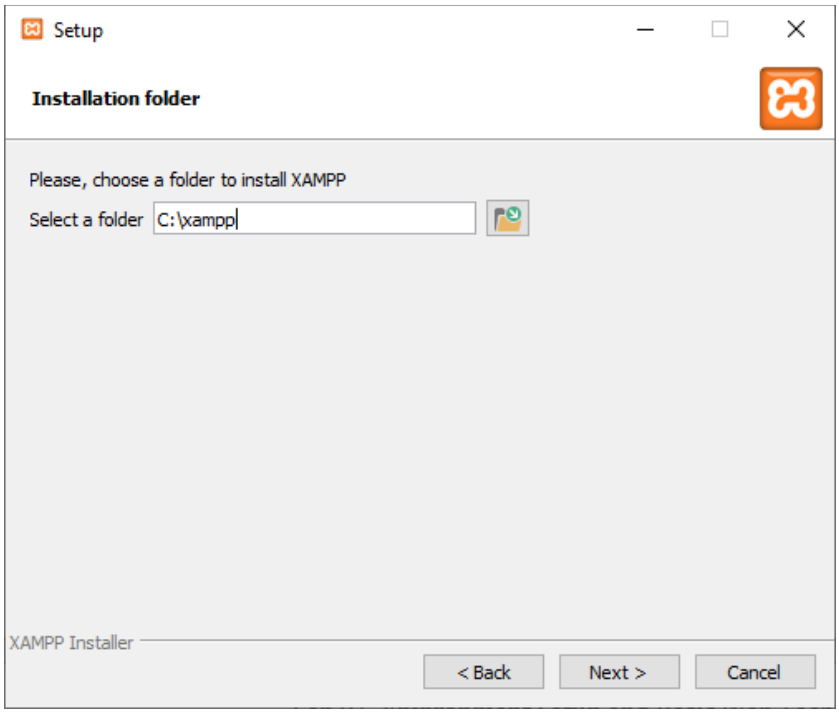
Click Next.



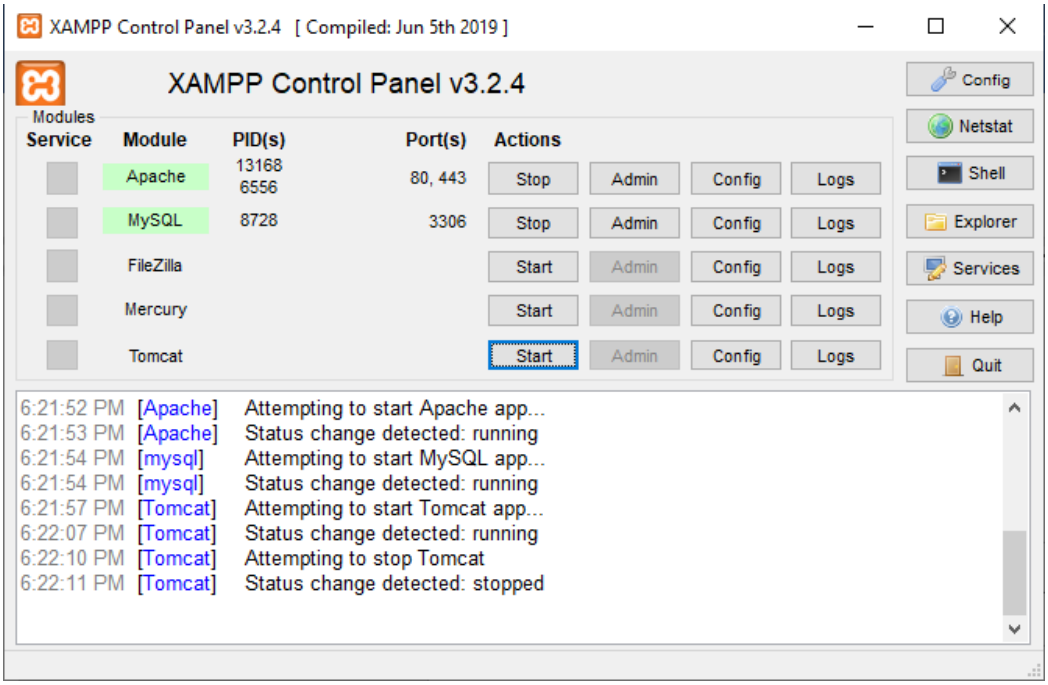
Confirm if all checkboxes are checked or not.



Click Next.



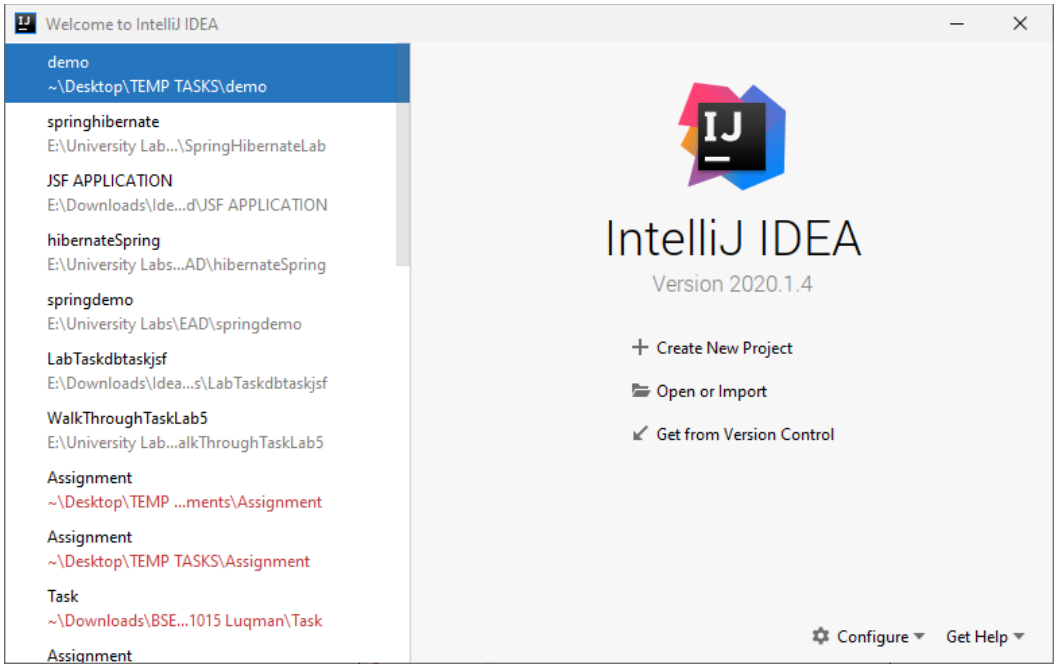
Follow the Next Next prompt to install xampp.
Start Apache & MySQL to confirm successful installation of xampp.



8. Walkthrough Tasks

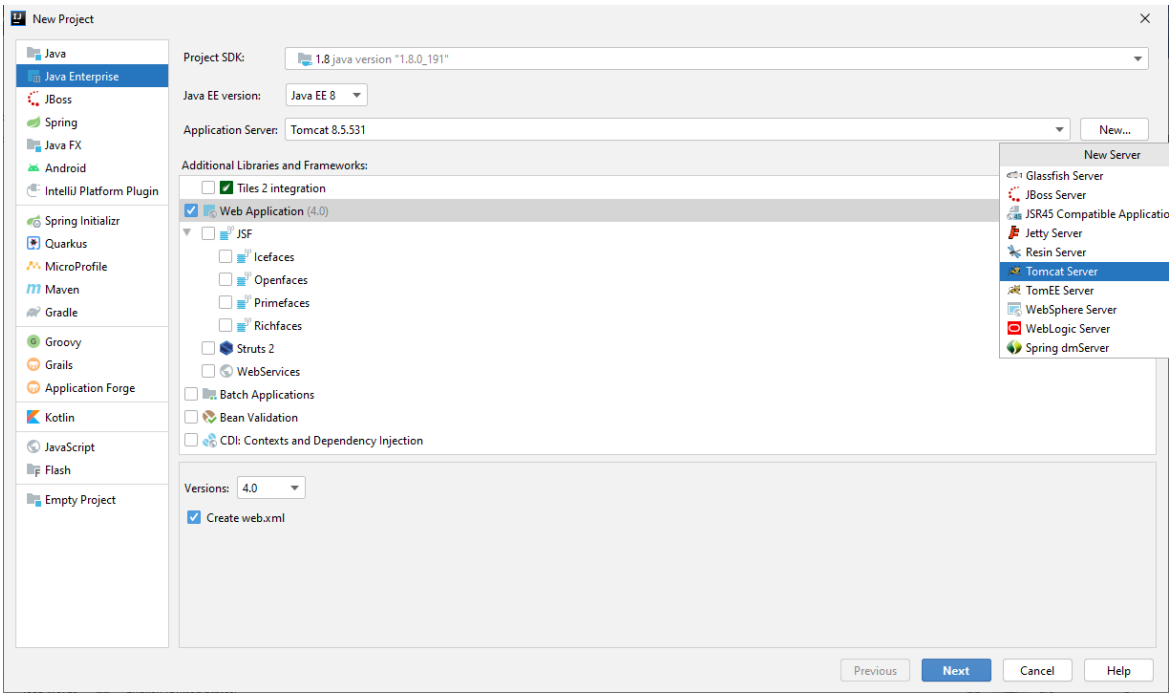
8.1. How to Create Project and Configure Tomcat in IDEA.

Download and install Tomcat server as a windows service.
To download and install Tomcat server click here https://www.liquidweb.com/kb/installing-tomcat-9-on-windows/
Open IntelliJ IDEA and Click Create New Project from the options given.

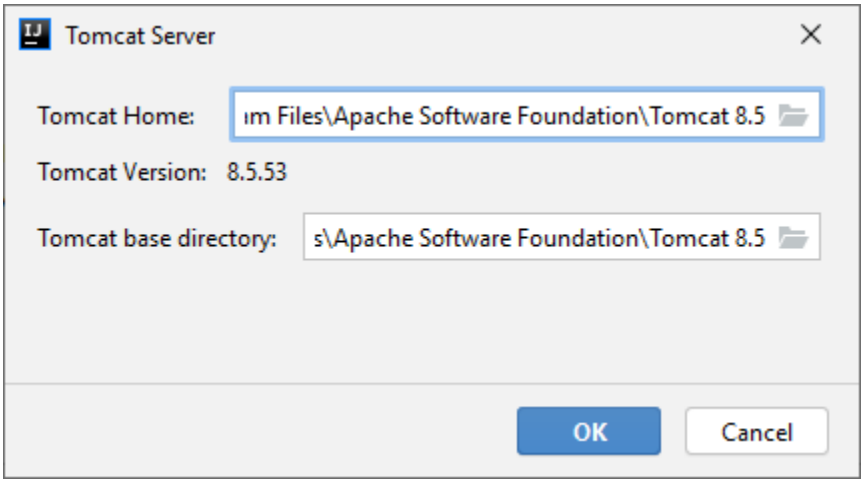


Select Java Enterprise and Check Web Application also select the installed JDK as your project SDK. In

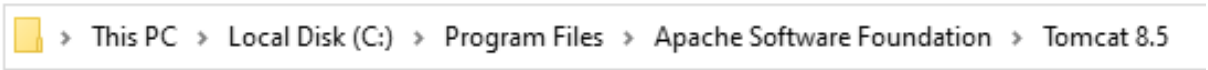
order to set the application servers, click new and select Tomcat Server.



After that you have to provide the path of the folder where your Tomcat Server Files are placed and then Click Ok.After setting up the Tomcat Server Click Ok.

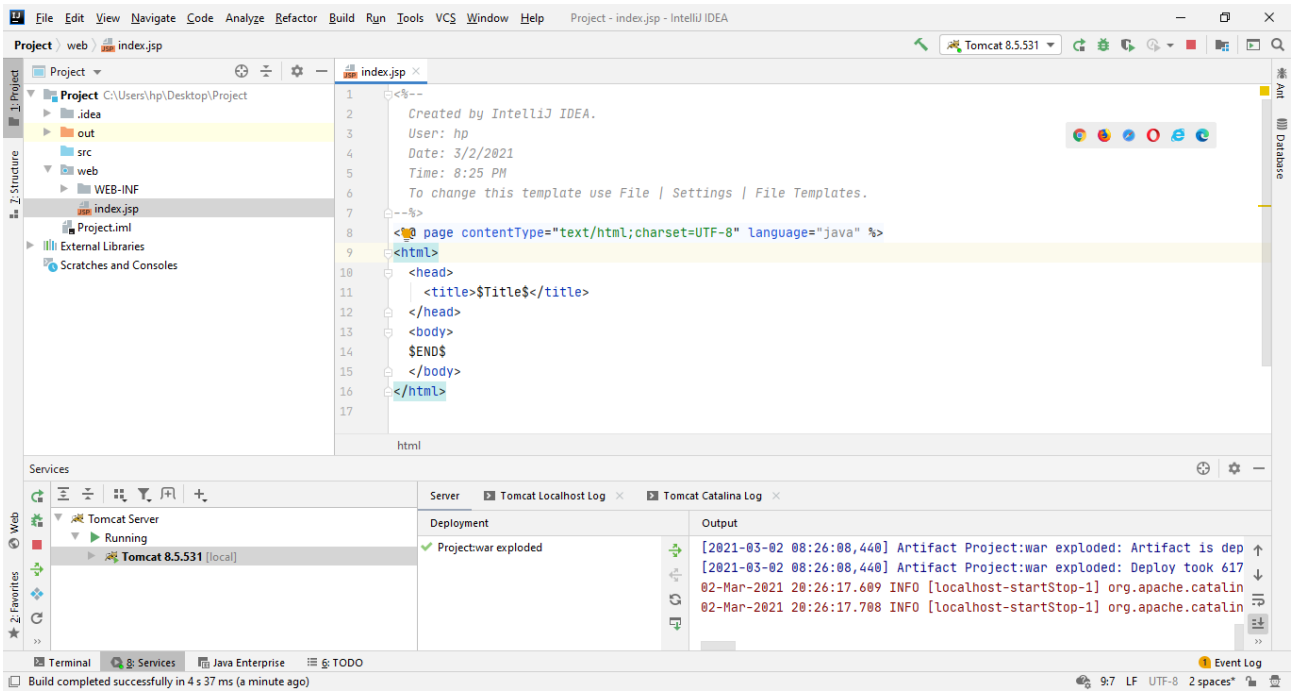


Example:



The path of the above location will be
C:\Program Files\Apache Software Foundation\Tomcat 8.5
Give Your Project the desired name, change project location if you want else leave it as it is and click Finish.

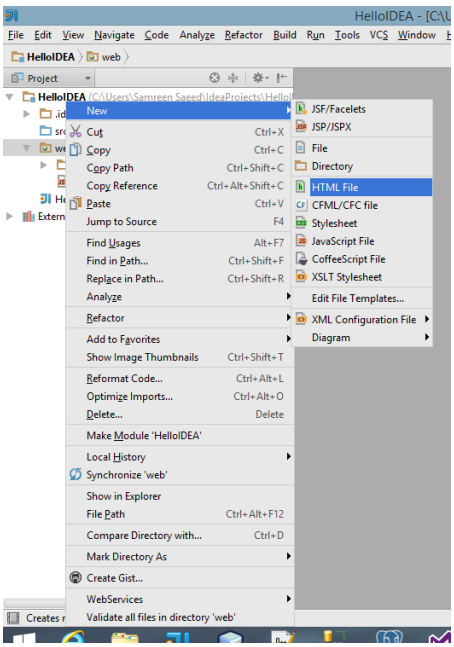
Now you are ready to write your first program.



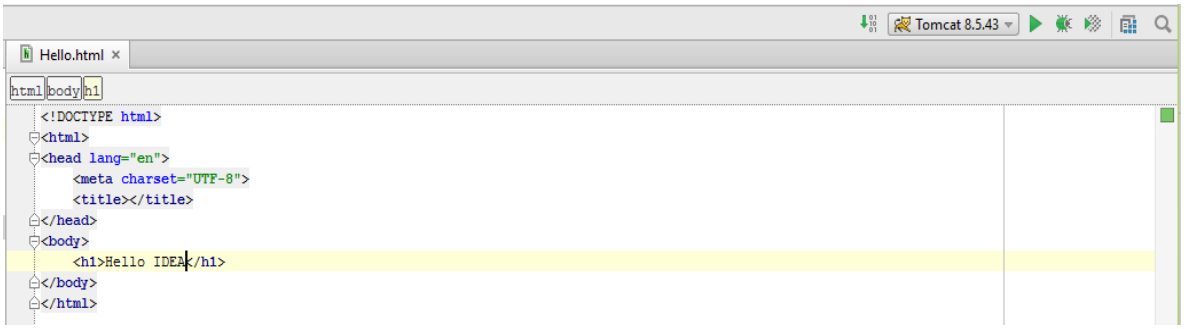
8.2. Displaying HelloIDEA in Browser.

Create Project in idea by taking help from following section

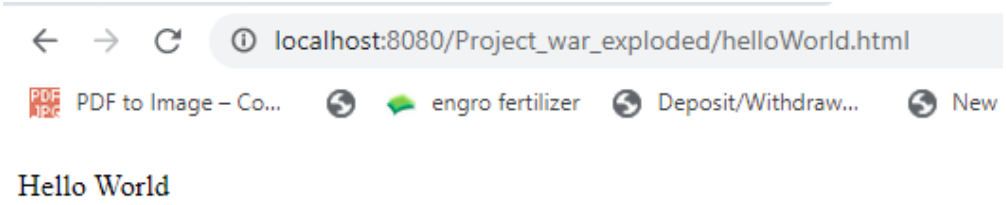
Create a new HTML file by Right Clicking Web than select **New>HTML File**



Type the Following code in HTML file and than choose browser from right corner of the window.



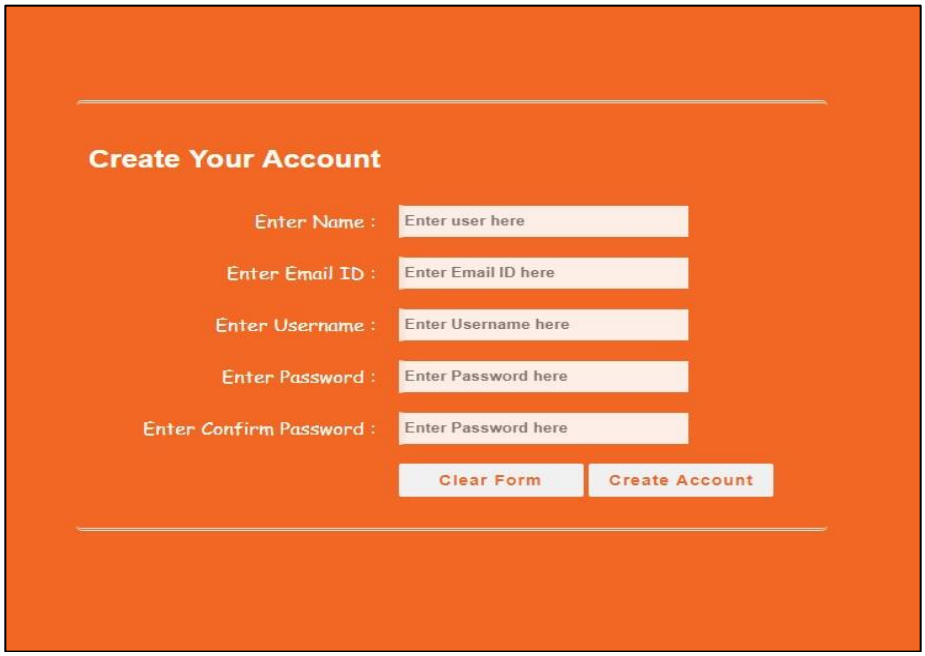
Output will be shown in the browser.



9. Practice Tasks

9.1. Practice Task 1

Create below form using HTML and CSS.



After that apply the following validations on the form

1. ***Email Validation*** using following regular Expression.

^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$

2. ***Password*** field should not be empty
3. ***Confirm Password*** field should contain same text as ***Password*** field.

When the user clicks ***Register Now*** button Validate all the fields and display relevant messages.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 02

Revision of Fundamental Features of Java

Lab 02: Revision of Fundamental Features of Java

1. Introduction

You must have studied Java before. In this lab you are going to revise some of the basic and fundamental concepts of java. These concepts will be helpful for you in this course.
Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as core component of Sun Microsystems' Java platform (**J2SE**).
The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example: **J2EE** for Enterprise Applications (which you are going to study in this course), **J2ME** for Mobile Applications.

2. Relevant Lecture Reading

https://www.w3schools.com/java/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	80 min	70 min
8.1	Walkthrough Tasks	40 min	50 min
9	Practice Tasks	As per time specified for each task	40 min
10	Evaluation Task (Unseen)	30 min	30 min

4. Objective of the Experiment

To understand the fundamental concepts of Java.

5. Concept Map

5.1. Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.
The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).
extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.
Following is an example demonstrating Java inheritance. In this example, you can observe two classes namely **Calculation** and **My_Calculation**.

```
class Calculation {
  int z;
```

```
  public void addition(int x, int y) {
     z = x + y;
     System.out.println("The sum of the given numbers:"+z);
  }

  public void Subtraction(int x, int y) {
     z = x - y;
     System.out.println("The difference between the given numbers:"+z);
  }
}

public class My_Calculation extends Calculation {
  public void multiplication(int x, int y) {
     z = x * y;
     System.out.println("The product of the given numbers:"+z);
  }

  public static void main(String args[]) {
     int a = 20, b = 10;
     My_Calculation demo = new My_Calculation();
     demo.addition(a, b);
     demo.Subtraction(a, b);
     demo.multiplication(a, b);
  }
}
```

5.2. Classes and Interfaces

Class

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

Following is a sample of a class.

```
public class Dog {
  String breed;
  int ageC
  String color;

  void barking() {
  }

  void hungry() {
  }

  void sleeping() {
  }

 void running(){
  }
}
```

5.3. Abstract class/Abstract Methods/Interfaces

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared but contains no implementation. Abstract classes may not be instantiated and require subclasses to provide implementations for the abstract methods. Let's look at an example of an abstract class, and an abstract method.

```
public abstract Animal
{
  public void eat(Food food){System.out.println("Eat!");}

  public void sleep(int hours){System.out.println("Sleep!");}

  public abstract void makeNoise();
}
```

Note that the abstract keyword is used to denote both an abstract method, and an abstract class. Now, any animal that wants to be instantiated (like a dog or cow) must implement the makeNoise method - otherwise it is impossible to create an instance of that class. Let's look at a Dog and Cow subclass that extends the Animal class.

```
public Dog extends Animal
{
  public void makeNoise() { System.out.println ("Bark!"); }
}

public Cow extends Animal
{
  public void makeNoise() { System.out.println ("Moo!"); }
}
```

Now you may be wondering why not declare an abstract class as an interface, and have the Dog and Cow implement the interface. Sure, you could - but you'd also need to implement the eat and sleep methods. By using abstract classes, you can inherit the implementation of other (non-abstract) methods. You can't do that with interfaces - an interface cannot provide any method implementations. It contains all the abstract methods.

5.4. Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.
Let us look at an example.

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
```

Now, the Deer class is considered to be polymorphic since this has multiple inheritance.
Following are true for the above examples −

A Deer IS-A Animal
A Deer IS-A Vegetarian
A Deer IS-A Deer
A Deer IS-A Object
When we apply the reference variable facts to a Deer object reference, the following declarations are legal

```
Deer d = new Deer();
Animal a = d;
Vegetarian v = d;
Object o = d;
```

All the reference variables d, a, v, o refers to the same Deer object in the heap.

5.5. Association, Composition, and Aggregation

5.5.1. Association

Association establishes relationship between two **classes** through their **objects**. The relationship can be one to one, one to many, many to one and many to many.
Let's understand with an example.

```
class CarClass{
  String carName;
  double carSpeed;
  int carId;
  CarClass(String name, double speed, int Id)
  {
        this.carName=name;
        this.carSpeed=speed;
        this.carId=Id;
  }
}
class Driver{
  String driverName;
  int driverAge;
  Driver(String name, int age){
    this.driverName=name;
    this.driverAge=age;
  }
}
class TransportCompany{
  public static void main(String args[])
  {
        CarClass obj= new CarClass("Ford", 180.15, 9988);
        Driver obj2 = new Driver("Andy", 45);
        System.out.println(obj2.driverName+" is a driver of car Id: "+obj.carId);
  }
}
```

In the above example, there is a one-to-one relationship (**Association**) between two classes: Car and Driver. Both the classes represent two separate entities.

5.5.2. Aggregation and Association

Aggregation is a special form of association where the child can exist independently of the parent. Example: Class (parent) and Student (child). Delete the Class and the Students still exist.

```
class Address
{
  int streetNum;
  String city;
  String state;
  String country;
  Address(int street, String c, String st, String coun)
  {
     this.streetNum=street;
     this.city =c;
     this.state = st;
     this.country = coun;
  }
}
class StudentClass
{
  int rollNum;
  String studentName;
  Address studentAddr;
  StudentClass(int roll, String name, Address addr){
     this.rollNum=roll;
     this.studentName=name;
     this.studentAddr = addr;
  }
  public static void main(String args[]){
     Address ad = new Address(55, "Agra", "UP", "India");
     StudentClass obj = new StudentClass(123, "Chaitanya", ad);
     System.out.println(obj.rollNum);
     System.out.println(obj.studentName);
     System.out.println(obj.studentAddr.streetNum);
     System.out.println(obj.studentAddr.city);
     System.out.println(obj.studentAddr.state);
     System.out.println(obj.studentAddr.country);
  }
}
```

The above example shows the **Aggregation** between Student and Address classes. You can see that in Student class has used Address class to obtain student address. It's typical example of Aggregation in Java.

5.5.3. Composition

It is a restricted form of Aggregation where the child cannot exist independent of the parent. Example: House (parent) and Room (child). Rooms don't exist separate to a House.

Imagine that a Student class is created. The class holds information about individual students at a school. One piece of information stored is the student's date of birth. It's held in a GregorianCalendar object:

```
import java.util.GregorianCalendar;
public class Student
{
```

```
private String name;
private GregorianCalendar dateOfBirth;
public Student(String name, int day, int month, int year)
{
this.name = name;
this.dateOfBirth = new GregorianCalendar(year, month, day);
}
 //rest of Student class..
}
```

As the *student* class is responsible for the creation of the *GregorianCalendar* object, it will also be responsible for its destruction (i.e., once the Student object no longer exists neither will the *GregorianCalendar* object). Therefore, the relationship between the two classes is composition because Student *has-a GregorianCalendar* and it also controls its lifetime. The *GreogrianCalender* object cannot exist without the Student object.

5.6. Exception Handling

The **exception handling in java** is one of the powerful *mechanisms to handle the runtime errors* so that normal flow of the application can be maintained.
http://www.tutorialspoint.com/java/java_exceptions.htm
Go to the above URL to have a detailed revision about exception handling in Java.

5.7. JDBC

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases. The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

1. Making a connection to a database.
2. Creating SQL or MySQL statements.
3. Executing SQL or MySQL queries in the database.
4. Viewing & modifying the resulting records.

https://www.tutorialspoint.com/jdbc/jdbc-introduction.htm
Follow the above link to have a detailed reading and revision about JDBC.

6. Homework before Lab

It is assumed that the user is both familiar and comfortable with the following prior to solving practice tasks.

1. Class and Interface
2. Inheritance
3. Polymorphism
4. Association, Composition, and Aggregation
5. Exception Handling
6. JDBC

You should be familiar with above terms before you can start J2EE. This lab is designed specially to revise all the core and important concepts of java that are going to be used in J2EE.

7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. Xampp

8. Walkthrough Tasks

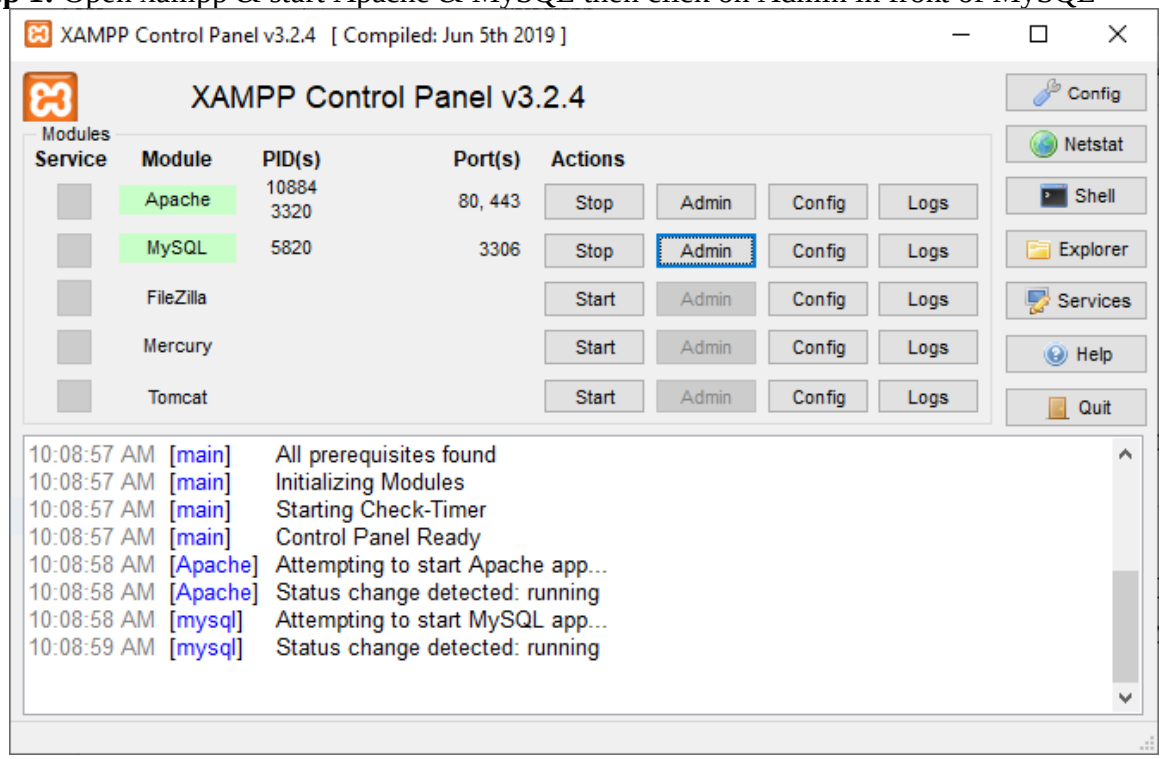
This section is designed such a way that you can complete the following tasks independently. However if there is any ambiguity you can refer it to the lab instructor.
Just revise the Concept map section of the manual so that you can perform the practice tasks easily.

8.1. Create Database using phpMyAdmin & show data using JDBC

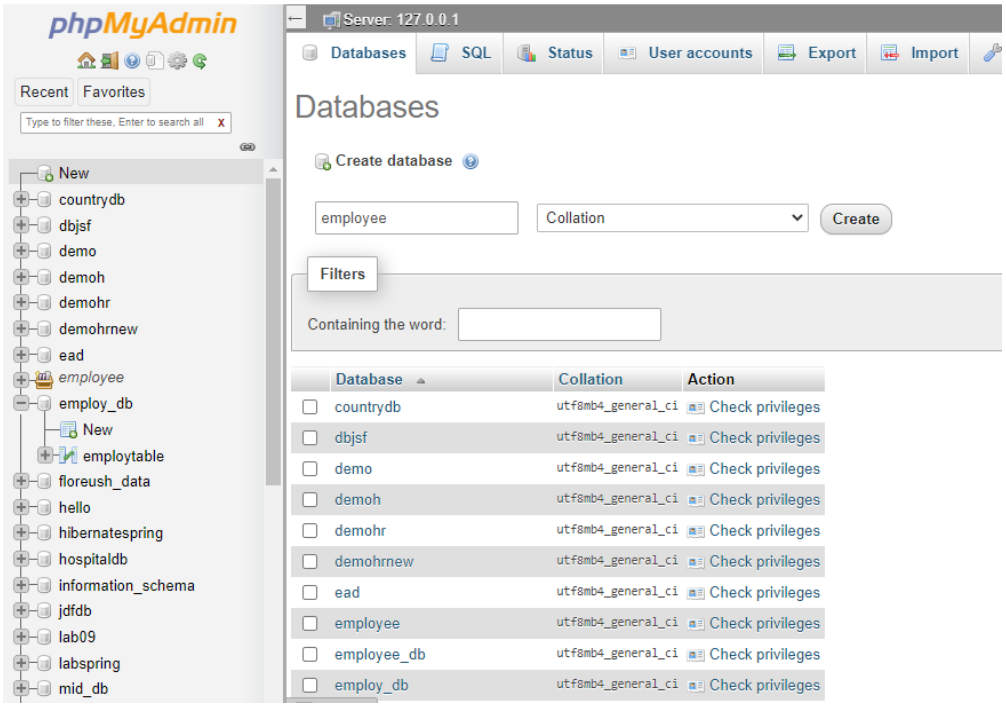
[Expected time = 40 mins]

Create database & insert dummy record:

Step 1: Open xampp & start Apache & MySQL then click on Admin in front of MySQL

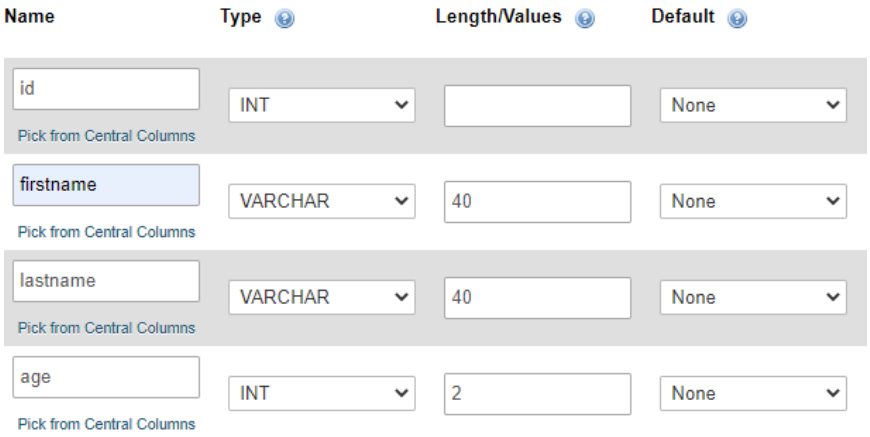


Step 2: Click on new button to create new database. Give database a name of your choice, for this task we are creating db with name employee. Select collation from dropdown and click on Create button to create database.

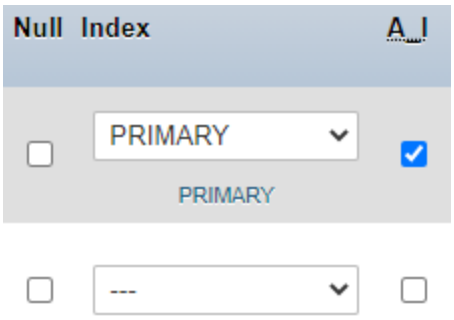


Step 3: Click on database to create new table with name employee_info with 4 columns.

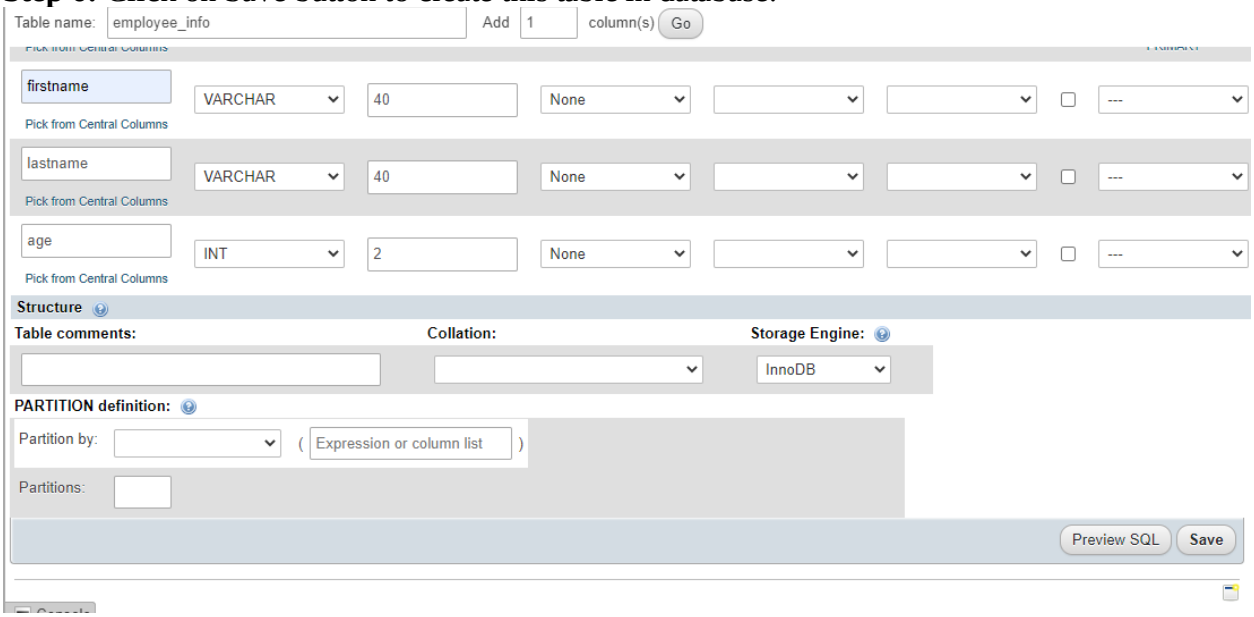
Step 4: Give name, type and length to your columns. Our next step is to make **id** column as primary key to uniquely identify each record.



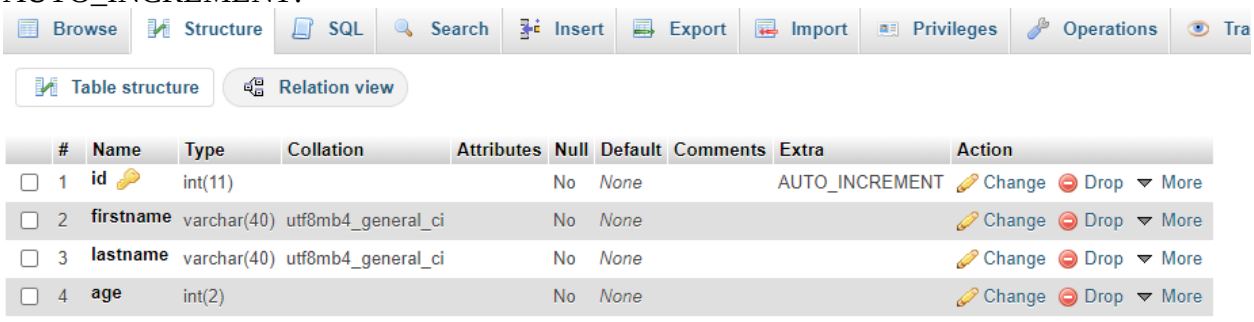
Step 5: Click on A_I (Auto Increment) checkbox to make id column as primary key in this table.



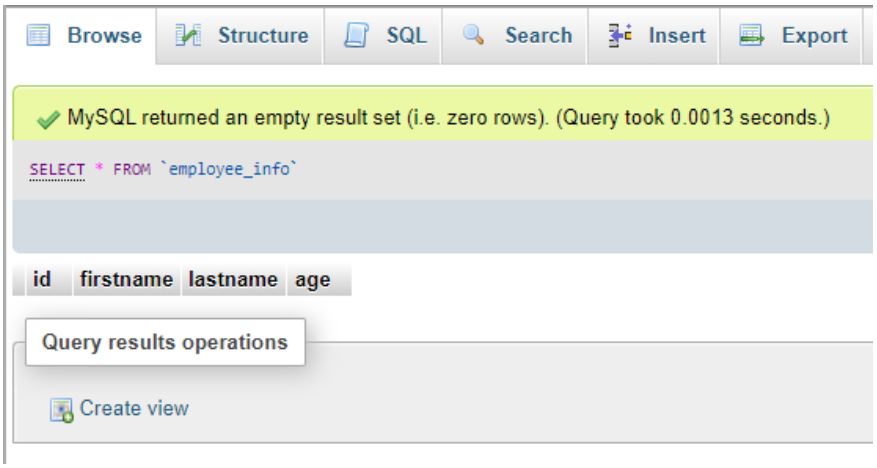
Step 6: Click on Save button to create this table in database.



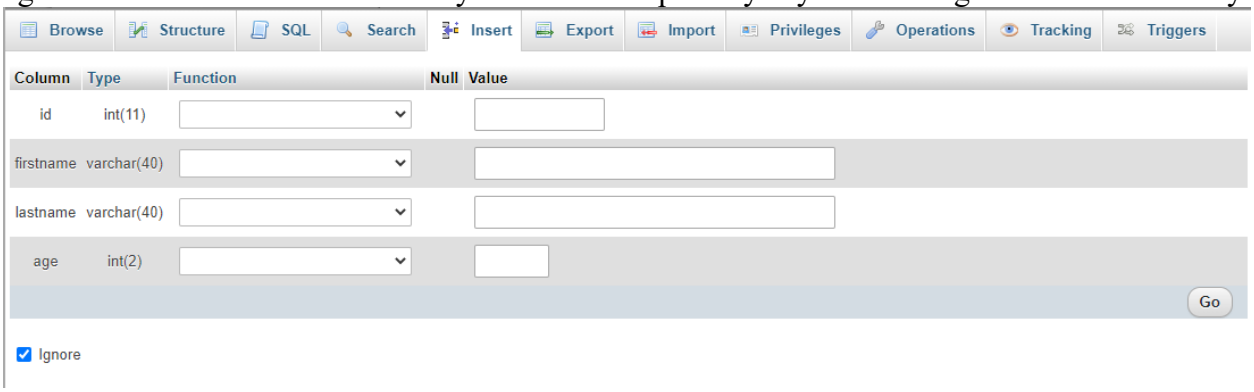
Step 7: In Structure tab you can see structure of your table. From this view you can see that id column is primary key (Sign: Golden key) and primary key generation method is AUTO_INCREMENT.



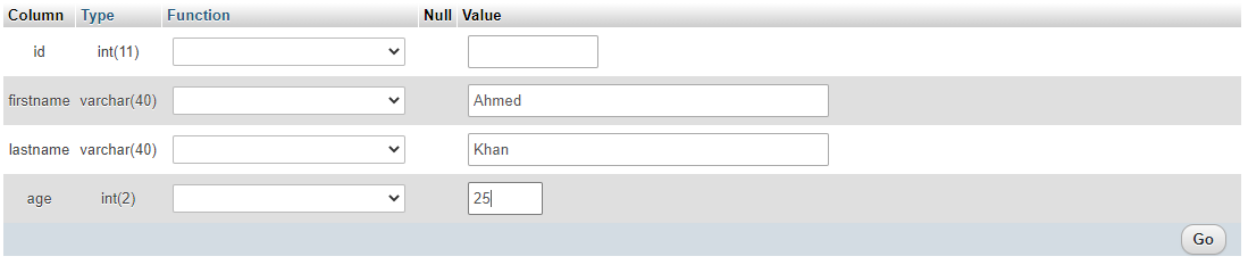
Click on browse tab to see all data available in database.



Step 8: For showing dummy data for our application, we will insert one record in database using phpMyAdmin. Follow below mentioned steps:
Click on Insert tab from top menu then following screen will appear. Fill firstname, lastname & age fields. Don't fill id column. Why? Because it is primary key and it will generate automatically.

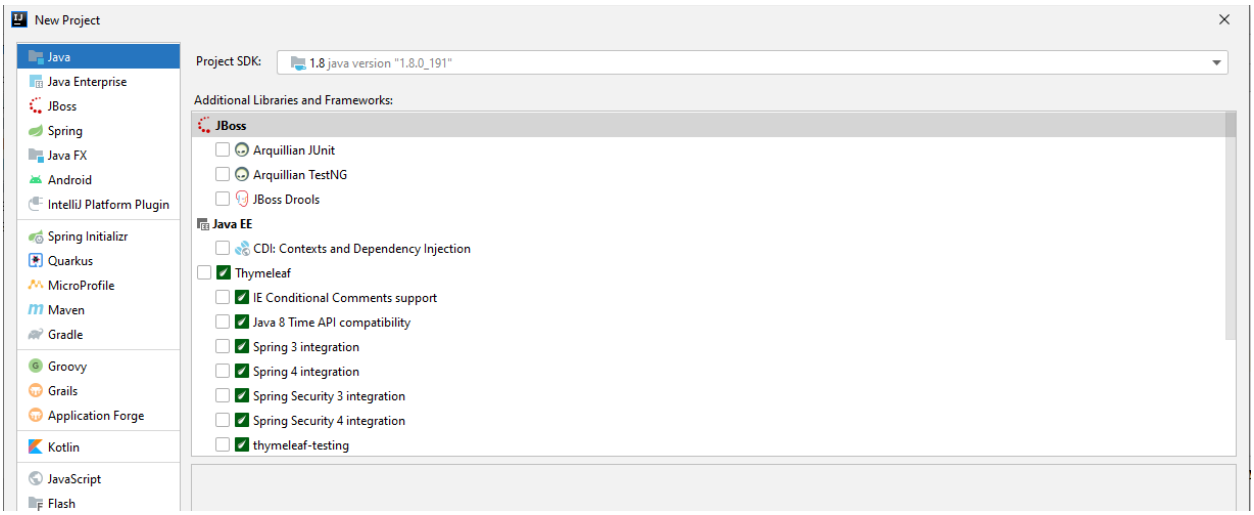


After inserting data click on Go button to save data in database.

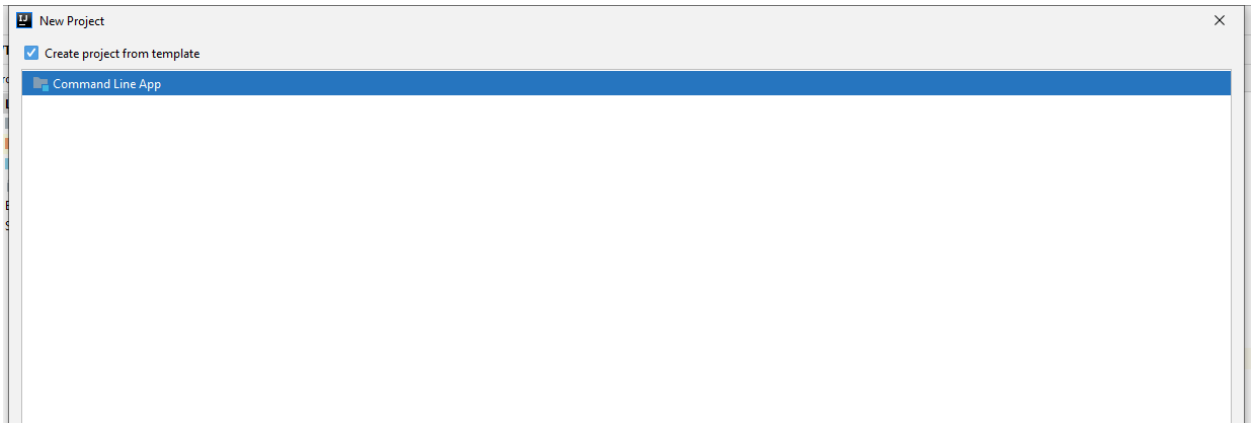


Create Java project in IntelliJ Idea & Show data from database on IntelliJ CLI.

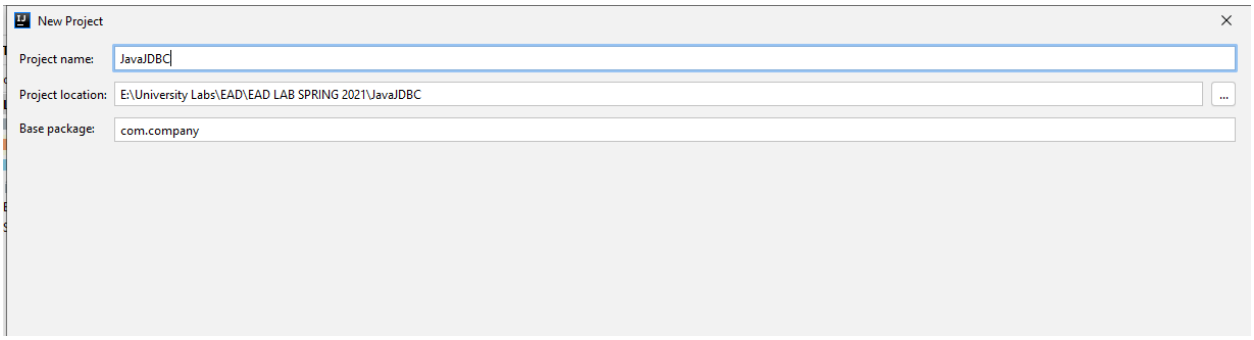
Step 1: Create new java project in IntelliJ Idea. Select Project SDK (In our case it is 1.8 version).



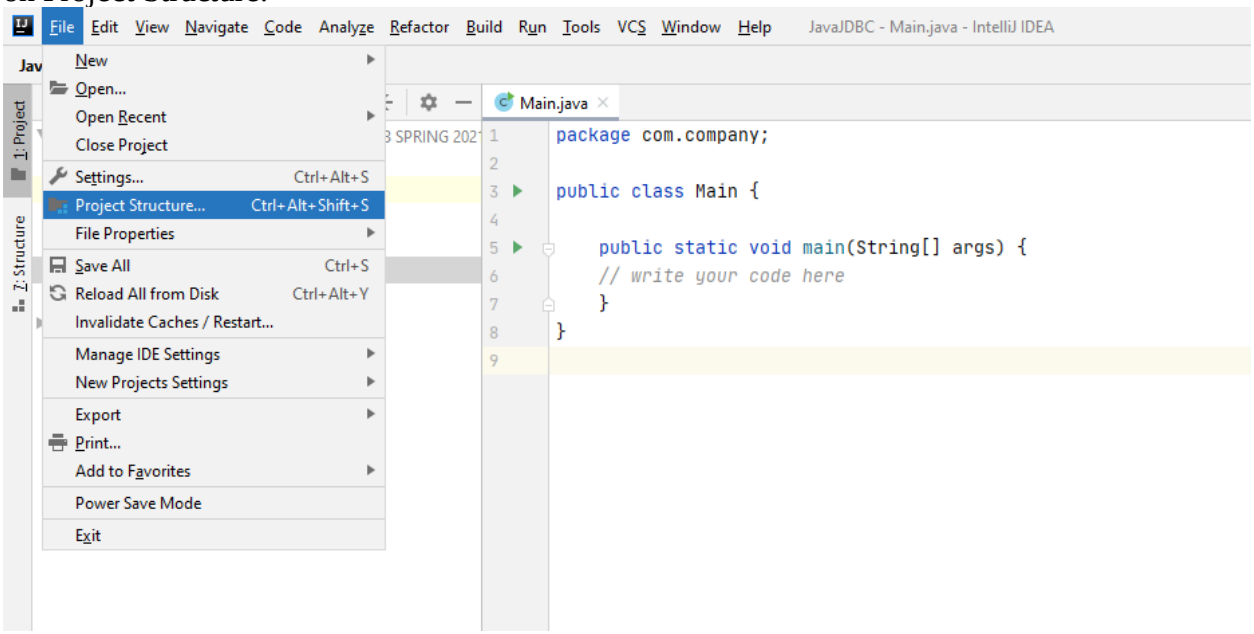
Step 2: Check Create project from template checkbox & click NEXT button.



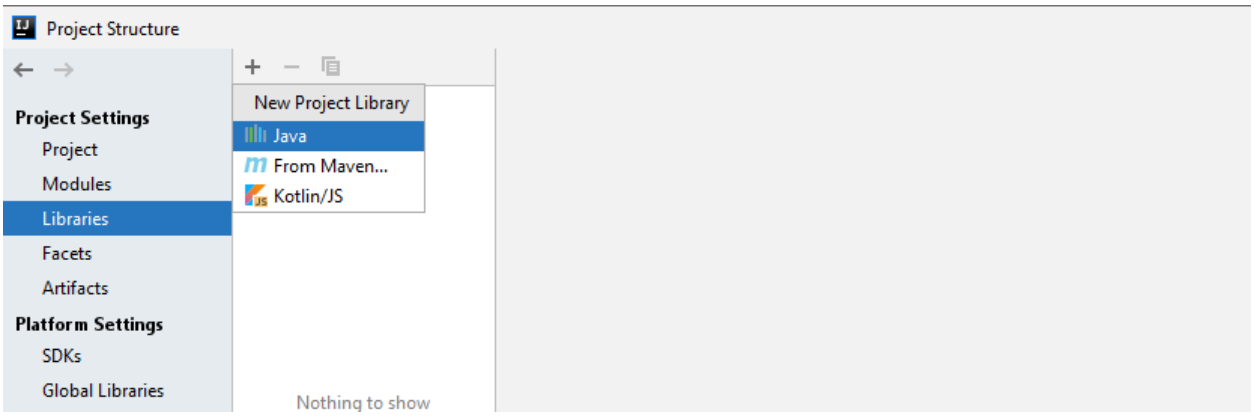
Step 2: Give name to your project and select Finish.

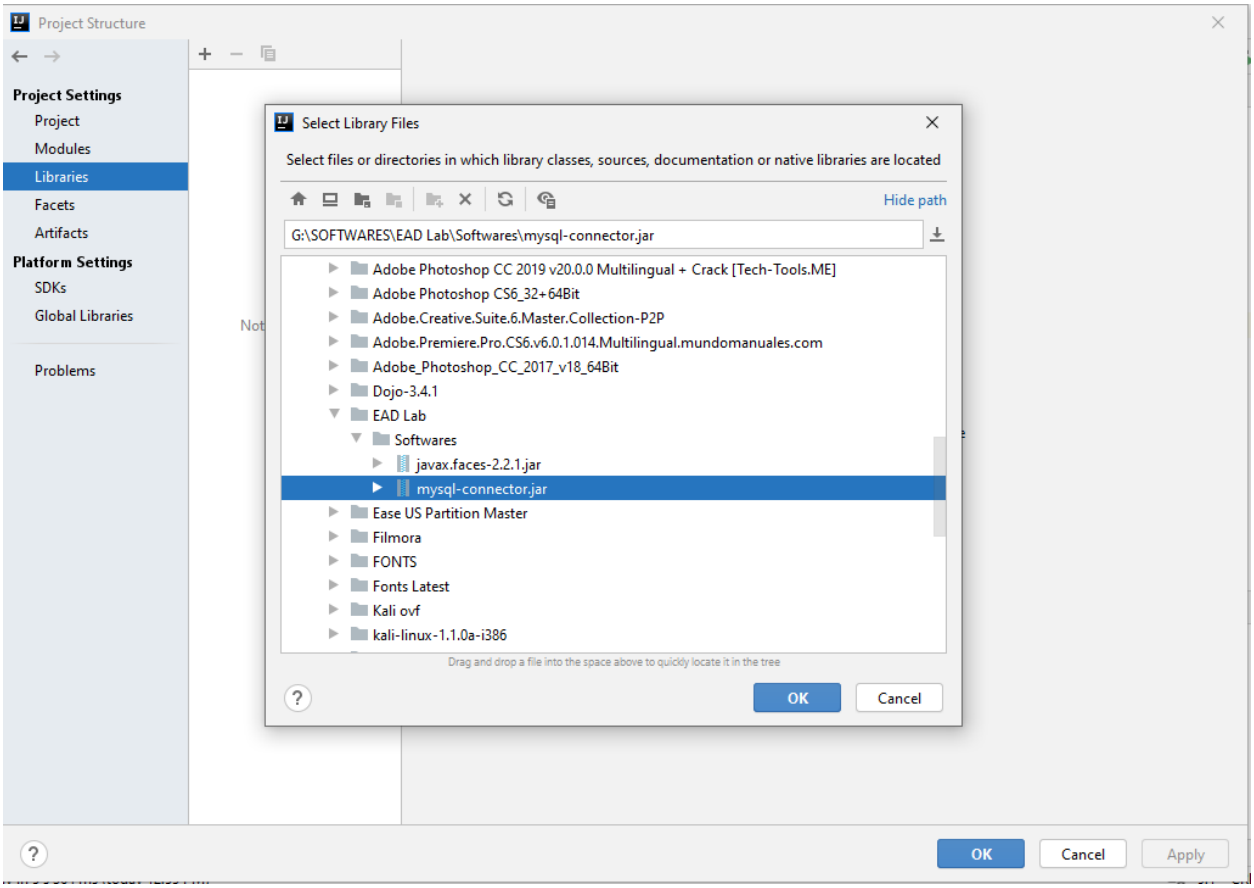


Step 3: To make JDBC work include sqlconnector in your project. Click on File button and click on Project Structure.



Step 4: Include sqlconnector.java file given in your lectures folder and click Apply.



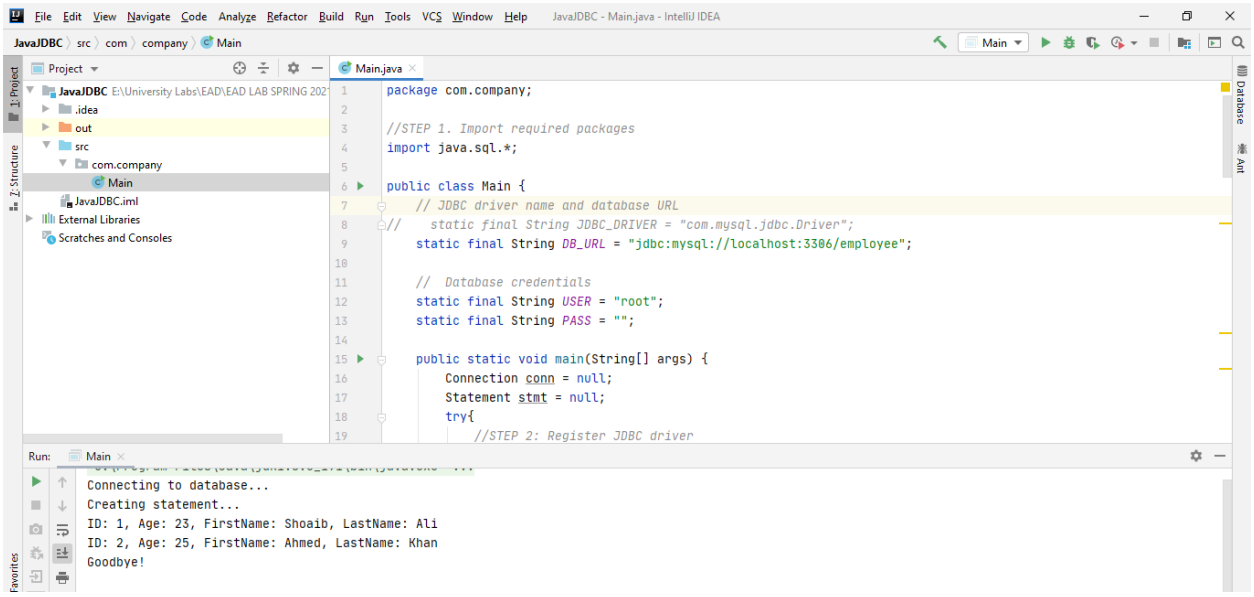


Step 5: Include following code in your main file and change it according to your database and table.

```
package com.company;

//STEP 1. Import required packages
import java.sql.*;

public class Main {
   // JDBC driver name and database URL
   static final String DB_URL = "jdbc:mysql://localhost:3306/employee";

   //  Database credentials
   static final String USER = "root";
   static final String PASS = "";

   public static void main(String[] args) {
      Connection conn = null;
      Statement stmt = null;
      try{
         //STEP 2: Register JDBC driver
         Class.forName("com.mysql.jdbc.Driver");

         //STEP 3: Open a connection
         System.out.println("Connecting to database...");
         conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

```
            //STEP 4: Execute a query
            System.out.println("Creating statement...");
            stmt = conn.createStatement();
            String sql;
            sql = "SELECT id, firstname, lastname, age FROM employee_info";
            ResultSet rs = stmt.executeQuery(sql);

            //STEP 5: Extract data from result set
            while(rs.next()){

               //Retrieve by column name
               int id  = rs.getInt("id");
               String first = rs.getString("firstname");
               String last = rs.getString("lastname");
               int age = rs.getInt("age");

               //Display values
               System.out.print("ID: " + id);
               System.out.print(", Age: " + age);
               System.out.print(", FirstName: " + first);
               System.out.println(", LastName: " + last);
            }
            //STEP 6: Clean-up environment
            rs.close();
            stmt.close();
            conn.close();
         }catch(SQLException se){
            //Handle errors for JDBC
            se.printStackTrace();
         }catch(Exception e){
            //Handle errors for Class.forName
            e.printStackTrace();
         }finally{
            //finally block used to close resources
            try{
               if(stmt!=null)
                  stmt.close();
            }catch(SQLException se2){
            }// nothing we can do
            try{
               if(conn!=null)
                  conn.close();
            }catch(SQLException se){
               se.printStackTrace();
            }//end finally try
         }//end try
         System.out.println("Goodbye!");
      }//end main
   }//end FirstExample
```

Step 6: Data from database will show on output.



9. Practice Tasks

9.1. Practice Task 1 [Expected time = 20 mins]

Create a class ***Glass*** with attributes opacity and thickness. The class contains an abstract method show. Extend a class ***WindowGlass*** from Glass and add two new attributes width and height. Override the show method that displays all the attributes. Create a class ***CarWindowGlass*** that extends ***WindowGlass*** and also define its own show method that displays all the attributes.

9.2. Practice Task 2 [Expected time = 20 mins]

Create a program in which you have two ***Rides (Car, Boat)*** and two ***locations (Land, Water)***. Ask from the user about the ride and the location. If user selects ride a car and location water, then throw an exception. Similarly, if user selects ride a boat and location water, then also throw an exception.

10. Evaluation Task (Unseen) [Expected time = 30 mins]

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.tutorialspoint.com/java/

From the above link you can have a detailed reading of all the concepts of Advanced Computer Programming Course.

Capital University of Science and Technology Islamabad
Department of Computing, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 03

Servlets-Part A

Lab 03: Servlets-Part A

1. Introduction

Servlet technology is primarily designed for use with the HTTP protocol of the Web. Servlets are Java programs that run on a Web server. Java servlets can be used to process client requests or produce dynamic Web pages. For example, you can write servlets to generate dynamic Web pages to process client registration form and store registration data into a database.
This Lab introduces the concept of Java servlets.

2. Relevant Lecture Reading

https://www.tutorialspoint.com/servlets/servlets_overview.htm

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
7.2	Setting up IDE	10 min	10 min
8.1	Walkthrough Tasks	65 min	70 min
9	Practice Tasks	70 min	70 min
10	Evaluation Task (Unseen)	30 min	30 min

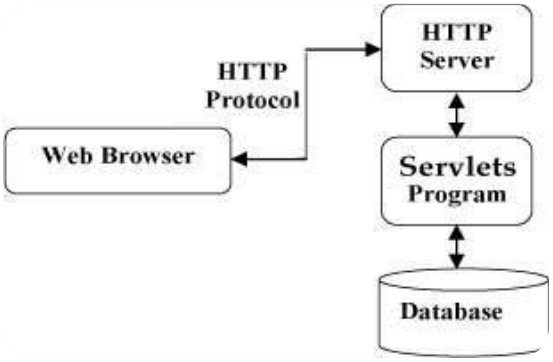
4. Objective of the Experiment

1. Understand the concept of servlets.
2. Know the servlets API.
3. Develop servlets to access databases using HTML.

5. Concept Map

5.1. Servlets Architecture:

Following diagram shows the position of Servlets in a Web Application.



5.2. Servlets Packages

Servlet is Java EE server driven technology to create web applications in java. The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing our own servlets. All servlets must implement the **javax.servlet.Servlet** interface, which defines servlet lifecycle methods. When implementing a generic service.

5.3. Servlet Mapping

Servlet mapping specifies the web container of which java servlet should be invoked for a url given by client. It maps url patterns to servlets. When there is a request from a client, servlet container decides to which application it should forward to. Then context path of url is matched for mapping servlets.

Servlets should be registered with servlet container. For that, you should add entries in web deployment descriptor web.xml. It is located in WEB-INF directory of the web application. Entries to be done in web.xml for servlet-mapping:

```
<servlet-mapping>
<servlet-name>milk</servlet-name>
<url-pattern>/drink/*</url-pattern>
</servlet-mapping>
```

servlet-mapping has two child tags, url-pattern and servlet-name. url-pattern specifies the type of urls for which, the servlet given in servlet-name should be called. Be aware that, the container will use case-sensitive for string comparisons for servlet matching.

5.4. HttpServlet class

HttpServlet is an abstract class that extends GenericServlet and provides base for creating HTTP based web applications. There are methods defined to be overridden by subclasses for different HTTP methods.

1. doGet(), for HTTP GET requests
2. doPost(), for HTTP POST requests
3. doPut(), for HTTP PUT requests
4. doDelete(), for HTTP DELETE requests

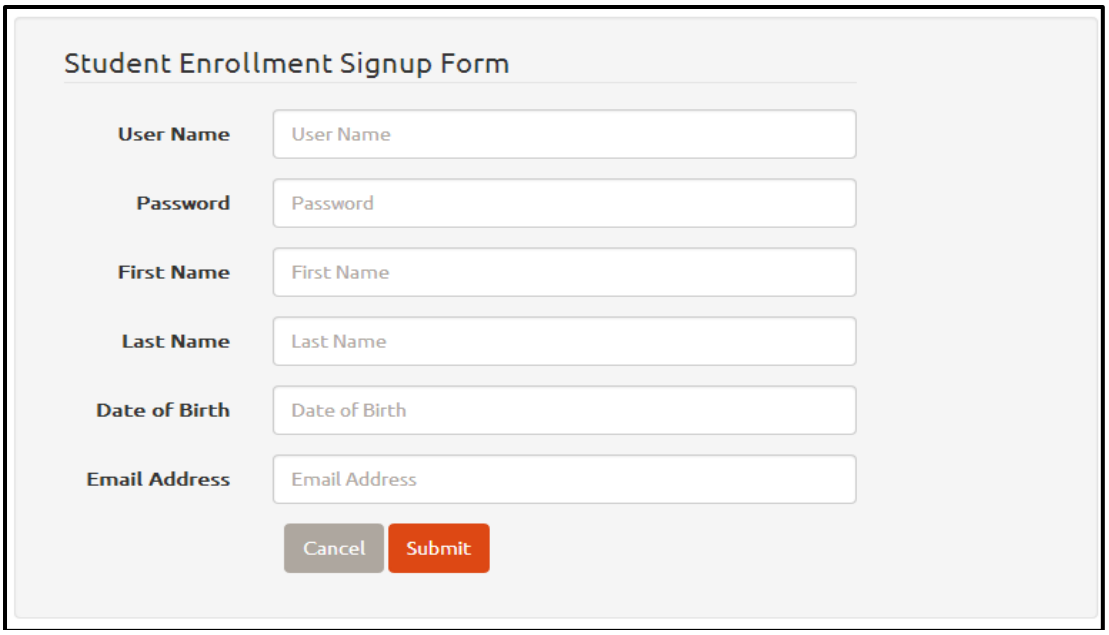
6. Homework before Lab

6.1. Homework 1

Install all the soft wares by getting help from ***section 7*** of ***Lab02***. It is assumed that you have installed all the soft wares before coming to labs.

6.2. Homework 2

Create a simple Servlet Class that displays the following ***LOGIN FORM*** on the browser. This task will be extended in your practice task.



7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. Xampp and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server

Refer to **Lab01** in order to setup environment and project.

8. Walkthrough task

8.1. Walkthrough task 1

This task explains how to create a servlet that has two controls

1. A text field.
2. A button.

You have to save the value of text field into the database using Servlets. Follow the below steps in order to perform that task.

It is assumed that you know how to create project and setup environment in IDEA.

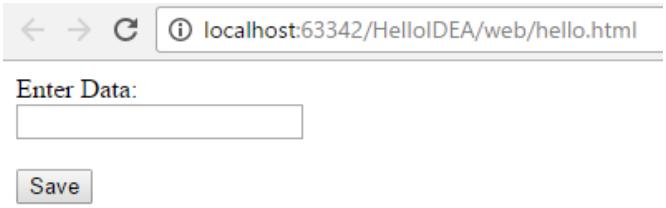
Step 1:

Create an HTML form and type the following code into it.

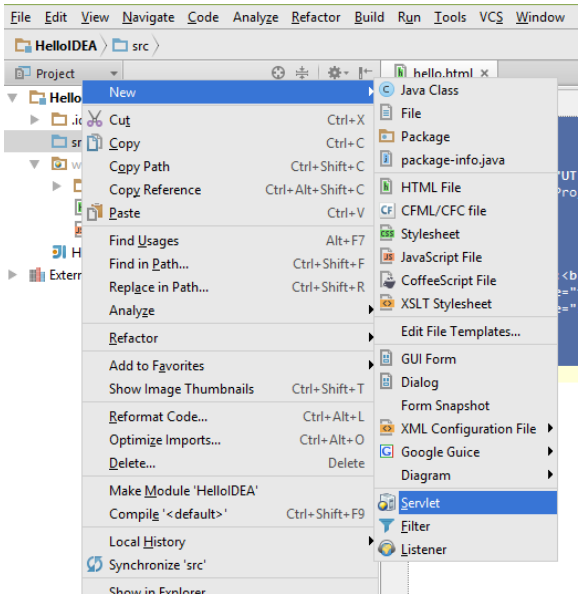
```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>myFirstProject</title>
</head>
<body>
```

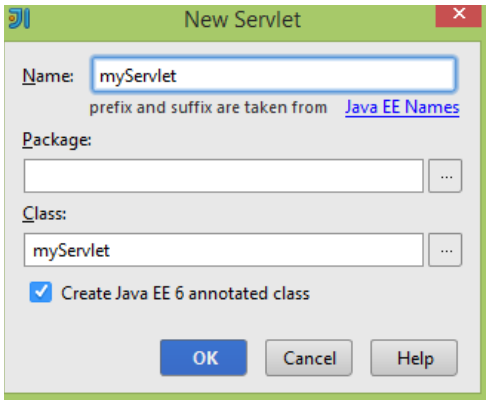
```
<div>
   <form>
      Enter Data:<br>
      <input type="text" name="firstname"><br><br>
      <input type="submit" name="save" value="Save"><br>
   </form>
</div>
</body>
</html>
```

Run and test the code, it should display something like this.



Step 2:
Create a servlet class by right clicking ***Src>New>Servlet*** and Enter the desired name of your servlet.After that Click Ok.





A servlet class will be creating having two methods doGet () and doPost ()

Step 3:
Next step is to do servlet mapping in web.xml file.
Enter the following code in ***web.xml*** file inside ***<web-app>..</web-app>*** tag.

```
<servlet>
   <servlet-name>myServlet</servlet-name> <!-- Should be the name of your Servlet-->
   <servlet-class>myServlet</servlet-class> <!-- Should be the path of your Servlet-->
</servlet>

<servlet-mapping>
   <servlet-name>myServlet</servlet-name> <!-- Should be the name of your Servlet which you have given inside servlet tag-->
   <url-pattern>/myServlet/*</url-pattern><!-- as desired but should be inside /../* -->
</servlet-mapping>
```

Step 4:
Make two changings in your html file.
Inside the form tag add two attributes.
Method should be ***post*** and the action should be the ***name*** of your servlet. Setting ***method="post"*** will call the ***doPost ()*** method of the servlet. If you want to call the ***doGet ()*** method you have to set ***method="get"***.

```
<form method="post" action="/myServlet/">
   Enter Data :<br>
   <input type="text" name="firstname"><br><br>
   <input type="submit" name="save" value="Save"><br>
</form>
```

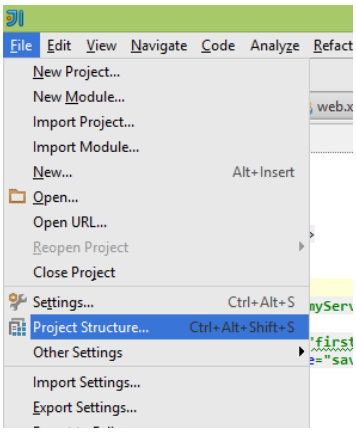
Step 5:
Now create a database named as **mydb,** create a table named as **mytable** and a column named as **myvalue** and then set its type to text.
You can execute following query in postgres to create table.

Step 6:

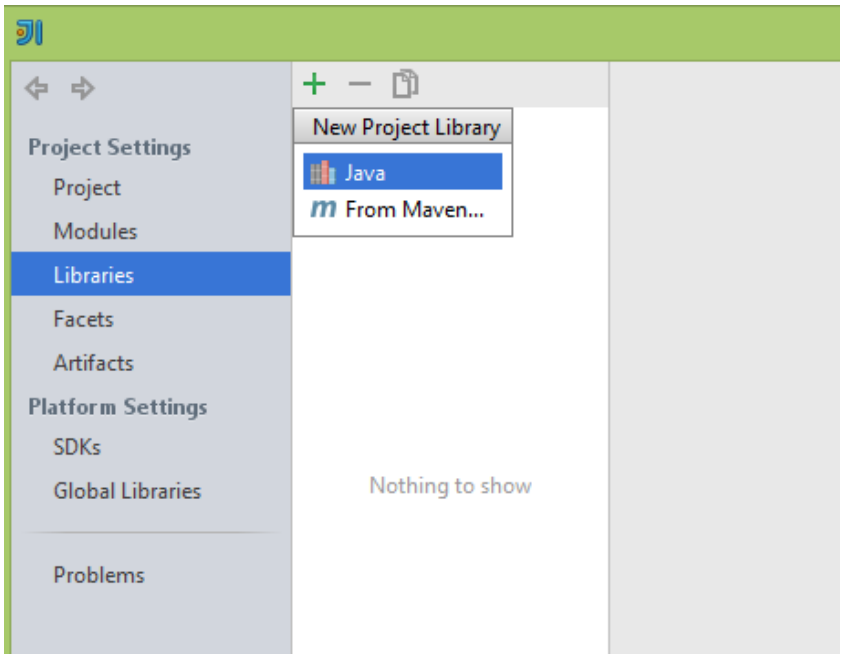
Download and include sqlconnector.java file as we done in Lab02 in your project.

Step 7:

Include the drivers in your project by going to ***File>Project Structure***



Go to Libraries Tab and select Java.



Go to the path where your Drivers are placed and after selecting driver file click Ok.

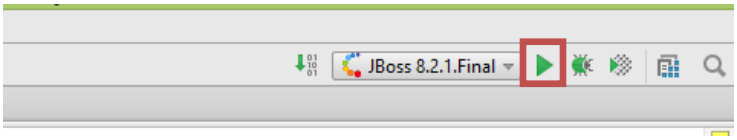
Step 8:

Now type the below code in the ***doPost ()*** method of the servlet and if you have set the form ***method="get"*** than type the code into ***doGet ()*** method

```
Connection c = null;
PreparedStatement pstmt;
```

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String data = request.getParameter("data");

try
{
   Class.forName("org.postgresql.Driver");
   c = DriverManager.getConnection("jdbc:postgresql://localhost:5432/testdb", "postgres", "samreen");
   pstmt=c.prepareStatement("insert into mytable(myvalue) values (?)");
   pstmt.setString(1,data);
   pstmt.executeUpdate();
   out.write("success");
}
catch (Exception ex)
{
   out.write(ex+"");
}
```

Step 9:
Run the program by clicking ▶ button on the top right side of window.



If you have followed all the steps than server will start successfully and you will be able to save the data into the database from the form you created.

9. Practice Tasks

9.1. Practice Task 1

Extend the home task given to you as follows.
The html form should be same as given to you in home task, save the data from the ***form*** into the ***database*** using ***servlets*** and if the data is saved successfully move to a page where you have to fetch all the data from the database and ***display into a table***. Else display ***error*** message to the user.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.tutorialspoint.com/servlets/
http://www.javatpoint.com/servlet-tutorial

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 04

Servlets-Part B

Lab 04: Servlets-Part B

1. Introduction

Servlet technology is primarily designed for use with the HTTP protocol of the Web. Servlets are Java programs that run on a Web server. Java servlets can be used to process client requests or produce dynamic Web pages. For example, you can write servlets to generate dynamic Web pages to process client registration form and store registration data into a database.
This Lab extends the concept of Java servlets. You will learn about Servlet filters.

2. Relevant Lecture Reading

http://tutorials.jenkov.com/java-servlets/servlet-filters.html

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	20 min	20 min
8.1	Walkthrough Tasks	60 min	60 min
9.1	Practice Tasks	As per time specified for each task	70 min
11	Evaluation Task (Unseen)	40 min	30 min

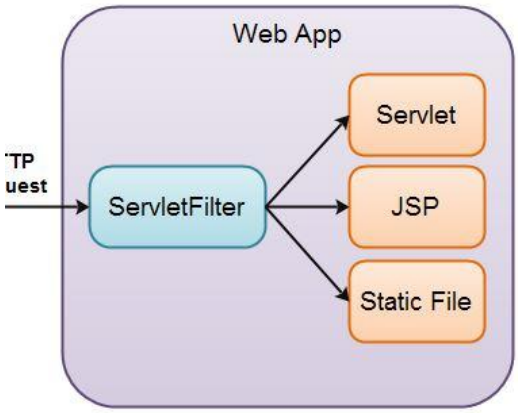
4. Objective of the Experiment

This Lab is designed to understand Servlets Filters

5. Concept Map

5.1. Servlet Filters

Java Servlet Filter is used to intercept request and do some pre-processing and can be used to intercept response and do post-processing before sending to client in web application.
Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor.
When the web container starts up your web application, it creates an instance of each filter that you have declared in the deployment descriptor. The filters execute in the order that they are declared in the deployment descriptor.



5.2. Usage of Filter

1. recording all incoming requests
2. logs the IP addresses of the computers from which the requests originate
3. conversion
4. data compression
5. encryption and decryption
6. Input validation etc.

The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

5.3. Advantage of Filter

1. Filter is pluggable.
2. One filter doesn't have dependency onto another resource.
3. Less Maintenance

5.4. Servlet Filter interface

Servlet Filter interface is similar to Servlet interface, and we need to implement it to create our own servlet filter. Servlet Filter interface contains lifecycle methods of a Filter and it's managed by servlet container.

Servlet Filter interface lifecycle methods are:

void init(FilterConfig paramFilterConfig):

When container initializes the Filter, this is the method that gets invoked. This method is called only once in the lifecycle of filter, and we should initialize any resources in this method. **FilterConfig** is used by container to provide init parameters and servlet context object to the Filter. We can throw ServletException in this method.

doFilter (ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain):

This is the method invoked every time by container when it has to apply filter to a resource. Container provides request and response object references to filter as argument. **FilterChain** is used to invoke the next filter in the chain.

void destroy():
When container offloads the Filter instance, it invokes the destroy() method. This is the method where we can close any resources opened by filter. This method is called only once in the lifetime of filter.

5.5. Simple Example of Filter

In this example, we are simply displaying information that filter is invoked automatically after the post processing of the request.

index.html

```
<a href="servlet1">click here</a>
```

MyFilter.java

In order to create a servlet filter you must implement the ***javax.servlet.Filter***.

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.*;

public class MyFilter implements Filter{

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp,
   FilterChain chain) throws IOException, ServletException {

   PrintWriter out=resp.getWriter();
   out.print("filter is invoked before");

   chain.doFilter(req, resp);//sends request to next resource

   out.print("filter is invoked after");
   }
   public void destroy() {}
}
```

HelloServlet.Java

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.*;
```

```
public class HelloServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      out.print("<br>welcome to servlet<br>");

   }

}
```

web.xml

For defining the filter, filter element of web-app must be defined just like servlet

```
<web-app>

<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>

<filter>
<filter-name>f1</filter-name>
<filter-class>MyFilter</filter-class>
</filter>

<filter-mapping>
<filter-name>f1</filter-name>
<url-pattern>/servlet1</url-pattern>
</filter-mapping>
</web-app>
```

6. Homework before Lab

6.1. Homework 1

Install all the soft wares by getting help from ***section 7*** of ***Lab02***. It is assumed that you have installed all the soft wares before coming to labs.

6.2. Homework 2

Create a filter for servlet ***myServlet.java*** that records the duration of all requests.

7. Tools and Procedure

7.1. Tools

- IntelliJ Idea.

- Xampp and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server

Refer to Section 7 of **Lab02** in order to setup environment and project.

8. Walkthrough task

8.1. Walkthrough Task 1

Creating a filter that blocks servlet request from localhost.

Create a package ***com*** and then create a servlet in it.Name the servlet as ***IPServlet.***
Type below code into it

```
package com;


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;

@WebServlet(name = "ServletIP")
public class ServletIP extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {

   }

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
      PrintWriter pw=response.getWriter();
      pw.write("Welcome To Servlet!");
   }
}
```

Now create a filter in the same package and name it as ***FilterIP***.Type below code into it.

```
package com;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;
import java.io.PrintWriter;

@WebFilter(filterName = "FilterIP")
public class FilterIP implements javax.servlet.Filter {
   public void destroy() {
   }

   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws ServletException,
```

```
IOException {
        PrintWriter pw=resp.getWriter();
        String localhost="";
        String ip=req.getRemoteHost();
        if(ip.matches("127.0.0.1"))
        {

            pw.write("127.0.0.1 is Blocked!");
        }
        else
        {
            chain.doFilter(req, resp);
        }
    }

    public void init(FilterConfig config) throws ServletException {

    }

}
```

Now configure and map the ***servlet*** in web.xml file as you have done before.

```
<servlet>
    <servlet-name>IPServlet</servlet-name>
    <servlet-class>com.ServletIP</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>IPServlet</servlet-name>
    <url-pattern>/IP</url-pattern>
</servlet-mapping>
```

Similarly map and configure the ***Servlet filter*** in web.xml file.

```
<filter>
    <filter-name>IPFilter</filter-name>
    <filter-class>com.FilterIP</filter-class>
</filter>
<filter-mapping>
    <filter-name>IPFilter</filter-name>
    <url-pattern>/IP</url-pattern>
</filter-mapping>
```

The URL pattern of the filter tells that this filter is for ***IPServlet***. Note that the url pattern of filter and servlet is same.

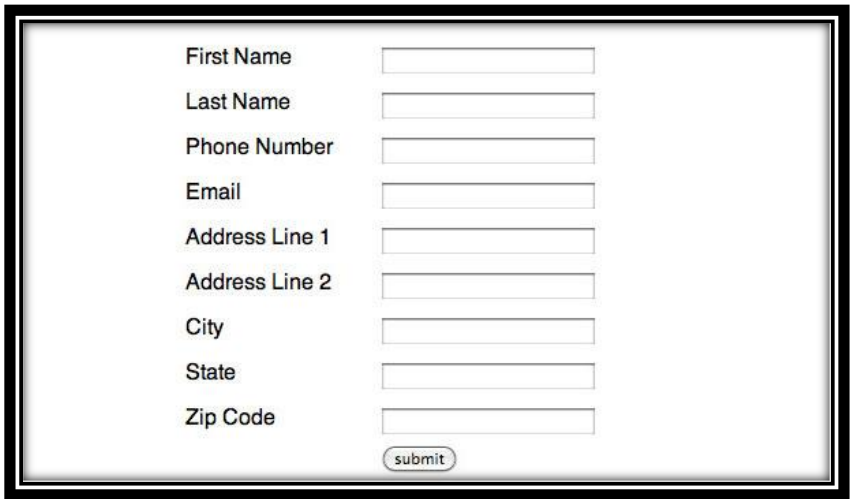
Now Run the Program by typing below URL of the servlet in the URL bar.

The output is shown below. Try to change the IP address in *doFilter ()* method and then observe the output.

9. Practice Tasks

9.1. Practice Task 1

Create below form using html



When ***submit*** button is clicked; validate all the fields except ***Address Line 2*** using filter.
If any of the fields is empty display a message to the user that ***"none of the fields should be empty"*** else save the data to ***database*** and after saving the data; ***Success*** message should be displayed to user using servlet filter.

Hint: Make some Logic in filter before processing of the servlet for validation and after processing display Success message.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.tutorialspoint.com/servlets/servlets-writing-filters.htm
http://www.javatpoint.com/servlet-filter

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 05

Java Server Pages-Part A

Lab 05: Java Server Pages- Part A

1. Introduction

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.
This Lab will teach you how to use Java Server Pages to develop your web applications in simple and easy steps.

2. Relevant Lecture Reading

http://www.tutorialspoint.com/jsp/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	30 min	30 min
8	Walkthrough Tasks	55 min	55 min
9.1	Practice Tasks	65 min	65 min
8	Evaluation Task (Unseen)	30 min	30 min

4. Objective of the Experiment

This Lab is designed to understand

1. What JSPs are.The differences between servlets and JSPs.
2. To create and deploy Java Server Pages.
3. To use JSP's implicit objects and Scriplets to create dynamic Web pages.
4. To use actions to manipulate JavaBeans in a JSP, to include resources dynamically and to forward requests to other JSPs.

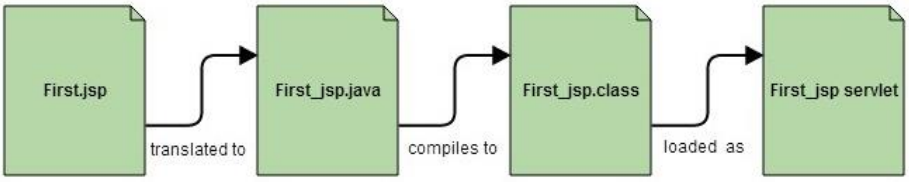
5. Concept Map

5.1. Introduction to JSP

JSP technology is used to create dynamic web applications. **JSP** pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets as a servlet adds HTML code inside Java code, while JSP adds Java code inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it. JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. **Using JSP, one can easily separate Presentation and Business logic** as a web designer can design and update JSP pages creating the presentation layer and java developer can write server-side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

5.2. In the end a JSP becomes a Servlet

JSP pages are converted into **Servlet** by the Web Container. The Container translates a JSP page into servlet **class source (.java)** file and then compiles into a Java Servlet class.



5.3. Why JSP is preferred over servlets?

1. JSP provides an easier way to code dynamic web pages.
2. JSP does not require additional files like, java class files, web.xml etc.
3. Any change in the JSP code is handled by Web Container (Application server like Tomcat) and doesn't require re-compilation.
4. JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

5.4. Advantage of JSP

1. Easy to maintain and code.
2. High Performance and Scalability.
3. JSP is built on Java technology, so it is platform independent.

5.5. JSP Scripting Element

JSP Scripting element is written inside <% %> tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

Example:

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
  <%
     int count = 0;
  %>
  <body>
      Page Count is  <% out.println(++count); %>
  </body>
```

Just to experiment, try removing the <% %> scriplet tag from the above code and run it as JSP. You will see that everything is printed as it is on the browser, because without the scriplet tag, everything is considered plain HTML code.

5.6. Five different types of scripting elements

Scripting Element	Example

Comment	<%-- comment --%>
Directive	<%@ directive %>
Declaration	<%! Declarations %>
Scriplets	<% Scriplets %>
Expression	<%= expression %>

5.7. JSP Comment

JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing. JSP comments are only seen in the JSP page. These comments are not included in servlet source code during translation phase, nor do they appear in the HTTP response. Syntax of JSP comment is as follows:

```
<%-- JSP comments -- %>
```

Simple Example of JSP Comment

```
<html>
   <head>
      <title>My First JSP Page</title>
   </head>
  <%
     int count = 0;
  %>
  <body>
      <%-- Code to show page count  --%>
      Page Count is  <% out.println(++count); %>
  </body>
```

NOTE: Adding comments in your code is considered to be a good practice in Programming world.

6. Homework before Lab

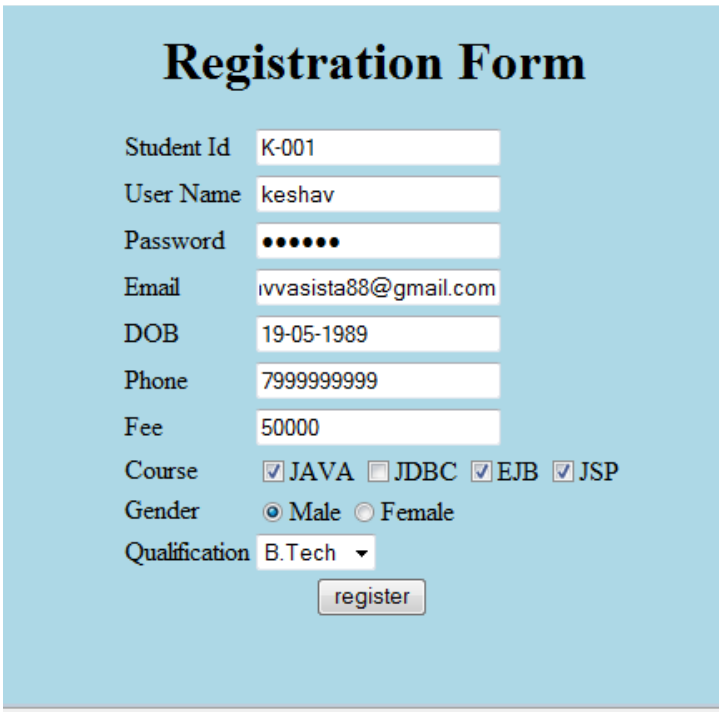
From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

6.1. Homework 1

Install all the tools and configure the environment before coming to Lab if you have not done it already.

6.2. Homework 2

Create the below simple registration form in ***JSP,*** named as ***Register.jsp***



When the user clicks on the register button, redirect to a new page named as ***display.jsp*** that displays all the ***values*** on the web page that user entered in the text fields.

7. Tools and Procedure

7.1. Tools

1. IntelliJ Idea.
2. Xampp and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server

Refer to **Lab01** in order to setup environment and project.

8. Walkthrough task

8.1. Walkthrough task 1

To create project and set **environment** and configure IDEA, refer to **Lab01.**
In order to create **database** refer to walkthrough task of ***Lab02***.

8.2. Walkthrough task 2

Creating a form and displaying its values when user clicks on Show button.

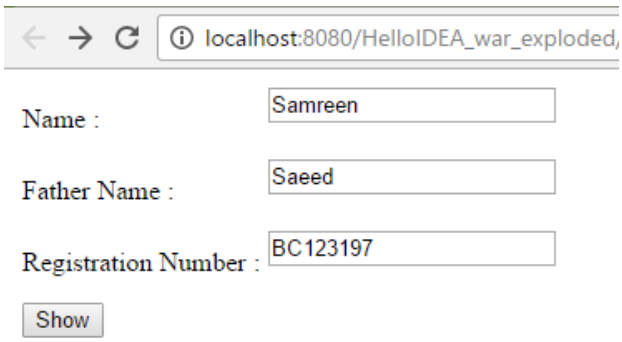
Create a new JSP page by right clicking ***Project>Web>New>JSP/JSPX***

Name it as ***student.jsp*** and type the following code into it.

```
<html>
 <head>
  <title></title>
 </head>
 <body>
 <form action="show.jsp" method="post">
 <div>
```

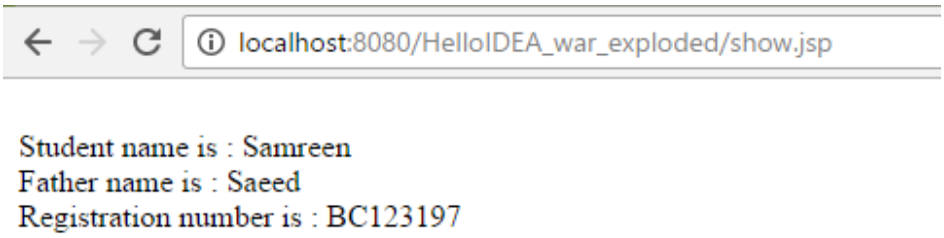
```
  <table>
    <tr>
      <td>Name :</td>
      <td><input type="text" name="name"/><br><br></td>
    </tr>
    <tr>
      <td> Father Name :</td>
      <td><input type="text" name="father"/><br><br></td>
    </tr>
    <tr>
      <td>Registration Number :</td>
      <td><input type="text" name="reg"/><br><br></td>
    </tr>
    <tr>
      <td><input type="submit" value="Show" name="buttonshow"/></td>
    </tr>
  </table>
 </div>
 </form>
 </body>
</html>
```

Understand the above code carefully that how the form is created using tables. Also the form tag is having two attributes ***action="show.jsp"*** and ***method="post".***



After that Create a new page named as ***show.jsp*** and type the following code into it.

```
<%@ page import="java.io.PrintWriter" %><!--Directives example-->

<%
 String user = request.getParameter("name");
 String father = request.getParameter("father");
 String reg = request.getParameter("reg");

 PrintWriter pw=response.getWriter();
 pw.write("<html>\n" +
        "<head>\n" +
        "  <title></title>\n" +
        "</head>\n" +
        "<body><br>" +
        "Student name is : "+user+"<br>"+
        "Father name is : "+father+"<br>"+
        "Registration number is : "+reg+"<br>"+
        "</body>\n" +
        "</html>");
%>
```



When the user clicks ***show*** button, control is directed to a new page where all the values are extracted from the ***request*** object and then displayed on the web page using ***PrintWriter.***

9. Practice Tasks

9.1. Practice Task 1

Extend the home task given to you as follows.
The html form should be same as given to you in home task
Save the data from the ***form*** into the ***database*** using ***JSP.***
When the user clicks Register button, control should move to a new page named as ***save.jsp*** and there you have to save all the values into the database.
It is assumed that you know how to create ***database, tables and columns*** in PostgreSql.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.javaknowledge.info/login-and-registration-example-in-jsp-with-session/
https://www.tutorialspoint.com/jsp
http://www.journaldev.com/2021/jsp-example-tutorial-for-beginners
http://www.studytonight.com/jsp

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 06

Java Server Pages-Part B

Lab 06: Java Server Pages-Part B

1. Introduction

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.
This Lab will teach you some advanced concepts of Java Server Pages to develop your web applications.

2. Relevant Lecture Reading

http://www.tutorialspoint.com/jsp/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
6.2	Setting up to use HTML	10 min	10 min
6.3	Walkthrough Tasks	65 min	65min
7	Practice Tasks	As per time specified for each task	45 min
8	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

1. JSP Custom Tags.
2. JavaBeans (POJO) with JSP.
3. Mixing Scriplets and bean Tags.
4. JSTL

5. Concept Map

5.1. What is Java Bean?

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.
Following are the unique characteristics that distinguish a JavaBean from other Java classes:

1. It provides a default, no-argument constructor.
2. It should be serializable and implement the **Serializable** interface.
3. It may have a number of properties which can be read or written.
4. It may have a number of "getter" and "setter" methods for the properties.

5.1.1. JavaBeans Properties:

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including classes that you define.
A JavaBean property may be read, write, read only, or write only. JavaBean properties are accessed through two methods in the JavaBean's implementation class:

Method	Description
getPropertyName()	For example, if property name is *firstName*, your method name would be getFirstName() to read that property. This method is called accessor.
setPropertyName()	For example, if property name is *firstName*, your method name would be setFirstName() to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName** () method, and a write-only attribute will have only a **setPropertyName** () method.

5.1.2. JavaBeans Example:

Consider a student class with few properties:

```
public class StudentsBean implements java.io.Serializable
{
  private String firstName = null;
  private String lastName = null;
  private int age = 0;

  public StudentsBean() {
  }
  public String getFirstName(){
    return firstName;
  }
  public String getLastName(){
    return lastName;
  }
  public int getAge(){
    return age;
  }
  public void setFirstName(String firstName){
    this.firstName = firstName;
  }
  public void setLastName(String lastName){
    this.lastName = lastName;
  }
  public void setAge(Integer age){
    this.age = age;
  }
}
```

5.1.3. Accessing JavaBeans:

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows:

```
<jsp:useBean id="bean's name" scope="bean's scope" typeSpec/>
```

Here values for the scope attribute could be page, request, session or application based on your requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other useBean declarations in the same JSP.
Following example shows its simple usage:

```
<html>
<head>
<title>useBean Example</title>
</head>
<body>

<jsp:useBean id="date" class="java.util.Date" />
<p>The date/time is <%= date %>

</body>
</html>
```

This would produce following result:

```
The date/time is Thu Sep 30 11:18:11 GST 2010
```

5.1.4. Accessing JavaBeans Properties:

Along with <jsp:useBean...>, you can use <jsp:getProperty/> action to access get methods and <jsp:setProperty/> action to access set methods. Here is the full syntax:

```
<jsp:useBean id="id" class="bean's class" scope="bean's scope">
  <jsp:setProperty name="bean's id" property="property name"
                   value="value"/>
  <jsp:getProperty name="bean's id" property="property name"/>
  ...........
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the get or set methods that should be invoked.
Following is a simple example to access the data using above syntax:

```
<html>
<head>
<title>get and set properties Example</title>
</head>
<body>

<jsp:useBean id="students" class="StudentsBean">
  <jsp:setProperty name="students" property="firstName"
```

```
            value="Zara"/>
  <jsp:setProperty name="students" property="lastName"
            value="Ali"/>
  <jsp:setProperty name="students" property="age"
            value="10"/>
</jsp:useBean>

<p>Student First Name:
  <jsp:getProperty name="students" property="firstName"/>
</p>
<p>Student Last Name:
  <jsp:getProperty name="students" property="lastName"/>
</p>
<p>Student Age:
  <jsp:getProperty name="students" property="age"/>
</p>

</body>
</html>
```

Try to access above JSP. This would produce following result:

```
Student First Name: Zara
Student Last Name: Ali
Student Age: 10
```

5.1.5. Attributes and Usage of jsp:useBean action tag

Id: is used to identify the bean in the specified scope.
Scope: represents the scope of the bean. It may be page, request, session or application. The default scope is page.
Page: specifies that you can use this bean within the JSP page. The default scope is page.
Request: specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
Session: specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
Application: specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
Class: instantiates the specified bean class (i.e., creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
Type: provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
beanName: instantiates the bean using the java.beans.Beans.instantiate() method.

5.2. JSP Custom Tags:

User-defined tags are known as **custom tags**. In this tutorial we will see how to **create a custom tag** and use it in JSP.
To create a custom tag, we need three things:

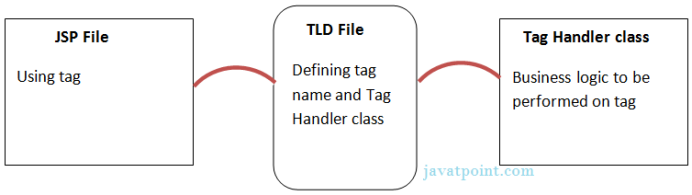
5.2.1. Tag handler class:

In this class we specify what our custom tag will do when it is used in a JSP page.

5.2.2. TLD files:

Tag descriptor file where we will specify our tag name, tag handler class and tag attributes. **5.2.3. JSP page:**

A JSP page where we will be using our custom tag.



Example:

In the below example we are creating a custom tag ***MyMsg*** which will display the message ***"This is my own custom tag"*** when used in a JSP page.

5.2.3. Tag handler class:

A tag handler class should implement ***Tag/IterationTag/ BodyTaginterface*** or it can also *extend* ***TagSupport/BodyTagSupport/SimpleTagSupport*** class. All the classes that support custom tags are present inside ***javax.servlet.jsp.tagext***. In the below we are extending the class ***SimpleTagSupport.***

Details.java

```
package beginnersbook.com;
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
import java.io.*;
public class Details extends SimpleTagSupport {
  public void doTag() throws JspException, IOException {
    /*This is just to display a message, when
     * we will use our custom tag. This message
     * would be displayed
     */
    JspWriter out = getJspContext().getOut();
    out.println("This is my own custom tag");
  }
}
```

5.2.4. TLD File

This file should present at the location: ***Project Name/WebContent/WEB-INF/*** and it should have a ***.tld*** extension.

Note:

<name> tag: custom tag name. In this example we have given it as ***MyMsg***

<tag-class> tag: Fully qualified class name. Our tag handler ***classDetails.java*** is in package ***com*** so we have given the value as ***com.Details.***

message.tld

```
<taglib>
 <tlib-version>1.0</tlib-version>
```

```
<jsp-version>2.0</jsp-version>
<short-name>My Custom Tag</short-name>
<uri>/WEB-INF/custom</uri>

<tag>
<name>MyMsg</name>
<tag-class>com.Details</tag-class>
</tag>
</taglib>
```

5.2.6. Using custom tag in JSP:

Above we have created a custom tag named ***MyMsg.*** Here we will be using it.
Note: taglib directive should have the ***TLD file path*** in uri field. Above we have created the ***message.tld*** file so we have given the path of that file.
Choose any prefix and specify it in taglib directive's prefix field. Here we have specified it as myprefix. Custom tag is called like this: ***<prefix:tagName/>.*** Our prefix is ***myprefix*** and tag name is ***MyMsg*** so we have called it as ***<myprefix:MyMsg/>*** in the below JSP page.

```
<%@ taglib prefix="myprefix" uri="/WEB-INF/message.tld"%>
<html>
<head>
  <title>Custom Tags in JSP Example</title>
</head>
<body>
  <myprefix:MyMsg/>
</body>
</html>
```

Output:

```
This is my own custom tag
```

5.3. JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

5.3.1. Advantage of JSTL

1. Fast Development JSTL provides many tags that simplifies the JSP.
2. Code Reusability We can use the JSTL tags in various pages.
3. No need to use scriplet tag it avoids the use of scriplet tag.

5.3.2. JSTL Tags

The JSTL mainly provides 5 types of tags:
For detail reading links are provided in 11. Further Reading section.

Tag Name	Description
Core tags	The JSTL core tag provides variable support, URL management, flow control etc. The url for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is c.
Function tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is fn.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting etc. The url for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is fmt.

XML tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml** and prefix is x.
SQL tags	The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is sql.

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

6.1. Homework 1

Install all the tools and configure the environment before coming to Lab if you have not done it already.

6.2. Homework 2

Create a custom tag named as **<myTime:today/>** that displays Current Date and time in the below format

Current Date and Time is: Wed Sep 21 20:02:10 PDT 2016

Hint: use **Calendar.getInstance ().getTime ()** to get time in Tag Handler Class.

7. Tools and Procedure

7.1. Tools

1. IntelliJ Idea.
2. Xampp and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to **Lab01** in order to setup environment and project.

8. Walkthrough task.

8.1. Walkthrough task 1

To create project and set **environment** and configure IDEA, refer to walkthrough task 1 of **Lab01.**
In order to create **database,** refer to walkthrough task of ***Lab02***.

8.2. Walkthrough task 2

Calculate ***Cube*** of a number through ***JavaBean***.
Create a JSP page and type the following code into it.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
   <title></title>
</head>
<body>
<form method="post" action="solution.jsp">
 Enter Number : <input type="text" name="number">
 <input type="submit" value="Calculate">
</form>
</body>
</html>
```

Create a package ***cube*** in ***src*** folder and create a class named as ***CubeBean.java*** in it.
After that type the following code into it.

```
package cube;

public class CubeBean {

   public int cube(int n)
   {
      return n*n*n;
   }
}
```

Create a ***JSP*** page and type following code into it.

```
<jsp:useBean id="obj" class="cube.CubeBean"/>
<%

  int number=Integer.parseInt(request.getParameter("number"));
  int cube=obj.cube(number);
%>
<p>
  <%
    out.print("Cube is "+cube);
  %>
</p>
```

After that run the program and test the output. Final output should be something like this.

8.3. Walkthrough task 3

Here is an example of a single evaluation where we extend the **BodyTagSupport** class. This example reads the body content and converts it to uppercase or lowercase depending on the attribute provided by us. The reason to use **BodyTagSupport** is that this tag will contain some body text that will be converted to uppercase or lowercase.

At first, we need to create the tag handler class: ***caseTagHandler.java***

```
public class caseTagHandler extends BodyTagSupport {
   private String mCase;

   public void setCase(String pCase) {
      mCase = pCase;
   }

   public int doAfterBody() throws JspException {
      BodyContent bc = getBodyContent();
      String body = bc.getString();
      JspWriter out = bc.getEnclosingWriter();
      try
```

```
    {
      if ("upper".equalsIgnoreCase(mCase)) {
        out.print(body.toUpperCase()+"<hr>");
      } else {
        out.print(body.toLowerCase());
      }

  } catch (IOException ioe) {
    throw new JspException("Error: " + ioe.getMessage());
  }
  return SKIP_BODY;
  }
}
```

Create TLD file named as ***customTag.tld*** or your desired name.
Note that this tag also contains attribute that accepts some value from the **JSP** file.

```
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
jsptaglibrary_2_1.xsd"
      version="2.1">

  <tlib-version>1.0</tlib-version>
  <short-name>myTag</short-name>
  <uri>/WEB-INF/customTag</uri>
  <tag>
    <name>changeCase</name>
    <tag-class>com.caseTagHandler</tag-class>
    <body-content>JSP</body-content>
    <attribute>
      <name>case</name>
      <required>true</required>
      <rtexprvalue>false</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```

After that create a ***JSP*** file and type below code into it.In first case ***case="upper"*** is the value of attribute, by this it will be understood that the body text is to be converted into Uppercase. Same is in the second case but the value of attribute ***case*** is lower that means the body text is to be converted in lower.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ taglib uri="/WEB-INF/customTag.tld" prefix="ct" %>
<html>
<head>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
 <title>Upper or Lower Case</title>
</head>
<body>
<h2>
 Upper Case :
 <ct:changeCase case="upper">
  Hello World
 </ct:changeCase>
</h2>
```

```
<h2>
  Lower Case :
  <ct:changeCase case="lower">
    Hello World
  </ct:changeCase>
</h2>
</body>
</html>
```

Run the JSP page and you will be displayed with following output.

9. Practice Tasks

9.1. Practice Task 1

Create a JSP page having a **textField** and a **button**. User should enter a string into textField and when the submit button is clicked; output should be the length of String and also print either the length of string is even or odd. Remember that you have to calculate length of String using ***JavaBean***; the name of JavaBean should be ***calculateLength***.

9.2. Practice Task 2

Create a JSP page that performs following task using ***JSTL***. Iterate a loop from 1 to 25 and check whether the iteration number is the multiple of 2 or 4. If the number is multiple of 2 or 4 it will store it into database having table ***mynumbers.*** The table should have only one column ***Selectednumbers.***
Remember that you have to perform the above task using JSTL tags.

Note: You have to include the JSTL jar file in your project in order to use its tags. The file will be provided to you by your instructor in Lab.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.javatpoint.com/jstl-function-tags
http://www.javatpoint.com/jstl-core-tags
http://www.javatpoint.com/jstl-formatting-tags
http://www.javatpoint.com/jstl-xml-tags
http://www.javatpoint.com/jstl-sql-tags

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 07

Java Server Faces-Part A

Lab 07: Java Server Faces-Part A

1. Introduction

Java Server Faces (JSF) technology is a front end framework which makes the creation of user interface components easier by reusing the UI components. JSF is designed based on the **Model View Controller** pattern (MVC) which segregates the presentation, controller and the business logic.

This Lab is designed to understand basics JSF basics and to create first JSP application.

2. Relevant Lecture Reading

http://www.tutorialspoint.com/jsf/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	10 min	10 min
8.1	Walkthrough Tasks	40 min	40 min
9.1	Practice Tasks	90 min	90 min
10	Evaluation Task (Unseen)	40 min	40 min

4. Objective of the Experiment

This Lab is designed to understand

1. JSF Basics.
2. JSF Basic Tags.
3. Managed Bean Basics.
4. Creating First JSF application.

5. Concept Map

5.1. What is JSF?

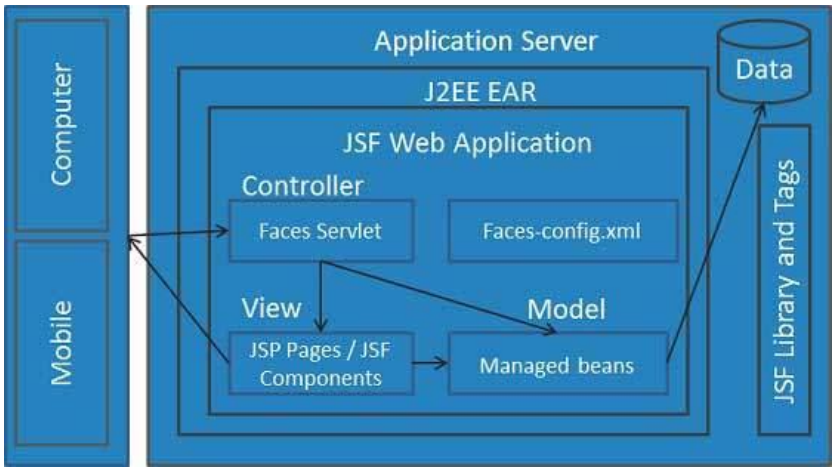
Java Server Faces (JSF) is a MVC web framework that simplifies the construction of user interfaces (UI) for server-based applications by using reusable UI components in a page.JSF provides facility to connect UI widgets with data sources and to server-side event handlers. The

JSF specification defines a set of standard UI components and provides an Application Programming Interface (API) for developing components.JSF enables the reuse and extension of the existing standard UI components.

5.2. JSF Architecture

A JSF application is similar to any other Java technology-based web application; it runs in a Java servlet container, and contains

1. JavaBeans components as models containing application-specific functionality and data
2. A custom tag library for representing event handlers and validators
3. A custom tag library for rendering UI components
4. UI components represented as stateful objects on the server
5. Server-side helper classes
6. Validators, event handlers, and navigation handlers
7. Application configuration resource file for configuring application resources



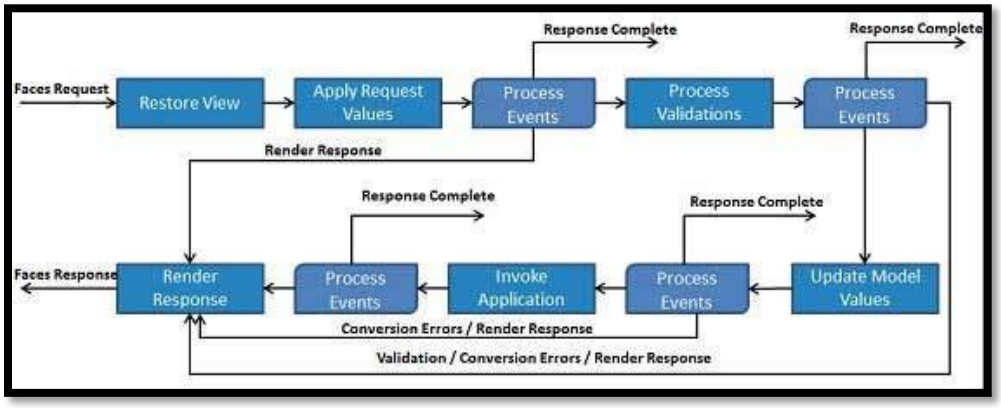
There are controllers which can be used to perform user actions.UI can be created by web page authors and the business logic can be utilized by managed beans.

JSF provides several mechanisms for rendering an individual component. It is up to the web page designer to pick the desired representation, and the application developer doesn't need to know which mechanism was used to render a JSF UI component.

5.3. JSF Life Cycle

JSF application lifecycle consist of six phases which are as follows

1. Restore view phase
2. Apply request values phase, process events
3. Process validations phase; process events
4. Update model values phase; process events
5. Invoke application phase, process events
6. Render response phase



Phase 1: Restore view

JSF begins the restore view phase as soon as a link or a button is clicked and JSF receives a request. During this phase, the JSF builds the view, wires event handlers and validators to UI components and saves the view in the FacesContext instance. The FacesContext instance will now contains all the information required to process a request.

Phase 2: Apply request values

After the component tree is created/restored, each component in component tree uses decode method to extract its new value from the request parameters. Component stores this value. If the conversion fails, an error message is generated and queued on FacesContext. This message will be displayed during the render response phase, along with any validation errors. If any decode methods / event listeners called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.

Phase 3: Process validation

During this phase, the JSF processes all validators registered on component tree. It examines the component attribute rules for the validation and compares these rules to the local value stored for the component. If the local value is invalid, the JSF adds an error message to the FacesContext instance, and the life cycle advances to the render response phase and display the same page again with the error message.

Phase 4: Update model values

After the JSF checks that the data is valid, it walks over the component tree and set the corresponding server-side object properties to the components' local values. The JSF will update the bean properties corresponding to input component's value attribute. If any updateModels methods called renderResponse on the current FacesContext instance, the JSF moves to the render response phase.

Phase 5: Invoke application
During this phase, the JSF handles any application-level events, such as submitting a form / linking to another page.

Phase 6: Render response
During this phase, the JSF asks container/application server to render the page if the application is using JSP pages. For initial request, the components represented on the page will be added to the component tree as the JSP container executes the page. If this is not an initial request, the component tree is already built so components need not to be added again. In either case, the components will render themselves as the JSP container/Application server traverses the tags in the page. After the content of the view is rendered, the response state is saved so that subsequent requests can access it and it is available to the restore view phase.

5.4. Benefits

JSF reduces the effort in creating and maintaining applications which will run on a Java application server and will render application UI on to a target client.JSF facilitates Web application development by

1. providing reusable UI components
2. making easy data transfer between UI components
3. managing UI state across multiple server requests
4. enabling implementation of custom components
5. wiring client-side event to server side application code

5.5. JSF Component Types

JSF provides the following components to create an user interface:

1. Standard basic input elements like fields, buttons etc. that forms the set of base UI components.
2. Rendering ability of JSF depending on the client specifications
3. Core library
4. Extending available UI components to add more components and use them for accomplishing client requirements.

5.6. What is MVC Design Pattern?

MVC design pattern designs an application using three separate modules:

Module	Description
Model	Carries Data and login
View	Shows User Interface
Controller	Handles processing of an application.

5.7. Managed Bean

1. Managed Bean is a regular Java Bean class registered with JSF. In other words, Managed Beans is a java bean managed by JSF framework.
2. The managed bean contains the getter and setter methods, business logic or even a backing bean (a bean contains all the HTML form value).
3. Managed beans work as Model for UI component.
4. Managed Bean can be accessed from JSF page.
5. In JSF 1.2, a managed bean had to register it in JSF configuration file such as faces-config.xml.
6. From JSF 2.0 onwards, Managed beans can be easily registered using annotations. This approach keeps beans and their registration at one place and it becomes easier to manage.

5.7.1. Scope Annotations

Scope annotations set the scope into which the managed bean will be placed. If scope is not specified, then bean will default to request scope. Each scope is briefly discussed below.

@SessionScoped: Indicates that the bean is valid for the whole session
@RequestScoped: The bean is valid for the http request.
@ApplicationScoped: Bean is valid as long as the web application is valid.

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

6.1. Homework 1

Install all the tools and configure the environment before coming to Lab if you have not done it already.

6.2. Homework 2

Study the concept map of this Lab and make a summary of 2 Pages.

7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. PostgreSql and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to Section 7 of **Lab02** in order to setup environment and project.

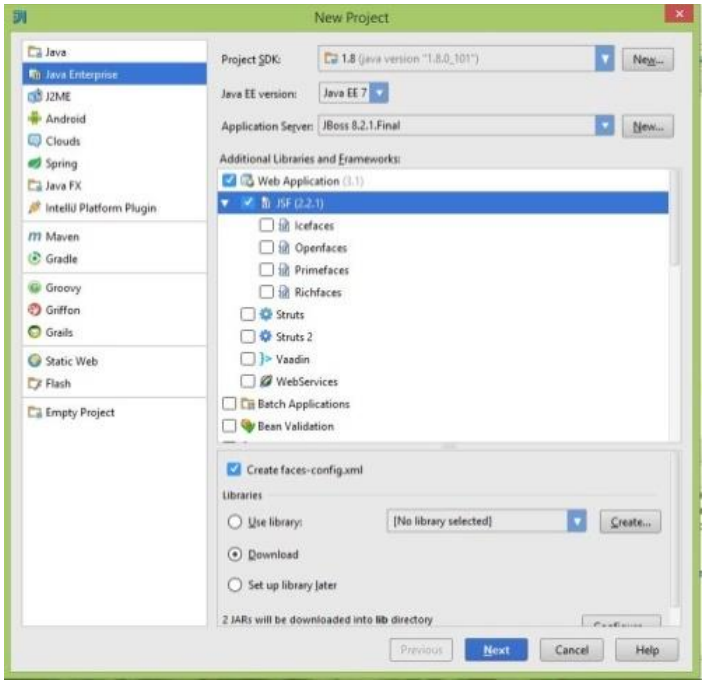
7.3. Installing PostgreSql

Section 7.3 of **Lab02** explains how to install PostgreSql.

8. Walkthrough Task

8.1. Walkthrough task 1

To create project and set **environment** and configure IDEA, refer to walkthrough task 1 of **Lab02.**
You just have to make one addition while making project, ***check JSF***.The rest of the procedure will be same.



8.2. Walkthrough task 2

Step	Description
1	Create a JSF Web application in IDEA.
2	Create a package com and then create a Java Class ***OrderBean.java*** in it.Add an annotation of ***@ManagedBean*** in OrderBean class.
3	Create a JSF page index.xhtml by going to ***Web>New>JSF/Facelet***
4	Compile and Run the Application.

OrderBean.Java

```
package com;

import javax.faces.bean.ManagedBean;
```

```
@ManagedBean(name = "obean")
public class OrderBean {
   private String name="Burger";
   private String quantity="5";
   private String amount="1000";

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getQuantity() {
      return quantity;
   }

   public void setQuantity(String quantity) {
      this.quantity = quantity;
   }

   public String getAmount() {
      return amount;
   }

   public void setAmount(String amount) {
      this.amount = amount;
   }

}
```

index.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:h="http://xmlns.jcp.org/jsf/html"
     xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
     xmlns:f="http://xmlns.jcp.org/jsf/core">

<h:head>

</h:head>
<h:body>
   <hr/>
   Food name is :
   <h:outputLabel value="#{obean.name}"></h:outputLabel><hr/>
   Food Quantity is :
   <h:outputLabel value="#{obean.quantity}"></h:outputLabel><hr/>
   Total Amount is :
   <h:outputLabel value="#{obean.amount}"></h:outputLabel><hr/>
</h:body>
</html>
```

If everything is fine with your application, this will produce following result:

9. Practice Tasks

9.1. Practice Task 1

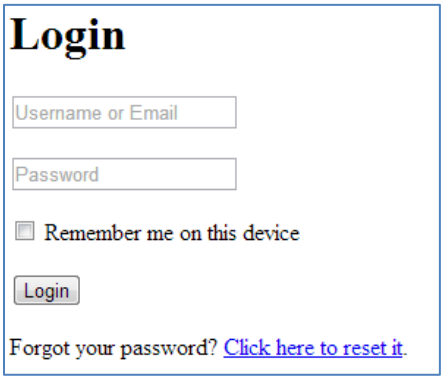
Create a managed bean and name it ***CarBean.Java***.It should have following properties and values

1. Carname
2. Model
3. Color
4. Id
5. Regno

Now create a JSF page having ***5 text fields*** for above properties and a ***Show Button.*** When the user clicks show button; the values of the text fields should be displayed on the output console of IDEA.

9.2. Practice Task 2

Create below Login form using JSF.



When user clicks the Login button; validate both text fields using managed bean that they should not be empty. If any of the fields is empty go to a page that displays **"Error"** else go to a page that displays

“Success”

Study Appendix A for the list of JSF tags.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.journaldev.com/6538/jsf-tutorial-for-beginners
https://www.tutorialspoint.com/jsf/jsf_basic_tags.htm
http://www.journaldev.com/6897/jsf-ui-component-tag-attributes-example-tutorial

Appendix A

S.N.	Tag & Description
1	**h:inputText** Renders a HTML input of type="text", text box.
2	**h:inputSecret** Renders a HTML input of type="password", text box.
3	**h:inputTextarea** Renders a HTML textarea field.
4	**h:inputHidden** Renders a HTML input of type="hidden".
5	**h:selectBooleanCheckbox** Renders a single HTML check box.
6	**h:selectManyCheckbox** Renders a group of HTML check boxes.
7	**h:selectOneRadio** Renders a single HTML radio button.
8	**h:selectOneListbox** Renders a HTML single list box.
9	**h:selectManyListbox** Renders a HTML multiple list box.
10	**h:selectOneMenu**

	Renders a HTML combo box.
11	**h:outputText** Renders a HTML text.
12	**h:outputFormat** Renders a HTML text. It accepts parameters.
13	**h:graphicImage** Renders an image.
14	**h:outputStylesheet** Includes a CSS style sheet in HTML output.
15	**h:outputScript** Includes a script in HTML output.
16	**h:commandButton** Renders a HTML input of type="submit" button.
17	**h:Link** Renders a HTML anchor.
18	**h:commandLink** Renders a HTML anchor.
19	**h:outputLink** Renders a HTML anchor.
20	**h:panelGrid** Renders an HTML Table in form of grid.
21	**h:message** Renders message for a JSF UI Component.
22	**h:messages** Renders all message for JSF UI Components.
23	**f:param** Pass parameters to JSF UI Component.
24	**f:attribute** Pass attribute to a JSF UI Component.
25	**f:setPropertyActionListener** Sets value of a managed bean's property

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 08

Java Server Faces-Part B

Lab 08: Java Server Faces-Part B

1. Introduction

Java Server Faces (JSF) technology is a front-end framework which makes the creation of user interface components easier by reusing the UI components. JSF is designed based on the **Model View Controller** pattern (MVC) which segregates the presentation, controller and the business logic.

2. Relevant Lecture Reading

http://www.tutorialspoint.com/jsf/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	10 min	10 min
8.1	Walkthrough Tasks	65 min	65min
9	Practice Tasks	As per time specified for each task	45 min
10	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

1. Backing Beans in JSF.
2. JDBC integration

5. Concept Map

5.1. Difference between Managed Beans and Backing Beans

A **Managed Bean** is a POJO that is constructed by JSF on-demand. It is defined by faces-config.xml (or whatever override name(s) you configure into web.xml).

A **Backing Bean** is a POJO that is referenced by one or more JSF View Definitions. The relationship between Views and Backing Beans is not 1-to-1, it's many-to-many. A single View can (and often does) reference more than one backing bean, and a single backing bean may be referenced by more than one View. Backing Bean is any bean that is bound with JSF UI. While Managed bean is any bean.

Example of Backing Bean

```
public class Person {
   //variables
```

```
   private int id;
   private String name;
   private String address;
   private int salary;


   public int getId() {
      return id;
   }

   public String getName() {
      return name;
   }

   public String getAddress() {
      return address;
   }

   public int getSalary() {
      return salary;
   }

   public void setId() {
      this.id = id;
   }

   public void setName() {
      this.name = name;
   }

   public void setAddress() {
      this.address = address;
   }

   public void setSalary() {
      this.salary = salary;
   }
}
```

5.2. JDBC Requirements

Database requirements

S.N.	Software & Description
1	**X**AMP and Apache
2	**MYSQL**

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

6.1. Homework 1

Install all the tools and configure the environment before coming to Lab if you have not done it already.

6.2. Homework 2

Create a Backing Bean class of Animals.

7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. Xamp (MySql) and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to Section 7 of **Lab02** in order to setup environment and project.

7.3. Installing MySql (XAMP)

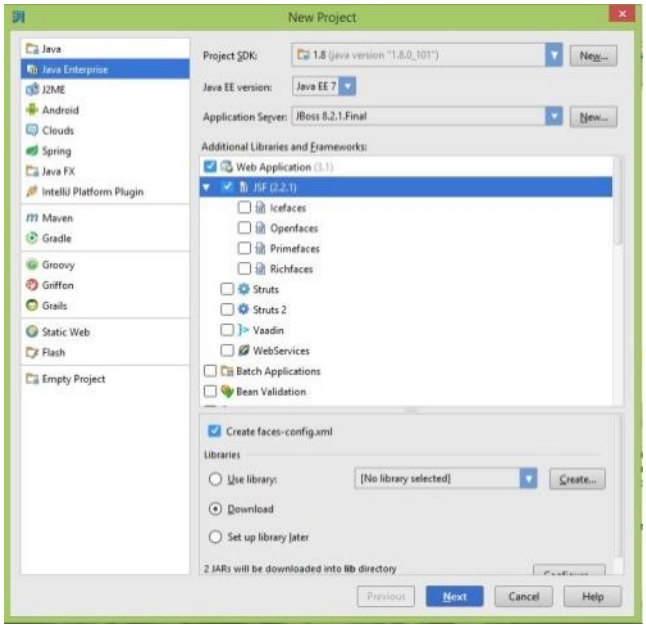
Lab03 explains how to install MySql using XAMP.

8. Walkthrough Task

8.1. Walkthrough task 1

To create project and set **environment** and configure IDEA, refer to walkthrough task 1 of **Lab02.**

You just have to make one addition while making project, ***check JSF***.The rest of the procedure will be same.



8.2. Walkthrough task 2

Save Details of Person in database using JSF.

1. Create a JSF application by the steps described in Walkthrough Task 1
2. Create a database in phpMyAdmin(XAMP) and name it as *persons*
3. Create a table named as *persondetails* and create below columns in it.
   - pname
   - pemail
   - pphone

4. Create a package *com* in *src* folder and create a backing bean *Person.Java* in it.In should have default constructor, getters and setters of the properties.

```
package com;

public class Person {
   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getEmail() {
      return email;
   }

   public void setEmail(String email) {
      this.email = email;
   }

   public String getPhone() {
      return phone;
   }

   public void setPhone(String phone) {
      this.phone = phone;
   }

   String name;
   String email;
   String phone;

}
```

5. Create a managed Bean in the same package and name it as PersonMBean.java.Change Database name,Table name,columns name and password of your database according to your requirements in *saveToDB()* function. Also create a class named DbConnection with insert function.

```
package com;

import javax.annotation.PostConstruct;
import javax.faces.bean.ManagedBean;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

```
@ManagedBean(name = "personBean")
public class PersonMBean {
   public Person getP() {
      return p;
   }

   public void setP(Person p) {
      this.p = p;
   }

   Person p;
   DbConnection db = new DbConnection();

   public PersonMBean()
   {

   }

   @PostConstruct
   public void init()
   {
      p=new Person();
   }

   public void SaveToDB()
{
            db.insertRecord(p.name, p.email. p. phone)

   }
}
```

DbConnection class code

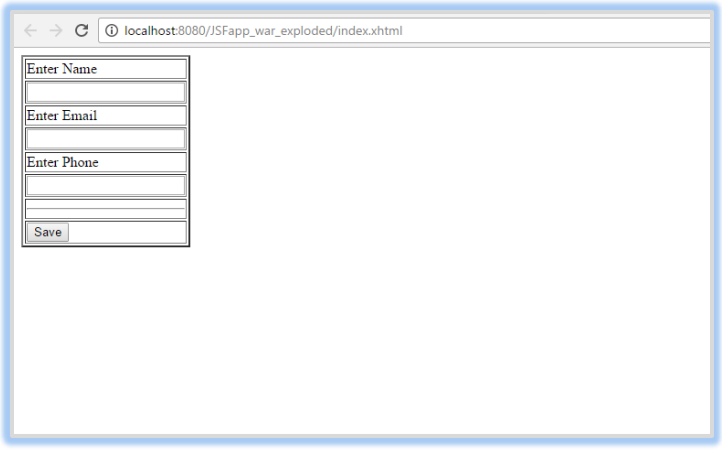
```
package com;

import java.sql.*;

public class DbConnection {
   private String dbURL = "jdbc:mysql://localhost:3306/test";
   private String username = "root";
   private String password = "";
   private Connection connection;
   public DbConnection(){
      try {
         Class.forName("com.mysql.jdbc.Driver");
         connection = DriverManager.getConnection(dbURL,username,password);
         if(connection!=null){
            System.out.println("Success");
         }
      }catch (ClassNotFoundException e){
         e.printStackTrace();
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }

   public void insertRecord(String pname,String pemail ,String pphone){
      try {
         String sqlQuery = "INSERT INTO persondetails(pname,pemail,pphone) VALUES(?,?,?)";
         PreparedStatement preparedStatement = connection.prepareStatement(sqlQuery);
         preparedStatement.setString(1, pname);
         preparedStatement.setString(2, pemail);
         preparedStatement.setString(3, pphone);

         int noOfRowsInserted = preparedStatement.executeUpdate();
         if(noOfRowsInserted>0){
            System.out.println(noOfRowsInserted + " rows inserted!");
         }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

}
```

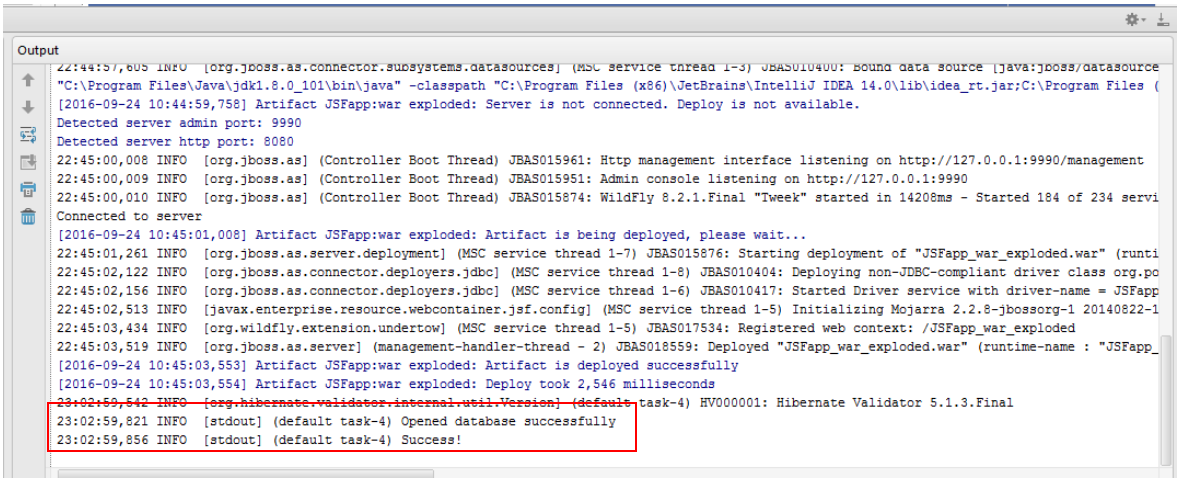
6.Create a JSF page ***index.xhtml***.Observe the use of ***panelGrid***.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:f="http://xmlns.jcp.org/jsf/core">

<h:head>

</h:head>
<h:body>
   <h:form>
   <h:panelGrid border="2">
      <h:outputLabel value="Enter Name" for="txtname"></h:outputLabel>
      <h:inputText id="txtname" value="#{personBean.p.name}"></h:inputText>

      <h:outputLabel value="Enter Email" for="txtemail"></h:outputLabel>
      <h:inputText id="txtemail" value="#{personBean.p.email}"></h:inputText>

      <h:outputLabel value="Enter Phone" for="txtphone"></h:outputLabel>
      <h:inputText id="txtphone" value="#{personBean.p.phone}"></h:inputText>

      <hr></hr>
      <h:commandButton value="Save" action="#{personBean.SaveToDB}"></h:commandButton>
   </h:panelGrid>
   </h:form>
</h:body>
</html>
```

7. Finally Compile and run the application; if you followed all the steps and everything is fine with your configuration you will be displayed with below output.



9. Enter values; click submit button and observe output console. It should show something like this.

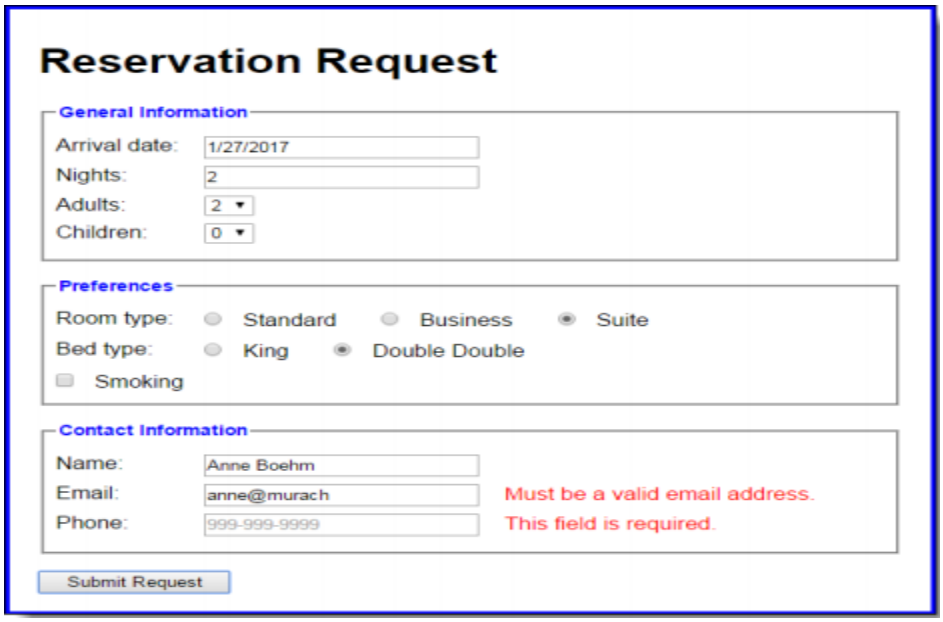


10. Go and check your database whether the values are being stored or not.

9. Practice Tasks

9.1. Practice Task 1

Create below JSF form using relevant UI components (see appendix A for help) and integrate it with database using backing bean and managed bean. When user clicks ***Submit Request;*** all the data should be saved into a database ***Reservations*** having table ***ReservationsRequests*** in relevant columns.



10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

https://www.tutorialspoint.com/jsf/jsf_jdbc_integration.htm
https://jdbc.postgresql.org/documentation/94/

Appendix A

S.N.	Tag & Description
1	**h:inputText** Renders a HTML input of type="text", text box.
2	**h:inputSecret** Renders a HTML input of type="password", text box.
3	**h:inputTextarea** Renders a HTML textarea field.
4	**h:inputHidden** Renders a HTML input of type="hidden".
5	**h:selectBooleanCheckbox** Renders a single HTML check box.
6	**h:selectManyCheckbox** Renders a group of HTML check boxes.
7	**h:selectOneRadio** Renders a single HTML radio button.
8	**h:selectOneListbox** Renders a HTML single list box.
9	**h:selectManyListbox** Renders a HTML multiple list box.
10	**h:selectOneMenu** Renders a HTML combo box.
11	**h:outputText** Renders a HTML text.
12	**h:outputFormat** Renders a HTML text. It accepts parameters.
13	**h:graphicImage** Renders an image.

14	**h:outputStylesheet** Includes a CSS style sheet in HTML output.
15	**h:outputScript** Includes a script in HTML output.
16	**h:commandButton** Renders a HTML input of type="submit" button.
17	**h:Link** Renders a HTML anchor.
18	**h:commandLink** Renders a HTML anchor.
19	**h:outputLink** Renders a HTML anchor.
20	**h:panelGrid** Renders an HTML Table in form of grid.
21	**h:message** Renders message for a JSF UI Component.
22	**h:messages** Renders all message for JSF UI Components.
23	**f:param** Pass parameters to JSF UI Component.
24	**f:attribute** Pass attribute to a JSF UI Component.
25	**f:setPropertyActionListener** Sets value of a managed bean's property

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 9

Java Server Faces-Part C

Lab 09: Java Server Faces-Part C

1. Introduction

Java Server Faces (JSF) technology is a front end framework which makes the creation of user interface components easier by reusing the UI components. JSF is designed based on the **Model View Controller** pattern (MVC) which segregates the presentation, controller and the business logic.

2. Relevant Lecture Reading

http://www.tutorialspoint.com/jsf/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
6.2	Setting up to use HTML	10 min	10 min
6.3	Walkthrough Tasks	65 min	65min
7	Practice Tasks	As per time specified for each task	45 min
8	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

1. Data Tables

5. Concept Map

5.1. DataTables

JSF provides a rich control named DataTable to render and format html tables.

1. DataTable can iterate over collection or array of values to display data.
2. DataTable provides attributes to modify its data in easy way.

5.2. Operations

Following are important *DataTable* operations in JSF 2.0:

S.N.	Tag & Description
1	**Display DataTable** How to display a datatable
2	**Add data** add a new row in a datatable
3	**Edit data** edit a row in a datatable
4	**Delete data** delete a row in datatable
5	**Using DataModel** Use DataModel to display row numbers in a datatable

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

6.1. Home Work 1

Install all the tools and configure the environment before coming to Lab if you have not done it already.

7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. Xamp (MySql) and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to Section 7 of **Lab02** in order to setup environment and project.

7.3. Installing MySql (XAMP)

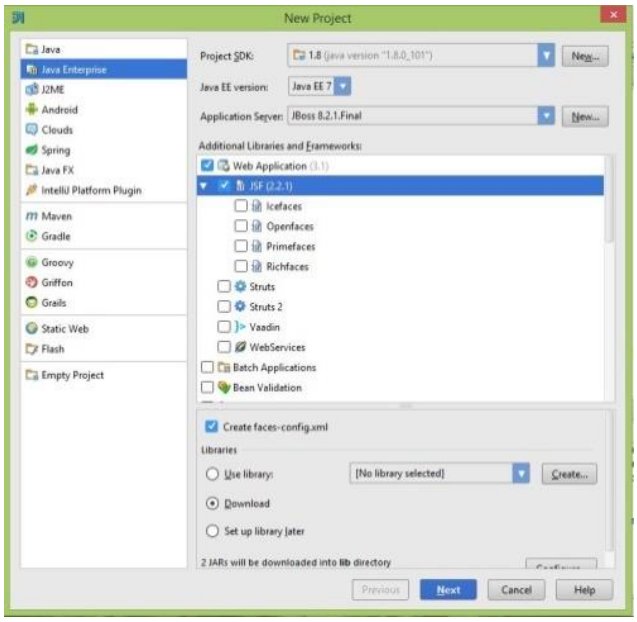
Lab03 explains how to install MySql using XAMP.

8. Walkthrough Task

8.1. Walkthrough Task 1

To create project and set **environment** and configure IDEA, refer to walkthrough task 1 of **Lab02.**
You just have to make one addition while making project, ***check JSF***.The rest of the procedure will be

same.



8.2. Walkthrough task 2

Let's extend walkthrough task 2 of Lab 09 to Create a ***dataTable*** that ***deletes and displays*** those database values.

1. Add a new function ***getAllPersons ()*** in ***PersonMBean.Java***

```
public List<Person> getPersonList() throws SQLException {
   List<Person> list=new ArrayList<Person>();
   PreparedStatement pstmt=c.prepareStatement("select * from persondetails");
   ResultSet rs=pstmt.executeQuery();

   while (rs.next())
   {
      Person person=new Person();
      person.name=rs.getString("pname");
      person.email=rs.getString("pemail");
      person.phone=rs.getString("pphone");

      list.add(person);
   }
   return list;
}
```

2. Create another function **DeletePerson(Person p)** in **PersonMBean.java**

```
public String DeletePerson(Person p) throws SQLException {
   String name=p.name;
   String email=p.email;
   String phone=p.phone;

   String sql="delete from persondetails where pname='"+name+"'"+"AND pemail='"+email+"'";
   Statement stmt=c.createStatement();
   stmt.executeUpdate(sql);
   list.remove(p);
   System.out.print("Success!");
```

```
    return null;
}
```

2. Add below code inside ***form*** tag after **panelGrid** in ***index.xhtml***.

```
<hr></hr>
<h1>List of Persons</h1>
<h:dataTable value="#{personBean.getPersonList()}" var="usernames" border="4" >
    <h:column>
        <f:facet name="header">
            Username
        </f:facet>
        #{usernames.name}
    </h:column>

    <h:column>
        <f:facet name="header">
            Email
        </f:facet>
        #{usernames.email}
    </h:column>

    <h:column>
        <f:facet name="header">
            Password
        </f:facet>
        #{usernames.phone}
    </h:column>

    <h:column>
        <f:facet name="header">Action</f:facet>
        <h:commandLink value="Delete" action="#{personBean.DeletePerson(usernames)}" />
    </h:column>
</h:dataTable>
```

Compile and run the application and you will get all the data in tabular format that was saved by you into database.

9. Practice Tasks

9.1. Practice Task 1

Extend Practice Task 1 of Lab 09 to display and deletes database values in ***dataTable***.

Note: Making Editable dataTable carries Bonus marks.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

https://www.tutorialspoint.com/jsf/jsf_jdbc_integration.htm

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computing

Lab Manual for Enterprise Application Development

Lab 10

GitHub Integration with IntelliJ Idea & use of Postman

Lab 10: GitHub Integration with IntelliJ Idea & use of Postman

1. Introduction

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. To understand GitHub, you must first have an understanding of Git. Git is an open-source version control system that was started by Linus Torvalds—the same person who created Linux.

So, Git is a version control system, but what does that mean? When developers create something (an app, for example), they make constant changes to the code, releasing new versions up to and after the first official (non-beta) release.

Version control systems keep these revisions straight, storing the modifications in a central repository. This allows developers to easily collaborate, as they can download a new version of the software, make changes, and upload the newest revision. Every developer can see these new changes, download them, and contribute.

Postman is a popular API client that makes it easy for developers to create, share, test and document APIs. This is done by allowing users to create and save simple and complex HTTP/s requests, as well as read their responses. The result - more efficient and less tedious work.

Postman is very convenient when it comes to executing APIs. Once you've entered and saved them, you can simply use them over and over again, without having to remember the exact endpoint, headers, API keys, etc.

2. Relevant Lecture Reading

https://guides.github.com/
https://learning.postman.com/docs/getting-started/introduction/

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5	Concept Map	30 min	20 min
8.1	Walkthrough Task	40 mins	70 min
8.2	Walkthrough Task	30 mins each	70 min
9	Practice Tasks	As per time specified for each task	60 min
10	Evaluation Task (Unseen)	30 min	30 min

4. Objective of the Experiment

To understand the fundamental concepts of version control system using git, github & how to use github with IntelliJ Idea. A brief practice of postman and how to consume API's using postman tool.

5. Concept Map

5.1. Use case for GitHub

Let's consider the case of Decathlon, the world's largest sporting goods retail brand. The company has 1600 stores in 52 countries, with nearly 100,000 employees.

Regardless of size, every company inevitably experiences challenging issues. Decathlon's problems could be best summed up as:

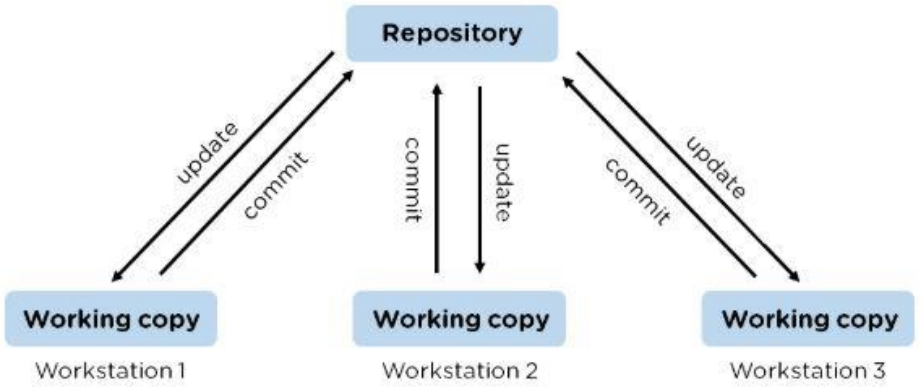
1. How will the company maintain workflow visibility and avoid redundancies in such a large workforce? Will all developers email each other for code sharing?
2. How will the company hire developers for so many locations?

GitHub to the rescue! GitHub is not only an affordable resource but also features a great open source community. Since it is a cloud-based tool, the code is conveniently visible across the entire client organization, facilitating contributions by every participant.

Git's VCS is the most significant feature available for tackling these two problems. So let's expand our knowledge of GitHub by taking a closer look at the Git version control system.

5.2. Git Version Control System

The Git version control system, as the name suggests, is a system that records all the modifications made to a file or set of data, so that a specific version may be called up later if needed. The system makes sure that all the team members are working on the file's latest version, and everyone can work simultaneously on the same project.



5.3. What is Git & GitHub?

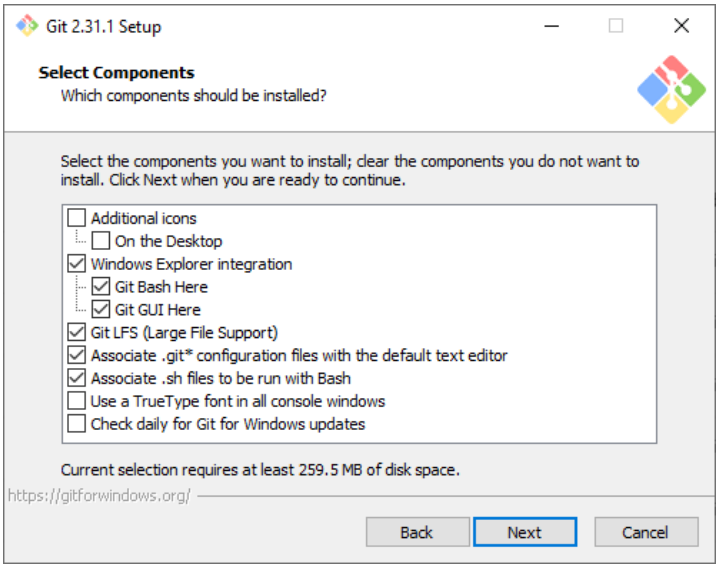
Git is a version control system used for tracking changes in computer files. This function makes it an extremely popular utility in the programming world.

Git is used to coordinating the workflow among the members of a project team and track their progress over time. It benefits both programmers and non-technical users by keeping track of their project files. Git allows multiple users to work together without disrupting each other's work.

GitHub is a Git repository hosting service that provides a web-based graphical interface. You can find source code in many different programming languages and keep track of all changes.
Commit in git creates a commit, which is like a snapshot of your repository. These commits are snapshots of your entire repository at specific times. You should make new commits often, based around logical units of change. Over time, commits should tell a story of the history of your repository and how it came to be the way that it currently is.
After you make and commit changes locally, you can share them with the remote repository using **git push**. Pushing changes to the remote makes your commits accessible to others who you may be collaborating with.
git pull updates your current local working branch, and all of the remote tracking branches. It's a good idea to run git pull regularly on the branches you are working on locally.
Without git pull, (or the effect of it,) your local branch wouldn't have any of the updates that are present on the remote.

5.4. Git Installation

Download Git from software folder that was shared earlier with you. Double click on setup to start installation. Simply click on Next, Next to install Git on your system without changing any option.



6. Homework before Lab

It is assumed that the user is familiar with API creation with .NET and knows all 4 types of API request.
Revise what is API and its request types. This lab is designed specially to give hands on experience with git/github and api testing using Postman tool.

7. Tools and Procedure.

7.1. Tools

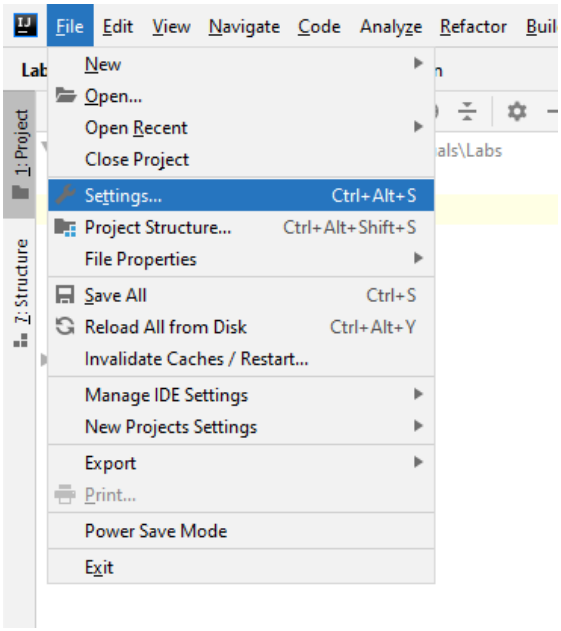
- IntelliJ Idea.
- Git & Github
- Postman tool

8. Walkthrough Tasks

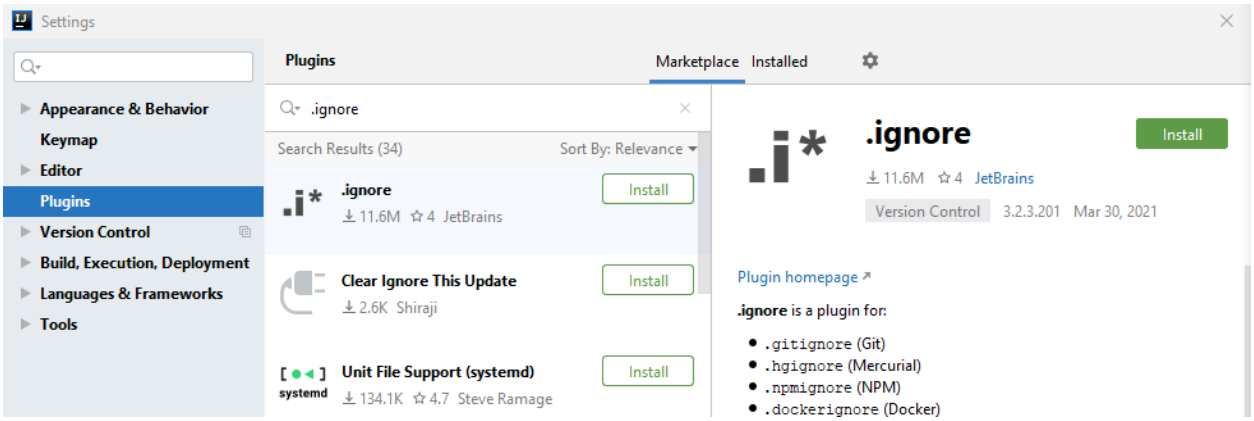
This section is designed such a way that you can complete the following tasks independently. However if there is any ambiguity you can refer it to the lab instructor.
Just revise the Concept map section of the manual so that you can perform the practice tasks easily.

8.1. GitHub Integration with IntelliJ Idea

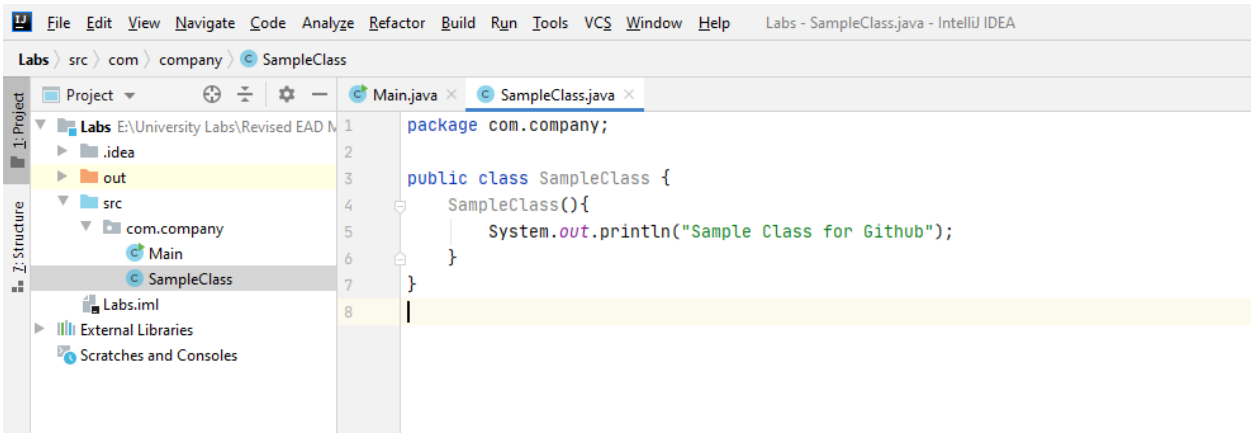
Step 1: Install .ignore plugin in IntelliJ to ignore unnecessary files to include in git. It is only the first time process. Click on File and select Settings option.



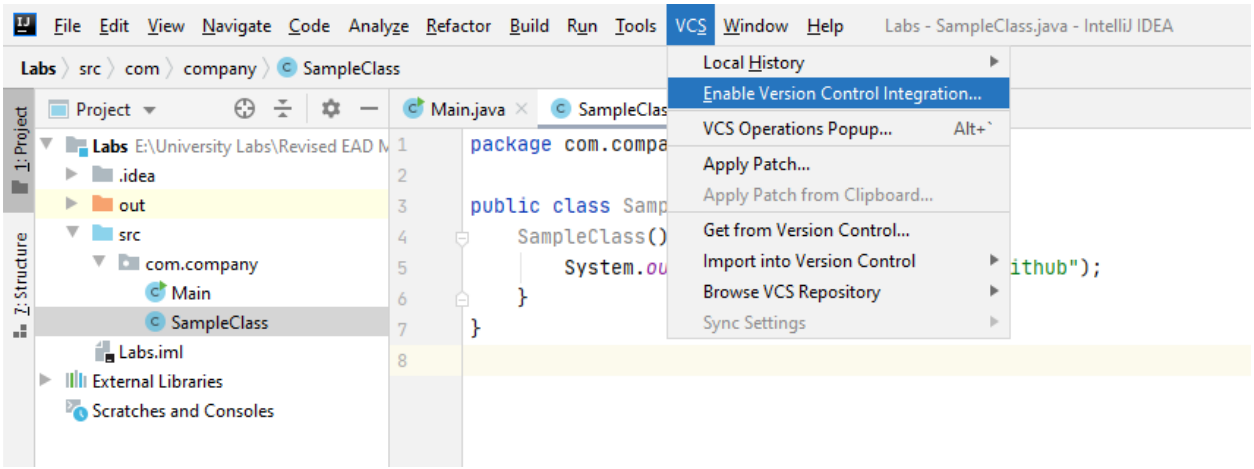
Then click on plugins and search for .ignore plugin and install it.



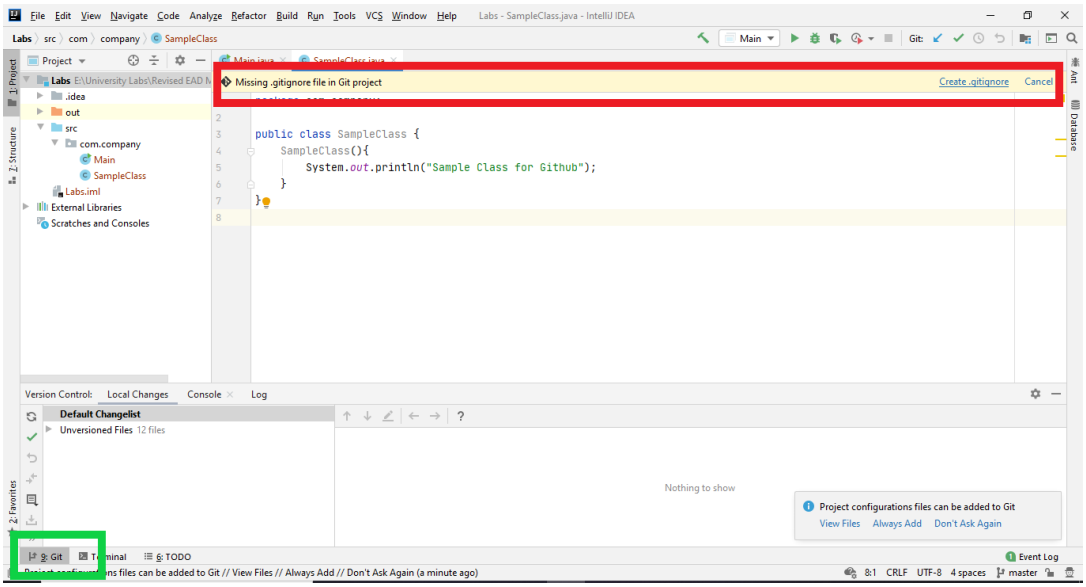
Step 2: Now from this step onwards all steps are for new project we create. I have created sample project and shared with you. In this project we just have 2 java files.



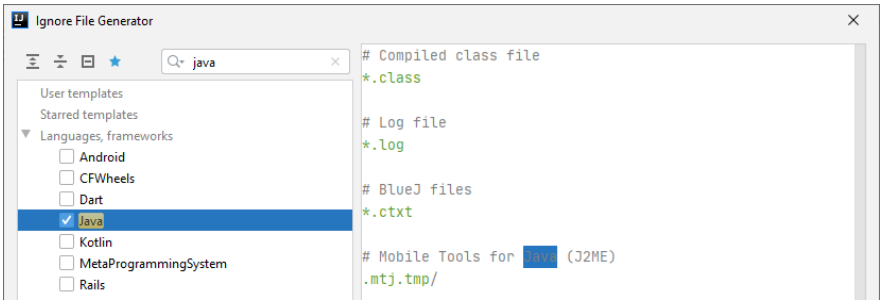
To save this project in local repository we have to enable Version Control Integration. Click on VCS and click **Enable Version Control Integration.**



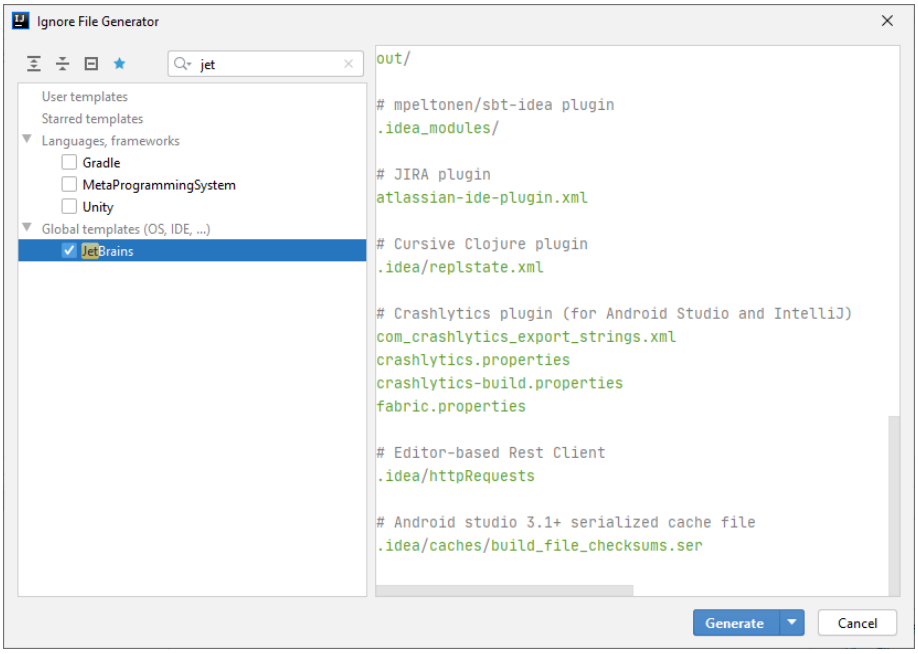
Make sure Git is selected in dialog box and hit Ok. After pressing ok IntellIj will tell you to create .gitignore file as shown in red box click on that and follow steps performed below. Furthermore, you can also see that you got an extra panel in bottom left as shown in green box.



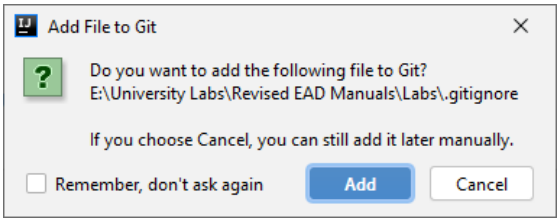
When you click on create .gitignore file follow these steps. Search for java and select that.



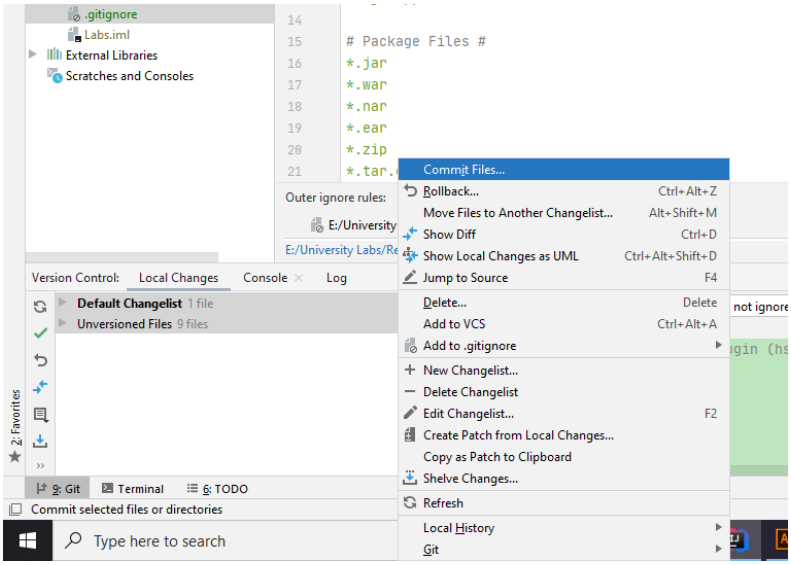
Then search for jetbrains and also check that. Then click on generate button to generate .gitignore file.



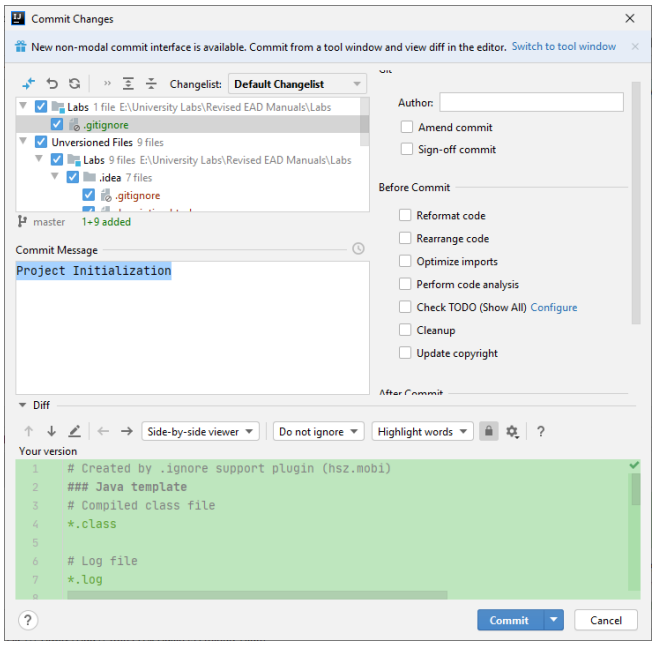
Click on Add to git to include this file in git.



Step 3: Our third step is to commit these files in local git repository. To do that click on left bottom option Git and select both **Default Changelist and Unversioned Files** and right click to commit Files. This step will add these files in local git repository.

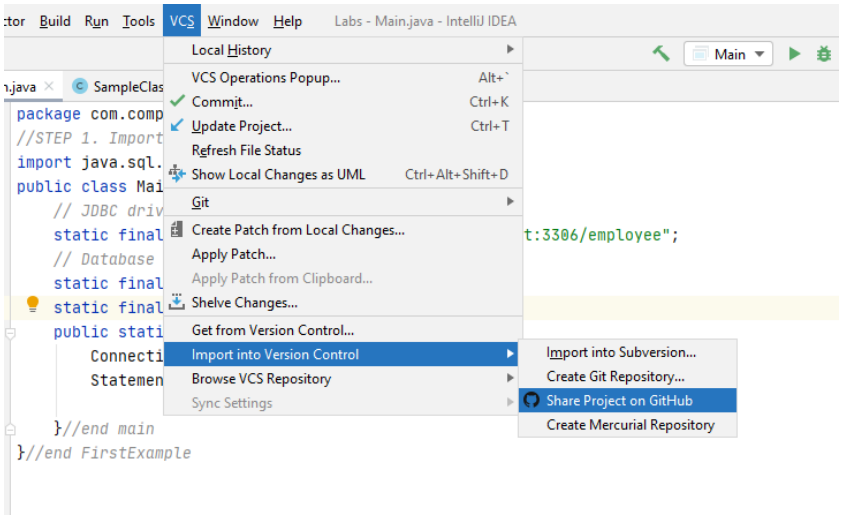


Give this version of your project a commit message. Basically commit message is useful to analyze what exactly done in this committed version.

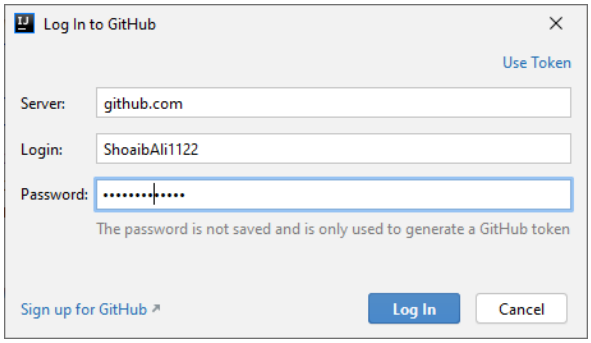


Step 4: Our Next Step is to upload this project to github. To do that first login to your github account. Simply visit https://github.com/join to create new account if not created yet.

Step 5: This step is only for first time when you first upload your project on github.

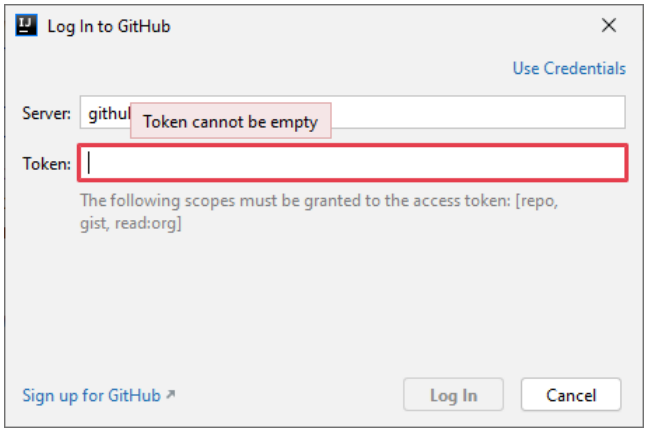


Enter GitHub Credentials.

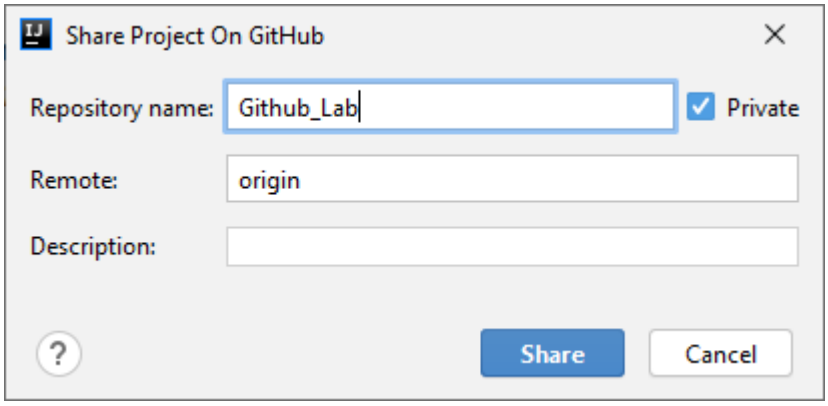


If it doesn't work go to https://github.com/settings/tokens/new and generate new token by selecting all checkbox and click on Generate token.

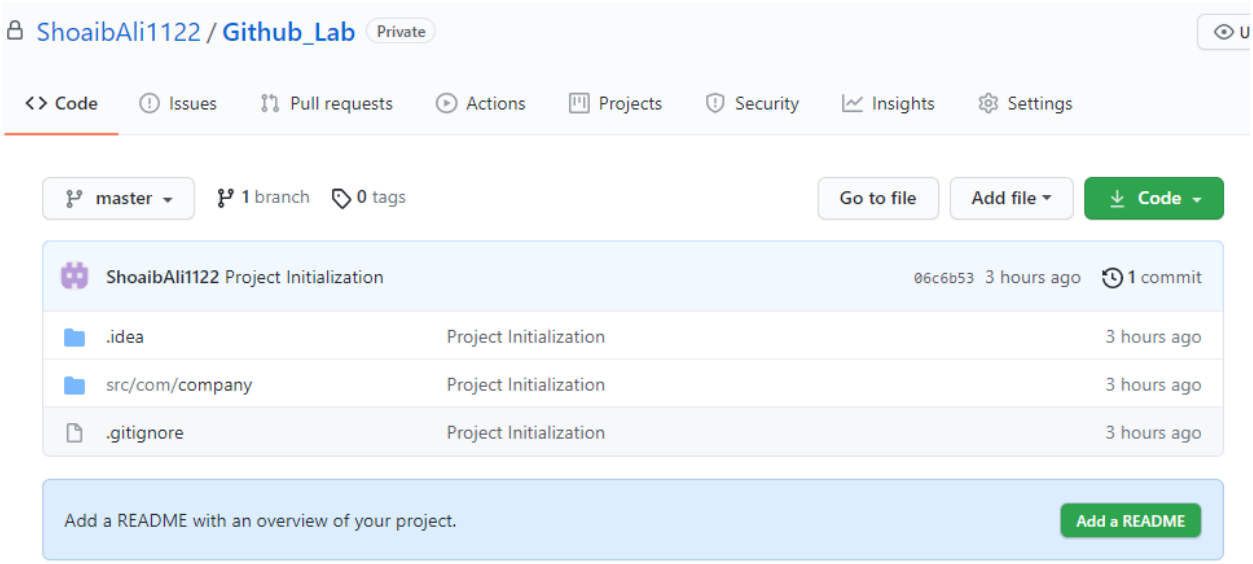
Copy that generated token and paste it by selecting **Use Token** option present at top right as shown in above picture.



Give your repository on github a name, make it private so that no one else on internet can access your code.



After your project is successfully shared check it on github by visiting that repository
This process is for first time you shared your code on github. Next step is to push your project when your code changes.



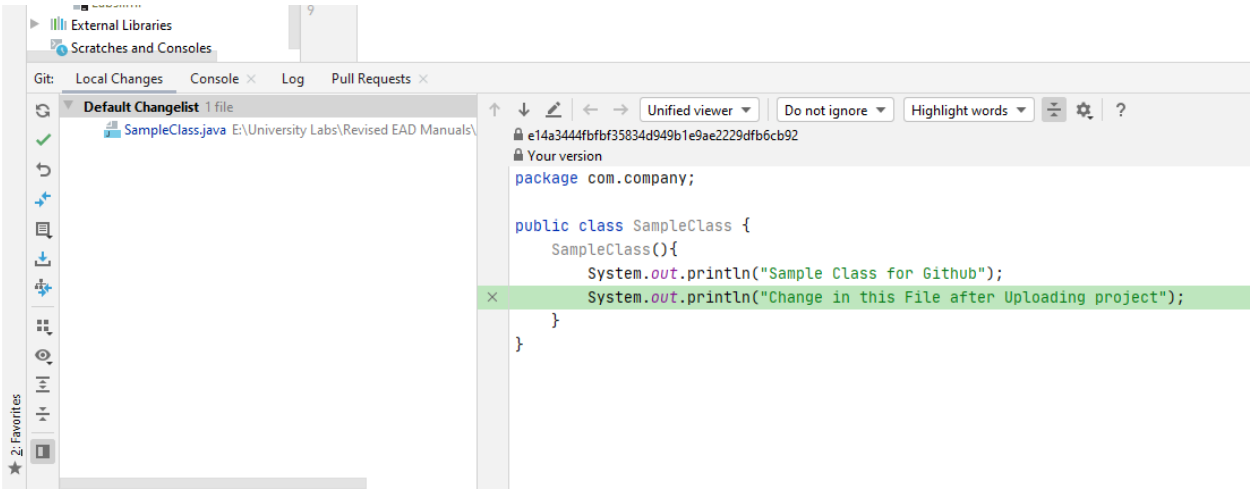
So there is 2 step process.

1. Commit changes to local repository (git).
2. Push that changes to remote repository (GitHub).

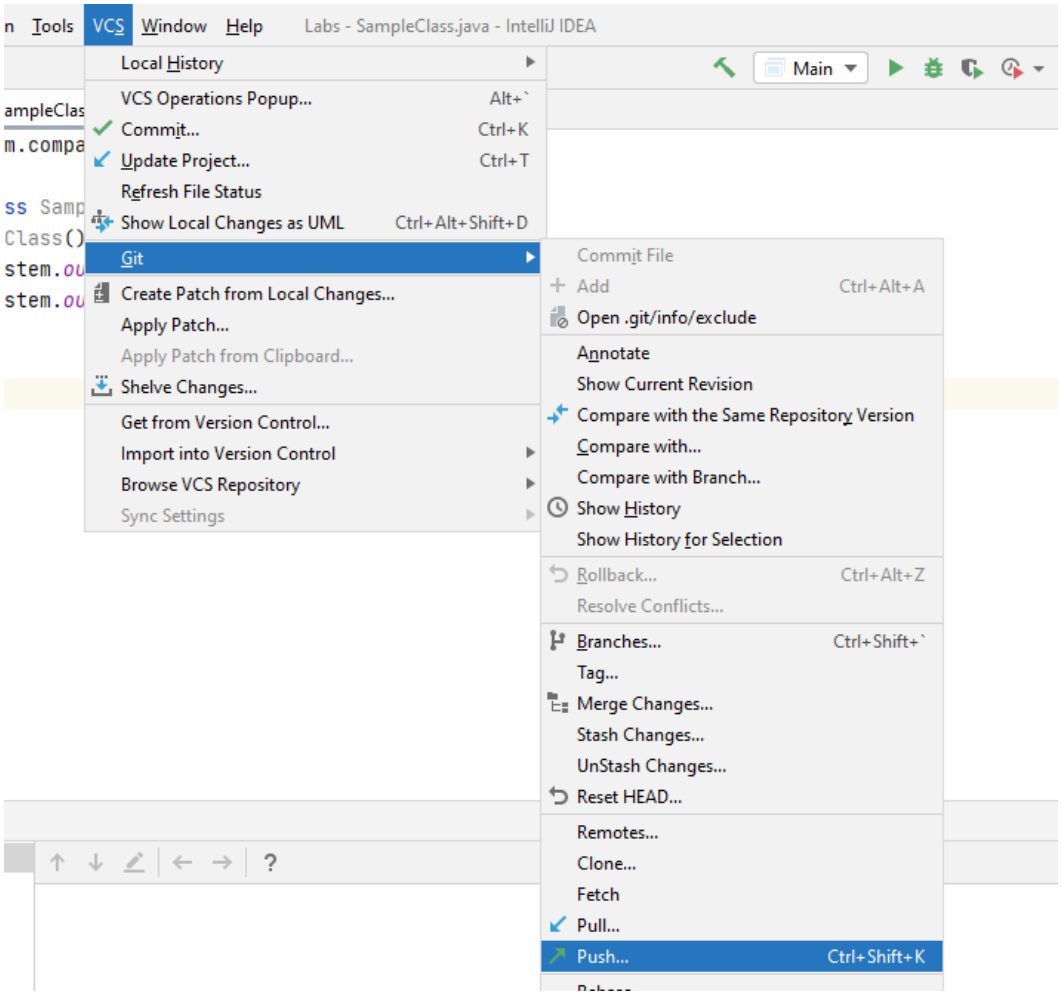
So I am changing **SampleClass.java.**

```
public class SampleClass {
    SampleClass(){
        System.out.println("Sample Class for Github");
        System.out.println("Change in this File after Uploading project");
    }
}
```

Now I have to commit this change. This panel will tell you what exactly is changed in code.



Do same procedure to commit files and change commit message.
After committing new files now we don't have to share this project on github but we will push new changes to github repository every time.

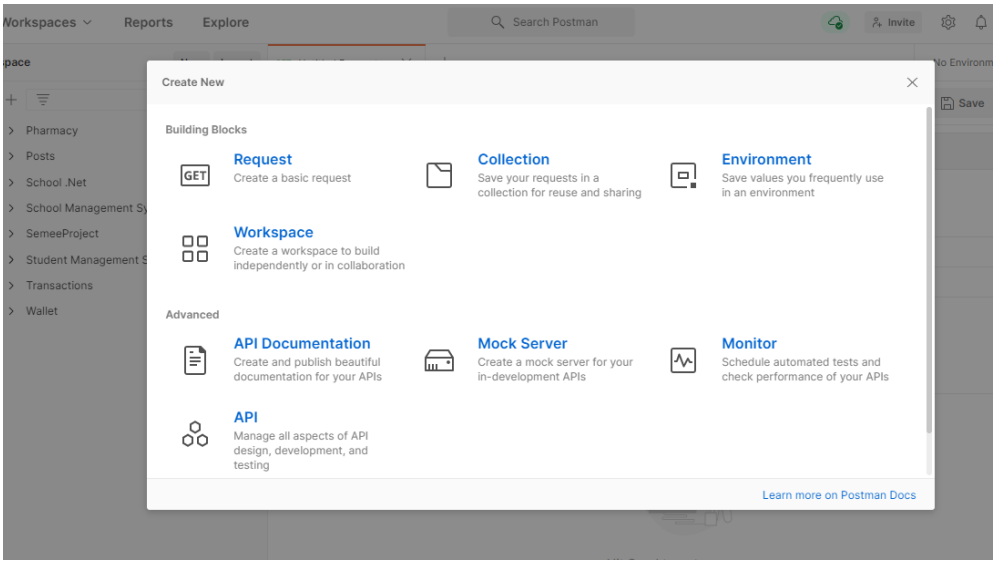


8.2. Test API using Postman Tool [Expected time = 30 mins]

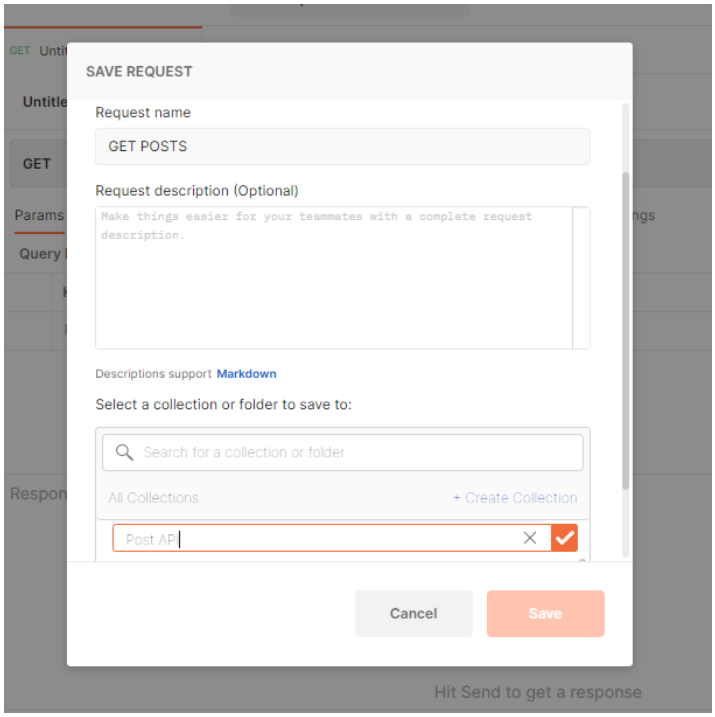
You can download Postman Tool from here https://www.postman.com/downloads/ or can use the web version. Both are same & for this task I am using web version.
API URL: https://jsonplaceholder.typicode.com/posts

GET Request :

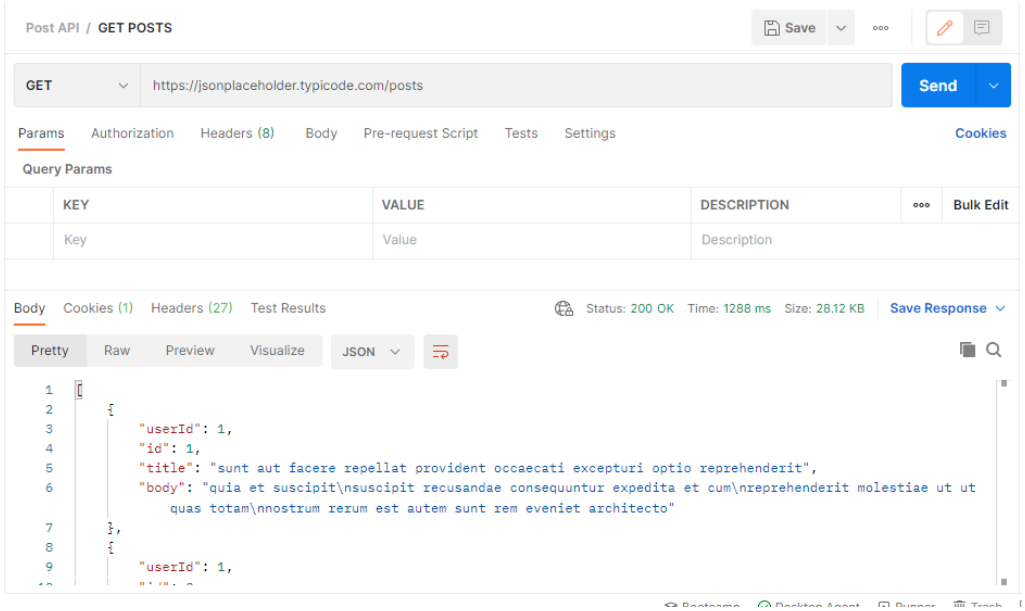
Create New Request by clicking New button present at top left panel.



Give Request name and create new collection (It is like folder) to organize api testing in better way. Then click save button.

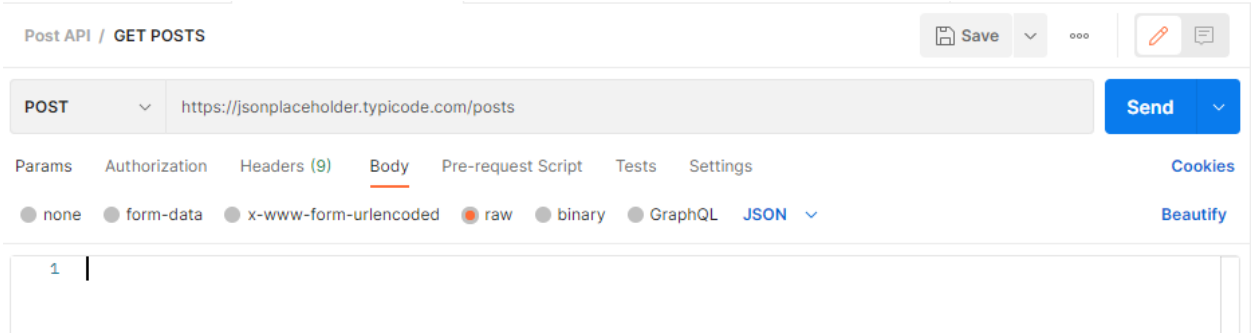


Select GET from dropdown and paste API url on right side then click on Send button to get data from Api endpoint.

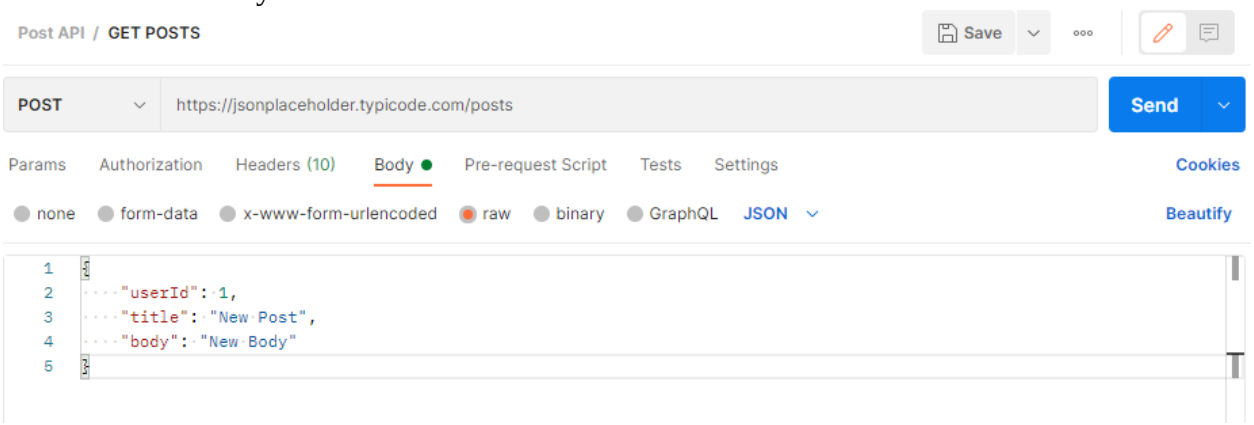


POST Request:

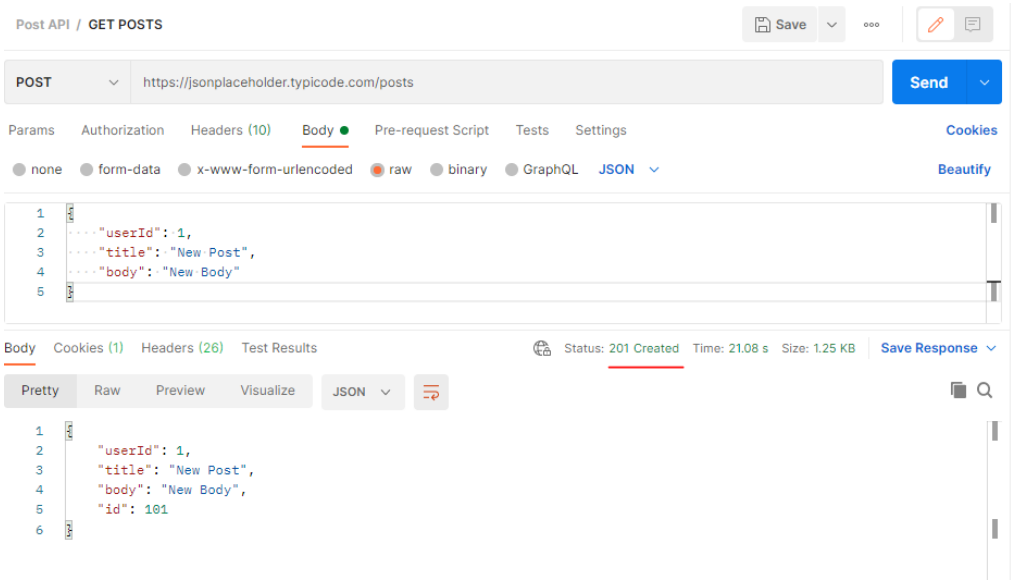
As in post request we basically want to save new data in database. So for this purpose we must provide body of request. Select body option and then click raw and select json from dropdown.



Now provide new post object and click on send button to save data through this api endpoint. Copy one json data of post and paste above. Cut id column because it is primary key so api will create new id automatically.



Now click on send button and see if status returned is **201 created.** Please note that this api is dummy and for testing purpose so no actual data will be save in that.



Practice Tasks

9.1. Practice Task 1 [Expected time = 30 mins]

Upload last lab task on GitHub and add your instructor as collaborator so that instructor can access your code.

9.2. Practice Task 2 [Expected time = 30 mins]

Complete Walkthrough task 2 by doing PUT and DELETE request to same endpoint. Documentation about Api can be find at: https://jsonplaceholder.typicode.com/guide/ Test this Api https://jsonplaceholder.typicode.com/comments URL by testing post, get, put, delete request.

10. Evaluation Task (Unseen) [Expected time = 30 mins]

Evaluation task will be given at the time of lab.

11. Further Reading

https://guides.github.com/
https://lab.github.com/

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 11

Primefaces

Lab 11: Primefaces

1. Introduction

PrimeFaces is an open source user interface (UI) component library for Java Server Faces (JSF) based applications, created by **PrimeTek, Turkey.**

PrimeFaces is a JSF component library. It was one of the first which supports JSF 2.0 from top to bottom. It contains large set of rich components which utilizes jQuery and jQuery UI under the covers. There is also a specific set of rich components for mobile devices.

2. Relevant Lecture Reading

http://www.journaldev.com/5516/primefaces-tutorial-with-example-projects

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
6.2	Setting up to use HTML	10 min	10 min
6.3	Walkthrough Tasks	65 min	65min
7	Practice Tasks	As per time specified for each task	45 min
8	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

1. PrimeFaces Basic Tags.
2. PrimeFaces DB Integration.
3. Primefaces AJAX

5. Concept Map

5.1. Adding PrimeFaces to Your Project

To add PrimeFaces to existing project; Download PrimeFaces Community version from http://www.primefaces.org/downloads and add it to your project by going **File>Project Structure>Libraries>**+

Add PrimeFaces Library to the artifact otherwise components won't show on webpage.

Or Check PrimeFaces Box while Creating New Project.

5.2. PrimeFaces Ajax

By using AJAX you can:

1. Update a web page without reloading the page
2. Request data from a server - after the page has loaded
3. Receive data from a server - after the page has loaded
4. Send data to a server - in the background

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

6.1. Home Work 1

Install all the tools and configure the PrimeFaces environment before coming to Lab.

7. Tools and Procedure.
7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. Xamp (MySql) and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to Section 7 of **Lab02** in order to setup environment and project.

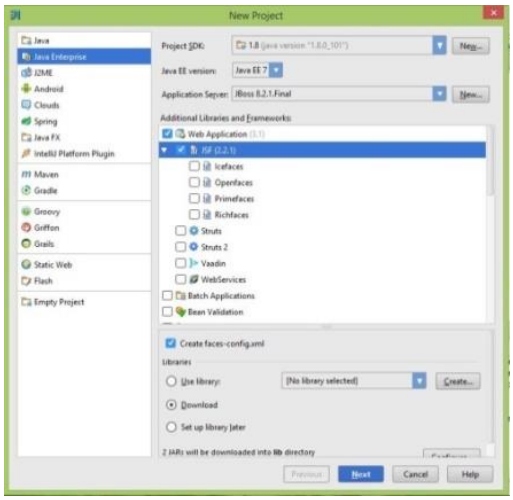
7.3. Installing MySql (XAMP)

Lab03 explains how to install MySql using XAMP.

8. Walkthrough Task

8.1. Walkthrough Task 1

To create project and set **environment** and configure IDEA, refer to walkthrough task 1 of **Lab02.**
You just have to make one addition while making project, ***check JSF and PrimeFaces***. The rest of the procedure will be same.



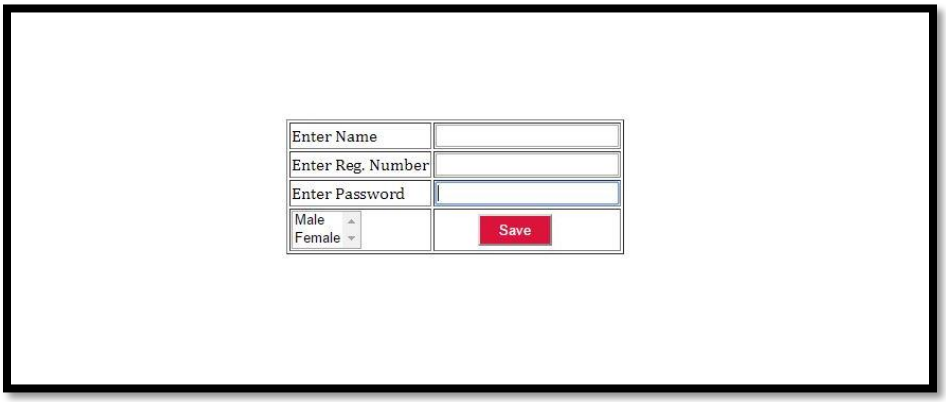
8.2. Walkthrough task 2

Creating PrimeFaces Form; saving its data in database and Displaying in Data Table.

Create a form ***primefaces.xhtml***

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
    xmlns:f="http://xmlns.jcp.org/jsf/core" xmlns:p="http://java.sun.com/jsf/html">

<h:head>

</h:head>
<h:body>
   <p:form>
   <p:panelGrid border="1" columns="2" style="margin: auto;margin-top: 200px;font-family: cambria;">


        <p:outputLabel value="Enter Name" for="txt1"></p:outputLabel>
        <p:inputText id="txt1"></p:inputText>

        <p:outputLabel value="Enter Reg. Number" for="txt3"></p:outputLabel>
        <p:inputText id="txt3"></p:inputText>

        <p:outputLabel value="Enter Password" for="txt2"></p:outputLabel>
        <h:inputSecret id="txt2"></h:inputSecret>


        <p:selectOneListbox id="gender" value="">
            <f:selectItem itemLabel="Male" itemValue="1" />
            <f:selectItem itemLabel="Female" itemValue="2" />
        </p:selectOneListbox>
        <p:commandButton id="submitButton" value="Save" style="width: 70px;height: 30px;margin-left: 40px;background-color:
crimson;color: white;"></p:commandButton>
   </p:panelGrid>
   </p:form>
</h:body>


</html>
```

Run it and it will look something like this.



To save, display and delete data from ***dataTable;*** follow the same procedure that is described to you in previous Lab.You just have to put **primeFaces** tags in place of JSF tags; rest of the procedure is same.

8.3. Walkthrough task 3

This example demonstrates a simple but common usage of posting the form, updating the backend value and displaying the output with Ajax.

Create a JSF page and type below code into body section.

```
<h:form>
   <h:panelGrid columns="4" cellpadding="5">
      <h:outputLabel for="name" value="Name:" style="font-weight:bold" />
      <p:inputText id="name" value="#{basicView.text}" />
      <p:commandButton value="Submit" update="display" icon="ui-icon-check" />
      <h:outputText id="display" value="#{basicView.text}" />
   </h:panelGrid>
</h:form>
```

Create a ManagedBean and type following code into it.

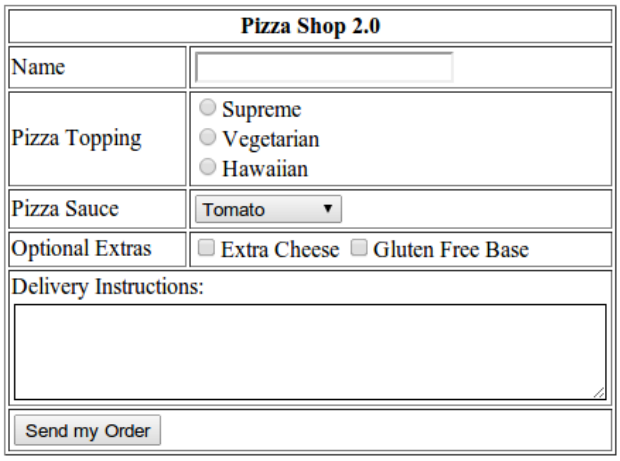
```
package com;

import javax.faces.bean.ManagedBean;

@ManagedBean
public class BasicView {

   private String text;

   public String getText() {
      return text;
   }
   public void setText(String text) {
      this.text = text;
   }
}
```

Compile and run the program.

9. Practice Tasks

9.1. Practice Task 1

Create below form using **primeFaces** and save its data into database using **ManagedBean, BackingBean.** After saving the data create a dataTable that displays and deletes the database values.



Note: Making Editable dataTable carries Bonus marks.

9.2. Practice Task 2

Find the reason for below exception and Explain the solution for it.

```
[2016-09-26 07:29:18,124] Artifact JSFapp:war exploded: Error during artifact deployment. See server log for details.
[2016-09-26 07:29:18,124] Artifact JSFapp:war exploded: java.lang.Exception: {"JBAS014671: Failed services" =>
{"Tomcat.undertow.deployment.default-server.default-host./JSFapp_war_exploded" => "org.Tomcat.msc.service.StartException in service
Tomcat.undertow.deployment.default-server.default-host./JSFapp_war_exploded: Failed to start service
    Caused by: java.lang.RuntimeException: java.lang.RuntimeException: com.sun.faces.config.ConfigurationException: The tag named inputFile
from namespace http://xmlns.jcp.org/jsf/html has a null handler-class defined
    Caused by: java.lang.RuntimeException: com.sun.faces.config.ConfigurationException: The tag named inputFile from namespace
http://xmlns.jcp.org/jsf/html has a null handler-class defined
    Caused by: com.sun.faces.config.ConfigurationException: The tag named inputFile from namespace http://xmlns.jcp.org/jsf/html has a null
handler-class defined"}}
```

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://www.primefaces.org/showcase/
https://github.com/primefaces/primefaces

Appendix A

S.N.	Tag & Description
1	**h:inputText** Renders a HTML input of type="text", text box.
2	**h:inputSecret** Renders a HTML input of type="password", text box.
3	**h:inputTextarea** Renders a HTML textarea field.
4	**h:inputHidden** Renders a HTML input of type="hidden".
5	**h:selectBooleanCheckbox** Renders a single HTML check box.
6	**h:selectManyCheckbox** Renders a group of HTML check boxes.
7	**h:selectOneRadio** Renders a single HTML radio button.
8	**h:selectOneListbox** Renders a HTML single list box.
9	**h:selectManyListbox** Renders a HTML multiple list box.
10	**h:selectOneMenu** Renders a HTML combo box.
11	**h:outputText** Renders a HTML text.
12	**h:outputFormat** Renders a HTML text. It accepts parameters.
13	**h:graphicImage** Renders an image.
14	**h:outputStylesheet** Includes a CSS style sheet in HTML output.
15	**h:outputScript** Includes a script in HTML output.

16	**h:commandButton** Renders a HTML input of type="submit" button.
17	**h:Link** Renders a HTML anchor.
18	**h:commandLink** Renders a HTML anchor.
19	**h:outputLink** Renders a HTML anchor.
20	**h:panelGrid** Renders an HTML Table in form of grid.
21	**h:message** Renders message for a JSF UI Component.
22	**h:messages** Renders all message for JSF UI Components.
23	**f:param** Pass parameters to JSF UI Component.
24	**f:attribute** Pass attribute to a JSF UI Component.
25	**f:setPropertyActionListener** Sets value of a managed bean's property

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 12

Hibernate Introduction

Lab 12: Hibernate Introduction

1. Introduction

Hibernate framework simplifies the development of java application to interact with the database. Hibernate is an open source, lightweight, **ORM (Object Relational Mapping)** tool. An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.

2. Relevant Lecture Reading

http://www.java4s.com/hibernate/why-and-what-is-hibernate-hibernate-introduction/
http://www.javatpoint.com/hibernate-tutorial

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
6.2	Setting up to use	10 min	10 min
6.3	Walkthrough Tasks	65 min	65min
7	Practice Tasks	As per time specified for each task	45 min
8	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

1. Hibernate Basics.
2. Hibernate Architecture.
3. Hibernate First application

5. Concept Map

5.1. Drawbacks of JDBC:

1 In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.

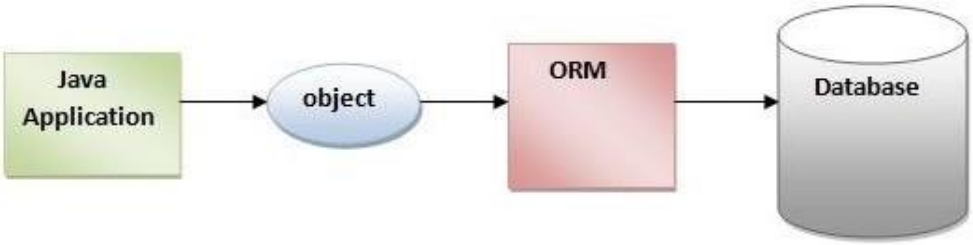
2 Here as a programmer we must close the connection, or we may get a chance to get out of connections message…!

3 Actually if we didn't close the connection in the finally block, then jdbc doesn't responsible to close that connection.

4 In JDBC we need to write Sql commands in various places, after the program has created if the table structure is modified then the JDBC program doesn't work, again we need to modify and compile and re-deploy required, which is tedious.
5 JDBC used to generate database related error codes if an exception will occur, but java programmers are unknown about this error codes right.
6 In the Enterprise applications, the data flow within an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text. Because JDBC doesn't transfer objects directly.

What is Hibernate?

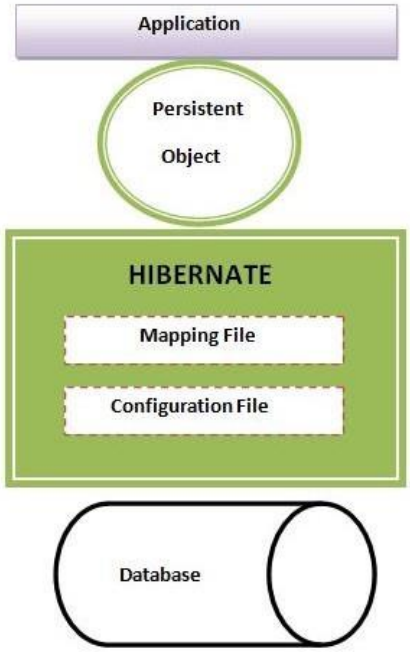
1 Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high-performance Object-Relational Persistence and Query service for any Java Application.
2 Hibernate is a non-invasive framework, means it won't force the programmers to extend/implement any class/interface, and in hibernate we have all POJO classes so its light weight.
3 Hibernate is purely for persistence (to store/retrieve data from Database).
4 Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.
5 Hibernate sits between traditional Java objects and database server to handle all the work in persisting those objects based on the appropriate O/R mechanisms and patterns.



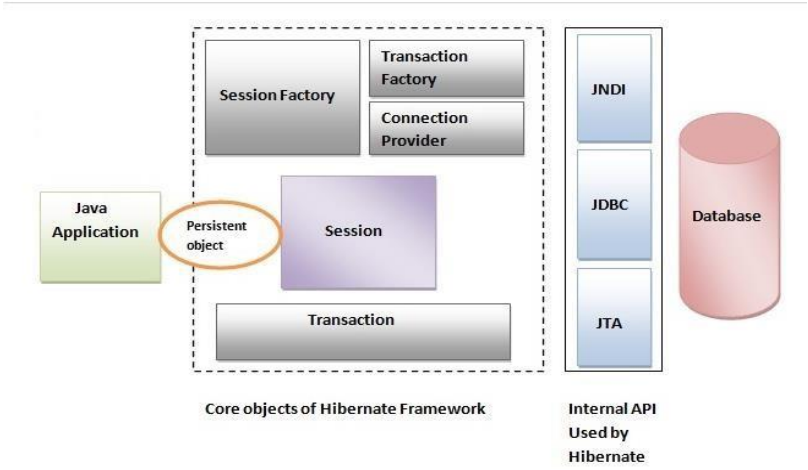
The ORM tool internally uses the JDBC API to interact with the database.

6.1. Hibernate Architecture

The Hibernate architecture includes many objects **persistent object, session factory, transaction factory, connection factory, session, transaction** etc. There are 4 layers in hibernate architecture java application layer, hibernate framework layer, backhand API layer and database layer. Let's see the diagram of hibernate architecture.



This is the high-level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects session factory, session, transaction etc. along with existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

6.2. Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transaction. It is optional.

6.3. Hibernate Advantages:

1. Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
2. Provides simple APIs for storing and retrieving Java objects directly to and from the database.
3. If there is change in Database or in any table then the only need to change XML file properties.
4. Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
5. Hibernate does not require an application server to operate.
6. Manipulates Complex associations of objects of your database.
7. Minimize database access with smart fetching strategies.
8. Provides simple querying of data.

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

7. Tools and Procedure.

7.1. Tools

1. IntelliJ Idea.
2. MySql and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to Section 7 of Lab02 in order to setup environment and project.

7.3. Installing MySql (XAMP)

Lab03 explains how to install MySql using XAMP.

7.4. Downloading Hibernate

Hibernate can be downloaded at the time of web application creation.
Other way to download hibernate is by going to http://hibernate.org/orm/downloads/ and download the latest stable version of hibernate.

8. Walkthrough Task

8.1. Creating First Hibernate Application

Here, we are going to create the first hibernate application. We will consider Laptop Object for it. For creating the first hibernate application, we need to follow following steps: (not necessary to follow in order but better for ease)

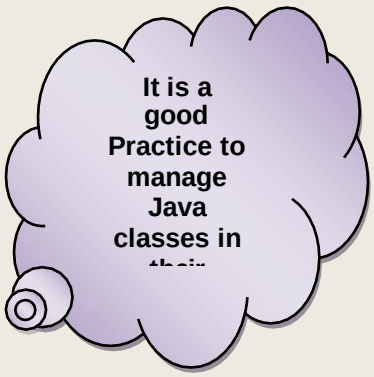
1. Create J2EE application by going to **File>New>Project** and then checking the following options from the menu
    a) **Web application**
    b) **JSF**
    c) **Prime Faces (Optional)**
    d) **Hibernate**
2. Create the Persistent class **(POJO Java Class)**
3. Create the mapping file for Persistent class **(hbm.xml file)**
4. Create the Configuration file if not created during project creation **(hibernate.cfg.xml)** and do configuration.
5. Create Hibernate Util class (Optional but recommended)
6. Create the class that retrieves or stores the persistent object **(Data Access Object (DAO) Class)**
7. Create **Managed Bean** Class for POJO.
8. Load the jar file **(Downloaded Jar files into Project)**
9. Create **JSF** page.
10. Create Database table.
11. Run the first Hibernate application

Step 1: Create hibernate web application

Create new Project and Select the above 4 options from the menu Select JDK and Server and click next. Check the Option of creating hibernate config file. You will be asked to download hibernate libraries, check Download and go next.

Step 2: Create POJO Class:

Create a package pojo and create a java class named as Laptop.java in it. It is the simple bean class representing the Persistent class in hibernate.




```
package pojo;

public class Laptop {
    int id;
    String company;
    String model;
    String ram;
    String hd;
    String processor;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getCompany() {
        return company;
    }

    public void setCompany(String company) {
        this.company = company;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public String getRam() {
        return ram;
    }

    public void setRam(String ram) {
        this.ram = ram;
    }

    public String getHd() {
        return hd;
    }

    public void setHd(String hd) {
        this.hd = hd;
    }

    public String getProcessor() {
        return processor;
    }

    public void setProcessor(String processor) {
        this.processor = processor;
    }
}
```

Step 3: Create HBM File (Mapping of Database Table and POJO)

It maps the User class with the table of the database.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="pojo.Laptop" table="laptop">
        <id name="id"><generator class="increment"></generator></id>
        <property name="company">
            <column name="laptopcompany"></column>
        </property>
        <property name="model">
            <column name="laptopmodel"></column>
        </property>
        <property name="ram">
            <column name="laptopram"></column>
        </property>
        <property name="hd">
            <column name="laptophd"></column>
        </property>
        <property name="processor">
            <column name="laptoppro"></column>
        </property>
    </class>
</hibernate-mapping>
```

Step 4: Create hibernate.cfg.xml (if not created during project creation)

It is a configuration file, containing information about the database and mapping files.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="connection.url">jdbc:postgresql://localhost:5432/examples</property>
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">root</property>

        <mapping resource="hbm/laptop.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Step 5: Create Hibernate Util class (Optional but recommended)

It simply creates and returns a SessionFactory Object.

Create a package util and create a java file named as **HibernateUtil.java** in it.

```
package util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory=buildSessionFactory();
   public static SessionFactory buildSessionFactory(){
        try {
            return new Configuration().configure().buildSessionFactory();
        } catch (Throwable ex) {
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Step 6: Create DAO Class for Laptop

A Dao class, containing method to store the instance of Laptop class. It is responsible to connect with the database.

```
package dao;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import pojo.Laptop;
import util.HibernateUtil;

public class LaptopDAO {

    public void SaveLaptop(Laptop laptop)
    {
        Session session= HibernateUtil.getSessionFactory().openSession();
        Transaction t=session.beginTransaction();
        session.save(laptop);
        t.commit();
        session.close();
    }
}
```

Step 7: Create Managed Bean

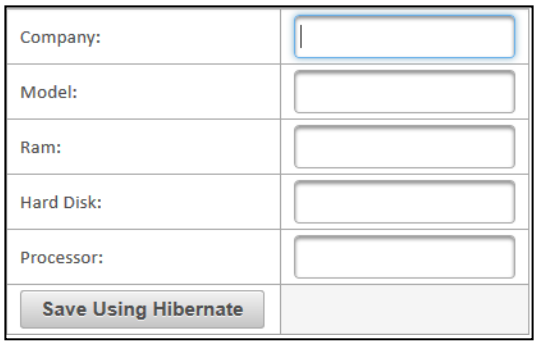
Create a package beans and create Managed bean LaptopBean.java in it.

```
package beans;

import dao.LaptopDAO;
import pojo.Laptop;

import javax.faces.bean.ManagedBean;

@ManagedBean(name = "laptopBean")
public class LaptopBean {
    Laptop l;

    public LaptopBean()
    {
        l=new Laptop();
    }

    public Laptop getL() {
        return l;
    }

    public void setL(Laptop l) {
        this.l = l;
    }

    public void savetoDB()
    {
        LaptopDAO ld=new LaptopDAO();
        ld.SaveLaptop(l);
        System.out.print("Success");

    }


}
```

Step 8: Load Jar files (if hibernate not selected at time of Project Creation)
Include all Jar Files from the Folder.

Step 9: Create JSF Page (xhtml file for front end)
Create a JSF Form index.xhtml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
    <h:outputStylesheet name="css/bootstrap.css"></h:outputStylesheet>
</h:head>
<h:body>
    <h:form >
        <p:panelGrid columns="2" column="2" style="font-family: calibri;" styleClass="well well-sm" >
            Company:
            <p:inputText styleClass="form-control" value="#{laptopBean.l.company}"></p:inputText>
            Model:
            <p:inputText styleClass="form-control" value="#{laptopBean.l.model}"></p:inputText>
            Ram:
            <p:inputText styleClass="form-control" value="#{laptopBean.l.ram}"></p:inputText>
            Hard Disk:
            <p:inputText styleClass="form-control" value="#{laptopBean.l.hd}"></p:inputText>
            Processor:
            <p:inputText styleClass="form-control" value="#{laptopBean.l.processor}"></p:inputText>
            <p:commandButton value="Save Using Hibernate" styleClass="btn btn-sm"
action="#{laptopBean.savetoDB()}"></p:commandButton>
        </p:panelGrid>
    </h:form>
</h:body>
</html>
```



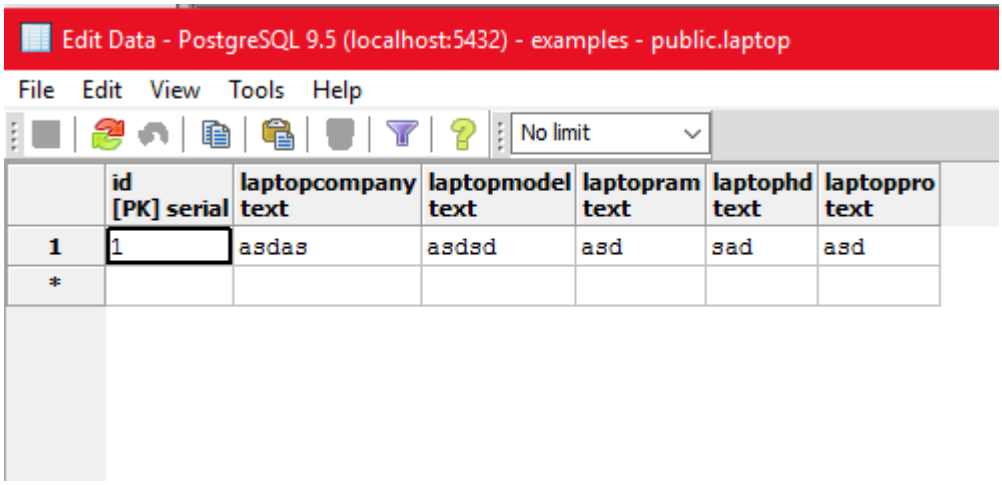
Step 10: Create Database table

Create Database table and remember to **name the table and columns** same as you mentioned in the **laptop.hbm.xml** file.

```
create table laptop
(
id serial primary key,
laptopcompany text,
laptopmodel text,
laptopram text,
laptophd text,
laptoppro text
)
```

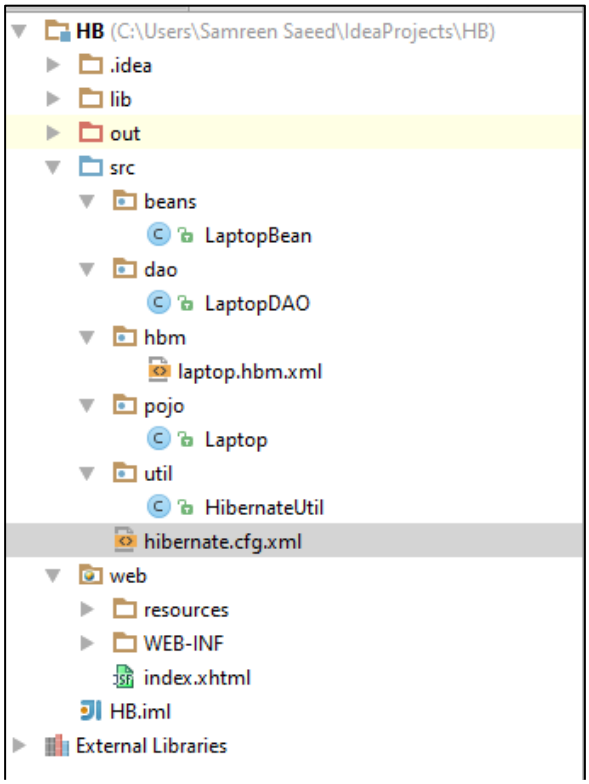
Step 11: Run and Test Application

Run and test your application it should produce the following result.



	id [PK] serial	laptopcompany text	laptopmodel text	laptopram text	laptophd text	laptoppro text
1	1	asdas	asdsd	asd	sad	asd
*						

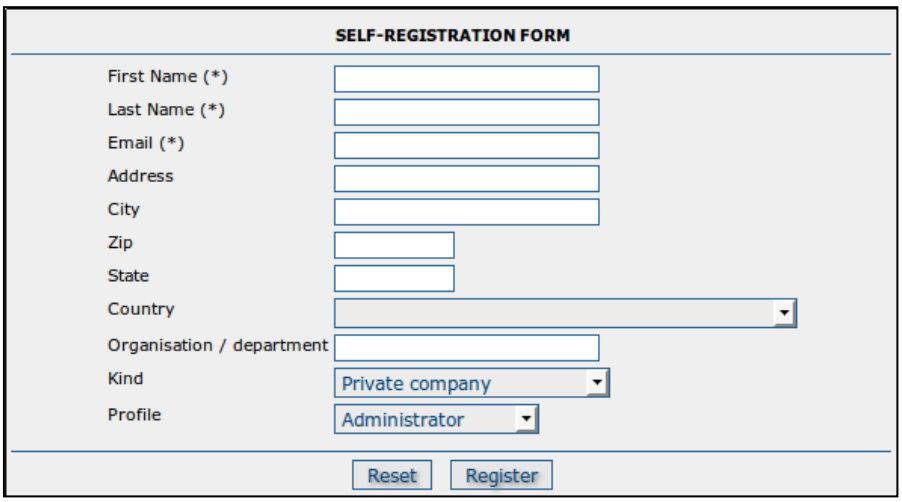
Project Structure



9. Practice Tasks

9.1. Practice Task 1

Create below form using JSF and prime faces tags and save the data into database using hibernate framework.



10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

http://geekonjava.blogspot.com/2015/06/simple-crud-with-jsf-and-hibernate.html

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 13

Hibernate Mappings and Relationships

Lab 13: Hibernate Association Mappings and Relationships

1. Introduction

The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways in which the cardinality of the relationship between the objects can be expressed. An association mapping can be unidirectional as well as bidirectional.

Mapping type	Description
Many-to-One	Mapping many-to-one relationship using Hibernate
One-to-One	Mapping one-to-one relationship using Hibernate
One-to-Many	Mapping one-to-many relationship using Hibernate
Many-to-Many	Mapping many-to-many relationship using Hibernate

2. Relevant Lecture Reading

https://www.tutorialspoint.com/hibernate/hibernate_or_mappings.htm

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
6.2	Setting up to use	10 min	10 min
6.3	Walkthrough Tasks	65 min	65 min
7	Practice Tasks	As per time specified for each task	45 min
8	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

- Mappings between Objects
- ORM using annotations.

5. Concept Map

5.1. Many to One Association

A many-to-one association is the most common kind of association where an Object can be associated with multiple objects. For example, a same address object can be associated with multiple employee objects.

5.2. One to One Association

A one-to-one association is similar to many-to-one association with a difference that the column will be set as unique. For example, an address object can be associated with a single employee object.

5.3. One to Many Association

https://www.tutorialspoint.com/hibernate/hibernate_set_mapping.htm

A One-to-Many mapping can be implemented using a Set java collection that does not contain any duplicate element. We already have seen how to map Set collection in hibernate, so if you already learned Set mapping then you are all set to go with one-to-many mapping.
A Set is mapped with a <set> element in the mapping table and initialized with java.util.HashSet. You can use Set collection in your class when there is no duplicate element required in the collection.

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab02*** and ***Lab03***.

9. Tools and Procedure.

7.5. Tools

IntelliJ Idea.
MySql and Tomcat Server

7.6. Setting up to use IntelliJ Idea and Tomcat Server.

Refer to Section 7 of Lab02 in order to setup environment and project.

7.7. Installing MySql (XAMP)

Lab03 explains how to install MySql using XAMP.

7.8.Downloading Hibernate

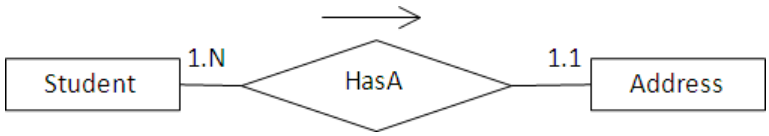
Hibernate can be downloaded at the time of web application creation.
Other way to download hibernate is by going to http://hibernate.org/orm/downloads/ and download the latest stable version of hibernate.

7. Walkthrough Task

Before starting this task, it is supposed that you know that what files are required for hibernate. Therefore, the whole procedure to run and hibernate application will not be explained.

8.1. Creating Many to One Association

In this example you will learn how to map many-to-one relationship using Hibernate Annotations.
Consider the following relationship between Student and Address entity.



According to the relationship many students can have the same address.
To create this relationship, you need to have a student and address table. Data base Queries to create tables are given below.

```
create table address
(
aid serial primary key,
street text,
city text,
state text
)


create table
student (
sid serial primary key,
sname text,
aid int,
foreign key(aid) references address(aid)
)
```

To create the STUDENT and ADDRESS table you need to create the following Java classes with hibernate annotations.

Student class is used to create the student table.

```
package pojo;

import javax.persistence.*;

@Entity
@Table(name = "student")
public class Student {
    int sid;
    String sname;
    StudentAddress studentAddress;


    @Id
    @Column(name = "sid")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int getSid() {
        return sid;
    }

    public void setSid(int sid) {
        this.sid = sid;
    }

    @Column(name = "sname")
    public String getSname() {
        return sname;
    }

    public void setSname(String sname) {
        this.sname = sname;
    }

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "aid")
    public StudentAddress getStudentAddress() {
        return studentAddress;
    }

    public void setStudentAddress(StudentAddress studentAddress) {
        this.studentAddress = studentAddress;
    }


}
```

The *@ManyToOne* annotation is used to create the many-to-one relationship between the *Student* and *Address* entities. The *cascade* option is used to cascade the required operations to the associated entity. If the *cascade* option is set to *CascadeType.ALL* then all the operations will be cascaded. For instance when you save a Student object, the associated Address object will also be saved automatically.

StudentAddress class is used to create the ***address*** table.

```java
package pojo;

import javax.persistence.*;

@Entity
@Table(name = "address")
public class StudentAddress {
    int aid;
    String street;
    String city;
    String state;

    @Id
    @Column(name = "aid")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int getAid() {
        return aid;
    }

    public void setAid(int aid) {
        this.aid = aid;
    }

    @Column(name = "street")
    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    @Column(name = "city")
    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    @Column(name = "state")
    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }
}
```

Now create hibernate configuration file with the *Student* and *Address* class mapping. In the case of Annotation mappings we do not require the hbm files.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <property name="connection.url">jdbc:postgresql://localhost:5432/examples</property>
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">root</property>
        <mapping class="pojo.Student"></mapping>
        <mapping class="pojo.StudentAddress"></mapping>
    </session-factory>
</hibernate-configuration>
```

Create DAO Class for Student

```
package dao;

import org.hibernate.Session;
import org.hibernate.Transaction;
import pojo.Student;
import util.HibernateUtil;

public class StudentDAO {

    public void storeStudent(Student s)
    {
        Session session= HibernateUtil.getSessionFactory().openSession();
        Transaction tx=session.beginTransaction();
        session.save(s);
        tx.commit();
        session.close();
    }
}
```

Create the Managed Bean class for Student to save Student into the database.

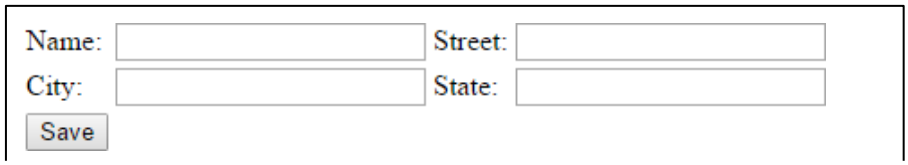
```
package beans;

import com.sun.jndi.cosnaming.IiopUrl;
import dao.StudentDAO;
import pojo.Student;
import pojo.StudentAddress;

import javax.faces.bean.ManagedBean;

@ManagedBean(name = "studentBean")
public class StudentBean {
    Student s;
    StudentAddress sa;

    public Student getS() {
```

```
        return s;
    }

    public StudentAddress getSa() {
        return sa;
    }

    public void setSa(StudentAddress sa) {
        this.sa = sa;
    }

    public void setS(Student s) {
        this.s = s;
    }

    public StudentBean()
    {
        s=new Student();
        sa=new StudentAddress();
    }

    public void savestudent()
    {

        StudentDAO sdao=new StudentDAO();
        s.setStudentAddress(sa);
        sdao.storeStudent(s);

    }
}
```

Create JSF Form

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head></h:head>
<h:body>
    <h:form>
        <h:panelGrid columns="4">
            Name:
            <h:inputText value="#{studentBean.s.sname}"></h:inputText>

            Street:
            <h:inputText value="#{studentBean.sa.street}"></h:inputText>

            City:
            <h:inputText value="#{studentBean.sa.city}"></h:inputText>

            State:
            <h:inputText value="#{studentBean.sa.state}"></h:inputText>

            <h:commandButton value="Save"
action="#{studentBean.savestudent()}"></h:commandButton>
        </h:panelGrid>
```

```
    </h:form>
</h:body>
</html>
```



8. Practice Tasks

9.1. Practice Task 1

By using JSF and Hibernate, Create a scenario of **One to Many relationship between** Customer and Orders. (By using List).Take Concept from below link.
http://www.javatpoint.com/list-one-to-many

9. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

10. Further Reading

http://www.javatpoint.com/mapping-list-in-collection-mapping
http://www.javatpoint.com/list-one-to-many
https://www.tutorialspoint.com/hibernate/hibernate_or_mappings.htm

Capital University of Science and Technology Islamabad
Department of Computer Science, Software Engineering
Faculty of Computer Science

Lab Manual for Enterprise Application Development

Lab 14

Spring Boot

Lab 14: Spring Boot

1. Introduction

Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

2. Relevant Lecture Reading

https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

3. Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
6.2	Setting up Project	10 min	10 min
6.3	Walkthrough Tasks	65 min	65min
9	Practice Tasks	As per time specified for each task	45 min
10	Evaluation Task (Unseen)	25 min for each assigned task	50 min

4. Objective of the Experiment

This Lab is designed to understand

1. Restful Web API's.
2. Spring Boot.
3. Hibernate

5. Concept Map

5.1. What is Spring Boot

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

5.2. Advantages

Spring Boot offers the following advantages to its developers –

1. Easy to understand and develop spring applications
2. Increases productivity
3. Reduces the development time

6. Homework before Lab

From now on it is assumed that you know how to ***configure environment, create project and manage database.***
For help revise ***Lab12*** and ***Lab13***.

6.1. Home Work 1

Install all the tools and configure the Spring Boot environment before coming to Lab.

7. Tools and Procedure

7.1. Tools

1 IntelliJ Idea.
2 MySql and Tomcat Server

7.2. Setting up to use IntelliJ Idea and Tomcat Server

Refer to Section 7 of **Lab02** in order to setup environment and project.

7.3. Installing MySql (XAMP)

Section 7.3 of **Lab02** explains how to install MySql.

8. Walkthrough Task

8.1. Walkthrough Task 1

To create our Hello World Application in Spring boot, follow the below steps:

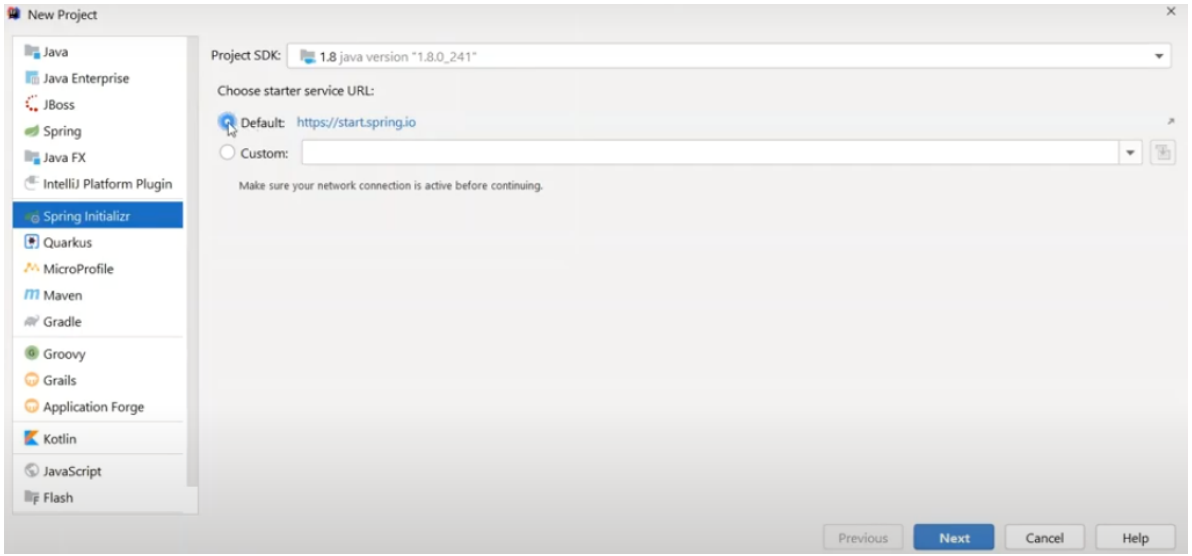


Figure 11: Select Spring Initializer

Give some name to your project

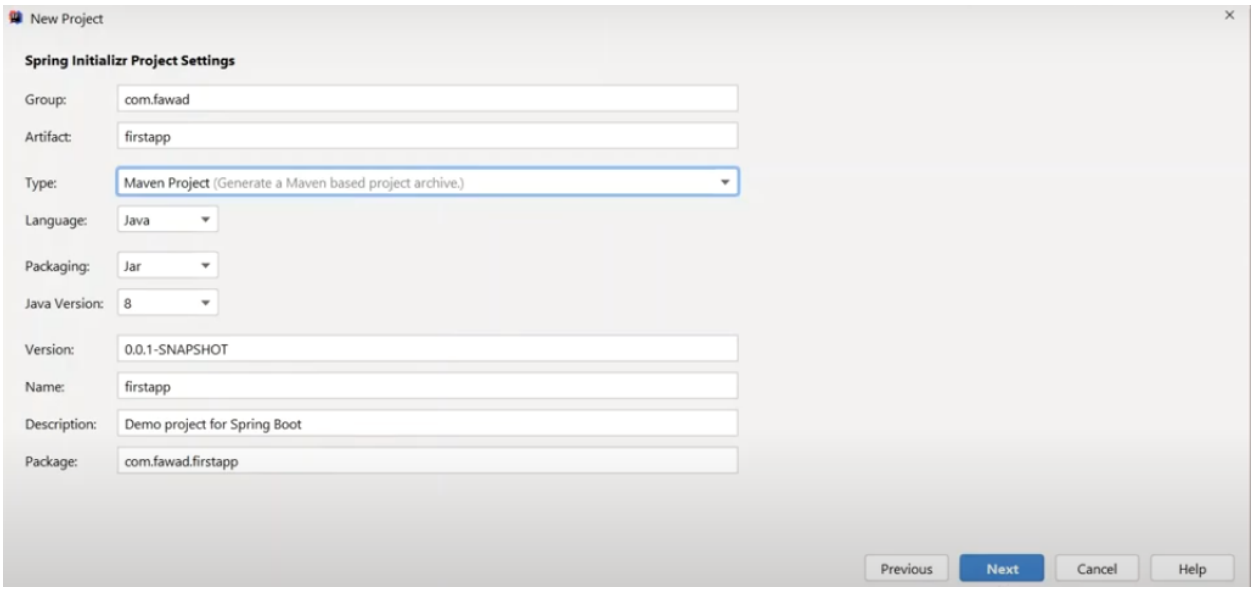
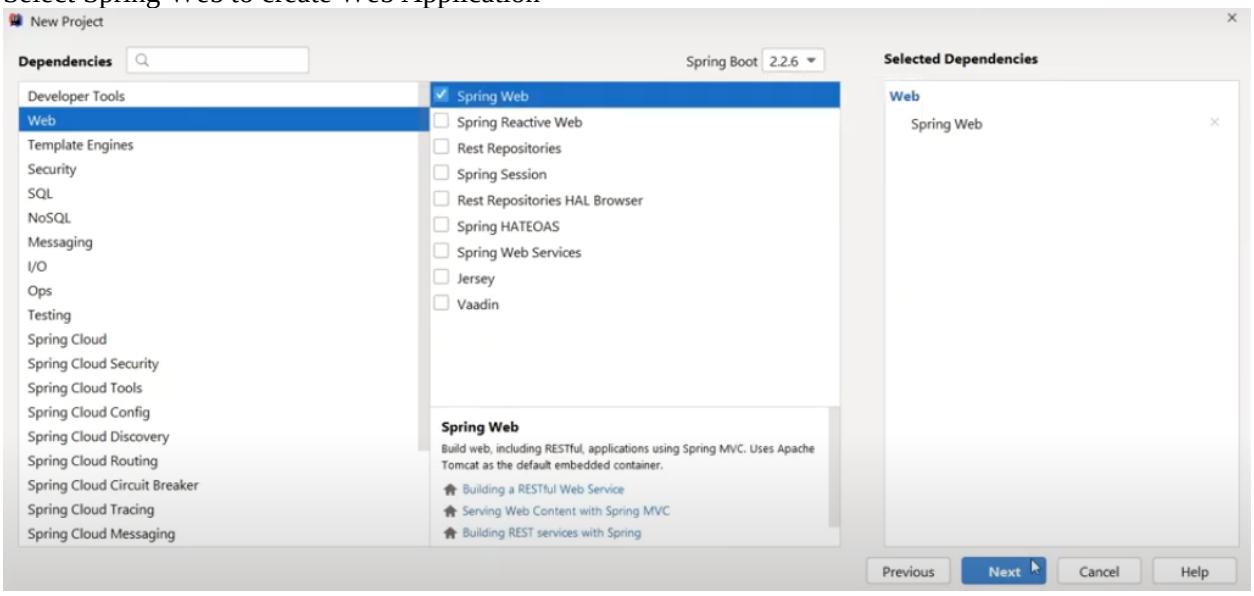


Figure 12: Project Naming

Select Spring Web to create Web Application



.

Now create a package inside the **src**>main>java>[YOUR PROJECT NAME FOLDER] named **controller** And Create java class named **HomeController**

```
@RestController
public class HelloController {
   @RequestMapping(value = "/hello")
   public String sayHello(){
      return "Hello World!";
   }
}
```

Add a controller to display hello world message. **@RestController** makes it restfull controller and **@RequestMapping(value = "/hello")** defines URL mapping

Now run this project and navigate to http://localhost:8080/hello to view the **Hello World** message.

8.2. Walkthrough task 2

CRUD Operation using H2 Database, Hibernate and Spring Boot.
To use database we need few dependencies to be installed Go to mavenrepository and search for the following dependencies and add it to your pom.xml *dependencies* section

1. Lombok
2. Data JPA Starter
3. H2 Database

Now you have added a database named H2 database(in-memory db), you need to specify database name too

1. Open application.properties in resources folder
2. specify database name as follows

```
spring.datasource.url=jdbc:h2:~/test;DB_CLOSE_ON_EXIT=FALSE
spring.jpa.hibernate.ddl-auto=update
```

Creating Model

Create a package **entity** where you will create Entity classes e.g we are creating **Person** entity

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int ID;
    private String Name;
    private int Age;
    private double Height;
    private String CNIC;
}
```

Use **@Entity** annotation to make it entity, **@Data** to make setter getters, **@NoArgsConstructor** to create no argument constructor, **@AllArgsConstructor** to make argument constructor. **@Id** to make it primary key, **@GeneratedValue(strategy = GenerationType.AUTO)** for autoincrement.

Creating Repository

Now create a repository for every entity, create package named **repository** and create interface,, e.g. PersonRepository that will extend **JpaRepository<Person,Integer>** and use **@Repository** annotation on

it.

```
@Repository
public interface PersonRepository extends JpaRepository<Person,Integer> {
}
```

Creating Service

Services will contain business logic, e.g. CRUD operation in this case. Create package **service** and create service for every repository. e.g. **PersonService**

Use **@Service** annotation on PersonService. In PersonService, create private object of PersonRepository and use **@Autowired** annotation on it, so spring framework will initilize that object.

```
@Service
public class PersonService {
    @Autowired
    private PersonRepository personRepository;
    public List<Person> getPersons(){
        return personRepository.findAll();
    }

    public Person getPerson(int id){
        return personRepository.findById(id).orElse(null);
    }

    public Person addPerson(Person person){
        personRepository.save(person);
        return person;
    }
    public Person updatePerson(int id,Person person){
        person.setID(id);
        personRepository.save(person);
        return person;
    }
    public void deletePerson(int id){
        personRepository.deleteById(id);
    }
}
```

Creating Controller

Now in the controller package, create **PersonController** that will manage Http requests. Use **@RestController**, **@RequestMapping(value = "/person")** as we do in controllers. Create an object of PersonService in PersonController and use **@Autowired** annotation on it, so spring framework will manage object creation.

Now create GET, POST, PUT and DELETE methods with **@GetMapping**,**@PostMapping**, **@PutMapping(value = "/{id}")** and **@DeleteMapping(value = "/{id}")**. In the function parameters, use **@PathVariable int id** to get data from URL like localhost/person/1, and if we use **@RequestParam** it would be like localhost/person?id=1 and **@RequestBody Person person** to get data from body.

```
@RestController
@RequestMapping(value = "/person")
public class PersonController {
    @Autowired
    private PersonService personService;
    @GetMapping
    public List<Person> getPersons(){
        return  personService.getPersons();
    }

    @GetMapping(value = "/{id}")
    public Person getPerson(@PathVariable int id){
        return personService.getPerson(id);
    }

    @PostMapping
    public Person addPerson(@RequestBody Person person){
        return personService.addPerson(person);
    }
    @PutMapping(value = "/{id}")
    public Person updatePerson(@PathVariable int id,@RequestBody Person person){
        return personService.updatePerson(id,person);
    }

    @DeleteMapping(value = "/{id}")
    public void deletePerson(@PathVariable int id){
        personService.deletePerson(id);
    }
}
```

Now you can test the API using Postman tool.

9. Practice Tasks

9.1. Practice Task 1

Create an Entity with the following attributes:

1. carId
2. carName
3. makeYear
4. carColor
5. carPrice

Using Spring Boot Web API's to Create, Update, Delete and View details of specific car.

10. Evaluation Task (Unseen)

Evaluation task will be given at the time of lab.

11. Further Reading

https://github.com/fawad1997/SpringWebAPI
https://www.youtube.com/playlist?list=PL4dV-t7IfKfOKLLMedklq7QPyEyGG_9M5