Spring Boot @Repository Annotation with Example

Last Updated : 23 Jul, 2025

Spring is one of the most popular frameworks for building enterprise-level Java applications. It is an open-source, lightweight framework that simplifies the development of robust, scalable, and maintainable applications. Spring provides various features such as Dependency Injection (DI), Aspect-Oriented Programming (AOP), and support for Plain Old Java Objects (POJOs), making it a preferred choice for Java developers.

In this article, we will learn about the **@Repository Annotation in Spring Boot** with an example.

@Repository Annotation in Spring Boot

@Repository Annotation is a specialization of the **@Component** annotation, which is used to indicate that the class provides the mechanism for storage, retrieval, update, delete, and search operation on objects. Though it is a specialization of @Component annotation, Spring Repository classes are autodetected by the spring framework through classpath scanning. This annotation is a general-purpose stereotype annotation that is very close to the DAO pattern where DAO classes are responsible for providing CRUD operations on database tables.

Key Points about @Repository Annotation:

- The @Repository annotation is used to indicate that the class is a Data Access Object(DAO) or repository
- The main purpose of the @Repository annotation is to interact with a database. It encapsulates the logic required to access and manipulate data.

- The @Repository annotation provides exception translation. Spring automatically translates data access exceptions (e.g., SQL exceptions) into Spring's DataAccessException hierarchy. This makes it easier to handle database-related exceptions consistently.
- The @Repository annotation can only be applied to classes, not methods or fields.

Steps to Use the @Repository Annotation

Let's consider a simple example to understand how to use the @Repository annotation in a Spring Boot application.

Step 1: Create a Simple Spring Boot Project

Refer to this article Create and Setup Spring Boot Project in Eclipse IDE and create a simple spring boot project.

Step 2: Add the Spring-Context Dependency

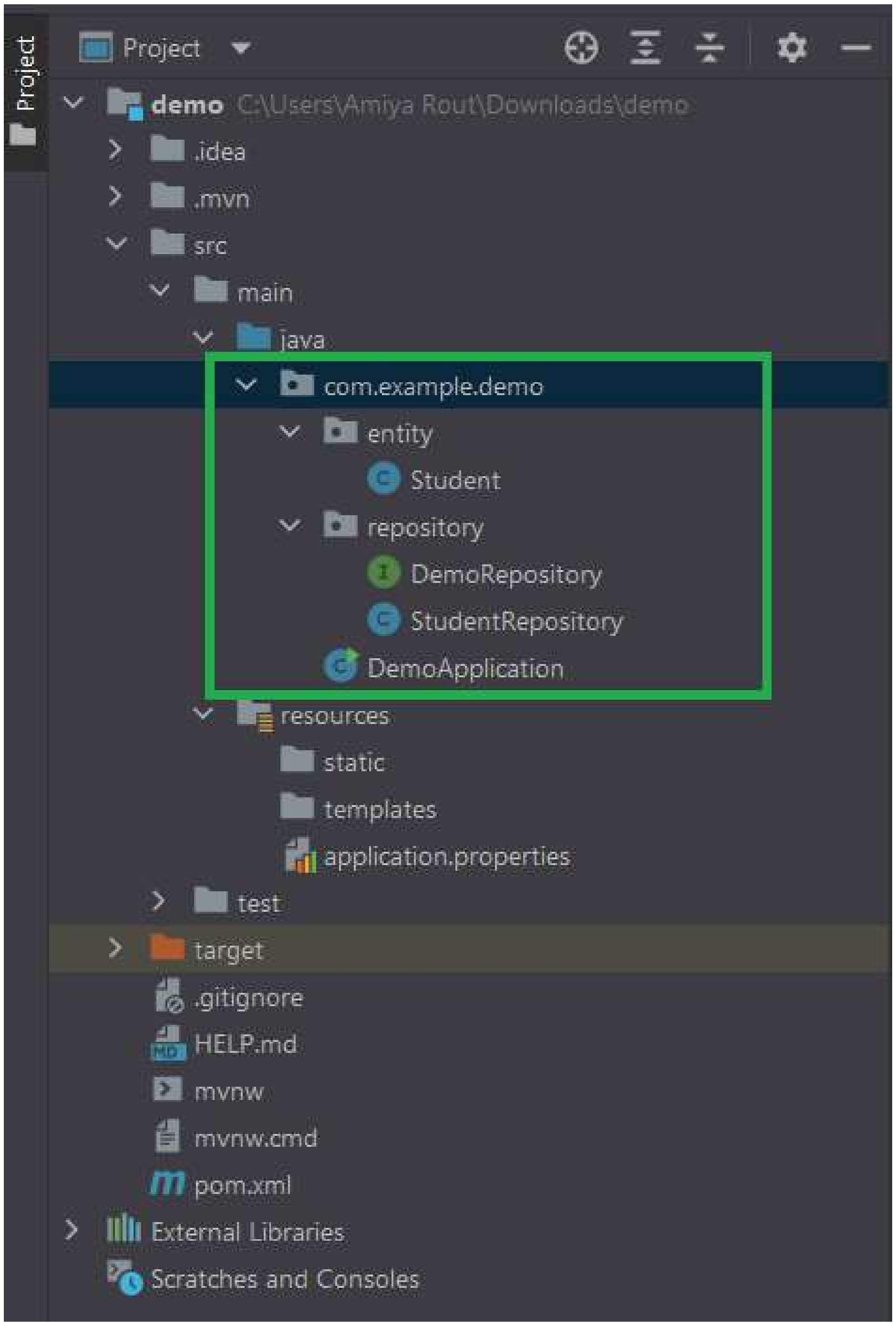
In your pom.xml file, add the following spring-context dependency.

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.13</version>
</dependency>
```

Step 3: Create Packages and Classes

In your project create two packages and name the package as "entity" and "repository". In the entity, package creates a class name it as Student. In the repository, the package creates a Generic Interface named as

DemoRepository and a class name it as StudentRepository. This is going to be our final project structure.



Step 4: Create the Entity Class

Create an entity class for which we will implement a spring repository. Here our entity class is Student. Below is the code for the **Student.java** file. This is a simple **POJO** (Plain Old Java Object) class in Java.

Student.java:

```java
package com.example.demo.entity;

public class Student {

    private Long id;
    private String name;
    private int age;

    public Student(Long id, String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }

    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

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

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

Step 5: Create the Repository Interface

Before implementing the Repository class we have created a generic DemoRepository interface to define the contract for our repository class. Below is the code for the **DemoRepository.java** file.

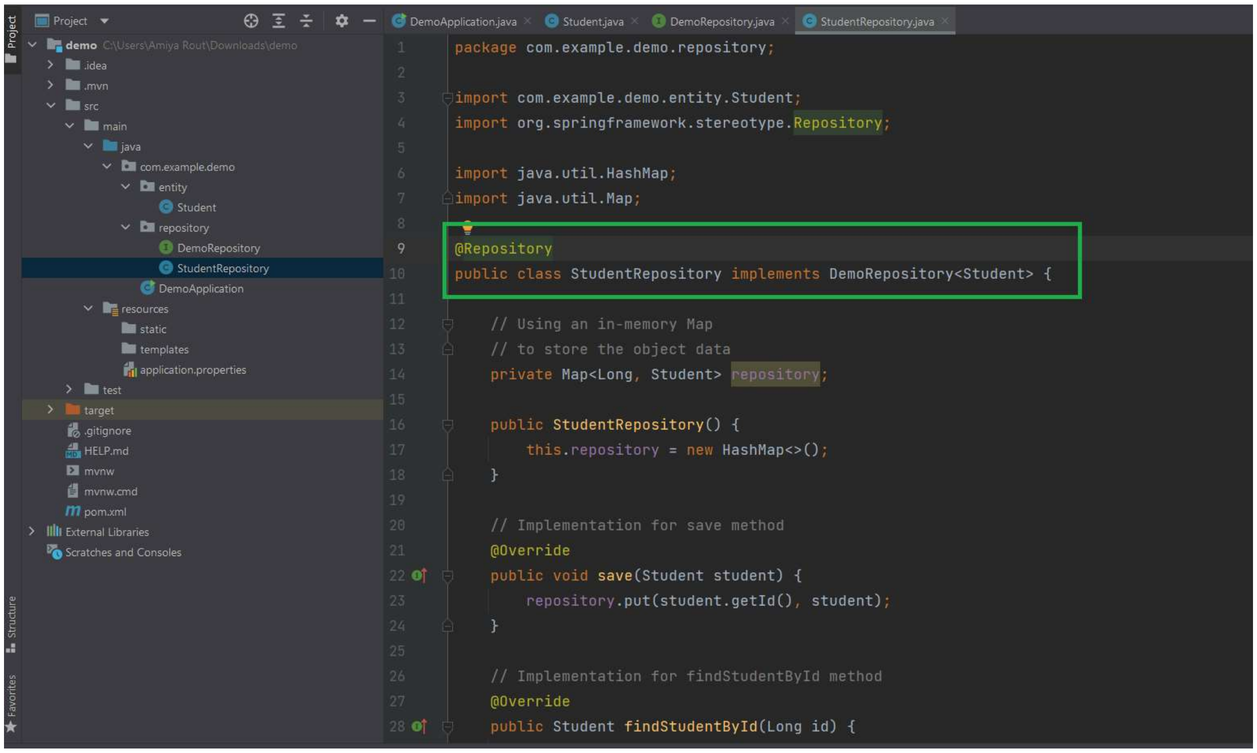
```java
// Java Program to illustrate DemoRepository File

package com.example.demo.repository;

public interface DemoRepository<T> {

    // Save method
    public void save(T t);

    // Find a student by its id
    public T findStudentById(Long id);

}
```

Step 6: Implement the Repository Class

Now let’s look at the implementation of our StudentRepository class.

```java
// Java Program to illustrate StudentRepository File

package com.example.demo.repository;

import com.example.demo.entity.Student;
import org.springframework.stereotype.Repository;

import java.util.HashMap;
import java.util.Map;

@Repository
public class StudentRepository implements DemoRepository<Student> {

    // Using an in-memory Map
    // to store the object data
    private Map<Long, Student> repository;

    public StudentRepository() {
        this.repository = new HashMap<>();
    }

    // Implementation for save method
    @Override
    public void save(Student student) {
        repository.put(student.getId(), student);
    }
    // Implementation for findStudentById method
    @Override
    public Student findStudentById(Long id) {
        return repository.get(id);
    }
}
```

In this **StudentRepository.java** file, you can notice that we have added the @Repository annotation to indicate that the class provides the mechanism for storage, retrieval, update, delete and search operation on objects.



```
package com.example.demo.repository;

import com.example.demo.entity.Student;
import org.springframework.stereotype.Repository;

import java.util.HashMap;
import java.util.Map;

@Repository
public class StudentRepository implements DemoRepository<Student> {

    // Using an in-memory Map
    // to store the object data
    private Map<Long, Student> repository;

    public StudentRepository() {
        this.repository = new HashMap<>();
    }

    // Implementation for save method
    @Override
    public void save(Student student) {
        repository.put(student.getId(), student);
    }

    // Implementation for findStudentById method
    @Override
    public Student findStudentById(Long id) {
```

> ***Note**: Here we have used an in-memory Map to store the object data, you can use any other mechanisms too. In the real world, we use Databases to store object data.*

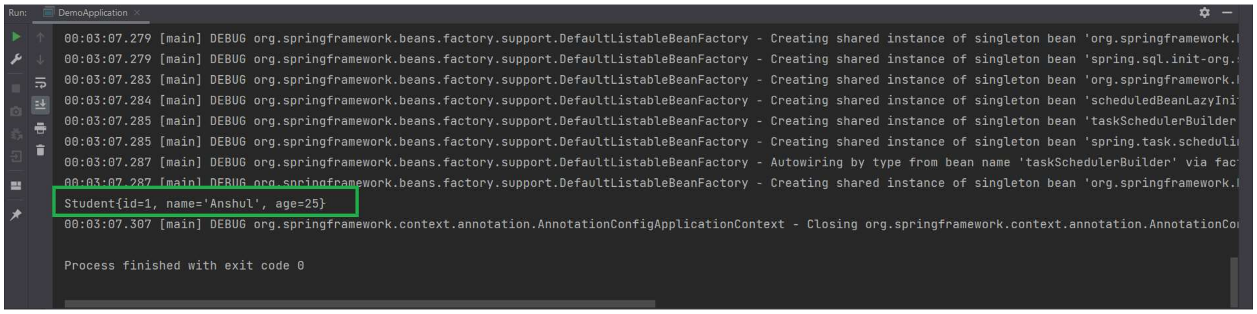
Step 7: Spring Repository Test

Now our Spring Repository is ready, let’s test it out. Go to the **DemoApplication.java** file and refer to the below code.

```
package com.example.demo;
```

```java
import com.example.demo.entity.Student;
import com.example.demo.repository.StudentRepository;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) {

        AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext();
        context.scan("com.example.demo");
        context.refresh();

        StudentRepository repository =
context.getBean(StudentRepository.class);

        // testing the store method
        repository.save(new Student(1L, "Anshul", 25));
        repository.save(new Student(2L, "Mayank", 23));

        // testing the retrieve method
        Student student = repository.findStudentById(1L);
        System.out.println(student);

        // close the spring context
        context.close();
    }

}
```

Output: Lastly, run your application and you should get the following output as shown below as follows:



```
00:03:07.279 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.
00:03:07.279 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.sql.init-org.
00:03:07.283 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.
00:03:07.284 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'scheduledBeanLazyIni
00:03:07.285 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'taskSchedulerBuilder
00:03:07.285 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'spring.task.scheduli
00:03:07.287 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Autowiring by type from bean name 'taskSchedulerBuilder' via fac
00:03:07.287 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'org.springframework.
Student{id=1, name='Anshul', age=25}
00:03:07.307 [main] DEBUG org.springframework.context.annotation.AnnotationConfigApplicationContext - Closing org.springframework.context.annotation.AnnotationCo

Process finished with exit code 0
```

Comment More info Advertise with us