Adobe

Java DevCamp 2023

Girls edition

Agenda Workshop #2

REST

MVC

Maven – pom.xml

Jackson

Spring

Spring Boot

Advertising Campaign Management App flow

Advertiser

= brand/app who want to promote a product

- the one who pays the money to get his advertisements shown

e.g. Zara

Campaign

= strategy that that is carried out across different mediums in order to both achieve results and to increase brand awareness, sales and communication within a specific market

name: Holiday Campaign
time: 1st - 31st december
target: Females interested in fashion with age between 18 & 30

Publisher

= digital space to display the campaign

- the one who gets the money for showing the ads on his site

Fashion Days

User

RESTful APIs

RESTful APIs

"RESTful APIs, or **Representational State Transfer APIs**, play a pivotal role in shaping modern web development by facilitating seamless communication between different software systems."

RESTful APIs - keypoints

- **Interoperability**
  - RESTful APIs enable interoperability between diverse systems and platforms, allowing them to communicate and exchange data effortlessly
- **Client-Server Architecture**
  - REST follows a client-server architecture, promoting a clear separation of concerns. This enhances scalability and allows for independent development of the client and server components
- **Statelessness**:
  - RESTful APIs are stateless, meaning each request from a client to a server contains all the information needed to understand and fulfil that request. This simplicity enhances reliability and scalability

RESTful APIs - keypoints

- **Standardized Communication**
  - HTTP, the foundation of RESTful APIs, provides a standardized and widely adopted communication protocol. This fosters consistency and ease of integration across different applications
- **Scalability and Performance**
  - The stateless nature of RESTful APIs makes them inherently scalable. As a result, systems can efficiently handle a growing number of users and requests, contributing to improved performance
- **Flexibility and Simplicity**
  - RESTful APIs are designed to be simple and flexible, making them accessible to developers of varying expertise. This simplicity encourages rapid development and ease of integration

RESTful APIs - keypoints

- **Support for Multiple Data Formats**
  - RESTful APIs support various data formats, such as JSON and XML, allowing developers to choose the format that best suits their application's needs
- **Ecosystem Integration**
  - In the age of microservices and distributed architectures, RESTful APIs play a pivotal role as a central component for seamlessly integrating diverse services within a larger ecosystem

RESTful APIs - Basics

- **Resource:**
  - A resource is a key abstraction in RESTful APIs. It can be any entity, such as data or a service, that is identifiable by a URI (Uniform Resource Identifier). Resources are manipulated using standard HTTP methods
- **URI (Uniform Resource Identifier):**
  - URIs uniquely identify resources. Each resource in a RESTful API is assigned a URI, allowing clients to interact with the resource using standard HTTP methods
- **Representation**
  - Resources are represented in different formats, such as **JSON** or **XML**. Clients can request or submit data in various representations based on their preferences

RESTful APIs - Basics

- **HTTP Methods (CRUD operations)**
  - RESTful APIs use standard HTTP methods for **CRUD** operations
    - **GET**: Retrieve a resource
    - **POST**: Create a new resource
    - **PUT** or **PATCH**: Update an existing resource
    - **DELETE**: Delete a resource
- **HTTP Status Codes**
  - HTTP status codes convey the *outcome of a request*. Common status codes include:
    - **2xx:** Success
    - **3xx:** Redirection
    - **4xx:** Client error
    - **5xx:** Server error

RESTful APIs - Basics

- **Query Parameters**
  - Query parameters are often used to *filter*, *sort*, or *paginate* resources. They provide additional information to the server about how the client wants to interact with the resources
- **Authentication and Authorization**
  - RESTful APIs use standard authentication mechanisms (such as API keys, OAuth, or tokens) to secure access. Authorization mechanisms control what actions a client can perform on specific resources
- **Error Handling**
  - API responses should include **clear and standardized error messages** in case of failures. This helps clients understand and handle errors effectively.

MVC

Model-View-Controller architecture

- **Model**
  - data & business logic
- **View**
  - user interface
- **Controller**
  - actions processing

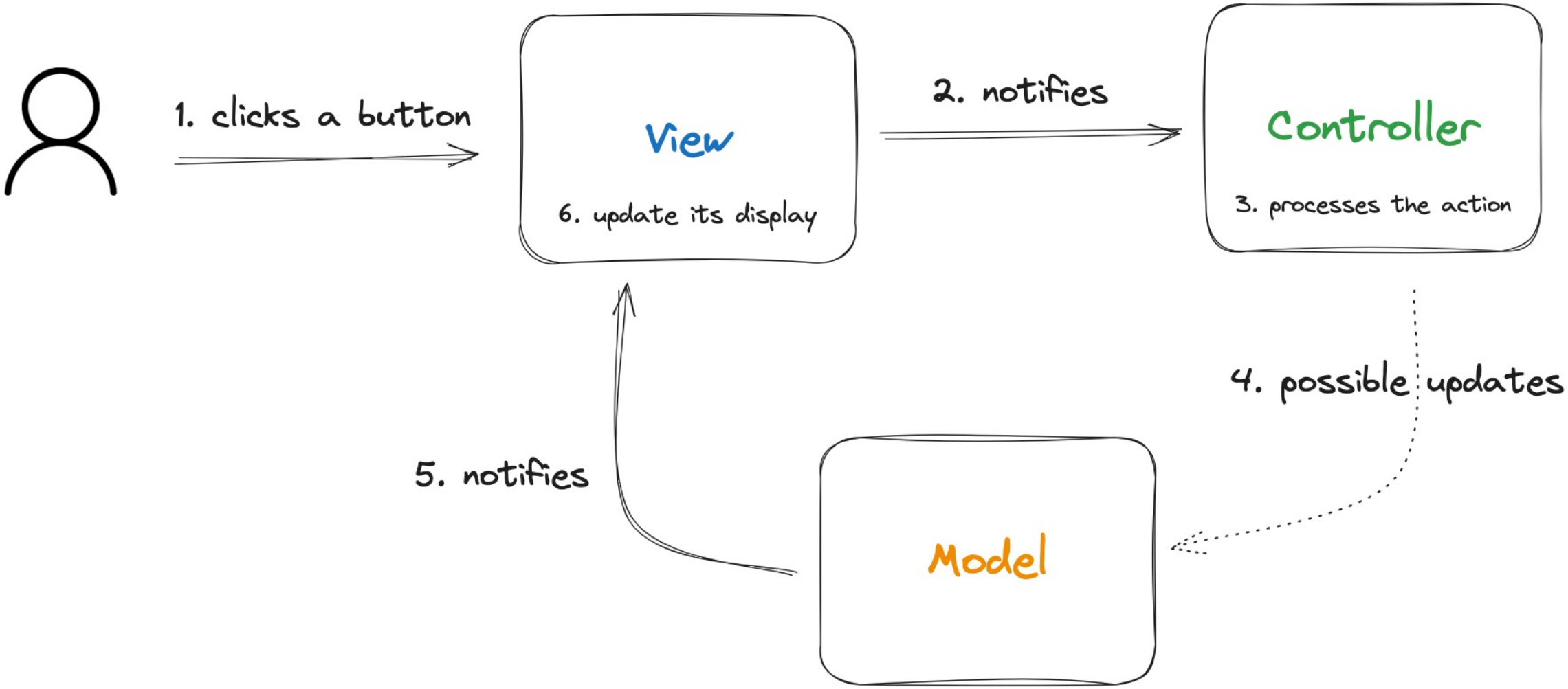
Why Model-View-Controller?

- **Separation of Concerns**
  - each component has a specific role

- **Modularity**
  - changes to one component should not imply changes in others => code reuse and maintainability

- **Flexibility**
  - the components can be developed and modified independently, allowing for easier testing and updates

Model-View-Controller workflow example

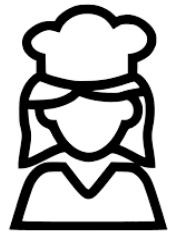


Model-View-Controller analogy with a bakery

- **Model**
  - the list of ingredients + recipe of the cupcake

- **View**
  - the beautifully decorated cupcake that customers see in the display case

- **Controller**
  - the baker who manages the entire process
  - it takes orders from customers (user input),
    - gets the ingredients (data) from the kitchen (Model),
      - follows the recipe (business logic),
      - and then presents the finished cupcake (updates the View)



project setup

Maven – What is a pom.xml file?

POM = Project Object Model

- Information about the project
- Configuration details
  - + dependencies used by Maven to build the project

Jackson

- Popular Java library for working with JSON data

- **jackson-databind** module
  - Deserialization = convert JSON data to Java objects
  - Serialization = covert Java objects to JSON data

- **ObjectMapper** class
  - central component that provides functionality for reading and writing JSON data.

```
ObjectMapper objectMapper = new ObjectMapper();

MyObject myObject = objectMapper.readValue(jsonString, MyObject.class); // Deserialization
String jsonString = objectMapper.writeValueAsString(myObject); // Serialization
```

Jackson

- **@JsonCreator**
  - indicates that a particular method/constructor should be used to deserialization
  - parameters are annotated with **@JsonProperty** mentioning the corresponding JSON property name

  - Let's go back to code: **UserDTO**

What is Spring?

A toolkit that simplifies the complexity of Java development, empowering developers to create robust, scalable, and flexible applications effortlessly

Spring Framework - keypoints

✨Empowering Java Development✨

✨Dependency Injection Magic✨

✨Harmony with AOP✨

✨Modular and Extensible Architecture✨

✨Configuration Made Easy✨

✨Versatility Across Layers✨

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run"

"We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration."

Spring – Core Concepts

Inversion of Control (IoC)

- Instead of objects creating and managing their dependencies, **Spring's IoC container** is responsible for *managing the objects and their relationships*
- This is achieved through techniques such as **dependency injection**

Dependency Injection

- A **design pattern** in which the dependencies of a class are *injected from the outside rather than being created within the class*

Spring Containers

- The containers provide a controlled environment for beans to live, offering a variety of services to enhance the overall application development experience
- Spring provides two main types of containers: the **BeanFactory** and the **ApplicationContext**

Spring Beans

- A bean is a **fundamental building block** — a managed object that is created, configured, and managed by the Spring IoC container
- Beans represent the various components and services in your application, such as *controllers*, *services*, *data access objects*, and more

Spring Boot – Core Concepts

Spring Boot – Core Concepts

- Convention over Configuration
- Embedded Servers
- Auto-Configuration
- Microservices Support
- Production-Ready Defaults
- Integrated Spring Ecosystem
- Externalized Configuration

Spring Boot – Core Concepts

- **Convention over Configuration**
- Embedded Servers
- Auto-Configuration
- Microservices Support
- Production-Ready Defaults
- Integrated Spring Ecosystem
- Externalized Configuration

Spring Boot embraces the principle of ***'Convention over Configuration***,' *minimizing the need for extensive setup and boilerplate code*. Developers can focus on application logic rather than intricate configurations

Spring Boot – Core Concepts

- Convention over Configuration
- **Embedded Servers**
- Auto-Configuration
- Microservices Support
- Production-Ready Defaults
- Integrated Spring Ecosystem
- Externalized Configuration

Say goodbye to external server setups. Spring Boot comes with embedded servers like **Tomcat**, **Jetty**, or **Undertow**, allowing you to *run your application as a standalone executable*, simplifying deployment.

Spring Boot – Core Concepts

- Convention over Configuration
- Embedded Servers
- **Auto-Configuration**
- Microservices Support
- Production-Ready Defaults
- Integrated Spring Ecosystem
- Externalized Configuration

No more manual configuration hassles. Spring Boot's **auto-configuration feature** analyzes your project's dependencies and automatically configures your application, offering a hassle-free setup experience.

Spring Boot – Core Concepts

- Convention over Configuration
- Embedded Servers
- Auto-Configuration
- **Microservices Support**
- Production-Ready Defaults
- Integrated Spring Ecosystem
- Externalized Configuration

In the era of microservices, Spring Boot ✨*shines*✨. Its modular design and embedded container support make it an ***ideal choice*** for building and deploying microservices architecture with minimal effort.

Spring Boot – Core Concepts

- Convention over Configuration
- Embedded Servers
- Auto-Configuration
- Microservices Support
- **Production-Ready Defaults**
- Integrated Spring Ecosystem
- Externalized Configuration

Spring Boot is designed with **production readiness** in mind. It provides default configurations for *security*, *logging*, and *monitoring*, allowing developers to focus on building features rather than worrying about infrastructure concerns.

Spring Boot – Core Concepts

- Convention over Configuration
- Embedded Servers
- Auto-Configuration
- Microservices Support
- Production-Ready Defaults
- **Integrated Spring Ecosystem**
- Externalized Configuration

Leveraging the entire Spring ecosystem seamlessly, Spring Boot integrates effortlessly with other Spring projects. This ensures a **cohesive development experience** and access to a rich set of features.

Spring Boot – Core Concepts

- Convention over Configuration
- Embedded Servers
- Auto-Configuration
- Microservices Support
- Production-Ready Defaults
- Integrated Spring Ecosystem
- **Externalized Configuration**

Spring Boot allows **externalized configuration**, enabling the modification of application properties without changing the code. This flexibility is crucial for *adapting to various environments and deployment scenarios*.


Adobe
Artwork by Frank Moth