# Oque é o Spring?

É um framework do Java, onde facilita o trabalho dos desenvolvedores, vamos imaginar que ele já vem com vários comandos complexos de uma forma mais amigável, onde muita coisa comum já esta criada.

# Utilizando spring a primeira vez

Acesse o site [www.start.spring.io](http://www.start.spring.io)

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Após fazer isso vai gerar um arquivo do seu projeto.

# Começando o trabalho

No nosso projeto vamos fazer linha a linha, então precisamos implementar o **CommandLineRunner**

Esse command ele vai deixar a gente ir fazendo o código linha a linha, e ele vai implementar um novo void run, ali é como se fosse o nosso MAIN vamos digitar tudo ali dentro.Texto

Descrição gerada automaticamente

CommandLineRunner

Ela permite iniciar algo logo após a inicialização da nossa aplicação, como exemplo um banco de dados, vamos imaginar que o usuário entrou no Alta Pressão GNV, usando essa interface, assim que ele abrir o aplicativo já vai iniciar meu banco de dados.

Como usar?

@SpringBootApplication

**public** **class** **MyCommandLineRunner** **implements** **CommandLineRunner** {

@Override

**public** **void** **run**(String... args) **throws** Exception {

System.out.println("Olá, Mundo!");

}

}

Repare que criamos uma Classe e implementamos o CommandLineRunner

No método "run" inserimos a ação que desejamos que seja executada logo depois que a aplicação for iniciada, nesse caso, apenas printamos "Olá, Mundo!".

Lembrar que só devemos utilizar para iniciar algum serviço, não devemos utilizar para tarefas que devem ocorrer periodicamente no meio da aplicação.

# Importando um Maven

Fomos ao site <https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind/2.17.1> - Pegamos sua dependência e colocamos no pom.xml

# Maven x Gradle

São ferramentas de automação de compilação e gerenciamento de dependências

Ou seja, lá vamos adicionar nossas dependências.

## Maven:

É ótimo para gerenciar as dependências do projeto, elas são declaradas no pom.xml. O maven é responsável por baixá-las automaticamente do repositório central do maven.

Convenção sobre Configurações: ela já vem com configurações pré-definidas e podem ser utilizadas fora da caixa sem a necessidade de configuração adicional.

Ciclo de vida padrão: Tem o ciclo de vida padrão com diferentes fases como (compile, test, package, install, deploy), e cada fase dessa executa uma serie de metas. O ciclo de vida permite que você construa o teste do seu projeto facilmente usando comandos do MVN compile, MVN test e MVN package.

Central repositório: O maven possui um site com um repositório gigante para variedades de bibliotecas para o Java prontas: <https://mvnrepository.com/> ; lembrar do GSON que pegamos lá.

## Gradle:

É uma ferramenta de construção e automação de projetos Java, usa linguagem de domínio específico, como **DSL** baseada em Groovy ou Kotlin para definir a estrutura do projeto e tarefas de construção.

Principais conceitos do Gradle:

* Flexibilidade: Ele permite que você mesmo construa seu processo de construção (Configurando o seu processo de construção de acordo com suas necessidades)
* Build by convention: Ele segue algumas convenções, mas te dá mais liberdade do que o Maven na forma como você vai organizar seu projeto.
* Dependency Management: Assim como o Maven ele também tem gerencia de dependências e você pode baixar em repositórios.
* Incremental builds: O Gradle é projetado para realizar compilações incrementais, o que significa que ele vai construir apenas as partes do projeto que foram alteradas desde a última compilação, fazendo o processo ser mais rápido.

# Consumindo uma Api

package br.com.alura.screenmatch.service;  
  
import java.io.IOException;  
import java.net.URI;  
import java.net.http.HttpClient;  
import java.net.http.HttpRequest;  
import java.net.http.HttpResponse;  
  
public class ConsumoApi {  
  
 public String obterDados(String filmeBuscado){  
  
  
 // Criando minhas configurações de acesso  
 String chave = "6e3143f0";  
 String endereco = "https://www.omdbapi.com/?t=" + filmeBuscado.replace(" ", "+") + "&apikey=" + chave;  
  
  
 // Aqui é como se fosse o cliente, onde vamos pedir as informações  
 HttpClient client = HttpClient.*newHttpClient*();  
 HttpRequest request = HttpRequest.*newBuilder*()  
 .uri(URI.*create*(endereco))  
 .build();  
 // Aqui é onde vamos receber as informações.  
 HttpResponse<String> response = null;  
 try { // Aqui vamos fazer uma exceção, se tudo der certo faz isso se não  
 response = client  
 .send(request, HttpResponse.BodyHandlers.*ofString*());  
 } catch (IOException e) { // exibe um erro "AMIGAVEL"  
 throw new RuntimeException(e);  
 } catch (InterruptedException e) {  
 throw new RuntimeException(e);  
 }  
  
 // Aqui é oque vai ser devolvido se tudo der certo, dentro dessa variavel  
 String json = response.body();  
 return json;  
  
 }  
}

## Fazendo meu record para pegar as informações

Lembrar que estamos usando o Jackson databind

<https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind/2.17.1>

* Devemos importar ela no MAVEN para tudo funcionar.

@JsonIgnoreProperties(ignoreUnknown = true) // ignora oque você nao encontrar. || Por padrao ele vem como false, devemos ativar ele para funcionar.

import com.fasterxml.jackson.annotation.JsonAlias;  
import com.fasterxml.jackson.annotation.JsonProperty;  
  
public record DadosSerie(@JsonAlias("Title") String titulo,  
 @JsonAlias("totalSeasons") Integer totalTemporadas,  
 @JsonAlias("imdbRating") String avaliacao,  
 @JsonProperty("imdbVotes") String votos) {  
}

@JsonIgnoreProperties – devemos adicionar isso, porque ele vai tentar puxar todos os dados, até os que você não pediu, e isso vai gerar um erro, fazendo isso ele vai ignorar os dados que você não pediu.

Observa que aqui estamos relacionando um dado com o outro no caso o Title ele vai entender que é igual a título.

@JsonProperty lá no meu arquivo JSON ele vai escrever o imdbVotes e também o votos

Já o @JsonAlias ele vai escrever somente Títulos eu apenas vou ter a informação em títulos.

Por exemplo, se você tiver uma classe Java com a propriedade "nomeCompleto" e o JSON usa "nome" em vez disso, você pode usar o @JsonAlias("nome") para mapear corretamente a propriedade. Dessa forma, tanto "nomeCompleto" quanto "nome" serão aceitos ao fazer o mapeamento:

**public** **class** **Pessoa** {

@JsonAlias({"nomeCompleto", "nome"})

**private** String nomeCompleto;

}

## Objeto do Jackson que faz a conversão

public class ConverteDados implements IConverteDados{  
 private ObjectMapper mapper = new ObjectMapper();

## Métodos genéricos – Criando o meu

public interface IConverteDados {  
 <T> T obterDados(String json, Class<T> classe);  
}

Aqui criamos um contrato (Interface) onde a entrada de dados vai ser do TIPO: genérico, e a classe que vamos usar também vai ser genérico.

## Criando o meu converte dados

@Override  
public <T> T obterDados(String json, Class<T> classe) {  
 try {  
 return mapper.readValue(json, classe);  
 } catch (JsonProcessingException e) {  
 throw new RuntimeException(e);  
 }  
}

Aqui estou dizendo para ele ler os dados e jogar na classe genérica.