



CARREIRA E LINGUAGEM

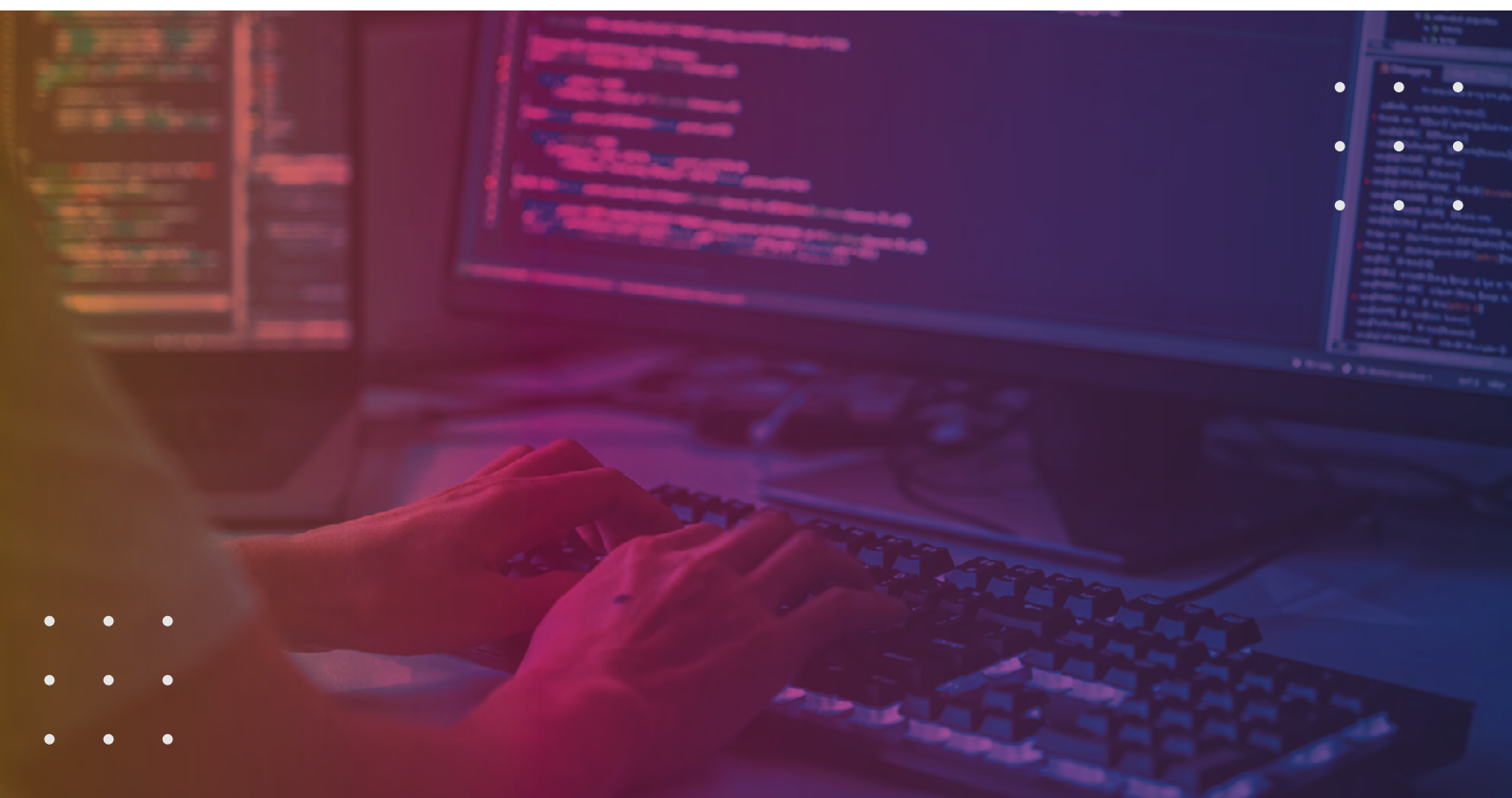
SUMÁRIO

A Carreira na área de Tecnologia da Informação.....	03
As várias áreas dentro da TI.....	04
O desenvolvedor de software	05
Desenvolvedor <i>front-end</i>	05
Desenvolvedor <i>back-end</i>	06
Outros tipos de profissionais desenvolvedores	06
 O que é preciso saber para se tornar um desenvolvedor	 07
Níveis de senioridade	08
Como se desenvolver profissionalmente	09
O dia a dia de um desenvolvedor de software	10
 Introdução à linguagem java	 12
Como o Java surgiu	13
Como o Java funciona.....	15
Características da linguagem	16
Java é uma linguagem com tipos de dados definidos	16
Operadores Java.....	18
Ferramentas de desenvolvimento	19
Orientação à objetos	20
 Referências	 24



A CARREIRA NA ÁREA DE TECNOLOGIA DA INFORMAÇÃO

Em algum momento do seu dia, você já deve ter ouvido dizer que a carreira de TI está aquecida, ou que existe uma alta demanda de profissionais para a área de TI. Talvez isso tenha sido o motivo de você estar aqui, neste momento, lendo este e-book, preparado para a introdução à carreira de desenvolvedor.



De fato, os profissionais de tecnologia da informação passam por um momento de alta demanda, e a falta de oferta de mão de obra fez com que os salários aumentassem e a profissão entrasse no radar das grandes mídias.

Mas você sabe o que um profissional de TI faz? Quais são os conhecimentos necessários para ingressar no mercado de trabalho como um profissional de tecnologia? A partir de agora suas dúvidas serão esclarecidas. Vamos lá!

As várias áreas dentro da TI

Trabalhar na área de tecnologia da informação não significa necessariamente que você precisa aprender a escrever programas de computador. Existem profissionais que são responsáveis por manter a infraestrutura da empresa, que vai desde a configuração das máquinas dos colaboradores até a preparação dos ambientes de tecnologia dos clientes, profissionais que coordenam equipes, que fazem recrutamento e seleção e assim por diante.



Quando falamos de desenvolvedores, podemos ainda categorizá-los em desenvolvedores *front-end*, *back-end* e *full stack*, termos que você compreenderá mais adiante. Podemos ainda ter analistas de testes, responsáveis por escrever casos de testes em ferramentas especializadas. Enfim, são incontáveis as possibilidades de trabalho.

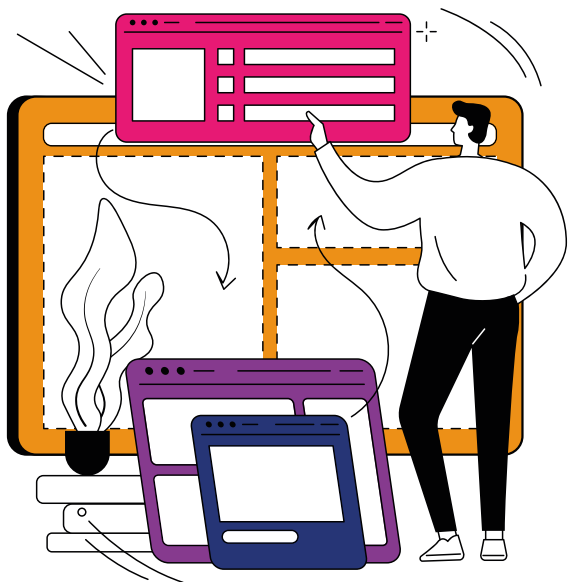
Também vale ressaltar que as suas experiências profissionais anteriores sempre acabam agregando algo à sua carreira de desenvolvedor.

Um desenvolvedor de software é, em essência, alguém pago para resolver problemas através de tecnologia. Portanto, a experiência de vida soma muitos pontos quando aliada à habilidade de escrever programas que vão atender a alguma necessidade de alguma pessoa ou empresa.

O desenvolvedor de software

Aprofundando um pouco o tema da profissão de desenvolvedor de software, podemos classificar o desenvolvedor em três categorias principais: desenvolvedor *front-end*, desenvolvedor *back-end* e desenvolvedor *full stack*. Antes de entender melhor o que faz cada categoria, precisamos entender o básico sobre a anatomia de um sistema. Confira a seguir.

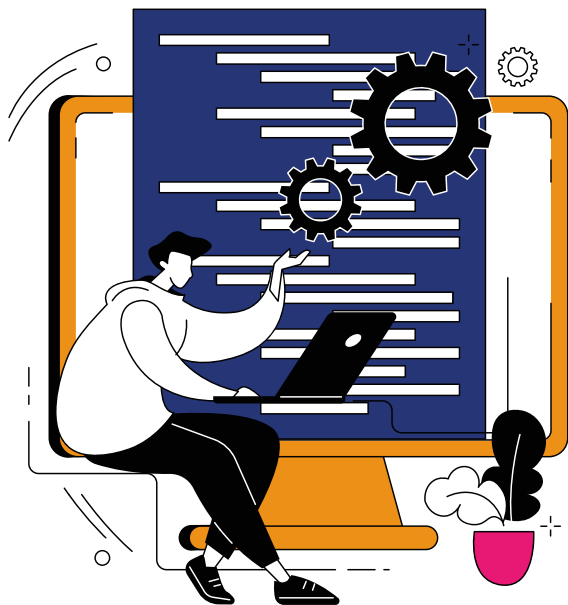
Desenvolvedor *front-end*



Em termos atuais, um sistema é um conjunto de camadas que se comunicam para realizar uma ou mais tarefas. Dentro de um sistema web, possuímos uma camada responsável por exibir a interface gráfica do nosso programa, registrar as interações do usuário e enviar essas interações para uma outra camada. Esta camada é chamada de *front-end*.

O profissional que atua no *front-end* é responsável por escrever o código de marcação, estilização e programação da inteligência da interface gráfica. Também é responsável pelas questões de localização e acessibilidade. O programa deve ser acessível por todos os usuários e deve se adaptar ao local (região, país, etc.) em que é acessado, exibindo corretamente os formatos de data, moeda, língua, etc.

Desenvolvedor *back-end*




A camada que recebe essas interações, realiza os processamentos necessários, aplica as regras do programa, salva e busca informações em bancos de dados e devolve essas informações processadas é chamada de *back-end*.

O profissional de *back-end* é responsável por criar e configurar os sistemas de servidor, já banco de dados e comunicação entre servidor e cliente, que é o programa que consome esses dados, geralmente, é o *front-end*. Também cria os comandos de consulta e gravação de dados nos bancos de dados.


Outros tipos de profissionais desenvolvedores



Podemos adicionar à lista de camadas a camada de infraestrutura, já que hoje em dia é muito comum que nossa infraestrutura seja programada através de código, chamamos de IAC, ou *Infrastructure as Code* (infraestrutura como código). O profissional que atua nesta camada, programa automatizações de configuração de ambientes, administração e monitoramento dos sistemas, entre outras coisas.




Há também, o profissional *full stack*, que atua em todas as camadas citadas anteriormente, tendo capacidade de desenvolver um sistema de ponta a ponta, desde a sua interface gráfica até o *deployment* (implantação do sistema), passando pela programação do *back-end*, definição do banco de dados, etc.



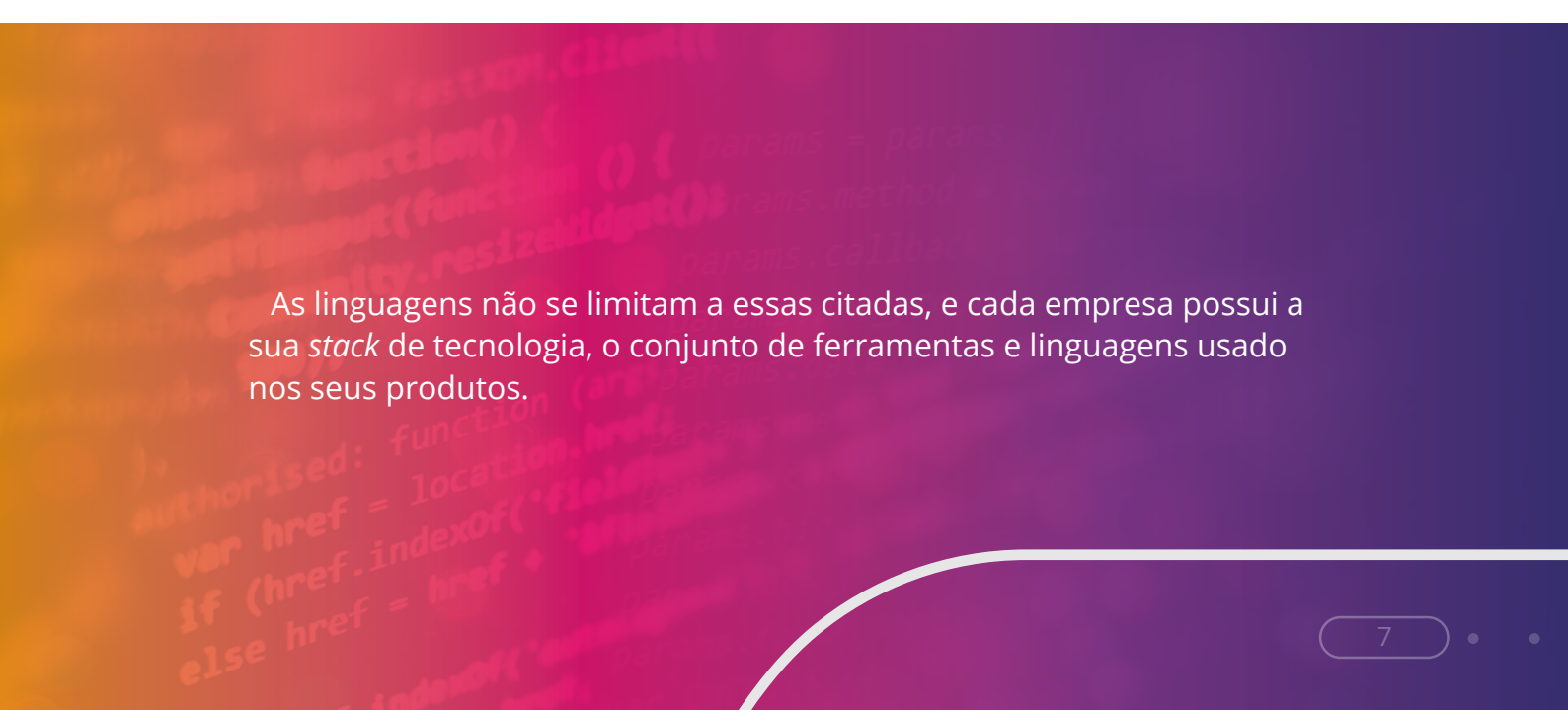

O que é preciso saber para se tornar um desenvolvedor

Hoje em dia, existem dezenas de linguagens, divididas entre linguagens de marcação, estilização e programação.


Cada profissional necessita saber uma ou mais linguagens relacionadas à sua área de atuação.



Por exemplo: um desenvolvedor *front-end* atuará com HTML (*Hypertext Markup Language*, ou linguagem de marcação de hipertexto), CSS (*Cascading Stylesheet*, ou folha de estilo em cascata) e JavaScript. Já um desenvolvedor *back-end* atuará com linguagens como Java (que falaremos mais adiante), C#, Ruby, Go Lang e até JavaScript.



As linguagens não se limitam a essas citadas, e cada empresa possui a sua *stack* de tecnologia, o conjunto de ferramentas e linguagens usado nos seus produtos.



Mais importante do que a linguagem de programação, um desenvolvedor precisa desenvolver o raciocínio lógico, a capacidade de dividir um problema maior em diversos problemas menores e ter pensamento sistêmico, ou seja, saber identificar conjuntos de funcionalidades que se relacionam e fazem parte de um mesmo contexto.

Níveis de senioridade

Podemos chamar de senioridade o nível profissional de determinada pessoa em uma determinada empresa. No Brasil, costumamos categorizar os profissionais em 3 grupos: júnior, pleno e sênior. Esta categorização varia de empresa para empresa, algumas utilizam números romanos (desenvolvedor de software I, desenvolvedor de software II, etc), outras incluem, além de seniores, os desenvolvedores especialistas.

O que determina a senioridade de um profissional também é bastante subjetivo, mas podemos considerar os seguintes cenários como sendo os mais comuns:

Estagiário/trainee: precisa de constante acompanhamento de um profissional experiente para desenvolver suas tarefas. Conhece o básico de lógica de programação e linguagens.

Desenvolvedor júnior: consegue desenvolver um sistema com mais autonomia, mas ainda precisa de ajuda de um profissional experiente. Tem bom domínio de lógica de programação, conhece bem as linguagens com que trabalha, mas não consegue aplicar as melhores técnicas de programação e organização de código. Já consegue resolver problemas de dificuldade menor sozinho.

Desenvolvedor pleno: domina a linguagem e lógica de programação. Consegue desenvolver uma aplicação completa, seguindo as orientações do projeto. Aplica boas práticas de programação e organização de código e sabe resolver problemas de pequena e média complexidade e até alguns de maior grau. Tem alto grau de autonomia para trabalhar, mas precisa de auxílio para saber o que fazer e quando fazer.

Desenvolvedor sênior: domina a linguagem de programação, lógica e arquitetura. Sabe resolver problemas mais complexos, e consegue diagnosticar problemas antes mesmo deles acontecerem. A principal tarefa de um desenvolvedor sênior é guiar a equipe para que o projeto seja bem escrito, manutenível e performático e manter todos da equipe em constante desenvolvimento.

Como se desenvolver profissionalmente

Tornar-se um bom desenvolvedor requer horas e horas de mão na massa. Não há segredo, **não há atalho**. Quanto mais código você escrever, maior vai ser o seu domínio da linguagem. Quanto mais código você ler, melhor vai ser o seu entendimento de lógica de programação e pensamento computacional.



Não tenha medo de cometer erros e aprenda a aprender com eles. Aprenda como e onde pesquisar os erros que acontecem durante sua jornada. A habilidade de entender e corrigir problemas é uma das mais valorizadas em um profissional, mas é uma das mais difíceis de se desenvolver, porque errar pode ser frustrante.

Quanto mais erros você cometer e aprender a resolver, melhor vai ser a sua capacidade de escrever códigos melhores e com menos problemas.

Faça projetos. Faça clones, mas dê uma chance a você mesmo de arriscar fazer projetos autorais. Pense em projetos pequenos no começo e aumente a complexidade conforme for ganhando confiança. Faça networking. Participe de comunidades de desenvolvedores. A habilidade de se comunicar em equipe é indispensável hoje em dia.

O dia a dia de um desenvolvedor de software

Desenvolvedores são contratados para encontrar soluções em tecnologia para problemas da vida real. Boa parte do tempo é gasto em planejar como atuar em uma tarefa. Outra parte é passada fazendo revisão de código para resolver bugs ou analisar uma nova implementação. Uma parte menor do tempo é gasta com a escrita de código efetivamente.

O dia de trabalho geralmente começa com a definição de uma ou mais tarefas que serão desenvolvidas. Em um ambiente de trabalho padrão, usam-se frameworks de desenvolvimento ágil, como Scrum e Kanban, em que as tarefas são quebradas em pedaços menores e divididas entre as equipes e os desenvolvedores.



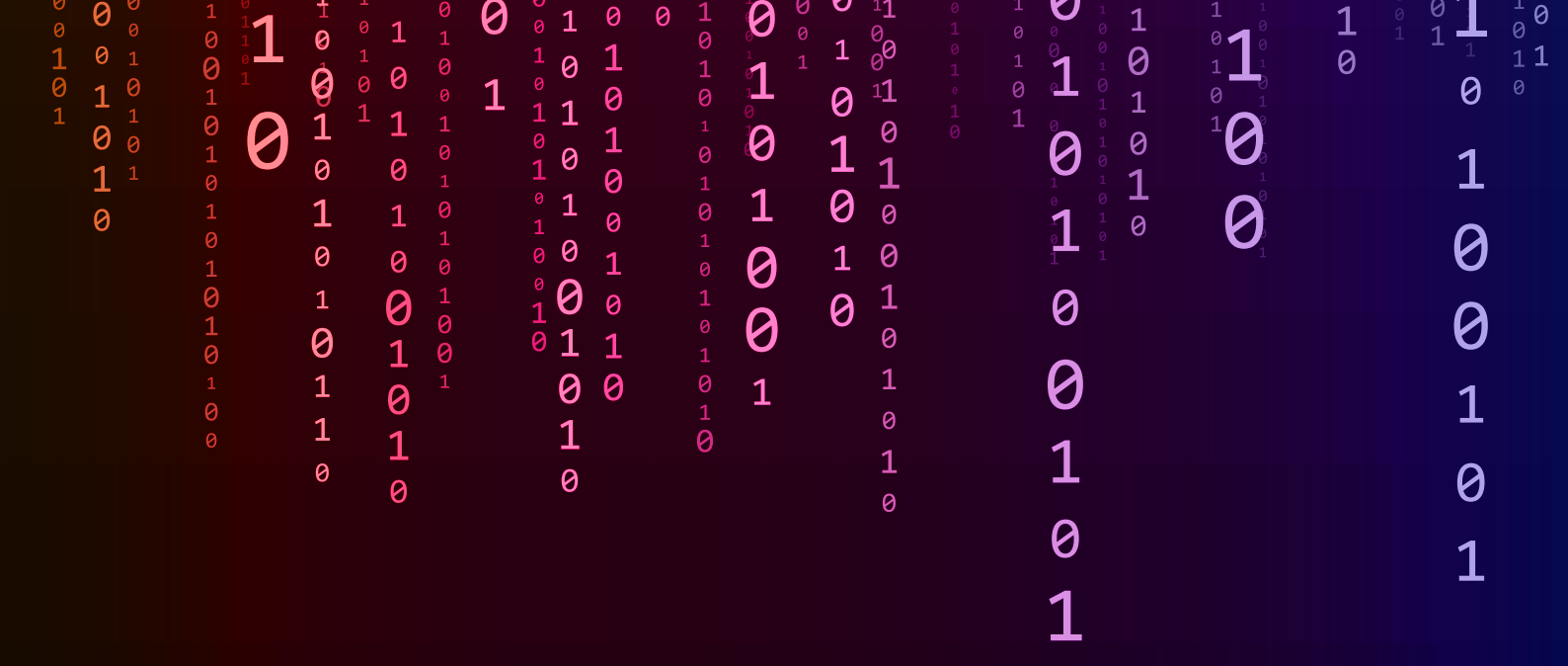
INTRODUÇÃO À LINGUAGEM JAVA

Para começarmos a falar sobre programar computadores, precisamos primeiro entender o que é uma linguagem de programação. E a história do Java faz parte da história da computação. De forma resumida, vamos entender o contexto histórico da sua criação.

No início da era do desenvolvimento, os primeiros computadores eram programados através de botões e fios, que interligavam circuitos especialistas para realizar um determinado cálculo. Toda vez que o cálculo precisava ser alterado, precisava-se de um programador para realizar as alterações das conexões físicas do computador. Nesta época, os computadores chegavam a ter tamanhos de salas inteiras.

Quando as máquinas passaram a ser portáteis, criou-se um mercado de computadores pessoais que precisavam ser programados com linguagem a nível de máquina. O problema disso é que, para cada processador diferente, existia um dicionário Assembly diferente (a linguagem Assembly era a linguagem de instruções que um determinado processador entendia).





Para facilitar a vida do programador, novas linguagens de mais alto nível foram sendo desenvolvidas. Dentre tantas, a linguagem C ficou mais conhecida por ser considerada multiplataforma, uma única base de código poderia ser utilizada em diversos sistemas operacionais e processadores diferentes.

Esta característica era possível porque a linguagem C, por ser uma linguagem de alto nível, mais próxima da linguagem humana, precisava passar por uma ferramenta para ser transformada em linguagem de máquina, os zeros e uns que o computador entende, e, para cada sistema operacional, existia um compilador que criava o executável de acordo com as instruções daquela determinada máquina.

Outras linguagens surgiram ao longo dos anos, cada uma com um objetivo diferente, mas com estrutura similar. A própria linguagem C evoluiu para C++, que adotava o paradigma de programação orientada a objetos.

Outra linguagem de programação criada dentro desse contexto e que também utilizava a orientação a objetos como base foi a linguagem Java.

Como o Java surgiu

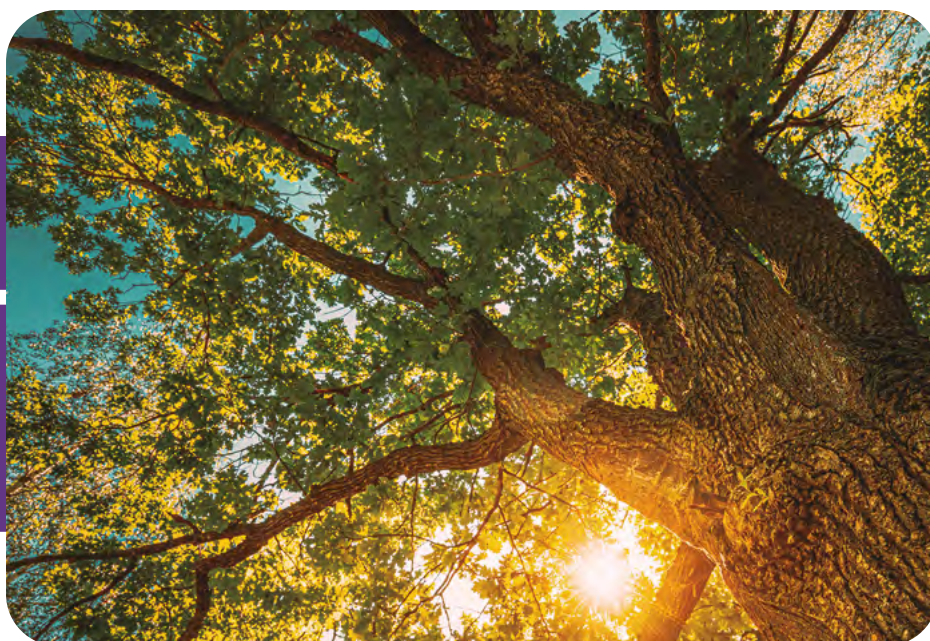
No começo dos anos 1990, uma empresa chamada Sun Microsystems iniciou um projeto de desenvolvimento de uma linguagem de programação que pudesse ser executada em qualquer dispositivo eletrônico, não só os computadores. O projeto Green, como foi batizado, tinha em sua equipe de mentores o engenheiro James Gosling, considerado o pai da linguagem.

O projeto Green deu origem a um dispositivo chamado **starseven**, um controle remoto com interface digital, projetado para ser uma interface de comunicação com dispositivos eletrônicos. Naquela época, acreditava-se na possibilidade de controlar o conteúdo em televisores e vídeo por demanda, algo que hoje é considerado trivial. O projeto não seguiu devido às limitações e dificuldades tecnológicas para implementação à época.



Fonte: BONFIM (2021).

Logo em seguida, inicia-se a era da internet, criando possibilidades de interatividade entre máquinas, algo que os engenheiros do projeto Green buscavam inicialmente com o starseven.



James Gosling havia especificado uma linguagem de programação para o desenvolvimento do dispositivo, que inicialmente foi batizada de Oak (carvalho). Curiosamente, este nome foi dado porque Gosling observava uma árvore desta espécie pela sua janela enquanto trabalhava.



Com o surgimento e o crescimento da internet, bastou apenas que a linguagem Oak fosse adaptada para funcionar nos computadores em 1995. A linguagem foi rebatizada para Java, acredita-se que o nome foi dado porque a equipe que trabalhou no projeto consumia muito café oriundo da ilha de Java, na Indonésia.

O Java já era uma linguagem preparada para trabalhar em dispositivos diferentes. A adaptação e adoção do Java na internet foi muito rápida, com diversos fornecedores de tecnologia anunciando suporte à linguagem.

Como o Java funciona

O Java é uma linguagem de programação compilada. Mas, diferentemente da linguagem C, o código Java não é transformado em um código binário executável.

Inicialmente, o código passa por um processo de compilação para uma linguagem intermediária, chamada de *bytecode*. Este *bytecode* é, enfim, interpretado por uma máquina virtual.

Quando trabalhamos com Java, nos deparamos com três siglas bastante comuns: JDK, JVM e JRE. É muito interessante saber o que cada sigla desta significa, pois durante a carreira de desenvolvedor você as verá com bastante frequência:

• • • • • • •

- JVM (Java Virtual Machine): ambiente de máquina virtual que faz a tradução do *bytecode* para a linguagem de máquina. Cada ambiente que roda Java possui sua versão da JVM.
- JRE (Java Runtime Environment): ferramenta de tempo de execução Java. É um pacote com classes e bibliotecas básicas do Java, além da JVM.
- JDK (Java Development Kit): o conjunto de ferramentas para desenvolvimento de aplicações Java, composto por compilador, JVM e outros recursos da linguagem.

• • • • • • •

Características da linguagem

O Java é uma linguagem de programação orientada a objetos. Isso significa que criamos conjuntos de propriedades e comportamentos e os agrupamos em uma estrutura que chamamos de classe.

Toda aplicação Java possui uma classe principal com um método de inicialização chamado *main*, que é o ponto de partida do processo de compilação. A estrutura da classe inicial em um projeto novo de Java é a que segue:

```
public class Exemplo {  
    public static void main(String[] args) {  
        System.out.printf("Olá, mundo");  
    }  
}
```

Note que, dentro do método *main*, é usado um comando nativo do Java, responsável pela exibição de uma mensagem ("Olá, mundo") no console da aplicação. Outro ponto importante é que toda expressão Java precisa ser terminada com ponto e vírgula (";").

Existem diversas ferramentas que já vêm incluídas na linguagem, dentro do JRE. Outras tantas são adicionadas conforme a necessidade, através de gerenciadores de pacotes, você verá o que é um gerenciador de pacotes durante os estudos.

Vamos agora passar por algumas características, mas todos os pontos serão abordados com profundidade posteriormente.

Java é uma linguagem com tipos de dados definidos

Uma outra característica da linguagem é que as propriedades das classes possuem um tipo predefinido de dado. Isso significa que uma propriedade que guarda um dado do tipo número só poderá guardar números durante todo o seu ciclo de vida. Chamamos esta característica de tipagem forte.

Os tipos de dados podem ser classificados em tipos primitivos e objetos. No Java, há 8 tipos primitivos:

- **byte:** armazena valor numérico inteiro entre -128 e 127;
- **short:** armazena valor numérico inteiro entre -32.768 e 32.767;
- **int:** armazena valor numérico inteiro entre -2.147.483.648 e 2.147.483.647;
- **long:** armazena números inteiros entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807;
- **float:** armazena números decimais com precisão de até 7 casas decimais;
- **double:** armazena números decimais com precisão de até 15 casas decimais;
- **boolean:** armazena valores booleanos *true* e *false*; e
- **char:** armazena um único caractere ou letra.

Para criar uma propriedade com um tipo definido, você utilizará a seguinte sintaxe:

```
public class Exemplo {  
    public static void main(String[] args) {  
        int idade = 18;  
    }  
}
```

Primeiro, você deve indicar qual o tipo de dado a propriedade irá armazenar (int). Depois, nomear a propriedade (idade) e atribuir o valor a ela (18).

A outra categoria de tipos de dados (a de objetos) possui objetos que representam tipos de dados primitivos, com o Integer, e outros objetos que são amplamente utilizados e quase se confundem com tipos primitivos, como é o caso do objeto *String* (que representa uma cadeia de caracteres).

```
public class Exemplo {  
    public static void main(String[] args) {  
        String nome = "José";  
        String cidade = "Florianópolis";  
    }  
}
```



Um exemplo de utilização dos objetos como tipo de dados é o que segue:

Os objetos que representam dados primitivos, como é o caso do Double, Integer e Float, tem a sua grafia iniciada com letra maiúscula e trazem consigo diversas outras propriedades que um tipo primitivo não possui, como funções de conversão de tipos, funções de comparação, entre outros, que serão apresentados durante os estudos. A seguir, entenda mais sobre os operadores Java.

Operadores Java

No exemplo anterior, você conferiu a declaração de uma propriedade e a atribuição de um valor inicial (`idade = 18`). O sinal de igualdade indica que o valor à direita está sendo atribuído à propriedade à esquerda, o sinal de igualdade é um exemplo de operador de atribuição.

No lado direito da operação, você pode ter tanto valores quanto expressões que utilizam outros tipos de operadores. Entenda mais com o exemplo a seguir:

```
public class Exemplo {  
    public static void main(String[] args) {  
        int idade = 17 + 1;  
    }  
}
```

Neste exemplo, foi utilizado o operador matemático "+" para indicar uma soma, que é lida antes da atribuição do valor à propriedade `idade`.

Existem também os operadores que fazem comparações entre dois valores, operadores de comparação. O retorno dessa comparação é sempre um valor booleano. Confira em seguida:

```
public class Exemplo {  
    public static void main(String[] args) {  
        int minhaIdade = 31  
        boolean maiorDeIdade = minhaIdade >= 18;  
    }  
}
```


Há diversos tipos de operadores que ainda serão abordados neste estudo, mas confira no quadro a seguir alguns dos principais modelos de operadores.

Operador	Descrição	Tipo	EXEMPLO
=	atribuir à	atribuição	int idade = 18
-	subtrair	aritmético	int subtracao = 20 - 2
+	somar	aritmético	int soma = 15 + 3
*	multiplicar	aritmético	int multiplicacao = 9 * 2
/	dividir	aritmético	int divisao = 36 / 2
==	é igual a	comparação	25 == 25
>	é maior que	comparação	25 > 12
<	é menor que	comparação	30 < 50
>=	é maior ou igual a	comparação	31 >= 30
<=	é menor ou igual a	comparação	29 <= 31

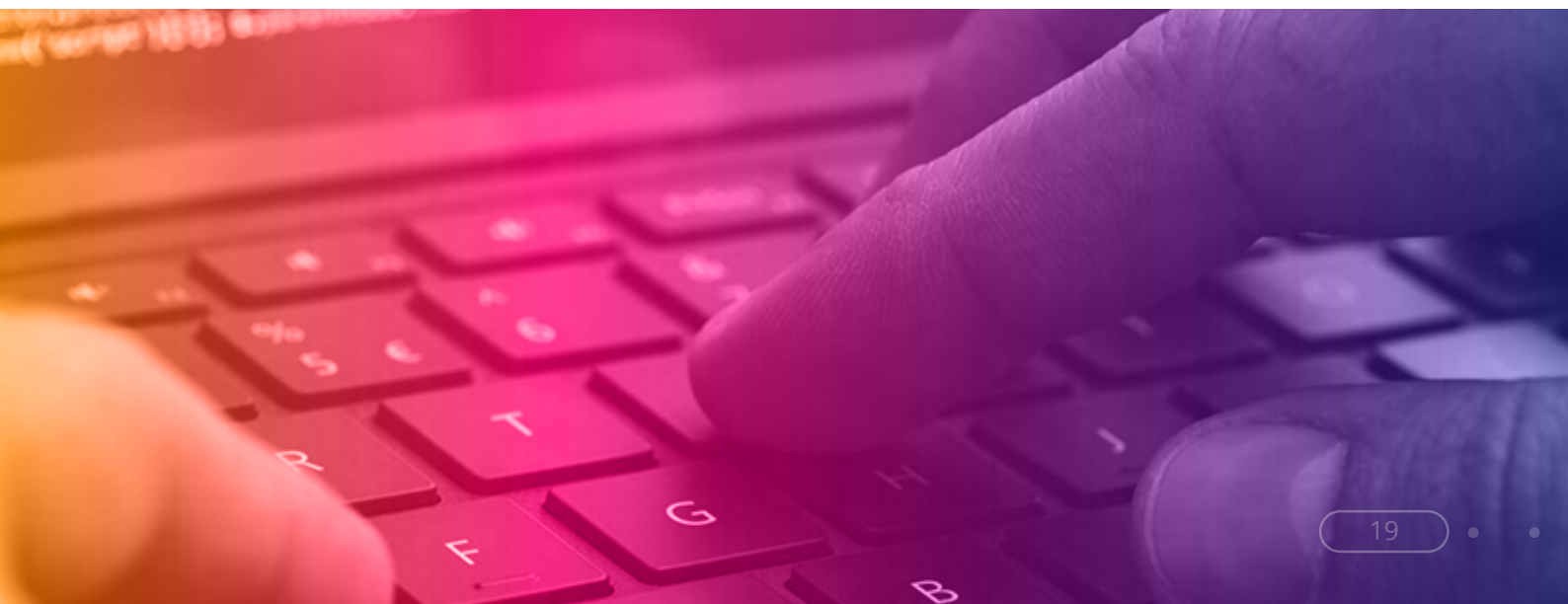
Quadro 1 - Tipos de operadores

Fonte: do Autor (2022)

Ferramentas de desenvolvimento

Tecnicamente, é possível criar um programa Java utilizando somente o aplicativo de bloco de notas e o terminal do computador. Um arquivo Java possui a extensão .java e pode ser compilado utilizando o comando javac no terminal.

Na prática, lança-se mão de ferramentas de desenvolvimento que auxiliam com a escrita e a verificação do código em tempo real. Existem duas categorias básicas de ferramentas: os editores de código e o ambiente de desenvolvimento integrado (IDE, ou *Integrated Development Environment*).



Como exemplo de editor de código, o mais utilizado é o *Visual Studio Code*, da Microsoft. É uma ferramenta de código aberto e uso gratuito muito poderosa. Pode-se configurar o editor adicionando plugins criados pela comunidade de desenvolvedores.

Já as IDEs são ferramentas bastante robustas, com diversas funcionalidades que auxiliam na criação de novos arquivos, na correta nomeação e definição das classes, na utilização de boas práticas de código, entre outras coisas. Também possui, de forma integrada, ferramentas de compilação e execução de código. Como exemplos mais comuns de IDEs para desenvolvimento Java, podemos citar o IntelliJ, NetBeans e o Eclipse.

Orientação à objetos

A linguagem Java é uma linguagem de programação orientada a objetos. Os objetos são estruturas de dados criadas a partir das classes. As classes são uma espécie de molde de um objeto, que define propriedades e comportamentos que o objeto terá quando for criado. Observe o código a seguir:

```
public class Pessoa {  
    String nome = "José";  
    int idade = 31;  
  
    public void saudacao() {  
        System.out.println("Olá, eu sou o José");  
    }  
}
```

O código descrito é a definição de uma classe Pessoa. A partir dessa classe, podemos criar vários objetos. Cada um desses objetos chamamos de instâncias.

Para criar uma instância de um objeto a partir de uma classe, usamos a palavra reservada `new` da linguagem. Quando o código é interpretado, ao chegar à linha que possui a construção de uma nova Pessoa (`new Pessoa`), uma estrutura concreta com as propriedades declaradas dentro da classe Pessoa passará a existir na aplicação:



```
public class Exemplo {  
    public static void main(String[] args) {  
        Object pessoa1 = new Pessoa();  
    }  
}
```

A propriedade **pessoa1** é do tipo *Object*, ou seja, pode receber qualquer valor que seja um objeto, que é o caso da instância de pessoa que foi criada a partir do comando `new Pessoa()`.

A partir da criação de `pessoa1`, podemos ter acesso às propriedades e comportamentos que foram definidos dentro do molde de uma Pessoa (dentro da classe Pessoa). Mas, para que o Java saiba que `pessoa1` é uma instância de Pessoa (ou seja, um objeto criado a partir de um comando `new Pessoa()`), precisamos dizer que o tipo da propriedade `pessoa1` é Pessoa:

```
public class Exemplo {  
    public static void main(String[] args) {  
        Pessoa pessoa1 = new Pessoa();  
    }  
}
```

E agora podemos acessar a propriedade da `pessoa1` com a notação de ponto. Vamos imprimir no console o valor da propriedade `nome` do objeto `pessoa1`:

```
public class Exemplo {  
    public static void main(String[] args) {  
        Pessoa pessoa1 = new Pessoa();  
  
        System.out.println(pessoa1.nome);  
    }  
}
```

Na orientação a objetos, são criadas classes, que definem como as instâncias serão estruturadas quando forem criadas.

Um objeto é uma representação em código de algo do mundo real. Assim como os objetos na vida real, os objetos na programação agrupam qualidades inerentes àquilo que estamos modelando.

No exemplo de código que você conferiu, uma Pessoa tem as propriedades nome e idade. Não precisamos parar por aí, afinal, uma pessoa pode ter uma quantidade gigantesca de outras características que podemos mapear em propriedades.

Vale ressaltar que, quando você está trabalhando em uma aplicação onde precisa modelar coisas do mundo real, a regra é usar apenas as características que fazem sentido dentro da sua aplicação. A Pessoa do exemplo poderia ter uma propriedade peso do tipo float, que é um número decimal. Mas será que essa propriedade seria necessária dentro do projeto? A habilidade de modelar objetos vem com o tempo e a experiência em desenvolvimento.

O verdadeiro poder da programação orientada a objetos é liberado quando começamos a trabalhar com o que é chamado de pilares da POO: **abstração, herança, encapsulamento e polimorfismo**. Essas propriedades fazem com que os códigos escritos com a abordagem orientada a objetos sejam seguros, fáceis de entender, reutilizáveis e manuteníveis.

Todas as características do paradigma orientado a objetos serão abordados no decorrer dos estudos de forma detalhada, mas, de forma resumida, podemos dizer que:

- **Abstração:** a abstração é a representação de uma estrutura complexa de maneira simplificada, como quando, por exemplo, modelamos uma pessoa selecionando apenas as características que realmente necessitamos em nosso exemplo.
- **Encapsulamento:** é a capacidade de uma classe guardar e proteger o acesso às suas propriedades, de modo que somente as propriedades desejadas sejam expostas ao programa.



- **Herança:** é a capacidade de uma classe herdar de outra as suas propriedades e métodos.
- **Polimorfismo:** é a capacidade de um método declarado em uma classe mãe (superclasse) ter seu comportamento redefinido nas classes que herdam este método.



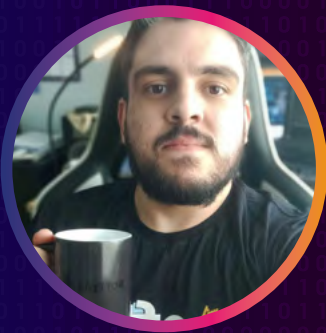
E então, gostou de conhecer sobre o dia a dia do TI? Sobre as diversas possibilidades de carreira nessa área tão promissora? E ainda teve a chance de aprender mais sobre a linguagem Java, entender suas características, e até explorar um exemplo de código. Seus estudos não acabam por aqui! Continue seus estudos e conheça ainda mais desse universo que com certeza pode mudar seus horizontes.



REFERÊNCIAS

BONFIM, Fernando. **A história do Java**. Disponível em: <https://tecnologiaeinformacao.netlify.app/java/java-history-ptbr.html>. Acesso em: 2 jun. 2022.

DEITEL, P.; DEITEL, H. **JAVA: como programar**. 10. Editora Pearson, 2003.



Michael Nascimento

Desenvolvedor especializado em front-end e JavaScript, com experiência em desenvolvimento de software enterprise no modelo SaaS. Mentor educacional no LAB365 do SENAI/SC.

SENAI <LAB365>