spring

INDEX

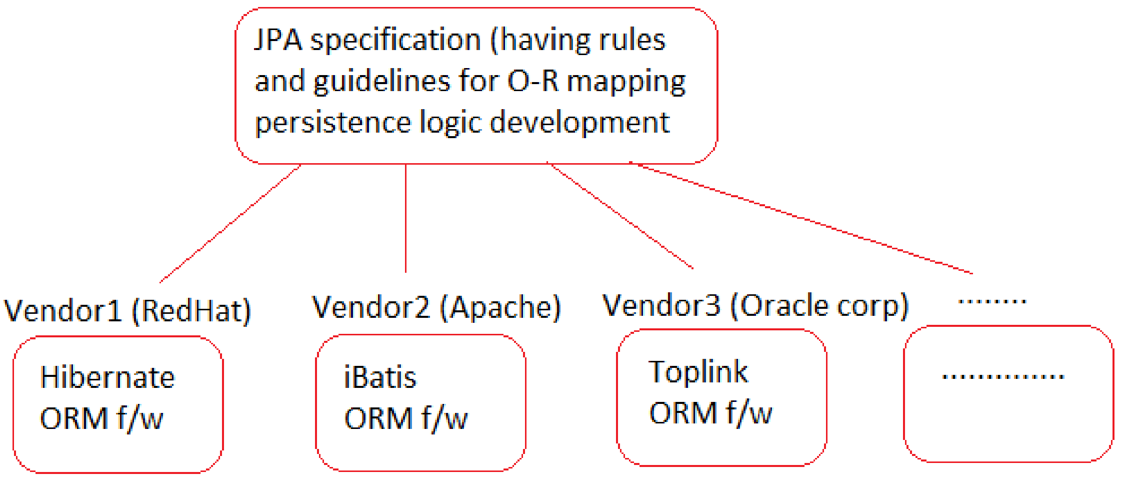
Spring ORM

Introduction

JDBC or Spring JDBC persistence logic:

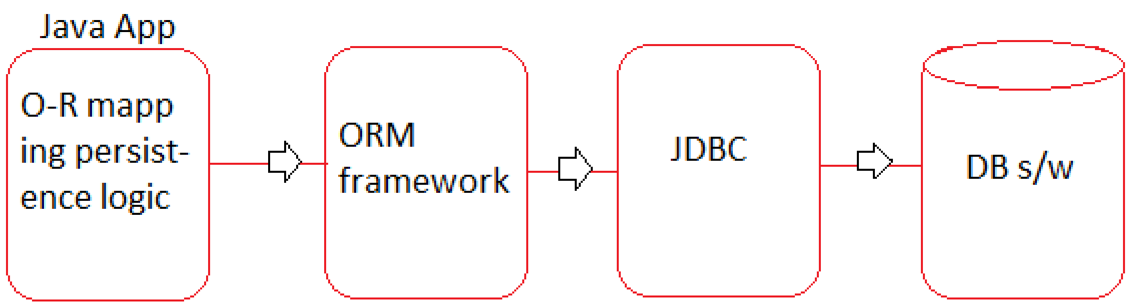
- SQL Queries based persistence logic.
- So, it is DB s/w dependent.
- Changing DB s/w in the middle of development or production is complex.
- Changing DB s/w in Development to Testing UAT Production is complex.

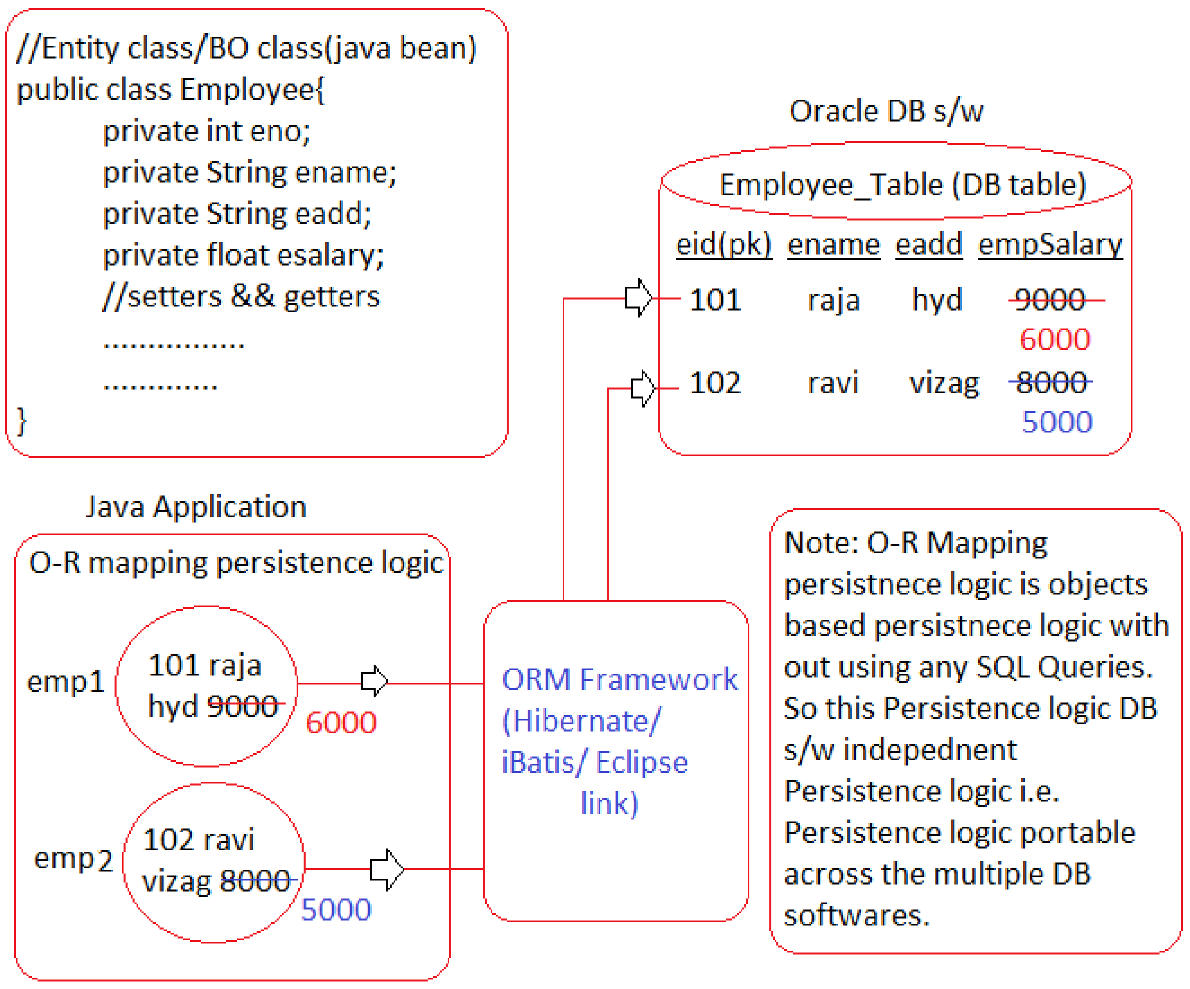
To overcome this problem, use O-R mapping Persistence logic given by JPA specification and implemented through ORM frameworks like hibernate, iBatis, eclipse link and etc.



What is O-R mapping?

- The process of linking DB tables with java classes (BO/ Entity/ Model classes) and DB table columns with the Properties of classes and having synchronization b/w them is called O-R mapping.
- Synchronization means the modification done in objects of java classes will reflect to DB tables records and vice-versa.





```
//Entity class/BO class(java bean)
public class Employee{
        private int eno;
        private String ename;
        private String eadd;
        private float esalary;
        //setters && getters
        ................
        .............
}
```

Oracle DB s/w
Employee_Table (DB table)
eid(pk) ename eadd empSalary
101 raja hyd ~~9000~~ 6000
102 ravi vizag ~~8000~~ 5000
Java Application
O-R mapping persistence logic
emp1
101 raja hyd ~~9000~~ 6000
emp2
102 ravi vizag ~~8000~~ 5000
ORM Framework (Hibernate/ iBatis/ Eclipse link)

Note: O-R Mapping persistnece logic is objects based persistnece logic with out using any SQL Queries. So this Persistence logic DB s/w indepednent Persistence logic i.e. Persistence logic portable across the multiple DB softwares.

Spring ORM

- It is not another O-R framework.
- It is a spring module providing abstraction on multiple ORM frameworks like hibernate, iBatis and etc. to simplify objects-based O-r mapping Persistence logic.
- It supplies multiple Template classes like HibernateTemplate TopLinkTemplate and etc. to avoid boiler plate code of O-R mapping persistence logic.

Plain Hibernate Code to insert record:

a. Create Configuration object (To activate HB f/w)
b. Create Session factory object
c. Create Session object
d. begin Tx

(Common logics)

e. Persistence operation code
....

Application specific logic

f. commit /rollback Tx
g. close session /session factory objects

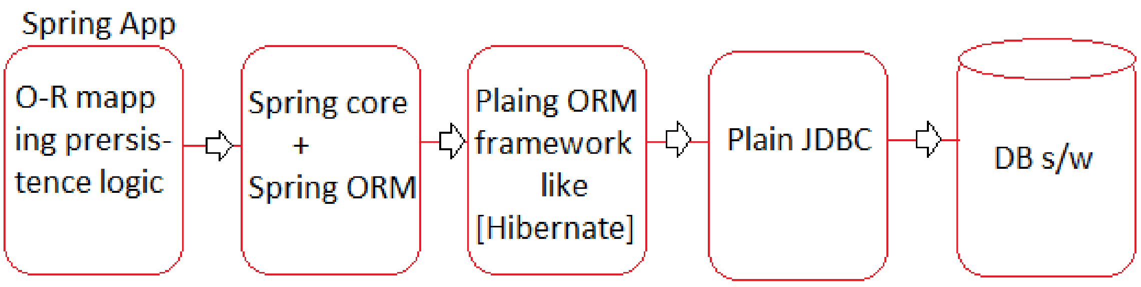
(Common logics)

Note: Common logics = boiler plate code.
[The code that repeats across the multiple applications either with no change or with change is called boiler plate code]

Spring ORM code (having integration with hibernate):

a. Create/inject HibernateTemplate class object.
b. Perform Persistence operations.
   ………………………

Note: No boilerplates code problem.


Spring App
O-R mapping persistence logic
Spring core + Spring ORM
Plaing ORM framework like [Hibernate]
Plain JDBC
DB s/w

Spring ORM Advantages:

a. Avoids boiler plate code by supplying Template classes.
b. Common exception handling (we need not to handle ORM f/w specific exceptions we need to catch and handle only the common DataAccessExcepion).
   Note: Spring JDBC, Spring ORM, Spring data modules throw common exceptions like DataAccessException class hierarchy classes.
c. Persistence logic is portable across the multiple DB softwares and Entity classes are portable across the multiple ORM frameworks.
d. Common Transaction Management support.
e. Common single row methods call given by JPA and common JPQL (Java Persistence Query Language)
   and etc.

Spring with Hibernate

- It says write Business logic in spring and write persistence logic in hibernate by taking the support of the Spring ORM supplied HibernateTemplate class.

HibernateTemplate: Given based Template method Design Pattern which say that it defines an algorithm where super class/ common class will take care common logic by leaving specific logics to sub classes/ developers to supply.

DAO class

```
HibernateTemplate object
          |---> SessionFactory object (dependent) (SessionFactory object
                                                     multiple services of hibernate)
                    |---> DS (DataSource object)
                    |---> Entity classes with Annotations (recommended) mapping
                                                                   files (old)
                    |---> hibernate properties
                             1. dialect (capable generating SQL queries)
                             2. show_sql (to see SQL queries on the console)
                             3. hbm2ddl.auto: validate (default), create, update
                                                          (best), create-drop
```

Update: creates the DB tables if they are not available, uses them if there are already available alters them by adding new columns if necessary.

We can develop Spring ORM Apps in following approaches:

a. Using xml driven configuration - (Configure both user-defined and pre-defined classes using xml).
b. Using xml + Annotation driven configuration (user-define classes using annotations and pre- defined class using xml).
c. Using 100% Code driven configuration (user-defined classes using annotations and pre-defined classes using @Bean methods in @Configuration class).
d. Using spring boot configuration (User-defined classes using annotation and pre-defined classes using @Bean methods if at all there are not coming through auto configuration).

Important spring Annotations for layered Apps:

- @Component - To make java class as spring bean with no specialties.
- @Service - To make java class as spring bean cum service class (support TxMgmt).
- @Controller - To make java class as spring bean cum controller.
- @Repository - To make java class as spring bean cum DAO class (With Exception translation support).

Annotations in Entity classes (priority order):

a. JPA Annotations
b. Hibernate Annotations
c. Java Config Annotations (JSE, JEE modules)
d. Third Party Annotations

Entity classes with Annotations - Basic Annotations:

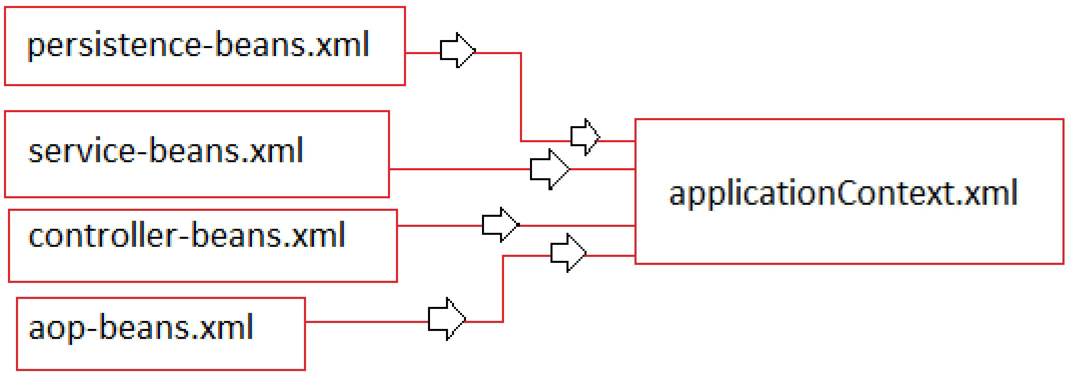
- @Entity (JPA) (mandatory)
- @Table (JPA) (optional) (JPA) (mandatory)
- @Colulmn (JPA) (optional)
- @Type (HB) (optional)

Note: If Entity class name is matching with DB table name and Entity properties are matching with DB table column names then placing @Table, @Column annotations optional. If want to use dynamic schema generation (DB tables generations) it is recommended to place them to control on type, length, unique and etc. details.

```
@Entity
@Table(name = “STUDENT”)
public class Student implements Serializable {
            @Column(name="SNO")
            @Type(type="int")
            private Integer sno;

            @Column(name="SNAME", length="20", nullable=false)
            @Type(type="string")
            private String sname;

            @Column(name="SADD", length="20")
            @Type(type="string")
            private String sadd;

            @Column(name="AVG")
            @Type(tvpe="float")
            private Float avg;
            //getters && getters
            ……………
            …………….
}
```

Note:

- ✓ To support java numeric data type ranges while creating Dynamic DB table length attribute values will not be reflected.
- ✓ In Db table creation but String columns the length will be reflected.
- ✓ The above operations will not take class, if you are working already available DB tables.



persistence-beans.xml

```
<beans ....>
          <!-- DataSource -->
          <bean id="hkDs" class="pkg.HikariDataSource"/>
          <bean id="sesfact" class="pkg.LocalSessionFactoryBean" >
                    <property name="dataSource" ref="hkDs"/>
                    <property name="annotatedClasses">
                              <array>
                                        <value>com.nt.entity.Student</value>
                              </array>
                    </property>
                    <property name="hibernateProperties">
                              <props>
                                        <prop key="dialect"
                                        org.hibernate.dialect.Oracle10gDialect</prop>
                                        <prop key-"show_sql">true</prop>
                                        <prop key-"hbm2ddl.auto">update</prop>
                              </props>
                    </property>
          </bean>
          <!-- Hibernate Template class -->
          <bean id="template" class="pkg.HibernateTemplate">
                    <constructor-arg ref=" sesfact"/>
          </bean>
```

Internal cache of IoC container

hkDs	hikariDataSource object reference
sesfact	SessionFactory object reference
template	HibernateTemplate object reference

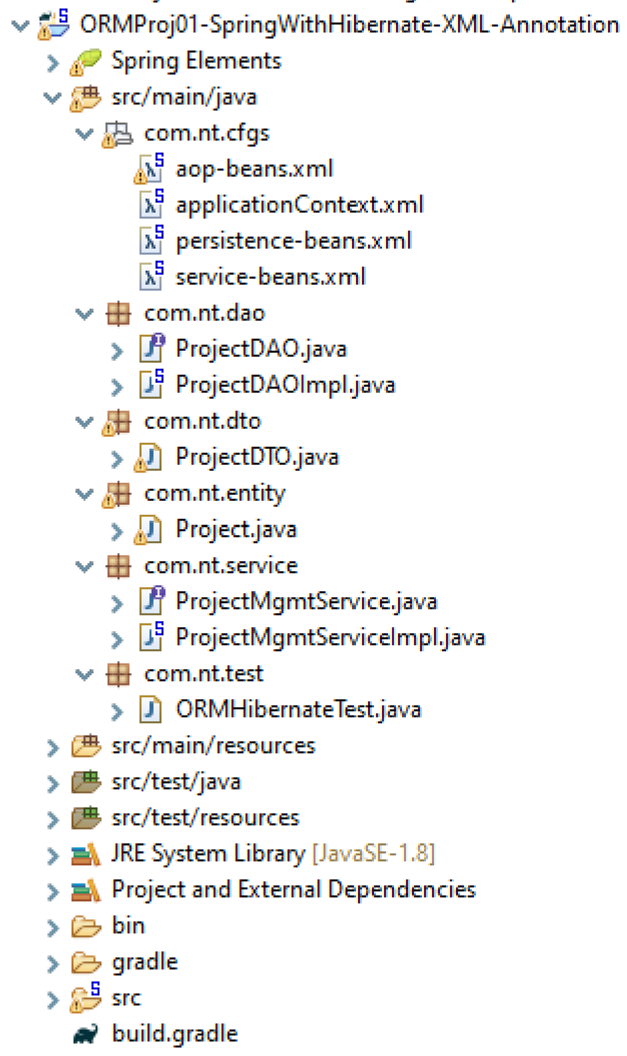
Note: Factory Bean that gives HB SessionFactory object as Resultant object based on the given injected values like DS, HB Properties and etc.). LocalSessionFactoryBean is a selfless Bean.

Using xml + Annotation driven configuration

Setup of our App:

Client App ---> Service class ---> DAO class ---> DB s/w
Inject HibernateTemplate

Directory Structure of ORMProj01-SpringWithHibernate-XML-Annotation:



```
ORMProj01-SpringWithHibernate-XML-Annotation
  Spring Elements
  src/main/java
    com.nt.cfgs
      aop-beans.xml
      applicationContext.xml
      persistence-beans.xml
      service-beans.xml
    com.nt.dao
      ProjectDAO.java
      ProjectDAOImpl.java
    com.nt.dto
      ProjectDTO.java
    com.nt.entity
      Project.java
    com.nt.service
      ProjectMgmtService.java
      ProjectMgmtServiceImpl.java
    com.nt.test
      ORMHibernateTest.java
  src/main/resources
  src/test/java
  src/test/resources
  JRE System Library [JavaSE-1.8]
  Project and External Dependencies
  bin
  gradle
  src
  build.gradle
```

- Develop the above directory structure and package, class, XML file and add the jar dependencies in build.gradle file then use the following code with in their respective file.

build.gradle

```
plugins {
   // Apply the java-library plugin to add support for Java Library
   id 'java-library'
}

repositories {
   // Use jcenter for resolving dependencies.
   // You can declare any Maven/Ivy/file repository here.
   jcenter()
}

dependencies {
   // https://mvnrepository.com/artifact/org.springframework/spring-context-support
        implementation group: 'org.springframework', name: 'spring-context-support', version: '5.2.8.RELEASE'
   // https://mvnrepository.com/artifact/org.springframework/spring-orm
        implementation group: 'org.springframework', name: 'spring-orm', version: '5.2.8.RELEASE'
   // https://mvnrepository.com/artifact/org.hibernate/hibernate-core
        implementation group: 'org.hibernate', name: 'hibernate-core', version: '5.4.20.Final'
        // https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc6
        implementation group: 'com.oracle.database.jdbc', name: 'ojdbc6', version: '11.2.0.4'
        // https://mvnrepository.com/artifact/com.zaxxer/HikariCP
        implementation group: 'com.zaxxer', name: 'HikariCP', version: '3.4.5'
}
```

Project.java

```
package com.nt.entity;

import java.io.Serializable;
```

```java
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import org.hibernate.annotations.Type;

@Entity
public class Project implements Serializable {

	@Id
	@Type(type = "int")
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Integer projId;

	@Column(length = 20, unique = true, nullable = false)
	@Type(type = "string")
	private String projName;

	@Type(type = "int")
	private Integer teamSize;

	@Column(length = 20, nullable = false)
	@Type(type = "string")
	private String company;

	@Column(length = 20)
	@Type(type = "string")
	private String location;

	@Type(type = "double")
	private Double cost;

	//setter and getters
	public Integer getProjId() {
		return projId;
	}
	public void setProjId(Integer projId) {
		this.projId = projId;
```

```
        }
        public String getProjName() {
                return projName;
        }
        public void setProjName(String projName) {
                this.projName = projName;
        }
        public Integer getTeamSize() {
                return teamSize;
        }
        public void setTeamSize(Integer teamSize) {
                this.teamSize = teamSize;
        }
        public String getCompany() {
                return company;
        }
        public void setCompany(String company) {
                this.company = company;
        }
        public String getLocation() {
                return location;
        }
        public void setLocation(String location) {
                this.location = location;
        }
        public Double getCost() {
                return cost;
        }
        public void setCost(Double cost) {
                this.cost = cost;
        }

}
```

ProjectDTO.java

```
package com.nt.dto;

import java.io.Serializable;

public class ProjectDTO implements Serializable {
```

```
package com.nt.dto;

import java.io.Serializable;

public class ProjectDTO implements Serializable {

	private Integer projId;
	private String projName;
	private Integer teamSize;
	private String company;
	private String location;
	private Double cost;

	// setter & getters
	public Integer getProjId() {
		return projId;
	}
	public void setProjId(Integer projId) {
		this.projId = projId;
	}
	public String getProjName() {
		return projName;
	}
	public void setProjName(String projName) {
		this.projName = projName;
	}
	public Integer getTeamSize() {
		return teamSize;
	}
	public void setTeamSize(Integer teamSize) {
		this.teamSize = teamSize;
	}
	public String getCompany() {
		return company;
	}
	public void setCompany(String company) {
		this.company = company;
	}
	public String getLocation() {
		return location;
	}
	public void setLocation(String location) {
		this.location = location;
	}
	public Double getCost() {
		return cost;
	}
	public void setCost(Double cost) {
		this.cost = cost;
	}

	//toString()
	@Override
	public String toString() {
		return "Project [projId=" + projId + ", projName=" + projName
+ ", teamSize=" + teamSize + ", company="
					+ company + ", location=" + location + ", cost=" +
cost + "]";
	}

}
```

ProjectDAO.java

```
package com.nt.dao;

import com.nt.entity.Project;

public interface ProjectDAO {
	public Integer insert(Project entity);
}
```

ProjectDAOImpl.java

```
package com.nt.dao;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;

import com.nt.entity.Project;
```

```
@Repository("projDAO")
public class ProjectDAOImpl implements ProjectDAO {

	@Autowired
	private HibernateTemplate ht;

	@Override
	public Integer insert(Project entity) {
		Integer idVal = null;
		// use HibernateTemplate
		idVal = (Integer) ht.save(entity);
		return idVal;
	}

}
```

ProjectMgmtService.java

```
package com.nt.service;

import com.nt.dto.ProjectDTO;

public interface ProjectMgmtService {
	public String registerProject(ProjectDTO dto);
}
```

ProjectMgmtServiceImpl.java

```
package com.nt.service;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.nt.dao.ProjectDAO;
import com.nt.dto.ProjectDTO;
import com.nt.entity.Project;
```

```java
@Service("projService")
@Transactional
public class ProjectMgmtServiceImpl implements ProjectMgmtService {

	@Autowired
	private ProjectDAO dao;

	@Override
	public String registerProject(ProjectDTO dto) {
		Project entity = null;
		Integer  idVal = null;
		//Convert DTO to BO/entity
		entity = new Project();
		BeanUtils.copyProperties(dto, entity);
		//use DAO
		idVal = dao.insert(entity);

		return "Project is Registered with the Project ID : "+idVal;
	}

}
```

service-beans.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
              http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

	<!-- Link user define annotation configuration with Spring bean
configuration file -->
	<context:component-scan base-package="com.nt.service"/>

</beans>
```

persistence-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

	<!-- DataSource configuration -->
	<bean name="hkDs" class="com.zaxxer.hikari.HikariDataSource">
		<property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"/>
		<property name="jdbcUrl"
value="jdbc:oracle:thin:@localhost:1521:xe"/>
		<property name="username" value="system"/>
		<property name="password" value="manager"/>
	</bean>

	<!-- LocalSessionFactoryBean configuration to get SessionFactory -->
	<bean id="sesfact"
class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
		<property name="dataSource" ref="hkDs"/>
		<property name="annotatedClasses">
			<array>
				<value>com.nt.entity.Project</value>
			</array>
		</property>
		<property name="hibernateProperties">
			<props>
				<prop
key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>
				<prop
key="hibernate.hbm2ddl.auto">update</prop>
				<prop key="hibernate.show_sql">true</prop>
				<prop key="hibernate.format_sql">true</prop>
			</props>
		</property>
	</bean>
```

```
            <!-- Configuration of HibernateTemplate -->
        <bean id="template"
class="org.springframework.orm.hibernate5.HibernateTemplate">
            <constructor-arg ref="sesfact"/>
        </bean>

        <!-- Link user defined class having annotation configuration with
spring bean configuration -->
        <context:component-scan base-package="com.nt.dao"/>

</beans>
```

aop-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

        <!-- Configure TransactionManager -->
        <bean id="hbTxMgmr"
class="org.springframework.orm.hibernate5.HibernateTransactionManager
">
            <property name="sessionFactory" ref="sesfact"/>
        </bean>

        <!-- Enable annotation driven TxMgmt -->
        <tx:annotation-driven transaction-manager="hbTxMgmr"/>

</beans>
```

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
```

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

        <import resource="service-beans.xml"/>
        <import resource="persistence-beans.xml"/>
        <import resource="aop-beans.xml"/>

</beans>
```

ORMHibernateTest.java

```
package com.nt.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.dao.DataAccessException;

import com.nt.dto.ProjectDTO;
import com.nt.service.ProjectMgmtService;

public class ORMHibernateTest {

        public static void main(String[] args) {
                ApplicationContext ctx = null;
                ProjectMgmtService service = null;
                ProjectDTO dto = null;
                //Create ApplicationContext IoC container
                ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Service class obejct
                service = ctx.getBean("projService", ProjectMgmtService.class);
                try {
                        //Create DTO
                        dto = new ProjectDTO();
                        dto.setProjName("BSE");
                        dto.setTeamSize(23);
                        dto.setCompany("NimuSoft.com");
                        dto.setLocation("Odisha");
                        dto.setCost(400000.0);
                //use service
                System.out.println(service.registerProject(dto));
            } catch (DataAccessException dae) {
                dae.printStackTrace();
            }

            //close container
            ((AbstractApplicationContext) ctx).close();
        }

}
```

- To perform all single row operations, place the following code with their respective files along with the previous codes.

ProjectDAO.java

```
    public Project getProjectById(int id);
    public boolean updateProjectById(int id, int teamSize, double cost);
    public boolean deleteProjectById(int id);
```

ProjectDAOImpl.java

```
    @Override
    public Project getProjectById(int id) {
        Project proj = null;
        //get Object
        proj = ht.get(Project.class, id);
        return proj;
    }

    @Override
    public boolean updateProjectById(int id, int teamSize, double cost) {
        Project proj = null;
        boolean flag = false;
        //get Object
        proj = ht.get(Project.class, id);
        if (proj!=null) {
            //update object
            proj.setTeamSize(teamSize);
				proj.setCost(cost);
				ht.update(proj);
				flag = true;
			}
			return flag;
		}

		@Override
		public boolean deleteProjectById(int id) {
			Project proj = null;
			boolean flag = false;
			//get Object
			proj = ht.get(Project.class, id);
			if (proj!=null) {
				//delete object
				ht.delete(proj);
				flag = true;
			}
			return flag;
		}
```

ProjectMgmtService.java

```
		public Object fetchProjectById(int id);
		public String modifyProjectById(int id, int teamSize, double cost);
		public String removeProjectById(int id);
```

ProjectMgmtServiceImpl.java

```
		@Override
		public Object fetchProjectById(int id) {
			Project proj = null;
			ProjectDTO dto = null;
			//use DAO
			proj = dao.getProjectById(id);
			//convert entity to dto
			if (proj!=null) {
				dto = new ProjectDTO();
				BeanUtils.copyProperties(proj, dto);
		}
		return dto!=null?dto:"Record not found";
	}

	@Override
	public String modifyProjectById(int id, int teamSize, double cost) {
		boolean flag = false;
		//use DAO
		flag = dao.updateProjectById(id, teamSize, cost);
		return flag==false?"Record not found for update":"Record 
updated";
	}

	@Override
	public String removeProjectById(int id) {
		boolean flag = false;
		//use DAO
		flag = dao.deleteProjectById(id);
		return flag==false?"Record not found for delete":"Record 
deleted";
	}
```

ORMHibernateTest.java

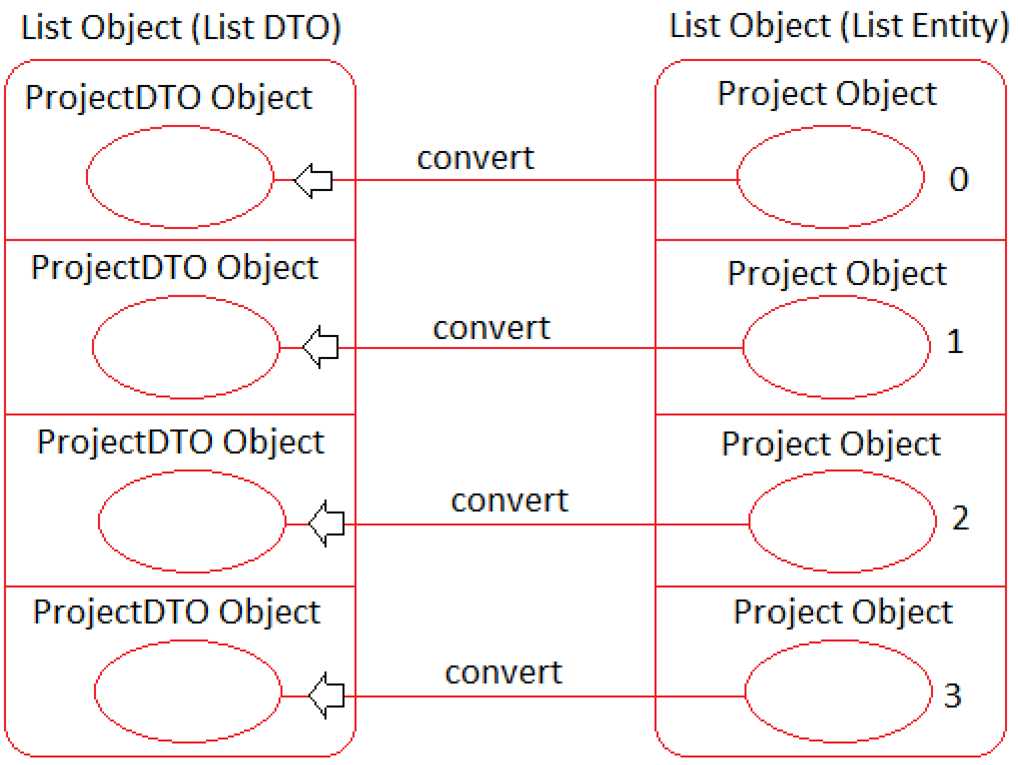
```
		System.out.println("------------------");
		System.out.println("Project details : 
"+service.fetchProjectById(1));
		System.out.println("------------------");
		System.out.println("Project update : 
"+service.modifyProjectById(1, 23, 2435465));
		System.out.println("------------------");
		System.out.println("Project delete : 
"+service.removeProjectById(3));
```

- Perform the operation one by one and mean while comments the other operations.
- Don’t remove the previous code just add this along with them.
- And comment also insert operation code while performing other.

Bulk Operations in Hibernate:

a. HQL (Hibernate Query Language)/ JPQL (Java Persistence Query language)
b. Native SQL
c. Criteria API

- To execute HQL/JPQL Select Queries we use find (), findXxx() methods on HibernateTemplate class object.
- Similarly for non-select queries use bulkUpdate(-) method.
- SQL DB s/w dependent Queries written by using Db table name, column names.
- HQL/ JPQL DB s/w independent queries written Using Entity class name and its Properties.
- HB dialect converts HQL/ JPQL queries into underlying DB s/w SQL queries.
- Up to HB 5.1 HQL/ JPQL supports both named (:<name>) and positional params(?) in the HQL/ JPQL queries, but from hibernate 5.2 we have support only for named Parameters.
- So most of the find () findXxx() that supports positional params are deprecated in HibernateTemplate class (They did this work in support to Spring Data JPA).



ProjectDAO.java

```
	//bulk operations
	public List<Project> getProjectByCostRange(double start, double 
end);
```

ProjectDAOImpl.java

```
	private static final String HQL_GET_PROJECT_BY_COST_RANGE = 
"FROM com.nt.entity.Project WHERE cost>=:min AND cost<=:max";

	@Override
	public List<Project> getProjectByCostRange(double start, double end) 
{
		List<Project> list = null;
		list = (List<Project>) 
ht.~~findByNamedParam~~(HQL_GET_PROJECT_BY_COST_RANGE, new String[] 
{"min", "max"}, new Object[] {start, end});
		return list;
	}
```

ProjectMgmtService.java

```
	public List<ProjectDTO> fetchProjectByCostRange(double start, 
double end);
```

ProjectMgmtServiceImpl.java

```
	@Override
	public List<ProjectDTO> fetchProjectByCostRange(double start, 
double end) {
		List<Project> listEntities = null;
		List<ProjectDTO> listDTO = new ArrayList();
		//use dao
		listEntities = dao.getProjectByCostRange(start, end);
		//covert listEntities to listDTO
		listEntities.forEach(entity -> {
			ProjectDTO dto = new ProjectDTO();
			BeanUtils.copyProperties(entity, dto);
			listDTO.add(dto);
		});
```

```
                return listDTO;
        }
```

ORMHibernateTest.java

```
                System.out.println("------------------");
                System.out.print("Projet details by cost range : ");
                listDTO = service.fetchProjectByCostRange(100000, 500000);
                listDTO.forEach(dto1 -> {
                        System.out.println(dto1);
                });
                System.out.println("-------------------");
                listDTO.forEach(System.out::println);
                System.out.println("------------------");
                System.out.println(listDTO);
                System.out.println("------------------");
                listDTO.stream().forEach(System.out::println);
```

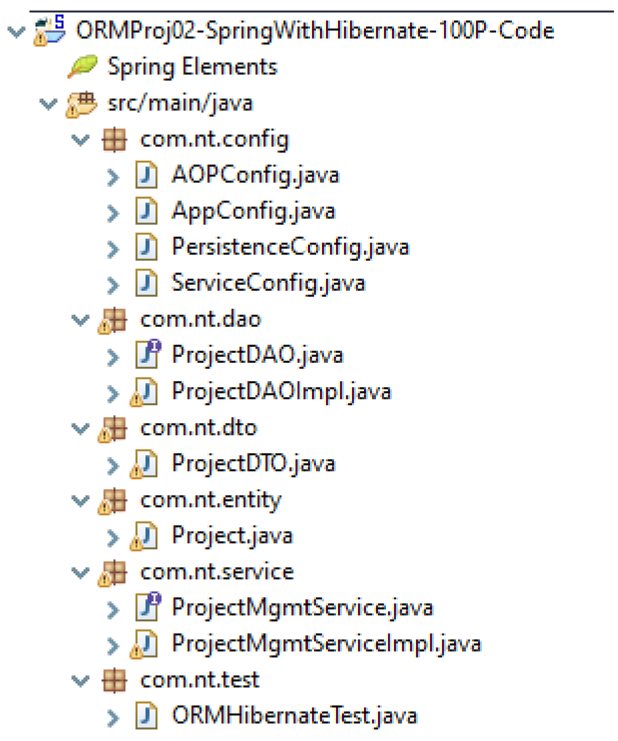
Using 100% Code driven configuration

Thumb rule to Spring app as 100%Code driven Application (no xml file):

Rule 1. Configure user-defined classes as spring bean using stereo type annotations (@Repository, @Service, @Component and etc.) and link them with @Configuration class using @ComponentScan annotation.
Rule 2. Configure pre-defined classes as spring beans using @Bean methods (1 method per 1 bean object) of @Configuration class (alternate spring bean configuration file (xml file)).
Rule 3. Create AnnotationConfigApplicationContext container as IoC container by giving @Configuration class as input class.

Directory Structure of ORMProj02-SpringWithHibernate-100P-Code:

- Copy paste ORMProj01-SpringWithHibernate-XML-Annotataion and change rootProject.name to ORMProj02-SpringWithHibernate-100P-code in settings.gradle file.
- Remove the XML package along with the file and create a new package com.nt.config with the following java files. (PersistenceConfig.java, Service.java, AOP.java, AppConfig.java)
- Add the following code in their respective files.



```
ORMProj02-SpringWithHibernate-100P-Code
    Spring Elements
    src/main/java
        com.nt.config
            AOPConfig.java
            AppConfig.java
            PersistenceConfig.java
            ServiceConfig.java
        com.nt.dao
            ProjectDAO.java
            ProjectDAOImpl.java
        com.nt.dto
            ProjectDTO.java
        com.nt.entity
            Project.java
        com.nt.service
            ProjectMgmtService.java
            ProjectMgmtServiceImpl.java
        com.nt.test
            ORMHibernateTest.java
```

PersistenceConfig.java

```java
package com.nt.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;

import com.nt.entity.Project;
import com.zaxxer.hikari.HikariDataSource;

@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

	@Bean(name = "hkDs")
	public DataSource createDataSource() {
		HikariDataSource hkDs = null;
		hkDs = new HikariDataSource();
		hkDs.setDriverClassName("oracle.jdbc.driver.OracleDriver");
		hkDs.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
		hkDs.setUsername("system");
		hkDs.setPassword("manager");
		hkDs.setMaximumPoolSize(100);
		hkDs.setMinimumIdle(10);
		return hkDs;
	}

	@Bean(name = "sesfact")
	public LocalSessionFactoryBean createLocalSessionFactBean() {
		LocalSessionFactoryBean bean = null;
		Properties props = null;
		bean = new LocalSessionFactoryBean();
		bean.setDataSource(createDataSource());
		bean.setAnnotatedClasses(Project.class);
		props = new Properties();
		props.setProperty("hibernate.dialect",
"org.hibernate.dialect.Oracle10gDialect");
		props.setProperty("hibernate.hbm2ddl.auto", "update");
		props.setProperty("hibernate.show_sql", "true");
		props.setProperty("hibernate.format_sql", "true");
		bean.setHibernateProperties(props);
		return bean;
	}

	@Bean(name = "ht")
	public HibernateTemplate createHibernateTemplate() {
		return new
HibernateTemplate(createLocalSessionFactBean().getObject());
	}

}
```

ServiceConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
@ComponentScan(basePackages = "com.nt.service")
public class ServiceConfig {

}
```

AOPConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.nt.service")
public class ServiceConfig {

}
```

AppConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@Import(value = {PersistenceConfig.class, ServiceConfig.class,
AOPConfig.class})
public class AppConfig {

}
```

- Mention the Transaction Manager bean id in ProjectMgmtServiceImpl.java because before we kept in XML file but here so that follow the below annotation the service implementation class.

ProjectMgmtServiceImpl.java

```
@Transactional(transactionManager = "hbTxMgmr")
public class ProjectMgmtServiceImpl implements ProjectMgmtService {
```

- Change the Application context container using the AnnotaionConfigApplicationContext container like below rest part is same.

ORMHibernateTest.java

```
public class ORMHibernateTest {

	public static void main(String[] args) {
		ApplicationContext ctx = null;
		ProjectMgmtService service = null;
		ProjectDTO dto = null;
		List<ProjectDTO> listDTO = null;
		//Create ApplicationContext IoC container
		ctx = new
AnnotationConfigApplicationContext(AppConfig.class);
```

Using spring boot configuration

Thumb rules to develop spring Boot App (No xml + auto configuration):
Auto configuration - Makes certain classes as spring beans based on the jar files the added to the Application.

Rule 1. Configure user-defined classes as spring beans using stereo type annotations.
Rule 2. Configure pre-defined classes as spring beans using @Bean methods only if they are not coming as spring beans through Auto Configuration.
Rule 3. Get IOC container from SpringApplication.run(-) from @SpringBootApplication class (main class/ starter class) to write further coding.

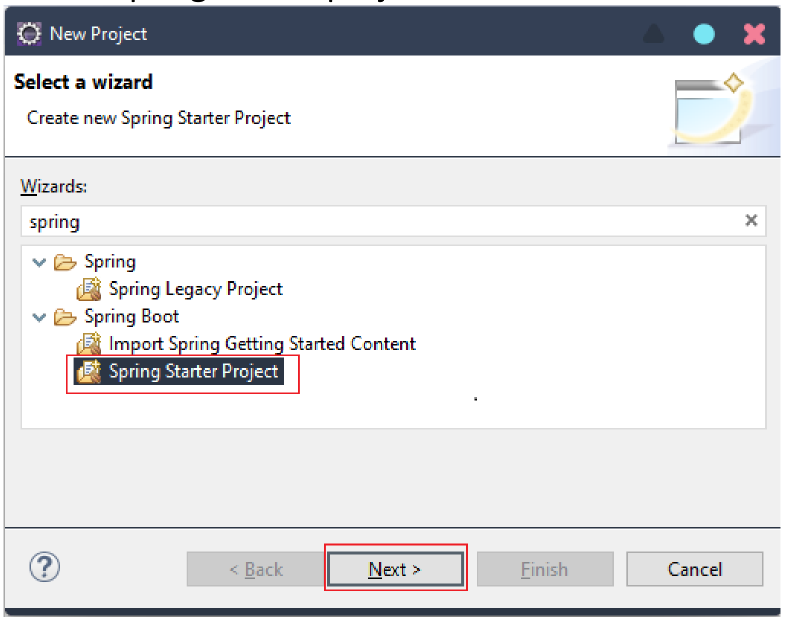
Note: We can give inputs to Auto Configuration beans with the support of application.properties/ YML file.

@SpringBootApplication is combination of 3:

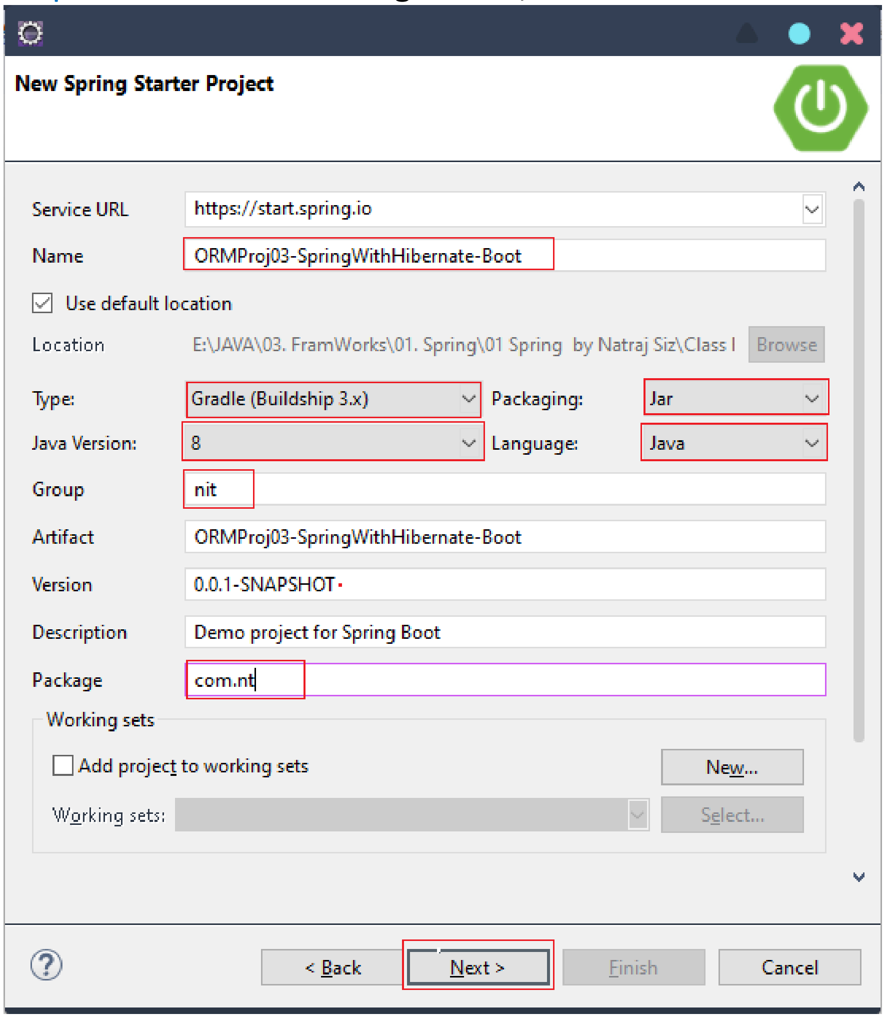
a. @Configuration (Make current class configuration class)
b. @ComponentScan (Automatically scans current package and sub packages spring beans, configuration classes).
c. @EnableAutoConfiguration (To give certain pre-defined classes as spring bean classes based on the jar files that are added)

Procedure to develop Spring ORM application using Spring boot in eclipse:

Step #1: Create a Sprig starter project, Click on File then New then other, search Spring starter project then choose that click on Next.



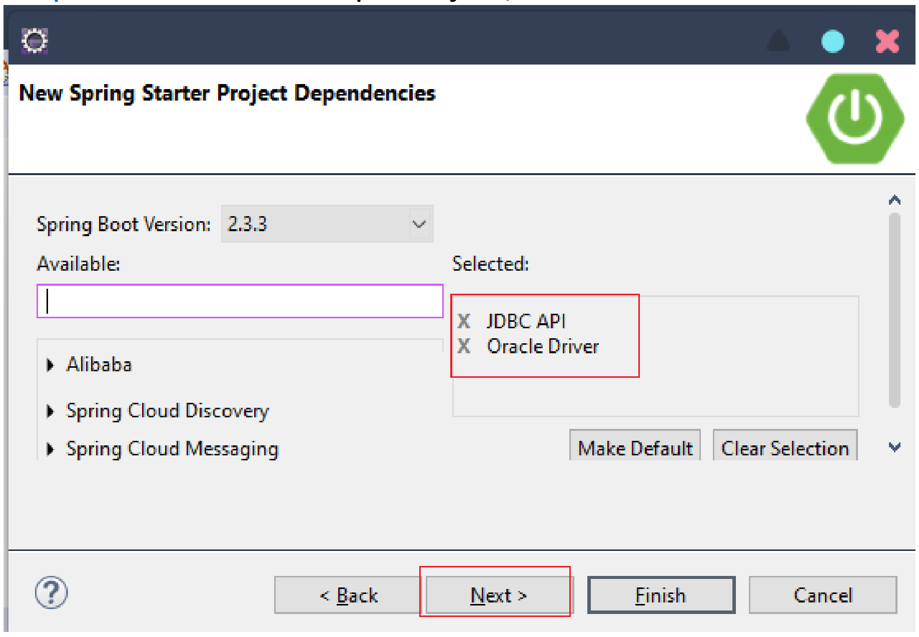
Step #2: Give the following details, then click on Next



Service URL: https://start.spring.io
Name: ORMProj03-SpringWithHibernate-Boot
☑ Use default location
Location: E:\JAVA\03. FramWorks\01. Spring\01 Spring by Natraj Siz\Class I
Type: Gradle (Buildship 3.x)
Packaging: Jar
Java Version: 8
Language: Java
Group: nit
Artifact: ORMProj03-SpringWithHibernate-Boot
Version: 0.0.1-SNAPSHOT
Description: Demo project for Spring Boot
Package: com.nt
☐ Add project to working sets
Working sets:

Details like
Name: Project name
Type: Gradle or maven		Packaging: Jar or war
Java Version: any version		Language: Java
Group: Company name
Artifact: Project name
Version: any version
Description: any description
Package: base package or parent package must be all package becomes is sub package.

Step #3: choose the required jars, then click on next.

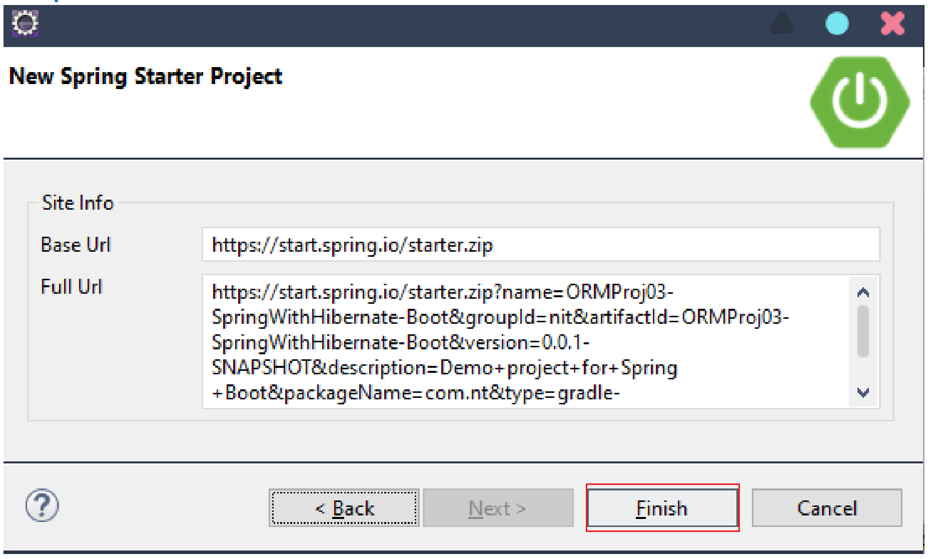

New Spring Starter Project Dependencies

Spring Boot Version: 2.3.3

Available:

Selected:
- JDBC API
- Oracle Driver

Step #4: Then click on Finish


New Spring Starter Project

Site Info

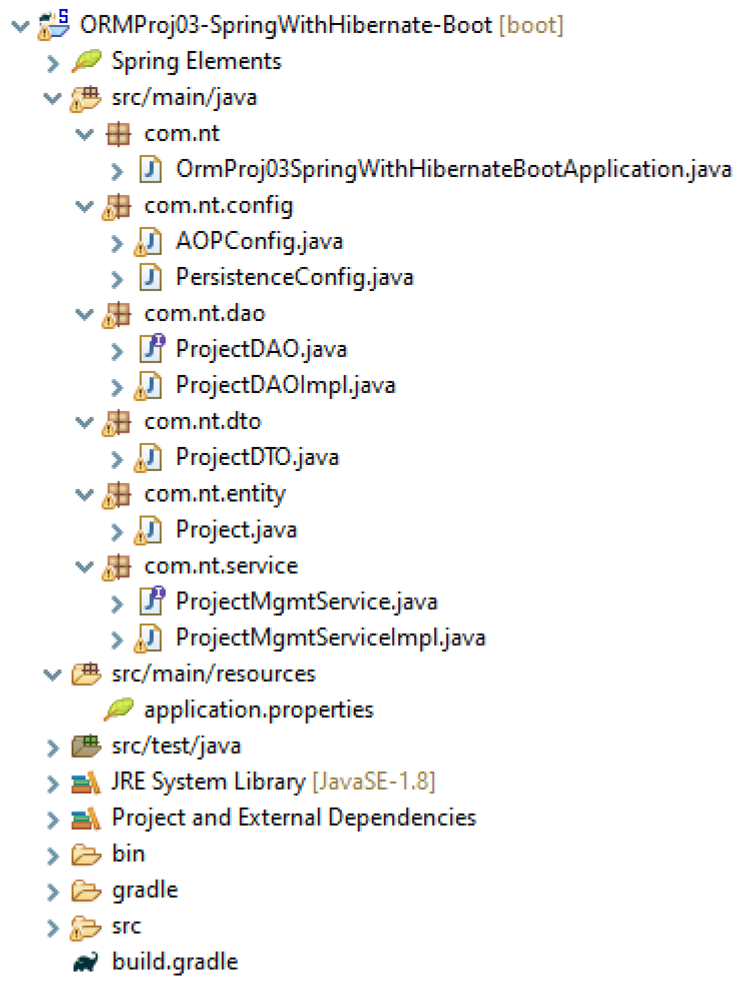
Base Url: https://start.spring.io/starter.zip

Full Url: https://start.spring.io/starter.zip?name=ORMProj03-SpringWithHibernate-Boot&groupId=nit&artifactId=ORMProj03-SpringWithHibernate-Boot&version=0.0.1-SNAPSHOT&description=Demo+project+for+Spring+Boot&packageName=com.nt&type=gradle-

Now your Spring Boot project is ready.

- If we added Spring-boot-JDBC-starter-<ver>.jar file to CLASSPATH/ BUILDPATH the following spring beans will come through auto Configuration:
    a. HikariDataSource object
    b. JdbcTemplate object
    c. DataSourceTransactionManager object

Directory Structure of ORMProj03-SpringWithHiberante-Boot:



```
ORMProj03-SpringWithHibernate-Boot [boot]
  Spring Elements
  src/main/java
    com.nt
      OrmProj03SpringWithHibernateBootApplication.java
    com.nt.config
      AOPConfig.java
      PersistenceConfig.java
    com.nt.dao
      ProjectDAO.java
      ProjectDAOImpl.java
    com.nt.dto
      ProjectDTO.java
    com.nt.entity
      Project.java
    com.nt.service
      ProjectMgmtService.java
      ProjectMgmtServiceImpl.java
  src/main/resources
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Project and External Dependencies
  bin
  gradle
  src
  build.gradle
```

- Develop the above directory structure using Spring Starter Project option.
- Copy the commonly used packages and class from ORMProj02-SpringWithHibenate-100P-Code project.
- Then change and add the following code in their respective files.
- Many jars dependencies will be came automatically in build.gradle because while developing Spring Starter Project we choose some jars and other required jar we will add in dependencies { } enclosure along with previous jars.

build.gradle

```
	// https://mvnrepository.com/artifact/org.springframework/spring-orm
	implementation group: 'org.springframework', name: 'spring-orm', version: '5.2.8.RELEASE'
	// https://mvnrepository.com/artifact/org.hibernate/hibernate-core
	implementation group: 'org.hibernate', name: 'hibernate-core', version: '5.4.20.Final'
```

application.properties

```
#DataSource configuration
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

AOPConfig.java

```
package com.nt.config;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
public class AOPConfig {

	@Autowired
	public SessionFactory factory;

	@Bean(name="hbTxMgmr")
	public HibernateTransactionManager createHBTxMgmr() {
```

```
                return new HibernateTransactionManager(factory);
        }

}
```

PersistenceConfig.java

```
package com.nt.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;

import com.nt.entity.Project;

@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

        @Autowired
        private DataSource ds;

        @Bean(name = "sesfact")
        public LocalSessionFactoryBean createLocalSessionFactBean() {
                LocalSessionFactoryBean bean = null;
                Properties props = null;
                bean = new LocalSessionFactoryBean();
                bean.setDataSource(ds);
                bean.setAnnotatedClasses(Project.class);
                props = new Properties();
                props.setProperty("hibernate.dialect",
"org.hibernate.dialect.Oracle10gDialect");
                props.setProperty("hibernate.hbm2ddl.auto", "update");
                props.setProperty("hibernate.show_sql", "true");
```

```
		props.setProperty("hibernate.format_sql", "true");
		bean.setHibernateProperties(props);
		return bean;
	}

	@Bean(name = "ht")
	public HibernateTemplate createHibernateTemplate() {
		return new
HibernateTemplate(createLocalSessionFactBean().getObject());
	}

}
```

OrmProj03SpringWithHibernateBootApplication.java

```
package com.nt;

import java.util.List;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.dao.DataAccessException;

import com.nt.dto.ProjectDTO;
import com.nt.service.ProjectMgmtService;

@SpringBootApplication
public class OrmProj03SpringWithHibernateBootApplication {

	public static void main(String[] args) {
		ApplicationContext ctx = null;
		ProjectMgmtService service= null;
		ProjectDTO dto = null;
		List<ProjectDTO> listDTO = null;
		//Get AC IoC container
		ctx =
SpringApplication.run(OrmProj03SpringWithHibernateBootApplication.class
, args);
```

```
                    // get Service class obejct
                    service = ctx.getBean("projService", ProjectMgmtService.class);
                    try {
                              // Create DTO
                              dto = new ProjectDTO();
                              dto.setProjName("BwSE23");
                              dto.setTeamSize(23);
                              dto.setCompany("NimuSo2ft2.com");
                              dto.setLocation("Odisha");
                              dto.setCost(400000.0);
                              // use service
                              System.out.println(service.registerProject(dto));
                    } catch (DataAccessException dae) {
                              dae.printStackTrace();
                    }
                    System.out.println("------------------");
                    System.out.print("Projet details by cost range : ");
                    listDTO = service.fetchProjectByCostRange(100000, 500000);
                    listDTO.forEach(dto1 -> {
                              System.out.println(dto1);
                    });
                    System.out.println("-------------------");
                    listDTO.forEach(System.out::println);
                    System.out.println("------------------");
                    System.out.println(listDTO);
                    System.out.println("------------------");
                    listDTO.stream().forEach(System.out::println);
                    ((AbstractApplicationContext) ctx).close();
          }

}
```

- You can all the Methods.
- You can Run as normal application and as Spring Boot App

-- The END --