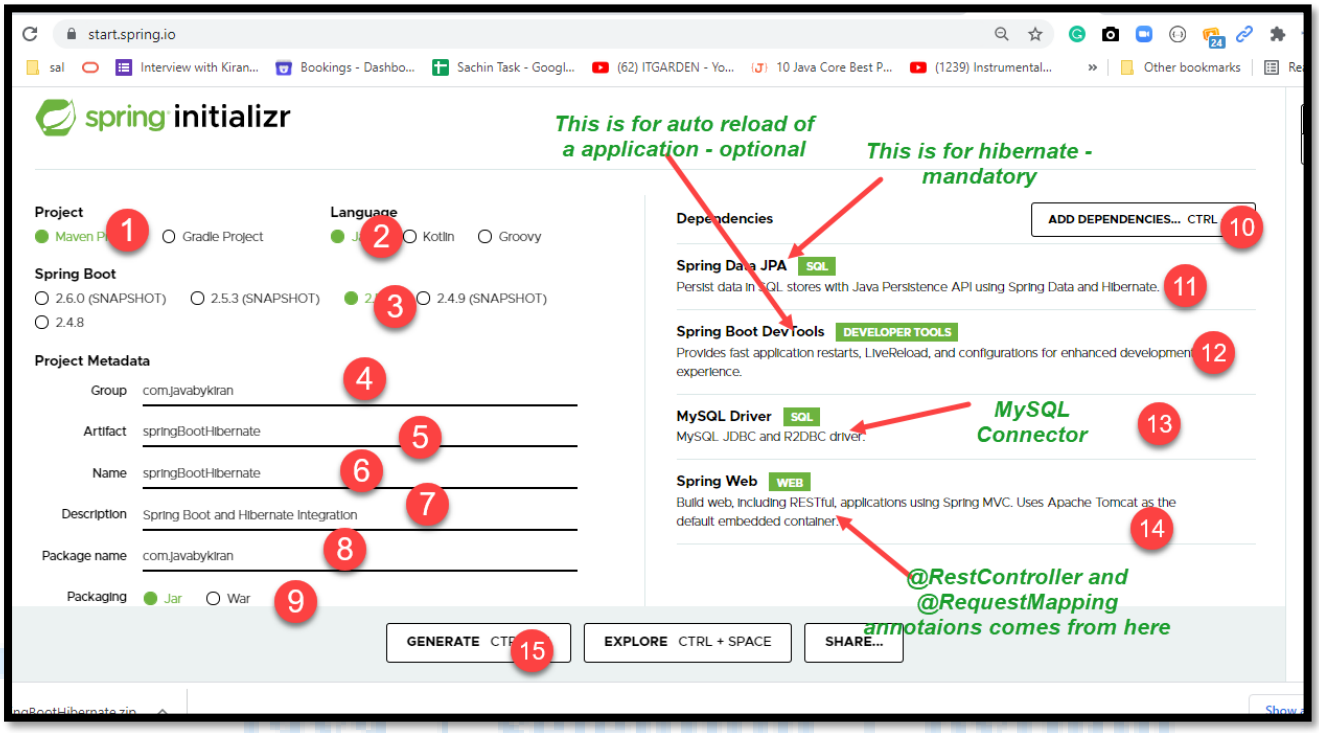
Hibernate and Spring boot Integration.

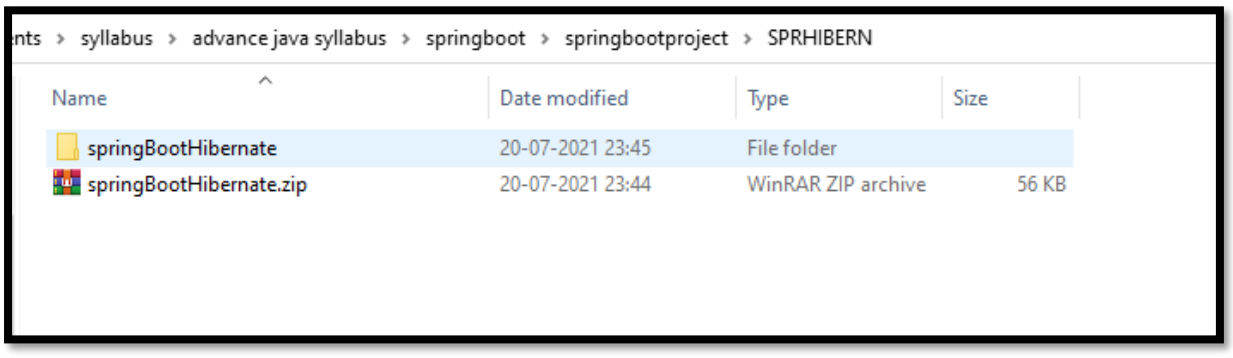
Prerequisite:

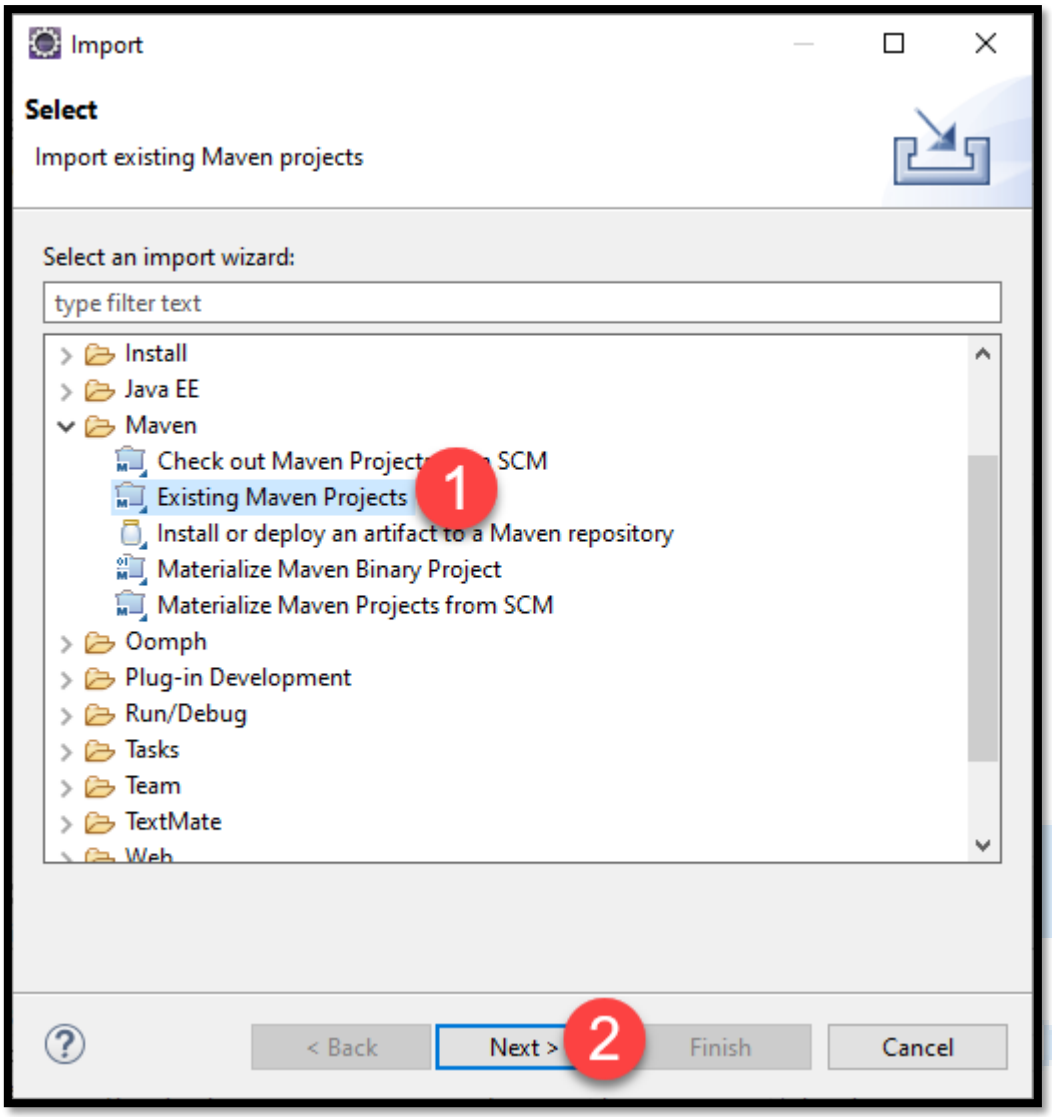
1. Should have MySQL installed and table should be ready.

Create spring boot project with dependencies JPA, web, dev tools and MySQL



Once downloaded extract and import into eclipse.



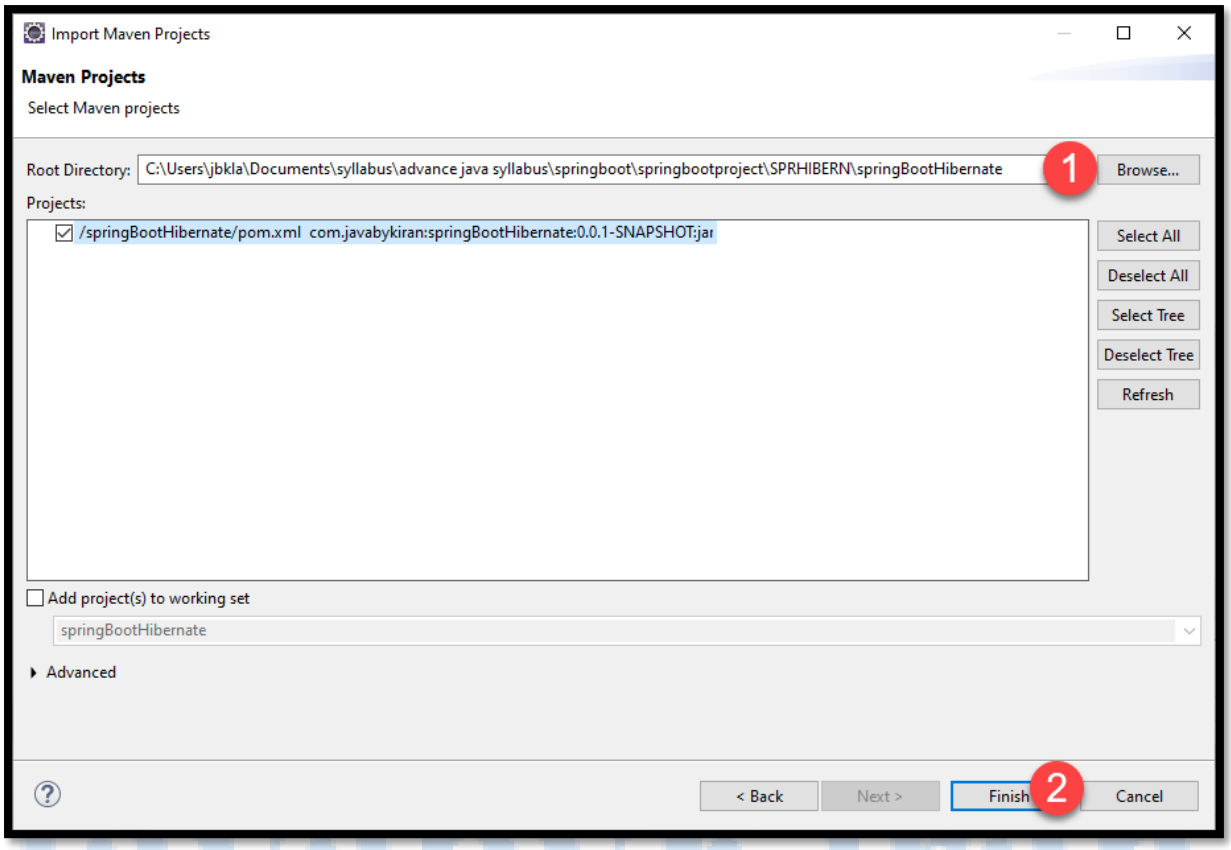


pom.xml will be as below add

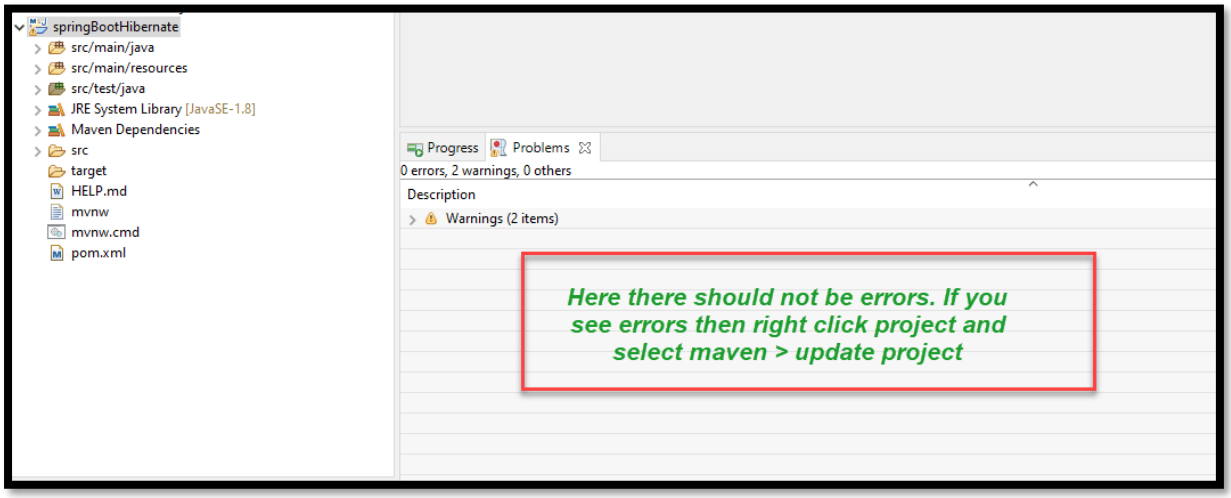
<version>5.1.6</version> in MySQL connector

```
<?xml version="1.0" encoding="UTF-8"?>
<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
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.5.2</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.javabykiran</groupId>
```

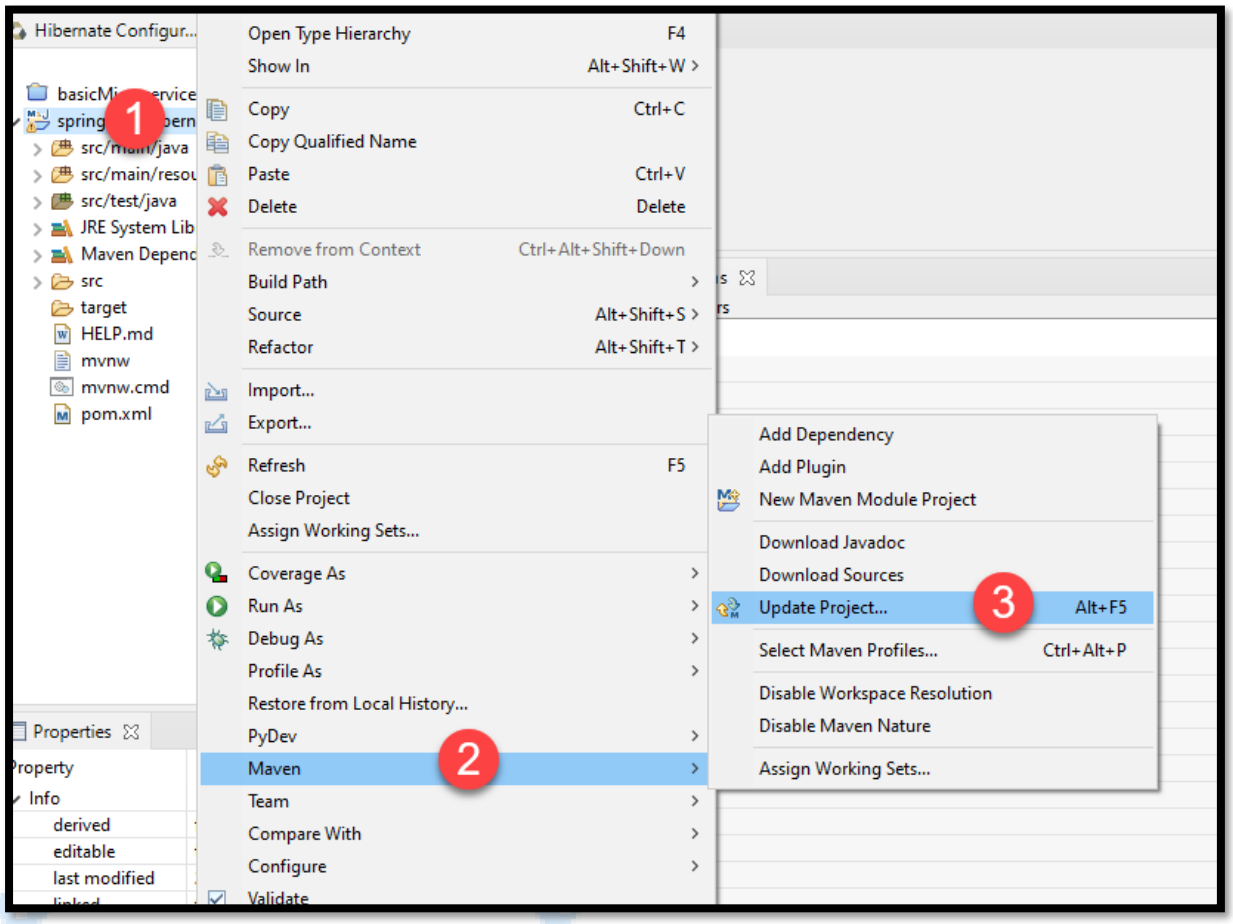
```
	<artifactId>springBootHibernate</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>springBootHibernate</name>
	<description>Spring Boot and Hibernate Integration</description>
	<properties>
		<java.version>1.8</java.version>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.6</version>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

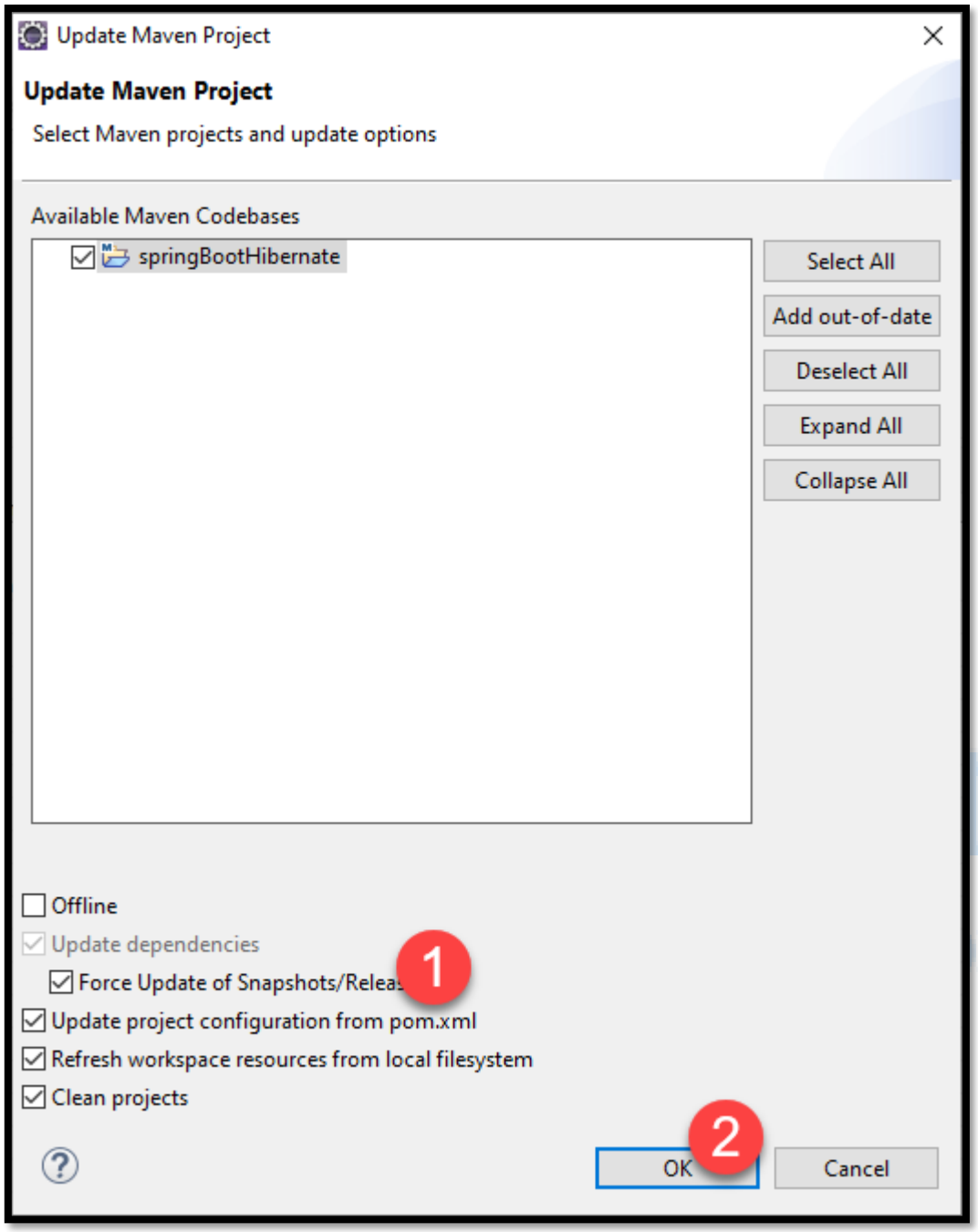


Once importing is done we can check if everything is correct.



While updating project with maven u should select below options. This is for any maven project.





This will bring all missing dependencies.

As we already have database ready, we need to create entity class as below. This is same as we created in hibernate.

```
package com.javabykiran;

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 = "student")
public class Student {
	int sid;
	String sname;
	String sphone;

	@Id
	@GeneratedValue(strategy = IDENTITY)
	@Column(name = "sid", unique = true, nullable = false)
	public int getSid() {
		return sid;
	}

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

	@Column(name = "sname", nullable = false, length = 45)
	public String getSname() {
		return sname;
	}

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

	@Column(name = "sphone", nullable = false, length = 45)
	public String getSphone() {
		return sphone;
	}

	public void setSphone(String sphone) {
		this.sphone = sphone;
	}

	@Override
	public String toString() {
		return "Student [sid=" + sid + ", sname=" + sname + ", sphone=" + sphone + "]";
	}

}
```

Configure hibernate in spring boot as below.

```
package com.javabykiran;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;

@SpringBootApplication
public class SpringBootHibernateApplication {
	@Autowired
	DataSource dataSource;

	public static void main(String[] args) {
		SpringApplication.run(SpringBootHibernateApplication.class, args);
	}

	@Bean
	public LocalSessionFactoryBean sessionFactory() {

LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
		sessionFactory.setDataSource(dataSource);
		System.out.println(dataSource);
		sessionFactory.setAnnotatedClasses(Student.class);
		return sessionFactory;

	}
}
```

Open properties file and add below configurations. These are fixed always only values you need to change.

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://localhost:3306/studentmanagement
spring.datasource.username = root
spring.datasource.password = root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
spring.jackson.serialization.fail-on-empty-beans=false
```

These are the fixed changes we need to do for hibernate configuration in spring projects.

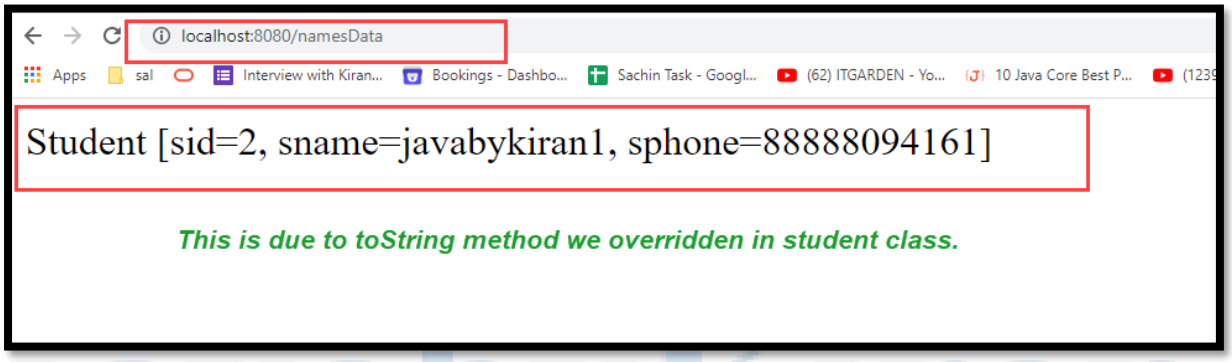
- Configuration of session factory – only once
- Entity class – if we have 10 tables then these classes count also be 10
- Properties file where we put all database configurations. While doing hibernate example same this we used to write in hibernate.cfg.xml

Now we need to create a controller class and add hibernate client code in it so that data will be fetched.

StudentController.java

```
package com.javabykiran;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class StudentController {
	@Autowired
	private SessionFactory sessionFactory;

	@RequestMapping("namesData")
	String giveYourNames() {// db call
		System.out.println("sessionFactory >>>" + sessionFactory);
		Session session = sessionFactory.getCurrentSession();
		Student student = (Student) session.load(Student.class, 2);
		return student.toString();
	}
}
```

Run a project and see on browser we will get student object from database.



Homework:

1. Separate Database code from controller to other class and annotate that class with @Repository and autowire that class in controller to call method.
2. Separate session factory code from the configuration file that is starter class to some other class and annotate that class with @Configuration.
3. Above 2 points will separate code between 2 layers.