CAPACITAÇÕES

JAVA PARA A PDPJ-Br

JAVA AVANÇADO

E-BOOK 1

Ronaldo Pinheiro Gonçalves Junior

Unifor | JUSTIÇA DO TRABALHO Conselho Superior da Justiça do Trabalho | Tribunal Superior Eleitoral | STJ SUPERIOR TRIBUNAL DE JUSTIÇA | JUSTIÇA 4.0 | PNUD | JUSTIÇA FEDERAL Conselho da Justiça Federal | CNJ CONSELHO NACIONAL DE JUSTIÇA

INTRODUÇÃO

OBJETIVO DA TRILHA

Introduzir conceitos avançados de desenvolvimento Java com Spring Boot e Spring Cloud, através da configuração de um ambiente de desenvolvimento e criação de um projeto Spring, e gerenciar a construção e integração, utilizando Apache Maven, e versionamento de código-fonte, utilizando Git.

CONTEÚDOS DA TRILHA

1. Introdução ao Spring Boot e Spring Cloud e configuração do ambiente de desenvolvimento.
2. Criação de um projeto Spring e gerência de build e integration, utilizando Apache Maven.
3. Gerência e versionamento de código-fonte, utilizando Git.

INDICAÇÃO DE MATERIAL COMPLEMENTAR

INDICAÇÃO 1
Tipo: Tutorial
Título: Construindo uma Aplicação com Spring Boot
Link: **https://spring.io/guides/gs/spring-boot/**

INDICAÇÃO 2
Tipo: Tutorial
Título: Manual Interativo de Comandos Git
Link: **https://ndpsoftware.com/git-cheatsheet.html**

1 INTRODUÇÃO AO SPRING BOOT E SPRING CLOUD E CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

1.1 O que é o framework Spring

As ferramentas utilizadas em processos de desenvolvimento de *software* estão em constante modernização e é possível encontrar soluções inovadoras a fim de facilitar e otimizar a construção de novos produtos. Neste contexto, podemos destacar os *frameworks* de desenvolvimento, soluções projetadas especificamente para auxiliar na construção de novas aplicações.

Diferentemente de bibliotecas de linguagens de programação, que visam adicionar componentes utilitários e de suporte ao desenvolvimento, *frameworks* controlam o fluxo de um programa. Isto é, um *framework* não é apenas um conjunto secundário de componentes adicionais, mas, sim, uma estrutura de base primária pré-definida.

Em geral, um *framework* é representado por um código-fonte que não é aberto a mudanças, mas é aberto a extensões. No contexto de desenvolvimento de *software* em Java, este código irá conter interfaces, classes abstratas e classes concretas. Logo, desenvolvedores podem fazer uso de composição e herança, para estender as funcionalidades originais que atendem requisitos genéricos e focar apenas nos trechos mais específicos da solução – a parte interessante.

O *framework* Spring provê uma estrutura básica que incentiva a modularização no processo de desenvolvimento de *software*. Além do próprio núcleo do *framework*, o Spring também oferece módulos muito populares, como o Spring Security para segurança, o Spring Data para persistência de dados e o Spring MVC (Model, View, Controller), que divide as responsabilidades de elementos de código entre modelo, interface de usuário e controladores. O *framework* Spring é muito utilizado pela comunidade Java, devido à sua modularização e pelo fato de que existe um certo nível de flexibilidade, escalabilidade e extensibilidade.

1.2 O que é o Spring Boot

Iniciando nossos estudos sobre a extensibilidade do Spring, dentre as mais diversas extensões que podemos encontrar, uma das mais populares, se não a mais popular, é o Spring Boot. Esta extensão do Spring tem como objetivo simplificar o processo de desenvolvimento de *software*. Isto inclui não apenas as tarefas de implementação, mas também as tarefas de configuração e implantação de *software*.

Ao utilizar o Spring Boot, desenvolvedores podem iniciar a implementação, a partir de uma versão padrão de projeto que tende a sater as necessidades básicas de configuração, diminuindo o trabalho ou até mesmo satisfazer a etapa de configuração. Por exemplo, se sua equipe iniciar o desenvolvimento de uma solução *web* para um tribunal, ao utilizar o Spring Boot, você pode utilizar a estrutura base disponibilizada e configurar apenas as necessidades específicas, adicionando, neste caso, por exemplo, o Spring Web. Outra importante característica do Spring Boot é a autoconfiguração. Uma vez configurado, o *framework* irá buscar automaticamente as bibliotecas presentes na configuração, reduzindo a necessidade de configurações manuais.

Além disso, o Spring Boot possui servidores internos leves, como o Tomcat, que simplificam, consideravelmente, a execução de aplicativos Java. Como se trata de uma extensão do Spring, consequentemente, é compatível com os módulos populares do *framework* e suporta as mais diversas arquiteturas, como a arquitetura microsserviços, que veremos nas próximas trilhas de aprendizagem do curso Java Avançado.

1.3 O que é o Spring Cloud

Em cenários mais complexos, onde um sistema não se encontra, necessariamente, limitado a uma única máquina, mas está, na verdade, distribuído através de uma rede como a Internet, o desenvolvimento em Java, utilizando Spring, precisa levar em consideração uma arquitetura com conceitos mais avançados. Alguns desses conceitos incluem: balanceamento de carga, que visa distribuir a demanda de recursos de maneira a atender as requisições ao sistema, sem sobrecarregar componentes internos específicos; tolerância a falhas, que busca manter o atendimento a uma requisição, mesmo quando erros ocorrem; e assim por diante. Neste contexto, o Spring Cloud é uma ferramenta comumente encontrada no ecossistema de desenvolvimento em Java, utilizando Spring.

O Spring Cloud conta com uma configuração simplificada, com opções centralizadas e distribuídas, o que permite o acesso a configurações de modo dinâmico e evita a reinicialização do sistema. Além disso, o Spring Cloud provê opções de autenticação e autorização em sistemas distribuídos, como o popular protocolo de segurança OAuth2, que veremos futuramente em nossa capacitação.

Podemos dizer que a utilização do *framework* Spring, com integração do Spring Boot e Spring Cloud, é muito popular entre desenvolvedores de sistemas de *software* em Java, desde arquiteturas simples a arquiteturas complexas.

1.4 Ambientes Integrados de Desenvolvimento (IDE – *Integrated Development Environments*)

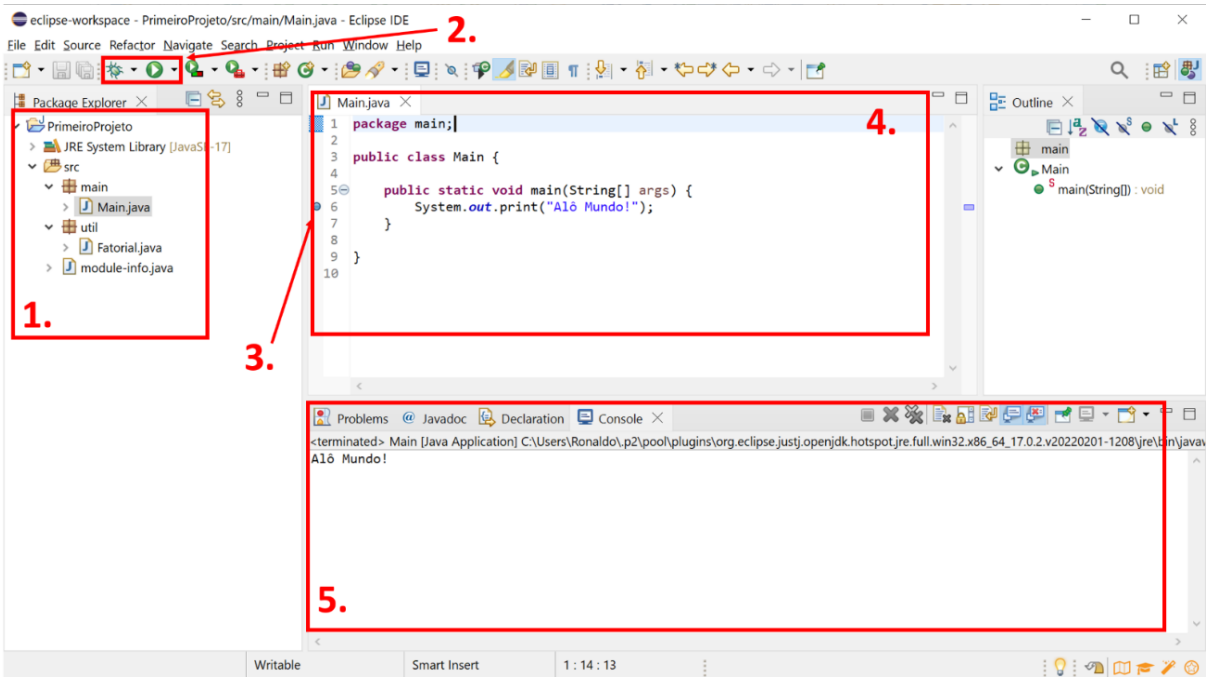
O uso de diversas bibliotecas disponíveis para as linguagens de programação e distintos *frameworks* para a construção de sistemas de *software* tendem a beneficiar o processo de desenvolvimento. Todavia, sem o auxílio de ferramentas apropriadas, a integração de múltiplos desses componentes, em uma arquitetura complexa, pode ser desafiadora. Este é um dos principais motivos pelos quais programadores optam pelo uso de um *Integrated Development Environment* (IDE), ou Ambiente Integrado de Desenvolvimento.

Um IDE pode ser descrito como um *software* assistente centralizador, onde *frameworks* como o Spring e demais bibliotecas de uma linguagem como Java são integrados para facilitar o processo de desenvolvimento de *software*. As principais funcionalidades, comumente encontradas em IDEs, são: visualização de projetos, assistência no controle de versão, suporte a linguagens de programação e assim por diante. Utilizaremos, em nossa capacitação, dois IDEs populares para desenvolvimento na linguagem de programação Java: o Eclipse IDE e o Visual Studio Code.

1.4.1 Eclipse IDE

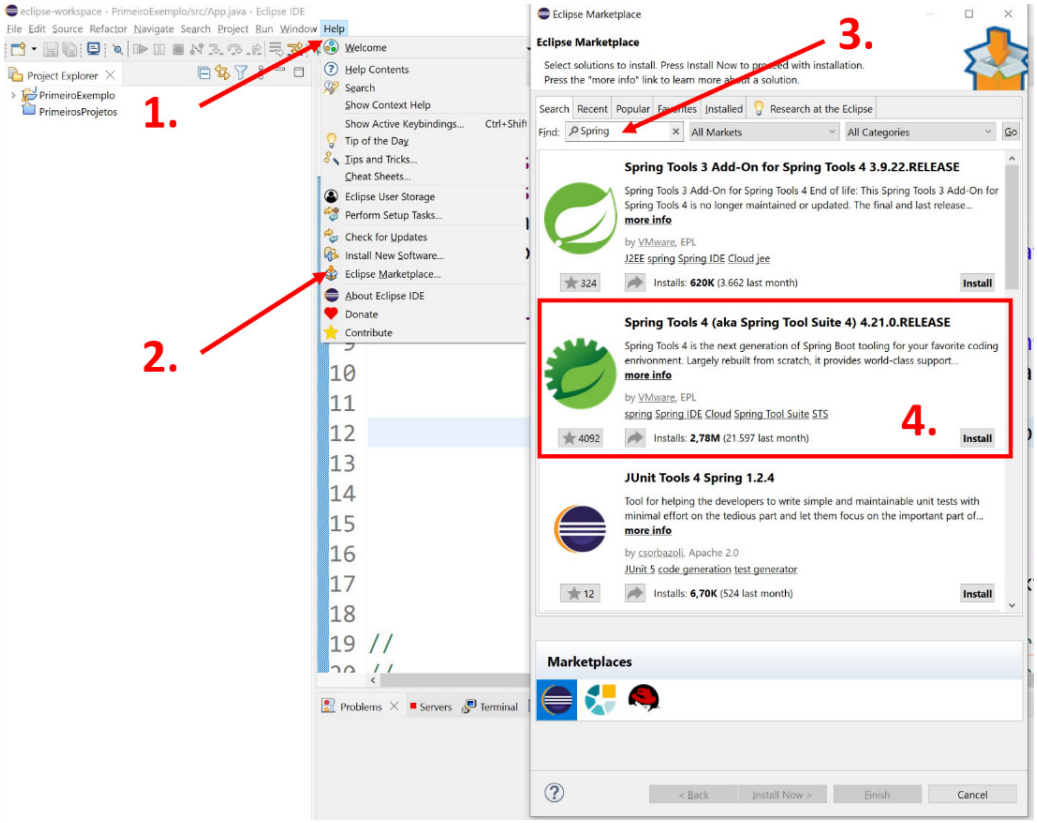
O Eclipse é um ambiente integrado de desenvolvimento bem estabelecido, criado em 2001 pela IBM e hoje mantido pela Eclipse Foundation. Este IDE ficou muito conhecido pelo suporte à linguagem Java e ainda é uma das opções mais utilizadas como ambiente de desenvolvimento.

Os principais utilitários básicos do Eclipse podem ser encontrados na imagem a seguir: (1) explorador de pacotes; (2) depurador; (3) ponto de parada; (4) editor de texto; e (5) console. Note que, no caderno de atividades, você encontrará o passo a passo para instalação, configuração e criação de um projeto no Eclipse.



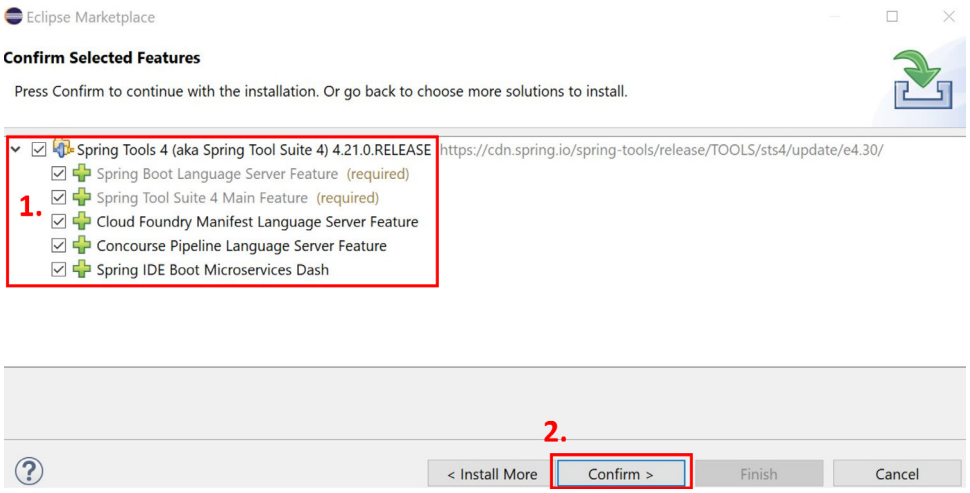
Elaborada pelo autor, 2023.

Em sua versão mais avançada, o Eclipse conta com plugins que contêm as ferramentas Spring que iremos utilizar nas nossas trilhas de aprendizagem.
Para configurar o Spring Tools em seu Eclipse, siga o passo a passo ilustrado pela imagem a seguir:



Elaborada pelo autor, 2024.

1. Acesse o item do menu “Help”;
2. Selecione a opção “Eclipse Marketplace”;
3. Procure por “Spring” no campo de busca;
4. Identifique o plugin “Spring Tools 4” e clique na opção de instalar;
5. Na próxima página, selecione todas as ferramentas do “Spring Tools 4” e confirme, conforme itens 1 e 2 da imagem que segue, respectivamente.

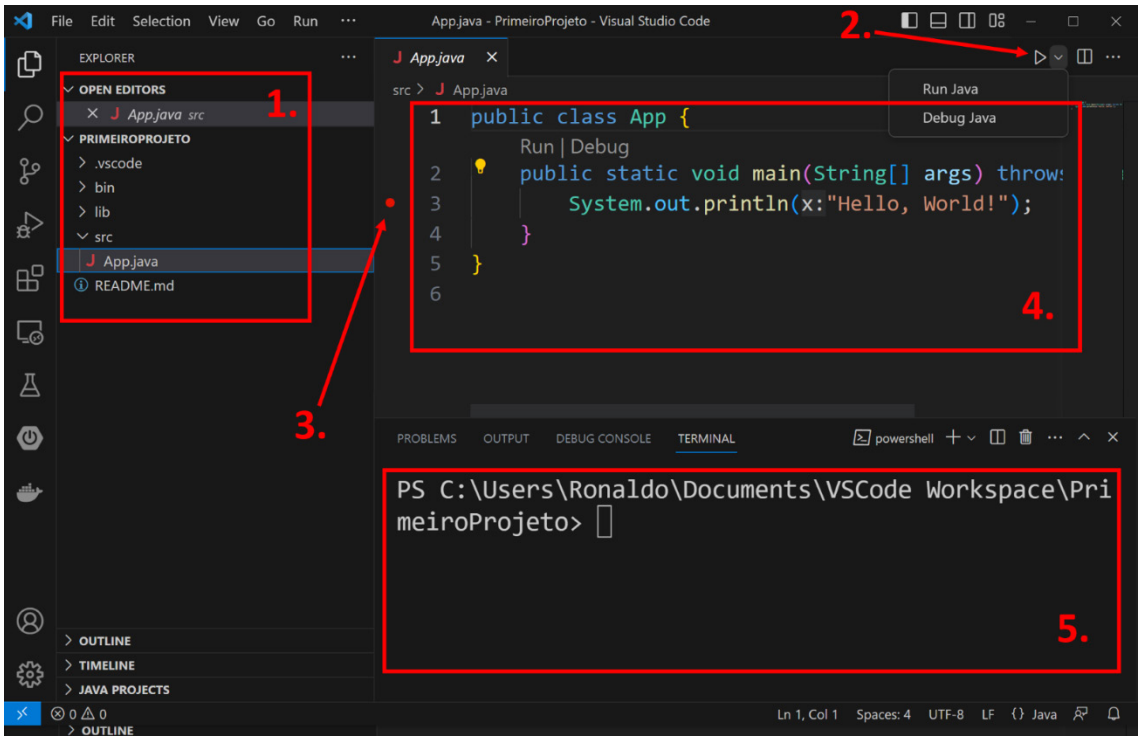


Elaborada pelo autor, 2024.

O processo de instalação e detecção de dependências ocorre automaticamente. Conforme a atual instalação do Eclipse IDE, é possível que você precise selecionar uma opção de atualização de sua instalação, devido a problemas de compatibilidade, para que todas as ferramentas Spring sejam instaladas. Ao final do processo, basta concordar com os termos e selecionar a opção de finalizar.

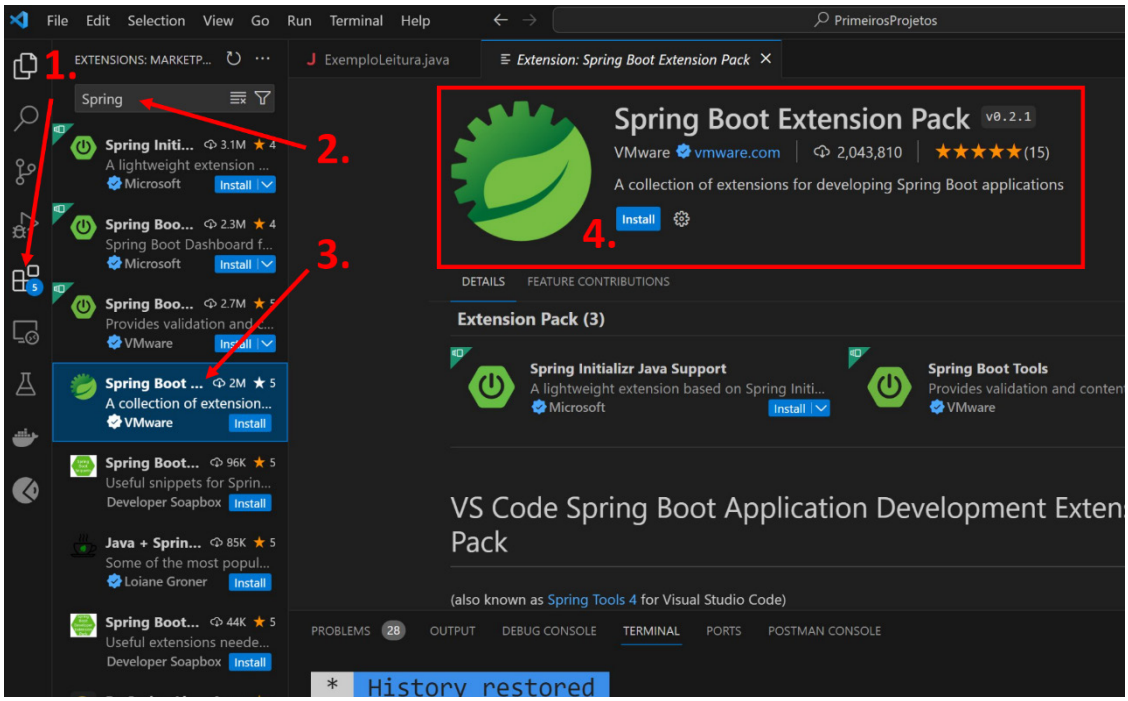
1.4.2 Visual Studio Code

Criado em 2015 pela Microsoft, o Visual Studio Code (VSCode) tem se tornado uma opção de ambiente de desenvolvimento cada vez mais popular. O VSCode possui o mesmo conjunto básico de utilitários que o Eclipse IDE, conforme mostra a seguinte imagem:



Elaborada pelo autor, 2023.

Note que as numerações e os utilitários são os mesmos encontrados no Eclipse IDE. Para fazer uso de recursos mais avançados, de maneira análoga aos plugins do Eclipse, o VSCode conta com extensões. O pacote de extensões que contém o Spring Tools pode ser instalado conforme o passo a passo ilustrado pela imagem que segue:



Elaborada pelo autor, 2024.

1. Selecione a opção de extensões do VSCode;
2. Procure por “Spring” no campo de busca;
3. Identifique a extensão “Spring Boot Extension Pack” na lista;
4. Clique na opção de instalação.

Ao seguir com o processo de instalação, seu ambiente de desenvolvimento estará configurado para a criação de projetos avançados em Java. Entretanto, antes de prosseguir com um projeto exemplo, precisamos entender o papel da JVM, no ambiente de desenvolvimento, e garantir que os requisitos mínimos estejam configurados: a instalação do Java. Criaremos nosso primeiro projeto em breve, na próxima seção desta trilha de aprendizagem.

1.5 Máquina Virtual do Java (JVM – *Java Virtual Machine*)

A Máquina Virtual Java, ou simplesmente JVM, é um componente essencial da estrutura de uma aplicação Java. Em arquiteturas mais complexas, como, por exemplo, a de um sistema de *software* judiciário que está interconectado com tribunais de todo o país, a criação de código em Java precisa ocorrer em apenas um momento e pode, então, ser executado em qualquer plataforma. Esta característica da JVM garante certo nível de portabilidade e segurança.

Outra característica importante é que a JVM permite que o desenvolvimento de código seja mais simplificado. Através do Garbage Collector, desenvolvedores não precisam gerenciar recursos de memória. Quando integrado com *frameworks* poderosos de desenvolvimento, como o Spring, a tarefa de implementação fica, consideravelmente, mais fácil e rápida, se comparada a outras linguagens de programação tradicionais.

Finalmente, em relação à eficiência, a técnica Just-in-Time de compilação da JVM traduz diretamente o código bytecode para código de máquina, implicando em uma melhoria de desempenho. Vamos agora seguir os passos de instalação da linguagem Java e demais configurações necessárias para iniciar nosso projeto exemplo.

1.6 Instalação do Java

As versões mais modernas de ambientes de desenvolvimento já trazem consigo um ambiente de execução Java. Todavia, vamos estudar quais são os principais requisitos para iniciar nosso projeto e entender os passos para realizar a instalação deles.

1.6.1 *Java Runtime Environment* (JRE)

O *Java Runtime Environment,* ou JRE, é usado para executar projetos Java em um sistema. Algumas vezes, a intenção de um desenvolvedor é apenas executar projetos Java. Neste caso, será suficiente para o ambiente contar com o JRE. Seguem os passos para instalação do JRE:

1. Acesse o *site* oficial:
   https://www.oracle.com/java/technologies/downloads/
2. Identifique a seção “JRE 8”;
3. Selecione a sua opção, de acordo com o sistema operacional;
4. Clique no *link* de *download* da versão de instalação (“Installer”).

Para o sistema operacional Windows, podemos realizar o *download* e a instalação da opção “jre-8u401-windows-x64.exe” e executá-la. Após o *download*, siga o passo a passo simples de instalação.

1.6.2 *Java SE Development Kit* (JDK)

Diferentemente do JRE, a edição padrão do Java, ou *Java Standard Edition* (JSE), não apenas executa projetos Java, mas também fornece bibliotecas de suporte ao desenvolvimento. Já o kit de desenvolvimento Java, ou *Java Development Kit* (JDK), além da edição padrão JSE, que

contém uma JRE, apresenta componentes adicionais, como um compilador. É recomendado o uso de um JDK, quando a intenção é construir novos projetos em Java. Para instalar o JDK, em seu sistema, siga os passos abaixo:

1. Acesse o *site* oficial:
   https://www.oracle.com/java/technologies/downloads/
2. Identifique a seção "Java 21"
   Alternativamente, caso deseje a edição padrão, identifique a seção "Java SE Development Kit".
3. Selecione a sua opção, de acordo com o sistema operacional;
4. Clique no *link* de *download* da versão de instalação ("Installer").

Para o sistema operacional Windows, podemos realizar o *download* da opção "x64 Installer" e executá-la. Após o *download*, siga o passo a passo de instalação. Para testar se a instalação foi bem-sucedida, abra um "prompt de comando" (para sistemas Windows) ou um "terminal" (para sistemas MAC ou Linux) e digite o comando "java -version". Se a versão do Java aparecer na tela de linha de comando, o Java foi instalado e estamos prontos para a criação de um projeto Spring. Se você já tinha uma versão anterior do Java, atualize sua variável de ambiente JAVA_HOME para o Java 21. Verifique também a variável de ambiente PATH, para que esteja apontando para a versão recentemente instalada.

2 CRIAÇÃO DE UM PROJETO SPRING E GERÊNCIA DE BUILD E INTEGRATION UTILIZANDO APACHE MAVEN

2.1 Criação de um projeto Spring

O *framework* Spring, em conjunto com a extensão Spring Boot, realiza o controle de fluxo de aplicações e conta com um servidor interno para facilitar o desenvolvimento do projeto e sua execução. Uma opção popular, como primeiro passo na criação de um projeto Spring Boot, é o *site* do Spring Initializr, o inicializador de projetos Spring (https://start.spring.io/), mas é possível também inicializar um projeto diretamente de seu IDE, através dos plugins ou extensões Spring Tools, que instalamos anteriormente.

Para que o projeto seja gerenciado automaticamente, iremos escolher a opção de projeto Maven. Desenvolvido pela Apache, o Maven é uma ferramenta de gerenciamento que organiza e constrói projetos e suas dependências, de forma automática, com base em um arquivo de Modelo de Objeto de Projeto, ou Project Object Model (POM), em formato XML. Antes de gerar o projeto, preencha os campos conforme a imagem a seguir:

Elaborada pelo autor, 2024.

Após realizar o *download*, faça a extração do projeto contido no arquivo zip, pois, futuramente, abriremos a pasta extraída no VSCode ou importaremos o projeto no Eclipse.

2.2 Instalação e configuração

Para fazer uso do arquivo gerado no passo anterior, precisamos ter o Maven instalado em nosso ambiente. Para realizar o procedimento, é suficiente instalar a extensão ou plugin, através da busca por “Maven for Java”. É possível, também, baixar um instalador e realizar o processo manualmente, por meio do *site* oficial (https://maven.apache.org/download.cgi). O comando mvn -v é uma boa opção para testar a instalação e verificar a versão atual. Abra o terminal, execute esse comando e fique atento(a) à impressão da versão atual.

O próximo passo seria a criação de um arquivo pom.xml, mas como utilizamos o Spring Initializr para a criação do projeto, um arquivo POM foi automaticamente gerado. Vamos abrir a pasta extraída no passo anterior, dentro do VSCode. Para conferir se tudo está configurado, abra o arquivo pom.xml e verifique se possui conteúdo. Em seguida, iremos testar a construção do projeto. Para isso, execute o comando mvn compile.

O comando de compilação irá gerar arquivos .class correspondentes. Como não utilizaremos esses arquivos diretamente, vamos executar o comando mvn package para gerar um arquivo executável, dentro do diretório alvo, ou target. Após baixar e empacotar todas as dependências, uma mensagem de sucesso indicará que o arquivo executável foi criado na pasta target. Para testar o arquivo, execute o comando java –jar target/projeto-0.0.1-SNAPSHOT.jar e note que o projeto está em execução. Para encerrar a execução no terminal, basta pressionar as teclas ctrl+c.

Por fim, o comando mvn install pode ser usado para compilar e empacotar, de uma única vez, o projeto, para que ele seja disponibilizado mais rapidamente e esteja pronto para uso.

2.3 Configuração de projeto e dependências

Durante a criação do projeto, escolhemos como dependência o Spring Web. Ao abrir o arquivo pom.xml, note que, dentro de <dependencies>, temos a dependência spring-boot-starter-web. Para adicionar outro projeto, basta incluir uma nova dependência neste arquivo pom.xml e executar o comando mvn compile ou mvn package.

No caderno de atividades, temos um projeto de aplicação e o uso de dependências adicionais. Logo, é de extrema importância, para a absorção desse conteúdo, a execução das atividades propostas no caderno. Por ora, vale ressaltar que o Maven irá gerenciar automaticamente o aspecto recursivo entre dependências. Isto é, você não precisará se preocupar se uma dependência de seu projeto possui suas próprias dependências e assim por diante.

3 GERÊNCIA E VERSIONAMENTO DE CÓDIGO-FONTE UTILIZANDO GIT

3.1 Git, Github e Gitlab

Quando abordamos tópicos avançados de desenvolvimento, a tendência é que o *software* em questão possua um elevado número de funcionalidades e/ou uma arquitetura complexa. Podemos definir, neste contexto, a premissa de que um *software* não é produzido individualmente, mas, sim, por uma equipe. Logo, tudo que é produzido pela equipe, em termos de código-fonte, deverá ser eventualmente incorporado em uma versão única do *software*.

A tarefa contínua e manual de unir todo o código produzido por todos os membros de uma equipe pode trazer problemas, especialmente em uma equipe grande, pois desenvolvedores podem introduzir erros de maneira inadvertida. Para resolver essa tarefa, com certo grau de automação, utilizamos Sistemas de Controle de Versão (SCV, ou VCS em inglês).

Git é um sistema de controle de versão, que auxilia um desenvolvedor no armazenamento de código-fonte produzido e no compartilhamento de código, com demais desenvolvedores.

O Git, como ferramenta, provê uma série de comandos – que veremos, em breve, nesta seção – para facilitar o gerenciamento dos arquivos de código-fonte, desde o armazenamento em uma máquina local, até o envio e recebimento de arquivos de um servidor remoto.

De uma maneira geral, o código pode ficar armazenado em quatro locais:

- Diretório de trabalho local: também chamado de Local Working Directory ou, simplesmente, WorkingDir, é representado por uma ou mais pastas, em seu sistema de arquivos local;
- Staging area: uma área temporária onde as mudanças de código ficam armazenadas, antes de serem persistidas;
- Repositório local: um componente de armazenamento do git, dentro da máquina local, onde ficam as mudanças de código;
- Repositório remoto: um servidor (muitas vezes, na nuvem) que armazena o código que ficará acessível para toda a equipe.

Em sua versão distribuída, um sistema de controle de versão conta com um servidor com repositórios remotos. Dois exemplos populares de plataformas que atendem essa necessidade

são o GitHub e o GitLab. Estas plataformas permitem que desenvolvedores criem repositórios remotos públicos e privados e conectem o código, sendo produzido por toda a equipe, de uma forma prática e em conjunto com o Git.

Essas plataformas contam também com características de rede social, disponibilizando funcionalidades adicionais, como a de seguir outros desenvolvedores, observar repositórios de outras equipes, participar de organizações etc. Para fazer um uso prático dessas ferramentas, vamos agora estudar como realizar a instalação do Git.

3.2. Instalação do Git

Para instalar o Git em sua máquina, siga os passos abaixo:

1. Acesse o *site* oficial:
   https://git-scm.com/
2. Identifique a área de *download*;
3. Selecione a sua opção, de acordo com o sistema operacional;
4. Clique no *link* de *download* para iniciar a instalação.

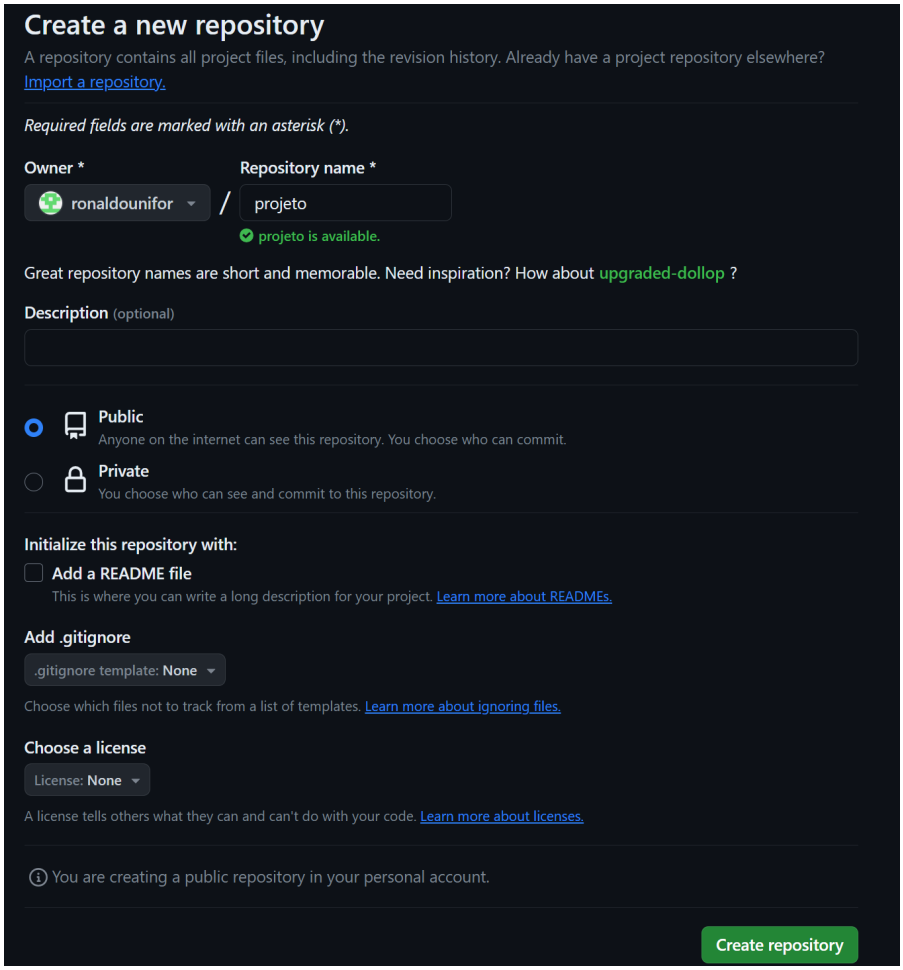
O processo de instalação é simples e pode ser feito com as opções por padrão. Todavia, caso as opções de instalação do "Git GUI Here" e "Git Bash Here" sejam oferecidas durante a instalação, é recomendada sua inclusão. Estas ferramentas permitem um acesso rápido a um terminal, pelo sistema de arquivos do Windows, mas não é necessário, pois podemos usar o terminal padrão, ou pelo IDE. Para continuar com a configuração do nosso ambiente, iremos agora realizar a conexão ao GitHub.

3.3 Conexão ao Github ou Gitlab

O primeiro passo para conectar nosso ambiente local a um repositório remoto no GitHub é a criação de uma conta gratuita, por meio dos passos a seguir:

1. Acesse o site oficial:
   https://github.com/
2. Seleciona a página de cadastro "Sign Up";
3. Selecione todos os campos do formulário de cadastro;
4. Confirme seu cadastro e crie sua conta.

Uma vez logado na plataforma, você verá a opção “New repository”, que pode ser utilizada para criar um repositório remoto. Você precisará indicar um nome para o repositório, selecionar a visibilidade pública ou privada e algumas outras opções adicionais. Por ora, vamos criar um repositório, conforme a seguinte imagem:



Elaborada pelo autor, 2024.

Um processo semelhante pode ser feito para a plataforma GitLab, mas, como precisamos apenas de um repositório no momento, utilizaremos o repositório que acabamos de criar no GitHub.

Vamos agora ver quais são os comandos básicos e avançados do Git, para realizar o gerenciamento local e a conexão com o repositório remoto.

3.4 Comandos básicos

Para fazer uso dos comandos básicos Git, crie uma pasta de trabalho, em seu sistema de arquivos local, e abra o terminal, dentro dessa pasta. Existe uma miríade de comandos combinados com seus parâmetros de execução e sua ordem de uso. Veremos aqui suas versões básicas e, no caderno de atividades, temos uma proposta de prática para esses comandos.

3.4.1 git init

Ao criar uma pasta em seu sistema de arquivos, isso não significa que a pasta ou os arquivos dentro dela já estão sendo controlados pelo Git. O comando git init irá inicializar a pasta, como um diretório de trabalho local. Este é o primeiro comando que iremos executar para fazer o controle de versões do nosso projeto.

3.4.2 git add

O comando git add permite que arquivos específicos sejam adicionados ao staging area. Em outras palavras, você está indicando ao Git que você quer que esses arquivos sejam adicionados ao controle de versões. Crie um arquivo de texto chamado README.txt e escreva algum conteúdo nele. Em seguida, execute, no terminal, o comando git add README.txt, para adicionar esse arquivo, ou git add *, para adicionar todos os arquivos, que, neste caso, é apenas um.

3.4.3 git rm

Caso um desenvolvedor adicione um ou mais arquivos ao staging area e se arrependa, é possível remover esses arquivos do controle de versão, através do comando git rm. Faça o teste, executando o comando git rm README.txt, para remover o arquivo adicionado anteriormente. Para remover todos os arquivos de um diretório, basta digitar git rm –r diretorio/ e substituir “diretorio”, no comando, pelo nome da pasta com os arquivos.

3.4.4 git commit

Depois de enviar um arquivo para o staging area através do comando git add, isso não significa que o arquivo foi enviado para seu repositório local. Para manter esses arquivos no repositório e movê-los da área temporária, use o comando git commit. Este comando deve ser seguido

de uma mensagem, pois precisamos identificar o que foi feito a todo instante, para fins de rastreabilidade e mudança de versões quando necessário. Execute o seguinte comando como exemplo: git commit -m “Criação do primeiro arquivo de texto”.

3.4.5 git status

A qualquer instante, desenvolvedores podem ver o estado do sistema de controle de versões, através do comando git status. Ao executá-lo, será impresso no terminal um estado atual dos arquivos, indicando o que precisa ser adicionado, o que está pronto para commit e assim por diante.

3.5 Comandos avançados

Os comandos básicos git representam uma boa parte da interação entre desenvolvedores e sistema de controle de versão. Todavia, precisamos ainda aprender a utilizar o conjunto de comandos avançados, parte fundamental no controle de versão de um projeto de *software*.

3.5.1 git clone

O comando git clone é utilizado para copiar um repositório remoto, como no GitHub ou GitLab, para o seu repositório local. Por exemplo, após realizar a inicialização de um diretório (git init), podemos executar o comando git clone, para receber todos os arquivos de um repositório remoto – git clone https://github.com/ronaldounifor/projeto.git – substituindo o endereço aqui demonstrado pelo endereço do seu repositório remoto.

3.5.2 git pull

Para receber as mudanças de código feitas por outros desenvolvedores do projeto no repositório remoto, em seu repositório local, use o comando git pull. Não é recomendado usar o comando git clone toda vez que uma nova mudança é gerada, pois grande parte dos arquivos que já foram copiados permanece igual. Sempre que você iniciar um dia de trabalho, e antes de enviar suas mudanças, é recomendado o uso do comando git pull, para garantir que você está trabalhando na versão mais recente do código compartilhado.

3.5.3 git push

Desenvolvedores podem enviar mudanças de código-fonte de seus repositórios locais para o repositório remoto por meio do comando git push. Sempre que terminar um dia de trabalho,

é recomendado usar o comando git push para enviar suas mudanças. Note que, se uma funcionalidade estiver com bugs ou incompleta, talvez seja melhor manter o código em seu repositório local, sem enviá-lo para o repositório remoto.

3.5.4 git fetch

Para saber quais são as mudanças de código-fonte mais recentes no repositório remoto, sem, necessariamente, copiá-las para seu repositório local, podemos usar o comando git fetch.

3.5.5 git log

O comando git log permite que desenvolvedores visualizem os últimos commits realizados no repositório, incluindo o autor da última modificação, data e as mensagens de commit. Isto pode ser bem útil para localizar diferentes versões.

3.5.6 git tag

Desenvolvedores podem utilizar rótulos, para marcar versões específicas do código no repositório, para melhorar o controle e rastreamento. Esses rótulos podem ser gerenciados pelo comando git tag, como, por exemplo: git tag versao-inicial.

3.6 Trabalhando com branches

Conforme o nome implica, branches – que, em português, significa "ramos" – são ramificações, ou diferentes linhas de desenvolvimento. É possível trabalhar apenas com uma branch, mas considere o seguinte caso: imagine que um *software* está funcional e sua equipe não quer mexer no código existente, com receio de introduzir bugs. Caso surja a necessidade de trabalhar nesse código, é possível criar uma nova linha de desenvolvimento paralela, uma branch nova, e mexer apenas no trecho respectivo. Se algum problema ocorrer, a branch principal não será afetada, apenas a branch nova. Na prática, desenvolvedores fazem uso de branches para desenvolvimento paralelo, correção de bugs, implementação de novas funcionalidades etc.

3.6.1 git branch

Podemos usar o comando git branch, para listar todas as branches existentes. Além disso, podemos criar uma nova branch, concatenando um nome para ela – git branch nova-funcionalidade – e, para remover uma branch, basta adicionar um parâmetro -d no comando – git branch -d nova- funcionalidade.

3.6.2 git checkout

O comando git checkout pode ser usado para mudar de uma branch para outra. Por vezes, é necessário mudar nossa atenção, de uma nova funcionalidade para uma correção de bugs, por exemplo. Para isso, basta digitar o comando e o nome da branch: git checkout correcao-bugs.

3.6.3 git merge

Quando um desenvolvedor deseja unir o código de uma branch ao código da branch principal, o comando git merge pode ser usado. Se o desenvolvedor estiver trabalhando na branch principal e quiser unir o código da branch denominada nova-funcionalidade, basta executar git merge nova- funcionalidade.

3.6.4 git stash e git pop

Se um desenvolvedor precisa mudar de branch, mas está no meio de um trabalho e não quer se comprometer ou arriscar um commit, este desenvolvedor pode executar o comando git stash para salvar, temporariamente, as alterações não commitadas e mudar de branch, com a segurança de não perder seu código. Por exemplo, git stash save “trabalho salvo”. Para aplicar novamente o código salvo temporariamente e, ao mesmo tempo, remover os arquivos salvos temporários, o comando git stash pop pode ser usado.

3.6.5 git reset e git clean

Por vezes, desejamos nos livrar de um código produzido, seja por conta de bugs ou por mudanças de requisitos. O comando git reset é usado para desfazer um ou mais commits no repositório local, reverter alterações no staging area ou até mesmo se livrar de mudanças no diretório de arquivos local. Vale ressaltar que esse comando deve ser usado com muito cuidado, pois arquivos de código-fonte podem ser perdidos. De maneira semelhante, o comando git clean pode ser usado para se livrar de código. Todavia, o código a ser removido será o código que não foi incluso no sistema de controle de versão. Ou seja, estão apenas no diretório de trabalho local e não na staging area ou repositório local. É aconselhado exercer o mesmo cuidado com este comando.

3.6.6 git cherry pick

O comando git cherry pick, que no inglês significa “escolher especificamente” e, em português, “escolher a dedo”, pode ser usado para aplicar um commit específico de uma branch em outra branch. Por exemplo, se você estiver na branch nova-funcionalidade e quiser aplicar o commit com hash “2222” da branch correcao-bugs, basta executar: git cherry-pick 2222.

3.6.7 git rebase

Para aplicar todos os commits de uma branch em outra, basta usar o comando git rebase. Por exemplo, se você estiver trabalhando na branch principal e quiser aplicar todos os comandos da branch nova-funcionalidade, você pode executar git rebase nova-funcionalidade, para aplicar todos os commits dessa branch na branch principal.

3.7 Trabalhando com o git flow

O Git Flow é um modelo desenvolvido, especificamente, para controle de branches, utilizando um conjunto de convenções. Dentre essas convenções, duas branches principais podem ser encontradas: a branch “master”, onde fica o código pronto e oficializado em sua versão estável; e a branch “develop”, onde fica o código atualmente sendo desenvolvido, com funcionalidades incompletas. Eventualmente, o que é produzido na branch develop será enviado para a branch master, quando estiver pronto. Além das duas branches principais, temos também outras, que são auxiliares, denominadas feature, release e hotfix, para as funcionalidades liberação e correções, respectivamente.

3.7.1 git flow init

Para iniciar o Git Flow, um desenvolvedor pode fazer uso do comando git flow init, que irá criar as branches principais. Ao final do ciclo de desenvolvimento de um projeto, o desenvolvedor pode encerrar o uso do Git Flow, por meio do comando git flow finish.

3.7.2 git flow feature

Quando desejar iniciar uma nova feature, isto é, iniciar o desenvolvimento de uma nova funcionalidade, o desenvolvedor pode usar o comando git flow feature start nova-funcionalidade. Para encerrar uma feature, basta trocar o start pelo finish: git flow feature finish nova-funcionalidade.

3.7.3 git flow release

Para iniciar um novo release, ou liberação de código-fonte, o comando git flow release é utilizado. De maneira análoga ao comando git flow feature, podemos fazer uso do start e finish, para iniciar ou encerrar uma release, respectivamente. Por convenção, devemos incluir uma das releases, por exemplo: git flow release start versao-0.1.0.

3.7.4 git flow hotfix

Por fim, nosso último comando é o git flow hotfix, que é utilizado para criação de uma branch, para ajustes ou correção de *bugs* com urgência. Para iniciar um hotfix, usamos o comando git flow hotfix start correcao-bugs e, para encerrar, usamos o comando git flow hotfix finish correcao-bugs.

Aproveite este momento de conclusão dessa trilha de aprendizagem, para realizar todas as práticas propostas no caderno de atividades. Essas tarefas são essenciais para a absorção desse conteúdo. Na próxima trilha de aprendizagem, veremos importantes conceitos avançados em aplicações Java.

REFERÊNCIAS

BLOCH, J. **Effective Java**. 3ª edição. Editora Addison-Wesley Professional, 2017.

CHACON, S.; STRAUB, B. **Pro Git**. 2ª edição. Editora Apress, 2014.

COSMINA, I.; SCHAEFER, C.; HARROP, R.; HO, C. **Pro Spring 5**: An In-Depth Guide to the Spring Framework and Its Tools. 5ª edição. Editora Apress, 2017.

LONG, J.; BASTANI, K. **Cloud Native Java**: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry. 1ª edição. Editora O'Reilly Media, 2017.

O'BRIEN, T.; ZYL, J. V.; FOX, B.; CASEY, J. **Maven**: The Complete Reference. 1ª edição. Editora Sonatype, 2011.

PONUTHORAI, P.; LOELIGER, J. **Version Control with Git**: Powerful Tools and Techniques for Collaborative Software Development. 3ª edição. Editora O'Reilly Media, 2022.

SCHILDT, H. **Java**: The Complete Reference. 11ª edição. Editora McGraw-Hill Education, 2019.

VARANASI, B. **Introducing Maven**: A Build Tool for Today's Java Developers. 2ª edição. Editora Apress, 2019.

WALLS, C. **Spring in Action**. 5ª edição. Editora Manning Publications, 2018.