Spring :

Spring in 1 word : Dependency Injection

Normal Java - Create objects with “new”
Spring - It will automatically create objects.

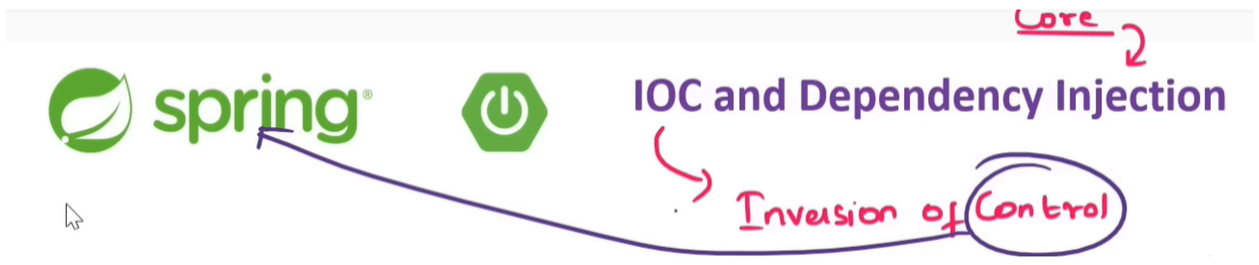
Spring boot - No configurations needed, just development

- Embedded server (Tomcat integrated)
- APIs can be built in minutes (REST API)
- Spring data JPA / Hibernate (Magic)

IOC and Dependency Injection :

Inversion of control - **object creation**

- Creation
- Managing
- Destroying

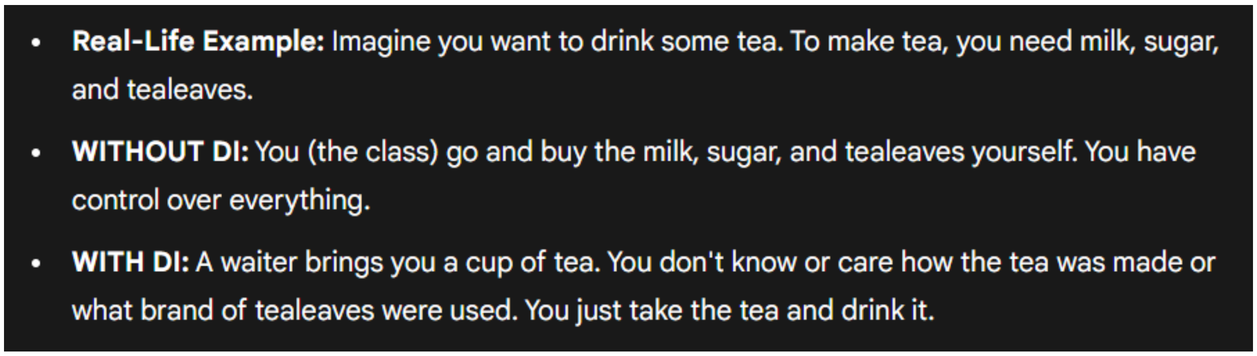


IOC vs DI :

Programming language is a concept
Java, Python, C++ implements it

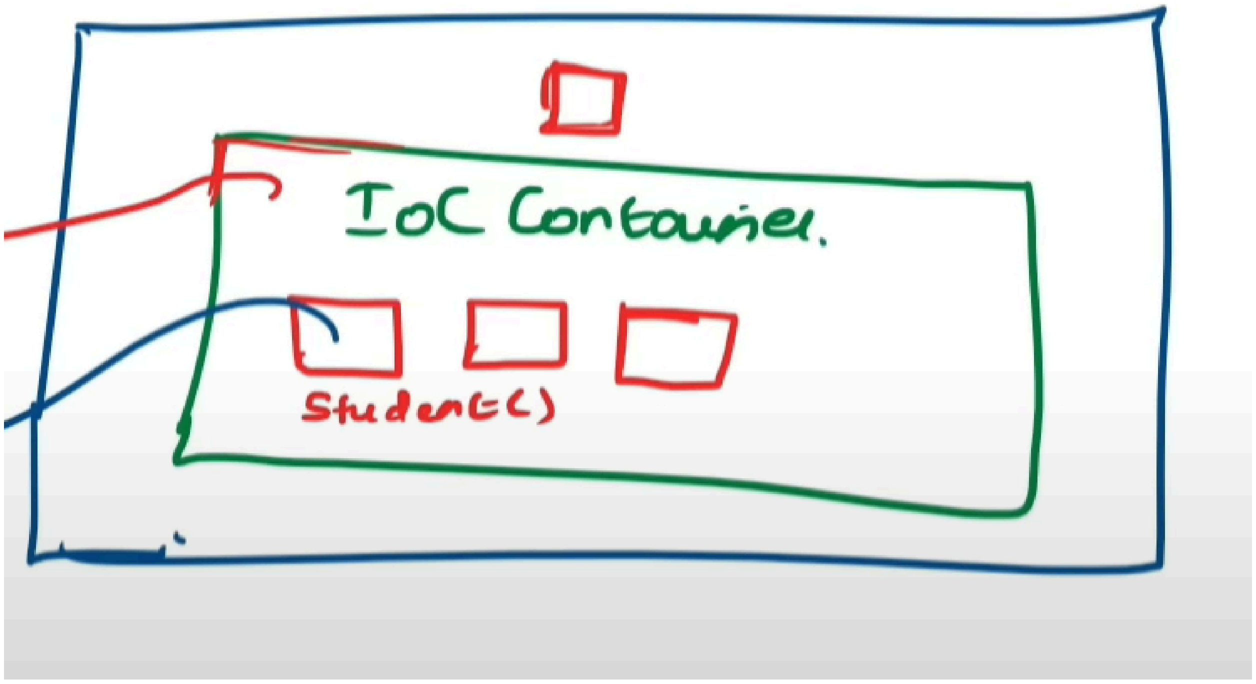
Like that

IOC is a concept
DI (Dependency Injection) implements it 🙂

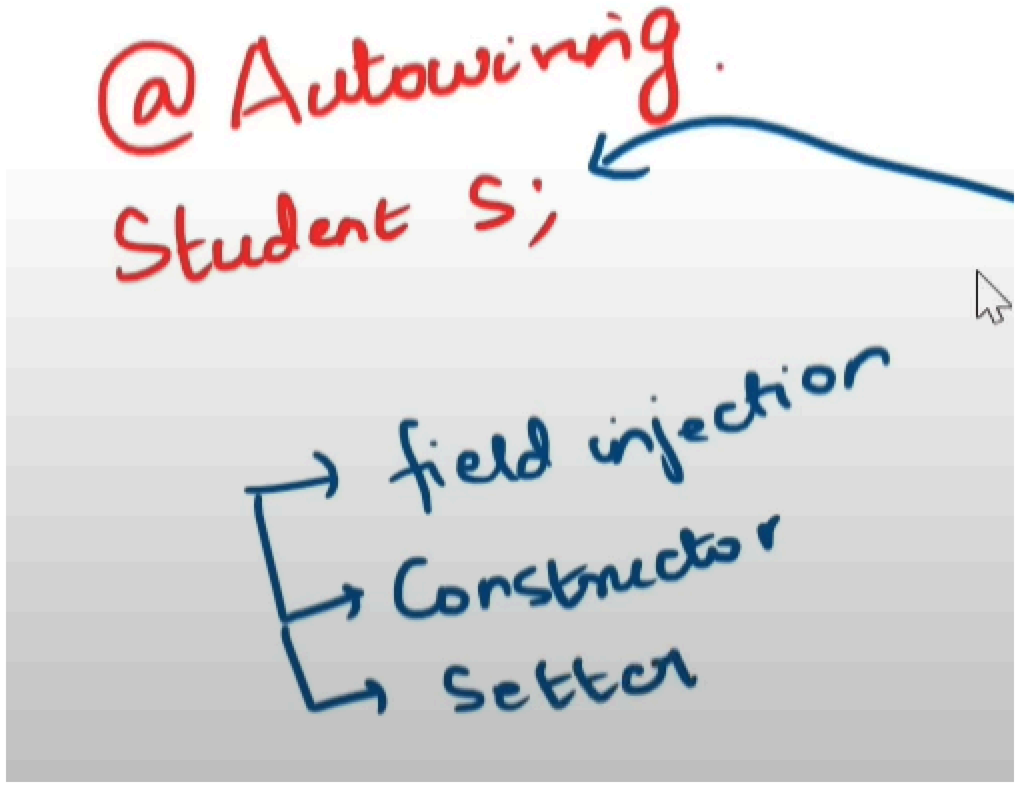


- **Real-Life Example:** Imagine you want to drink some tea. To make tea, you need milk, sugar, and tealeaves.
- **WITHOUT DI:** You (the class) go and buy the milk, sugar, and tealeaves yourself. You have control over everything.
- **WITH DI:** A waiter brings you a cup of tea. You don't know or care how the tea was made or what brand of tealeaves were used. You just take the tea and drink it.

In heap memory :

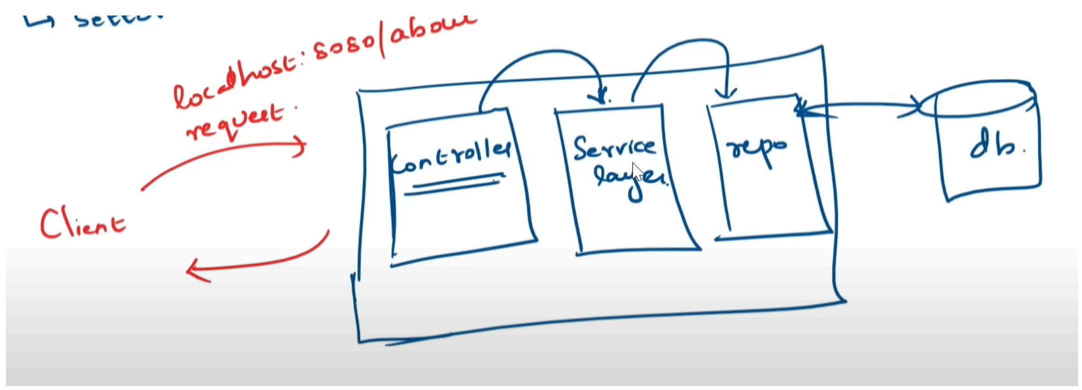


3 types of injection :



- Field injection
- Constructor
- Setter

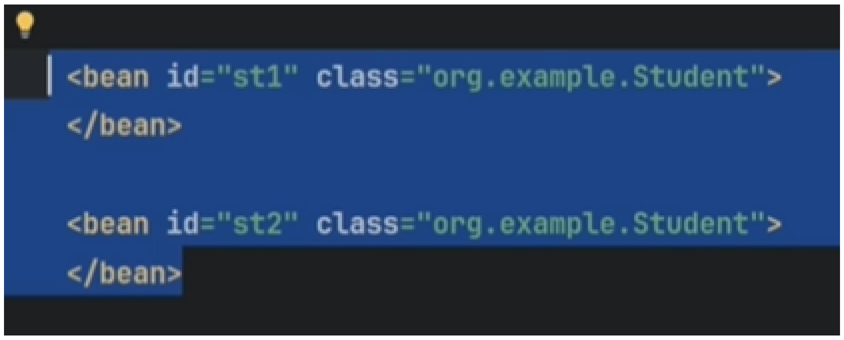
How spring works :



Bean = Objects in Spring

Create beans in config file for the objects needed (XML)

And create an object for application context with config file given and it will automatically create objects



```xml
<bean id="st1" class="org.example.Student">
</bean>

<bean id="st2" class="org.example.Student">
</bean>
```

Setter Injection :

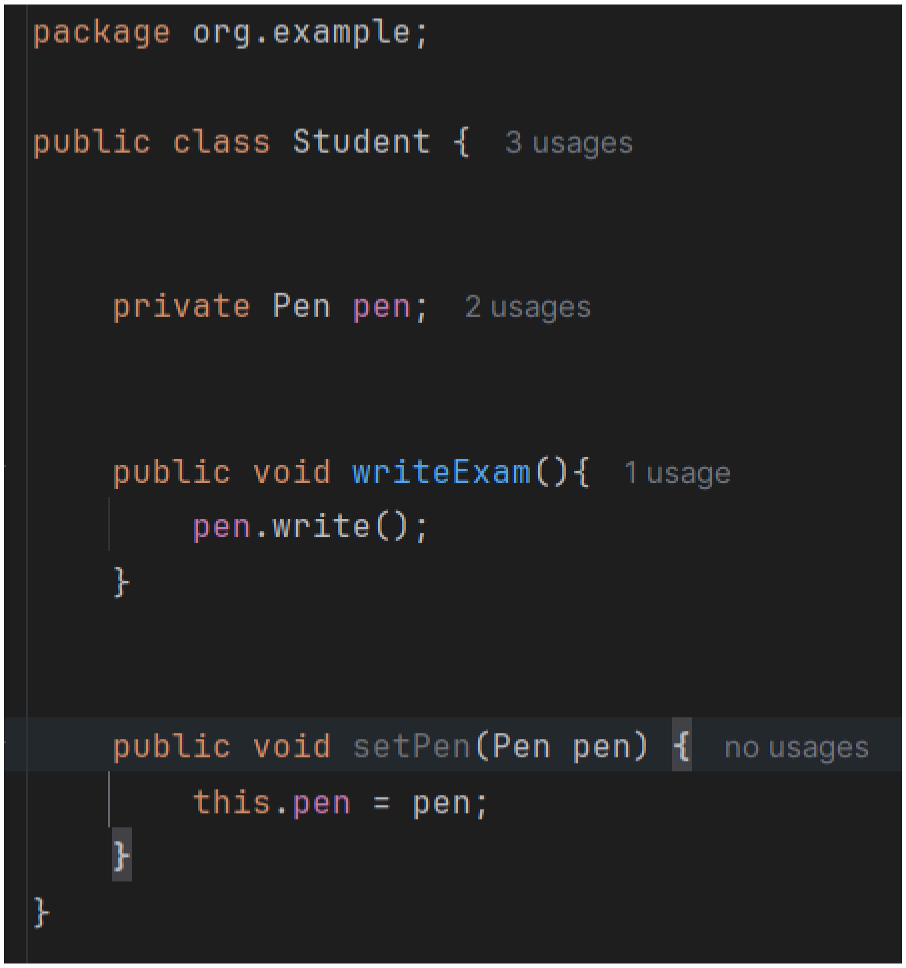
<property/> = calling setter method (setter injection)

Inside property,
    Name = variable name
    Value = variable value

We can also inject a reference
Using object name, and object id (bean id)

Loose coupling :

Now, student class in purely dependent on pen



```
package org.example;

public class Student {  3 usages


    private Pen pen;  2 usages


    public void writeExam(){  1 usage
        pen.write();
    }


    public void setPen(Pen pen) {  no usages
        this.pen = pen;
    }
}
```

If there is no pen, student cant write exam

So we need loose coupling.

So
Create **Writer as Interface**

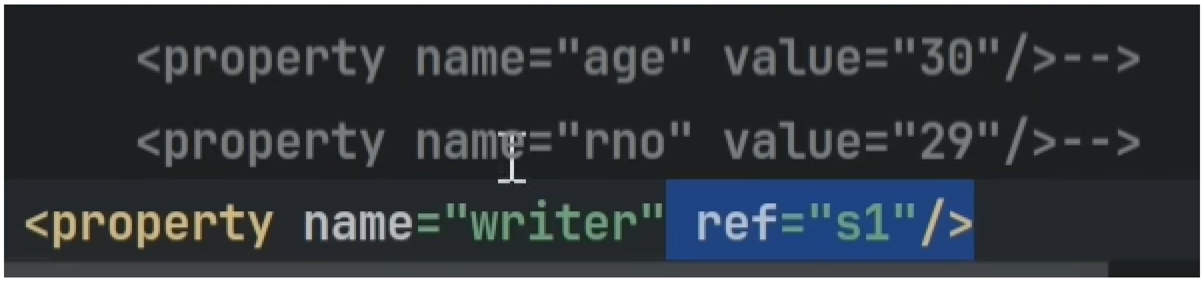
- Have multiple classes
- Pen
- Pencil
- Sketch
- marker

Everything denotes a writer, so no need to change in student class, this is called loose coupling

Applies to all concepts,
Like computer - Laptop, smart watch 👍

Autowiring :

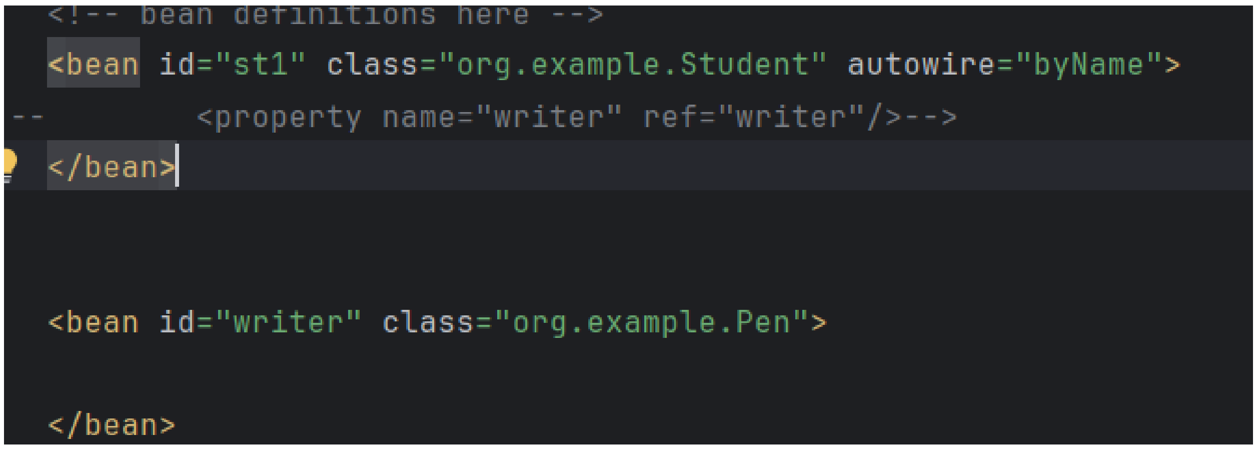
Wiring - connects name with value



```
    <property name="age" value="30"/>-->
    <property name="rno" value="29"/>-->
<property name="writer" ref="s1"/>
```

By name :

Object creation bean id name = ref id in property

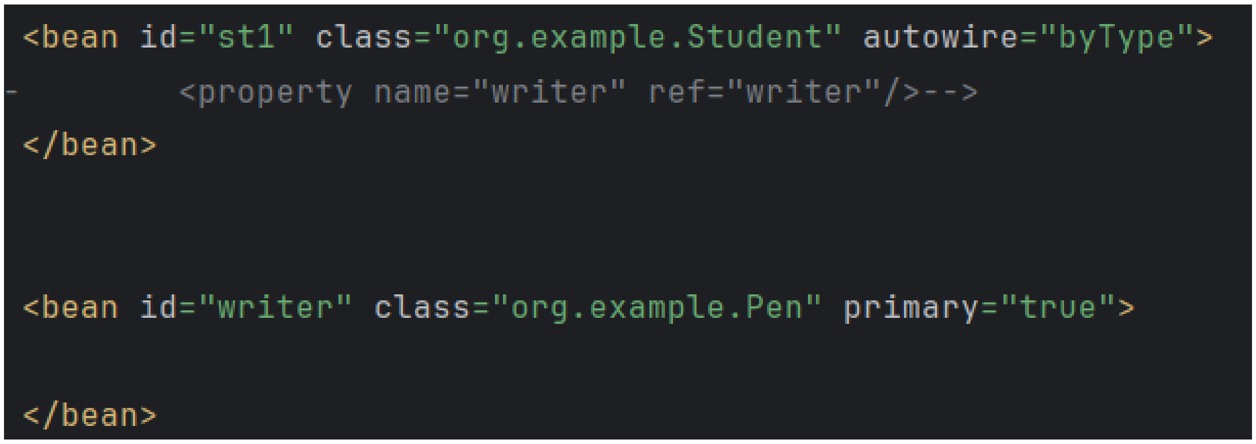


```
<!-- bean definitions here -->
<bean id="st1" class="org.example.Student" autowire="byName">
--        <property name="writer" ref="writer"/>-->
</bean>


<bean id="writer" class="org.example.Pen">

</bean>
```

By type :

In student class, in setter method, we said we are setting a writer type
So config file sees all beans which is writer, thats it 👍

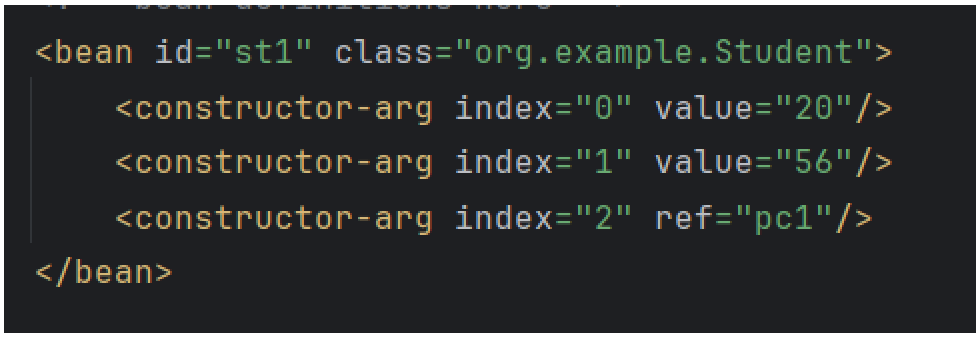
More than one writer? Error
So keep one as primary



```
<bean id="st1" class="org.example.Student" autowire="byType">
-        <property name="writer" ref="writer"/>-->
</bean>



<bean id="writer" class="org.example.Pen" primary="true">

</bean>
```

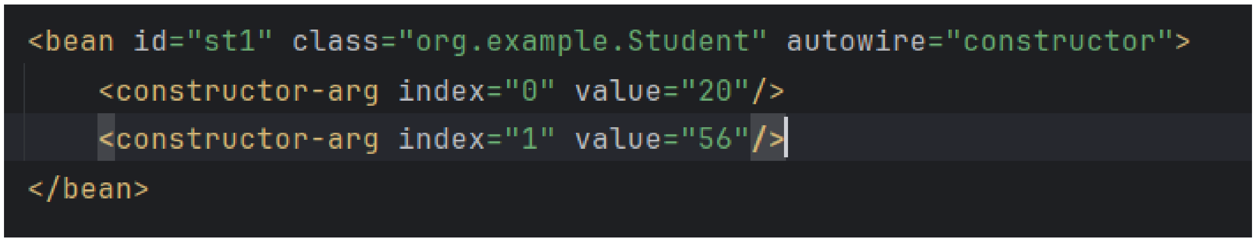
Construction injection :

We have variables, objects in student class
So to set values, we use constructors using spring



```xml
<bean id="st1" class="org.example.Student">
    <constructor-arg index="0" value="20"/>
    <constructor-arg index="1" value="56"/>
    <constructor-arg index="2" ref="pc1"/>
</bean>
```

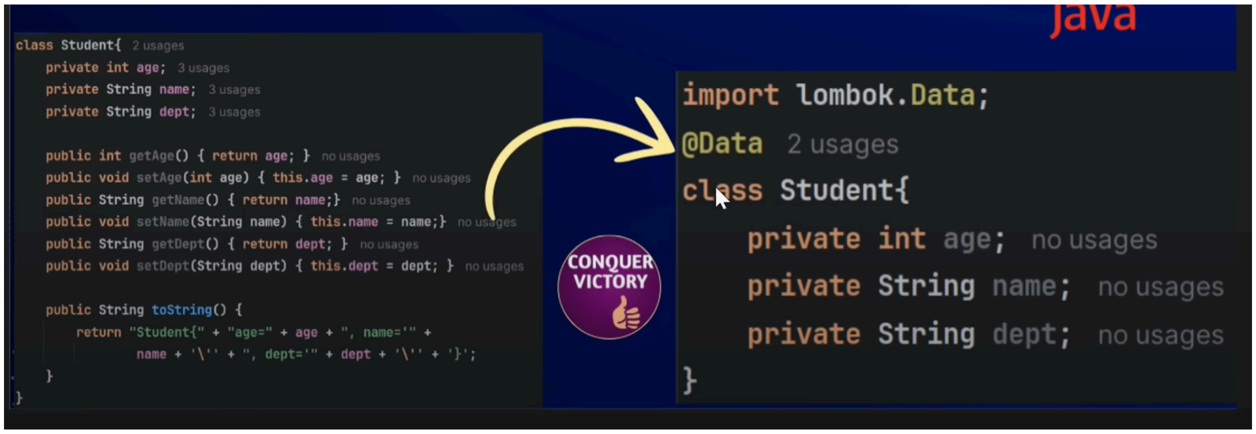
Autowire in constructor injection -> only using **autowire="constructor"**



```xml
<bean id="st1" class="org.example.Student" autowire="constructor">
    <constructor-arg index="0" value="20"/>
    <constructor-arg index="1" value="56"/>
</bean>
```

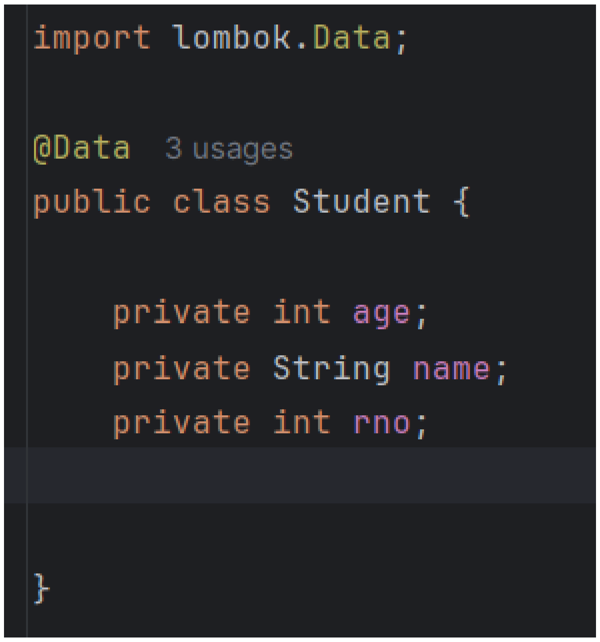
Lombok :

To minimize coding



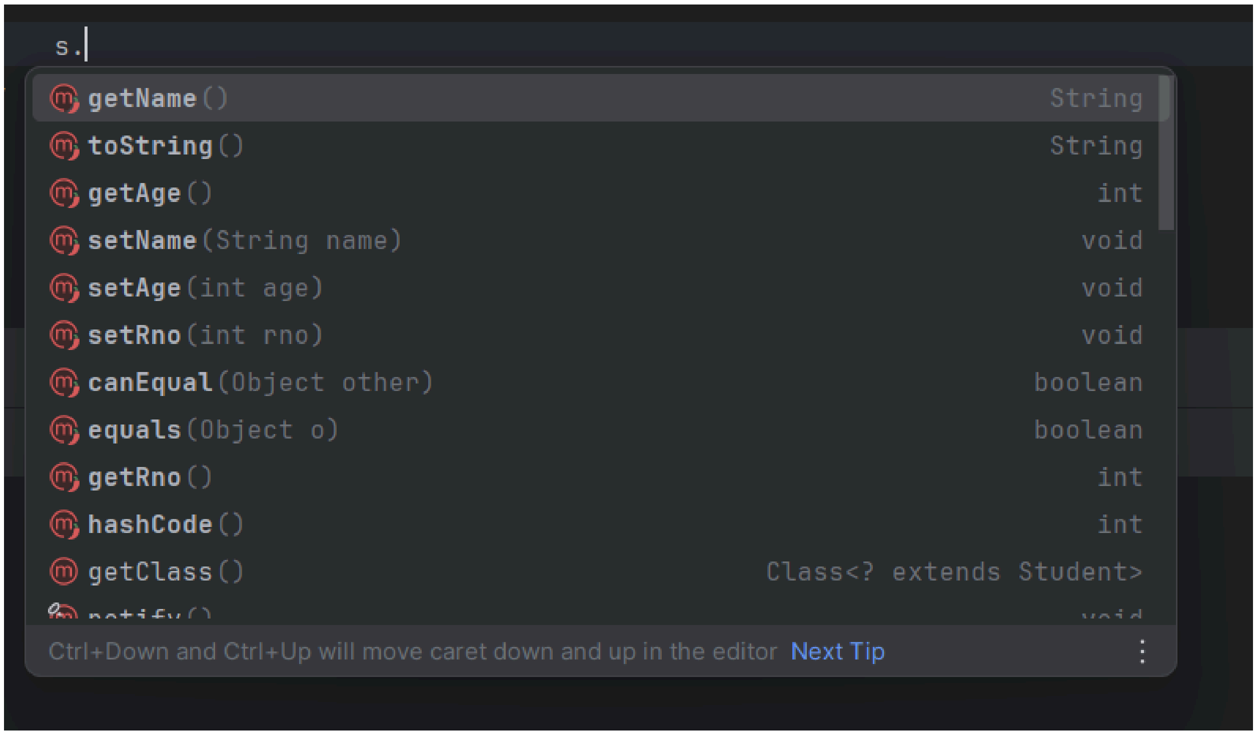
```
class Student{  2 usages
    private int age;  3 usages
    private String name;  3 usages
    private String dept;  3 usages

    public int getAge() { return age; }  no usages
    public void setAge(int age) { this.age = age; }  no usages
    public String getName() { return name;}  no usages
    public void setName(String name) { this.name = name;}  no usages
    public String getDept() { return dept; }  no usages
    public void setDept(String dept) { this.dept = dept; }  no usages

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

```
import lombok.Data;
@Data  2 usages
class Student{
    private int age;  no usages
    private String name;  no usages
    private String dept;  no usages
}
```



```
import lombok.Data;

@Data  3 usages
public class Student {

    private int age;
    private String name;
    private int rno;


}
```

In default, it implements all getters, setters and more methods



```
s.
getName()                                  String
toString()                                 String
getAge()                                   int
setName(String name)                       void
setAge(int age)                            void
setRno(int rno)                            void
canEqual(Object other)                     boolean
equals(Object o)                           boolean
getRno()                                   int
hashCode()                                 int
getClass()                Class<? extends Student>
```

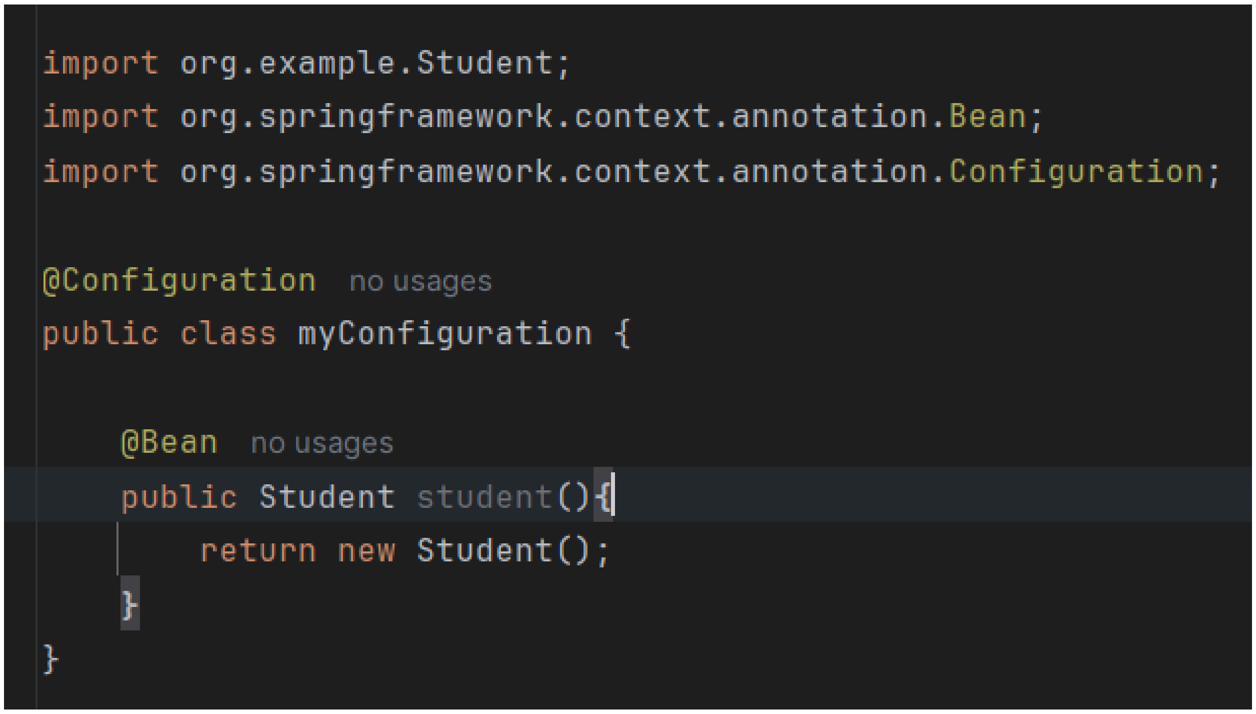
Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip

Java based config :

Instead of an xml file, we can have **separate java class** to write config using java.

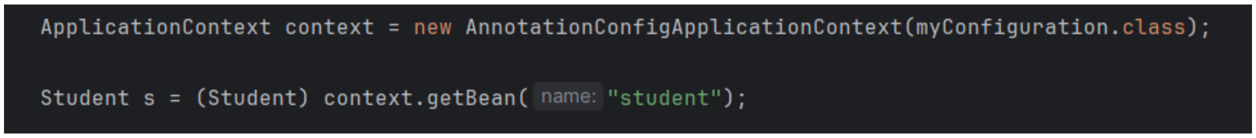
@ bean is used to annotate

Also at beginning, in application context, we should mention annotation instead of xml



```java
import org.example.Student;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration  no usages
public class myConfiguration {

    @Bean  no usages
    public Student student(){
        return new Student();
    }
}
```

We can create n number of objects using this config



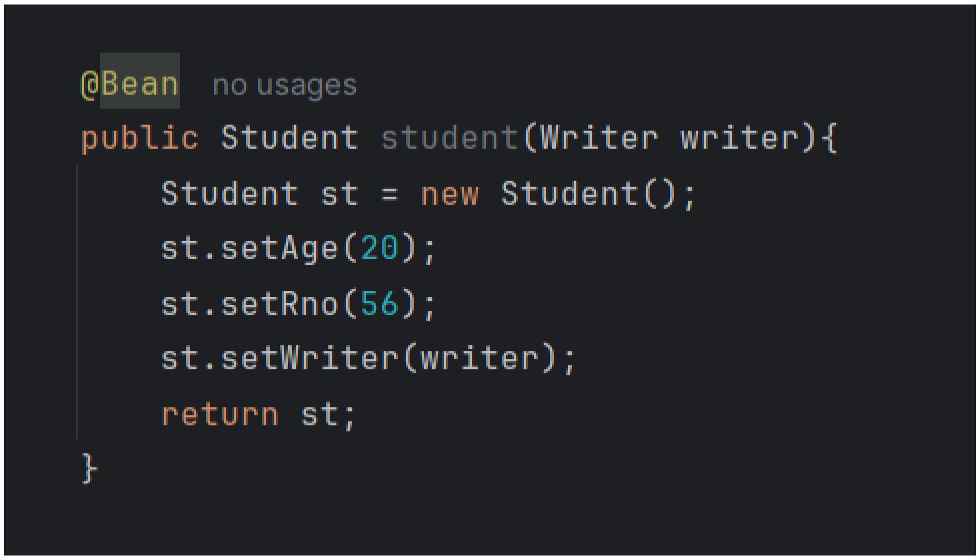
```java
ApplicationContext context = new AnnotationConfigApplicationContext(myConfiguration.class);

Student s = (Student) context.getBean( name: "student");
```

Same like xml to inject

Use **method name** of that config file when injecting in student class

We mentioned **Writer writer** when creating student object itself,
It's called **autowiring** in java based.

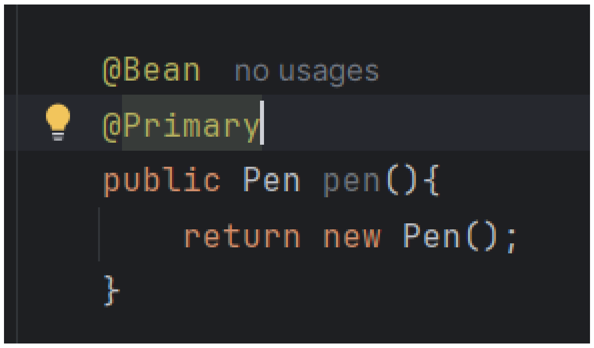
Automatically inject 👍



```
@Bean  no usages
public Student student(Writer writer){
    Student st = new Student();
    st.setAge(20);
    st.setRno(56);
    st.setWriter(writer);
    return st;
}
```

What if more than one writer presents ? 😣

Use **@ primary**



```
@Bean  no usages
@Primary
public Pen pen(){
    return new Pen();
}
```

Stereotype annotations :

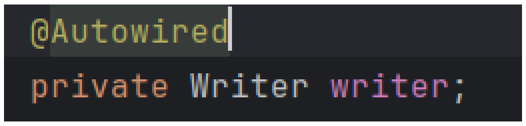
Bean = objection creation

@ Component in top of all classes where we need automatic object creation, managing, deletion by spring

@ ComponentScan in top of java config class
It says that scan all classes where it has @ component

Create objects for them 👍

Field injection :

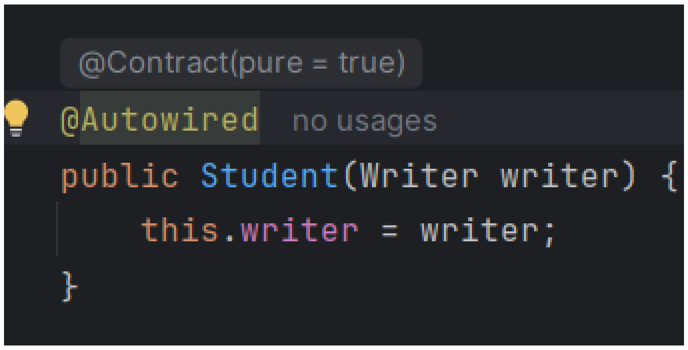


```
@Autowired
private Writer writer;
```

Setter injection :

Use @ autowired before setter method

Constructor injection :



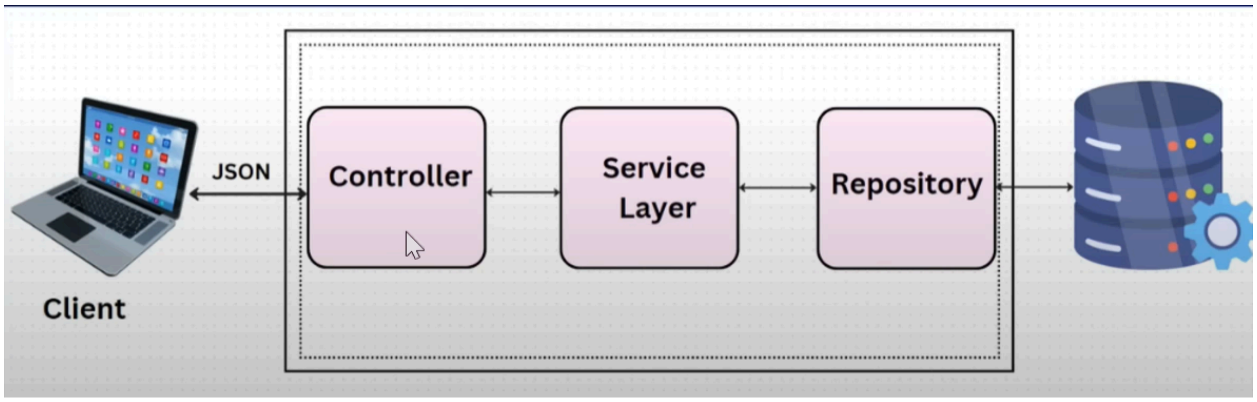
```
@Contract(pure = true)
@Autowired  no usages
public Student(Writer writer) {
    this.writer = writer;
}
```

Spring Boot :

Convention over configuration
Embedded server - tomcat

REST stands for REpresentational State Transfer
Api stands for Application Programming Interface

API :



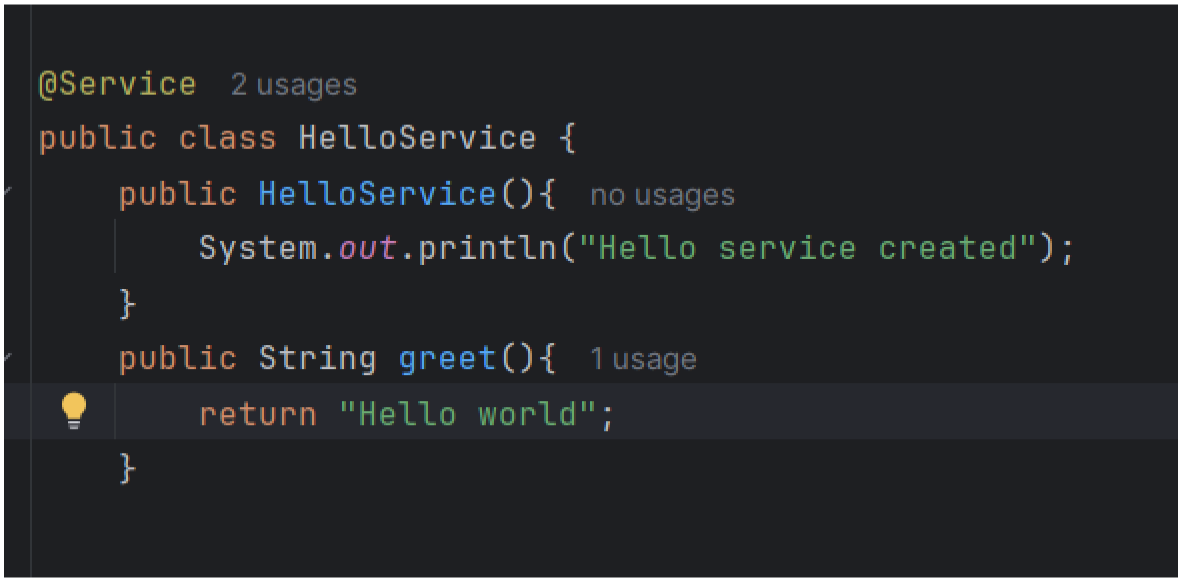
Controller :

It denotes the web page, like for “/” it shows the home page
For “/about” it goes to about page

@ GetMapping() is used to denote API
@ Controller should be mentioned in top of the class which is a controller class

Service :

It is the main layer, like we write the most of the **java code**
Control layer gets the code from here
And **sends to the client**

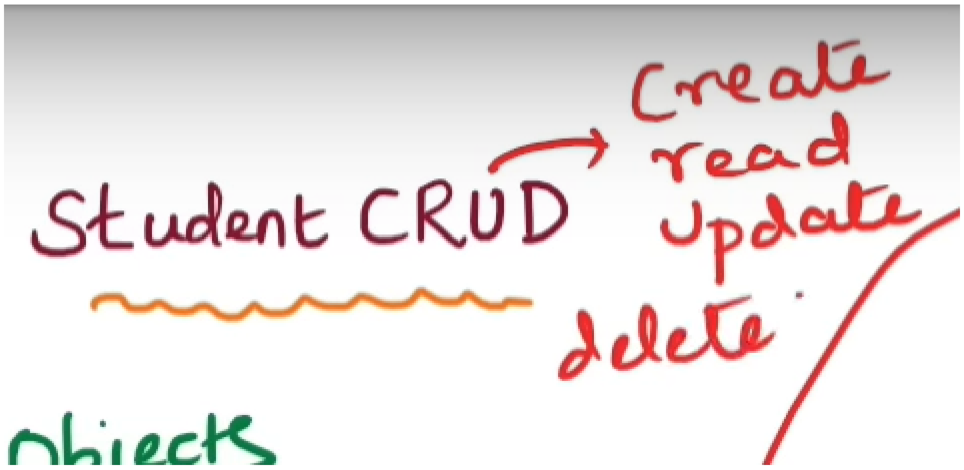


```java
@Service  2 usages
public class HelloService {
    public HelloService(){  no usages
        System.out.println("Hello service created");
    }
    public String greet(){  1 usage
        return "Hello world";
    }
```

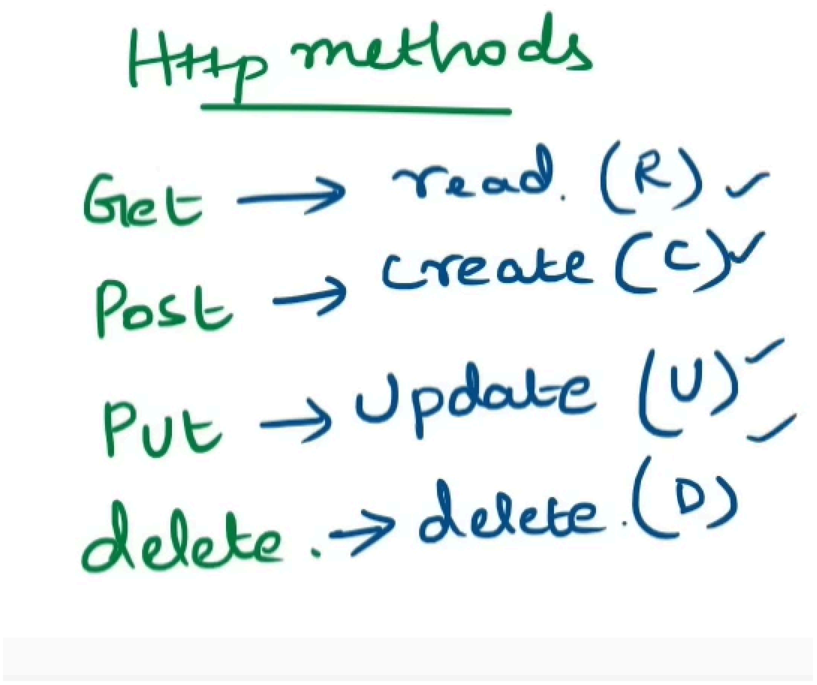
@ service

Automatically creates object for this class
We can write any java code
And we can use in controller classes

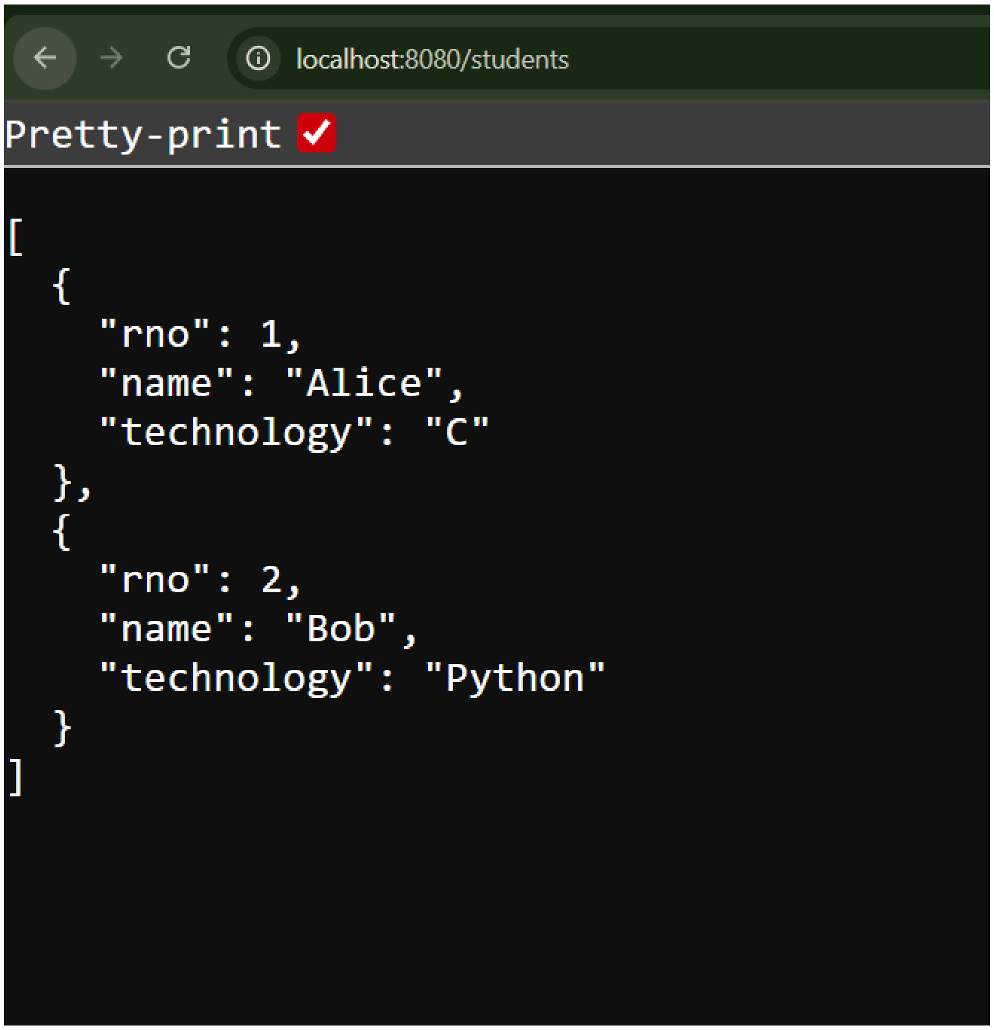
CRUD :



- Create
- Read
- Update
- Delete



Get method - Op in **JSON** format

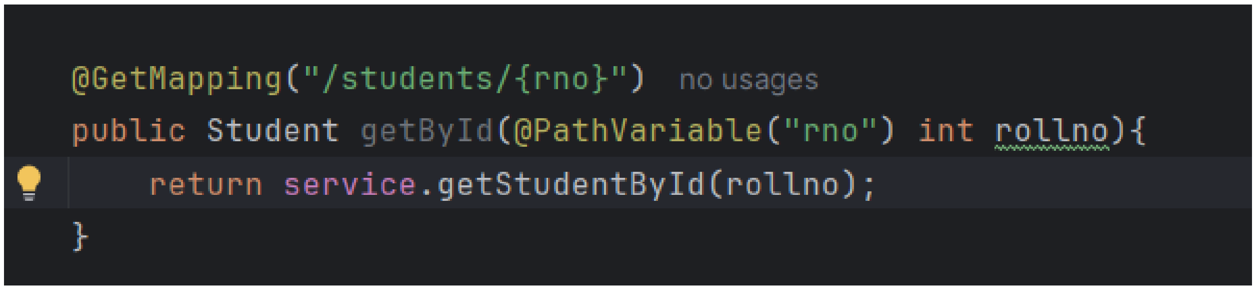


localhost:8080/students

Pretty-print ✓

```
[
  {
    "rno": 1,
    "name": "Alice",
    "technology": "C"
  },
  {
    "rno": 2,
    "name": "Bob",
    "technology": "Python"
  }
]
```

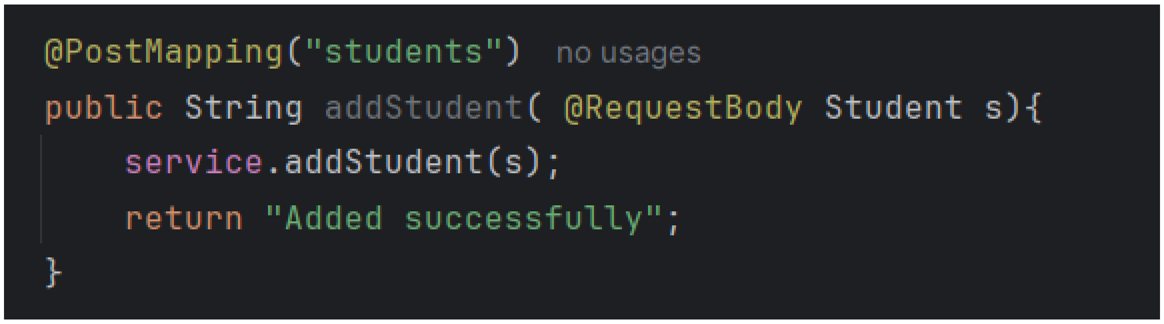
Get with id :



```
@GetMapping("/students/{rno}")  no usages
public Student getById(@PathVariable("rno") int rollno){
    return service.getStudentById(rollno);
}
```

We use pathvariable to get that data

Post mapping :

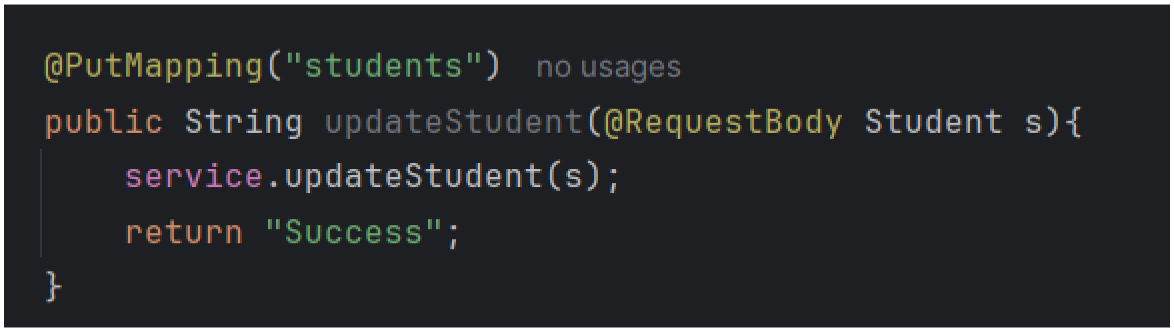


```
@PostMapping("students")  no usages
public String addStudent( @RequestBody Student s){
    service.addStudent(s);
    return "Added successfully";
}
```

@ RequestBody is used to mention we are requesting a data to add in our service class list

Put Request :

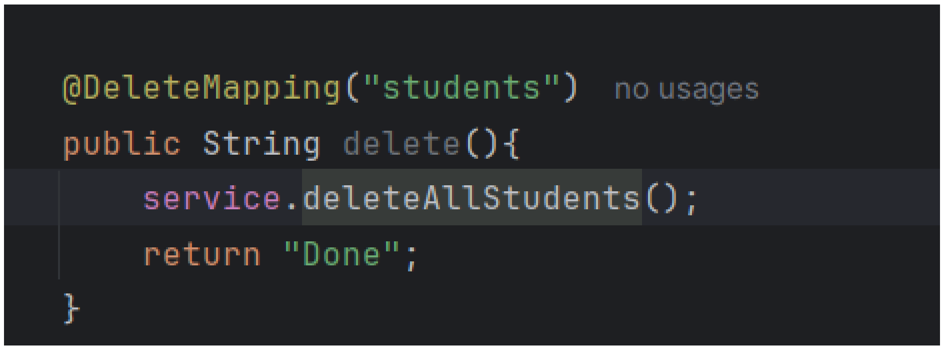
Same like post



```
@PutMapping("students")  no usages
public String updateStudent(@RequestBody Student s){
    service.updateStudent(s);
    return "Success";
}
```

Getting update student object
Matching with the current object in students List

Delete Request :



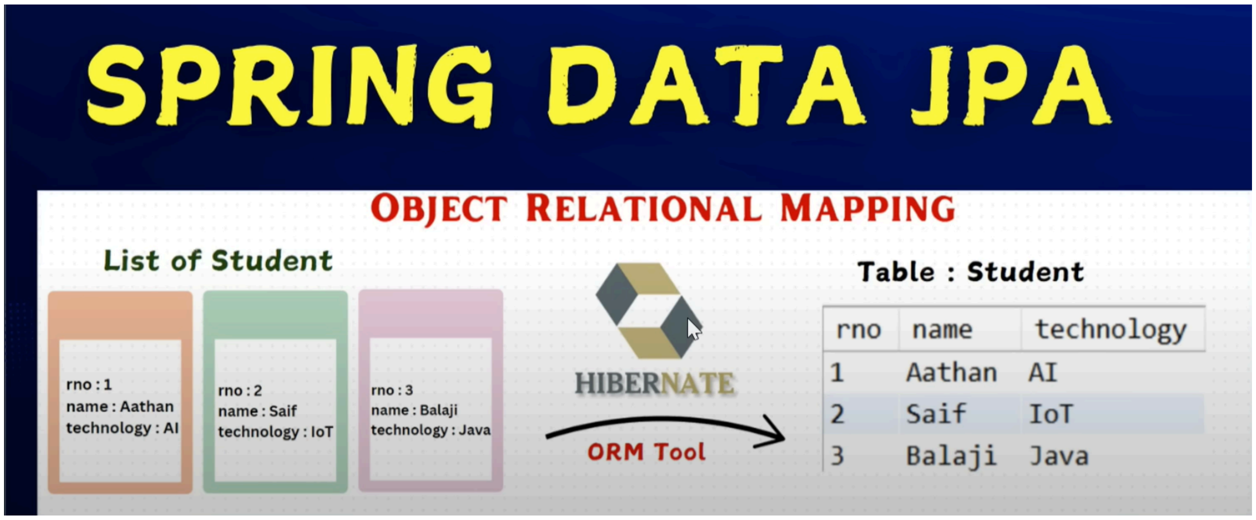
```
@DeleteMapping("students")  no usages
public String delete(){
    service.deleteAllStudents();
    return "Done";
}
```

We can do the work in service layer

To delete only 1 student

@ DeleteMapping("students/{rno}")
Then matching rollno will be deleted in list

Data Jpa :



rno	name	technology
1	Aathan	AI
2	Saif	IoT
3	Balaji	Java

MAGIC !!!!!!!!

Easier than sql queries ? YESSSS
ORM

ORM

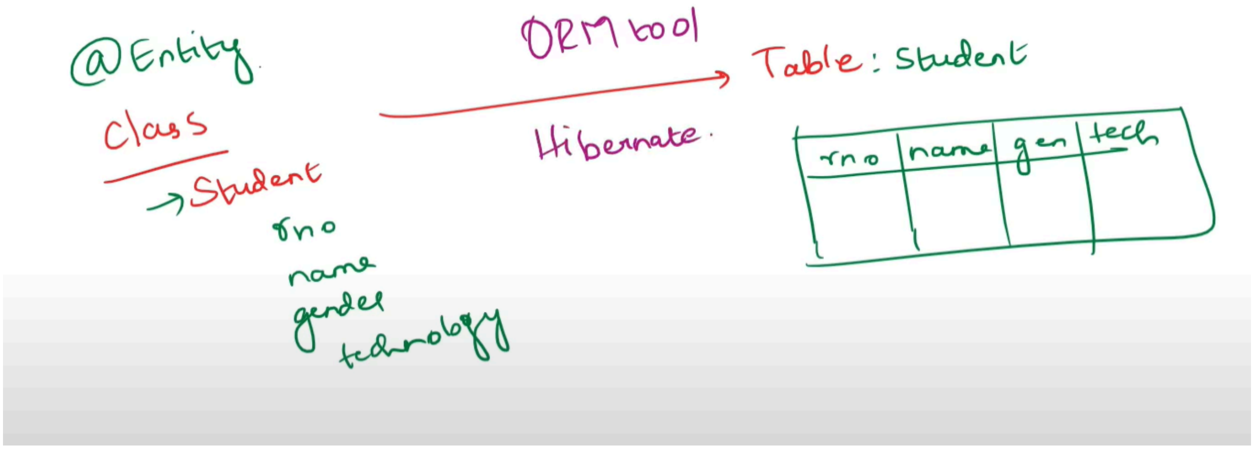
Object Relational Mapping

Orm tools :

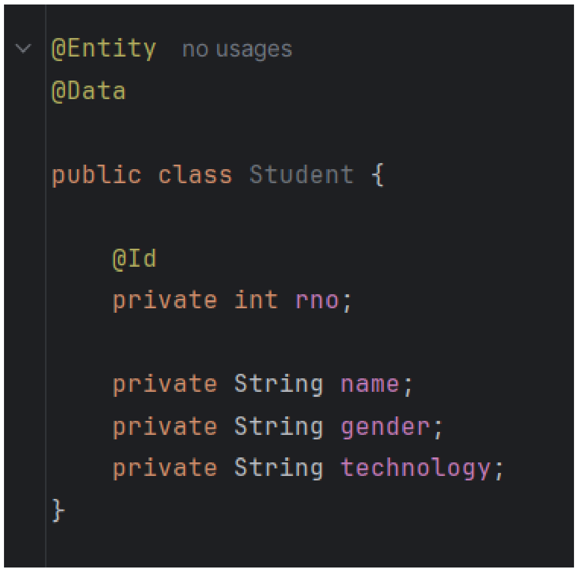
- HIbernate
- Sql alchemy
- Entity framework
- Eclipse link

We use Hibernate 👍
Java Persistence APi = JPA

In model package :
- Mention @ entity
- Class will be converted into table
- Variables as table values

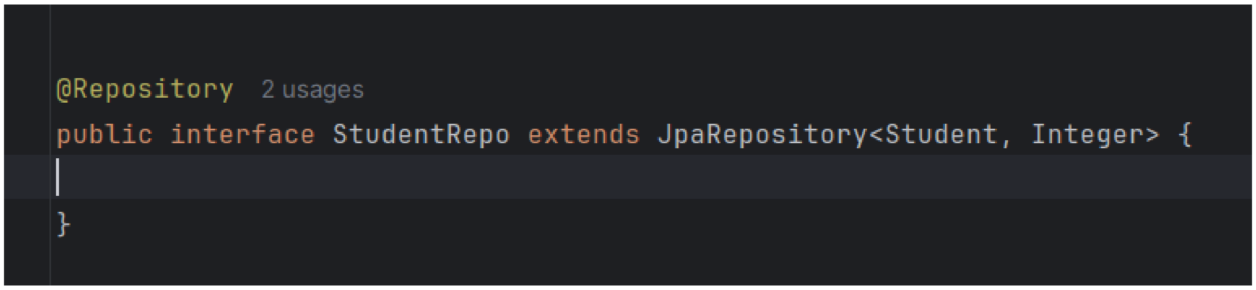


We can mention the primary key which will be used for SQL DBMS



```
@Entity  no usages
@Data

public class Student {

    @Id
    private int rno;

    private String name;
    private String gender;
    private String technology;
}
```

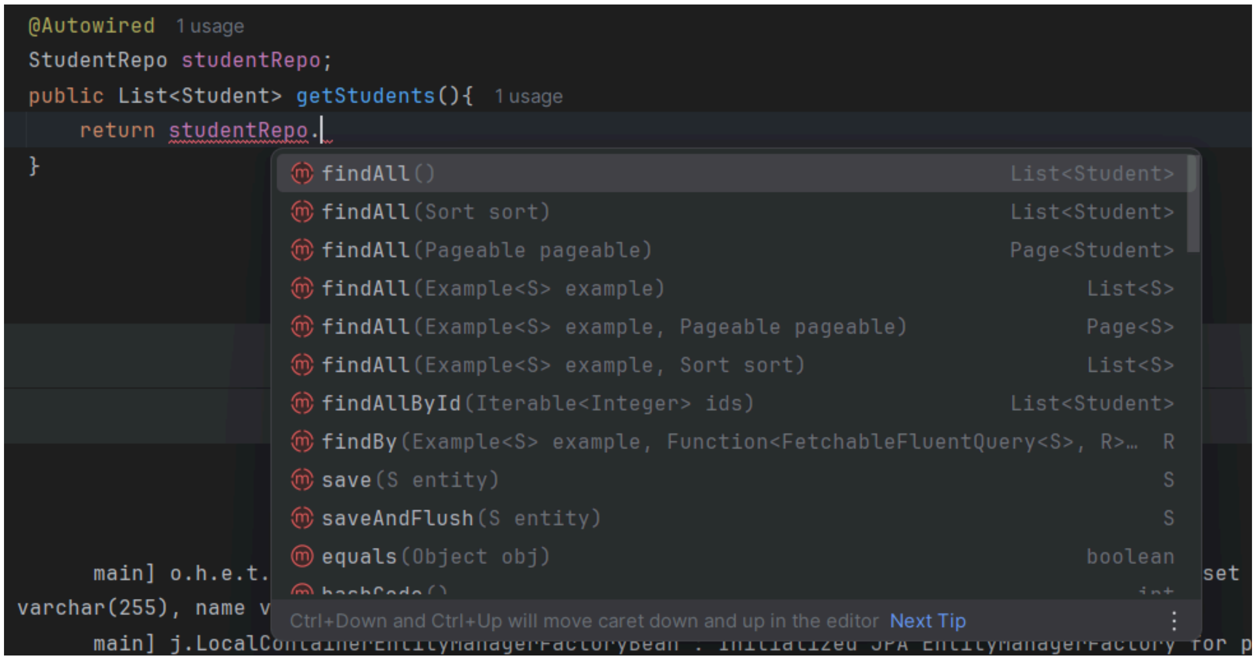
In Repo Interface



```
@Repository  2 usages
public interface StudentRepo extends JpaRepository<Student, Integer> {

}
```

Extend JpaRepository
And
<Student, Integer>

- Student : Entity to create table
- Integer : datatype of primary key



```
@Autowired  1 usage
StudentRepo studentRepo;
public List<Student> getStudents(){  1 usage
    return studentRepo.
}
```

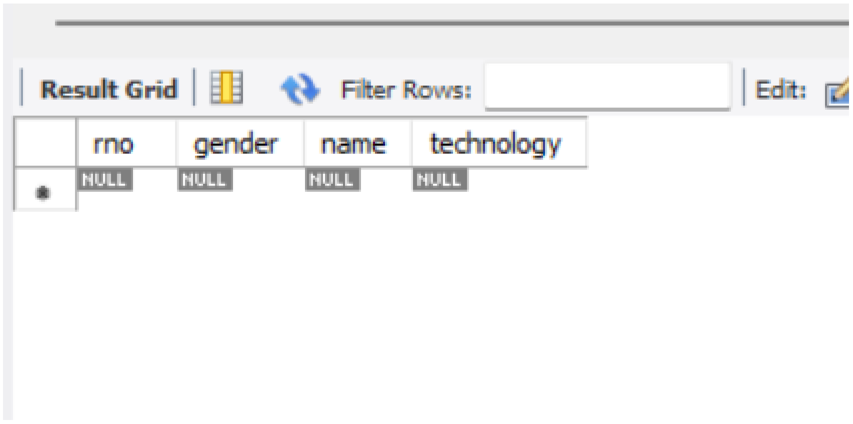
```
findAll()                                                          List<Student>
findAll(Sort sort)                                                 List<Student>
findAll(Pageable pageable)                                         Page<Student>
findAll(Example<S> example)                                              List<S>
findAll(Example<S> example, Pageable pageable)                           Page<S>
findAll(Example<S> example, Sort sort)                                   List<S>
findAllById(Iterable<Integer> ids)                                 List<Student>
findBy(Example<S> example, Function<FetchableFluentQuery<S>, R>…               R
save(S entity)                                                                 S
saveAndFlush(S entity)                                                         S
equals(Object obj)                                                       boolean
hashCode()                                                                   int
```

Ctrl+Down and Ctrl+Up will move caret down and up in the editor Next Tip

```
     main] o.h.e.t.                                                                  set
varchar(255), name v
     main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for p
```

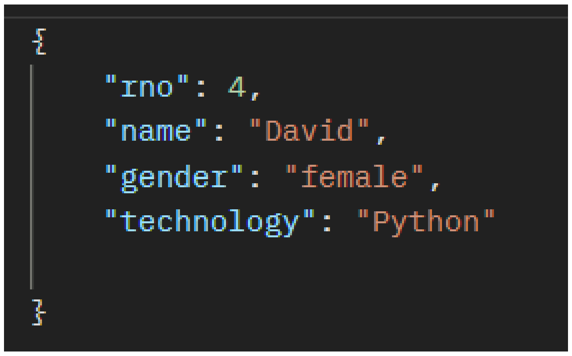
That JpaRepository gives all these functions

MAGIC !!!



	rno	gender	name	technology
*	NULL	NULL	NULL	NULL

Table created in SQL drive

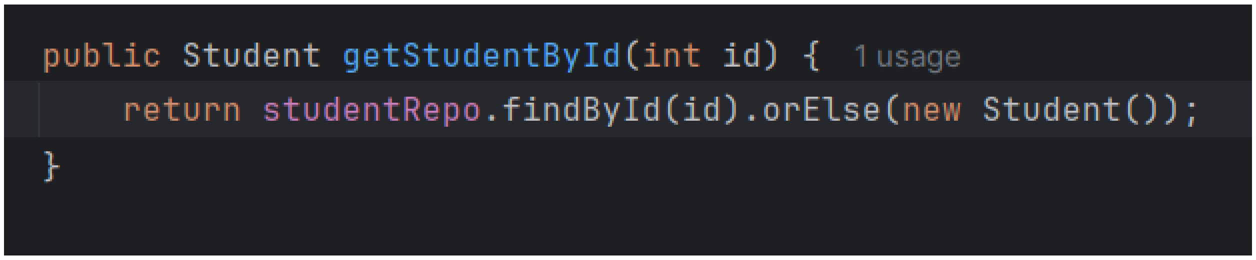


```
{
    "rno": 4,
    "name": "David",
    "gender": "female",
    "technology": "Python"

}
```

In postMapping,
If we give rno as 4 for 2 Entities (Student)
It would delete old data and add the new

Becase **rno is primary**

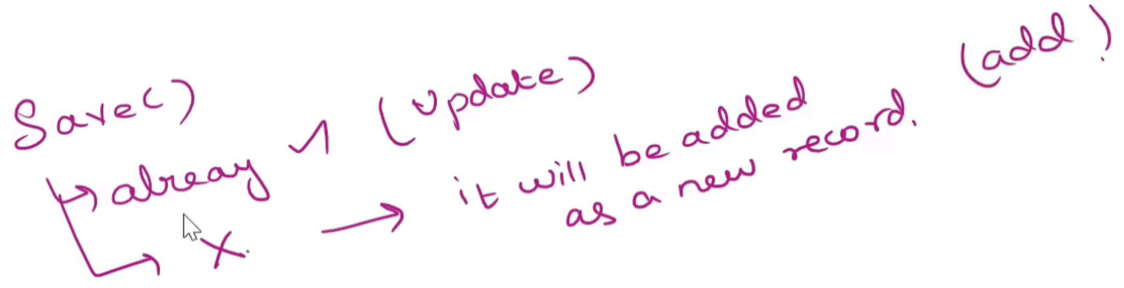
GetMapping with id :



```
public Student getStudentById(int id) {  1 usage
    return studentRepo.findById(id).orElse(new Student());
}
```

Normally, we traversed full list
But now **Jpa does** for us from the table 👍

In put mapping also,
We use **.save() to update in db through repo**



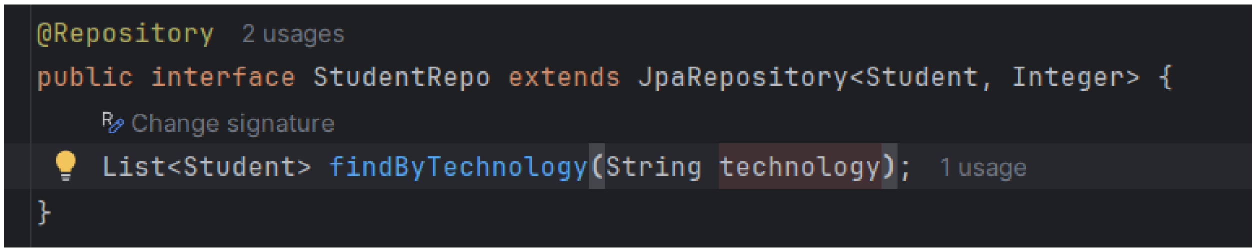
If already exists -> updates
Else -> creates new entity

Custom Methods in Repo class :

Methods in Repo is limited
findById() is available by default bcz its primary key.

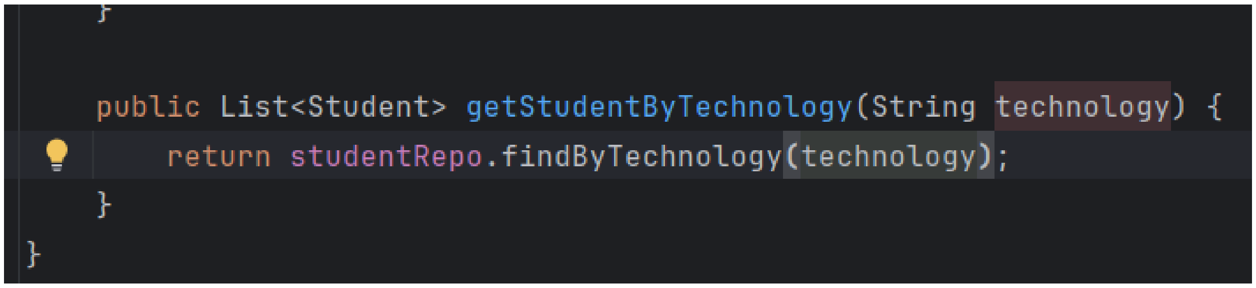
So we can create custom methods using

getBy + attribute -> getByTechnology()
-> getByGender()



```
@Repository  2 usages
public interface StudentRepo extends JpaRepository<Student, Integer> {
    Change signature
    List<Student> findByTechnology(String technology);  1 usage
}
```

No need to write body of the method



```
    }

    public List<Student> getStudentByTechnology(String technology) {
        return studentRepo.findByTechnology(technology);
    }
}
```

We can use in service class using Repo object

Connecting with Front end :

Post :
Just get data using a form in HTML, and
Top of the form, action = **(Url in backend)**
localhost://enroll-course

That’s it
Done

Delete :

Just in buttons in HTML, pass the Url in backend to delete mapping

Get Mapping is little bit harder
Bcz, we are fetching data from server

In, Js
fetch(GET url)
.then() convert to json
.then() get data as JSON arrays

We can traverse this JSON, and view as table in HTML + JS