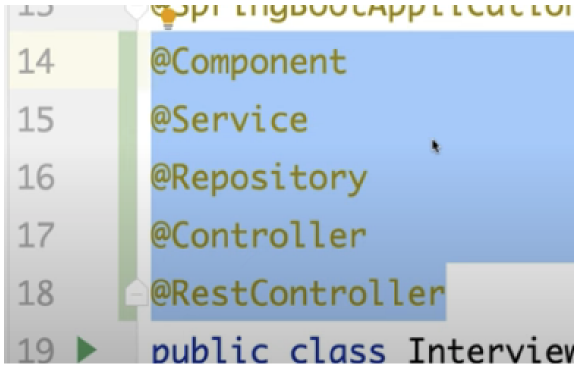
11. Can you explain the purpose of Stereotype annotations in the Spring Framework ?

Stereotype annotations indicate the purpose of the class.
The five are the stereotype annotations. Component is the parent interface and the rest are the child interface of Component.



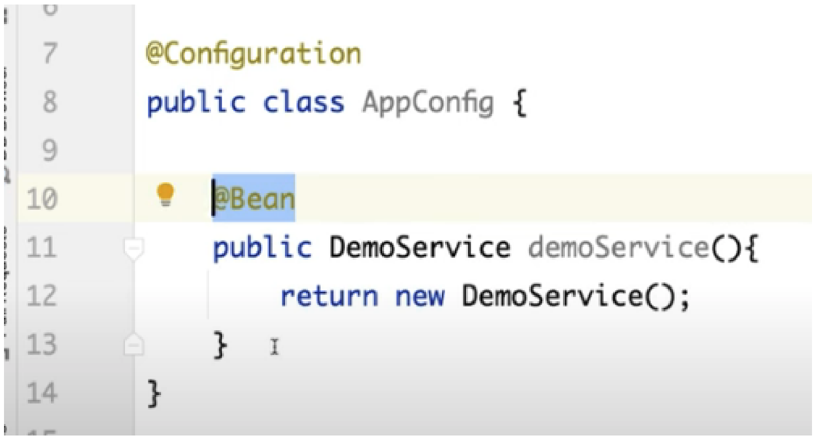
```
@SpringBootApplication
@Component
@Service
@Repository
@Controller
@RestController
public class Interview
```

We can interchangeably use them like in case of service annotation we can use repository annotation. But that is not recommended.

12. How can you define bean in spring framework ?

1. Every stereotype annotation class is just a bean. This is one way of creating beans.

2. The below one is a java based configuration for creating a bean.

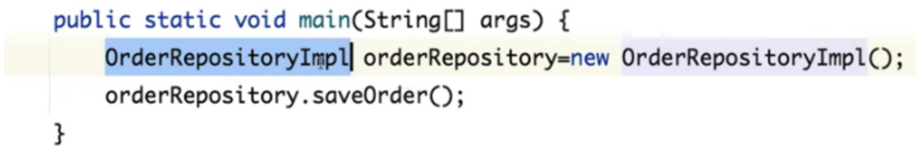


```
@Configuration
public class AppConfig {

    @Bean
    public DemoService demoService(){
        return new DemoService();
    }
}
```

13. What is dependency injection ?

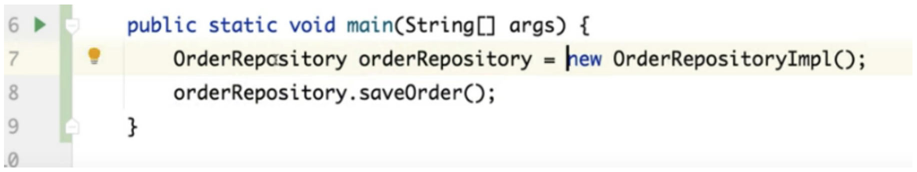
What is DI, types of DI and when to use what type of DI are important to know.

Before spring how was DI.



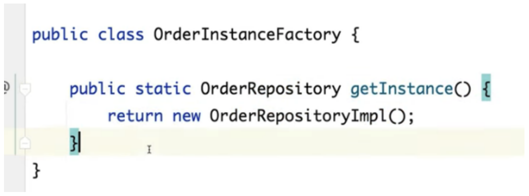
```java
public static void main(String[] args) {
    OrderRepositoryImpl orderRepository=new OrderRepositoryImpl();
    orderRepository.saveOrder();
}
```

To avoid tight coupling we create an interface and use it as shown below. This is partially achieving loose coupling but not complete.

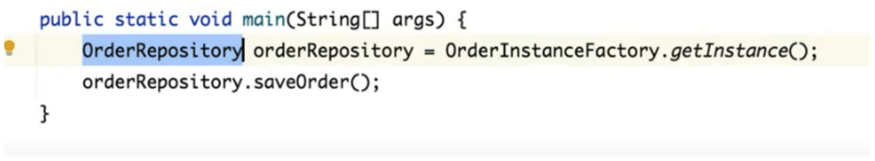


```java
public static void main(String[] args) {
    OrderRepository orderRepository = new OrderRepositoryImpl();
    orderRepository.saveOrder();
}
```

The below approach also partial loose coupling

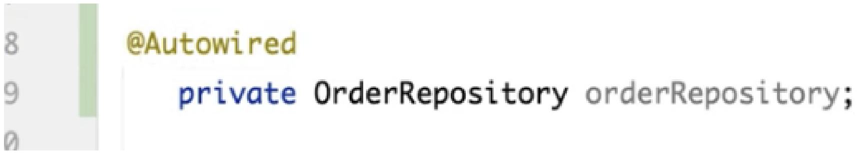


```java
public class OrderInstanceFactory {

    public static OrderRepository getInstance() {
        return new OrderRepositoryImpl();
    }
}
```



```java
public static void main(String[] args) {
    OrderRepository orderRepository = OrderInstanceFactory.getInstance();
    orderRepository.saveOrder();
}
```

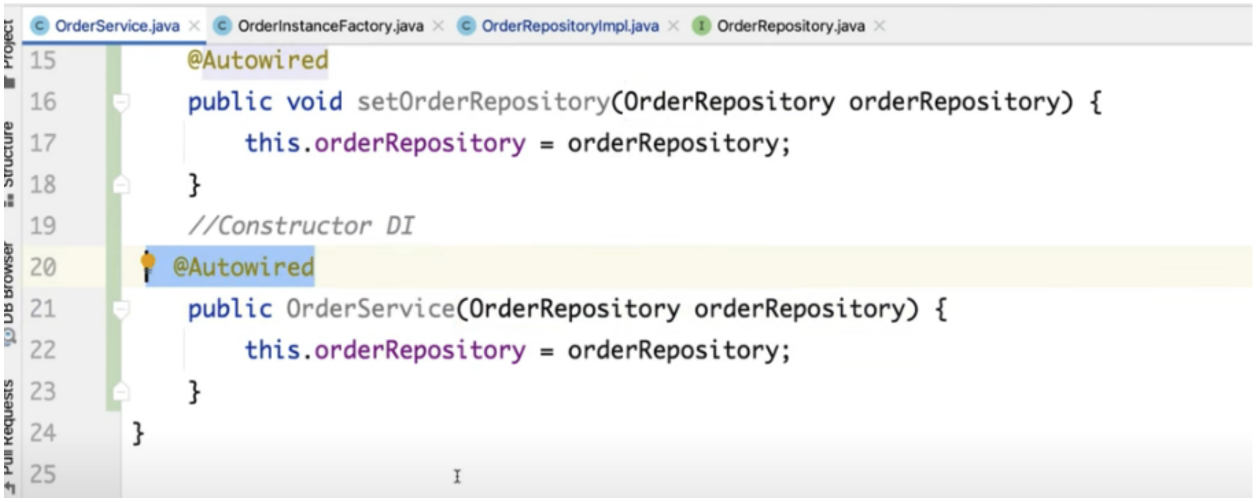
In every above case we are managing DI. So Spring came up with the solution of DI with annotation. From object creation to destruction the entire lifecycle is managed by Spring. This concept is the backbone of spring.



```java
@Autowired
    private OrderRepository orderRepository;
```

14. How many ways we can perform dependency injection in spring or spring boot ?

Field Level DI , Setter DI and constructor DI

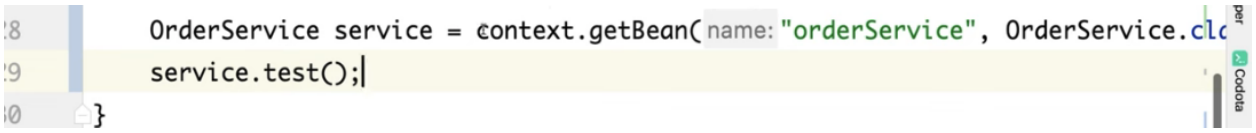


OrderService.java × OrderInstanceFactory.java × OrderRepositoryImpl.java × OrderRepository.java ×

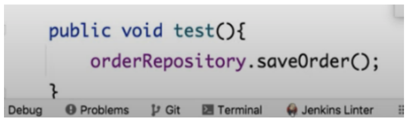
```java
    @Autowired
    public void setOrderRepository(OrderRepository orderRepository) {
        this.orderRepository = orderRepository;
    }
    //Constructor DI
    @Autowired
    public OrderService(OrderRepository orderRepository) {
        this.orderRepository = orderRepository;
    }
}
```

For constructor autowired is optional if we have a single dependency. If we have more than one then autowired is mandatory.

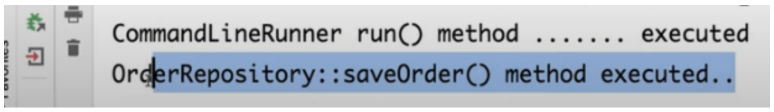
Just get the bean in the run method.



```java
        OrderService service = context.getBean( name: "orderService", OrderService.clc
        service.test();
}
```



```java
    public void test(){
        orderRepository.saveOrder();
    }
```

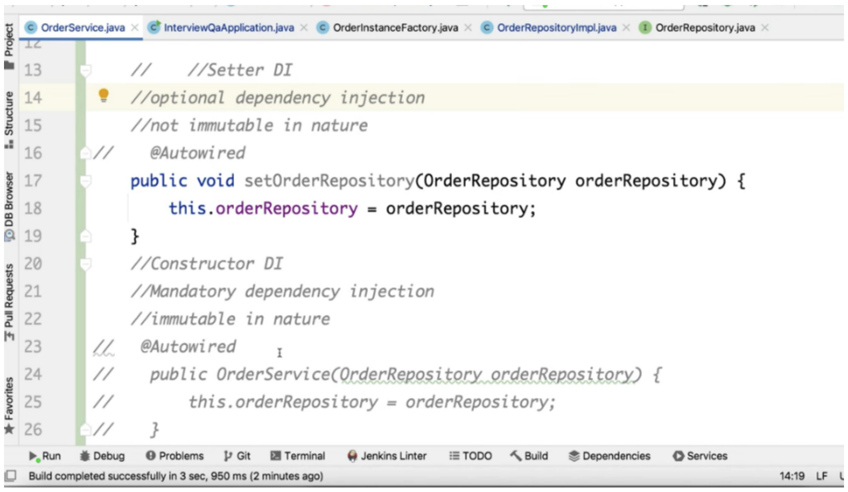


```
CommandLineRunner run() method ....... executed
OrderRepository::saveOrder() method executed..
```

Even though we have not created any bean, spring created it and managed it.

15. where you would choose to use setter injection over constructor injection, and vice versa?

When dependency is optional then we go for setter level DI . If dependency is mandatory then we go for constructor injection.

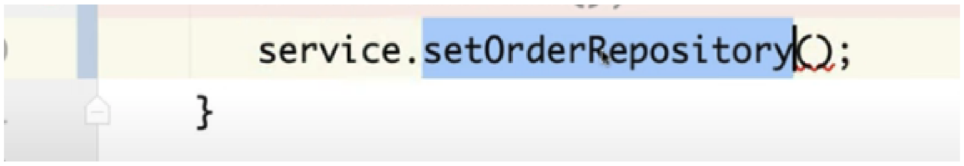


```java
//    //Setter DI
//optional dependency injection
//not immutable in nature
//    @Autowired
public void setOrderRepository(OrderRepository orderRepository) {
    this.orderRepository = orderRepository;
}
//Constructor DI
//Mandatory dependency injection
//immutable in nature
//  @Autowired
//    public OrderService(OrderRepository orderRepository) {
//        this.orderRepository = orderRepository;
//    }
```

Build completed successfully in 3 sec, 950 ms (2 minutes ago)

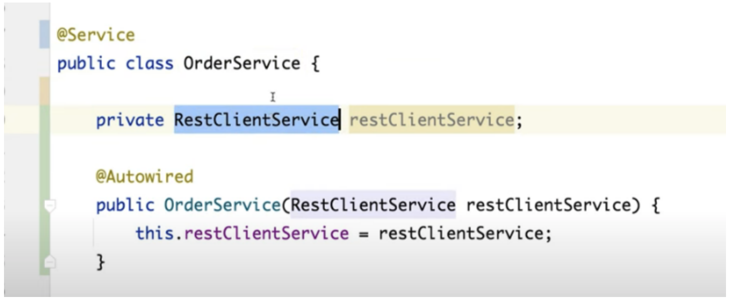
Dependencies are immutable after object creation using a constructor DI. because we won't get a chance to change object properties.

Where as setter DI mutable because we can have the choice of changing DI before calling setter injection with below statement

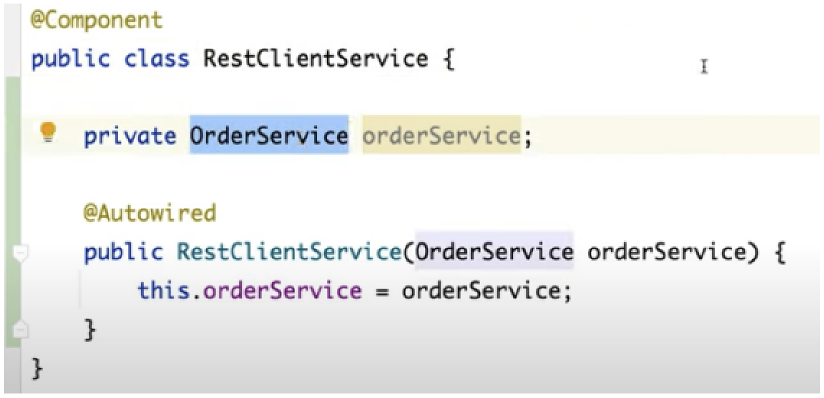


```java
        service.setOrderRepository();
    }
```

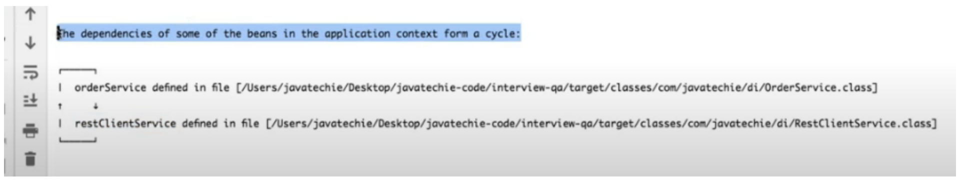
When we have Circular Dependency: we should not go for constructor DI it will show error like below



```java
@Service
public class OrderService {

    private RestClientService restClientService;

    @Autowired
    public OrderService(RestClientService restClientService) {
        this.restClientService = restClientService;
    }
```

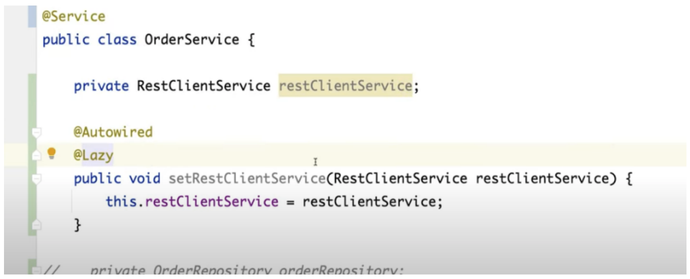


```java
@Component
public class RestClientService {

    private OrderService orderService;

    @Autowired
    public RestClientService(OrderService orderService) {
        this.orderService = orderService;
    }
}
```

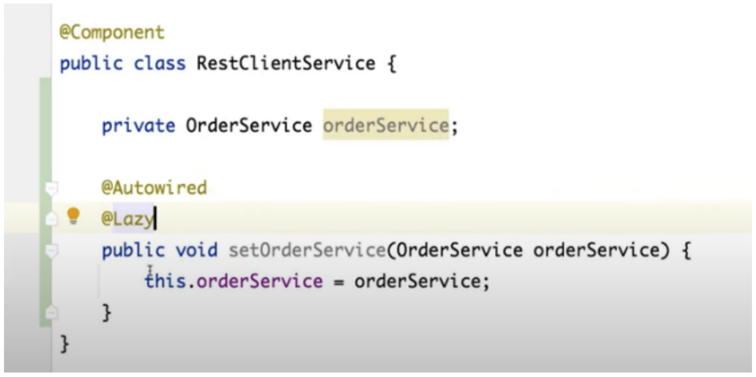


```
The dependencies of some of the beans in the application context form a cycle:

┌─────┐
|  orderService defined in file [/Users/javatechie/Desktop/javatechie-code/interview-qa/target/classes/com/javatechie/di/OrderService.class]
↑     ↓
|  restClientService defined in file [/Users/javatechie/Desktop/javatechie-code/interview-qa/target/classes/com/javatechie/di/RestClientService.class]
└─────┘
```

For this we have to use setter injection

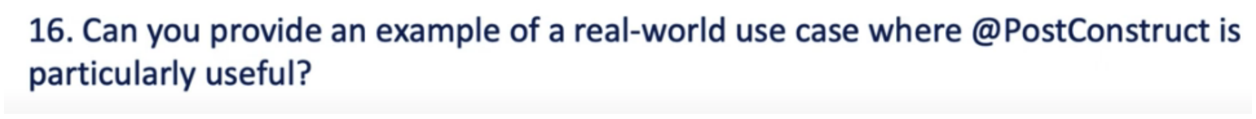


```java
@Service
public class OrderService {

    private RestClientService restClientService;

    @Autowired
    @Lazy
    public void setRestClientService(RestClientService restClientService) {
        this.restClientService = restClientService;
    }

//    private OrderRepository orderRepository;
```



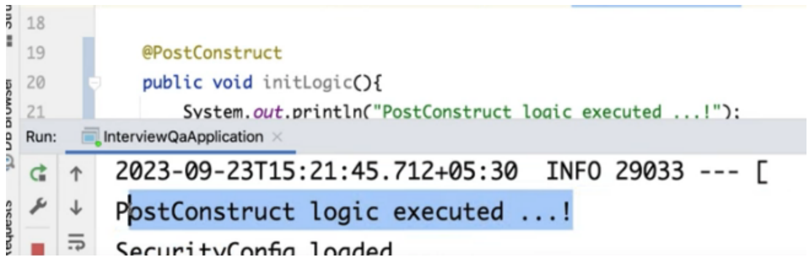
```java
@Component
public class RestClientService {

    private OrderService orderService;

    @Autowired
    @Lazy
    public void setOrderService(OrderService orderService) {
        this.orderService = orderService;
    }
}
```

These logics we implement based on our business requirements.



16. Can you provide an example of a real-world use case where @PostConstruct is particularly useful?

When we have spring boot run and command line runner and post construct methods first it run springboot run method and post construct and last command line run.

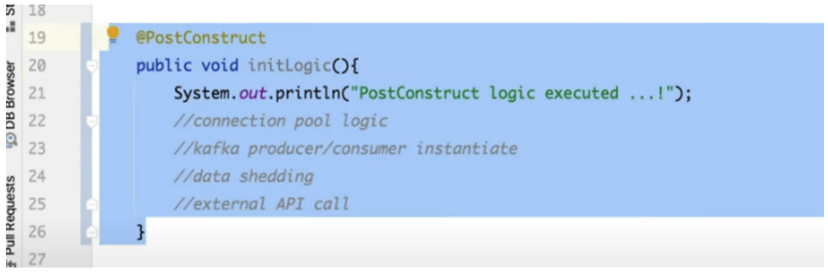


```
@PostConstruct
public void initLogic(){
    System.out.println("PostConstruct logic executed ...!");
```

Run: InterviewQaApplication

```
2023-09-23T15:21:45.712+05:30  INFO 29033 --- [
PostConstruct logic executed ...!
SecurityConfig loaded
```

Refer comments in the below screenshot



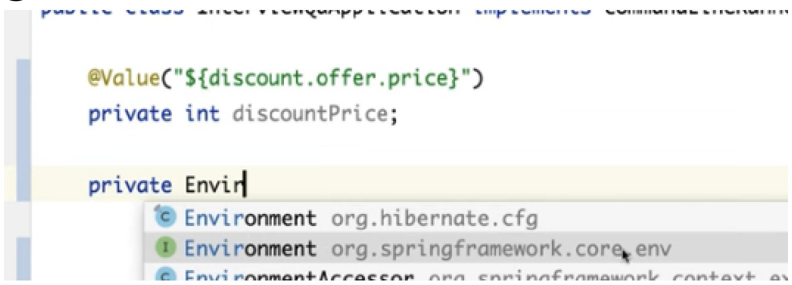
```
@PostConstruct
public void initLogic(){
    System.out.println("PostConstruct logic executed ...!");
    //connection pool logic
    //kafka producer/consumer instantiate
    //data shedding
    //external API call
}
```

17. How can we dynamically load values in a Spring Boot application?

👉 @Value

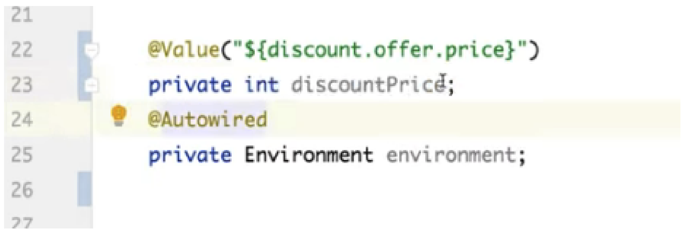
Two ways

@Value or environment

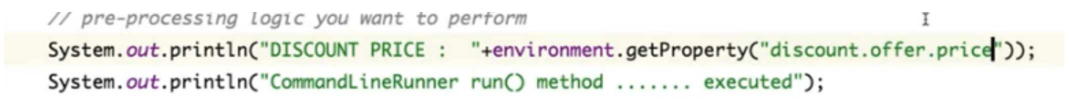


```
    @Value("${discount.offer.price}")
    private int discountPrice;

    private Envir
```

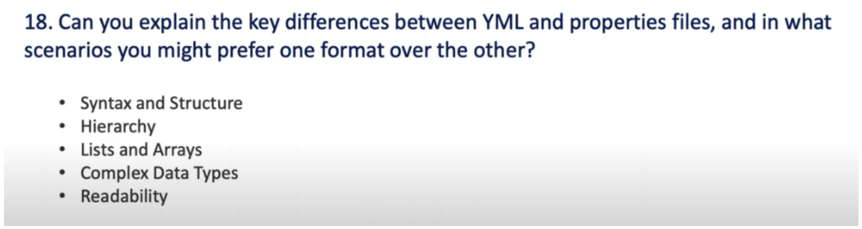
Environment org.hibernate.cfg
Environment org.springframework.core.env



```java
@Value("${discount.offer.price}")
private int discountPrice;
@Autowired
private Environment environment;
```

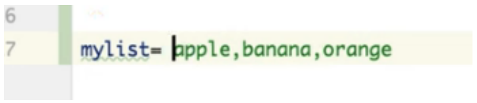


```java
// pre-processing logic you want to perform
System.out.println("DISCOUNT PRICE :  "+environment.getProperty("discount.offer.price"));
System.out.println("CommandLineRunner run() method ....... executed");
```

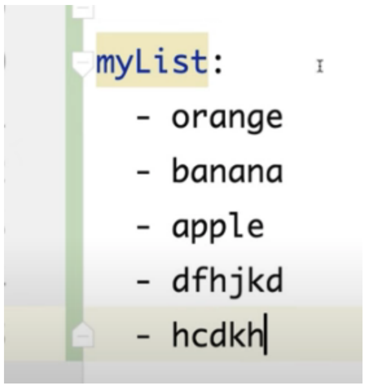


18. Can you explain the key differences between YML and properties files, and in what scenarios you might prefer one format over the other?

- Syntax and Structure
- Hierarchy
- Lists and Arrays
- Complex Data Types
- Readability



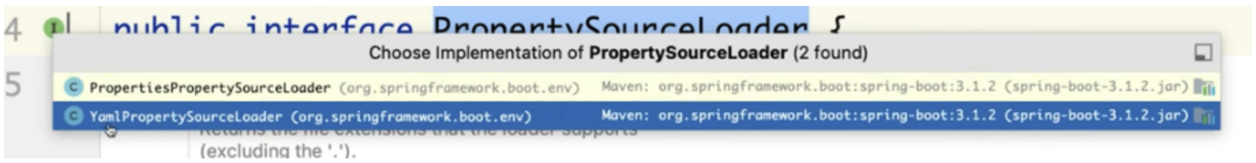
```
mylist= apple,banana,orange
```



```yaml
myList:
  - orange
  - banana
  - apple
  - dfhjkd
  - hcdkh
```

20. If I will configure same values in both properties then which value will be load in spring boot OR Who will load first properties or yml file ?

properties values will come default



```
public interface PropertySourceLoader {
```

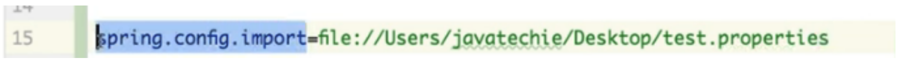
Choose Implementation of **PropertySourceLoader** (2 found)

```
PropertiesPropertySourceLoader (org.springframework.boot.env)   Maven: org.springframework.boot:spring-boot:3.1.2 (spring-boot-3.1.2.jar)
YamlPropertySourceLoader (org.springframework.boot.env)         Maven: org.springframework.boot:spring-boot:3.1.2 (spring-boot-3.1.2.jar)
```

(excluding the '.').

21. How to load External Properties in Spring Boot

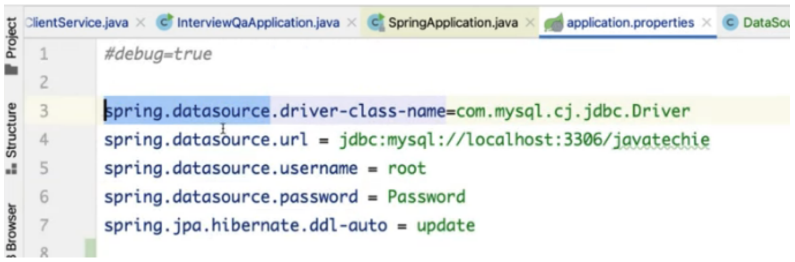
👉 spring.config.import



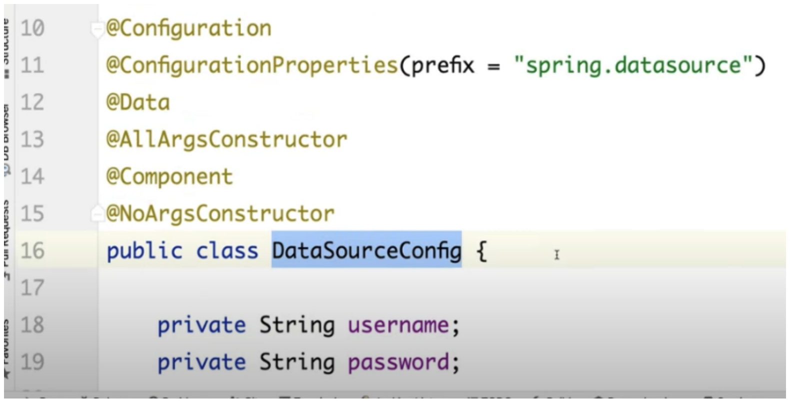
```
spring.config.import=file://Users/javatechie/Desktop/test.properties
```

It ignore application.properties and application.yml - it will consider only above properties

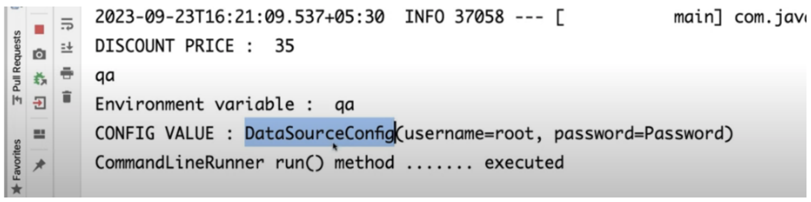
22. How to map or bind config properties to java Object ?



```
#debug=true

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/javatechie
spring.datasource.username = root
spring.datasource.password = Password
spring.jpa.hibernate.ddl-auto = update
```



```
@Configuration
@ConfigurationProperties(prefix = "spring.datasource")
@Data
@AllArgsConstructor
@Component
@NoArgsConstructor
public class DataSourceConfig {

    private String username;
    private String password;
```



```
2023-09-23T16:21:09.537+05:30  INFO 37058 --- [           main] com.jav
DISCOUNT PRICE :  35
qa
Environment variable :  qa
CONFIG VALUE : DataSourceConfig(username=root, password=Password)
CommandLineRunner run() method ....... executed
```