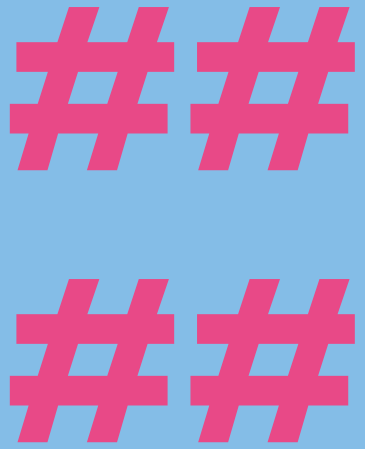
Spring Framework

Marek Brož and Jason Tavernier

Contents

What is Spring?

- Java Framework
- Mostly used for web app development
- makes programming Java quicker, easier, and safer for everybody
- Spring’s focus is on speed, simplicity, and productivity
- Live reload - makes it easier to work with by reloading the page if any changes are made in the files



History

- The first version was written by Rod Johnson in 2003
- Acquired by VMWare in 2009
- Spring framework 4.0 was released in 2013, adding support for Websockets
- Spring Boot 1.0 was released in April 2014
- Spring Framework 4.2.0 was released on 31 July 2015, bringing support for Java 6, 7 and 8
- Spring 4.3. released in 2016 with Java 6+ and Servlet 2.5+
- Spring Framework 6.0 released in 2022 with support for Java 17+ and a move to Jakarta EE 9+

Languages & tools

Build Tools

- Maven
- Gradle
- Apache ant

Languages

- Kotlin
- Groovy
- Java
- JRuby

Comparison

```java
@FXML
    void ValidateBtn_click(ActionEvent event) {
        idErrorLbl.setText("");
        pcnErrorLbl.setText("");
        firstNameErrorLbl.setText("");
        lastNameErrorLbl.setText("");
        SuccessLabel.setText("");

        boolean idValid = true;
        boolean pcnValid = true;
        boolean firstNameValid = true;
        boolean lastNameValid = true;

        Color red = Color.RED;
        Color green = Color.GREEN;
        if(idTxtFld.getText().isEmpty()){
            idErrorLbl.setTextFill(red);
            idErrorLbl.setText("Please fill in an ID");
            idValid = false;
        }
        if(PCNTxtFld.getText().isEmpty()){
            pcnErrorLbl.setTextFill(red);
            pcnErrorLbl.setText("Please fill in a PCN");
            pcnValid = false;
        }
        if(firstNameTxtFld.getText().isEmpty()){
            firstNameErrorLbl.setTextFill(red);
            firstNameErrorLbl.setText("Please fill in a first name");
            firstNameValid = false;
        }
        if(lastNameTxtFld.getText().isEmpty()){
            lastNameErrorLbl.setTextFill(red);
            lastNameErrorLbl.setText("Please fill in a last name");
            lastNameValid = false;
        }
        Pattern digitPattern = Pattern.compile("[^0-9]");
        if(digitPattern.matcher(idTxtFld.getText()).find() && idValid){
            idErrorLbl.setTextFill(red);
            idErrorLbl.setText("ID has to be numbers");
            idValid = false;
        }
        if(digitPattern.matcher(PCNTxtFld.getText()).find()&& pcnValid){
            pcnErrorLbl.setTextFill(red);
            pcnErrorLbl.setText("PCN has to be numbers");
            pcnValid = false;
        }
        Pattern textPattern = Pattern.compile("[^a-zA-Z]");
        if(textPattern.matcher(firstNameTxtFld.getText()).find() && firstNameValid){
            firstNameErrorLbl.setTextFill(red);
            firstNameErrorLbl.setText("First name has to be only letters");
            firstNameValid = false;
        }
        if(textPattern.matcher(lastNameTxtFld.getText()).find() && lastNameValid){
            lastNameErrorLbl.setTextFill(red);
            lastNameErrorLbl.setText("Last name has to be only letters");
            lastNameValid = false;
        }


        if((firstNameTxtFld.getText().length() < 2 || firstNameTxtFld.getText().length() > 20) && firstNameValid){
            firstNameErrorLbl.setTextFill(red);
            firstNameErrorLbl.setText("First name has to be at least 2 and at most 20 characters");
        }
        if((lastNameTxtFld.getText().length() < 2 || lastNameTxtFld.getText().length() > 20) && lastNameValid){
            lastNameErrorLbl.setTextFill(red);
            lastNameErrorLbl.setText("Last name has to be at least 2 and at most 20 characters");
        }

        if(idValid && pcnValid && firstNameValid && lastNameValid){
            int id;
            int pcn;
            String firstName = firstNameTxtFld.getText();
            String lastName = lastNameTxtFld.getText();

            try{
                id = Integer.parseInt(idTxtFld.getText());
                pcn = Integer.parseInt(PCNTxtFld.getText());
            }catch(NumberFormatException ex){
                SuccessLabel.setTextFill(red);
                SuccessLabel.setText("Something went wrong, please make sure all fields are correct");
                return;
            }

            SuccessLabel.setTextFill(green);
            SuccessLabel.setText("congratulations, everything is filled in correctly.");
            Student student = new Student(id, pcn, firstName, lastName);
            System.out.println(student.toString());
        }
    }
```

```java
public class Student {

    //INITIALISE VARIABLES
    @NotNull
    private Integer id;
    @NotNull
    private Integer PCN;
    @NotNull
    @Pattern(regexp="^[A-Za-z]*$", message = "Must be characters")
    @Size(min=2, max=30)
    private String firstName;
    @NotNull
    @Pattern(regexp="^[A-Za-z]*$", message = "Must be characters")
    @Size(min=2, max=30)
    private String lastName;
    private Specialisation specialisation;

    ...
}
```

Main Features

Dependency Injection

Inversion of control

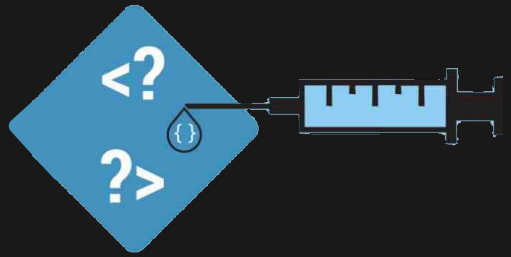
MVC architecture

Dependency Injection

Manages dependencies between objects

We use @Autowired annotation to do this. Used for setter methods, non-setter methods, constructor and properties.

Dependency Injection is achieved by passing object dependencies at runtime rather than at compile time



Dependency Injection

Without

```java
public class Foo {
  private Bar bar;

  public Foo() {
    this.bar = new Bar();
 }

  //Other methods that use bar...
}
```

With

```java
public class Foo {
  private Bar bar;

  @Autowired
  public Foo() {
    this.bar = new Bar();
 }

  //Other methods that use bar...
}
```

Annotations

@Component

Annotation labels down a class as a bean which we need to manage

```
@Component
public class MyModule {
  public String retrieveEnrollmentMessage(){
    return "Welcome to Module";
```

```
@RestController
public class MyTerm {
  @Autowired
  private MyModule module;
  @RequestMapping("/enrollment")
  public String enrollment() {
    return service.retrieveEnrollmentMessage();
  }
}
```

@Autowired

Marks and implicitly injects the variables for which needs to find the correct matching object. It supports both XML and annotation configurations as well

Dependency Injection

Without

```
@RestController
public class MyTerm {
   private MyModule module= new MyModule();
  @RequestMapping("/enrollment")
  public String enrollment() {
      return service.retrieveEnrollmentMessage();
   }
}
```

With

```
@Component
public class MyModule {
  public String retrieveEnrollmentMessage(){
     return "Welcome to Module";
 }
}
@RestController
public class MyTerm {
  @Autowired
  private MyModule module;
  @RequestMapping("/enrollment")
  public String enrollment() {
     return service.retrieveEnrollmentMessage();
  }
}
```

Inversion of control

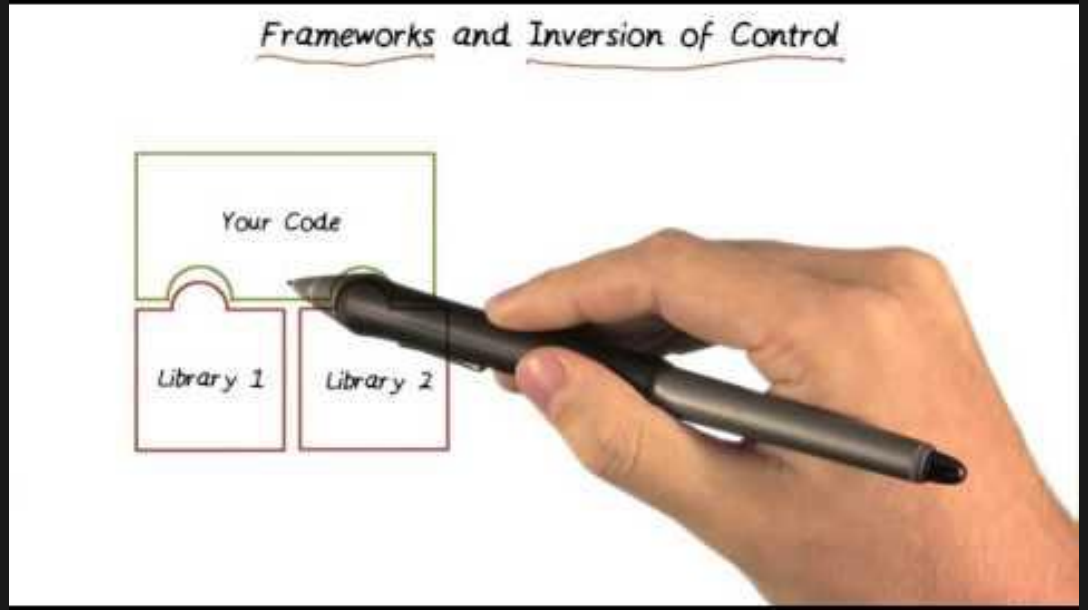
Core of spring

Creates and manages objects for us

Helps our applications be more modular

Uses dependency injection

Inversion of Control



Inversion of Control

Here is an example of how IoC can be implemented in Spring using the @Component and @Autowired annotations:

```java
// The component that will be managed by the IoC container
@Component
public class Foo {
    // A dependency that will be injected by the IoC container
    private Bar bar;

    // The constructor is annotated with @Autowired, indicating that
    // an instance of Bar should be provided by the IoC container
    @Autowired
    public Foo(Bar bar) {
        this.bar = bar;
    }

    // Other methods that use bar...
}
```

Inversion of Control

To use this code with the IoC container, the Foo and Bar classes must be registered with the container and the container must be started:

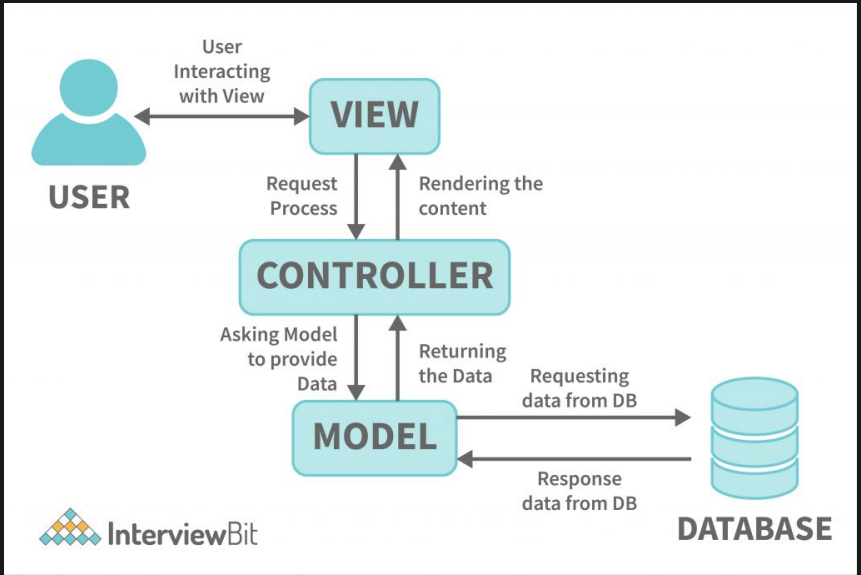
```java
// Create the IoC container
AnnotationConfigApplicationContext context =
    new AnnotationConfigApplicationContext();

// Register the components with the IoC container
context.register(Foo.class, Bar.class);

// Start the IoC container
context.refresh();

// Get the Foo instance from the IoC container
Foo foo = context.getBean(Foo.class);
```

MVC architecture

Model-view-controller model

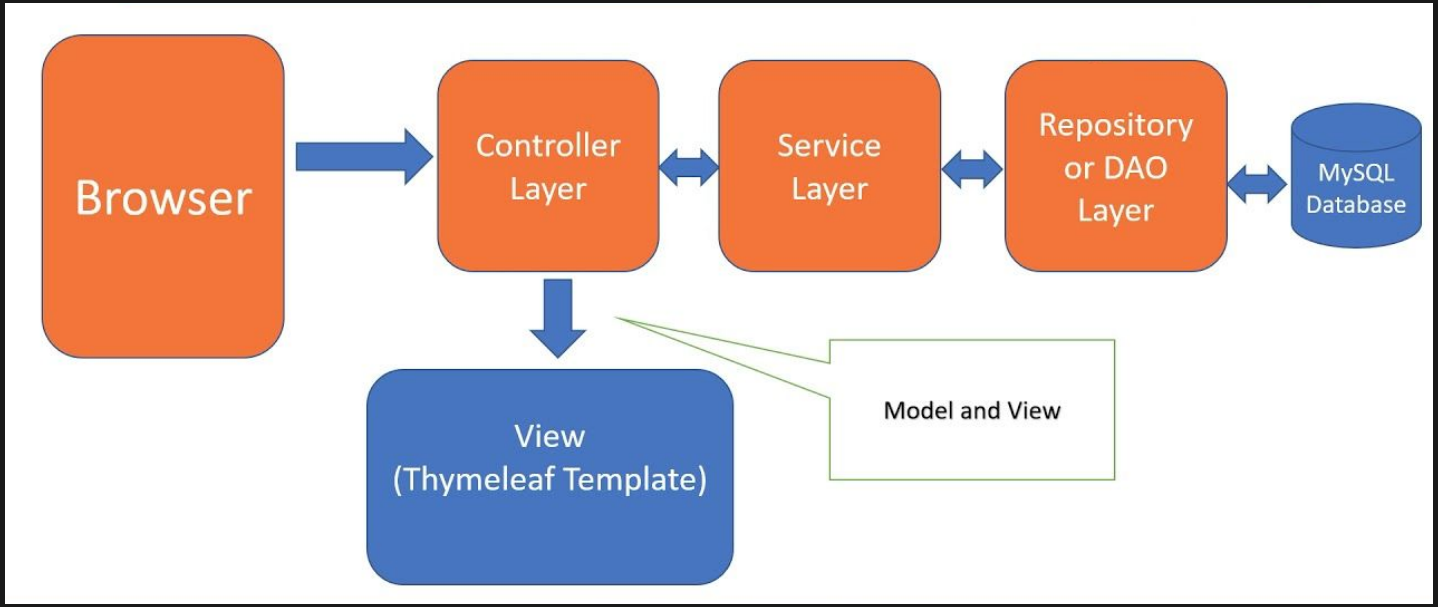
Architecture for websites

Helps with separation of parts



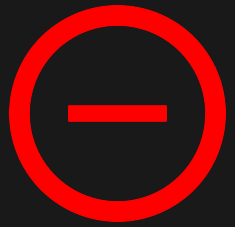
MVC architecture

The implementation of the MVC framework in the context of Spring:



Pros and Cons

- **Flexible** - provides trusted flexible libraries
- **Loose Coupling** - prevents spaghetti + easy to test
- **Lifecycle** - efficient management of all your application components
- **Fast** - fast startup, shutdown, and optimized execution



- **Complex**
- **No specific guidelines** - It does not handle XSS or cross-site scripting. Own security required
- **High learning curve** - new programming methods
- **Parallel Mechanism** - multiple options lead to confusion

Assignment 1

XX

%%

Annotations

SpringBootApplication

Create and configure a new Spring app. Mark a configuration class that declares one or more @Bean methods and also triggers auto-configuration and component scanning.

Controller

Auto-detect implementation classes through the classpath scanning.

RestController

is a special controller used in RESTful Web services. Responds with a json format

Annotations

RequestParam

extracts query parameters, form parameters, and even files from the request.

Getmapping

Indicates that the URL is a GET request to the provided route

```
@GetMapping("/student")
    public Student createStudent(@RequestParam(defaultValue = "Marek") String firstName,
@RequestParam(defaultValue = "Broz") String lastName) {
        return new Student(id, PCN, String.format(firstNameTemplate,
firstName),String.format(lastNameTemplate, lastName));
    }
```

Assignment 2

XX

%%%

Annotations

NotNull

Indicates that the field can not be null

Size

Indicates the minimum and maximum length of the field

Pattern

Adds a regex pattern that the field has to conform to

PostMapping

Indicates that the URL is a POST request to the provided route

Quiz

Joinmyquiz.com

XXX

Summary

XXX

https://spring.io

spring by VMware Tanzu

Why Spring | Learn | Projects | Training | Support | Community

Hear from the Spring team this January at SpringOne

Virtual • Free • Jan 24–26

REGISTER NOW

NEWS Native Support in Spring Boot 3.0.0-M5

What Spring can do

Microservices

Quickly deliver production-grade features with independently

Reactive

Spring's asynchronous, nonblocking architecture means you can get more

Cloud

Your code, any cloud—we've got you covered. Connect and scale your services

Web apps

Frameworks for fast, secure, and responsive web applications connected to

Questions?

Feedback