Project Management - Git

1. **Github and Bitbucket**
   Visit **https://github.com/**
   Visit **https://bitbucket.org/product**

2. **Installation Git**
   Visit **https://git-scm.com/downloads** and download latest version of Git for Windows and install follow the instruction step by step

3. **Basic command line for Git**
   git checkout
   git branch
   git fetch
   git pull
   git push
   git add
   git merge
   git log
   git stash
   git revert
   git cherry-pick
   …

4. **Installation SourceTree**
   Visit **https://www.sourcetreeapp.com/** and download latest version of SourceTree for Windows and install follow the instruction step by step

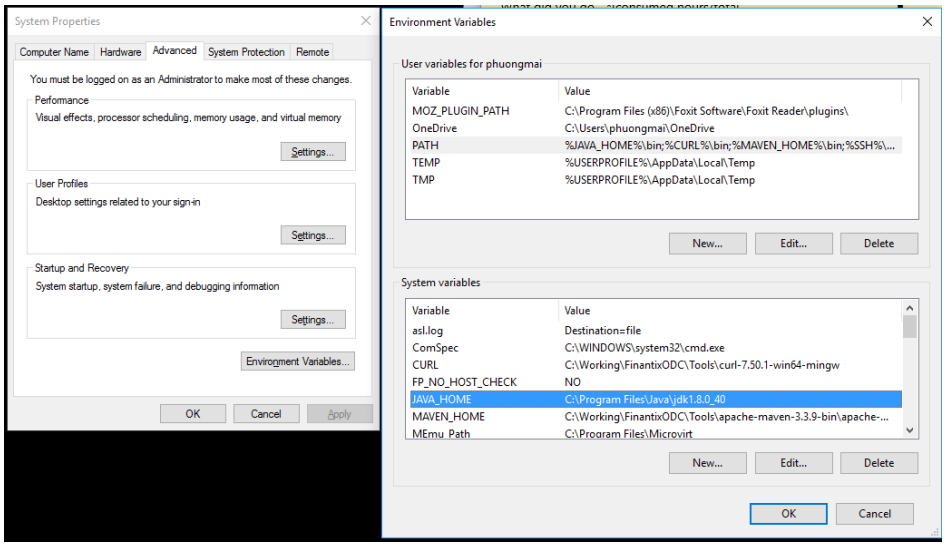
5. **Git Integration for Eclipse**
   Open Eclipse -> **Help** -> **Eclipse Marketplace** -> Search **EGit** keyword -> Click **Install** button at **EGit - Git Integration for Eclipse** section and follow the instruction step by step

Apache Maven

1. **Maven is a tool with multiple facets:**
   - A **Build** tool – Generated artifacts (jar, war, ear, …)
   - A **Dependency Management** tool
   - A **Documentation** tool – Generated test results, quality metrics, Javadoc, …

2. **Require**

   Make sure JDK is installed and JAVA_HOME variable is added



3. **Installation:**

   Visit http://maven.apache.org/download.cgi and download Maven zip file ***apache-maven-3.5.0-bin.zip***
   Unzip the file in a local directory such as*:* ***C:\maven***
   Set environment variables:
   MAVEN_HOME=C:\maven
   PATH=%MAVEN_HOME%\bin

4. **Verification**

   Open cmd and enter **mvn – version** in the command line

5. **Maven eclipse plugin (for eclipse old version)**

   1. Open Eclipse -> Click **Help** -> **Install New Software** -> Click **Add** button at top corner and fill up ***Name=Maven*** *and* ***Location=http://download.eclipse.org/technology/m2e/releases/***
      A **check-box** will appear in the pop window, **Check** and click **Next** button and follow the instruction step by step

   Another way to install Maven plugin:

   2. Open Eclipse -> **Help** -> **Eclipse Marketplace** -> Search **Maven** keyword -> Click **Install** button at **Maven Integration for Eclipse** section and follow the instruction step by step

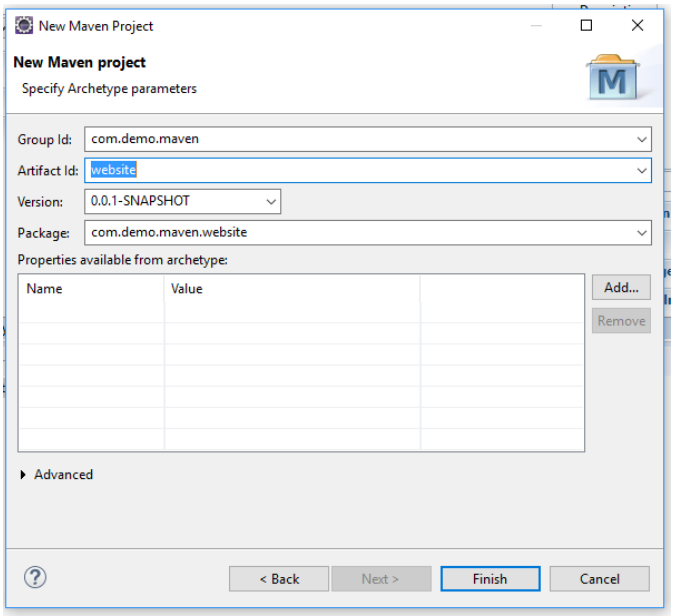
6. **Maven command structure**

   mvn eclipse:eclipse *//generated maven project for Eclipse IDE*
   mvn clean

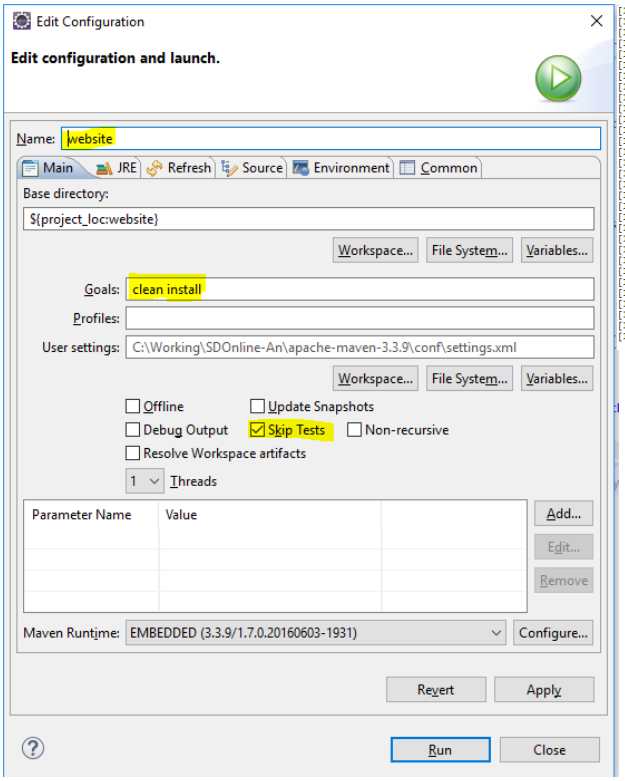
mvn clean install *// install the artifact in your local respository* ***(${user.home}/.m2/repository)***
mvn compile
mvn package
mvn test
...

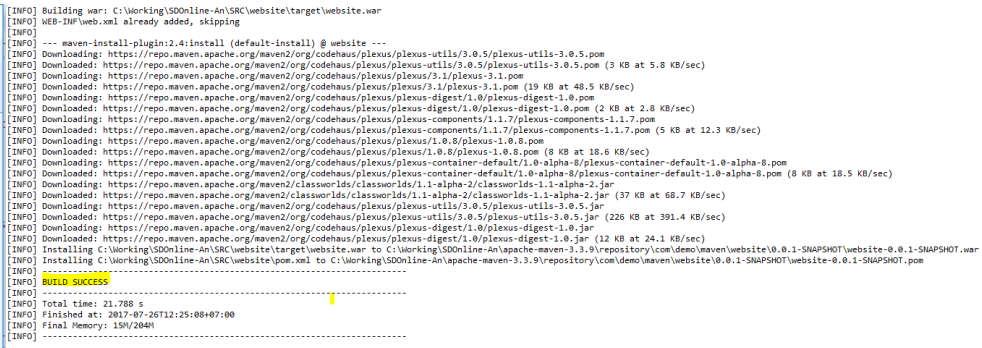
7. **Create Maven project using Eclipse IDE**

Open Eclipse -> **File** -> **New** -> **Maven Project**
Select default Workspace location
Select the maven archetype as: **maven-archetype-webapp**
Fill out details and click Finish



Now build project with **maven clean install** to check dependency issues with project and deploy application on



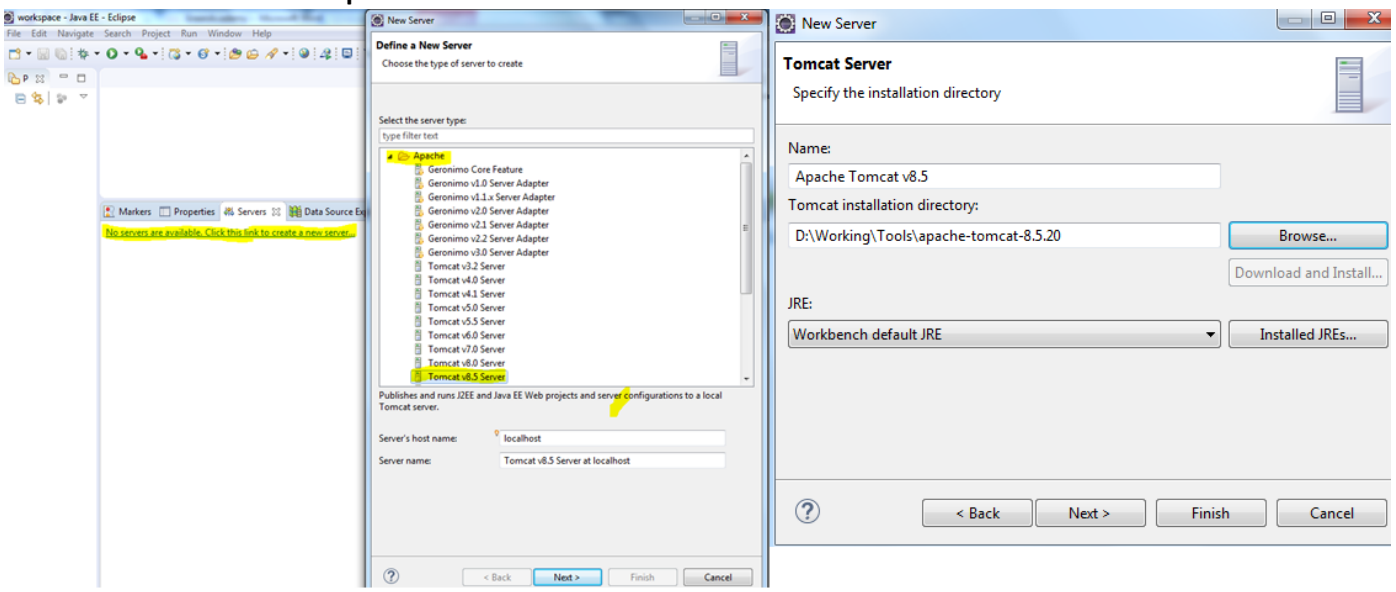


Apache Tomcat Server

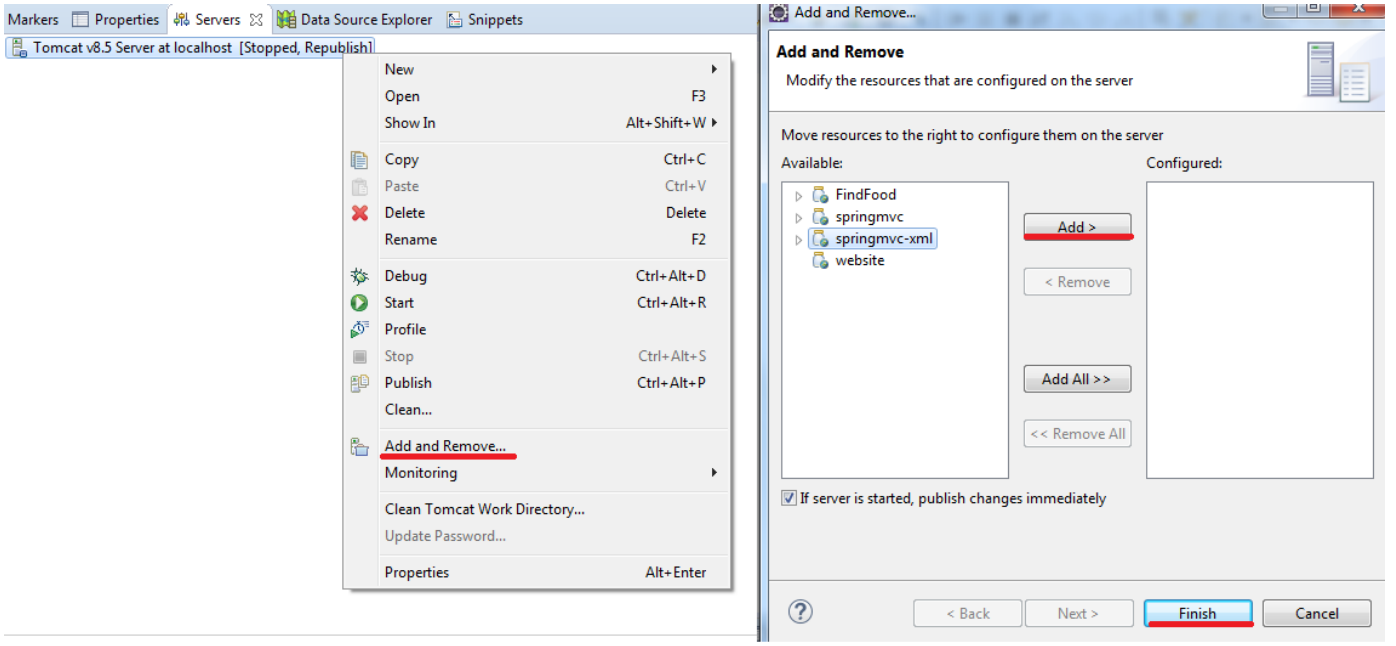
1. Installation

Visit https://tomcat.apache.org/download-70.cgi and download Apache tomcat server zip file
Upzip the file in local directory such as: ***C:\tomcat***

2. Create tomcat server on Eclipse



3. Deploy project into tomcat server on Eclipse



Hibernate with MySQL

1. **Create maven project from Eclipse IDE**
   (Review last session)

2. **Add Hibernate and Mysql dependency**
   Hibernate is required ***dom4j, commons-logging, commons-collections and cglib*** as dependency library
   In case using Anotation, it's required to download the Hibernate annotation library ***hibernate-annotations and hibernate-commons-annotations and repository.jboss***
   Modify the ***pom.xml*** file.

```
...
<repositories>
	<repository>
		<id>JBoss repository</id>
		<url>http://repository.jboss.com/maven2/</url>
	</repository>
</repositories>

<dependencies>
	<!-- MySQL database driver -->
	<dependency>
		<groupId>mysql</groupId>
		<artifactId>mysql-connector-java</artifactId>
		<version>5.1.9</version>
	</dependency>

	<!-- Hibernate core -->
	<dependency>
		<groupId>hibernate</groupId>
		<artifactId>hibernate3</artifactId>
		<version>3.2.3.GA</version>
	</dependency>

	<!-- Hibernate annotation -->
	<dependency>
		<groupId>hibernate-annotations</groupId>
		<artifactId>hibernate-annotations</artifactId>
		<version>3.3.0.GA</version>
	</dependency>
	<dependency>
		<groupId>hibernate-commons-annotations</groupId>
		<artifactId>hibernate-commons-annotations</artifactId>
		<version>3.0.0.GA</version>
	</dependency>

	<!-- Hibernate library dependecy start -->
	<dependency>
		<groupId>dom4j</groupId>
		<artifactId>dom4j</artifactId>
		<version>1.6.1</version>
	</dependency>
	<dependency>
		<groupId>commons-logging</groupId>
		<artifactId>commons-logging</artifactId>
		<version>1.1.1</version>
	</dependency>
	<dependency>
		<groupId>commons-collections</groupId>
		<artifactId>commons-collections</artifactId>
		<version>3.2.1</version>
	</dependency>
	<dependency>
		<groupId>cglib</groupId>
		<artifactId>cglib</artifactId>
		<version>2.2</version>
	</dependency>
	<!-- Hibernate library dependecy end -->
	<dependency>
		<groupId>javax.transaction</groupId>
```

```
                <artifactId>jta</artifactId>
                <version>1.1</version>
        </dependency>
        <dependency>
                <groupId>javax.persistence</groupId>
                <artifactId>persistence-api</artifactId>
                <version>1.0.2</version>
        </dependency>
        <dependency>
                <groupId>javax.persistence</groupId>
                <artifactId>persistence-api</artifactId>
                <version>1.0.2</version>
        </dependency>
</dependencies>
...
```

3. Open MySQL server and create new database - The database's name is **website**
   WARNING: *NOT* *create* *ANY* *tables in this database*

4. Now build project with **mvn eclipse:eclipse** to check dependency issues with project and update the Eclipse's project classpath
5. **Create Hibernate configuration file**
   Create a Hibernate's configuration file and put under the resources root folder ***src/main/resources/hibernate.cfg.xml***

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/website</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">admin</property>
                <property name="hibernate.bytecode.use_reflection_optimizer">false</property>
        <!-- JDBC connection pool settings -->
        <property name="connection_pool_size">10</property>
        <!-- Echo the SQL -->
        <property name="show_sql">true</property>
        <!-- Create or Update the database schema on startup -->
        <property name="hibernate.hbm2ddl.auto">update</property>

        <!-- Using for xml mapping file -->
        <mapping resource="com/website/hibernate/Student.hbm.xml"></mapping>

        <!-- Using for annotation -->
        <mapping class="com.website.hibernate.entities.Student"></mapping>
    </session-factory>
</hibernate-configuration>
```

6. **Create Model class**
   - **For XML Mapping**: create hibernate mapping file

   Create **Student.java** class in package **src/main/java/com/website/hibernate/entities** and **Student.hbm.xml** in **src/main/resources/com/website/hibernate/**

```
package com.website.hibernate.model;

public class Student implements java.io.Serializable {
        private static final long serialVersionUID = 1L;

        private Long id;
        private String firstName;
        private String lastName;
        private Integer age;

        public Student() { }
```

```
        public Student(String firstName, String lastName, Integer age) {
                this.firstName = firstName;
                this.lastName = lastName;
                this.age = age;
        }
        // TODO: generate getter and setter method
}
```

```
<?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="com.website.hibernate.model.Student" table="student" catalog="website">
        <id name="id" type="long">
            <column name="ID" />
            <generator class="identity" />
        </id>
        <property name="firstName" type="string">
            <column name="FIRST_NAME" length="20" not-null="true"/>
        </property>
        <property name="lastName" type="string">
            <column name="LAST_NAME" length="20" not-null="true"/>
        </property>
        <property name="age" type="integer">
            <column name="AGE" length="20" not-null="true"/>
        </property>
    </class>
</hibernate-mapping>
```

❖ **For Annotation:** hibernate mapping on entities class

Create **Student.java** class in package **src/main/java/com/website/hibernate/entities**

```
package com.website.hibernate.entities;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "student", catalog = "website", appliesTo = "")
public class Student implements java.io.Serializable {
        private static final long serialVersionUID = 1L;

        private Long id;
        private String firstName;
        private String lastName;
        private Integer age;

        //TODO: generate constructor for Student

        @Id
        @GeneratedValue(strategy = IDENTITY)
        @Column(name = "ID", unique = true, nullable = false)
        public Long getId() {
                return id;
        }
        public void setId(Long id) {
                this.id = id;
        }

        @Column(name = "FIRST_NAME", length = 20)
        public String getFirstName() {
                return firstName;
        }
        public void setFirstName(String firstName) {
                this.firstName = firstName;
        }

        @Column(name = "LAST_NAME", length = 20)
        public String getLastName() {
                return lastName;
        }
```

```
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	@Column(name = "AGE")
	public Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}
}
```

7. Create Hibernate Utility class

To take care of Hibernate start up and retrieve the session easily

Create **HibernateUtil.java** in **src/main/java/com/website/hibernate/persistence**

❖ **For XML Mapping**

```
package com.website.hibernate.persistence;

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

public class HibernateUtil {
	private static final SessionFactory sessionFactory = buildSessionFactory();

	private static SessionFactory buildSessionFactory() {
		try {
			// Create the SessionFactory from hibernate.cfg.xml
			return new Configuration().configure().buildSessionFactory();
		} catch (Throwable ex) {
			// Make sure you log the exception, as it might be swallowed
			System.err.println("Initial SessionFactory creation failed." + ex);
			throw new ExceptionInInitializerError(ex);
		}
	}

	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}

	public static void shutdown() {
		// Close caches and connection pools
		getSessionFactory().close();
	}
}
```

❖ **For Annotation Mapping**

```
package com.website.hibernate.persistence;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

public class HibernateUtil {
	private static final SessionFactory sessionFactory = buildSessionFactory();

	private static SessionFactory buildSessionFactory() {
		try {
			// Create the SessionFactory from hibernate.cfg.xml
			return new AnnotationConfiguration().configure().buildSessionFactory();
		} catch (Throwable ex) {
			// Make sure you log the exception, as it might be swallowed
			System.err.println("Initial SessionFactory creation failed." + ex);
			throw new ExceptionInInitializerError(ex);
		}
	}

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

```
        public static void shutdown() {
                // Close caches and connection pools
                getSessionFactory().close();
        }
}
```

8. Testing

Create **Test.java** in **src/main/java/com/website/hibernate/main**

```
public class HibernateUtil {

        public static void main(String[] args) {
                System.out.println("Maven + Hibernate + MySQL");
                Session session = HibernateUtil.getSessionFactory().openSession();
                session.beginTransaction();

                Student student = new Student();
                student.setFirstName("Phuong");
                student.setLastName("Mai");
                student.setAge(29);

                session.save(student);
                session.getTransaction().commit();
        }
}
```

Hibernate with MySQL (2)

1. Mapping Types

MAPPING TYPE	JAVA TYPE	SQL TYPE
integer	int or java.lang.Integer	INTEGER
long	long or java.lang.Long	BIGINT
float	float or java.lang.Float	FLOAT
double	double or java.lang.Double	DOUBLE
big_decimal	java.math.BigDecimal	NUMERIC
character	java.lang.String	CHAR(1)
string	java.lang.String	VARCHAR
byte	byte or java.lang.Byte	TINYINT
boolean	boolean or java.lang.Boolean	BIT
binary	byte[]	VARBINARY (or BLOB)
blob	java.sql.Blob	BLOB
date	java.util.Date or java.sql.Date	DATE
time	java.util.Date or java.sql.Time	TIME
timestamp	java.util.Date or java.sql.Timestamp	TIMESTAMP
calendar	java.util.Calendar	TIMESTAMP
calendar_date	java.util.Calendar	DATE

2. Collection Mapping

COLLECTION TYPE	MAPPING
<list>	java.util.List
<set>	java.util.Set
<map>	java.util.Map

3. Fetch Type and Casecase Type

- **Fetch Type –** Default value is **LAZY**
  - ***LAZY***
  - ***EAGER***

- **Casecase Type –** Default value is **NONE**

  With XML mapping
  - ***save***
  - ***update***
  - ***save-update***
  - ***delete***
  - ***all***
  - ***all-delete-orphan*** *– breaking relation between objects not deleting the objects from the database*

  With Annotaion
  - ***PERSIST –*** *same with save and update*
  - ***MERGE –*** *same with update*
  - ***REMOVE –*** *same with delete*
  - ***ALL –*** *same with all*

4. ORM – Object Relationship Mapping

- **One – to – One**

  Using XML mapping

  Create **Address.java** class in package **src/main/java/com/website/hibernate/model** and **Address.hbm.xml** in **src/main/resources/com/website/hibernate/**

```java
package com.website.hibernate.model;

public class Address implements java.io.Serializable {
        private static final long serialVersionUID = 1L;

        private Long id;
        private String street;
        private String district;
        private String city;

        //TODO: generate constructor & getter and setter methods
}
```

```xml
<?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="com.website.hibernate.model.Address" table="address" catalog="website">
        <id name="id" type="long">
            <column name="ID" />
            <generator class="identity" />
        </id>
        <property name="street" type="string">
            <column name="STREET" length="20" not-null="true"/>
        </property>
        <property name="district" type="string">
            <column name="DISTRICT" length="20" not-null="true"/>
        </property>
        <property name="city" type="string">
            <column name="CITY" length="20" not-null="true"/>
        </property>
    </class>
</hibernate-mapping>
```

Update **Student.java** class in package **src/main/java/com/website/hibernate/model** and **Student.hbm.xml** in **src/main/resources/com/website/hibernate/**

```java
package com.website.hibernate.model;

public class Student implements java.io.Serializable {
        private static final long serialVersionUID = 1L;

        private Long id;
        private String firstName;
        private String lastName;
        private Integer age;
        private Address address;

        //TODO: generate constructor & getter and setter methods
}
```

```xml
<?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="com.website.hibernate.model.Student" table="student" catalog="website">
        <id name="id" type="long">
            <column name="STUDENT_ID" />
            <generator class="identity" />
        </id>
        <property name="firstName" type="string">
            <column name="FIRST_NAME" length="20" not-null="true"/>
        </property>
        <property name="lastName" type="string">
            <column name="LAST_NAME" length="20" not-null="true"/>
        </property>
        <property name="age" type="integer">
            <column name="AGE" length="20" not-null="true"/>
        </property>
        <one-to-one name="address" class="com.website.hibernate.model.Address" cascade="save-update"
lazy="false"></one-to-one>
    </class>
</hibernate-mapping>
```

Using Annotation mapping

Create **Address.java** class in package **src/main/java/com/website/hibernate/entities** and Update **Student.java** class in package **src/main/java/com/website/hibernate/entities**

```
package com.website.hibernate.entities;

import static javax.persistence.GenerationType.IDENTITY;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "address", catalog = "website")
public class Address implements java.io.Serializable {
	private static final long serialVersionUID = 1L;

	private Long id;
	private String street;
	private String district;
	private String city;

	//TODO: generate constructor for Address

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "ID", unique = true, nullable = false)
	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}

	@Column(name = "STREET", length = 20)
	public String getStreet() {
		return street;
	}
	public void setStreet(String street) {
		this.street = street;
	}

	@Column(name = "DISTRICT", length = 20)
	public String getDistrict() {
		return district;
	}
	public void setDistrict(String district) {
		this.district = district;
	}

	@Column(name = "CITY", length = 20)
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
}
```

```
package com.website.hibernate.entities;

import static javax.persistence.GenerationType.IDENTITY;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name = "student", catalog = "website")
public class Student implements java.io.Serializable {
	private static final long serialVersionUID = 1L;

	private Long id;
	private String firstName;
	private String lastName;
	private Integer age;
	private Address address;

	//TODO: generate constructor for Student

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "STUDENT_ID", unique = true, nullable = false)
	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}

	@Column(name = "FIRST_NAME", length = 20)
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	@Column(name = "LAST_NAME", length = 20)
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	@Column(name = "AGE")
	public Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}

	@OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
	@PrimaryKeyJoinColumn
	public Address getAddress() {
		return address;
	}
	public void setAddress(Address address) {
		this.address = address;
	}
}
```

❖ **Many – to – Many**

Using XML Mapping

Create **Course.java** class in package **src/main/java/com/website/hibernate/model** and **Course.hbm.xml** in **src/main/resources/com/website/hibernate/**

```
package com.website.hibernate.model;

public class Course implements java.io.Serializable {
	private static final long serialVersionUID = 1L;
```

```
        private Long id;
        private String courseName;

        //TODO: generate constructor & getter and setter methods

}
```

```
<?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="com.website.hibernate.model.Course" table="course" catalog="website">
        <id name="id" type="long">
            <column name="COURSE_ID" />
            <generator class="identity" />
        </id>
        <property name="courseName" type="string">
            <column name="COURSE_NAME" length="20" not-null="true"/>
        </property>
    </class>
</hibernate-mapping>
```

Update **Student.java** class in package **src/main/java/com/website/hibernate/model** and **Student.hbm.xml** in **src/main/resources/com/website/hibernate/**

```
package com.website.hibernate.model;

import java.util.List;

public class Student implements java.io.Serializable {
        private static final long serialVersionUID = 1L;

        private Long id;
        private String firstName;
        private String lastName;
        private Integer age;
        private Address address;
        private List<Course> courses;

        //TODO: generate constructor & getter and setter methods

}
```

```
<?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="com.website.hibernate.model.Student" table="student" catalog="website">
        <id name="id" type="long">
            <column name="STUDENT_ID" />
            <generator class="identity" />
        </id>
        <property name="firstName" type="string">
            <column name="FIRST_NAME" length="20" not-null="true"/>
        </property>
        <property name="lastName" type="string">
            <column name="LAST_NAME" length="20" not-null="true"/>
        </property>
        <property name="age" type="integer">
            <column name="AGE" length="20" not-null="true"/>
        </property>
        <one-to-one name="address" class="com.website.hibernate.model.Address"
                        cascade="save-update" lazy="false"></one-to-one>
        <list name="courses" table="STUDENT_COURSE" cascade="all">
                <key column="STUDENT_ID" />
                <list-index column="idx" />
                <many-to-many column="COURSE_ID" class="com.website.hibernate.model.Course" />
        </list>
    </class>
</hibernate-mapping>
```

Using Annotation Mapping

Create **Course.java** class in package **src/main/java/com/website/hibernate/entities** and Update **Student.java** class in package **src/main/java/com/website/hibernate/entities**

```
package com.website.hibernate.entities;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;

import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "course", catalog = "website")
public class Course implements java.io.Serializable {
	private static final long serialVersionUID = 1L;

	private Long id;
	private String courseName;

	//TODO: generate constructor for Course

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "COURSE_ID", unique = true, nullable = false)
	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	@Column(name = "COURSE_NAME", length = 20)
	public String getCourseName() {
		return courseName;
	}
	public void setCourseName(String courseName) {
		this.courseName = courseName;
	}
}
```

```
package com.website.hibernate.entities;

import static javax.persistence.GenerationType.IDENTITY;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinTable;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToMany;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name = "student", catalog = "website")
public class Student implements java.io.Serializable {
	private static final long serialVersionUID = 1L;

	private Long id;
	private String firstName;
	private String lastName;
	private Integer age;
	private Address address;
	private List<Course> courses;
```

```
        //TODO: generate constructor for Student

        @Id
        @GeneratedValue(strategy = IDENTITY)
        @Column(name = "STUDENT_ID", unique = true, nullable = false)
        public Long getId() {
                return id;
        }
        public void setId(Long id) {
                this.id = id;
        }

        @Column(name = "FIRST_NAME", length = 20)
        public String getFirstName() {
                return firstName;
        }
        public void setFirstName(String firstName) {
                this.firstName = firstName;
        }

        @Column(name = "LAST_NAME", length = 20)
        public String getLastName() {
                return lastName;
        }
        public void setLastName(String lastName) {
                this.lastName = lastName;
        }

        @Column(name = "AGE")
        public Integer getAge() {
                return age;
        }
        public void setAge(Integer age) {
                this.age = age;
        }

        @OneToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
        @PrimaryKeyJoinColumn
        public Address getAddress() {
                return address;
        }
        public void setAddress(Address address) {
                this.address = address;
        }

        @ManyToMany(cascade = CascadeType.ALL)
        @JoinTable(name = "STUDENT_COURSE", joinColumns = { @JoinColumn(name = "STUDENT_ID") },
inverseJoinColumns = { @JoinColumn(name = "COURSE_ID") })
        public List<Course> getCourses() {
                return courses;
        }
        public void setCourses(List<Course> courses) {
                this.courses = courses;
        }
}
```

- **One – to – Many / Many – to – One**
  *(base on **one – to – one** and **many – to – many** for practice yourselft)*
  For ex: Object **Course** *contain* ***List<HomeWork>*** *as properties and Object* ***HomeWork*** *contain* ***Course*** *as property also.*

Hibernate with MySQL (3)

1. HQL – Hibernate Query Language

the syntax is quite similar to database SQL language.

The main difference between is HQL uses **class name** instead of **table name**, and **property names** instead of **column name**.

❖ **SELECT**

```
Session session = HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();

Query query = session.createQuery("from Address as A where A.city = 'HCM'");
List list = query.list(); // return list objects

Query query1 = session.createQuery("select S from Student as S where S.firstName = :fName");
Query1.setParameter("fName", "Phuong");
Student student = (Student) query1.uniqueResult(); // return single object

session.getTransaction().commit();
```

❖ **UPDATE**

```
Query query = session.createQuery("update Student set firstName = :fName, lastName = :lName where id = :id");
query.setParameter("fName", "Phuong1");
query.setParameter("lName", "Mai1");
query.setParameter("id", 1);
int result = query.executeUpdate(); // return number of record has been updated
```

❖ **DELETE**

```
Query query = session.createQuery("delete Student where id = :id");
query.setParameter("id", 1);
int result = query.executeUpdate(); // return number of record has been deleted
```

❖ **INSERT** – HQL only support insert from another table**: INSERT INTO … SELECT ….**

```
Query query = session.createQuery("insert into Address(street, district, city) select tmp.street,
tmp.district, tmp.city from Address_temp as tmp");
int result = query.executeUpdate(); // return number of record has been inserted
```

❖ **PAGINATION**

```
Query query = session.createQuery("from Address");
query.setFirstResult(20); // set offset
query.setMaxResults(10); // set limit
List list = query.list();
```

❖ **AGGREGATE** – HQL support function such as **min(…), max(…), avg(…), sum(…), count(…), etc**

```
// count total number of Student
Query query = session.createQuery("select count(id) from Student");

// get min age of Student
Query query = session.createQuery("select min(age) from Student ");

// get max number of Student
Query query = session.createQuery("select max(age) from Student ");

// get average age of Student
Query query = session.createQuery("select avg(age) from Student ");

// get total age of Student
Query query = session.createQuery("select sum(age) from Student ");
```

2. HCQL – Hibernate Criteria Query Language

```
// get all the records
Criteria criteria = session.createCriteria(Student.class);
```

```
List students = criteria.list();
// pagination with limit and offset
Criteria criteria = session.createCriteria(Student.class);
criteria.setFirstResult(0);
criteria.setMaxResults(10);
List students = criteria.list();

// set order ASC
Criteria criteria = session.createCriteria(Student.class);
criteria.addOrder(Order.asc("age"));
List students = criteria.list();

// set order DES
Criteria criteria = session.createCriteria(Student.class);
criteria.addOrder(Order.desc("age"));
List students = criteria.list();

// set restrictions
Criteria criteria = session.createCriteria(Student.class);
criteria.add(Restrictions.between("age", 10, 20)); // set between constraint

criteria.add(Restrictions.like("firstName", "Phuong")); // set like constraint

criteria.add(Restrictions.eq("age", 29)); // set equal constraint
criteria.add(Restrictions.ne("age", 20)); // set not equal constraint

criteria.add(Restrictions.lt("age", 20)); // set less than constraint
criteria.add(Restrictions.le("age", 20)); // set less than or equal constraint

criteria.add(Restrictions.gt("age", 20)); // set greater than constraint
criteria.add(Restrictions.ge("age", 20)); // set greater than or equal constraint
List students = criteria.list();
```

3. **Native SQL** – You can use native SQL to express database queries

```
// Entity queries
SQLQuery query = session.createSQLQuery("SELECT * FROM STUDENT");
query.addEntity(Student.class);
List students = query.list();

// Scalar queries
SQLQuery query = session.createSQLQuery("SELECT * FROM STUDENT as s, ADDRESS as a where s.STUDENT_ID = a.ID");
query.setResultTransformer(Criteria.ALIAS_TO_ENTITY_MAP);
List students = query.list();

// Named SQL queries
SQLQuery query = session.createSQLQuery("SELECT * FROM STUDENT WHERE id = :id");
query.addEntity(Student.class);
query.setParameter("id", 1);
List students = query.list();
```

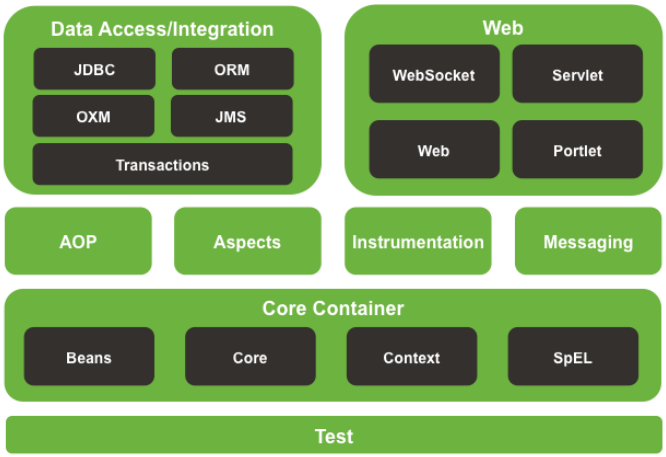
4. **Practice**

   Create **StudentDao, AddressDao, CourseDao** and **HomeWorkDao** classes extends from **abstract Dao** as below; and write the implementation for these method from **abstract Dao**

```
public abstract class Dao<T> {
        public abstract List<T> getAll();

        public abstract T get(Long id);

        public abstract T add(T t);

        public abstract Boolean update(T t);

        public abstract Boolean delete(T t);
}
```

Spring – Spring MVC

1. Spring framework and SpringMVC

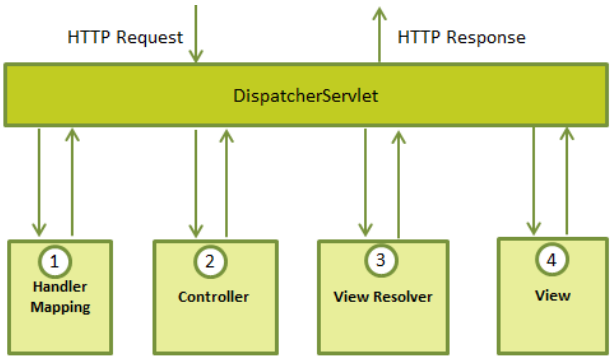


The Spring Framework consists of features organized into about 20 modules. These modules are grouped into Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation, Messaging, and Test, as shown in the following diagram.

- ✓ **Modularity** – Spring has layered architecture. Use what you need and leave you don't need now
- ✓ **Dependency Injection and Inversion of Control**: removes the dependency from the programming code.
- ✓ **Spring web MVC** is helpful for develop web application. We can develop web application to fast
- ✓ **Spring bean and spring context** are helpful for initial and manage bean
- ✓ Open source and no vendor lock-in

2. The Life-Cycle in Spring MVC

The Spring Web MVC framework is designed around a **DispatcherServlet** that handles all the HTTP requests and responses.



(1) After receiving an HTTP request, **DispatcherServlet** consults the **HandlerMapping** to call the appropriate Controller.
(2) The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the **DispatcherServlet**.
(3) The **DispatcherServlet** will take help from **ViewResolver** to pick-up the defined view for the request.
(4) Once view is finalized, The **DispatcherServlet** passes the model data to the view which is finally rendered on the browser.

3. IoC – Inversion of Control

Inversion of Control is a design pattern that removes the dependency from the programming code. That means we have inverted the control of creating the object. We provide metadata to the IOC container either by XML file or annotation and container will create the object for us.

In Spring support 3 types of IoC containers: **BeanFactory**, **ApplicationContext** and WebApplicationContext.

4. DI – Dependency Injection

Dependency Injection (DI) is a sub-type of IoC and It's implemented by **Constructor Injection** and **Setter Injection**. Dependency Injection makes our programming code loosely coupled and easier for testing.

❖ **Constructor Injection**: Dependencies are provided as constructor parameters.

```
<bean id="Student" class="com.website.springmvc.Student">
        <constructor-arg>
                <ref bean="Address" />
        </constructor-arg>
</bean>
<bean id="Address" class="com.website.springmvc.Address" />
```

❖ Setter Injection: Dependencies are assigned through JavaBeans properties.

```
<bean id="Student" class="com.website.springmvc.Student">
        <property name="address" ref="Address" />
</bean>
<bean id="Address" class="com.website.spingmvc.Address" />
```

5. **What is @Component, @Controller and @Service** annotations in Spring?

❖ **@Component** is used to indicate that a class is a component. These classes are used for auto detection and configured as bean, when annotation based configurations are used

❖ **@Controller** is a specific type of component, used in MVC applications and mostly used with **@RequestMapping** annotation.

❖ **@Service** is used to indicate that a class is a **Service**. Usually the business facade classes that provide some services are annotated with this.

❖ **@Repository** is used to indicate that a component is used as **Repository**. We can apply this annotation with DAO pattern implementation classes.

6. **Practice: Create maven project from Eclipse IDE and add SpringMVC dependency**

Update pom.xml

```
...
<properties>
        <!-- Generic properties -->
        <java.version>1.7</java.version>

        <!-- Web -->
        <jsp.version>2.2</jsp.version>
        <jstl.version>1.2</jstl.version>
        <servlet.version>3.1.0</servlet.version>

        <!-- Spring -->
        <spring-framework.version>4.1.8.RELEASE</spring-framework.version>

        <!-- Logging -->
        <slf4j.version>1.7.5</slf4j.version>

        <!-- Skip Test -->
        <skipTests>false</skipTests>
</properties>

<dependencies>
        <!-- Spring MVC -->
        <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-webmvc</artifactId>
                <version>${spring-framework.version}</version>
        </dependency>
        <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-context</artifactId>
                <version>${spring-framework.version}</version>
        </dependency>

        <!-- Other Web dependencies -->
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>jstl</artifactId>
                <version>${jstl.version}</version>
        </dependency>
        <dependency>
```

```
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>${servlet.version}</version>
                <scope>provided</scope>
        </dependency>
        <dependency>
                <groupId>javax.servlet.jsp</groupId>
                <artifactId>jsp-api</artifactId>
                <version>${jsp.version}</version>
                <scope>provided</scope>
        </dependency>

        <!-- Logging with SLF4J -->
        <dependency>
                <groupId>org.slf4j</groupId>
                <artifactId>slf4j-api</artifactId>
                <version>${slf4j.version}</version>
                <scope>compile</scope>
        </dependency>
</dependencies>
<build>
        <plugins>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <configuration>
                                <source>${java.version}</source>
                                <target>${java.version}</target>
                                <encoding>UTF-8</encoding>
                        </configuration>
                </plugin>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-surefire-plugin</artifactId>
                        <version>2.18.1</version>
                        <configuration>
                                <skipTests>${skipTests}</skipTests>
                        </configuration>
                </plugin>
        </plugins>
</build>
...
```

7. SpringMVC Configuration

❖ **With XML configuration**

Create **web.xml** (if not) and **dispatcher-servlet.xml** in folder **src/main/webapp/WEB-INF**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID" version="3.0">

        <display-name>Spring MVC Web Application</display-name>
        <servlet>
                <servlet-name>dispatcher</servlet-name>
                <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                <servlet-name>dispatcher</servlet-name>
                <url-pattern>/</url-pattern>
        </servlet-mapping>
        <session-config>
                <session-timeout>60</session-timeout>
        </session-config>
</web-app>
```

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xsi:schemaLocation="
```

```
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.2.xsd">

        <context:component-scan base-package="com.website.springmvc" />
        <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
                <property name="prefix">
                        <value>/views/</value>
                </property>
                <property name="suffix">
                        <value>.jsp</value>
                </property>
        </bean>
        <mvc:resources mapping="/resources/**" location="/resources/" />
        <mvc:annotation-driven />
</beans>
```

❖ **With Annotation configuration**

Create **web.xml** (if not) in folder **src/main/webapp/WEB-INF**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns="http://java.sun.com/xml/ns/javaee"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        id="WebApp_ID" version="3.0">

        <display-name>Spring MVC Web Application</display-name>
        <session-config>
                <session-timeout>60</session-timeout>
        </session-config>
</web-app>
```

Create **DispatcherServletInitializer.java and SpringWebConfig.java** in package **com.website.springmvc.config**

```
public class DispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

        private static final String DISPATCH_OPTIONS_REQUEST = "dispatchOptionsRequest";

        @Override
        protected Class<?>[] getRootConfigClasses() {

                return new Class<?>[] { SpringWebConfig.class };
        }

        @Override
        protected Class<?>[] getServletConfigClasses() {

                return new Class<?>[] {};
        }

        @Override
        protected String[] getServletMappings() {

                return new String[] { "/" };
        }

        @Override
        protected void customizeRegistration(Dynamic registration) {

                registration.setInitParameter(DISPATCH_OPTIONS_REQUEST, Boolean.TRUE.toString());
        }
}
```

```
@EnableWebMvc // mvc:annotation-driven
@Configuration
@ComponentScan({ "com.website.springmvc" })
public class SpringWebConfig extends WebMvcConfigurerAdapter {
        @Override
        public void addResourceHandlers(ResourceHandlerRegistry registry) {
                registry.addResourceHandler("/resources/**").addResourceLocations("/resources/");
        }
```

```
	@Bean
	public InternalResourceViewResolver viewResolver() {
		InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();
		viewResolver.setViewClass(JstlView.class);
		viewResolver.setPrefix("/views/");
		viewResolver.setSuffix(".jsp");
		return viewResolver;
	}
}
```

8. Controller & Mapping

Create **LoginController.java** and **HomeController.java** in package **com.website.springmvc.controller**

```
@Controller
public class LoginController {
	@RequestMapping(value = "/", method = RequestMethod.GET)
	public String onload() {
		return "login";
	}
	@RequestMapping(value = "/login", method = RequestMethod.POST)
	public String login(@RequestParam String userName, @RequestParam String password) {
		if ("admin".equalsIgnoreCase(userName) && "admin".equalsIgnoreCase(password))
			return "redirect:/home/" + userName;
		return "login";
	}
}
```

```
@Controller
@RequestMapping(value = "/home")
public class HomeController {
	@RequestMapping(value = "/{name}", method = RequestMethod.GET)
	public ModelAndView printWelcome(@PathVariable("name") String name) {
		ModelAndView model = new ModelAndView();
		model.setViewName("home");
		model.addObject("name", name);
		return model;
	}
}
```

9. **JSP Views:** Create **login.jsp** and **home.jsp** as below

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<h2>Login Page</h2>
	<form action="login" method="post">
		<div><input type="text" name="userName" size="20" /></div>
		<div><input type="password" name="password" size="20" /></div>
		<div><input type="submit" value="Login" /></div>
	</form>
</body>
</html>
```

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
	<h2>Home</h2>
	Wellcome: ${name}
</body>
</html>
```

Spring MVC – Hibernate

1. **Create maven project from Eclipse IDE and add SpringMVC dependency**
   (Review last session)
2. **Add Hibernate and Mysql dependency**
   Update pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.website.springmvc</groupId>
        <artifactId>springmvc-hibernate</artifactId>
        <packaging>war</packaging>
        <version>0.0.1-SNAPSHOT</version>
        <name>Spring MVC - Hibernate Webapp</name>
        <url>http://maven.apache.org</url>

        <properties>
                <!-- Generic properties -->
                <jdk.version>1.7</jdk.version>

                <!-- Spring -->
                <spring.version>4.2.1.RELEASE</spring.version>

                <!-- Web -->
                <jstl.version>1.2</jstl.version>
                <servlet.version>3.1.0</servlet.version>

                <!-- Other -->
                <jackson.version>2.5.4</jackson.version>
                <hibernate.version>4.3.5.Final</hibernate.version>
                <org.aspectj-version>1.7.4</org.aspectj-version>
                <mysql.connector.version>5.1.31</mysql.connector.version>
        </properties>
        <dependencies>
                <!-- Jackson -->
                <dependency>
                        <groupId>org.codehaus.jackson</groupId>
                        <artifactId>jackson-mapper-asl</artifactId>
                        <version>1.9.10</version>
                </dependency>

                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-databind</artifactId>
                        <version>${jackson.version}</version>
                </dependency>

                <dependency>
                        <groupId>com.fasterxml.jackson.core</groupId>
                        <artifactId>jackson-core</artifactId>
                        <version>${jackson.version}</version>
                </dependency>

                <!-- Spring MVC -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-core</artifactId>
                        <version>${spring.version}</version>
                </dependency>
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-webmvc</artifactId>
                        <version>${spring.version}</version>
                </dependency>

                <!-- Spring ORM -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-orm</artifactId>
                        <version>${spring.version}</version>
```

```
		</dependency>

		<!-- Hibernate -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>${hibernate.version}</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-entitymanager</artifactId>
			<version>${hibernate.version}</version>
		</dependency>

		<!-- Other Web dependencies -->
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>3.8.1</version>
			<scope>test</scope>
		</dependency>
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>${servlet.version}</version>
		</dependency>

		<!-- Apache Commons DBCP -->
		<dependency>
			<groupId>commons-dbcp</groupId>
			<artifactId>commons-dbcp</artifactId>
			<version>1.4</version>
		</dependency>

		<!-- AspectJ -->
		<dependency>
			<groupId>org.aspectj</groupId>
			<artifactId>aspectjrt</artifactId>
			<version>${org.aspectj-version}</version>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>${mysql.connector.version}</version>
		</dependency>

		<dependency>
			<groupId>jstl</groupId>
			<artifactId>jstl</artifactId>
			<version>${jstl.version}</version>
		</dependency>
	</dependencies>
	<build>
		<finalName>springmvc-hibernate</finalName>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.1</version>
				<configuration>
					<source>${jdk.version}</source>
					<target>${jdk.version}</target>
				</configuration>
			</plugin>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-war-plugin</artifactId>
				<configuration>
					<failOnMissingWebXml>false</failOnMissingWebXml>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. SpringMVC Configuaration

(Review last session)

Update **DispatcherServletInitializer.java**

```
public class DispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
...
	@Override
	protected Class<?>[] getRootConfigClasses() {

		return new Class<?>[] { SpringWebConfig.class, SpringDatabaseConfig.class };
	}
...
```

4. Hibernate Configuration

```
@EnableTransactionManagement
@Configuration
public class SpringDatabaseConfig extends WebMvcConfigurerAdapter {
	@Bean
	public LocalSessionFactoryBean sessionFactory(BasicDataSource dataSource) {
		LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
		sessionFactory.setDataSource(dataSource);
		sessionFactory.setPackagesToScan(new String[] { "com.website.springmvc.entities" });
		sessionFactory.setAnnotatedClasses(Student.class, Address.class, Course.class,
HomeWork.class);
		sessionFactory.setHibernateProperties(hibernateProperties());
		return sessionFactory;
	}

	private Properties hibernateProperties() {
		Properties properties = new Properties();
		properties.put("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
		properties.put("hibernate.show_sql", true);
		properties.put("hibernate.format_sql", true);
		properties.put("hibernate.hbm2ddl.auto", "update");
		return properties;
	}

	@Bean(name = "dataSource")
	public BasicDataSource getDataSource() {
		BasicDataSource dataSource = new BasicDataSource();
		dataSource.setDriverClassName("com.mysql.jdbc.Driver");
		dataSource.setUrl("jdbc:mysql://localhost:3306/website");
		dataSource.setUsername("root");
		dataSource.setPassword("admin");
		return dataSource;
	}

	@Bean
	public HibernateTransactionManager transactionManager(SessionFactory s) {
		HibernateTransactionManager txManager = new HibernateTransactionManager();
		txManager.setSessionFactory(s);
		return txManager;
	}
}
```

5. Add Student entity, Dao and StudentDao

*(Review **Hibernate with MySQL (3)** session)*

```
@Repository
public class StudentDao extends Dao<Student> {

	@Autowired
	private SessionFactory sessionFactory;

	@Override
	public List<Student> getAll() {
		Session session = this.sessionFactory.getCurrentSession();
		List<Student> students = session.createQuery("from Student").list();
		return students;
	}
```

```
	@Override
	public Student get(Long id) {
		Session session = this.sessionFactory.getCurrentSession();
		return (Student) session.get(Student.class, new Long(id));
	}

	@Override
	public Student add(Student student) {
		Session session = this.sessionFactory.getCurrentSession();
		session.save(student);
		return student;
	}

	@Override
	public Boolean update(Student student) {
		Session session = this.sessionFactory.getCurrentSession();
		try {
			session.update(student);
			return Boolean.TRUE;
		} catch (Exception e) {
			return Boolean.FALSE;
		}
	}

	@Override
	public Boolean delete(Student student) {
		Session session = this.sessionFactory.getCurrentSession();
		if (null != student) {
			try {
				session.delete(student);
				return Boolean.TRUE;
			} catch (Exception e) {
				return Boolean.FALSE;
			}
		}
		return Boolean.FALSE;
	}

	@Override
	public Boolean delete(Long id) {
		Session session = this.sessionFactory.getCurrentSession();
		Student student = (Student) session.load(Student.class, new Long(id));
		if (null != student) {
			session.delete(student);
			return Boolean.TRUE;
		}
		return Boolean.FALSE;
	}
}
```

6. Create StudentService.java and StudentController.java

```
@Transactional
@Service
public class StudentService {
	@Autowired
	Dao<Student> studentDao;

	public List<Student> getAll() {
		return studentDao.getAll();
	}

	public Student get(Long id) {
		return studentDao.get(id);
	}

	public Student add(Student student) {
		return studentDao.add(student);
	}

	public Boolean update(Student student) {
		return studentDao.update(student);
	}
```

```
	public Boolean delete(Student student) {
		return studentDao.delete(student);
	}

	public Boolean delete(Long id) {
		return studentDao.delete(id);
	}
}
```

```
@Controller
@RequestMapping(value = "/views/student")
public class StudentController {
	@Autowired
	private StudentService studentService;

	@RequestMapping(value = "/", method = RequestMethod.GET, headers = "Accept=application/json")
	public ModelAndView getStudents(ModelAndView model) {
		model.setViewName("students");
		model.addObject("students", studentService.getAll());
		return model;
	}

	@RequestMapping(value = "/addStudent", method = RequestMethod.GET)
	public ModelAndView addStudent(ModelAndView model) {
		model.setViewName("studentDetail");
		model.addObject("student", new Student());
		return model;
	}

	@RequestMapping(value = "/save", method = RequestMethod.POST)
	public String save(@ModelAttribute("student") Student student) {
		if (student.getId() == null) {
			studentService.add(student);
		} else {
			studentService.update(student);
		}
		return "redirect:/views/student/";
	}

	@RequestMapping(value = "/getStudent", method = RequestMethod.GET)
	public ModelAndView getStudent(@RequestParam("id") Long id, @RequestParam("mode") String mode,
ModelAndView model) {
		model.setViewName("studentDetail");
		model.addObject("student", studentService.get(id));
		model.addObject("mode", mode);
		return model;
	}

	@RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
	public void delete(@PathVariable("id") Long id) {
		studentService.delete(id);
	}
}
```

7. Create View: students.jsp and studentDetail.jsp

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" href="<c:url value="/resources/css/bootstrap.min.css" />">
<script type="text/javascript" src="<c:url value="/resources/js/jquery.1.10.2.min.js" />"></script>
<script type="text/javascript" src="<c:url value="/resources/js/bootstrap.min.js" />"></script>
<script type="text/javascript" src="<c:url value="/resources/js/students.js" />"></script>
<title><spring:message code="student.label" /></title>
</head>
<body>
   <div class="panel panel-default">
```

```
      <div id="abc" class="panel-heading h3 text-center">
         <spring:message code="student.header" />
      </div>
      <div class="panel-body">
         <table class="table table-striped">
            <thead>
               <tr>
                  <th><spring:message code="student.table.id" /></th>
                  <th><spring:message code="student.table.name" /></th>
                  <th><spring:message code="student.table.age" /></th>
                  <th><spring:message code="student.table.address" /></th>
                  <th><spring:message code="student.table.action" /></th>
               </tr>
            <thead>
            <c:choose>
                 <c:when test="${!empty students}">
                  <c:forEach items="${students}" var="student">
                     <tr>
                        <td>${student.id}</td>
                        <td>${student.firstName} ${student.lastName}</td>
                        <td>${student.age}</td>
                        <td>${student.address.street}, ${student.address.district}, ${student.address.city}</td>
                        <td>
                           <button class="btn btn-info" onclick="getStudent(${student.id}, 'VIEW');">
                              <spring:message code="student.btn.view" />
                           </button>
                           <button class="btn btn-primary" onclick="getStudent(${student.id}, 'EDIT');">
                              <spring:message code="student.btn.edit" />
                           </button>
                           <button class="btn btn-danger" onclick="deleteStudent(${student.id});">
                              <spring:message code="student.btn.delete" />
                           </button>
                        </td>
                     </tr>
                  </c:forEach>
               </c:when>
               <c:otherwise>
                  <tr>
                     <th colspan="5" class="text-center"><spring:message code="student.table.empty" /></th>
                  </tr>
               </c:otherwise>
            </c:choose>
            <tr>
               <th colspan="5">
                  <button onclick="location.href='addStudent'" class="btn btn-primary">
                     <spring:message code="student.btn.add" />
                  </button>
               </th>
            </tr>
         </table>
      </div>
   </div>
</body>
</html>
```

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" href="<c:url value="/resources/css/bootstrap.min.css" />">
<script type="text/javascript" src="<c:url value="/resources/js/jquery.1.10.2.min.js" />"></script>
<script type="text/javascript" src="<c:url value="/resources/js/bootstrap.min.js" />"></script>
<title><spring:message code="student.detail.label" /></title>
</head>
<body>
   <div class="panel panel-default">
      <div class="panel-heading h3 text-center"><spring:message code="student.detail.label" /></div>
      <div class="panel-body">

         <spring:message code="student.detail.fName.plhd"  var="fName_plhd" />
```

```
<spring:message code="student.detail.lName.plhd" var="lName_plhd" />
<spring:message code="student.detail.age.plhd" var="age_plhd" />
<spring:message code="student.detail.street.plhd" var="street_plhd" />
<spring:message code="student.detail.district.plhd" var="district_plhd" />
<spring:message code="student.detail.city.plhd" var="city_plhd" />

<form:form class="form-horizontal" action="./save" method="post" modelAttribute="student">
   <form:input path="id" type="hidden"/>
   <div class="form-group">
      <label class="control-label col-sm-4" for="fName">
         <spring:message code="student.detail.fName" />
      </label>
      <div class="col-sm-6">
         <form:input path="firstName" type="text" class="form-control" id="fName"
                     name="fName" placeholder="${fName_plhd}" readonly="${mode == 'VIEW'}"/>
      </div>
   </div>
   <div class="form-group">
      <label class="control-label col-sm-4" for="lName">
         <spring:message code="student.detail.lName" />
      </label>
      <div class="col-sm-6">
         <form:input path="lastName" type="text" class="form-control" id="lName"
                     name="lName" placeholder="${lName_plhd}" readonly="${mode == 'VIEW'}"/>
      </div>
   </div>
   <div class="form-group">
      <label class="control-label col-sm-4" for="age">
         <spring:message code="student.detail.age" />
      </label>
      <div class="col-sm-6">
         <form:input path="age" type="text" class="form-control" id="age"
                     name="age" placeholder="${age_plhd}" readonly="${mode == 'VIEW'}"/>
      </div>
   </div>
   <form:input path="address.id" type="hidden"/>
   <div class="form-group">
      <label class="control-label col-sm-4" for="street">
         <spring:message code="student.detail.street" />
      </label>
      <div class="col-sm-6">
         <form:input path="address.street" type="text" class="form-control" id="street"
                     name="street" placeholder="${street_plhd}" readonly="${mode == 'VIEW'}"/>
      </div>
   </div>
   <div class="form-group">
      <label class="control-label col-sm-4" for="district">
         <spring:message code="student.detail.district" />
      </label>
      <div class="col-sm-6">
         <form:input path="address.district" type="text" class="form-control" id="district"
                     name="district" placeholder="${district_plhd}" readonly="${mode == 'VIEW'}"/>
      </div>
   </div>
   <div class="form-group">
      <label class="control-label col-sm-4" for="city">
         <spring:message code="student.detail.city" />
      </label>
      <div class="col-sm-6">
         <form:input path="address.city" type="text" class="form-control" id="city"
                     name="city" placeholder="${city_plhd}" readonly="${mode == 'VIEW'}"/>
      </div>
   </div>
   <div class="form-group">
      <div class="col-sm-offset-4 col-sm-6">
         <c:if test = "${mode == 'VIEW'}">
            <button disabled="disabled" type="submit" class="btn btn-primary">
               <spring:message code="student.detail.btn.save" />
            </button>
         </c:if>
         <c:if test = "${mode != 'VIEW'}">
            <button type="submit" class="btn btn-primary">
               <spring:message code="student.detail.btn.save" />
            </button>
```

```
                </c:if>
                <button type="button" onclick="location.href='./'" class="btn btn-default">
                   <spring:message code="student.detail.btn.cancel" />
                </button>
             </div>
          </div>
       </form:form>
    </div>
  </div>
</body>
</html>
```

8. Practice

Base on **StudentController** and **StudentSevice**, create **controllers** and **services** for **Address, Course, Homework**

JSTL Tag libs

1. JSTL – JSP Standard Tag Library

Core tags	<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
Function tags	<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
Formatting tags	<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
XML tags	<%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
SQL tags	<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

2. JSTL Core Tags

c:out	It display the result of an expression, similar to the way <%=...%> tag work. <c:out value="${'Hello World'}"/>
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var', a Reader in 'varReader' or the page. <c:import var="data" url="http://www.google.com"/> <c:out value="${data}"/>
c:set	It sets the result of an expression under evaluation in a 'scope' variable. <c:set var="Income" scope="session" value="${4000*4}" /> <c:out value="${Income}" />
c:remove	It is used for removing the specified scoped variable from a particular scope. <c:remove var="income" /> <c:out value="${income}" />
c:catch	It is used for Catches any Throwable exceptions that occurs in the body. <c:catch var="catchtheException"> <%int x = 2 / 0;%> </c:catch> <c:if test="${catchtheException != null}"> The type of exception is : ${catchtheException} There is an exception: ${catchtheException.message} </c:if>
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:choose	It is the simple conditional tag that includes its body content if the evaluated condition is true. <c:choose> <c:when test="${income <= 1000}"> Income is not good. </c:when> <c:when test="${income > 10000}"> Income is very good. </c:when> <c:otherwise> Income is undetermined... </c:otherwise> </c:choose>

c:when	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
c:otherwise	It iterates over tokens which is separated by the supplied delimeters.
c:forEach	It adds a parameter in a containing 'import' tag's URL. <c:forEach var="j" begin="1" end="3"> Item <c:out value="${j}" /> </c:forEach>
c:forTokens	It redirects the browser to a new URL and supports the context-relative URLs. <c:forTokens items="Rahul-Nakul-Rajesh" delims="-" var="name"> <c:out value="${name}" /> </c:forTokens>
c:param	It creates a URL with optional query parameters. <c:url value="/index1.jsp" var="completeURL"> <c:param name="trackingId" value="786" /> <c:param name="user" value="Nakul" /> </c:url>
c:redirect	It display the result of an expression, similar to the way <%=...%> tag work. <c:redirect url="http://facebook.com" />
c:url	It Retrives relative or an absolute URL and display the contents to either a String in 'var', a Reader in 'varReader' or the page. <c:url value="/home.jsp" />

3. JSTL Function Tags

fn:contains()	It is used to test if an input string containing the specified substring in a program. <c:if test="${fn:contains('Hello World', 'Hell')}"> Found Hell string </c:if>
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith()	It is used to test if an input string ends with the specified suffix.
fn:escapeXml()	It escapes the characters that would be interpreted as XML markup. <c:set var="str" value="Hello <xyz>World</xyz>"/> ${str} ${fn:escapeXml(str)}
fn:indexOf()	It returns an index within a string of first occurrence of a specified substring.
fn:trim()	It removes the blank spaces from both the ends of a string.
fn:startsWith()	It is used for checking whether the given string is started with a particular string value.
fn:split()	It splits the string into an array of substrings.
fn:toLowerCase()	It converts all the characters of a string to lower case.
fn:toUpperCase()	It converts all the characters of a string to upper case.
fn:substring()	It returns the subset of a string according to the given start and end position.

fn:substringAfter()	It returns the subset of string after a specific substring.
fn:substringBefore()	It returns the subset of string before a specific substring. <c:set var="str" value="Hi, my name is PhuongMai"/> ${fn:substringAfter(str, "is")} ${fn:substringBefore(str, "is")}
fn:length()	It returns the number of characters inside a string, or the number of items in a collection.
fn:replace()	It replaces all the occurrence of a string with another string sequence.

4. JSTL Formatting Tags

fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting. <c:forEach var="zone" items="<%=java.util.TimeZone.getAvailableIDs()%>"> <fmt:timeZone value="${zone}"> <fmt:formatDate value="<%=new java.util.Date()%>" timeZone="${zn}" type="both" timeStyle="long" dateStyle="long" /> </fmt:timeZone> </c:forEach>
fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.
fmt:formatNumber	It is used to format the numerical value with specific format or precision. <c:set var="Amount" value="9850.14115" /> 1: <fmt:formatNumber value="${Amount}" type="currency" /> 2: <fmt:formatNumber type="number" groupingUsed="true" value="${Amount}" /> 3: <fmt:formatNumber type="number" maxIntegerDigits="3" value="${Amount}" /> 4: <fmt:formatNumber type="number" maxFractionDigits="6" value="${Amount}" /> 5: <fmt:formatNumber type="percent" maxIntegerDigits="4" value="${Amount}" /> 6: <fmt:formatNumber type="number" pattern="###.###$" value="${Amount}" />
fmt:parseDate	It parses the string representation of a time and date.
fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
fmt:message	It display an internationalized message.

5. JSTL XML Tags

x:out	Similar to <%= ... > tag, but for XPath expressions.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:set	It is used to sets a variable to the value of an XPath expression.
x:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations.
x:when	It is a subtag of that will include its body if the condition evaluated be 'true'.
x:otherwise	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.

x:if	It is used for evaluating the test XPath expression and if it is true, it will processes its body content.
x:transform	It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.
x:param	It is used along with the transform tag for setting the parameter in the XSLT style sheet.

6. JSTL SQL Tags

sql:setDataSource	It is used for creating a simple data source suitable only for prototyping. `<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/test" user="root" password="1234"/>`

sql:query	It is used for executing the SQL query defined in its sql attribute or the body.
	`<sql:query dataSource="${db}" var="rs">`
	`SELECT * from Students;`
	`</sql:query>`

sql:update	It is used for executing the SQL update defined in its sql attribute or in the tag body.
sql:param	It is used for sets the parameter in an SQL statement to the specified value.
sql:dateParam	It is used for sets the parameter in an SQL statement to a specified java.util.Date value.
sql:transaction	It is used to provide the nested database action with a common connection.

Final Project Suggestion

1. **Store online**
- Cellphones, clothers, books, …
- Multi-language
- Search, filter, auto-complete
- Shopping cart, payment,…
- Admin site

2. **Booking online**
- Ticket booking, Room bookings, …
- Multi-language
- Search, filter, auto-complete
- Shopping cart, payment,…
- Admin site

3. **Cinema booking online**
- Allow seat booking online
- Multi-language
- Search, filter, auto-complete
- Shopping cart, payment,…
- Admin site

4. **Post-trade online**
- Post request, Post trade request, accept/reject trade request
- Multi-language
- Search, filter, auto-complete
- Admin site

5. **Rao vặt**
- Post news, comment, ….
- Multi-language
- Search, filter, auto-complete
- Admin site

6. **Car rental online**
- Multi-language
- Search, filter, auto-complete
- Admin site