Autowire:

- @Component: is used to create a bean, this will create the bean and initialize the obj even if we don't specifically call it anywhere
- @Scope(value=” prototype ”): It will create the bean/obj only when its demanded or asked for, if we don't ask for it, it won't create the bean
- @Autowired: from what I understood, it is used to not use the new keyword to instantiate the object.

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
```

Simple h2 configs

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
</dependency>
```

/h2-console to get h2 db console

- @Service: Added it ot the service implementation and not the interface.

Streams:
Streams cannot be used multiple times
.stream method can be applied to Lists
And it can be chained with other methods

```
List<Integer> newList = integerList.stream()
        .map(num-> num*2)
        .toList();
```

Map impl to double the numbers

Predicate -> returns a boolean, used for filtering and similar processing (test)

- Produces booleans

Supplier -> returns a value of type T, you cannot send a parameter to it. (get)
Consumer -> takes in argument but **does not** produce a result (accept)
Function -> takes in an argument and returns a result (apply)

`Supplier<T>`	`() -> "This is a message"`
`Consumer<T>` `BiConsumer<T, U>`	`s -> System.out.println(s)` `(element, sink) -> sink.accept(element)`
`Predicate<T>` `BiPredicate<T, U>`	`s -> s.length() > 0` `(i1, i2) -> i1 > i2`
`Function<T, R>` `UnaryOperator<T>`	`s -> s.length()` `s -> s.toUpperCase()`
`BiFunction<T, U, R>` `BinaryOperator<T>`	`(word, sentence) -> sentence.indexOf(word)` `(i1, i2) -> i1 + i2`

Threads
States of thread
New -> Runnable -> Running -> Terminated

How to implement multiple inheritance in Java??
If the class is already a child of another class you can use interfaces as a substitute to multiple inheritance

Middlewares (Interceptors, filters, AOPs)
Filters work BEFORE they reach the controllers (registration optional)
Interceptors work before AND after the controller methods (have to register this also annotate the config with @Configurable)
Aspect Oriented Programming used for cross-cutting concerns?? Adds several life cycle points to add methods and processing

Mapping between entities:
Take emp and address entity

Emp would map one-to-one mapping with address like so. This is unidirectional
@OneToOne(cascade = CascadeType.ALL) //this will save to address class when adding to emp class
@JoinColumn(name = “fk_add_id) //this will create column with custom name
Private Address address;

In bi-directional we have owning side and referencing side
The class which has @OneToOne is the owning side
Then in the reference class add

@OnetoOne(mappedBy = “address”)
Private Employee employee;

If one emp can have multiple addresses
In emp class add
@OnetoMany(cascade = CascadeType.ALL)
@JoinColumn(name = “fk_emp_id”, referencedColumn = “SameColumnNameAsIdOfEmp”)
Private List<Address> address;
In Address, don't reference employee

add_id	address_type	city	fk_emp_id
25	Current	Banglore	24
26	Permanent	Banglore	24
28	Current	Pune	27
29	Permanent	Delhi	27

@ManyToOne
Many emps (child) to one dep(parent)

```
@ManyToOne
@JoinColumn(name = "dep_id", nullable = false)
private Department department;
```