J2SE - Java 2 Standard Edition (Core Java)
J2EE - Java 2 Enterprise Edition (Advanced Java)

https://github.com/PriteshGajjarJava/advJava2025/

Types of applications:
- Web Application - facebook.com, twitter.com, gmail.com
- Mobile Application - Mail, Contact, Games
- Desktop Application - Word, Notepad, Editor

Client side technologies - HTML, CSS, JavaScript, jQuery, (Angular, React, View Js)
Server side technologies - python, java, Ruby, Go, C++ (J2EE Spring)
Middleware - Database (JDBC, ORM- Hibernate)

Server side :
JSP, Servlet, Session, Cookies, How HTTP protocol works?
REST API

DB:
MySQL, JDBC, Hibernate

List of softwares:
- JDK 11
- Web server (Tomcat zip)
- VS Code (Editor)
- Latest version of Chrome and Firefox
- MySQL/PostgreSQL/Oracle (MySQL workbench)
- Eclipse JEE

HTML
- Hypertext Markup Language
- Used to build static web page.
- It defines “Structure” of a webpage
- Latest version is HTML5
- File extension is “.html”

CSS
- Cascading StyleSheet
- It gives style to your web page.
- It helps to beautify webpage. It controls look & feel of your webpage.
- Color, Positioning, Font, Placement
- We can write CSS in different ways.
    a) Inline CSS - Written witting tag itself using “style” attribute. e.g. <div style=“text-align:center”> … </div>
    b) CSS class - We can add CSS class inside <head> => <style> tag. We write this using dot (.) e.g.
        .cetner-align { … }
    c) ID based CSS - We can add CSS for particular ID’s using #
- File extension : If we have to externalize CSS code then separate

file can be written using “.css” extension.

JavaScript:
- It defines “Behavior” of your web page.
- Using JS we can build “dynamic” web page.
- JS is used to add validations, DOM manipulation, dynamic CSS changes
- It is also used to make AJAX call (API call)
- Node JS - Special package of JS which can be used for server side programming.

Session:
- HTTP is state-less protocol.
- Request “n” doesn’t have any information about previous request “n-1”.
- If we have to store data between multiple requests, “Session” memory can be used.
- Session is type of storage on server side (Server side memory)
- Data required across requests can be stored in the session.

e.g. Shopping WebSite
Step : You added “shoes” to cart (Shoes info get stored in session)
Step 2: You added “t-shirt” to cart (T-Shirt info get stored in session)
Step 3: Go to cart -> Pay bill -> For summary you would access data from session and calculate the bill

- Session data is “key-value” pair
- In JSP “session” is in-built variable
- On Servlet -> you can get session object using “request.getSession()”

- To add data in session : “session.setAttribute(key, value)”
- To read data from session : “session.getAttribute(key)”

- Session memory get destroyed when you close the browser OR Logout.

- If 100 users login to a site, then there will be 100 different sessions on the server side.

Cookie:
- Cookie is small piece of information, generated by a server and stored on client side inside browser.
- Cookies have age (seconds, min, hrs, years, no expiry)
- Every browser has cookie storage. It stores cookie information domain (website) wise.
- For every HTTP response: Browser checks if there are cookies in the response. If Yes, browser will add that cookie inside browser storage.
- For every HTTP request: Browser check if there are existing valid cookie in browser storage, if yes those cookies will be sent in HTTP request.
- Best example of cookie is: “JSESSION_ID”. For login request, once authentication is successful - server generate cookie named

“JSESSION_ID” and send in HTTP response. From next request browser would send JSESSION_ID to server which helps server to identify user.
- Cookies are browser and site specific.

Restaurant mgmt system:

```
Create database hotel_order_mgmt
create table menu(menuId integer primary key NOT NULL
AUTO_INCREMENT, name varchar(50), price float);
insert into menu(name,price) values('Sandwich', 100);

create table food_table(table_id integer primary key, capacity
integer);

create table orders(order_id integer primary key NOT NULL
AUTO_INCREMENT, table_id integer, menu_id integer, qty integer);

create table transactions(tId integer primary key NOT NULL
AUTO_INCREMENT, orderId integer, orderDate date, bill integer,
FOREIGN KEY (orderId) references orders(order_id));
```

Hibernate:
- Install Maven
- Configure Maven in Eclipse
- Create new Maven Project
- Select arch type as “maven-arctype-quickstart”
- IN Pom.xml add following:

```
  <dependencies>
          <!-- https://mvnrepository.com/artifact/org.hibernate.orm/
hibernate-core -->
        <dependency>
            <groupId>org.hibernate.orm</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.6.8.Final</version>
        </dependency>
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <version>8.0.32</version>
        </dependency>
  </dependencies>
```

- Right click project -> Maven -> Update Project (This will download required libs)

- Create folder src/main/resources
- Under this folder create new XML file named “hibernate.cfg.xml”
- Add following in xml file.

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-
```

```
configuration-3.0.dtd">

<hibernate-configuration>
        <session-factory>
                <!-- JDBC Database connection settings -->
                <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
                <property name="connection.url">jdbc:mysql://
localhost/college</property>
                <property name="connection.username">root</property>
                <property name="connection.password"></property>

                <!-- Echo the SQL to stdout -->
                <property name="show_sql">true</property>

                <mapping class="com.pga.Student" />

        </session-factory>

</hibernate-configuration>
```

- Create Java class named “Student” (under package com.pga) and define properties which matches to table columns.

```
package com.pga;

import jakarta.persistence.*;

@Entity
@Table(name = "student")
public class Student {
        @Id
        private int id;
        private String name;
        private float marks;
        .. constructor
        .. getter + setter methods
}
```

- Add HibernateUtils class

```
package com.pga;

import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.*;

public class HibernateUtils {

        static SessionFactory sessionFactory = null;
        public static SessionFactory getSessionFactory() throws
Exception {
                if (sessionFactory == null) {
                        // A SessionFactory is set up once for an
application!
                        final StandardServiceRegistry registry = new
StandardServiceRegistryBuilder()
                                        .configure() // configures
settings from hibernate.cfg.xml
                                        .build();
                        try {
                                sessionFactory = new
MetadataSources( registry ).buildMetadata().buildSessionFactory();
                        }
                        catch (Exception e) {
                                // The registry would be destroyed
by the SessionFactory, but we had trouble building the
SessionFactory
                                // so destroy it manually.

StandardServiceRegistryBuilder.destroy( registry );
                        }
                }
                return sessionFactory;
        }
}
```

- Actual Hibernate Application code to add new Student record:

```
package com.pga;

import org.hibernate.*;

public class HibernateAppDemo {
        public static void main(String[] args) throws
HibernateException, Exception {
                Transaction tx = null;
                Session session =
HibernateUtils.getSessionFactory().openSession();

                tx = session.beginTransaction();
                session.persist(new Student(10, "Hibernate-Demo",
90)); // insert into student values(10 ..)
                tx.commit();
        }
}
```

Spring Framework:

        spring.io

        Core features:
        - Dependency Injection (IOC - Inversion of Control)

```
        class Student {
                int id;
                String name;
                float marks;
                Address add;

                Student(int id, String name, float m, Address a) {

                }
        }


        class Address {
                String city, state;
                int pincode;
        }
```

MVC: Model View Controller

```
        Model - Data
        View  - UI
        Controller - Business logic
```

Model < —— > Controller < – View – >

Important modules of Spring Framework:
- Web
- Core
- DAO (Data access object)
- AOP (Aspect Oriented Programming)

API architectural styles:
1) SOAP - Simple Object Access Protocol.   (XML based)
2) REST - Representational Starte Transfer. (JSON Based)

pga.com/project1/students

```
<xml>
        <students>
                <student id = “1”>
                        <name> Sagar </name>
                        <marks> 78 </marks>
                </student>
                <student id = “2”>
                        <name> Pga </name>
                        <marks> 98 </marks>
                </student>
        </students>
</xml>
```

REST
JSON:

```
{
  "students": [
    {
      "id": 1,
      "name": "Sagar",
      "marks": 78
    },
    {
      "id": 2,
      "name": "Pga",
      "marks": 98
    }
  ]
}
```

```
        {
  "students": [
    {
      "id": 1,
      "name": "Sagar",
      "marks": 78
    },
    {
      "id": 2,
      "name": "Pga",
      "marks": 98
    }
  ]
}
```

REST v/s SOAP

1) REST is architectural style
   SOAP is a protocol

2) REST support multiple data formats. JSON is more popular.
   SOAP works only with XML

3) REST is faster than SOAP

4) REST supports caching, SOAP doesn’t

5) Security for REST achieved using HTTPs, JWT, OAuth
   Security for SOAP is handled by built-in security (WS-security)

6) REST is used for fast API development.
   SOAP is still used by Banking applications.

Steps for “Spring Hello World”

Pre-requisite : Java , Eclipse, Maven plugin in eclipse

- Create Maven Project with archetype (Quickstart)
- This will help you to have “pom.xml” file in your project.
- In pom.xml file add dependencies for “Spring Core” and “Spring Context” as below

```
  <dependencies>
         <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>6.2.3</version>
        </dependency>

         <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>6.2.3</version>
        </dependency>
  </dependencies>
```

- If you don’t dependencies in project then right click -> Maven -> Update Project

- Create “resources” folder under /src/main/resources

- Create new file named “beans.xml” under /src/main/resources(Code of beans.xml as below)

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/
beans
        http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd">

        <bean id="studentBean" class="com.pga.Student">
                <property name="name" value="Mkyong" />
        </bean>

</beans>
```

- Create package named “com.pga”
- Add Student class with field name (getter, setter, constructor)

```
package com.pga;

public class Student {
        private String name;


        public Student(String name) {
                super();
                this.name = name;
        }
        public Student() {}
```

```
	public String getName() {
		return name;
	}

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

	public void sayHello() {
		System.out.println("Hello - " + this.name);
	}
}
```

- We can have new file “SpringAppDemo.java” which will have code for “Spring initialization” using Application Context and we will try to read
Student bean named as “studentBean”.

```
package com.pga;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringAppDemo {
	public static void main(String[] args) {
		ApplicationContext context = new
ClassPathXmlApplicationContext(
				"beans.xml");

		Student s = (Student)
context.getBean("studentBean");
		s.sayHello();
	}
}
```

```
	REST:
	4 API endpoints

	Read book information
	GET https://pga.com/library/api/v1/books         (Aggregated
Data)
		 [{ book1 }, {book2} …]

	GET https://pga.com/library/api/v1/books/book1 (Specific
book information)


	Create new book (Add new book)
	POST https://pga.com/library/api/v1/books/book3
	Payload
	{
		“id”: “book3”,
                “name”: “Atomi Habits”,
                …
        }

        Update Book:
        PUT https://pga.com/library/api/v1/books/book2
        Payload
        {
                “id”: “book2”,
                “price”: 350 // Updating price
                …
        }

        Delete book
        Delete https://pga.com/library/api/v1/books/book1

        HTTP Response Status Code

        1XX     Informational
        2XX     Successful
        3XX     Redirecation
        4XX     Client Error
        5XX     Server Error

        200 – OK

        400 – Bad request
        401 – Unauthorized
        403 – Forbidden
        404 – Page Not Found (URL is incorrect)

        500 – Internal Server error
        503 – Service unavailable
        504 – Gateway Timeout (n/w problem)

Spring Hello World REST Tutorial by MkYong


https://mkyong.com/spring-boot/spring-rest-hello-world-example/

        // Setting up spring-boot environment

        1) Go to https://start.spring.io/
                Add dependency as
                - Spring Web WEB
                - H2 Database SQL
                - Spring Data JPA SQL

        2) Click “Generate” button -> You will get one .zip file in
Downloads folder (unzip)

        3) Go to eclipse -> Import -> Maven (Existing Maven Project)
-> Select above folder
        New project will be created in background (You will get
pom.xml)

        // Coding steps

        1) Create POJO class named “Book”
        2) Create interface named “BookRepository” which extends
“JpaRepository”
        3) Create Rest Controller (BookRestController)
        4) Main class (SpringBootApplication class) -> Initialize
Bean named “initializeDatabase” with 3 dummy records
        5) Run
        6) Verify using “Curl” or “Postman”
```