PERCURSO DE APRENDIZAGEM 1

Universidade de Fortaleza | EAD

AMBIENTES DE DESENVOLVIMENTO DE SOFTWARE

CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO DE SOFTWARE

Este trabalho está licenciado com uma Licença Creative Commons
Atribuição-NãoComercial-SemDerivações 4.0 Internacional.

Sumário

1. Plataformas de desenvolvimento de software: web, mobile, cloud e outros

2. Linguagens de programação: tipagens e paradigmas

3. Ambiente integrado de Desenvolvimento: Instalação e configuração

4. Utilitários do ambiente de desenvolvimento: Depuração, console e estrutura de projetos

Sumário clicável

Voltar ao sumário

Seja bem-vindo (a) ao componente curricular Ambientes de Desenvolvimento de Software. Aqui, iremos estudar em detalhes as principais atividades e tarefas presentes no dia a dia de um desenvolvedor a fim de debater quais são as maneiras corretas e apropriadas de se utilizar as tecnologias disponíveis. Para isso, começaremos explicando os conceitos básicos de um ambiente de desenvolvimento de software e seus detalhes. Em seguida, visitaremos os componentes essenciais e suas interações para a construção de um software. Este componente curricular irá caminhar pelos principais processos existentes através de exemplos e resolução de exercícios, para que os fundamentos de desenvolvimento de software possam ser absorvidos de forma compreensiva. É importante ter uma visão geral dos assuntos abordados nos quatro percursos de aprendizagem deste componente, mas também é necessária uma visão prática e aplicável para diversos cenários de trabalho, então se prepare para acompanhar o material de estudo e assistir e participar das aulas. Bons estudos!

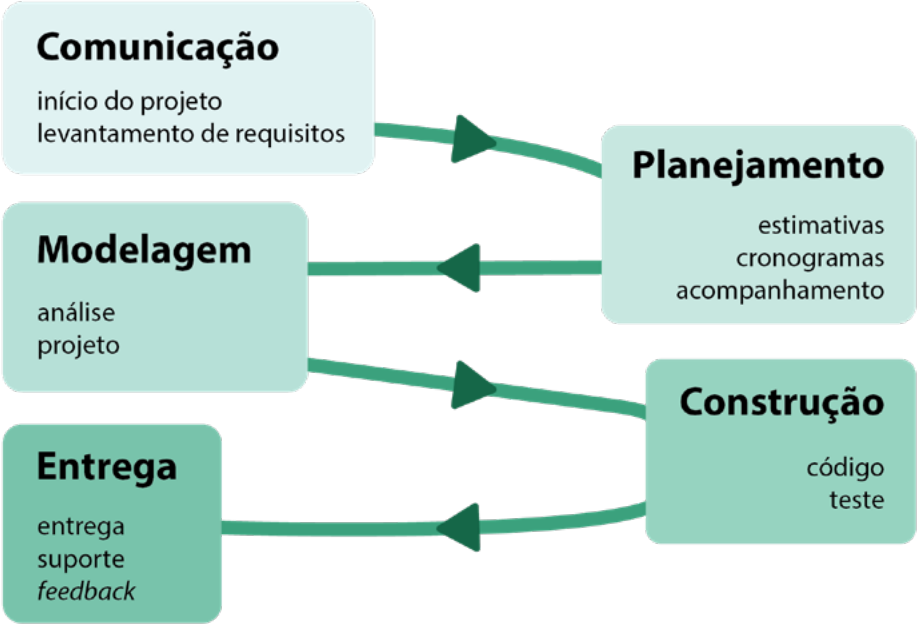
Olá

1. Plataformas de desenvolvimento de software: web, mobile, cloud e outros

O desenvolvimento de software pode seguir variadas metodologias, mas podemos dizer que as duas principais são: a metodologia sequencial, também conhecida como cascata; e a metodologia iterativa e incremental, também conhecida como ágil.

Podemos definir uma metodologia cascata como como um modelo sequencial e sistemático para o desenvolvimento de um software, no qual existe um planejamento ordenado das disciplinas da engenharia de software, de forma que uma disciplina não começa enquanto a anterior não estiver concluída. A primeira seria a especificação dos requisitos do cliente, seguida pelo planejamento, pela modelagem, pela construção e pela entrega (PRESSMAN, 2021).

Figura 1 - Desenvolvimento de um software



Fonte: Adaptado de (PRESSMAN, 2021).

Esta metodologia é uma das mais antigas e continua sendo utilizada até hoje. Todavia, por não se adaptar bem às mudanças que surgem ao longo de um projeto, seu uso vem sendo reduzido. Em vez disso, empresas estão optando cava dez mais por metodologias ágeis, nas quais as mudanças já são previstas pelo processo de desenvolvimento. Alguns analistas afirmam que esta metodologia surgiu em resposta ao cenário tecnológico, que se busca priorizar agilidade à qualidade (MASCHIETTO, 2022).

Voltar ao sumário

Em uma abordagem ágil, os requisitos não são levantados por completo em um único momento, apenas parte deles. Em seguida, planeja-se a respectiva parte do projeto em cada iteração, o que permite que mudanças sejam mais facilmente incorporadas. Por vezes, as necessidades do cliente mudam depois que o projeto foi concluído (SOMMERVILLE, 2011).

Existem outras metodologias para o desenvolvimento de software e, de uma forma geral, podemos dizer que a maior parte das empresas tendem a eleger uma estratégia de desenvolvimento que melhor se encaixa a sua realidade. Em seguida, a empresa aplica esta estratégia em todos os seus projetos. Entretanto, note que uma empresa pode utilizar mais de uma estratégia, a depender do contexto de cada projeto. Não iremos estudar o domínio de negócio neste componente curricular, pois iremos nos preocupar em entender o básico para focar nas atividades de desenvolvimento em si.

Atualmente, é comum que o cliente e/ou principais usuários participem do processo de desenvolvimento em momentos-chave, característica de uma metodologia ágil de desenvolvimento. As necessidades são capturadas e então podemos pensar em como elaborar e construir o software em consideração.

Em geral, podemos identificar, em alto nível, dois importantes processos para a entrega de um software em uma versão completa: a análise e o desenvolvimento do sistema. É importante ressaltar que não é coincidência este ser o nome do curso - Análise e Desenvolvimento de Sistemas (ADS).

De uma forma resumida, podemos dizer que, na análise, iremos estudar as necessidades de um cliente e de usuários para modelar uma solução. Já o desenvolvimento refere-se à implementação ou concretização da solução modelada.

Note que não iremos abordar em detalhes todos os assuntos presentes na análise e no desenvolvimento de sistemas, pois não é viável estudar o assunto de um curso em um único componente curricular. Todavia, precisaremos visitar muitos de seus conceitos básicos enquanto avançamos neste componente. Para iniciar nossa introdução ao mundo do software, vamos identificar os principais domínios de aplicação de software (PRESSMAN, 2021):

1. Software de sistema: um ou mais programas desenvolvidos para atender às necessidades de outro programa (por exemplo, compiladores, editores, componentes de um sistema operacional e assim por diante).

Voltar ao sumário

2. Software de aplicação: operam em nível de negócio e buscam auxiliar no processo de tomadas de decisões administrativas ou facilitar as operações comerciais (exemplo, programas independentes que processam dados comerciais ou técnicos).
3. Software de engenharia/científico: voltados para uma atividade específica de uma área de estudo (por exemplo, programas de "cálculo em massa" que abrangem astronomia, vulcanologia, análise genética e meteorologia, entre outros).
4. Software embarcado: projetados para atuar dentro de um sistema, a fim de realizar um limitado conjunto de tarefas (exemplo, painel com funções de controle em um refrigerador).
5. Software para linha de produtos: voltados especificamente para lidar com um mercado determinado e/ou consumidores de massa (exemplo, controle de inventário).
6. Software de aplicações Web/aplicativos móveis: buscam atender requisições de clientes e gerar respostas em uma ampla rede, para as mais variadas necessidades (por exemplo, loja virtual, serviços de computação em nuvem, aplicativos em dispositivos móveis e assim por diante).
7. Software de inteligência artificial: utilizam técnicas de inferência e construção de modelos matemáticos para solucionar problemas complexos (exemplos incluem jogos e aprendizado de máquina).

É interessante conhecer os variados domínios onde softwares são utilizados, mas em geral nos especializamos em um (ou mais) deles para nos profissionalizar. Neste componente curricular, iremos focar primeiramente os softwares de aplicações web e aplicativos móveis, estudando as plataformas dessas aplicações e outras plataformas populares, bem como a comunicação entre cliente e servidor e como as requisições são atendidas. Para isso, iremos estudar a comunicação dessas entidades na rede mais popular para esse tipo de aplicação, a Internet.

Importante!

Quando falamos da Internet, estamos nos referindo ao uso de poderosos protocolos que garantem um serviço rápido e seguro, como o HTTP (do inglês, *Hyper Text Transfer Protocol*), ou Protocolo de Transferência Hiper Texto.

Voltar ao sumário

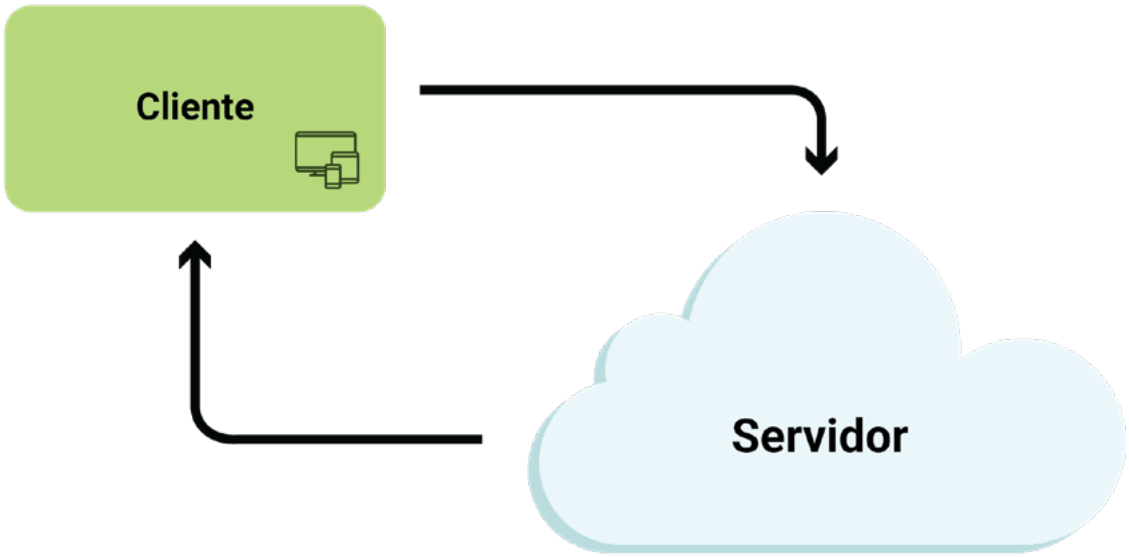
Ao digitar um endereço HTTP na barra de URL de um navegador, você gera uma requisição a um servidor. Neste caso, você é o cliente e o navegador web lhe auxilia na geração dessa requisição. Chamamos o agrupamento de componentes e interações presentes nessa comunicação de uma arquitetura cliente-servidor.

Importante!

Uma arquitetura cliente-servidor pode ser considerada a mais importante dentre outros modelos de comunicação, pois representa um modelo tradicional o qual é amplamente utilizado (MONTEIRO, 2020).

Em uma arquitetura cliente-servidor, conforme diz o nome, as duas principais entidades são: cliente e servidor. Um cliente faz requisições e o servidor atende essas requisições. De uma maneira simplificada, podemos dizer que, neste modelo arquitetural, os recursos estão centralizados nos servidores. Estes recursos podem ser estáticos ou dinâmicos e são enviados a um ou mais clientes mediante requisição.

Figura 2 - Modelo arquitetural



Fonte: Adaptado de (MONTEIRO, 2020).

Recursos estáticos são recursos que não mudam em relação a sua forma, ou seja, possuem uma estrutura final bem definida.

Voltar ao sumário

Um exemplo seria uma página de um blog contendo uma matéria textual.

Já recursos dinâmicos mudam em tempo de execução para se adequar a vários critérios, como de onde a requisição vem, o momento da execução e assim por diante.

Por exemplo, uma página da web pode ser apresentada em inglês nos Estados Unidos, mas ao detectar que o cliente está no Brasil, o servidor adapta o recurso de forma dinâmica traduzindo a página para português.

Mais pra frente, iremos estudar em detalhes os componentes de uma arquitetura e como escolher as tecnologias corretas para nosso contexto. Todavia, como desenvolvedores, é essencial entendermos onde um software é produzido.

Damos o nome de **plataforma de desenvolvimento** aos ambientes condicionados que auxiliam um desenvolvedor na construção de um software, através de funcionalidades como: suporte a linguagens de programação e seus detalhes (tipagens e paradigmas), depuração, console, estrutura de projetos e assim por diante. Veremos em breve o que é um ambiente de desenvolvimento (também conhecido como IDE) em detalhes, mas, por hora, vamos seguir a seguinte definição (DE OLIVEIRA, 2019):

IDE, do inglês integrated development environment, ou ambiente de desenvolvimento integrado, é um software voltado para o desenvolvimento de programas ou aplicativos. O desenvolvedor encontra nos IDEs todas as funções necessárias para o desenvolvimento, desde programas de computador a aplicativos mobile, assim como recursos que possibilitam a criação de códigos com uma incidência menor de erros.

Vários aplicativos populares possuem mais de uma versão. Isto é algo que está se tornando cada vez mais comum, talvez pelo fato de que equipamentos e dispositivos estejam cada vez mais acessíveis. Vamos tomar por exemplo a aplicação Discord, um software de conversa de texto e áudio organizado em servidores e canais. Este software possui uma versão desktop, uma versão mobile e uma versão web. Cada versão foi construída em uma plataforma distinta e apropriada para seus fins. A seguir, visitamos as principais plataformas existentes.

Voltar ao sumário

Desktop

Já instalou algum software em sua máquina e ficou apertando o botão "Próximo" ("Next") várias vezes? Tudo indica que este software foi construído em uma plataforma desktop.

Estas plataformas têm como objetivo auxiliar o desenvolvedor na construção de aplicações que atuarão em um computador, não necessariamente fazendo uso da Internet. Apesar disso, a maioria toma como premissa uma existente conexão com a Internet.

Para o exemplo do Discord, podemos instalar sua versão desktop. Esta versão é mais rápida, pois muitos componentes da aplicação estão na máquina do cliente. Todavia, é necessário que o cliente separe um tempo para realizar a instalação e um espaço suficiente em disco deve ser reservado para a instalação. Outras aplicações Desktop populares incluem: Microsoft Excel, Adobe Reader, Skype etc.

Sabemos o que são aplicações desktop, mas o que compõe uma plataforma destkop? Como utilizaremos Java neste componente curricular, um exemplo popular de plataforma de desenvolvimento desktop para esta linguagem conta com o uso de Java Swing para a criação de interfaces gráficas desktop através do uso do Eclipse como ambiente de desenvolvimento (uma IDE - veremos em detalhes no circuito 3).

Algumas classes disponíveis no Java permitem o desenvolvimento de interfaces gráficas, como a AWT (Abstract Window Toolkit). Com o avanço de tecnologias, incluindo a própria linguagem de desenvolvimento Java, uma versão mais popular para criação de interfaces gráficas foi desenvolvida a partir da AWT – o Swing. Dessa forma, podemos dizer que Swing é uma extensão das classes da AWT, possuindo melhor aparência, melhor tratamento de eventos, recursos estendidos, entre outros (FURGERI, 2012). O agrupamento de classes do Swing possibilita a construção de interfaces gráficas poderosas e ainda assim fáceis de usar através da criação de componentes visuais, como botões, campos de texto, painéis de rolagem, caixas de seleção, árvores e tabelas e assim por diante (SCHILDT, 2015).

É comum encontrarmos em uma plataforma de desenvolvimento desktop tecnologias de auxílio à manipulação de arquivos, gerenciamento de banco de dados, entre outras. Dependendo do sistema operacional onde o software será instalado, esta lista de tecnologias pode variar drasticamente.

Importante!

Uma vantagem de se utilizar Java é que a linguagem é multiplataforma, no sentido de que ela irá funcionar independentemente do sistema operacional em consideração.

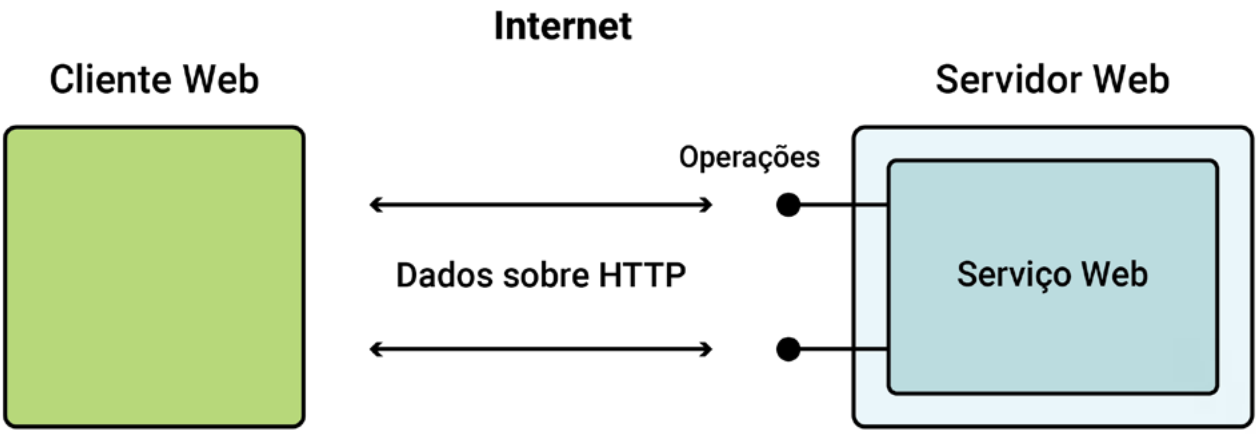
Voltar ao sumário

Web

Mais popular que a desktop, a plataforma web foi pensada para ser acessada por computadores, tablets, celulares e demais dispositivos. A tela desenvolvida se ajusta ao aparelho e permite que o usuário acesse às funcionalidades web sem causar problemas em sua aparência. A forma mais tradicional de se consumir um serviço web é visitar um site fazendo uso de um navegador, como o Google Chrome ou o Safari. Para o exemplo do Discord, uma vantagem de sua versão web quando comparada a sua versão desktop é que não é necessário fazer uma instalação. Outros exemplos de aplicações desenvolvidas em uma plataforma web incluem lojas on-line, serviços de stream ao vivo ou sob demanda, ambiente de estudo (como o AVA) etc.

Atualmente, é muito comum que as plataformas de desenvolvimento web contarem com componentes para auxiliar na construção de Web Services, conjuntos de programas acessíveis por meio da Internet. Esses programas abstraem o cliente da aplicação como um todo, permitindo que este acesse apenas os recursos de seu interesse. Web Services podem realizar um simples processo como disponibilização de um recurso estático, como também complexas transações comerciais ou industriais, por exemplo, um processo de compra de produtos (FERREIRA, 2021). Podemos dizer que todo Web Service é um componente para disponibilizar uma API (Application Programming Interface) e pode ser acessado por meio do HTTP, como mostra a figura abaixo (MACHADO, 2016).

Figura 3 – Cliente Web/Servidor Web



Fonte: Adaptada de (MACHADO, 2016).

Em uma plataforma web, os principais componentes envolvem um navegador, um servidor web, um servidor de aplicação e um servidor de dados. Todos esses componentes estão presentes em uma plataforma de desenvolvimento, mas, devido à complexidade, serão estudados em detalhes apenas nos últimos materiais deste componente curricular.

Voltar ao sumário

Mobile

Plataformas de desenvolvimento mobile são utilizadas para a construção de aplicativos projetados para uso em dispositivos móveis, disponíveis em lojas como App Store ou Play Store. Uma vez instalado em um celular, um aplicativo habilita um conjunto de funcionalidades agora presentes no dispositivo.

Importante!

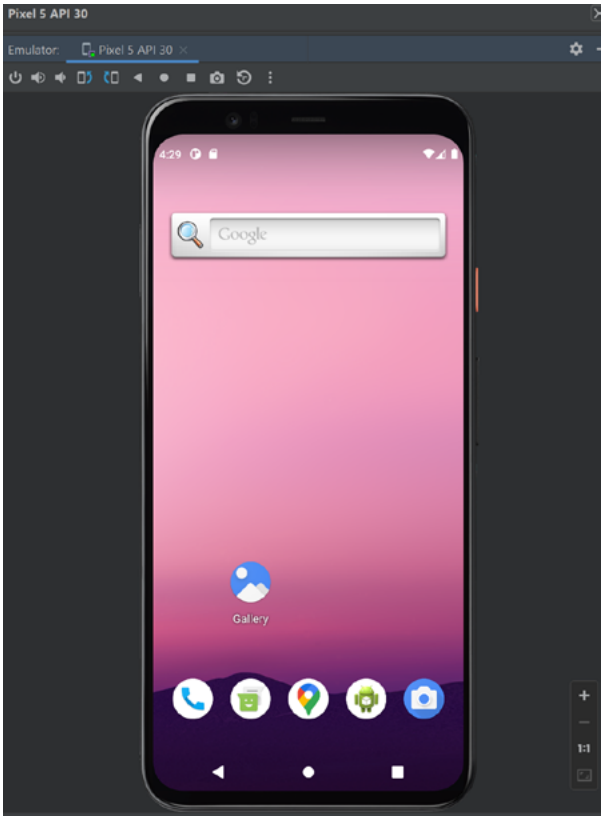
Uma grande vantagem de aplicativos mobile é a capacidade de usar os recursos do dispositivo, então uma plataforma mobile se torna uma ótima opção quando o software em desenvolvimento é projetado para ser utilizado por celulares.

Uma das mais populares IDEs mobile é o Android Studio. Desenvolvido pela própria Android e lançado em 2013, o Android Studio teve como base a popular IDE Intellij (que utilizaremos em breve) e foi projetado para a construção de aplicativos em linguagem Java - o que não é uma coincidência, dado que a Android trabalha com Java de forma nativa para o desenvolvimento de seus aplicativos (DE OLIVEIRA, 2019).

Plataformas de desenvolvimento mobile contam, entre outros componentes, com um emulador de dispositivo, que permite que um desenvolvedor simule uma aplicação em construção em um dispositivo virtual diretamente de seu ambiente de desenvolvimento. **Isto é importante**, pois empacotar um software em construção e instalá-lo em um dispositivo móvel para cada atualização seria muito custoso. A Figura 4 mostra um exemplo de emulador para o dispositivo Pixel 5 da Google.

Voltar ao sumário

Figura 4 - Emulador para o dispositivo Pixel 5 da Google



Fonte: Elaborada pelo autor (2022)

Cloud

Diferente das demais plataformas, a computação em nuvem, em inglês *cloud computing* ou apenas *cloud*, é a plataforma mais recente e possui características marcantes que a diferencia das demais. Vem se tornando popular, pois, em determinados cenários, agiliza o desenvolvimento e reduz os passos necessários para se iniciar o trabalho.

Em geral, um cliente não sabe se está usando uma nuvem ou não, pois existe, por definição, uma abstração da camada de servidores. Os recursos são disponibilizados de maneira tradicional, mas os servidores, muitas vezes, são virtualizados. Ou seja, não são necessariamente servidores físicos ou máquinas físicas instaladas em algum local, mas sim servidores virtuais em execução em máquinas virtuais. Aplicações comuns que estão disponíveis na nuvem incluem: Microsoft Office, Google Mail, Dropbox, entre outros. Mais pra frente, estudaremos alguns detalhes de computação em nuvem, mas, por enquanto, podemos usar a seguinte definição tradicional (ARMBRUST, 2009):

> A computação em nuvem é um modelo computacional que permite o acesso, sob demanda e através da rede (comumente na internet), a um “pool” de recursos computacionais (redes, servidores virtuais ou físicos, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados.

Voltar ao sumário

Existem vários modelos de implantação e modelos de distribuição, mas o que melhor se encaixa no assunto que estamos estudando é o PaaS, ou Plataforma como Serviço (do inglês *Platform as a Service)*. Neste modelo de distribuição, uma plataforma completamente pronta pode ser disponibilizada a um desenvolvedor em questão de minutos. Note que isto reduz significativamente o tempo necessário para configurar uma plataforma, pois não existe a necessidade de instalar componentes nem adquirir máquinas físicas, tornando as plataformas *cloud* opções populares e bem atraentes. Vale ressaltar que elas são altamente escaláveis e possuem um custo sob demanda. Ou seja, apesar de não haver compra de computadores, por exemplo, ainda é necessário pagar pelos recursos enquanto estes são utilizados. Além disso, existem algumas questões relacionadas à privacidade e segurança, que em alguns contextos podem ser fatores negativos ao se considerar uma plataforma *cloud* (BROOKSHEAR, 2013).

Importante!

Uma plataforma *cloud* é uma boa opção quando não queremos perder tempo com instalação e configuração, pois tudo estará pronto para o uso. Uma solução gratuita e bastante popular para desenvolvimento em plataforma *cloud* é o Heroku.

E agora? Qual plataforma é a mais apropriada? Muitos critérios podem ter um papel importante na decisão da plataforma que você irá utilizar. Logo aprenderemos a instalar e configurar ambientes de desenvolvimento e veremos quais são as principais ferramentas de apoio ao desenvolvimento de software.

2. Linguagens de programação: tipagens e paradigmas

Estudar conceitos gerais sobre o projeto e a avaliação de linguagens de programação é parte indispensável na formação de uma sólida base de conhecimento de qualquer programador ou programadora em busca de uma carreira de sucesso. Naturalmente, todos devem estar se perguntando: **"Certo, mas porque é tão importante assim e como me beneficiará como futuro profissional?".** Afinal, muitos outros tópicos em ciência da computação estão "sob os holofotes" do mercado de trabalho e também são merecedores de um estudo sério.

Voltar ao sumário

Segundo Sebesta (2018), “Acredita-se que a profundidade de nossa capacidade intelectual seja influenciada pelo poder expressivo da linguagem em que comunicamos nossos pensamentos. Os que possuem uma compreensão limitada da linguagem natural são limitados na complexidade de expressar seus pensamentos, isto é, é difícil para as pessoas conceber estruturas que não podem descrever, verbalmente ou por escrito.”

Em outras palavras, se analisarmos o que o autor fala, considerando programadores no processo de desenvolvimento software, podemos concluir que se eles não conhecem outras linguagens e paradigmas existentes, os softwares que eles desenvolvem sempre estarão limitados pelas restrições (tipos de estruturas de controle, estruturas de dados e abstrações) da linguagem na qual foram escritos. O resultado prático disso é que a maioria dos programadores, ao começarem um novo projeto, utilizam sempre a mesma linguagem que já estão mais familiarizados, mesmo que ela não seja a mais adequada. Como consequência, o desenvolvimento do produto será prejudicado, exigindo esforço extra para “contornar” as limitações impostas pela linguagem, gerando uma complexidade desnecessária para o projeto e com risco de que o produto final não consiga atender alguns requisitos. Tudo isso, no fim das contas, aumenta o tempo de projeto, aumenta os custos de desenvolvimento e diminui a qualidade.

Podemos dizer também que o uso de uma linguagem “favorita” pode estimular o uso de um mesmo ambiente de desenvolvimento “favorito”, as mesmas tecnologias “favoritas” e assim por diante. Mas note que nem sempre a mesma plataforma será utilizada.

Importante!

Imagine que você utiliza C++ em todos os seus projetos e você sempre opta pelo desenvolvimento Web. Agora suponha que sua empresa de desenvolvimento foi contratada para trabalhar com outra equipe em um novo projeto para dispositivos móveis, usando a IDE Android Studio e a linguagem de programação Java. Ao dominar mais de uma linguagem de programação, podemos aproveitar mais oportunidades de trabalho e decidir com mais propriedade qual plataforma é mais adequada para diferentes contextos.

Outros pontos positivos incluem (SEBESTA, 2018):

- Aumento da capacidade de expressar ideias;
- Embasamento para escolher linguagens adequadas;
- Aumento da habilidade para aprender novas linguagens;

Voltar ao sumário

- Melhor entendimento da importância da implementação;
- Melhor uso de linguagens já conhecidas;
- Avanço geral na computação.

Agora que temos um entendimento sobre a importância do assunto e a motivação para estudá-lo, vamos falar um pouco sobre o que são paradigmas de programação e quais os principais existentes no mercado atualmente.

EXEMPLO: Para compreender o conceito de paradigmas de programação, vamos pensar na seguinte situação: você quer fazer brigadeiro de colher na sua casa, você pesquisa na internet e encontra diversas receitas e tutoriais de como preparar o doce, todas com o mesmo objetivo final de produzir um delicioso brigadeiro de colher.

Porém, ao abrir as receitas, percebe que cada uma tem alguns detalhes diferentes: uma utiliza açúcar e cacau puro, outra usa o achocolatado já pronto; uma usa leite condensado integral, outra semidesnatado; uma é feita no fogão, outra no micro-ondas. Confuso com todas essas receitas diferentes e complexas, quase desistindo de fazer, você se lembra que sua avó tem uma receita bem simples e saborosa, liga pra ela, e ela te passa a receita e pronto!

Em programação, também podemos recorrer a diferentes metodologias (receitas) para alcançar um mesmo objetivo (brigadeiro de colher). A estas metodologias, damos o nome de paradigmas. Veja, então, como é importante conhecer diversos paradigmas para se escolher o mais adequado para o problema em consideração.

Agora que entendemos o conceito de paradigmas, vamos detalhar os seis principais paradigmas de programação da atualidade:

Paradigma Imperativo

No paradigma imperativo, como o nome já diz, o programa deve comandar imperativamente como e em qual sequência as instruções e os procedimentos devem ser executados pela máquina. Ou seja, neste paradigma, também chamado de procedural, o programador descreve um passo-a-passo detalhado do que deve ser executado pela máquina.

Perceba que dependendo do problema, este paradigma pode se tornar relativamente complexo, visto que uma simples mudança na ordem das instruções pode gerar um resultado não esperado e o código pode acabar ficando grande demais e de difícil legibilidade. Como consequência quase direta disto, ele não é indicado para a construção de aplicações que necessitarão de manutenção no curto prazo ou mudanças muito frequentes. Em contrapartida, tem a vantagem de ser eficiente e de permitir uma modelagem tal qual o mundo real.

Voltar ao sumário

Importante!

As principais linguagens de programação que utilizam este paradigma são Cobol, Pascal e Fortran. Uma das suas principais áreas de aplicação é a científica.

Paradigma Declarativo

O paradigma declarativo, ao contrário do paradigma imperativo, está mais focado em "o que" deve ser resolvido do que em "como" isso será feito. Sendo assim, o nível de abstração é maior, no qual a pessoa que o programa tem a preocupação de descrever determinada sequência lógica e qual o resultado esperado.

Ele leva este nome porque, ao utilizar este paradigma, o programador ou programadora declara, inicialmente, verdades lógicas imutáveis para as quais os resultados serão sempre os mesmos após suas interações. Assim, depois de realizadas algumas interações entre elas, o resultado encontrado será sempre igual (para as declarações outrora feitas). Isso entra em contraste com a programação imperativa, na qual um mesmo trecho de código pode retornar resultados diferentes.

As principais linguagens declarativas são também linguagens de marcação: HTML, XML, XSLT e XAML.

Paradigma Orientado a Objetos

O paradigma orientado a objetos é, talvez, o mais popular do mercado atualmente, tendo sido popularizado na década de 90, especialmente com a linguagem de programação Java com o objetivo de facilitar o desenvolvimento de aplicações web, possibilitando uma programação multiplataforma de uma mesma maneira, algo inédito para época.

Em outras palavras, com este paradigma, sites, aplicativos e softwares são desenvolvidos uma única vez e são interpretados por diferentes plataformas sem obstáculos. Antes, era preciso adaptar a aplicação de acordo com onde ela iria rodar. Dito isto, fica fácil perceber que este paradigma é altamente modular e reusável, com um ampla área de aplicação.

Para que uma linguagem de programação seja do tipo de paradigma orientado a objetos, ela deve implementar seus três alicerces básicos, que são:

- Conceito de herança,
- Polimorfismo
- Encapsulamento.

Voltar ao sumário

As principais linguagens de programação que utilizam este paradigma são Python, C++ e Java.

Paradigma Orientado a Eventos

O paradigma orientado a eventos é aquele que define a execução de determinadas funções conforme eventos disparados pelo usuário, sendo muito utilizado no mercado de aplicações que utilizam recursos gráficos, como jogos e formulários.

Por exemplo, pressionar um botão de envio, terminar de preencher um formulário, atingir determinado estágio em um jogo para desbloquear ações futuras etc.

As principais linguagens de programação que utilizam este paradigma são: visual Basic e Delphi.

Paradigma Funcional

Como o nome já sugere, neste paradigma destaca-se o uso das funções. Na hora de resolver um problema, este é dividido em blocos nos quais são implementadas funções responsáveis pela sua resolução. Estas funções definem algumas variáveis que podem ou não retornar resultados.

É bastante indicado quando a solução requerida é fortemente dependente de uma base matemática. Assim, subdivide-se o problema proposto e as funções implementadas farão os cálculos matemáticos. Após a conclusão, esta solução deverá ser integrada pelo programador.

Importante!

As principais linguagens de programação que utilizam este paradigma são Haskell, Scheme, Clojure e LISP. Uma área na qual tem muita aplicabilidade é a de Inteligência Artificial.

Paradigma Lógico

O paradigma lógico é um tanto distinto dos demais paradigmas e deriva do declarativo. Fundamentalmente, utiliza formas de lógica simbólica como padrões de entrada e saída. A partir daí, realiza inferências para produzir os resultados.

Os principais elementos deste paradigma são: proposições, regras de inferência e busca.

A principal linguagem de programação a utilizar este paradigma é o Prolog.

Voltar ao sumário

Tipagens

Outro conceito muito importante das linguagens de programação é o conceito de tipagem. De maneira simplificada, a tipagem de uma linguagem de programação nos diz como ela trata os tipos de dados. Logo, seu entendimento é essencial para que a lógica do programa funcione conforme o projetado e evite erros de compilação ou resultados inesperados. Afinal de contas, a programação é basicamente processar dados e, para fazer isso, o programa precisa saber qual é o tipo de dado que será processado.

De maneira geral e simplificada, uma linguagem deve ter uma tipagem estática ou dinâmica e deve ter uma tipagem forte ou fraca. Logo, existem quatro possíveis combinações de tipagem para uma linguagem de programação: com tipagem dinâmica e forte, com tipagem dinâmica e fraca, com tipagem estática e forte e com tipagem estática e fraca.

Uma linguagem é dita de **tipagem estática** quando a pessoa que está programando precisa informar explicitamente o tipo de cada dado utilizado no sistema: variáveis, parâmetros de funções, valores de retorno etc. Uma vez definido o tipo, estas variáveis estão restritas ao tipo declarado.

```
public class MyClass {
    public static void main(String args[]) {
      int idade= 30;

      idade= "Ronaldo"; //error: incompatible types: String cannot
be converted to int
    }
}
```

Veja o exemplo acima na linguagem Java: quando tentamos atribuir um valor de um tipo diferente do que foi declarado na variável, temos um erro. Exemplos de linguagens estaticamente tipadas são C, C++, Java. Go e Rust.

Já na **tipagem dinâmica**, como o nome sugere, a linguagem de programação define o tipo de dado de acordo com o valor atribuído à variável em tempo de execução:

```
$variavel = "Teste";
echo gettype($variavel); //string
$variavel = 3;
echo gettype($variavel); //integer
$variavel = 3.89;
echo gettype($variavel); //double
$variavel = true;
echo gettype($variavel); //boolean
```

Voltar ao sumário

No exemplo em PHP acima, veja que a mesma variável assumiu diferentes tipos de dados conforme a execução do programa. Alguns exemplos de linguagens com tipagem dinâmica são: JavaScript, Python, PHP e Ruby.

A **tipagem fraca** está ligada à característica da linguagem de realizar conversões automaticamente entre diferentes tipos de dados. Veja o exemplo abaixo em Javascript:

const num = “10”;

const operacao = num + 10; *//1010*

O programa não gerou erros, pois o JavaScript fez uma conversão implícita do valor 10 de número para string, para então concatenar com a string ”10”.

Já uma linguagem com **tipagem forte** não realiza conversões automaticamente. Veja este exemplo em Python:

num = “10”

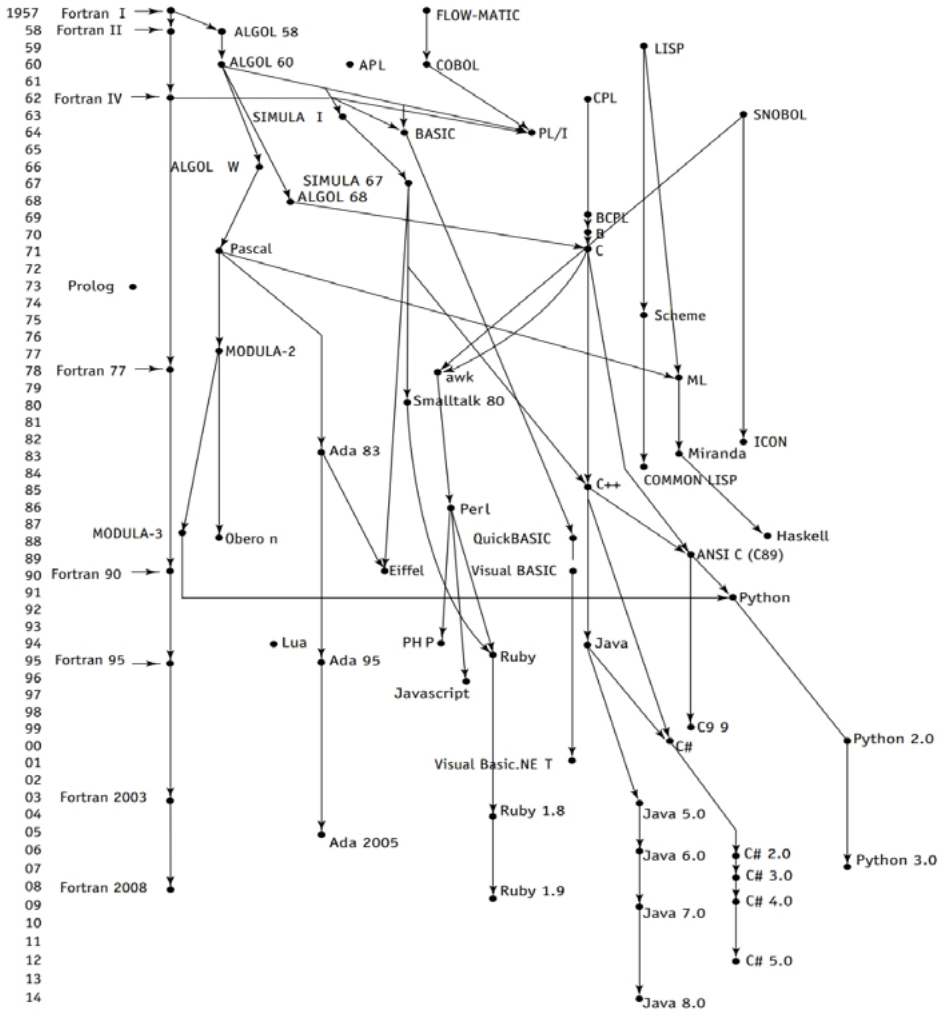
operacao = num + 10*; //TypeError: cannot concatenate ‘str’ and ‘int’ objects*

Vimos que uma tentativa de operação de soma entre string e número resulta justamente em um type error, ou seja, um erro de tipo: “não é possível concatenar objetos str e int”.

É interessante notar, também, que linguagens com tipagem dinâmica não têm necessariamente tipagem fraca, como vimos acima com Python.

Voltar ao sumário

Figura 5 - Linguagens de programação



Fonte: Adaptada de (SEBESTA, 2018).

A *Figura 5* mostra uma série de linguagens de programação populares e uma noção de evolução baseada nas datas em que foram disponibilizadas. Como é importante sabermos as principais linguagens de acordo com seus detalhes, aqui estão alguns exemplos:

- Linguagens com tipagem dinâmica e forte: Python, Ruby, Erlang, Clojure
- Linguagens com tipagem dinâmica e fraca: JavaScript, PHP, Perl
- Linguagens com tipagem estática e forte: C#, Java, Scala
- Linguagens com tipagem estática e fraca: C, C++

Voltar ao sumário

3. Ambiente integrado de Desenvolvimento: Instalação e configuração

Já aprendemos sobre plataformas de desenvolvimento e detalhes importantes de linguagem de programação. Agora, como tudo isso se encaixa? Neste circuito, vamos utilizar o conhecimento adquirido para fazer boas escolhas de ambiente.

Anteriormente, mencionamos que um ambiente de desenvolvimento possui ferramentas para auxiliar desenvolvedores em suas tarefas, que incluem: programação, teste, depuração, projeto e assim por diante. Chamamos de ambiente integrado de desenvolvimento, ou IDE (do inglês, Integrated Development Environment), o local de trabalho de desenvolvedores. Vimos que uma possível definição seria (DE OLIVEIRA, 2019):

Importante!

IDE, do inglês integrated development environment, ou ambiente de desenvolvimento integrado, é um software voltado para o desenvolvimento de programas ou aplicativos. O desenvolvedor encontra nos IDEs todas as funções necessárias para o desenvolvimento, desde programas de computador a aplicativos mobile, assim como recursos que possibilitam a criação de códigos com uma incidência menor de erros.

Podemos dizer que a principal funcionalidade de uma IDE é permitir que desenvolvedores escrevam linhas de código. Isto é, existe um espaço para edição de texto que representa o código sendo produzido. É possível desenvolver páginas HTML (paradigma declarativo) usando apenas um bloco de notas. Todavia, dizemos que construir uma página HTML em uma IDE é mais fácil, pois existem vários componentes que auxiliam sua construção.

Iremos aprender neste circuito a instalar e configurar dois ambientes de desenvolvimento muito populares: o Eclipse e o Android Studio. Mais para frente, visitaremos outras IDEs, como o IntelliJ IDEA e o Visual Studio Code (VS Code). Antes de começarmos o processo de instalação, vamos rever os componentes mais comuns encontrados em IDEs.

Marcador de sintaxe

Destaca palavras que foram escritas de forma errada e indica começo e fim de blocos de loop, linhas selecionadas, entre outras marcações visuais diretamente aplicadas no texto (ou código) dos desenvolvedores.

Voltar ao sumário

Autocompletador de código

Fornece sugestões para preenchimento automático de palavras sendo escritas e completa elementos de sintaxe enquanto são preenchidos (digitar '{' inclui automaticamente '}').

Refatorador

Imagine que você cria uma variável chamada "valor". Ela está sendo utilizada em várias linhas do seu código. Agora imagine que você deseja renomear essa variável para "valorTotal". Em uma IDE, chamamos esse processo de refatoração, no qual você muda o nome de uma variável ou outro elemento de código em apenas um local e a IDE busca, identifica e muda todas as outras ocorrências e faz os ajustes automaticamente.

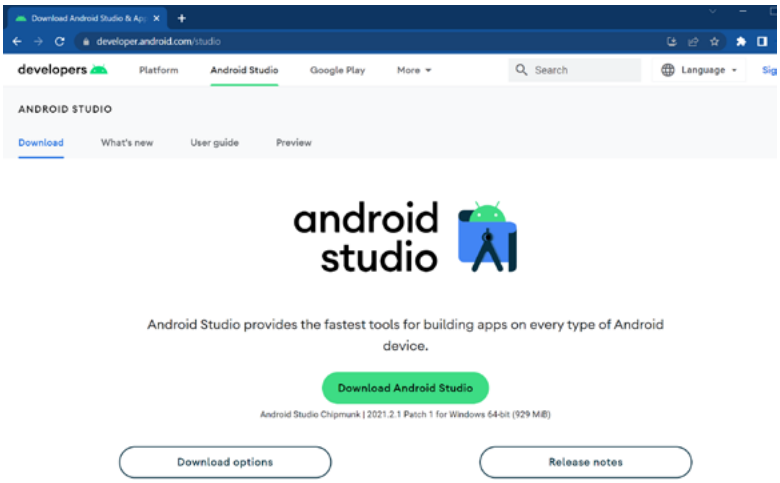
Controlador de versão

É possível recuperar uma versão diferente de um mesmo arquivo ao longo do tempo. Suponha que você está escrevendo um código pela manhã e no final da tarde percebe que, no meio do caminho, você adotou uma abordagem errada. Com um controlador de versões, você pode revisitar estados antigos do seu código, visualizando o código que você tinha no final da manhã ou início da tarde, podendo reverter para um deles.

A maior parte das IDEs tem pontos em comum, mas os critérios que utilizamos na escolha de uma IDE podem incluir: plataformas de desenvolvimento e linguagens a serem utilizadas. Digamos que você deseja trabalhar em uma plataforma mobile. Neste caso, uma boa opção de ambiente de desenvolvimento seria o Android Studio.

Para instalar o Android Studio, deve-se fazer o download da versão mais recente da IDE, conforme mostra a *Figura 6*. Após clicar no botão para baixar a versão mais recente, espere para que o download seja concluído e execute o arquivo baixado.

Figura 6 - Download Android Studio

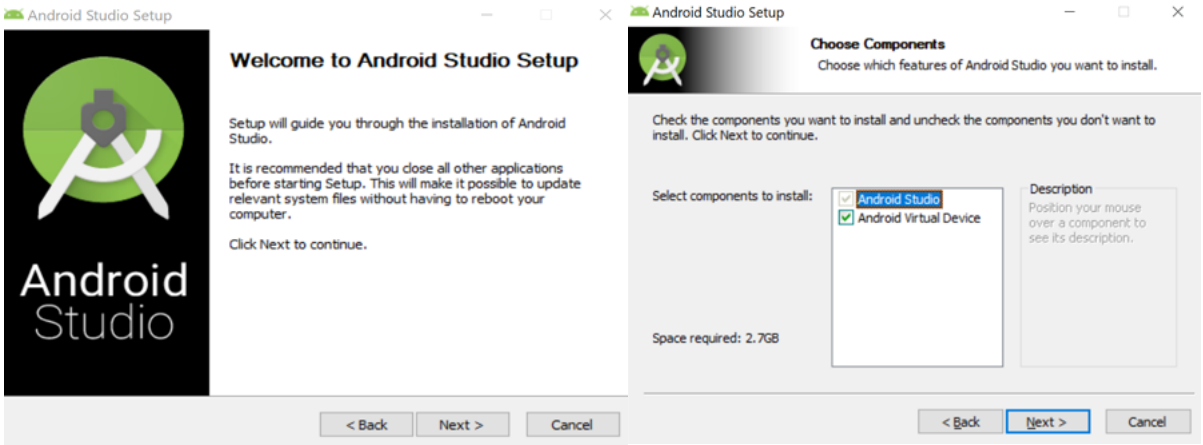


Fonte: Elaborada pelo autor (2022).

Voltar ao sumário

Após abrir o arquivo executável recém baixado, você verá uma tela de instalação, conforme a figura abaixo. Aperte “Next” e na próxima tela você terá a opção de escolher quais componentes instalar. Note que falamos anteriormente sobre a importância de um emulador quando desenvolvemos aplicativos móveis, então tenha certeza de que a opção “Android Virtual Device” está selecionada, pois um dispositivo virtual Android pré-configurado e otimizado será instalado para testes com o emulador.

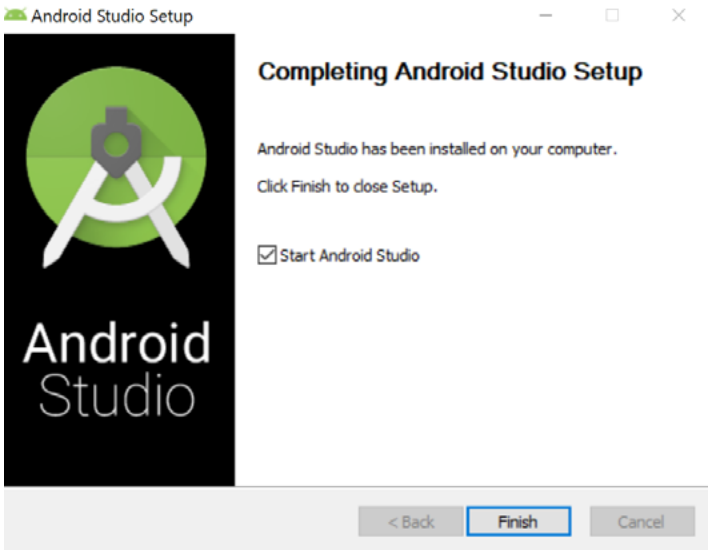
Figura 7 - Android Virtual Device



Fonte: Elaborada pelo autor (2022).

Depois de selecionar os componentes desejados, você irá escolher o local da instalação e o nome da pasta de atalho para fácil acesso no menu de seu sistema operacional. Ou seja, em outras palavras, você terá que apertar “Next” algumas vezes até que seja possível clicar em “Install”, o que iniciará o processo de instalação. Separe alguns minutos para a instalação, pois alguns pacotes são baixados durante a instalação.

Figura 8 - Android Virtual Device



Fonte: Elaborada pelo autor (2022).

Voltar ao sumário

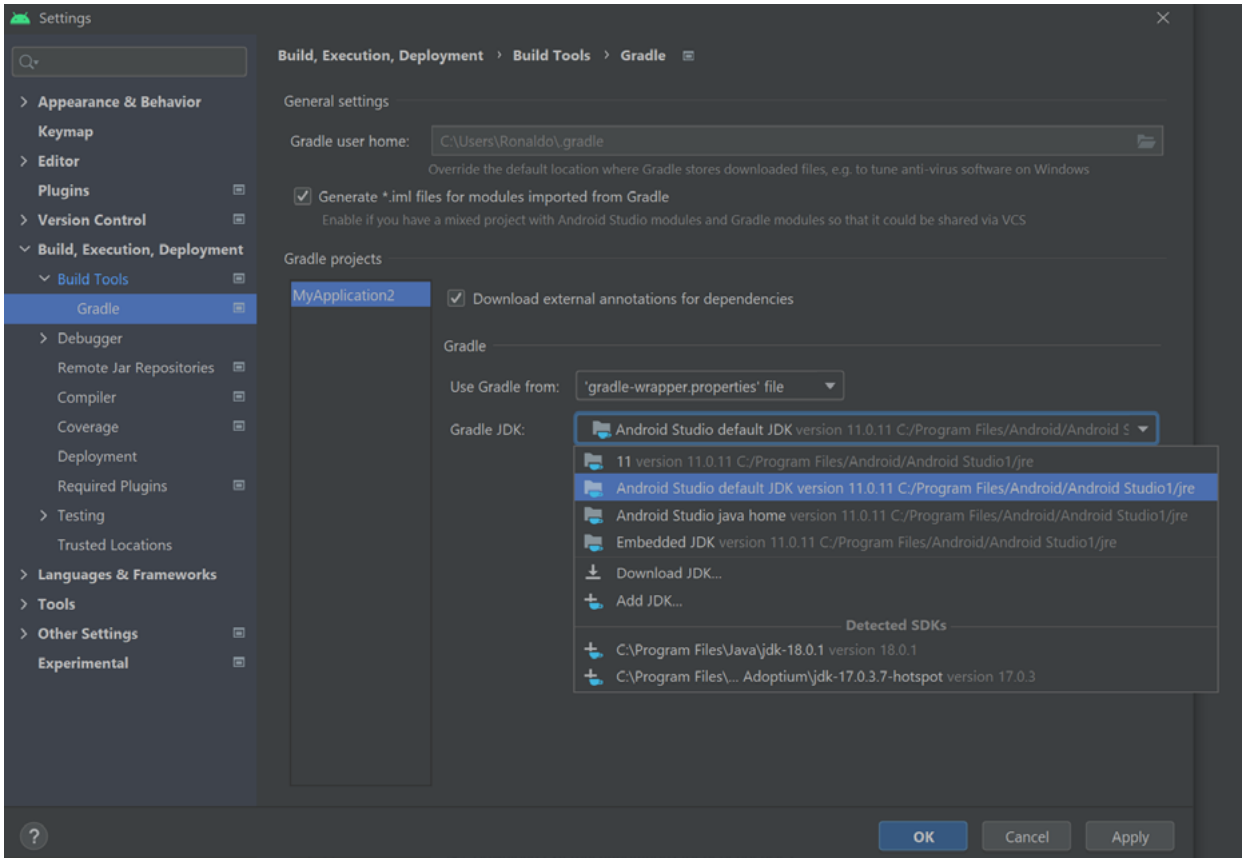
Assim que o processo de instalação é concluído, você deverá apertar “Next” e depois “Finish”. Deixe a opção “Start Android Studio” selecionada, pois iremos configurar nossa IDE.

Para aqueles que já utilizaram o Intellij IDEA, mas não utilizaram o Android Studio, será possível notar a semelhança entre os dois ambientes de desenvolvimento. Isto se dá pelo fato de que o Android Studio foi desenvolvido tendo como base o Intellij. Caso você não tenha familiaridade com nenhuma dessas IDEs, não tem problema. Iremos visitar passo a passo cada componente. Primeiramente, iremos verificar se a JDK está configurada.

1. Abra um novo projeto no Android Studio
    a. Neste momento, não se preocupe com o conteúdo do projeto, apenas crie um projeto com o nome padrão (geralmente chamado “My application”)
2. Clique na aba File (em português, Arquivo).
    a. Caso seu sistema operacional seja uma versão de Mac OS, clique em Preferences (em português, preferências)
3. Selecione a opção Build, Execution, Deployment
4. Clique em Build Tools
5. Selecione a opção Gradle
6. Verifique se a opção Gradle JDK possui uma JDK selecionada
    a. Caso não tenha, você deve escolher a opção Dowload JDK presente neste campo
7. Clique aplicar
8. Clique OK

Voltar ao sumário

Figura 9 - Novo projeto no Android Studio



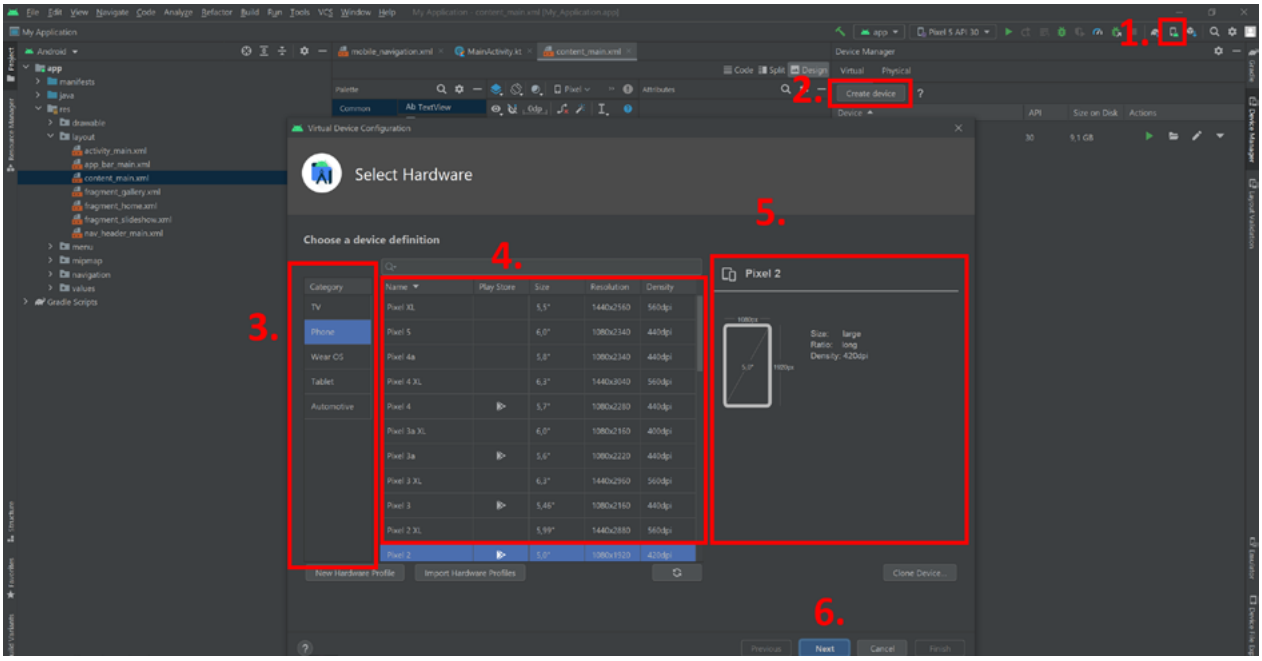
Fonte: Elaborada pelo autor (2022).

Muitos componentes podem ser configurados, dependendo do contexto em que o desenvolvimento de software será empregado. Neste caso, vamos visitar a configuração de um novo dispositivo para o emulador.

1. Clique em Device Manager (Gerenciador de Dispositivos)
2. Selecione a opção de criar um novo dispositivo
3. Escolha a categoria do dispositivo dentre as opções: TV, Celular, Tablet, etc.
4. Dentre os dispositivos existentes que se encaixam com a categoria escolhida, selecione o dispositivo que você gostaria de emular, para que este seja criado
5. Verifique as especificações do dispositivo
6. Confirme as seleções dos passos anteriores.

Voltar ao sumário

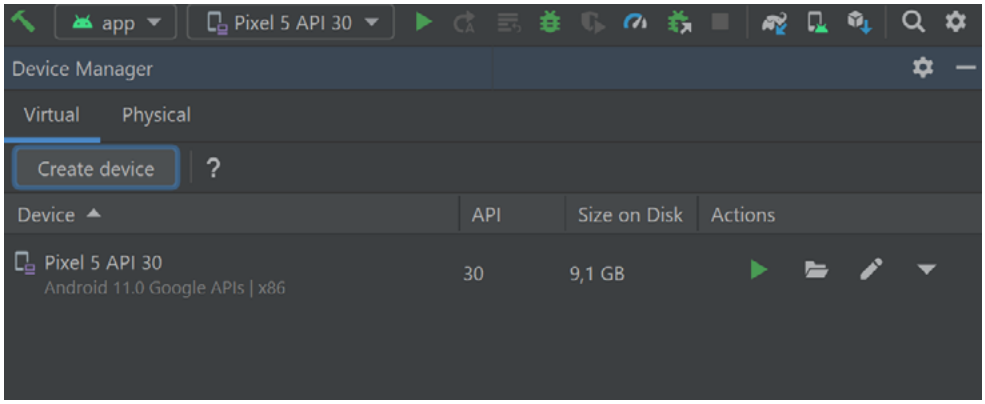
Figura 10 - Dispositivo para o emulador



Fonte: Elaborada pelo autor (2022).

Veja, na figura acima, como realizar cada um dos passos de configuração descritos. Após realizados, você deverá realizar o download de uma imagem de sistema, arquivo responsável por armazenar a simulação do seu dispositivo. Uma vez baixado, você poderá criar seu dispositivo e emulá-lo através da opção Launch (botão de “Play”) na área do Gerenciador de dispositivos.

Figura 11 - dispositivo para o emulador



Fonte: Elaborada pelo autor (2022).

Em outro cenário, suponha que você irá utilizar a linguagem Java (tipagem estática e forte) em seu ambiente de desenvolvimento. Esta escolha já diminui consideravelmente a quantidade de opções disponíveis. Neste caso, uma boa escolha seria o IntelliJ IDEA.

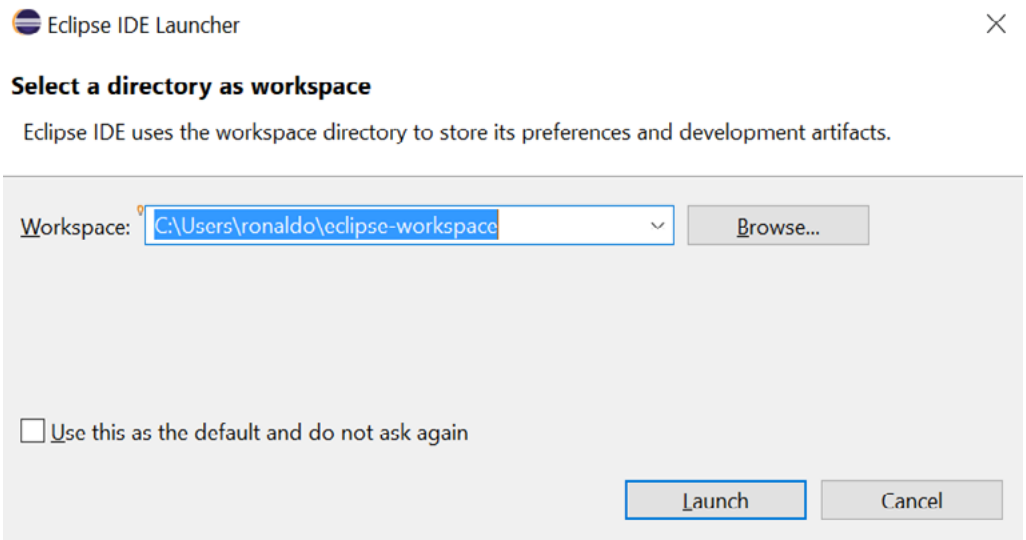
Voltar ao sumário

Caso não haja planos de investimento financeiro, como a versão completa do IntelliJ é paga, podemos optar em utilizar o Eclipse, IDE gratuita da IBM, que se torna uma ótima opção. Para estudarmos ambientes de desenvolvimento neste primeiro material, iremos iniciar pelo Eclipse. Ao todo, leva-se cinco minutos para a instalação, mas como alguns pacotes são descarregados durante a instalação, ela pode demorar um pouco mais dependendo da sua velocidade de conexão com a Internet. Além disso, considerando a configuração, separe um tempo de 30 minutos para fazer essa atividade.

Dirija-se até o site oficial (https://www.eclipse.org/downloads/) para o download:

1. Baixe a versão mais recente.
2. Após o download, execute o arquivo baixado
3. Durante o processo de instalação, o eclipse mostrará seus vários packages.
    a. Por hora, usaremos a versão Eclipse IDE for Java Developers.
    b. Escolha a opção correspondente
    c. Clique em “Install”.
4. Após instalado, clique “Launch” para executar sua IDE.

Figura 12 – Instalação do Eclipse



Fonte: Elaborada pelo autor (2022)

Se você estiver vendo a tela acima, clique em “Launch” para darmos início ao passo de configuração:

Voltar ao sumário

1. Vá até *Window*
2. Clique em *Preferences*
3. Procure e selecione a opção Java
4. Clique em *Installed JREs*
5. Uma vez com a lista de JREs, cheque se existe uma JRE ou uma JDK na lista. Se sim, o Java está configurado em seu ambiente. Caso não esteja, faça o download por este campo.
6. Pronto!

Configurações avançadas

A maior parte dos componentes presentes em um ambiente de desenvolvimento pode ser configurada para melhor atender a um desenvolvedor em específico. Cada IDE possui uma miríade de opções de configurações avançadas. Algumas dessas opções podem estar agrupadas em um configurador gráfico. Vamos conferir como funciona um exemplo desse configurador para o Eclipse. Para isso, na tela de boas-vindas (disponível na aba Ajuda), clique em “Review IDE configuration settings".

Atualização automática de recursos (Refresh resources automatically)

A primeira funcionalidade a ser configurada é a atualização automática de recursos, que detecta modificações em arquivos. Para que você não precise ficar atualizando (Refresh ou F5) seus recursos, vamos configurar nosso ambiente de desenvolvimento para que isto seja feito de forma automática. Basta clicar no ícone de afirmativo presente na imagem abaixo.

Figura 13 - Atualização automática de recursos

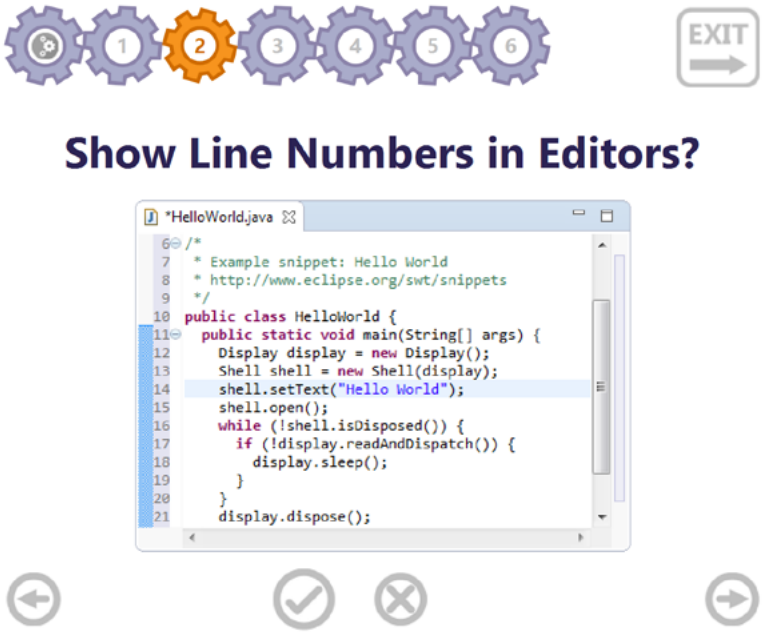
Fonte: Elaborada pelo autor (2022)

Voltar ao sumário

Exibir numeração de linhas de Código (Show line numbers in Editors)

Enquanto estou trabalhando em um código, eu gosto de ver o número da linha, então em geral configuro minha IDE para exibir as linhas de código. Caso não goste ou acredite que os números poluem a tela, você pode configurar sua IDE para não mostrar as numerações de linhas de código.

Figura 14 - Linhas de Código



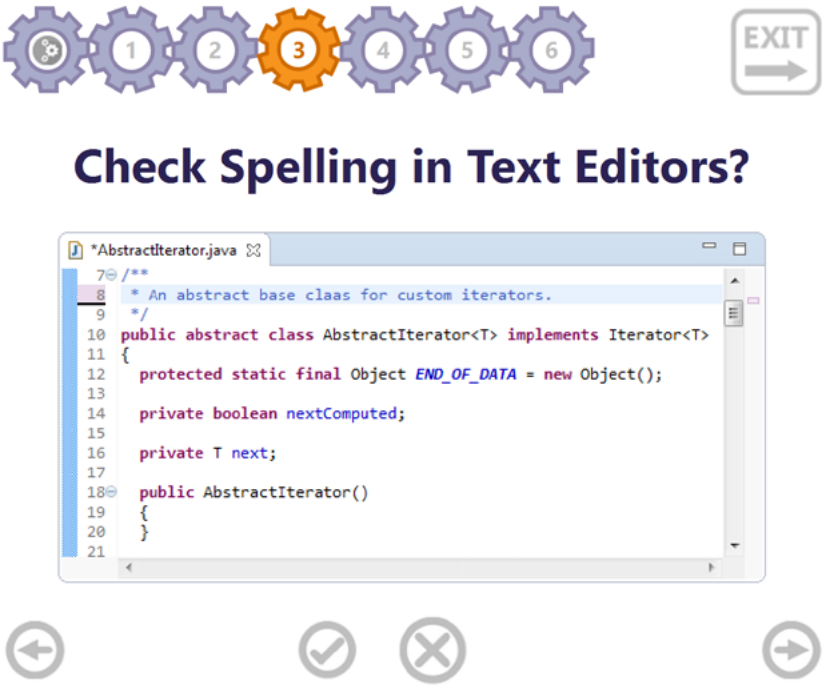
Fonte: Elaborada pelo autor (2022)

Checagem gramatical (Check Spelling in Text Editors)

A terceira funcionalidade é a de checagem gramatical em comentários e elementos de código. Um desenvolvedor experiente sabe que, na área da computação, existem muitas siglas e em alguns projetos é possível utilizar mais de uma língua, como português e inglês, a checagem gramatical pode prejudicar a leitura do código, indicando haver erros em siglas e/ou em porções do código que estão em mais de um idioma natural. Em geral, podemos optar por deixar esta funcionalidade desativa. Todavia, em um projeto no qual sabe-se que apenas uma linguagem será utilizada, sinta-se livre para ativar esta funcionalidade na sua IDE.

Voltar ao sumário

Figura 15 - Checagem gramatical

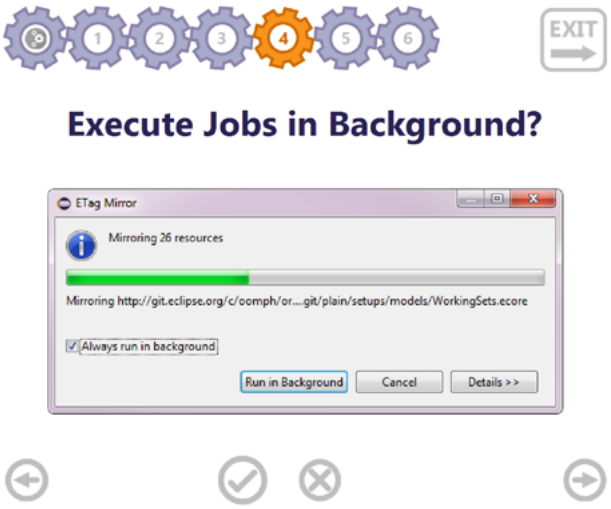


Fonte: Elaborada pelo autor (2022)

Executar tarefas ao fundo (Execute Jobs in Background)

Esta próxima funcionalidade nos ajuda a manter nossa atenção onde importa. Então, se existem tarefas em execução, iremos colocá-las para executar ao fundo, ou seja, escondido, para que elementos gráficos não apareçam durante o uso da IDE e desconcentre o desenvolvedor. Em geral, configuramos nosso ambiente de desenvolvimento para manter essa funcionalidade ativa. Basta clicar no ícone de afirmativo presente na imagem abaixo.

Figura 16 - Executar tarefas ao fundo



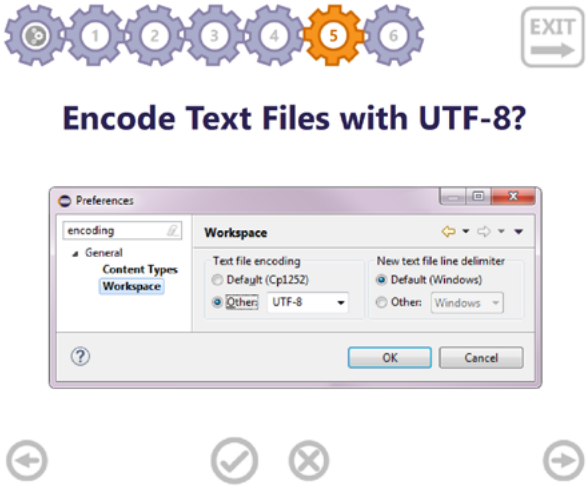
Fonte: Elaborada pelo autor (2022)

Voltar ao sumário

Codificar arquivos em UTF-8 (Encode Text Files with UTF-8)

Geralmente, podemos utilizar o padrão UTF-8 de codificação para nossos arquivos de código-fonte. Colocaremos um sim nesta funcionalidade. Se você espera trabalhar com caracteres especiais ou algo mais específico, essa pode ser uma configuração a ser revisitada no futuro.

Figura 17 - Codificar arquivos em UTF-8

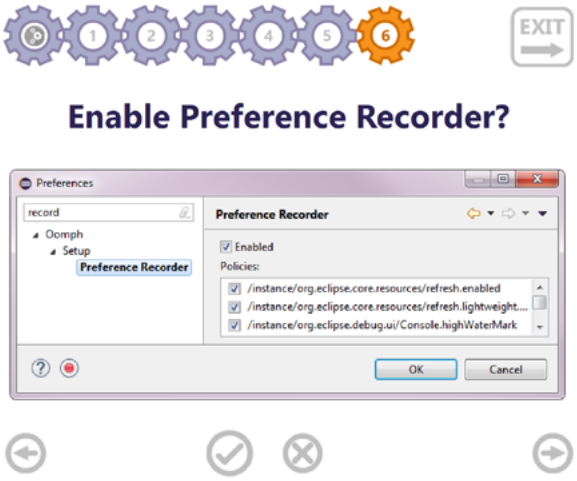


Fonte: Elaborada pelo autor (2022)

Habilitar registro de preferências (Enable Preference Recorder)

Por fim, gostaríamos de habilitar o registro de nossas preferências. No futuro, podemos ter múltiplas configurações em um único ambiente de desenvolvimento. Ou seja, dependendo do projeto ou software em consideração, você pode alternar facilmente entre configurações, poupando um tempo considerável.

Figura 18 - Habilitar registro de preferências



Fonte: Elaborada pelo autor (2022)

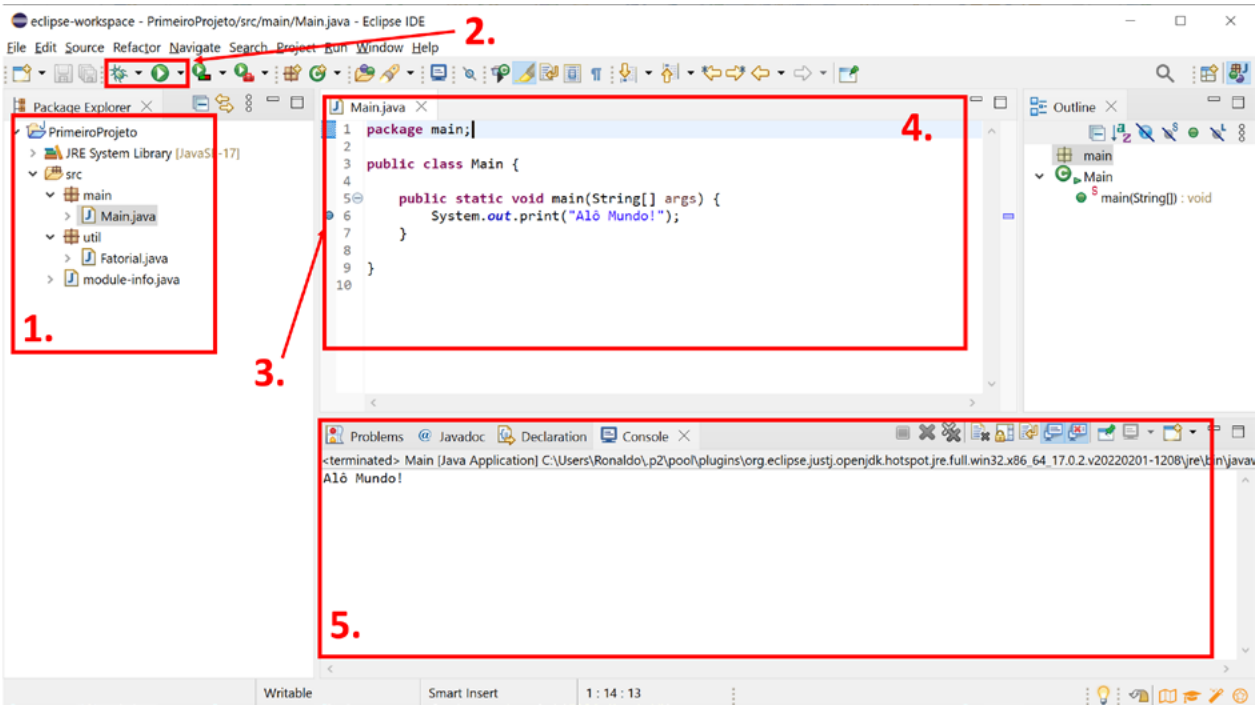
Voltar ao sumário

4. Utilitários do ambiente de desenvolvimento: Depuração, console e estrutura de projetos

Neste último circuito, iremos aprender o que são os utilitários de um ambiente de desenvolvimento. Anteriormente, realizamos a instalação dos ambientes de desenvolvimento Eclipse e Android Studio. Agora, para um novo projeto qualquer, visitaremos os principais utilitários dessas duas IDEs.

Eclipse

Figura 19 - Eclipse



Fonte: Elaborada pelo autor (2022)

1. Explorador de pacotes

Seu projeto Java fica estruturado aqui – a pasta src (que vem de source, ou fonte) – é onde fica seu código fonte. Colocamos aqui dois pacotes: o pacote main, que guarda as classes de controle; e o pacote util, que guarda as classes de utilidade. A manipulação de arquivos é feita toda aqui: criação, atualização e remoção de pacotes, classes Java e assim por diante.

Voltar ao sumário

2. Área de depuração e execução

O símbolo de “bug” (do inglês, que significa inseto) indica o modo Debug, ou Depuração. Podemos dizer que o utilitário mais usado para corrigir erros em código é o debugger, ou depurador. O símbolo de “play” indica o modo Run, ou rodar, uma execução normal, não havendo uma intenção de teste como no modo Debug.

3. Ponto de depuração

Quando não sabemos o que há de errado com nosso código, podemos executá-lo de modo a parar em determinados pontos para testá-lo. Por exemplo, ao executar a classe Main em modo Debug, a execução será pausada ao chegar na linha 6, para que o desenvolvedor veja diversos detalhes antes que a linha seja executada, na esperança de corrigir um possível erro nela. Para adicionar pontos de depuração (ou breakpoints), basta clicar duas vezes ao lado esquerdo da linha.

4. Editor de texto

Área dedicada à criação de texto, é aqui que escrevemos nosso código. Todas as configurações que fizemos anteriormente irão nos ajudar a desenvolver código nesse espaço. Note que palavras reservadas específicas da linguagem Java são coloridas de roxo. Muitas outras funcionalidades estão presentes aqui: clique botão direito na parte branca do editor de texto para ver muitas dessas opções. Todavia, este não é um curso de desenvolvimento, então vamos manter o foco no ambiente.

5. Console

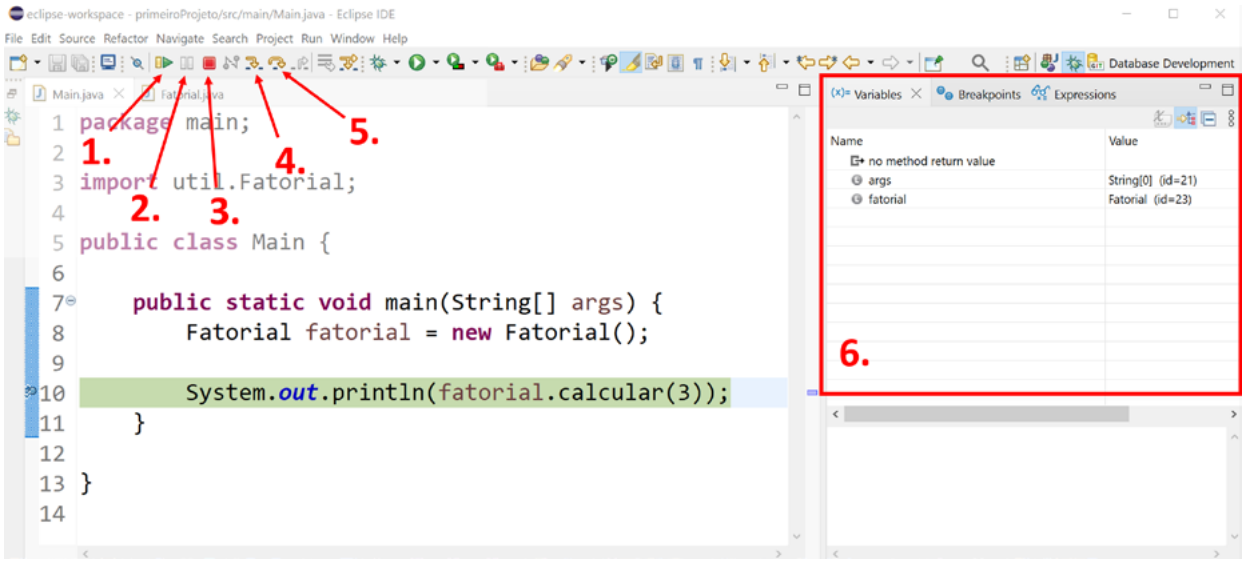
Após executar a classe Main presente na imagem acima, uma saída – ou output – será gerada. Toda comunicação de entrada e saída se dá através da janela de console. Neste componente, encontraremos outputs do sistema e input – ou entrada – de usuários. Na imagem acima, vemos que a execução da classe Main envia para o console o texto “Alô Mundo!”.

O console é utilizado para muitas funcionalidades, como: emitir avisos internos do ambiente de desenvolvimento ou de componentes instalados, imprimir erros de execução no código-fonte, estabelecer canal de interação com um usuário, etc. Todavia, o console não é de direto interesse do cliente, que, em geral, tem acesso às funcionalidades através de uma interface gráfica. Podemos dizer o mesmo sobre a maior parte dos utilitários de uma IDE, pois são ferramentas voltadas para desenvolvedores. De toda forma, é importante mencionar em mais detalhes sobre o utilitário de depuração e seus componentes, pois estão envolvidos no processo de teste. Alguns testes envolvem clientes de forma direta.

Voltar ao sumário

Um software de qualidade passa, potencialmente, por diversos tipos de teste. Falaremos de testes no próximo circuito, mas por hora precisamos definir que um teste de software e um processo de depuração de código-fonte são atividades distintas. Apesar deste ser o caso, um processo de depuração deve estar associado a alguma estratégia de teste (PRESSMAN, 2021). Ou seja, devemos entender que quando produzimos um código-fonte, podemos introduzir "bugs" de forma não intencional. Encontra-se abaixo uma imagem do ambiente de desenvolvimento *Eclipse* em modo de depuração.

Figura 20 - Ambiente de desenvolvimento Eclipse em modo de depuração



Fonte: Elaborada pelo autor (2022)

No Eclipse, ao executar um código em modo debug, o ambiente irá mudar para a perspectiva de depuração. Isto é, haverá uma mudança na posição dos componentes da IDE e nas opções mais acessíveis disponíveis para desenvolvedores.

1. *Resume*: esta opção estará disponível quando a execução estiver suspensa. Neste caso, note que a execução está suspensa na linha 10 pois existe um breakpoint nesta linha. Ao clicar no resume, a execução não estará mais suspensa e irá continuar até concluir ou atingir outro breakpoint.

2. *Suspend*: esta opção estará disponível se um código estiver em execução e pode ser útil quando o desenvolvedor obtém o que buscava e não precisa esperar que a execução completa, em casos de suspeita de uma execução presa em um loop infinito e assim por diante.

3. *Terminate*: interrompe imediatamente a execução, independente do seu estado atual.

Voltar ao sumário

4. *Step into*: em uma execução suspensa, a opção *step into* é utilizada para executar apenas a linha atual do código-fonte. Neste exemplo, a linha 10 será executada e o fluxo de execução irá entrar no método "calcular" da classe Fatorial. Ou seja, após a execução da linha 10, a depuração será suspensa dentro do método calcular da classe fatorial. Nesta nova classe, desenvolvedores podem continuar o processo de depuração.

5. *Step over*: de forma semelhante ao *step into*, o *step over* é utilizado para executar apenas a linha atual do código-fonte. A diferença é que no *step over* a execução não entra em chamadas de método, mesmo que exista uma ou mais chamadas presentes na linha. Neste exemplo, a linha 10 será executada e o fluxo de execução irá para a linha 11, sem adentrar o método "calcular" da classe Fatorial.

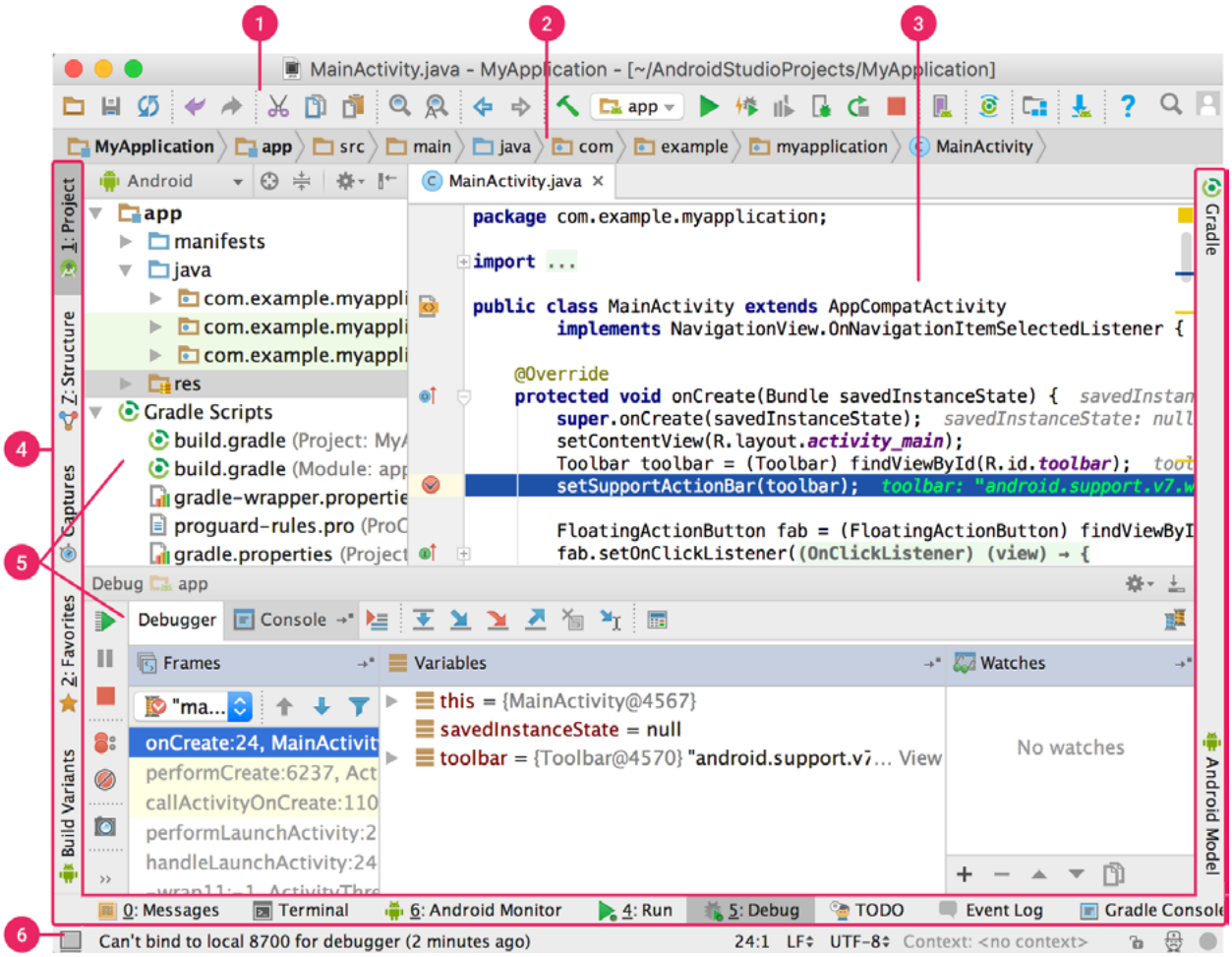
6. *Variáveis*: neste espaço, encontram-se todas as variáveis em memória devido à execução atual. Em uma execução com muitas variáveis, parâmetros, atributos, etc., podemos acompanhar detalhes e valores atuais, auxiliando na compreensão de como e quando estes valores mudam.

Nem todos os ambientes de desenvolvimento trabalham com perspectivas como o Eclipse. Dependendo da plataforma, podemos ver a adição de outros componentes e/ou a ausência de alguns componentes, mas em termos de utilitários podemos dizer que os ambientes de desenvolvimento são semelhantes. A seguir, descrevemos os principais componentes de um ambiente de desenvolvimento especializado na plataforma mobile, o Android Studio.

Voltar ao sumário

Android Studio

Figura 21 - Android Studio



Fonte: Adaptada de DE OLIVEIRA (2019).

A *Figura 21* mostra uma visão geral do Android Studio para um projeto de software móvel. Pode-se notar os principais componentes da interface gráfica disponíveis nesta ferramenta (DE OLIVEIRA, 2019):

1. Barra de ferramentas

Área que possibilita a manipulação do arquivo de código-fonte atualmente sendo visualizado e/ou edição de texto. Dentre as opções de manipulação de arquivos estão: abrir um novo arquivo, salvar o arquivo atual e atualizar o arquivo atual. Dentre as opções de edição de texto, temos desfazer e refazer última operação de texto, recortar texto selecionado, copiar texto selecionado e colar texto.

2. Barra de navegação.

Área que possibilita a visualização da estrutura de pastas do projeto atual. É possível clicar nas pastas para realizar uma navegação pela estrutura de pastas do projeto.

Voltar ao sumário

3. Editor de texto

Área dedicada à criação de texto, é aqui que escrevemos nosso código. Todas as configurações que fizemos anteriormente irão nos ajudar a desenvolver código nesse espaço. Note que palavras reservadas específicas da linguagem Java são coloridas de roxo ou azul.

4. Barra de janela de ferramentas

Área onde estão localizados botões para expandir ou esconder ferramentas secundárias, como o Terminal, Monitor Android, Estrutura, Modelos Android e assim por diante.

5. Janela de ferramentas

Área onde realiza-se gerenciamento dos projetos, com importantes tarefas como depuração, controle de versão, console, entre outros.

6. Barra de status

Área dedicada para exibição de mensagens e advertências do ambiente de desenvolvimento, como, por exemplo, o status do projeto atual.

Voltar ao sumário

Resumo:

Neste percurso, estudamos os conceitos básicos envolvidos no processo de desenvolvimento de um software. Primeiramente, vimos o que é uma plataforma de desenvolvimento e quais são as principais: Desktop, Web, Mobile e Cloud. Aprendemos ainda como identificar aplicações e plataformas dentre os principais tipos existentes. Além disso, aprendemos que o contexto em que um software está inserido é de grande importância para decisões sobre o ambiente de desenvolvimento.

Em segundo lugar, vimos que linguagens de programação são importantes por vários motivos, incluindo uma melhor utilização das tecnologias disponíveis em nosso ambiente de desenvolvimento. Estudamos as principais metodologias para programação, que podem ser classificadas em seis paradigmas: o paradigma imperativo, o paradigma declarativo, o paradigma orientado a objetos, o paradigma orientado a eventos, o paradigma funcional e o paradigma lógico. Vimos também as tipagens estática e dinâmica e as tipagens fracas e fortes, bem como as possíveis combinações entre elas.

As necessidades de um cliente geralmente são acompanhadas de restrições que resultam na escolha de uma (ou mais) plataforma (s) de desenvolvimento e uma (ou mais) linguagem(ns) de programação. Dessa forma, estes são importantes critérios para a definição de um ambiente de desenvolvimento.

O próximo assunto estudado foi a instalação e configuração de um ambiente de desenvolvimento, ou IDE. Uma vez que sabemos qual ambiente de desenvolvimento é apropriado, é necessário instalar seus componentes e efetuar uma configuração adequada. Visitamos as principais IDEs que se encontram no mercado e vimos como fazer a instalação e configuração de cada uma delas. Em seguida, uma vez que temos um ambiente de desenvolvimento pronto, precisamos entender quais são as funcionalidades disponíveis que auxiliam as principais tarefas do dia a dia de um desenvolvedor na construção de um software. Isto é, estudamos os principais utilitários de um ambiente de desenvolvimento, incluindo depuração, console, explorador de pacotes, editor de texto e assim por diante.

Voltar ao sumário

Referências

ARMBRUST, Michael et al. **Above the clouds: A berkeley view of cloud computing**. 2009. EECS Department, University of California, Berkeley, 2009.

BROOKSHEAR, J G. **Ciência da Computação**. Grupo A, 2013. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582600313/. Acesso em: 06 jul. 2022.

DE OLIVEIRA, Diego Bittencourt D.; SILVA, Fabrício Machado; PASSOS, Ubiratan R C.; et al**. Desenvolvimento para dispositivos móveis**. Grupo A, 2019. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595029408/. Acesso em: 06 jul. 2022.

FERREIRA, Arthur G. **Interface de programação de aplicações (API) e web services**. Editora Saraiva, 2021. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786553560338/. Acesso em: 06 jul. 2022.

FURGERI, Sérgio. **Java 7** - Ensino Didático. Editora Saraiva, 2012. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788536505251/. Acesso em: 06 jul. 2022.

HUMBLE, Jez; FARLEY, David. **Entrega contínua**: como entregar software de forma rápida e confiável. 1. ed. Porto Alegre: Bookman, 2013. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788582601044. Acesso em: 06 jul. 2022.

KRUG, Steve. **Não me faça pensar!**: uma abordagem de bom senso à usabilidade na web. Tradução Acauan Pereira Fernandes. 2. ed. Rio de Janeiro: Alta Books, 2008.

MACHADO, Rodrigo P.; FRANCO, Márcia H I.; BERTAGNOLLI, Silvia de C. **Desenvolvimento de software III**: programação de sistemas web orientada a objetos em java (Tekne). Grupo A, 2016. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582603710/. Acesso em: 06 jul. 2022.

MASCHIETTO, Luis G. et al. **Desenvolvimento de Software com Metodologias Ágeis**. Grupo A, 2021. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556901824/. Acesso em: 06 jul. 2022.

MEMORIA, Felipe. **Design para a internet:** projetando a experiência perfeita. Rio de Janeiro: Elsevier, 2006.

MONTEIRO, Eduarda R. et al. Sistemas Distribuídos. Grupo A, 2020. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786556901978/. Acesso em: 06 jul. 2022.

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software.** Grupo A, 2021. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786558040118/. Acesso em: 06 jul. 2022.

SCHILDT, Herbert. **Java para iniciantes**. Grupo A, 2015. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582603376/. Acesso em: 06 jul. 2022.

SEBESTA, Robert. **Conceitos de Linguagens de Programação**. Grupo A, 2018. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788582604694/. Acesso em: 06 jul. 2022.

SIMAS, Victor Luiz et al. **Desenvolvimento para dispositivos móveis** - v. 2. Porto Alegre: SAGAH, 2019. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788595029774. Acesso em: 06 jul. 2022.

SOMMERVILLE, Ian. **Engenharia de software**. Tradução Maurício de Andrade. 9. ed. São Paulo: Pearson Education do Brasil; Addison-Wesley, 2011.

URMA, Raoul-Gabriel; WARBURTON, Richard. **Desenvolvimento Real De Software**. Editora Alta Books, 2021. 9786555202021. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9786555202021/. Acesso em: 06 jul. 2022.

VAZQUEZ, Carlos Eduardo; SIMÕES, Guilherme Siqueira; ALBERT, Renato Machado. **Análise de pontos de função: medição, estimativas e gerenciamento de projetos de software.** 13. ed. São Paulo: Érica, 2013. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788536518824. Acesso em: 06 jul. 2022.

Voltar ao sumário

FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

UNIVERSIDADE DE FORTALEZA (UNIFOR)

Presidência
Lenise Queiroz Rocha

Vice-Presidência
Manoela Queiroz Bacelar

Reitoria
Fátima Maria Fernandes Veras

Vice-Reitoria de Ensino de Graduação e Pós-Graduação
Maria Clara Cavalcante Bugarim

Vice-Reitoria de Pesquisa
José Milton de Sousa Filho

Vice-Reitoria de Extensão
Randal Martins Pompeu

Vice-Reitoria de Administração
José Maria Gondim Felismino Júnior

Diretoria de Comunicação e Marketing
Ana Leopoldina M. Quezado V. Vale

Diretoria de Planejamento
Marcelo Nogueira Magalhães

Diretoria de Tecnologia
José Eurico de Vasconcelos Filho

Diretoria do Centro de Ciências da Comunicação e Gestão
Danielle Batista Coimbra

Diretoria do Centro de Ciências da Saúde
Lia Maria Brasil de Souza Barroso

Diretoria do Centro de Ciências Jurídicas
Katherinne de Macêdo Maciel Mihaliuc

Diretoria do Centro de Ciências Tecnológicas
Jackson Sávio de Vasconcelos Silva

AUTOR
RONALDO PINHEIRO GONÇALVES JUNIOR

Graduado em Ciência da Computação (2011) e Mestre em Informática Aplicada (2015), ambos pela Universidade de Fortaleza, e Doutorando em Engenharia de Software pela University of Texas at Dallas. Possui experiência em Desenvolvimento Full Stack de aplicativos para Web e Nuvem. Docente do Centro de Ciências Tecnológicas da Universidade de Fortaleza e participante de projetos com foco nas áreas de Computação em Nuvem, Engenharia de Requisitos e Sistemas de Recomendação.

RESPONSABILIDADE TÉCNICA

UNIVERSIDADE DE FORTALEZA ENSINANDO E APRENDENDO
VRE Vice-Reitoria de Ensino de Graduação e Pós-Graduação
Universidade de Fortaleza | EAD

COORDENAÇÃO DA EDUCAÇÃO A DISTÂNCIA

Coordenação de Ensino e Recursos EAD
Andrea Chagas Alves de Almeida

Supervisão de Planejamento Educacional
Ana Flávia Beviláqua Melo

Supervisão de Recursos EAD
Francisco Weslley Lima

Supervisão de Ensino e Recurso
Liliane Rios Cajazeiras Mendes

Analista Educacional
Lara Meneses Saldanha Nepomuceno

Projeto Instrucional
Maria Mirislene Vasconcelos

Revisão Gramatical
Janaína de Mesquita Bezerra

Identidade Visual / Arte
Francisco Cristiano Lopes de Sousa

Editoração / Diagramação
Régis da Silva Pereira

Produção de Áudio e Vídeo
José Moreira de Sousa
Pedro Henrique de Moura Mendes

Programação / Implementação
Renan Alves Diniz
Thaís Rozas Teixeira