Minicurso Spring e Android

Criado por Gabriel Schaidhauer - Aula 2

Design Patterns

Design Patterns

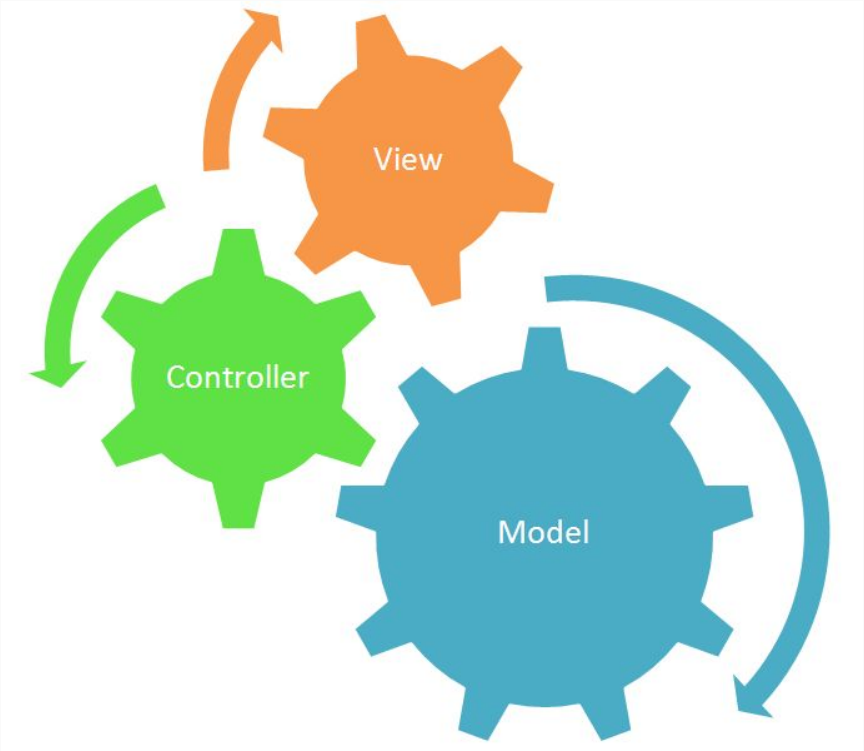
Design Patterns ou Padrões de Design/Desenvolvimento são conjuntos de boas práticas e técnicas de desenvolvimento já reconhecidas como efetivas nas suas determinadas usabilidades tanto pelo mercado quanto por estudiosos das áreas onde foram desenvolvidas

Design Patterns - Exemplo

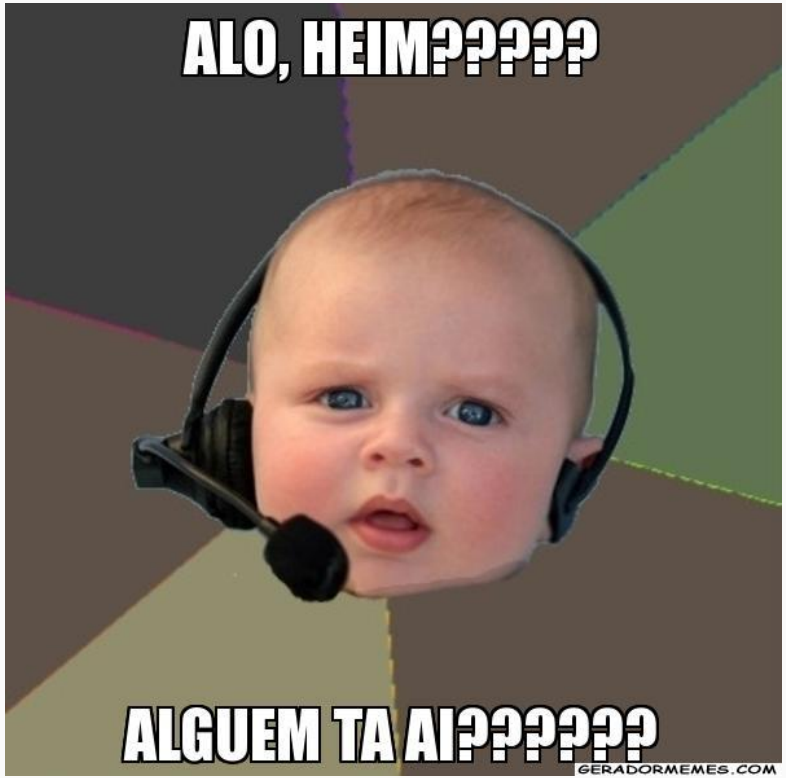
Padrão MVC

Padrão MVC

Padrão de desenvolvimento de software em camadas, simples, que permite maior abstração das lógicas de negócio no código.



Ahn?



Desenvolvimento em Camadas

Em um restaurante trabalham o garçom Cláudio, o auxiliar de pedidos de cozinha Geraldo e o cozinheiro Marcelo.

Cláudio recebe os pedidos dos clientes, e informa eles de como está sendo realizado o pedido. Geraldo recebe o pedido pronto avalia se é um prato servido no dia e passa para o Marcelo. Marcelo avalia se tem tudo para realizar o prato, cozinha tudo e entrega o prato pronto para o Geraldo, que entrega para o Cláudio que serve o cliente.

E o que a cozinha têm a ver com programação?

Desenvolvimento em Camadas

Essa história da cozinha é basicamente um exemplo de MVC, onde o garçom é a **View** o auxiliar de pedidos de cozinha é a **Controller** e o cozinheiro é a **Model**

View

A View é a camada de apresentação, onde o nosso usuário vai interagir com os dados, e fazer as suas requisições para a aplicação

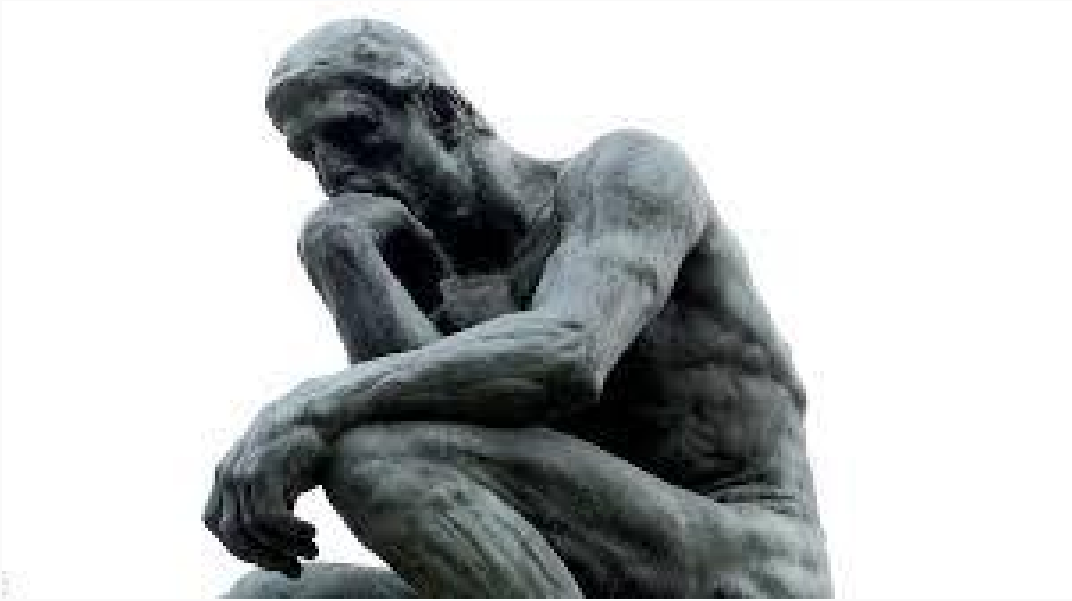
Controller

A Controller acaba agindo como uma porta de entrada para nossa aplicação. Ela é o meio de campo entre a View e a Model e seu papel é transmitir o que a View relata para a Model.

Model

Model é a camada responsável por executar as lógicas de negócio da aplicação, realizar validações e cuidar da parte de dados.

Então para cada camada eu crio uma classe e deu?



Modelo MVC

Camada não é classe! Uma camada pode conter uma série de classes, até mesmo uma série de pacotes. Uma camada é um compartimento lógico, as suas partes devem se relacionar quanto às suas destinações dentro do sistema.

REST

REST

REST é um padrão de desenvolvimento de WebServices baseado no protocolo HTTP, funcionando através de chamadas e respostas de forma pré-definida por este protocolo.

Protocolo HTTP

Protocolo HTTP

O protocolo HTTP se baseia em Requisições e Respostas, onde cada tipo de requisição representa um tipo de interação com o servidor

Verb	Path	Action
GET	`/photo`	index
GET	`/photo/create`	create
POST	`/photo`	store
GET	`/photo/{photo}`	show
GET	`/photo/{photo}/edit`	edit
PUT/PATCH	`/photo/{photo}`	update
DELETE	`/photo/{photo}`	destroy

GET

GET é o tipo de requisição utilizado para buscar, filtrar, encontrar um ou mais itens no servidor, ou ainda acionar métodos que não possuam parâmetros de entrada.

POST

POST é um tipo de requisição usado geralmente para inserção de recursos ou execução de rotinas que recebam um parâmetro.

PUT/PATCH

PUT e PATCH são dois tipos de requisições utilizados para edições. PUT é usado para editar campos de um recurso e PATCH para substituí-lo por completo.

DELETE

DELETE é um tipo de requisição voltado para a exclusão de um registro seja ela física ou lógica

Respostas

As respostas no protocolo HTTP são formadas de um corpo e um código. O corpo pode conter tanto um conteúdo a ser exibido quando objetos em formato JSON, XML, Texto, etc. Os códigos diversos significando cada um algo diferente. Para bem estruturar a sua API é indicado que utilize os códigos de resposta mais adequados com o que foi realizado. Os principais podem ser estudados AQUI.

MVC no Spring

Controllers

Controllers dentro do spring são classes anotadas com “@RestController”, com métodos ligados a endereços HTTP

```java
package hello;

import java.util.concurrent.atomic.AtomicLong;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World")
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

@RestController

Anotação que indica que a classe é um controller e que seus métodos anotados com @RequestMapping devem ser publicados em endpoints acessíveis via HTTP nos tipos de requisição indicados

@RequestMapping

Esta anotação indica a qual recurso Recurso HTTP o método estará ligado, e a qual tipo de requisição (GET, POST...). Este método pode ser usado tanto na classe quanto nos métodos, entretanto, quando usado na classe irá indicar o recurso pai para acesso a todos os métodos da classe

@RequestParam

Esta anotação é vínculada a parâmetros de entrada de um método anotado com ‘@RequestMapping’. Ela indica que o parâmetro terá como origem um item de QueryString na URL da requisição HTTP, e pode indicar o nome do atributo e sua obrigatoriedade ou não.

@RequestBody

Esta anotação indica que o parâmetro de entrada do método se refere a um corpo de requisição. O corpo pode ser um objeto mapeado a partir de um JSON. Esta anotação suporta que seja indicado a sua obrigatoriedade ou não.

ResponseEntity

ResponseEntity é um objeto do Spring criado para servir como uma resposta HTTP, podendo conter um corpo, e um código de retorno.

Exception Handling

No java, quando ocorre um erro geralmente ele se apresenta como uma Exception, sendo este um objeto que representa um erro. Exceptions nada significam para quem consome a nossa API, portanto é essencial que tratemos as exceções para que elas não cheguem até o nosso usuário.

Exception Handling

Uma das melhores formas de tratar exceções em uma API é no momento em que elas ocorrem, indicando a mensagem apropriada, mas caso não seja possível é bom estar pronto.

```
@ControllerAdvice
class GlobalControllerExceptionHandler {
    @ResponseStatus(HttpStatus.CONFLICT)  // 409
    @ExceptionHandler(DataIntegrityViolationException.class)
    public void handleConflict() {
        // Nothing to do
    }
}
```

@ControllerAdvice

Esta é uma anotação, utilizada em classes, que permite que a classe anotada com ela sirva como um tratador de exceções ao longo de toda a aplicação. Qualquer exceção que seja lançada por uma controller, poderá ser capturada pelo classe anotada com @ControllerAdvice e assim tratada para que emita a saída correta.

@ExceptionHandler

Esta anotação deve ser utilizada nos métodos onde uma exceção irá ser tratada. A ela deve ser informado qual exceção o método irá tratar, e este método poderá receber por parâmetro uma instância desta exceção.

JPA

JPA

JPA é um Framework Java para persistência de dados, que funciona como um ORM (Object Relational Mapping). Este tipo de framework faz a ligação entre objetos do código com a base de dados, facilitando a integração entre a linguagem orientada a objetos e os bancos relacionais.

Spring Data

Spring Data

O Spring data é um dos módulos do Spring, sendo o responsável por trabalhar com implementações do JPA, e traz muita agilidade no momento de criar as nossas interações com a base de dados da nossa aplicação

Entidades

Entidades são os objetos POJO (Plain Old Java Object) que através de anotações é ligado a uma tabela do banco de dados, servindo como se estivessemos manipulando o próprio registro.

```
@Entity
public class Tarefa {

    @Id
    @GeneratedValue
    private Long id;

    private String descricao;
    private boolean finalizado;

    @Temporal(TemporalType.DATE)
    private Calendar dataFinalizacao;

    // métodos...
}
```

@Entity

Esta é a anotação que indica que o objeto se trata de uma entidade mapeada.

@Table

Indica a tabela do banco de dados a qual a entidade representa.

@Column

Indica a coluna da tabela do banco de dados a qual um campo da tabela representa

@JoinColumn

Representa uma coluna referente a um relacionamento de chave estrangeira, sendo geralmente acompanhado de uma das anotações que indicam o tipo de relacionamento.

Anotações de Relacionamento

As anotações abaixo indicam os tipos de relacionamento entre tabelas:

- @ManyToOne
- @OneToMany
- @OneToOne

CrudRepository

CrudRepository é uma interface implementada em tempo de execução pelo Spring que realiza as operações em banco de dados relativas ao sistema.

```
Page<User> findByLastname(String lastname, Pageable pageable);

Slice<User> findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, Sort sort);

List<User> findByLastname(String lastname, Pageable pageable);
```

CrudRepository

CrudRepository é uma interface que permite a realização de consultas no banco de dados através de nomes de métodos. Por exemplo se eu quiser buscar um objeto Pessoa pelo seu atributo nome e atributo email:

```
Pessoa findByNomeAndEmail (String nome, String email);
```

@Query

Esta anotação é utilizada em métodos de uma CrudRepository para definir uma query personalizada. Esta query deve estar no formato JPQL utilizado pelo hibernate, podendo ser aprendido AQUI.

Atividade para Casa

Atividade Para Casa

Você deve mapear e criar o banco de dados que prevê ser utilizado no seu trabalho, mapear as entidades referentes a estas tabelas e disponibilizar as cinco operações básicas de uma API para cada uma das suas entidades, empregando os conceitos vistos em aula como @RestController e @ControllerAdvice.

Dúvidas?