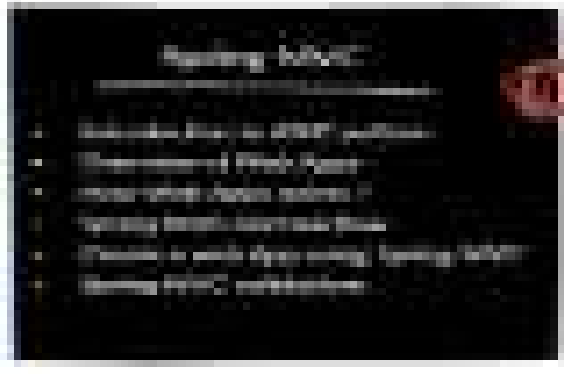
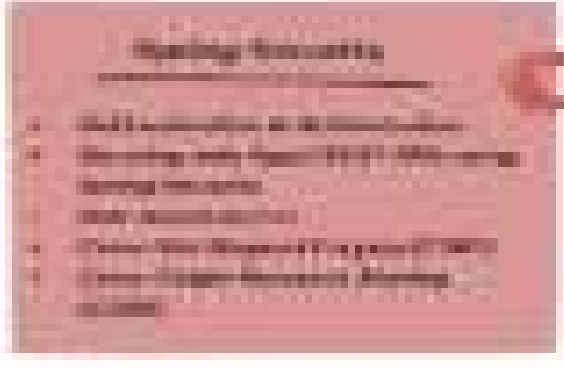

Spring, SpringBoot, JPA, Hibernate

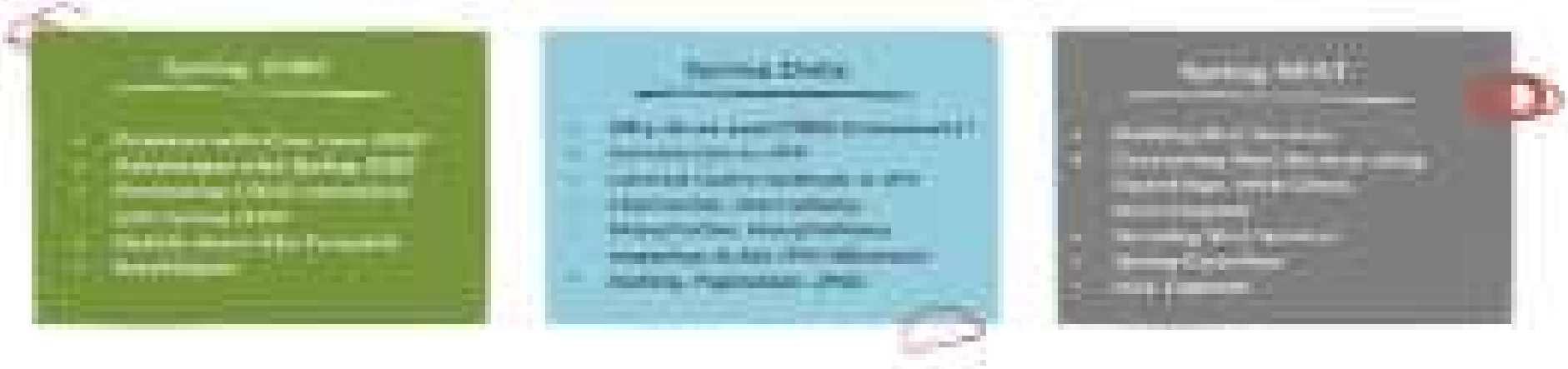
Agenda of the course



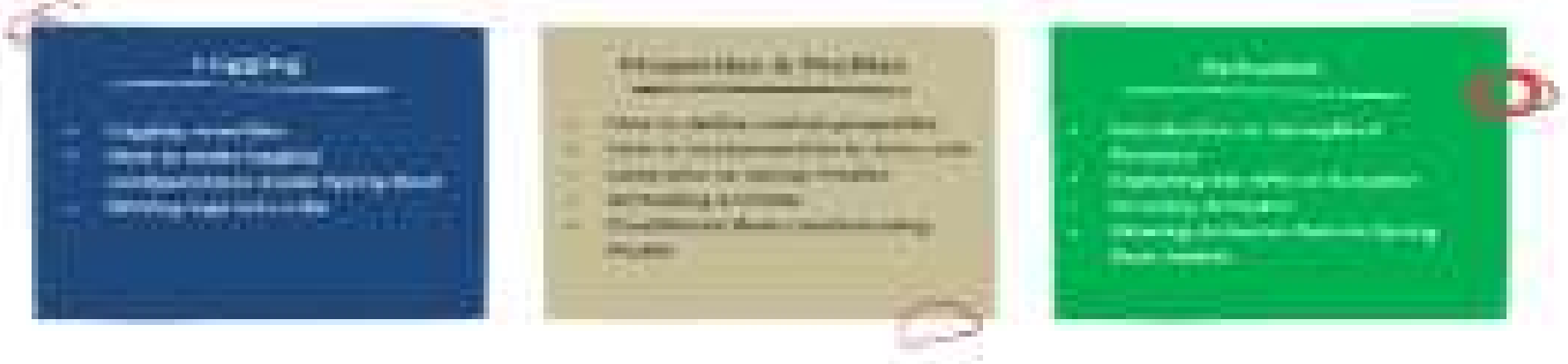
Agenda of the course

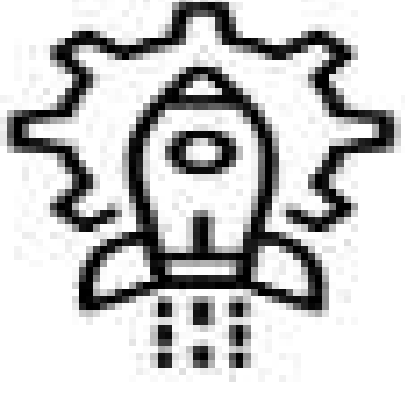


Agenda of the course



Agenda of the course

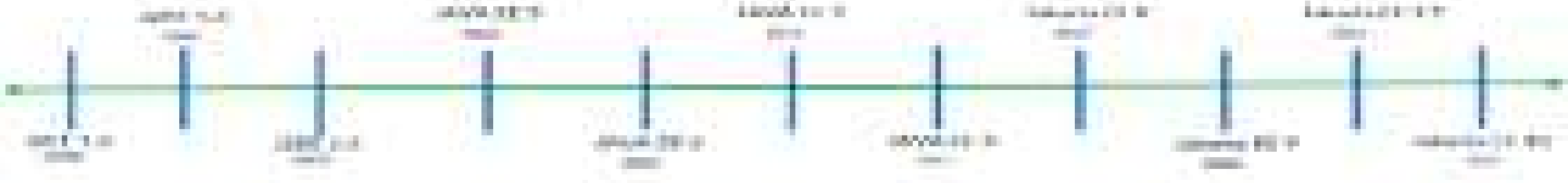




DEPLOYING SPRINGBOOT WEBAPP INTO CLOUD (AWS)

WHAT IS SPRING?

JAVA EE RELEASE TIMELINE

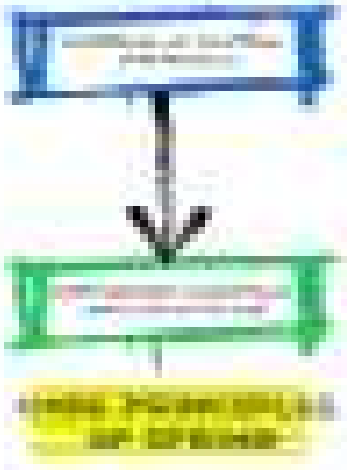


SPRING RELEASE TIMELINE

SPRING CORE

```
```

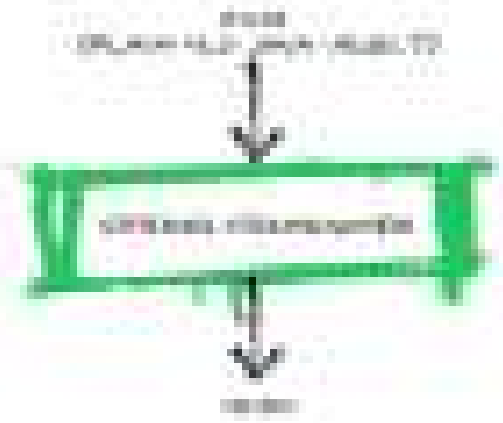
INVERSION OF CONTROL & DEPENDENCY INJECTION



ADVANTAGES OF IoC & DI

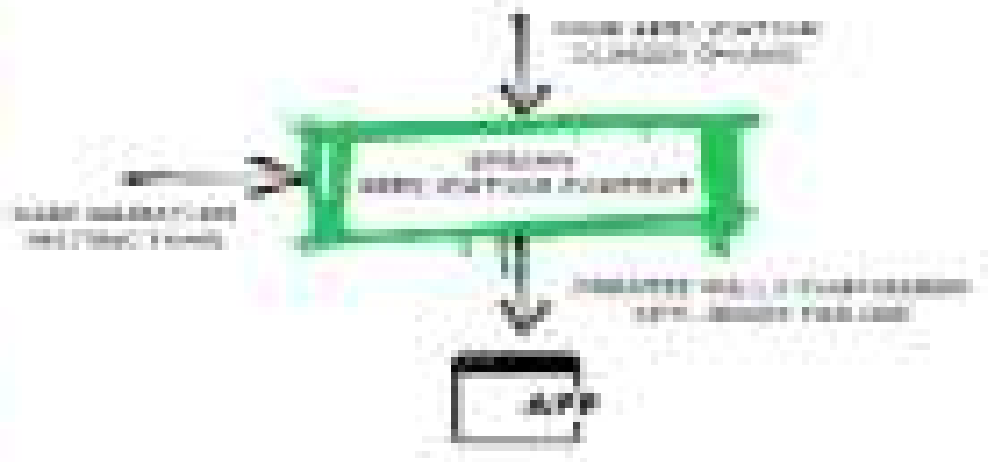
REAL LIFE LOOSE COUPLING EXAMPLE

SPRING BEANS, CONTEXT, SpEL



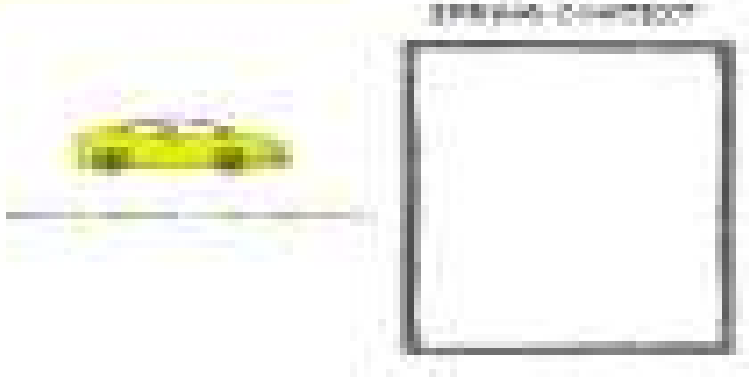
SPRING IoC CONTAINER

SPRING IoC CONTAINER

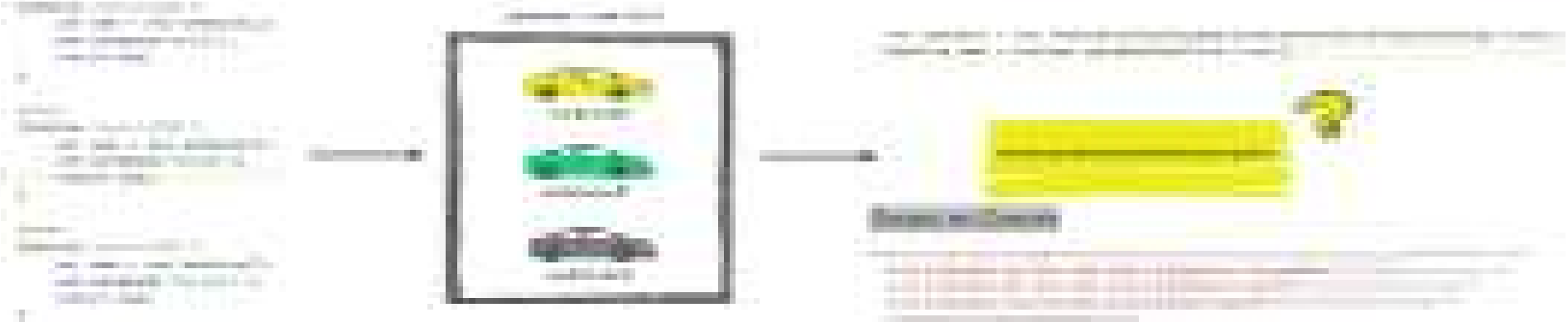



MAVEN

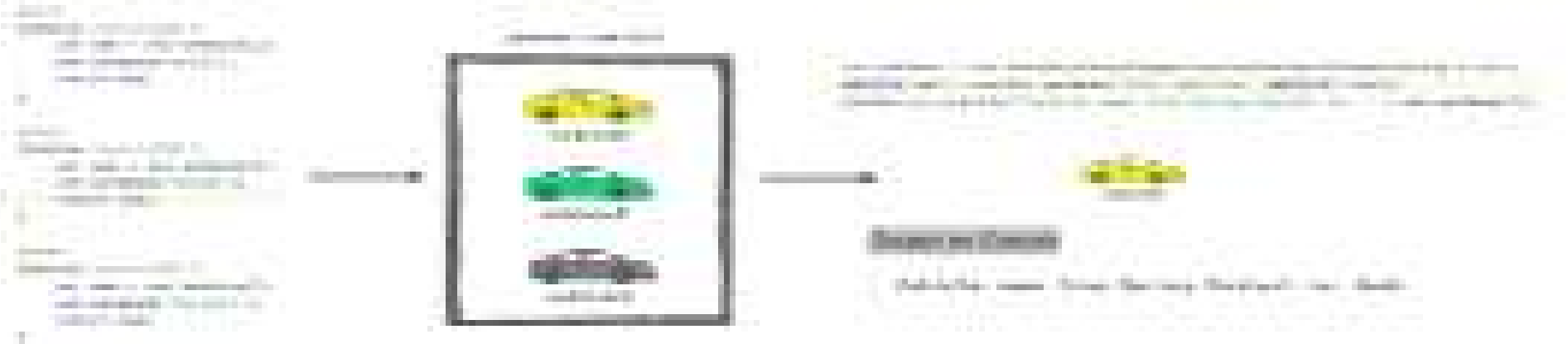
ADDING NEW BEANS TO SPRING CONTEXT



NoUniqueBeanDefinitionException



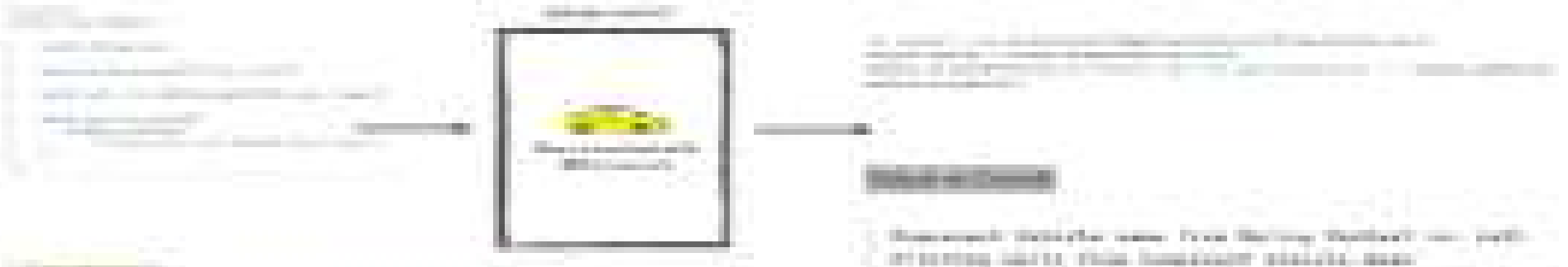
NoUniqueBeanDefinitionException



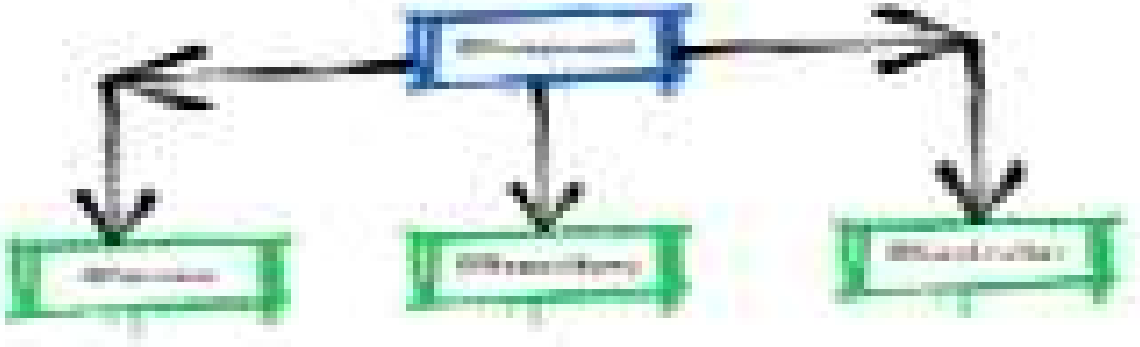
DIFFERENT WAYS TO NAME A BEAN

@Primary Annotation

@Component Annotation



Spring Stereotype Annotations



@Component is a class-level annotation. It is a generic stereotype for any Spring-managed component. Other stereotype annotations such as @Repository, @Service, and @Controller are specializations of @Component.

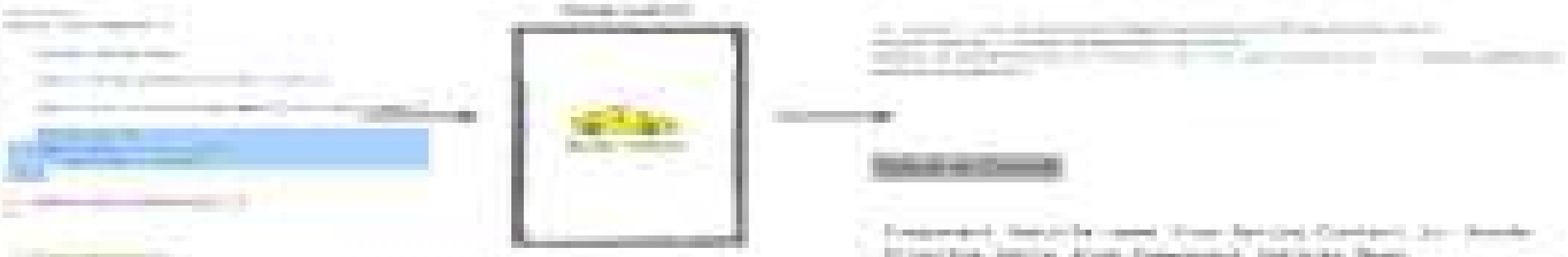
@Service is used to indicate that a class provides some business functionality. Spring context scanning will detect these classes automatically and register them as Spring beans.

@Repository is used to indicate that a class provides the mechanism for storage, retrieval, search, update and delete operation on objects.

@Controller is used to indicate that a class is a Spring Controller. This annotation is used to handle requests fired from the UI.

@Bean Vs @Component

@PostConstruct Annotation

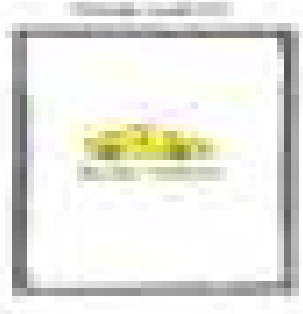


@PreDestroy Annotation

ADDING NEW BEANS PROGRAMMATICALLY

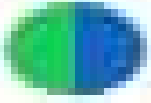
ADDING NEW BEANS PROGRAMMATICALLY

ADDING NEW BEANS USING XML CONFIGS



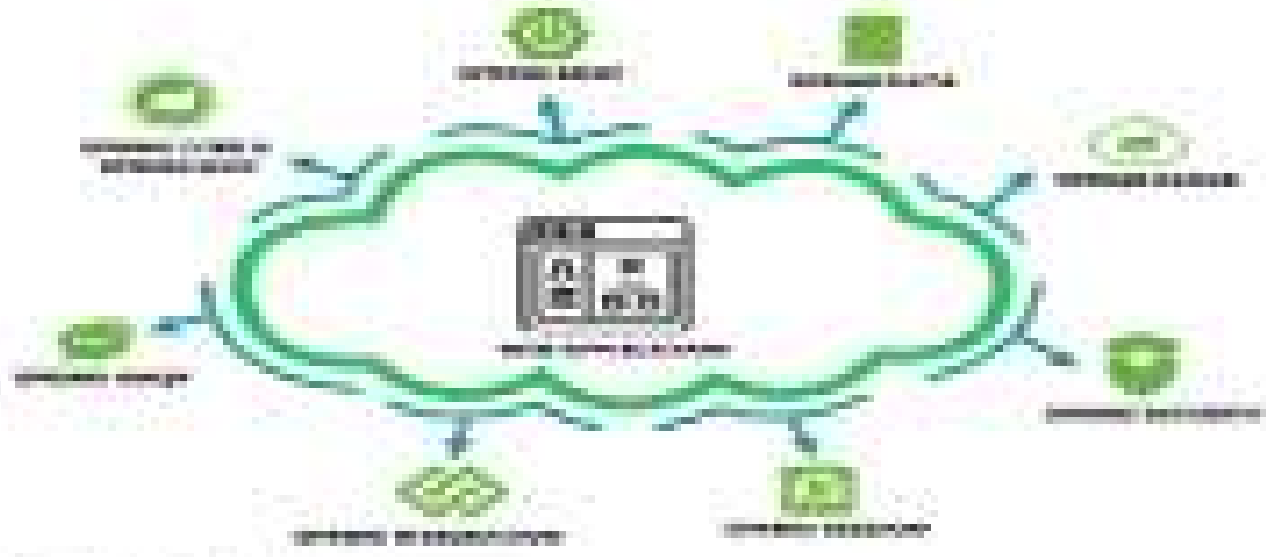
BEHIND THE SCENES OF A WEB APP

WHY SHOULD WE USE FRAMEWORKS?

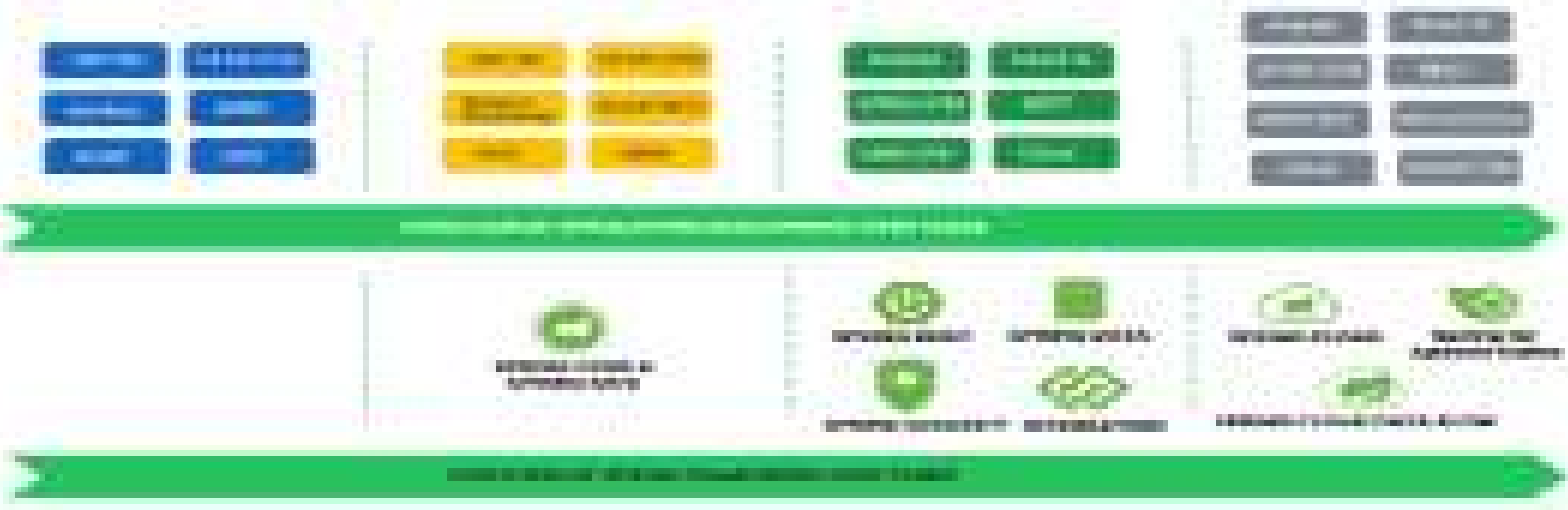


WHY SHOULD WE USE FRAMEWORKS?

SPRING PROJECTS

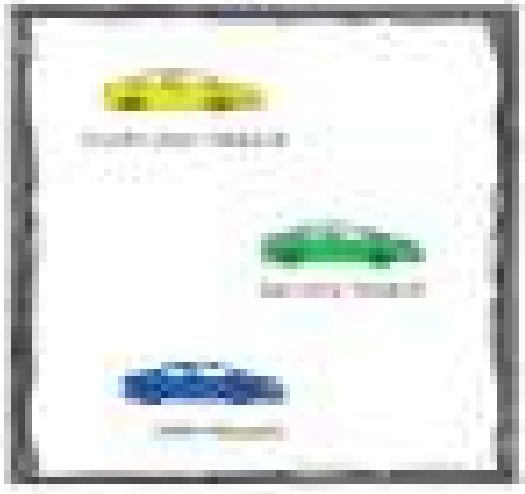


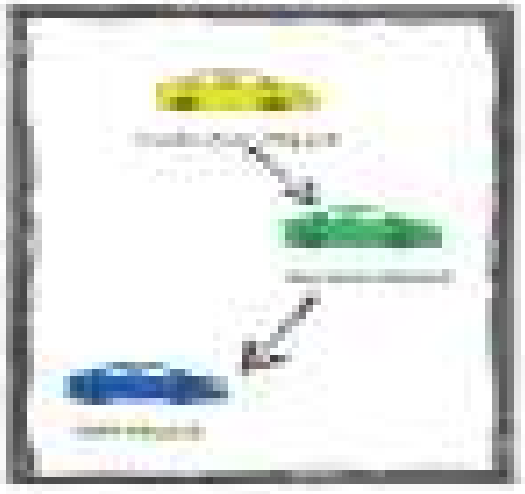
SPRING PROJECTS



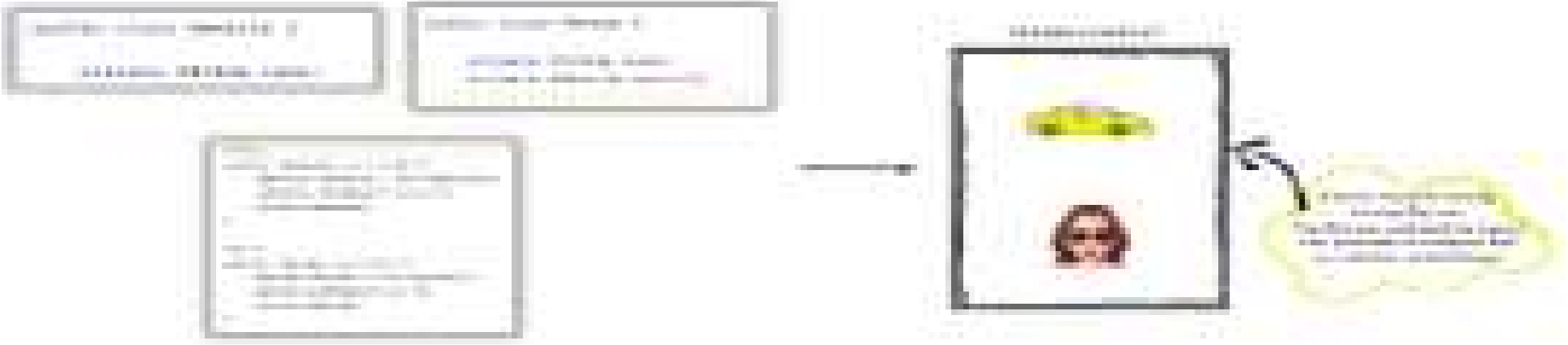
INTRODUCTION TO BEANS WIRING INSIDE SPRING

INTRODUCTION TO BEANS WIRING INSIDE SPRING

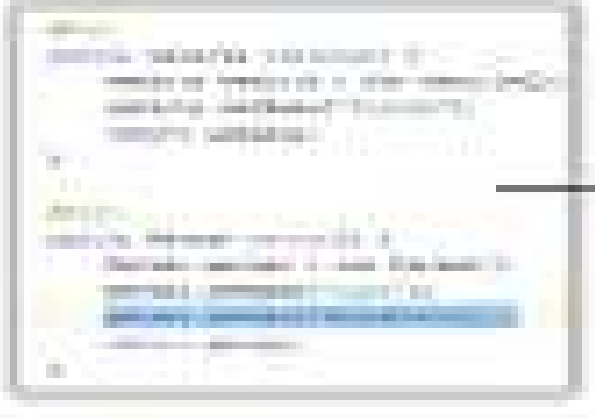


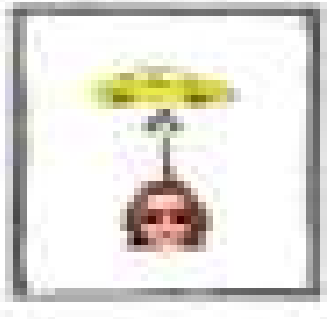


NO WIRING SCENARIO INSIDE SPRING

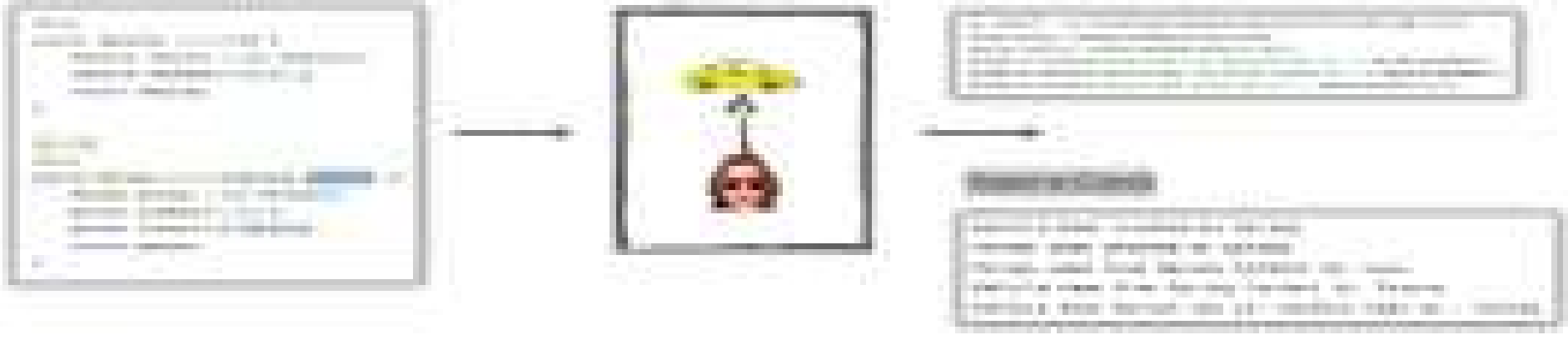


WIRING BEANS USING METHOD CALL





WIRING BEANS USING METHOD PARAMETERS

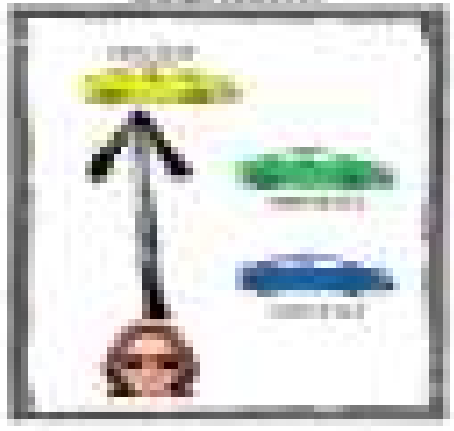


Inject Beans using @Autowired on class fields

Inject Beans using @Autowired on setter method

Inject Beans using @Autowired with constructor

How Autowiring works with multiple beans of same type



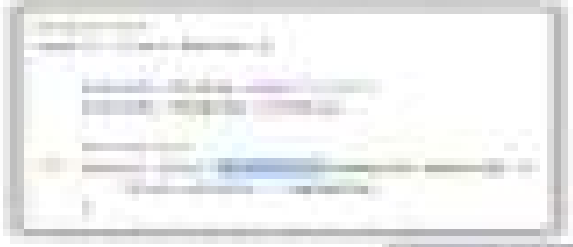
How Autowiring works with multiple beans of same type

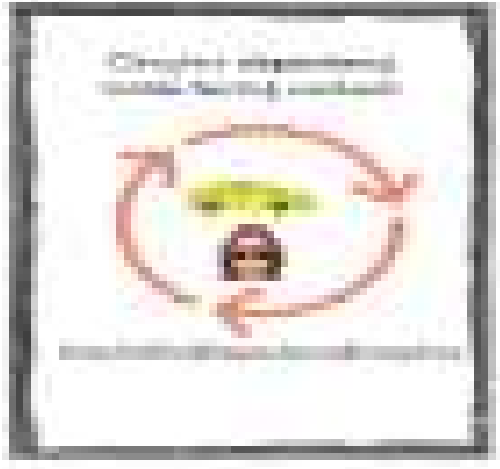
STEP 3

How Autowiring works with multiple beans of same type

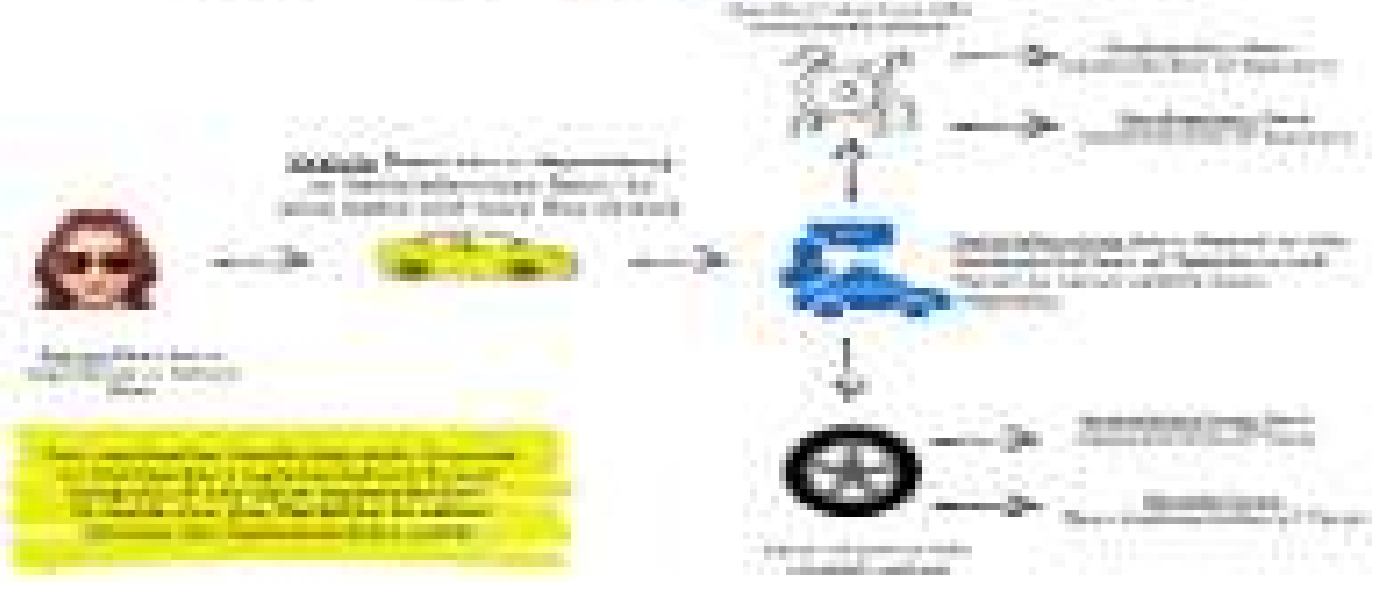
STEP 3

Understanding & Avoiding Circular Dependencies





ASSIGNMENT RELATED TO BEARS, AUTOMOBILES, ET

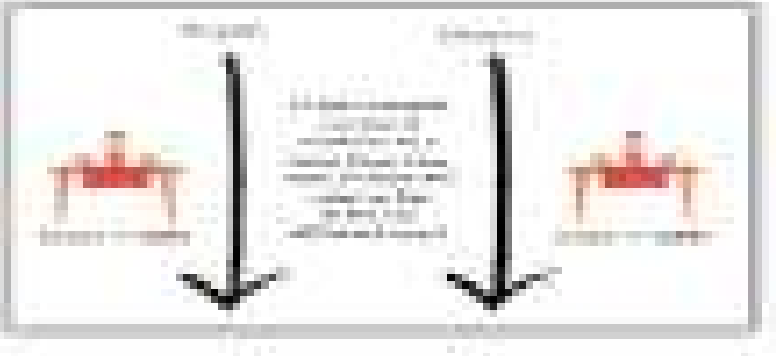


Bean Scopes inside Spring

1. Singleton
2. Prototype
3. RequestScope
4. SessionScope
5. ApplicationScope

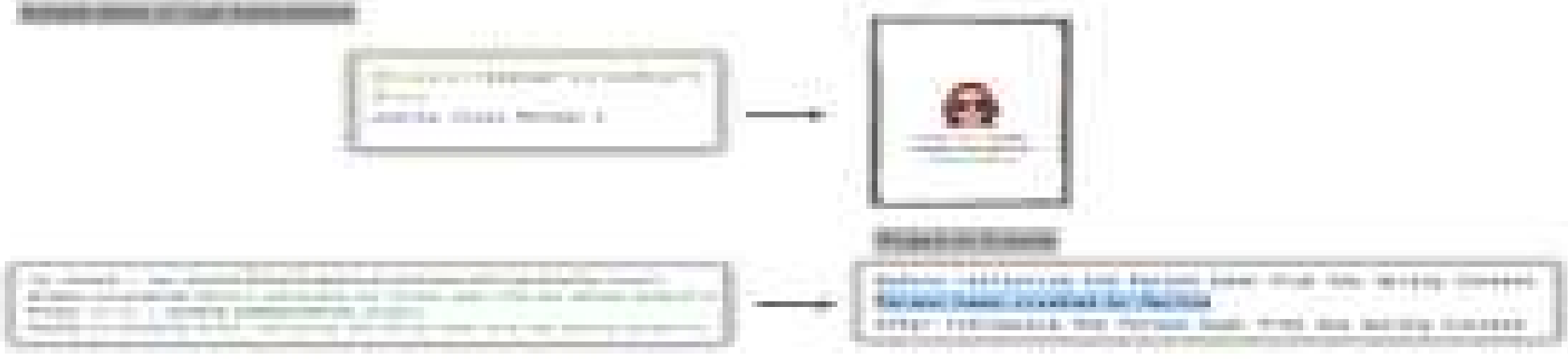
SINGLETON BEAN SCOPE

RACE CONDITION

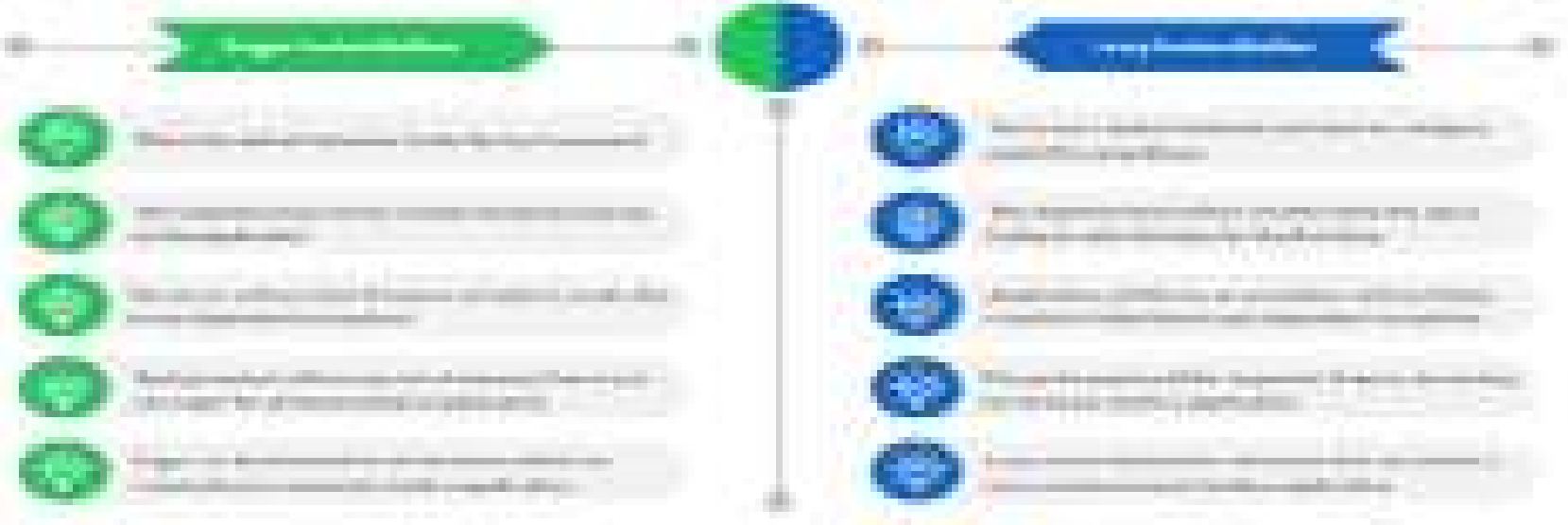


USE CASES OF SINGLETON BEANS

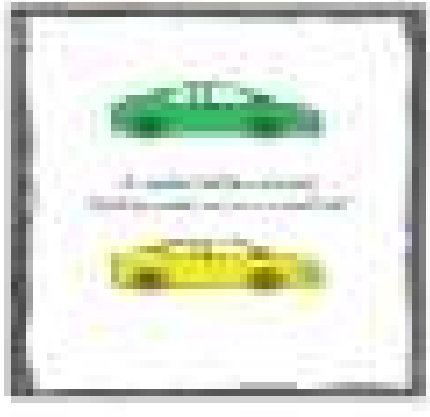
EAGER & LAZY INSTANTIATION



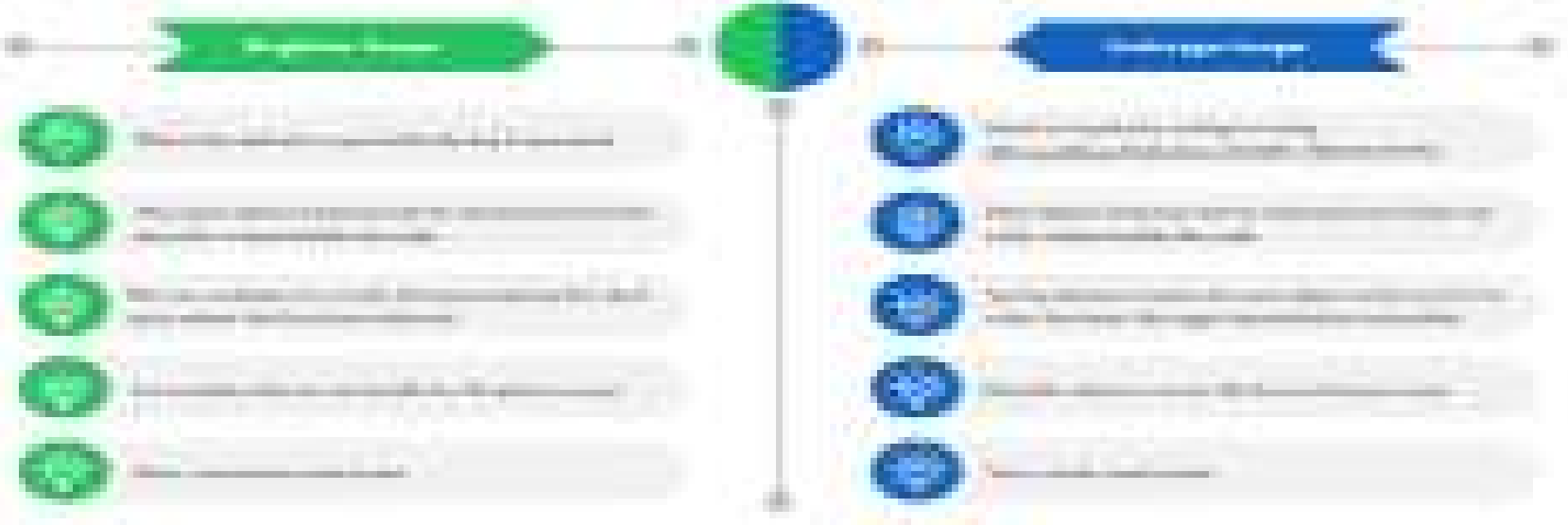
Eager Vs Lazy



PROTOTYPE SCAN SCOPE

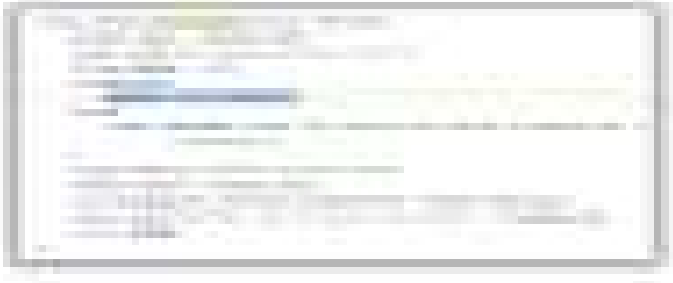


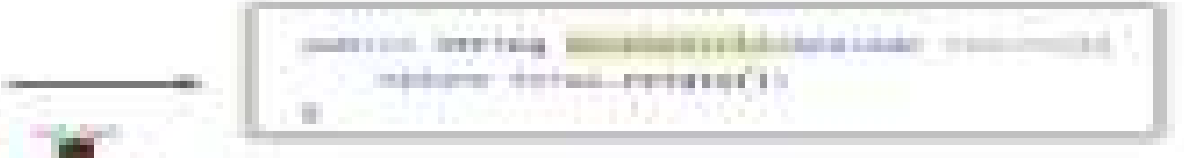
Singleton Vs Prototype



ASPECT-ORIENTED PROGRAMMING (AOP)

ASPECT-ORIENTED PROGRAMMING (AOP)

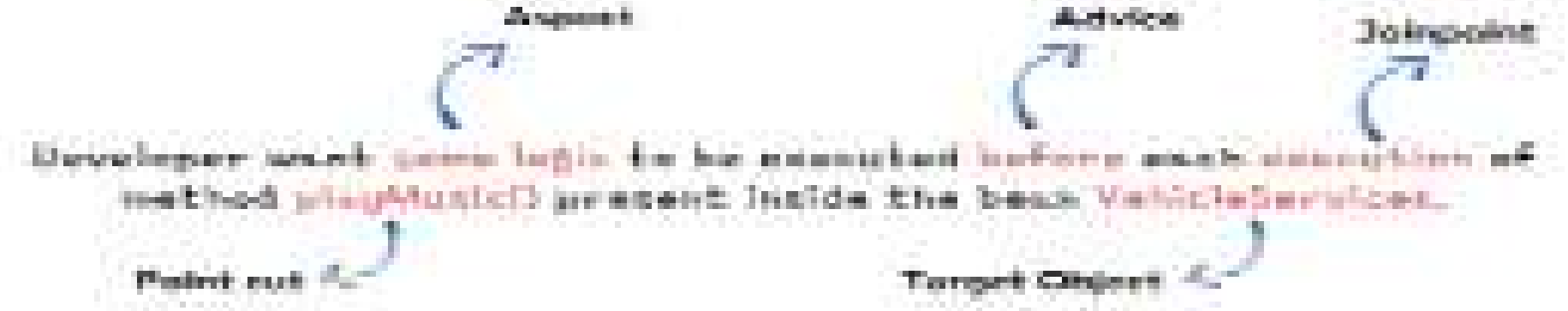




ASPECT-ORIENTED PROGRAMMING (AOP)

ASPECT-ORIENTED PROGRAMMING (AOP)

Typical Scenario of AOP implementation



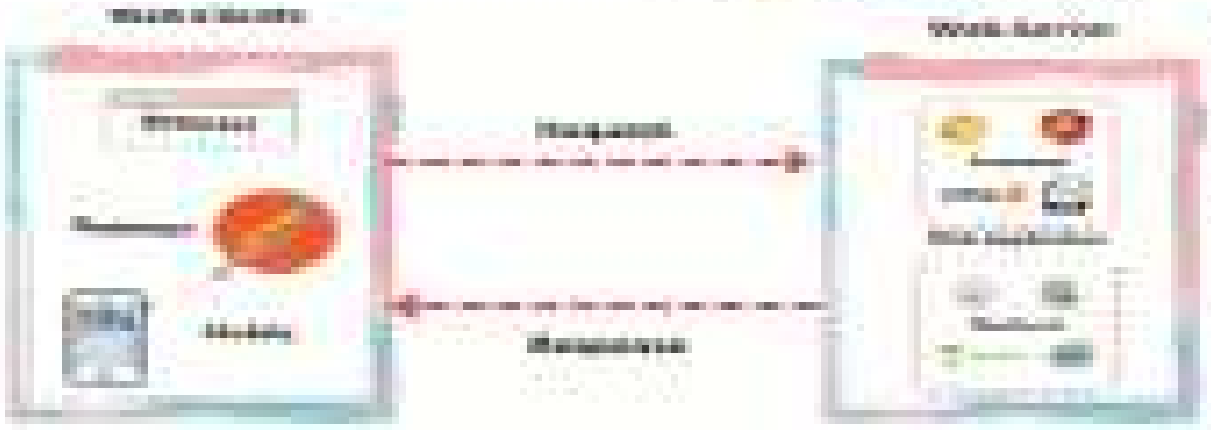
[illegible]

ADVICE TYPES INSIDE AOP

CONFIGURING SERVICES WHEN AOP

CONFIGURING ADVICES INSIDE AOP

OVERVIEW OF A WEB APP



1.

2.

3.

4.

ROLE OF SERVLETS INSIDE WEB APPs

SPRINGBOOT : THE HERO OF SPRING FRAMEWORK

1.
2.
3.
4.
5.

SpringBoot Important features

SpringBoot Starters

Autoconfiguration

Actuator & DevTools

Quick Recap

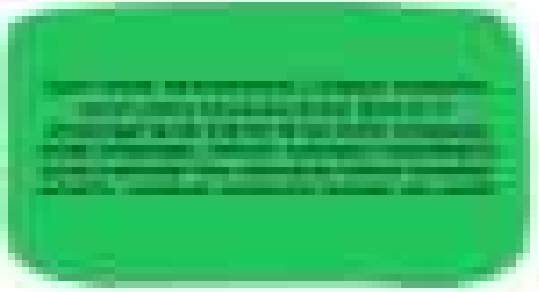
Quick Recap

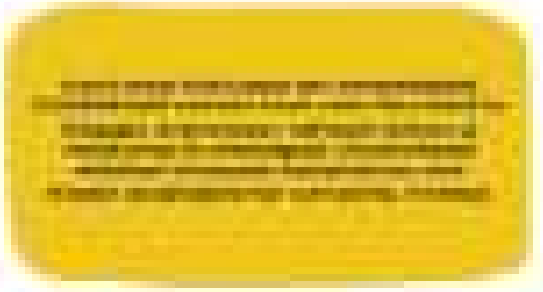
Do you know we can configure metadata using a single command using Spring MVC annotation ?

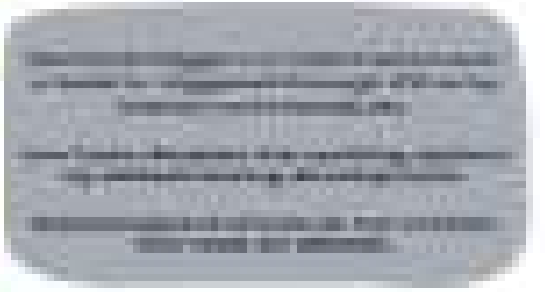
```
@SpringBootApplication
public class SwaggerConfiguration {

    @Bean
    public Docket productApi() {
        return new Docket(DocumentationType.SWAGGER_2).select()
          .apis(RequestHandlerSelectors.basePackage("net.javaguides.springboot.swagger.api"))
          .paths(PathSelectors.regex("/product.*"))
          .build();
    }

}
```

INTRODUCTION TO THYMELEAF

SPRINGBOOT DEVTOOLS

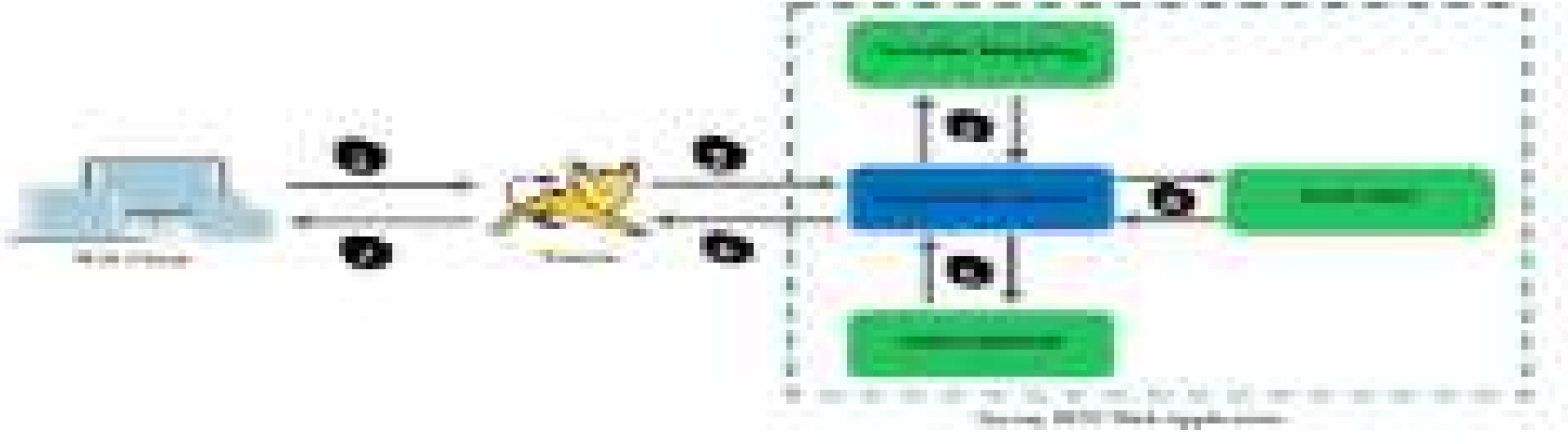







INTRODUCTION TO MVC PATTERN
Controller
View
Model

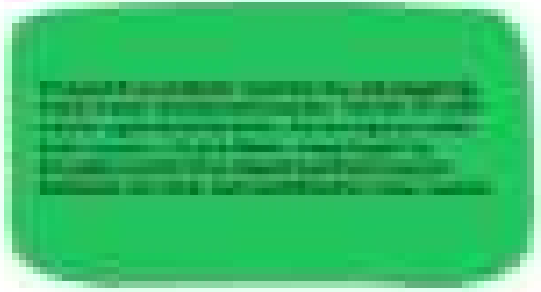
SPRING MVC ARCHITECTURE & INTERNAL FLOW



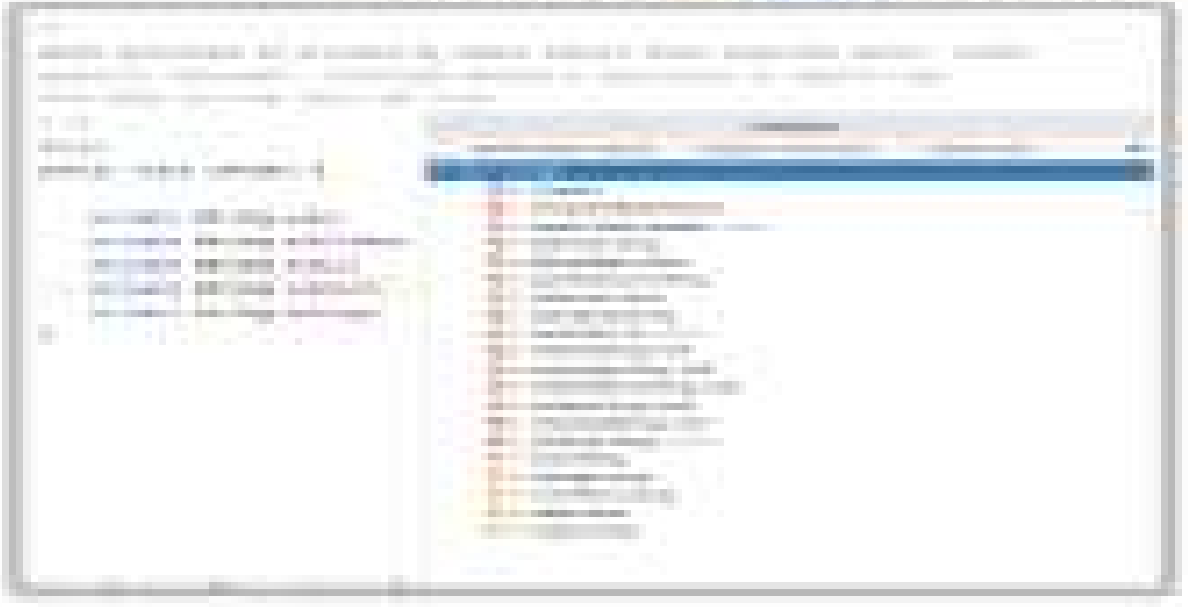
SPRING MVC ARCHITECTURE & INTERNAL FLOW

QUICK TIP

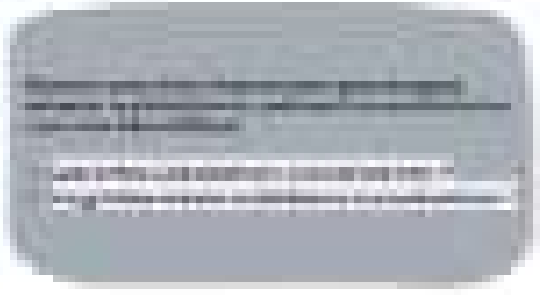
REDUCING BOILERPLATE CODE WITH LOMBOK



REDUCING BOILERPLATE CODE WITH LOMBOK



VALIDATION WITH SPRING BOOT



Important Validation Annotations

- [illegible]

```
[illegible]
```

VALIDATION WITH SPRING BOOT

Bean Scopes inside Spring

1. Singleton
2. Prototype
3. Request
4. Session
5. Application

1.
2.
3.

Key points of Spring Web scopes

Request Scope

- [illegible]

- [illegible]

- [illegible]

Session Scope

- [illegible]

- [illegible]

- [illegible]

Application Scope

- [illegible]

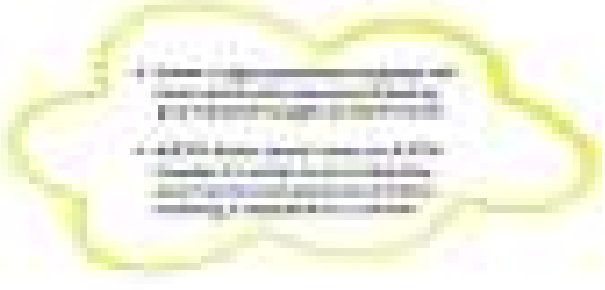
- [illegible]

- [illegible]

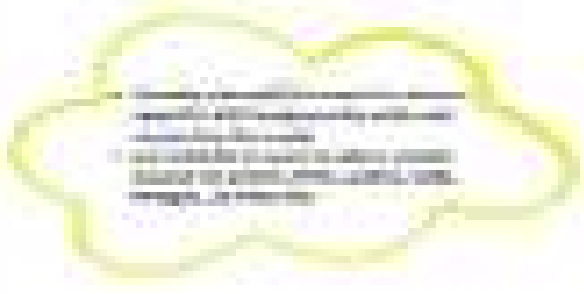
QUICK TIP

DEFAULT SECURITY CONFIGURATIONS IN SPRING SECURITY

CONFIGURE permitAll() WITH SPRING SECURITY



CONFIGURE denyAll() WITH SPRING SECURITY



CONFIGURE CUSTOM SECURITY CONFIGs & CIDR DISABLE

IN-MEMORY AUTHENTICATION IN SPRING SECURITY

CONFIGURING LOGIN & LOGOUT PAGE

QUICK TIP

@ControllerAdvice & @ExceptionHandler

QUICK TIP

CROSS-SITE REQUEST FORGERY (CSRF)

CROSS-SITE REQUEST FORGERY (CSRF)

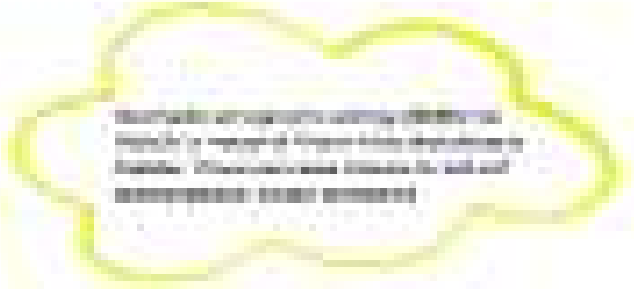
```
[illegible]
```

SOLUTIONS TO CSRF

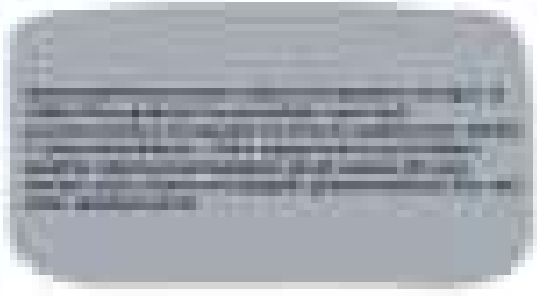
Quick Tip

SPRING BOOT & H2 DATABASE

Key points of JDBC

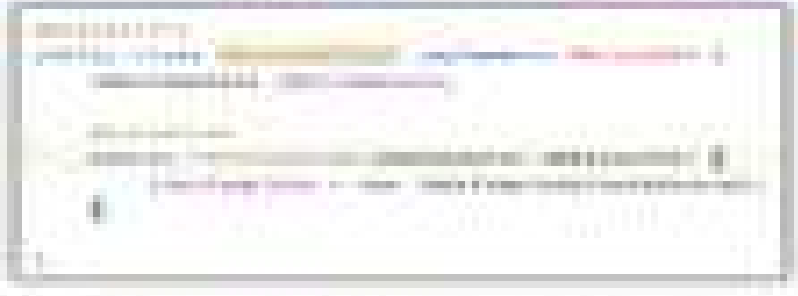


INTRO TO SPRING JDBC



Spring JDBC : advantages

Using JdbcTemplate



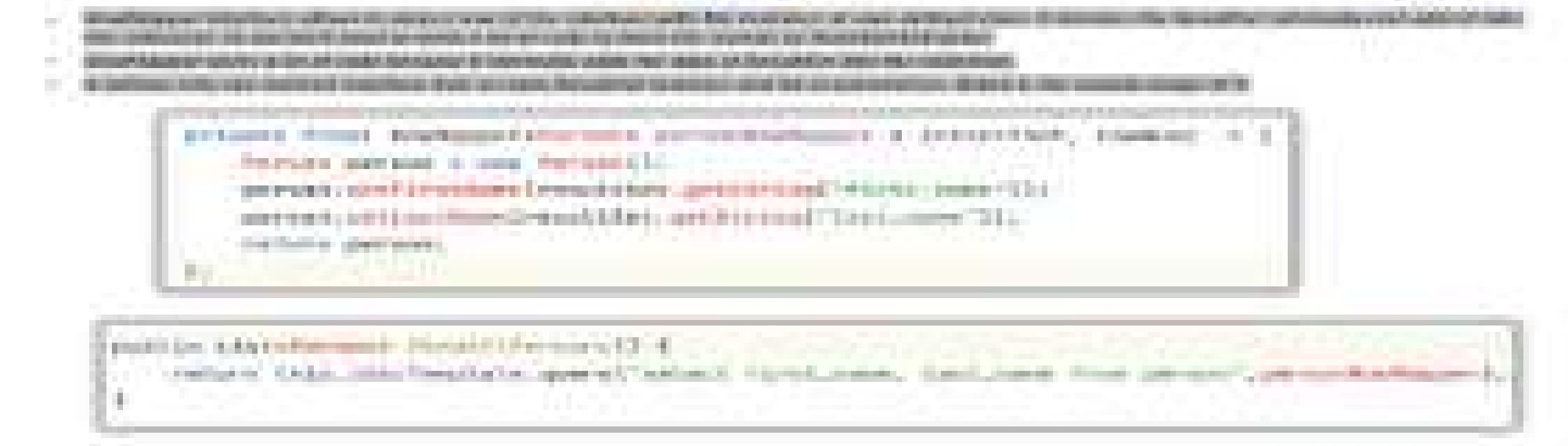
-

```
// Use try-with-resources to ensure the connection is closed automatically
try (connection c = this.dataSource.getConnection();
```

```
// Use parameterized queries to prevent SQL injection
public List<User> findUsersByAge(int minAge) throws SQLException {
```

```
// Use PreparedStatement for batch operations
String sql = "INSERT INTO products (name, price) VALUES (?, ?)";
try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

Using JobScheduler



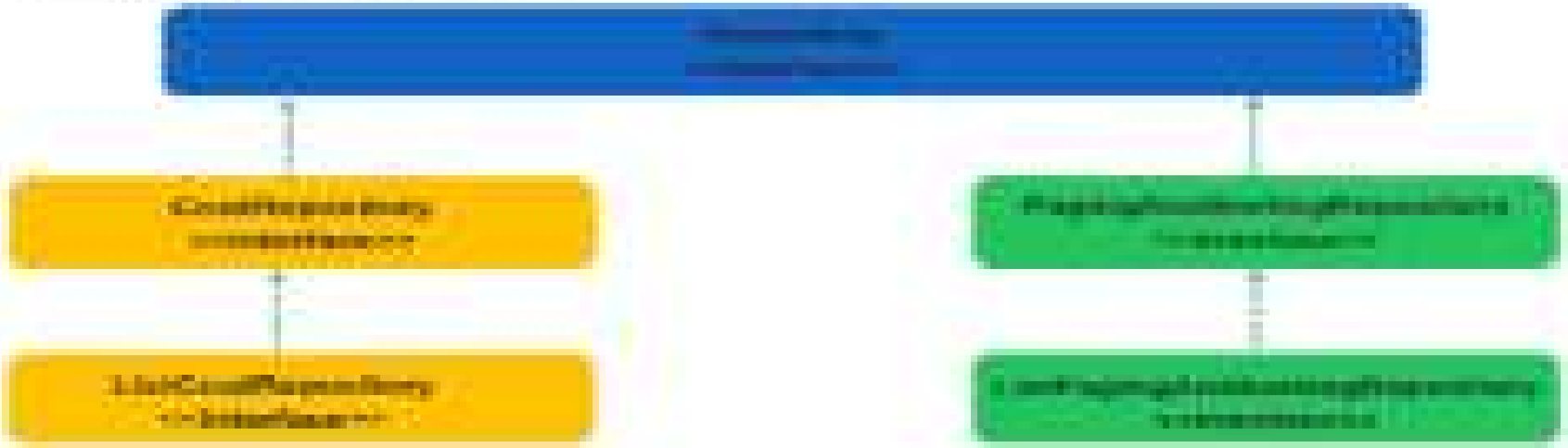
QUICK TIP

```
```

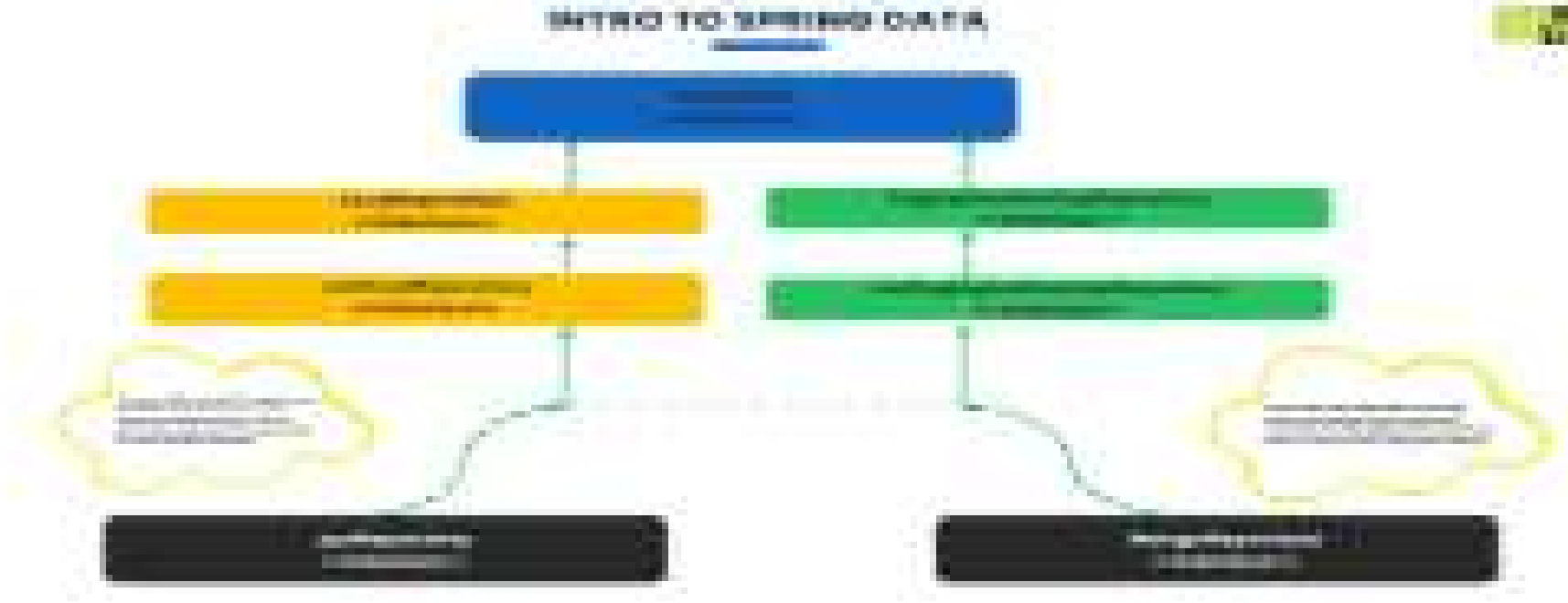
INTRO TO SPRING DATA

INTRO TO SEISMO DATA

INTRO TO SEISMO DATA

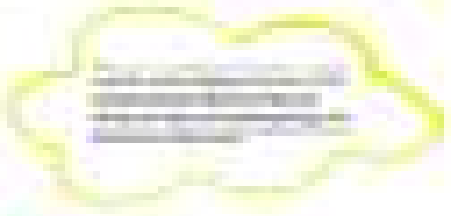


QUICK TIP


INTRO TO SEISMO DATA

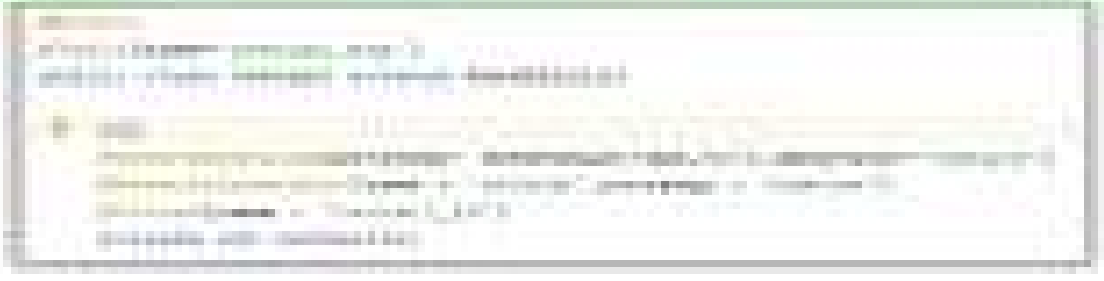
INTRO TO SPRING DATA JPA

- [illegible]
- [illegible]



- [illegible]

[illegible]



Derived Query Methods in Spring Data JPA

- [illegible]
- [illegible]
- [illegible]

```
// [illegible]
List<Employee> [illegible]findBy[illegible]LastNameIs
```

```
// [illegible]
Optional<[illegible]> [illegible]findBy[illegible](String email);
```

```
// [illegible]
List<[illegible]> [illegible]findBy[illegible]AndLastName(String email, String lastName);
```

QUICK TIP

Derived Query Methods in Spring Data JPA

Derived Query Methods in Spring Data JPA

Supported keywords inside method names

Keyword	Sample	Logical result
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

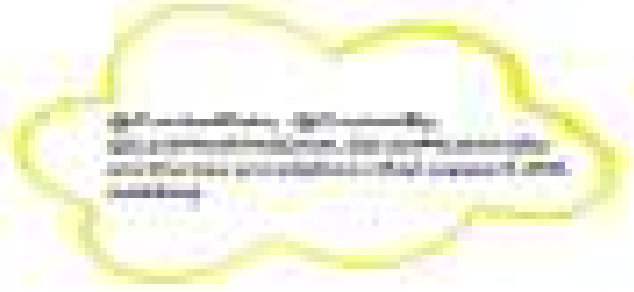
Derived Query Methods in Spring Data JPA

Unsupported keywords (cause method names

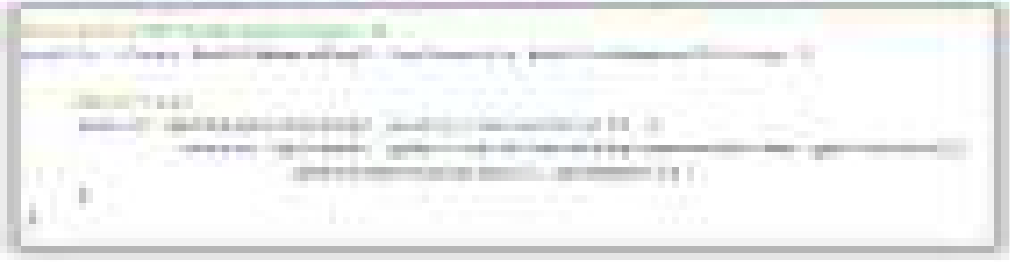
Keyword	[illegible]	Example query that throws by JPA
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

[illegible]

Auditing Support By Spring Data JPA

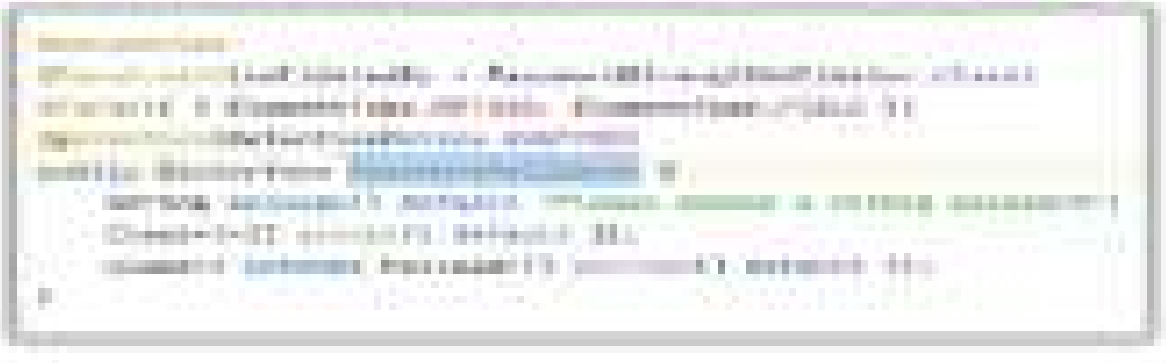


Auditing support By Spring Data JPA

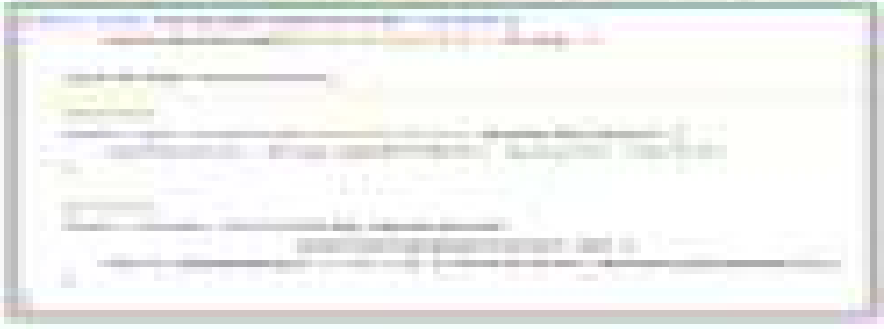


QUICK TIP

Spring MVC Custom Validations

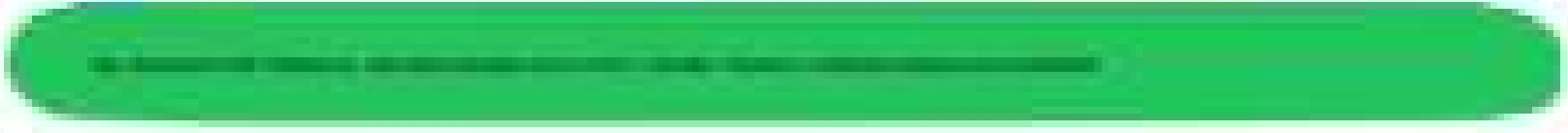


Spring MVC Custom Validations



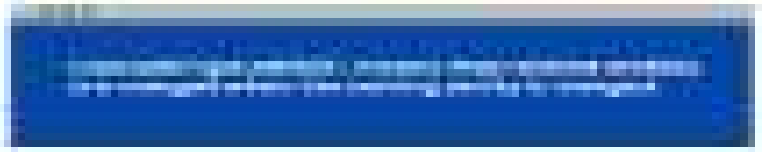
One to One Relationship Inside JPA

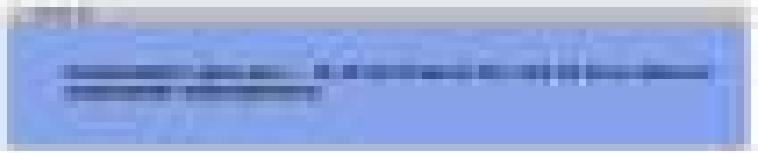
One to One Relationship Inside JPA



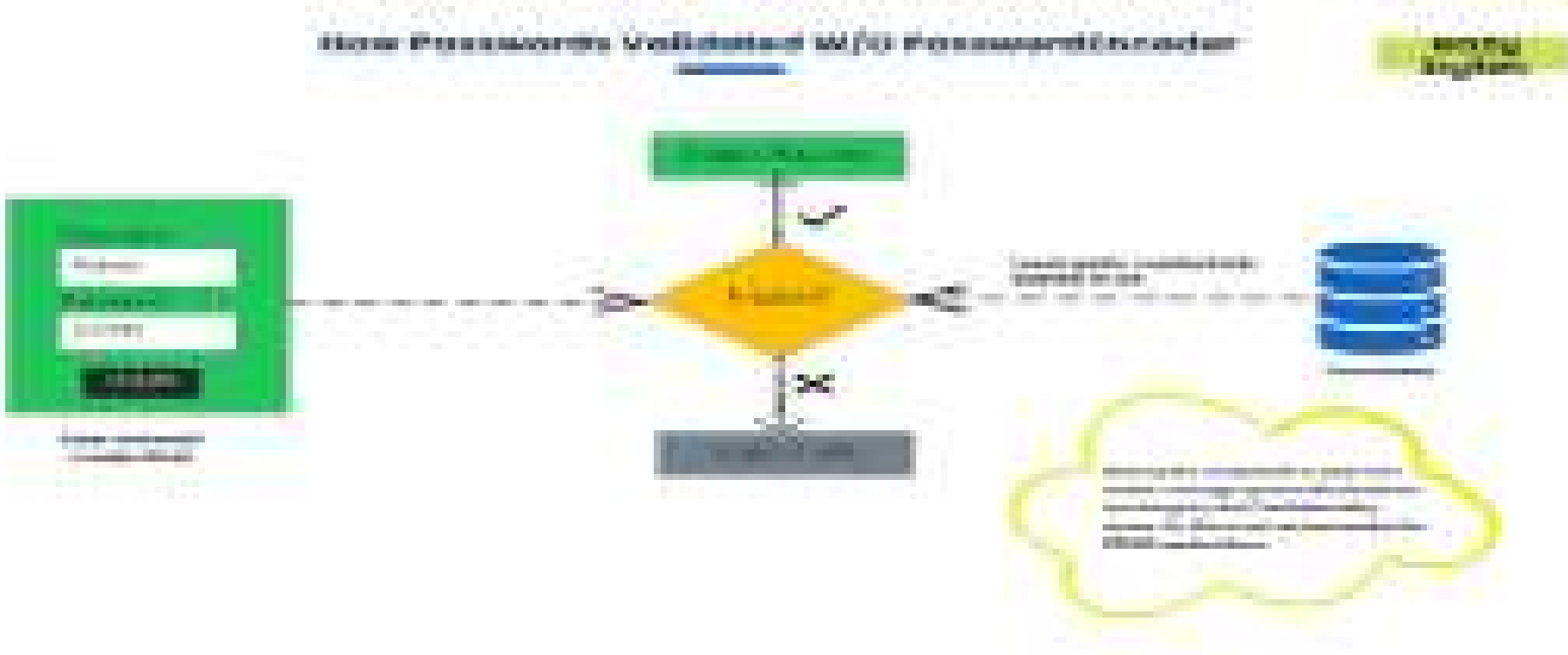
Key points of Cascade Types

CASCADE TYPES





Spring Security AuthenticationProvider

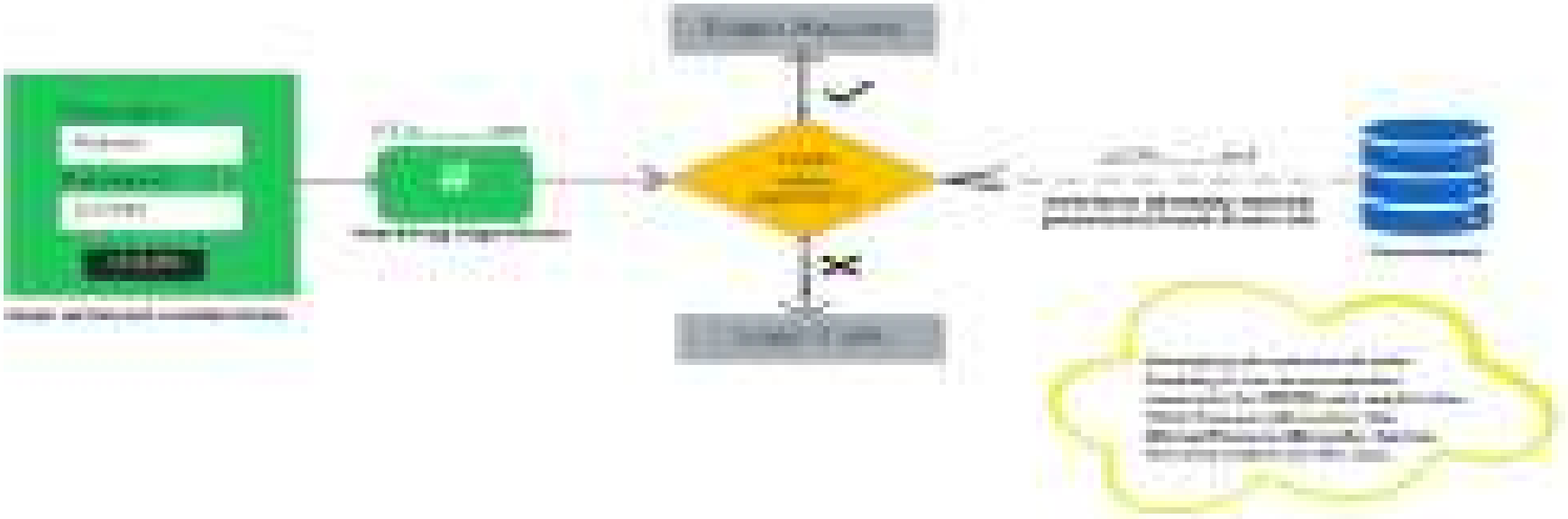

How Passwords Validated W/O PasswordEncoder

PASSWORDS MANAGEMENT

Different ways of Pwd management

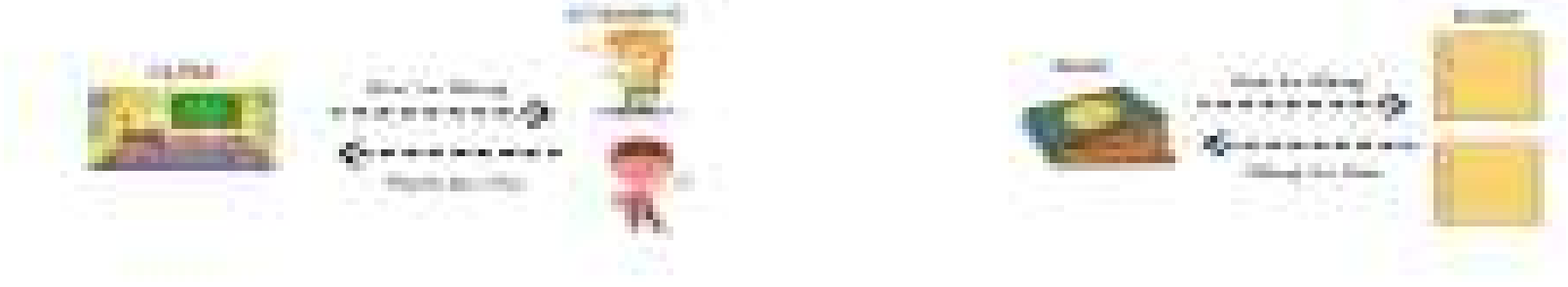
QUICK TIP

How Passwords Validated With Hashing & Passwordhunder

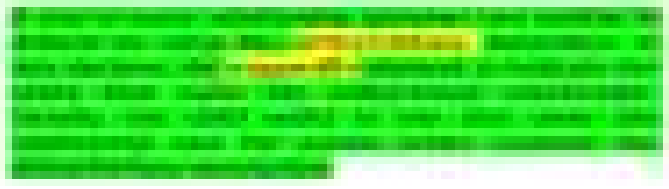


QUICK TIP

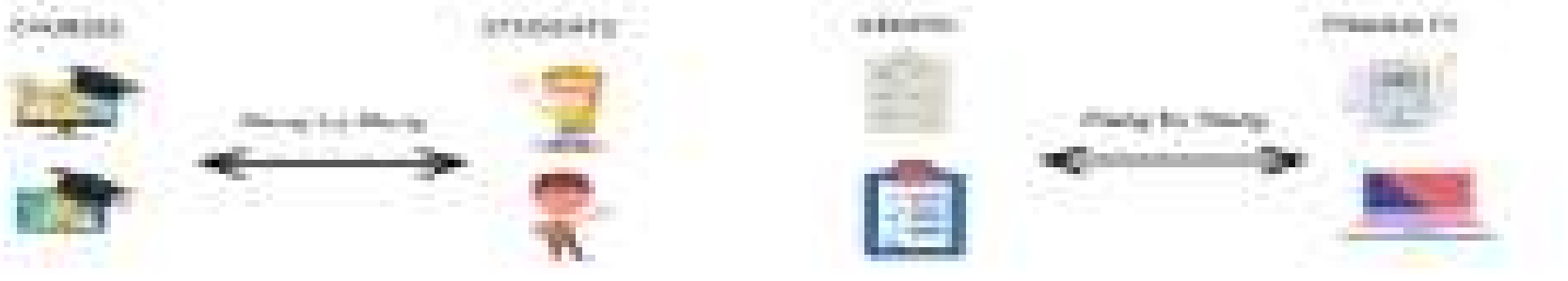
One to Many & Many to One Relationship inside JPA



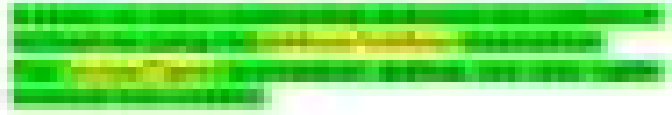
One to Many & Many to One Relationship Inside JPA



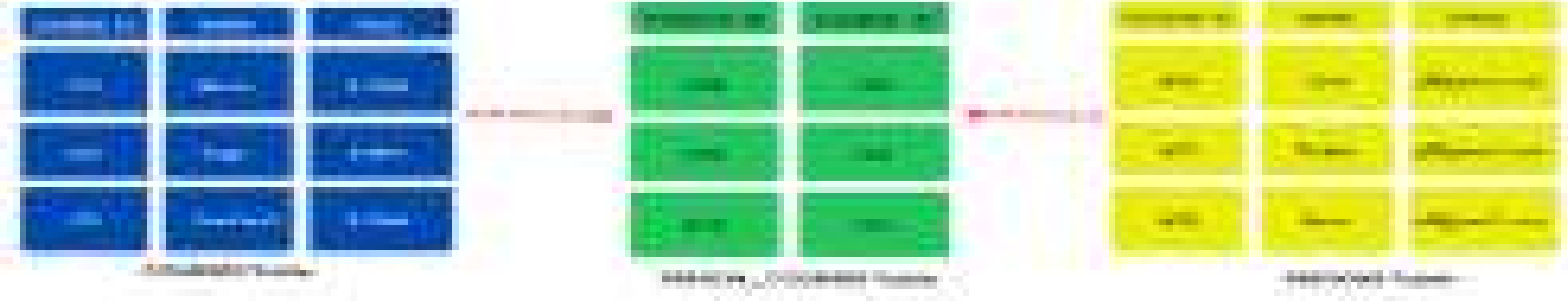
Money to Money Invisible JPEG



Money for Money Invisible JPA



Moving to Moving Invisible JPA



Sorting

Pagination

Custom Queries with JPA

JPQL

Introduction

- The Java Persistence Query Language (JPQL) is a platform-independent object-oriented query language defined as part of the Java Persistence API (JPA) specification.

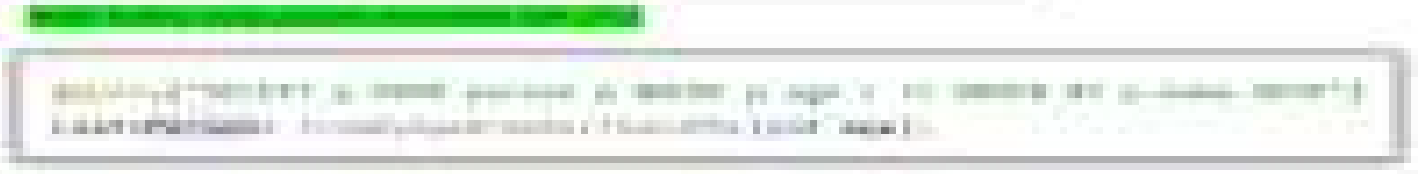
- JPQL is used to make queries against entities stored in a relational database. It is heavily inspired by SQL, but it operates directly on JPA entity objects. JPQL queries are created and executed using the JPA `EntityManager` API.

- This guide illustrates all (using JPQL) to show by examples a subset of the JPQL standard. You'll learn how to get started with the JPQL language.

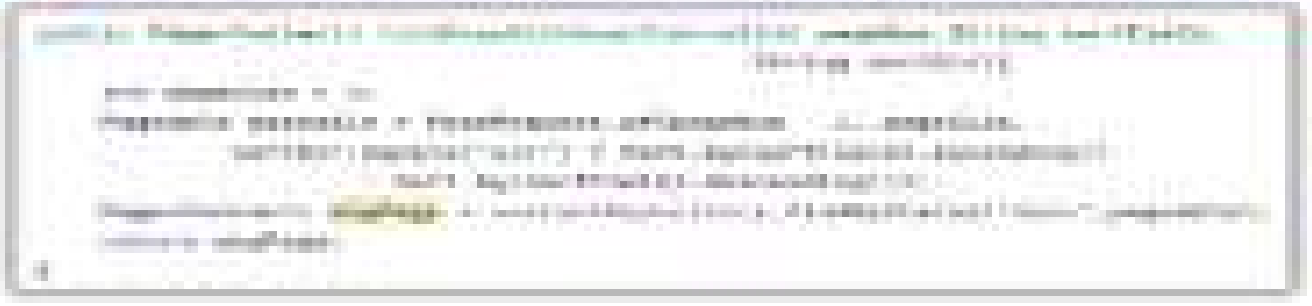
JPQL Examples

- Below is an example of JPQL. The `countBooks` method uses the `createQuery` method of the `EntityManager` interface to create a query object. The query is defined as a string that selects the count of `Book` entities.

```
public Long countBooks() {
    return em.createQuery("SELECT COUNT(b) FROM Book b", Long.class)
             .getSingleResult();
}
```



Pagination Examples



[illegible]

```
// [illegible]
request('SELECT a FROM [illegible] a WHERE a.[illegible] = ? LIMIT 1', [id], (err, [illegible]) => {
  [illegible]; // [illegible] [illegible], [illegible]
});
```

[illegible]

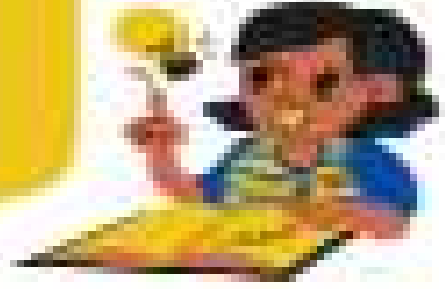
```
// [illegible]
[illegible] = [illegible] FROM [illegible] WHERE a.[illegible] = ? ORDER BY a.[illegible] DESC', [id], (err, [illegible]) => {
  [illegible]; // [illegible] [illegible], [illegible]
});
```

```
[illegible]
```

[illegible]

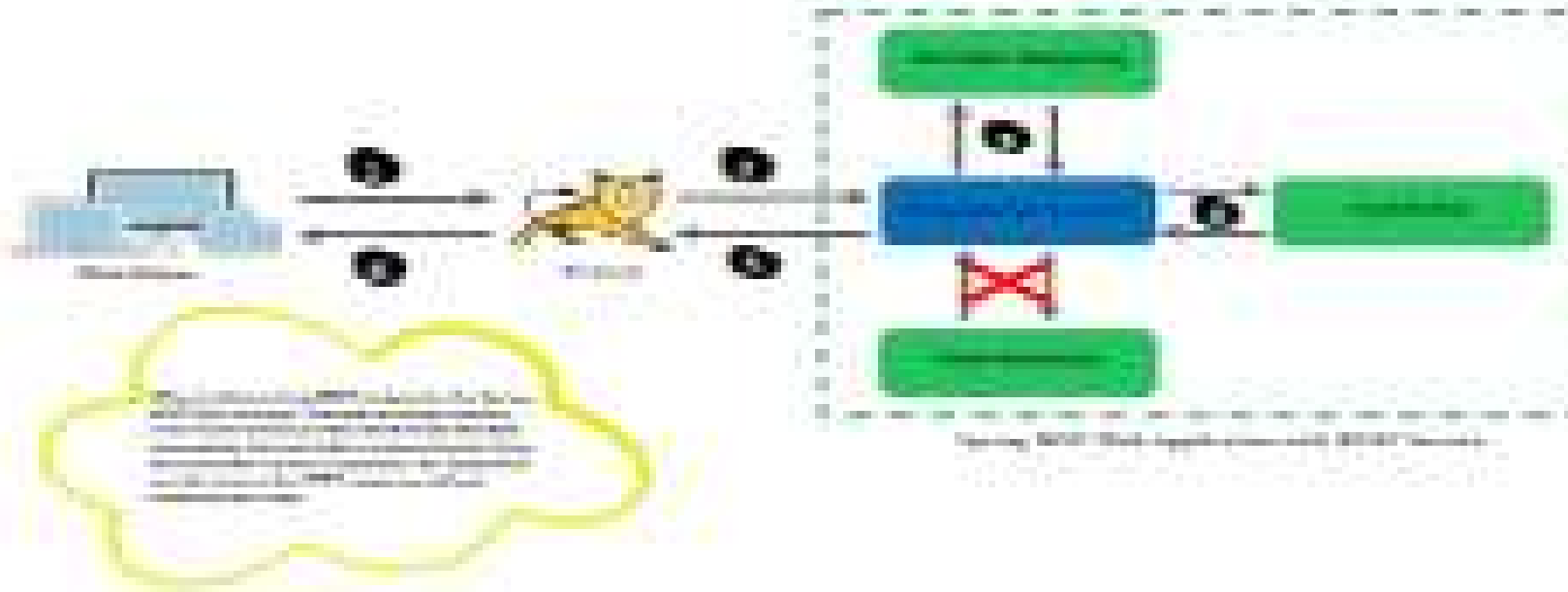
[illegible]

QUICK TIP



Implementing REST Services

SPRING MVC ARCHITECTURE WITH REST SERVICES



Implementing REST Services

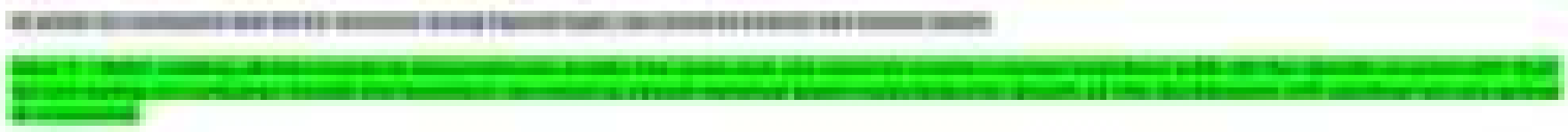
Different Annotation & Classes in REST

CROSS-ORIGIN RESOURCE SHARING (CORS)

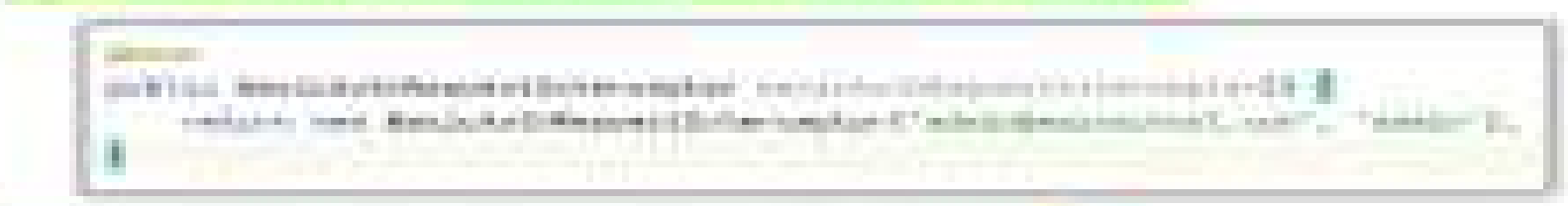
CROSS-ORIGIN RESOURCE SHARING (CORS)

QUICK TIP

Consuming REST Services using Openfeign



Consuming REST services using Openfeign

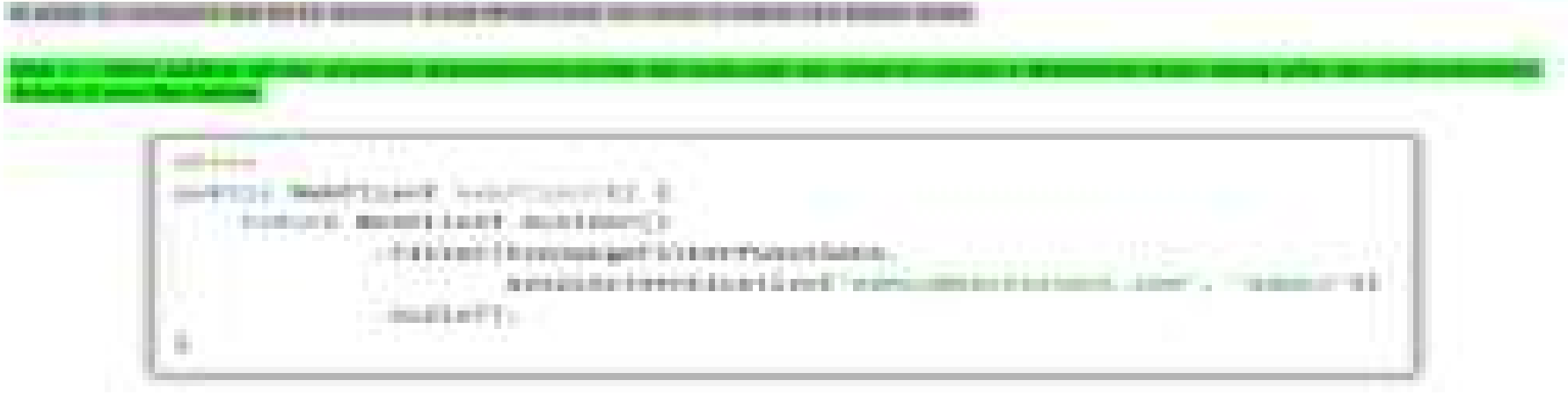


Consuming REST Services using RestTemplate

Consuming REST Services using RestTemplate

```
@Override
public ResponseEntity<Response> saveEmployee(Employee employee){
    String url = "http://localhost:8080/spring-rest/saveEmp";
    HttpHeaders headers = new HttpHeaders();
    headers.add("Accept", "application/json");
    HttpEntity<Employee> entity = new HttpEntity<Employee>(employee, headers);
    ResponseEntity<Response> response = restTemplate.exchange(url, HttpMethod.POST,
                    entity,Response.class);
    return response;
}
```

Consuming REST services using WebClient

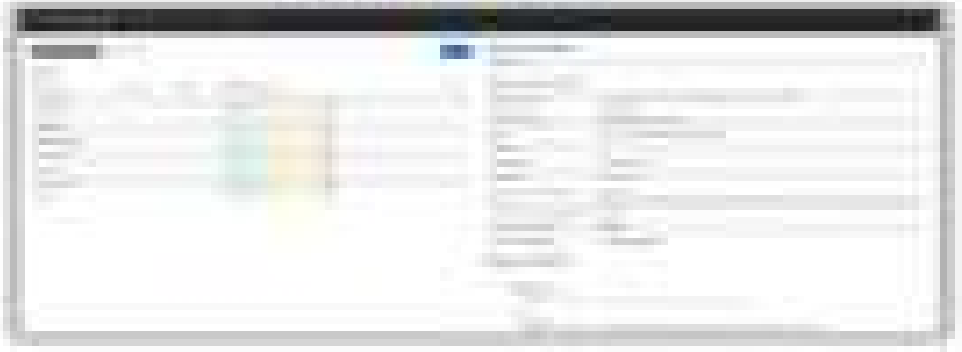


Consuming REST services using WebClient

```
@SpringBootApplication
public class SpringBootWebclientExampleApplication {

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

}
```

[illegible]

- [illegible]
- [illegible]
- [illegible]



QUICK TIP

Logging

Logging inside springboot

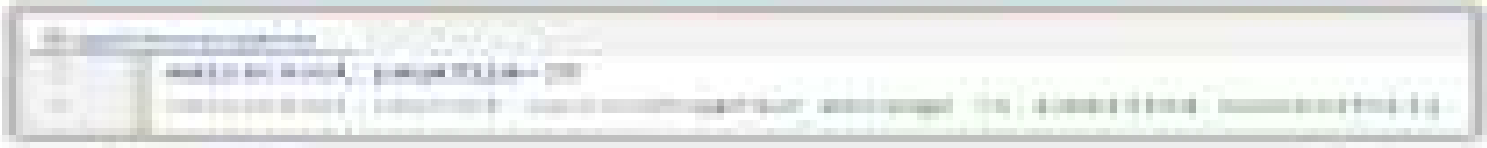
Logging inside spring boot

QUICK TIP

Configurations

Reading properties with @Value

Reading properties using Environment



Reading properties with @ConfigurationProperties

- [redacted]
- [redacted]

```
[redacted]
```

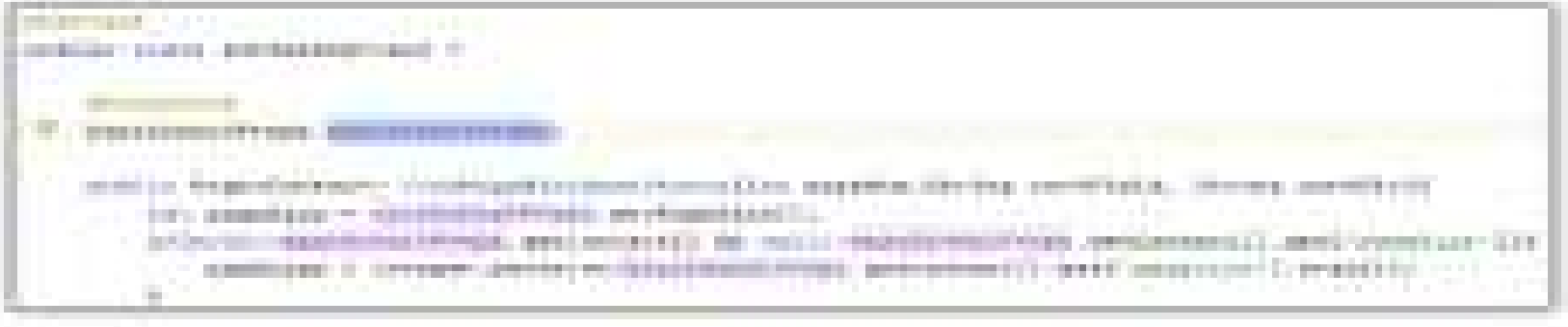
Reading properties with getConfigurationProperties

- Ideal in Gradle scripts that access some external resources

```
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

    [illegible]
    [illegible]
    [illegible]
    [illegible]
    [illegible]
}
```

Reading properties with @ConfigurationProperties

- [illegible]



Reading properties with @PropertySource

Profiles

[illegible]

- [illegible]

- [illegible]

- [illegible]

Configuring Profiles

- [illegible]

- [illegible]

- [illegible]

QUICK TIP

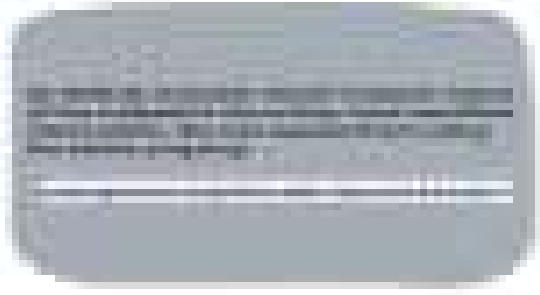
Conditional Bean creation using Profiles

- [illegible]

```
[illegible]
[illegible]
[illegible]
[illegible]
}
```

```
[illegible]
[illegible]
[illegible]
[illegible]
}
```

Spring Boot Actuator



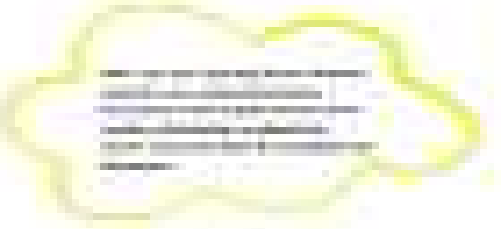
API points provided by Actuator

[illegible]	Path	Description
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

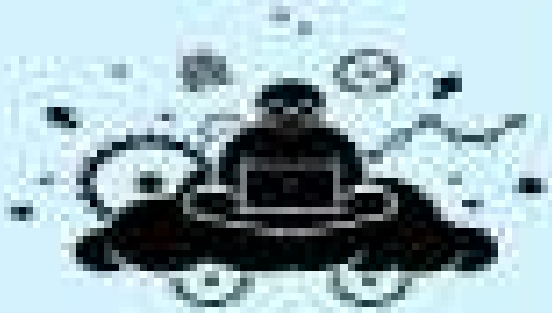
Spring Boot Actuator

API points provided by Actuator

HTTP method	Path	Description
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]



Deploying Spring Boot Applications



AWS

Elastic Beanstalk

AWS EC2 vs AWS Elastic Beanstalk

AWS Elastic Beanstalk - How it works

THANK YOU