

E-BOOK GRATUITO

HTML e Java do zero

Crie a sua primeira página e o seu primeiro programa no VS Code, com três projetos de verdade.

index.html

```
1 <body>
2   <h1>Olá, mundo!</h1>
3 </body>
```

Ola.java

```
1 public class Ola {
2     public static void main(String[] args) {
3         System.out.println("Olá, mundo!");
4     }
5 }
```



Gregório Karabachian
@developers_gk

HTML e Java do zero

Versão 1.0, outubro de 2026

© 2026 Gregório Karabachian · @developers_gk

Este e-book é gratuito. Pode compartilhar o arquivo à vontade, do jeito que ele está. Só não pode vender nem tirar os créditos.

Todos os códigos foram rodados e testados com Java 21 e com o Google Chrome antes de entrar no livro. Nomes de empresa, endereços e telefones dos exemplos são inventados.

VS Code, Java, Chrome e as outras marcas citadas pertencem aos seus donos e aparecem aqui só para identificar as ferramentas.

Achou um erro ou ficou com dúvida? Manda no direct do **@developers_gk** no Instagram.

Sumário

	Antes de começar	4
01	Java não é JavaScript	6
02	Preparando o VS Code	9
PARTE 1 • HTML		
03	Sua primeira página	13
04	As tags do dia a dia	16
05	Estrutura que o Google entende	20
06	Formulários	23
07	Um pouco de CSS	26
08	Projeto 1: a página da padaria	30
PARTE 2 • JAVA		
09	Seu primeiro programa em Java	39
10	Variáveis: guardando informação	43
11	Decisões com if e switch	47
12	Repetições com for e while	50
13	Listas: arrays e ArrayList	53
14	Métodos: dando nome às tarefas	56
15	Classes e objetos	58
16	Conversando com o usuário	61
17	Projeto 2: o caixa da padaria	64
PARTE 3 • JUNTANDO OS DOIS		
18	Como o navegador conversa com o Java	69
19	Projeto 3: seu servidor em Java	72
20	Próximos passos	81
	Colinha de bolso	83

// antes de começar

Bem-vindo ao seu **primeiro código**

Este livro é para quem nunca programou, ou para quem já começou, travou e desistiu. Você não precisa saber nada antes. Só precisa de um computador e de um pouco de paciência.

O que você vai construir

Em vez de decorar teoria, você vai aprender fazendo três projetos, todos sobre a mesma padaria de bairro. Um puxa o outro:

- 1. A página da padaria**, com cardápio, endereço e formulário de contato, feita em HTML e CSS.
- 2. O caixa da padaria**, um programa em Java que monta o pedido e imprime o cupom.
- 3. O servidor da padaria**, um programa em Java que coloca a página no ar e recebe as mensagens do formulário.

No final, você vai entender como uma página e um programa conversam, que é exatamente o que acontece em quase todo site e aplicativo que você usa.

Como ler este livro

- **Digite o código, não copie e cole.** Parece perda de tempo, mas é digitando que a mão aprende. E os erros de digitação vão te ensinar mais do que qualquer explicação.
- **Rode tudo.** Cada exemplo foi testado antes de entrar no livro. Se o seu não funcionar, compare letra por letra: a diferença está lá.

- **Faça os desafios.** Eles estão no fim de cada capítulo e são pequenos de propósito.
- **Um capítulo por dia.** São 20 capítulos. Em menos de um mês, os três projetos estão prontos.

As caixas que você vai encontrar

DICA

Um atalho, um jeito mais fácil ou algo que os profissionais fazem no dia a dia.

ATENÇÃO

Um erro comum, uma pegadinha ou algo que pode dar problema mais tarde.

DESAFIO

Um exercício curto para você fazer sozinho antes de seguir.

Java não é JavaScript

O nome confunde quase todo mundo que está começando. Antes de escrever a primeira linha, vale entender o que cada linguagem faz e onde ela entra.

Pense numa casa

Um site é parecido com uma casa. Cada linguagem cuida de uma parte dela:

- **HTML é a estrutura:** as paredes, as portas, os cômodos. Ele diz o que existe na página: um título, um parágrafo, uma foto, um botão.
- **CSS é o acabamento:** a pintura, os móveis, a decoração. Ele diz como cada coisa aparece: cor, tamanho, posição.
- **JavaScript é a parte elétrica:** você aperta o interruptor e a luz acende. Ele faz a página reagir quando alguém clica, digita ou rola a tela.

E o Java? **O Java é a companhia de luz e água.** Ele não mora dentro da casa: fica do outro lado da rua, num computador chamado servidor, e entrega para a casa o que ela precisa. É o Java que guarda os pedidos de uma loja, confere a senha de quem faz login ou calcula o frete.

Navegador e servidor

Toda vez que você abre um site, duas máquinas conversam. O seu navegador pede, o servidor responde:

Navegador

O Chrome, no seu celular ou computador. Entende HTML, CSS e JavaScript.



Servidor

Um computador ligado o tempo todo. Roda Java (ou outra linguagem) e guarda os dados.

LINGUAGEM	O QUE FAZ	ONDE RODA
HTML	Estrutura e conteúdo da página	Navegador
CSS	Cores, tamanhos e posições	Navegador
JavaScript	Interação na página	Navegador (e servidor)
Java	Programas, sistemas, servidores e apps Android	Computador e servidor

Então por que os nomes são parecidos?

Por marketing. O JavaScript nasceu em 1995 com outro nome e foi rebatizado para pegar carona na fama do Java, que era a grande novidade do momento. Por dentro, são linguagens diferentes, criadas por empresas diferentes, para problemas diferentes. Tem uma piada antiga entre programadores que resume bem: **Java está para JavaScript assim como carro está para carpete.**

HTML é programação?

Não exatamente. HTML é uma **linguagem de marcação**: ela descreve o que é cada coisa, mas não toma decisões nem faz contas. Java é uma **linguagem de programação**: ela decide ("se a loja está aberta, mostre isso"), repete ("faça isso para cada produto") e calcula.

Por isso o livro começa pelo HTML. É o caminho mais rápido para ver algo seu na tela. Depois, com o Java, você dá um cérebro para o que construiu.

DICA

Por que aprender os dois juntos? Porque é assim no trabalho de verdade: quem faz a página precisa entender como ela chega ao navegador, e quem faz o servidor precisa entender o que está entregando.

Preparando o VS Code

Tudo neste livro acontece num programa só, gratuito: o Visual Studio Code. Em uns 15 minutos ele fica pronto para HTML e para Java.

1. Instale o VS Code

Entre em `code.visualstudio.com`, baixe a versão do seu sistema (Windows, Mac ou Linux) e instale. No Windows, durante a instalação, marque as opções **Add to PATH** e **Add "Open with Code" action**: elas deixam o VS Code mais fácil de abrir depois.

2. Instale as extensões

Extensões são pequenos programas que dão superpoderes ao VS Code. Abra a aba de extensões com `Ctrl` + `Shift` + `X`, pesquise pelo nome e clique em **Install**:

EXTENSÃO	PARA QUE SERVE
Live Server	Abre a sua página no navegador e atualiza sozinha toda vez que você salva.
Extension Pack for Java	Pacote da Microsoft que ensina o VS Code a entender, rodar e corrigir Java.
Portuguese (Brazil)	Opcional: deixa os menus em português.

Neste livro, os nomes de menus e botões aparecem em inglês, que é como o VS Code vem de fábrica.

3. Instale o Java

Para escrever e rodar programas Java, você precisa do **JDK** (Java Development Kit). Uma opção gratuita e confiável é o **Eclipse Temurin**, em adoptium.net. Escolha a versão **LTS** mais recente: todos os códigos deste livro funcionam da versão 21 em diante.

No instalador do Windows, deixe marcada a opção **Add to PATH** e marque também **Set JAVA_HOME variable**. Depois, abra o terminal do VS Code com **Ctrl** + **`** e confira:

```
terminal

$ java -version
openjdk version "21.0.12.1" 2026-08-18 LTS
OpenJDK Runtime Environment Temurin-21.0.12.1+1
OpenJDK 64-Bit Server VM Temurin-21.0.12.1+1
```

O número da versão no seu computador pode ser outro. O que importa é aparecer `openjdk version` e um número de 21 para cima.

ATENÇÃO

Se aparecer `java não é reconhecido como um comando`, feche o VS Code inteiro e abra de novo: ele só enxerga programas instalados depois que reinicia. Se continuar, reinstale o Java com a opção **Add to PATH** marcada.

4. Crie a sua pasta de estudos

Crie uma pasta chamada `estudos` e, dentro dela, duas outras: `html` e `java`. No VS Code, use **File > Open Folder** e abra a pasta `estudos`.

DICA

Abra sempre a **pasta**, nunca um arquivo solto. É assim que o VS Code entende que aquilo é um projeto, e as extensões funcionam direito.

Atalhos que você vai usar neste livro

ATALHO	O QUE FAZ
Ctrl + S	Salva o arquivo
Ctrl + `	Abre e fecha o terminal
Ctrl + Shift + X	Abre as extensões
Ctrl + P	Abre um arquivo pelo nome
Ctrl + /	Comenta e descomenta a linha
Shift + Alt + F	Arruma a indentação do arquivo inteiro

No Mac, troque **Ctrl** por **Cmd** e **Alt** por **Option**. O atalho do terminal continua **Ctrl** + **`**.

// PARTE 1

HTML a página

Você vai montar uma página de verdade, do zero, entendendo cada linha. No fim desta parte, a página da padaria estará aberta no seu navegador, pronta para o celular.

- 03 Sua primeira página
- 04 As tags do dia a dia
- 05 Estrutura que o Google entende
- 06 Formulários
- 07 Um pouco de CSS
- 08 Projeto 1: a página da padaria

Sua primeira página

Em menos de cinco minutos você vai ter uma página sua aberta no navegador. Depois, vamos entender cada linha dela.

Crie o arquivo

Na pasta `estudos/html`, clique no ícone de novo arquivo e crie o `index.html`. Por que esse nome? Porque é o arquivo que os servidores procuram primeiro quando alguém abre um endereço. A página inicial de quase todo site do mundo se chama assim.

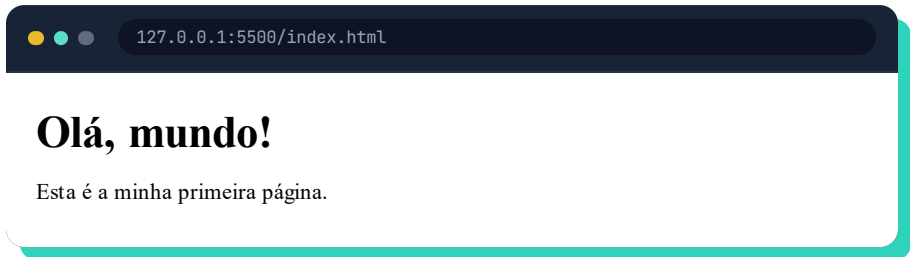
Digite ! e aperte Tab

Dentro do arquivo vazio, digite só um ponto de exclamação e aperte `Tab`. O VS Code escreve a estrutura inteira para você. Esse truque se chama **Emmet** e já vem instalado. Troque `en` por `pt-BR`, mude o título e escreva o conteúdo do `body`:

```
index.html
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport"
6     content="width=device-width, initial-scale=1.0">
7   <title>Minha primeira página</title>
8 </head>
9 <body>
10  <h1>Olá, mundo!</h1>
11  <p>Esta é a minha primeira página.</p>
12 </body>
13 </html>
```

Abra no navegador

Salve com **Ctrl** + **S**. Depois, clique com o botão direito no arquivo e escolha **Open with Live Server** (ou clique em **Go Live**, no canto de baixo do VS Code). O navegador abre sozinho:



Agora mude o texto do parágrafo e salve de novo. A página atualiza sozinha. É o Live Server trabalhando.

Linha por linha

LINHA	O QUE FAZ
<code><!DOCTYPE html></code>	Avisa o navegador que a página usa o HTML moderno.
<code><html lang="pt-BR"></code>	Começa o documento e diz o idioma. Ajuda o Google e quem usa leitor de tela.
<code><head></code>	Informações sobre a página que não aparecem nela.
<code><meta charset></code>	Deixa acentos e ç aparecerem certos, sem virar símbolo estranho.
<code><meta name="viewport"></code>	Faz a página se ajustar à tela do celular.
<code><title></code>	O texto da aba do navegador e do resultado no Google.
<code><body></code>	Tudo que aparece na tela mora aqui dentro.

A anatomia de uma tag

HTML é feito de **tags**: palavras entre os sinais `<` e `>`. A maioria abre, guarda um conteúdo e fecha:

```
<a href="contato.html">Fale comigo</a>
```

<code><a</code>	<code>href</code>	<code>"contato.html"</code>	<code>Fale comigo</code>	<code></code>
abre a tag	atributo	valor	conteúdo	fecha

A tag de fechamento é igual à de abertura, com uma barra antes do nome. Os **atributos** ficam sempre na abertura e dão informações extras: para onde o link vai, qual imagem mostrar, qual é o idioma.

Algumas tags não guardam conteúdo e por isso não fecham. São as tags vazias, como `<meta>`, `<img>`, `<input>` e `<br>`.

DICA

Os espaços no começo das linhas (a indentação) não mudam nada para o navegador, mas mudam tudo para quem lê. Aperte `Shift` + `Alt` + `F` e o VS Code arruma o arquivo inteiro.

DESAFIO

Troque o título da aba pelo seu nome. Embaixo do parágrafo, escreva um segundo parágrafo contando o que você quer construir quando terminar este livro. Salve e veja a página atualizar.

As tags do dia a dia

Com umas dez tags você já monta a maior parte das páginas da internet. Aqui estão elas, com exemplo de cada uma.

Títulos

Vão de `<h1>` (o mais importante) até `<h6>`. Funcionam como o sumário de um livro: o `h1` é o título da página, os `h2` são os capítulos, os `h3` são as partes de cada capítulo.

ATENÇÃO

Use um `h1` por página e não escolha o título pelo tamanho da letra. Se quiser um `h2` maior, quem resolve é o CSS, no capítulo 7.

Parágrafos e destaques

Todo texto corrido vai dentro de `<p>`. Para destacar uma parte, use `<strong>` (importante, aparece em negrito) ou `<em>` (ênfase, aparece em itálico).

Links

A tag `<a>` leva para outro lugar. O destino vai no atributo `href`:

```
Links.html
1 <a href="https://www.google.com" target="_blank">Google</a>
2 <a href="sobre.html">Sobre nós</a>
3 <a href="#contato">Ir para o contato</a>
4 <a href="tel:+55DDDDNUMERO">Ligar agora</a>
5 <a href="mailto:contato@exemplo.com">Mandar e-mail</a>
```

O primeiro abre outro site numa aba nova (`target="_blank"`). O segundo abre outra página sua, na mesma pasta. O terceiro rola até o elemento com `id="contato"` . Os dois últimos abrem o telefone e o e-mail de quem clicou.

Imagens

```
index.html
1 
```

O `src` diz onde está o arquivo. O `alt` descreve a imagem para quem usa leitor de tela e aparece no lugar dela se o arquivo falhar. `width` e `height` reservam o espaço e evitam que a página "pule" enquanto carrega.

Listas

Use `<ul>` para lista com bolinha (a ordem não importa) e `<ol>` para lista numerada (a ordem importa). Cada item vai num `<li>` .

Tudo junto

```
index.html

1 <h1>Padaria Pão Quente</h1>
2 <p>Pão francês saindo <strong>a cada hora</strong>.</p>
3
4 <!-- Isto é um comentário: não aparece na página -->
5 <h2>Os mais pedidos</h2>
6 <ol>
7   <li>Pão francês</li>
8   <li>Pão de queijo</li>
9   <li>Bolo de milho</li>
10 </ol>
11
12 <p>Dúvidas? <a href="#contato">Fale com a gente</a>.</p>
```

```
127.0.0.1:5500/index.html
```

Padaria Pão Quente

Pão francês saindo **a cada hora**.

Os mais pedidos

1. Pão francês
2. Pão de queijo
3. Bolo de milho

Dúvidas? [Fale com a gente](#).

Repare: sem nenhum CSS, o navegador usa a letra e os tamanhos padrão dele. Está feio, mas está tudo no lugar certo. Primeiro a estrutura, depois a beleza.

DESAFIO

Crie uma página `sobre.html` com um título, um parágrafo sobre você, uma lista com três coisas de que você gosta e um link de volta para `index.html`. Na `index.html`, coloque um link para a `sobre.html`.

Estrutura que o Google entende

Uma página bem organizada é lida melhor pelo Google, pelos leitores de tela e por você mesmo daqui a seis meses.

O problema da div

A tag `<div>` é uma caixa sem significado. Dá para montar uma página inteira só com ela, e vai funcionar. O problema é que ninguém sabe o que é cada parte: nem o Google, nem quem vai mexer no código depois.

Por isso o HTML tem tags com nome. Elas aparecem na tela do mesmo jeito que uma `div`, mas dizem o que cada pedaço é:

`<header>` topo

`<nav>` menu

`<main>` conteúdo principal

`<section>` cardápio

`<section>` contato

`<aside>` apoio

`<footer>` rodapé

TAG	PARA QUE SERVE
<code><header></code>	Topo da página: nome, logo e, muitas vezes, o menu.
<code><nav></code>	Grupo de links de navegação.
<code><main></code>	O conteúdo principal. Só um por página.
<code><section></code>	Uma parte com assunto próprio, quase sempre com título.
<code><article></code>	Algo que faz sentido sozinho: um post, uma notícia, um produto.
<code><aside></code>	Conteúdo de apoio, como uma barra lateral.
<code><footer></code>	Rodapé: contato, redes sociais, direitos.

Na prática

```

index.html

1  <body>
2    <header>
3      <p>Pão Quente</p>
4      <nav>
5        <a href="#cardapio">Cardápio</a>
6        <a href="#contato">Contato</a>
7      </nav>
8    </header>
9
10   <main>
11     <section id="cardapio">
12       <h2>Cardápio</h2>
13     </section>
14     <section id="contato">
15       <h2>Contato</h2>
16     </section>
17   </main>
18
19   <footer>
20     <p>© 2026 Padaria Pão Quente</p>
21   </footer>
22 </body>

```

Por que isso importa

- **Google:** ele entende o que é conteúdo principal e o que é menu, e isso ajuda a página a aparecer na busca certa.
- **Acessibilidade:** quem usa leitor de tela consegue pular direto para o `main` ou para o menu, em vez de ouvir a página inteira.
- **Manutenção:** quando você abrir o código daqui a meses, vai achar cada parte na hora.

DICA

A `div` continua útil. Quando nenhuma tag com nome servir, e você só precisar agrupar coisas para o visual, aí sim use uma `div`.

DESAFIO

Reorganize a sua `index.html` com `header`, `main` e `footer`. A página não vai mudar de aparência, e é isso mesmo: a mudança é para quem lê o código.

Formulários

Formulário é o jeito que a página tem de ouvir o visitante: contato, orçamento, cadastro, login. Todo site que vende alguma coisa tem um.

Um formulário de encomenda

```
index.html

1 <form action="/contato" method="post">
2   <label for="nome">Seu nome</label>
3   <input type="text" id="nome" name="nome" required>
4
5   <label for="email">E-mail</label>
6   <input type="email" id="email" name="email">
7
8   <label for="pedido">Você precisa de</label>
9   <select id="pedido" name="pedido">
10    <option>Bolo de aniversário</option>
11    <option>Salgados para festa</option>
12  </select>
13
14  <label for="mensagem">Mensagem</label>
15  <textarea id="mensagem" name="mensagem" rows="4"></textarea>
16
17  <button type="submit">Enviar</button>
18 </form>
```

A screenshot of a web browser window with the address bar showing "127.0.0.1:5500/index.html". The page displays a simple form with the following elements: "Seu nome" followed by a text input field, "E-mail" followed by a text input field, "Você precisa de" followed by a text input field, "Mensagem" followed by a larger text area, and a button labeled "Enviar".

Sem CSS, os campos ficam um do lado do outro, quebrando linha onde der. No Projeto 1 você vai organizar isso.

As peças

TAG	O QUE É
<code><form></code>	A caixa que junta os campos e diz para onde enviar.
<code><label></code>	O nome do campo. Clicar nele coloca o cursor no campo.
<code><input></code>	Campo de uma linha. O <code>type</code> muda o tipo.
<code><textarea></code>	Campo de várias linhas, para mensagens.
<code><select></code>	Lista de opções para escolher. Cada uma é um <code><option></code> .
<code><button></code>	O botão. Com <code>type="submit"</code> , ele envia o formulário.

Os atributos que fazem a mágica

- `name` é o nome com que o dado chega ao servidor. **Campo sem `name` não é enviado.**
- `id` e `for` ligam o `label` ao campo. Os dois precisam ter o mesmo valor.
- `type` muda o teclado do celular: `email` mostra o @, `tel` mostra só números. Também existem `number`, `date`, `password` e `checkbox`.

- `required` faz o navegador não deixar enviar com o campo vazio.

Para onde vai o que a pessoa digita?

O `action` diz o endereço que vai receber os dados, e o `method="post"` diz que eles vão escondidos no corpo do pedido, e não no endereço. Se você clicar em **Enviar** agora, vai aparecer uma página de erro. É normal: ainda não existe ninguém do outro lado para receber. **No Projeto 3, esse alguém vai ser um programa seu, em Java.**

ATENÇÃO

Nunca confie só no `required`. Quem entende de programação consegue enviar dados sem passar pelo navegador. A conferência de verdade sempre acontece no servidor, e você vai ver isso no capítulo 19.

DESAFIO

Monte um formulário de inscrição numa lista de novidades, com nome, e-mail e uma caixinha de marcar (`type="checkbox"`) escrito "Aceito receber novidades", que seja obrigatória.

Um pouco de CSS

Este é um livro de HTML e Java, mas página sem CSS parece documento dos anos 90. Aqui vai o mínimo para deixar a sua apresentável.

Ligando o CSS ao HTML

Crie um arquivo `estilo.css` na mesma pasta da página e avise o HTML que ele existe, com esta linha dentro do `head` :

```
index.html

<link rel="stylesheet" href="estilo.css">
```

A regra do CSS

Todo CSS segue o mesmo formato: **quem** vai mudar (o seletor) e, entre chaves, **o que** muda (propriedade e valor):

```
estilo.css

1 h1 {
2   color: #5A3A22;
3   font-size: 32px;
4 }
```

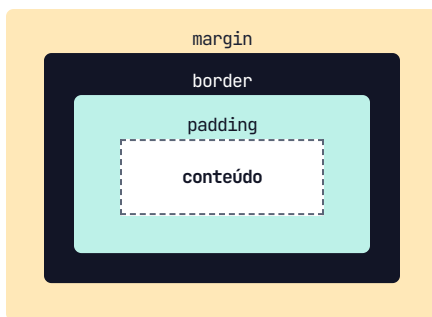
SELETOR	PEGA	NO HTML
<code>h1</code>	Todas as tags h1	<code><h1></code>
<code>.botao</code>	Tudo com a classe botao	<code></code>
<code>#contato</code>	O elemento com id contato	<code><section id="contato"></code>
<code>nav a</code>	Os links que estão dentro do nav	<code><nav><a></code>

As propriedades que mais aparecem

PROPRIEDADE	O QUE MUDA
<code>color</code>	Cor do texto
<code>background</code>	Cor (ou imagem) de fundo
<code>font-family</code>	A fonte
<code>font-size</code>	O tamanho da letra
<code>text-align</code>	Alinhamento: left, center, right
<code>padding</code>	Espaço de dentro da caixa
<code>margin</code>	Espaço de fora da caixa
<code>border-radius</code>	Arredonda os cantos
<code>max-width</code>	Largura máxima

Caixas dentro de caixas

Para o CSS, todo elemento é uma caixa. Por dentro fica o conteúdo, em volta dele o `padding`, depois a borda e, por fora, a `margin`:



DICA

Comece todo CSS com `* { box-sizing: border-box; }`. Assim, a largura que você escolhe já inclui o padding e a borda, e as contas ficam fáceis.

Cores com nome: variáveis

Se a mesma cor aparece em dez lugares, guarde ela num lugar só. Para trocar a cor do site inteiro, você muda uma linha:

```
estilo.css

1 :root {
2   --marrom: #5A3A22;
3 }
4
5 h1 {
6   color: var(--marrom);
7 }
```

Um do lado do outro: flex

Para colocar o nome da padaria na esquerda e o menu na direita, use `display: flex` na caixa que contém os dois:

estilo.css

```
1 .topo {  
2   display: flex;  
3   justify-content: space-between;  
4   align-items: center;  
5 }
```

`justify-content: space-between` empurra o primeiro item para a esquerda e o último para a direita. `align-items: center` alinha os dois pela altura.

DESAFIO

Escolha três cores para a sua página, guarde em variáveis e use nas tags `body`, `h1` e `a`.

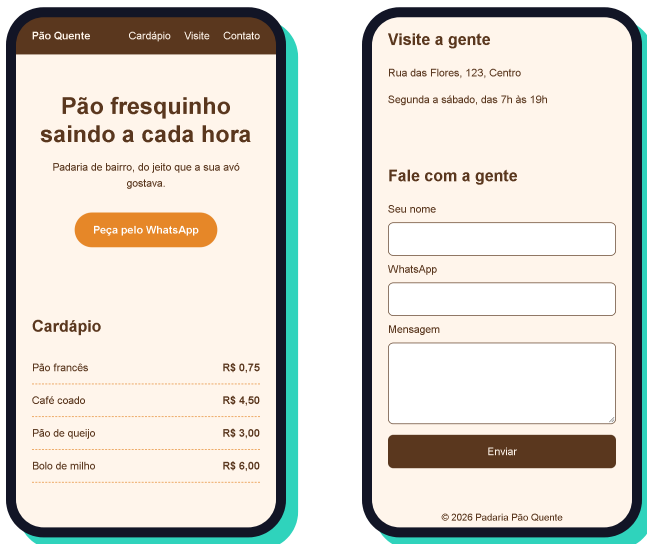
A página da padaria

Hora de juntar tudo. Você vai montar a página completa de uma padaria de bairro, pronta para abrir no celular.

O que a página tem

- Um topo com o nome e o menu.
- Um destaque com a chamada principal e um botão de WhatsApp.
- O cardápio com os preços.
- Endereço e horário.
- Um formulário de contato.
- O rodapé.

É assim que ela vai ficar no celular quando você terminar:



onde vamos chegar: a página inteira, dividida em duas telas

Crie uma pasta `projeto1` com dois arquivos:

```
projeto1/  
├── index.html  
└── estilo.css
```

O HTML, parte 1: cabeça e topo

index.html

```
1 <!DOCTYPE html>  
2 <html lang="pt-BR">  
3 <head>  
4   <meta charset="UTF-8">  
5   <meta name="viewport"  
6     content="width=device-width, initial-scale=1">  
7   <title>Padaria Pão Quente | Pão fresquinho todo dia</title>  
8   <meta name="description"  
9     content="Pão francês saindo a cada hora e café coado.">  
10  <link rel="stylesheet" href="estilo.css">  
11 </head>  
12 <body>  
13   <header class="topo">  
14     <p class="marca">Pão Quente</p>  
15     <nav>  
16       <a href="#cardapio">Cardápio</a>  
17       <a href="#visite">Visite</a>  
18       <a href="#contato">Contato</a>  
19     </nav>  
20   </header>  
21  
22   <main>  
23     <section class="destaque">  
24       <h1>Pão fresquinho saindo a cada hora</h1>  
25       <p>Padaria de bairro, do jeito que a sua avó gostava.</p>  
26       <a class="botao" href="https://wa.me/55DDDNUMERO"  
27         target="_blank">Peça pelo WhatsApp</a>  
28     </section>
```

- **Linhas 8 e 9:** a `description` é o textinho que aparece embaixo do título no Google.
- **Linha 13:** o `header` ganhou a classe `topo`, que o CSS vai usar.
- **Linhas 16 a 18:** os links do menu apontam para os `id` das seções. Clicar rola a página até elas.
- **Linha 26:** troque `55DDDNUMERO` pelo WhatsApp de verdade: 55, o DDD e o número, só os dígitos.

O HTML, parte 2: cardápio e endereço

```

29
30 <section id="cardapio">
31   <h2>Cardápio</h2>
32   <ul class="cardapio">
33     <li>Pão francês <strong>R$ 0,75</strong></li>
34     <li>Café coado <strong>R$ 4,50</strong></li>
35     <li>Pão de queijo <strong>R$ 3,00</strong></li>
36     <li>Bolo de milho <strong>R$ 6,00</strong></li>
37   </ul>
38 </section>
39
40 <section id="visite">
41   <h2>Visite a gente</h2>
42   <p>Rua das Flores, 123, Centro</p>
43   <p>Segunda a sábado, das 7h às 19h</p>
44 </section>
45

```

O cardápio é uma lista. O preço fica num `strong`, que o CSS vai empurrar para a direita.

O HTML, parte 3: contato e rodapé

```
index.html

46     <section id="contato">
47         <h2>Fale com a gente</h2>
48         <form action="/contato" method="post">
49             <label for="nome">Seu nome</label>
50             <input type="text" id="nome" name="nome" required>
51
52             <label for="telefone">WhatsApp</label>
53             <input type="tel" id="telefone" name="telefone">
54
55             <label for="mensagem">Mensagem</label>
56             <textarea id="mensagem" name="mensagem" rows="4"
57                 required></textarea>
58
59             <button type="submit">Enviar</button>
60         </form>
61     </section>
62 </main>
63
64 <footer>
65     <p>© 2026 Padaria Pão Quente</p>
66 </footer>
67 </body>
68 </html>
```

Repare no `action="/contato"` da linha 48. Ele vai fazer sentido no Projeto 3, quando o Java receber essas mensagens.

O CSS, parte 1: cores, base e topo

```
estilo.css

1  /* As cores da padaria, guardadas em variáveis */
2  :root {
3      --creme: #FFF8EE;
4      --marrom: #5A3A22;
5      --laranja: #E8892B;
```

```

6 }
7
8 * {
9   box-sizing: border-box;
10 }
11
12 body {
13   margin: 0;
14   font-family: Arial, sans-serif;
15   color: var(--marrom);
16   background: var(--creme);
17   line-height: 1.5;
18 }
19
20 .topo {
21   display: flex;
22   justify-content: space-between;
23   align-items: center;
24   padding: 16px 24px;
25   background: var(--marrom);
26 }
27
28 .marca {
29   margin: 0;
30   font-weight: bold;
31   color: var(--creme);
32 }
33
34 nav a {
35   margin-left: 16px;
36   color: var(--creme);
37   text-decoration: none;
38 }

```

As três cores da padaria estão em variáveis (linhas 2 a 6). O `body` define a fonte e as cores de toda a página, e o `.topo` usa `flex` para separar o nome do menu.

O CSS, parte 2: conteúdo e botão

```
estilo.css

40 main {
41     max-width: 720px;
42     margin: 0 auto;
43     padding: 0 24px;
44 }
45
46 section {
47     padding: 24px 0;
48 }
49
50 .destaque {
51     text-align: center;
52     padding: 56px 0;
53 }
54
55 .destaque h1 {
56     font-size: 2.2rem;
57     line-height: 1.2;
58     margin: 0 0 12px;
59 }
60
61 .botao {
62     display: inline-block;
63     margin-top: 16px;
64     padding: 14px 28px;
65     border-radius: 999px;
66     background: var(--laranja);
67     color: white;
68     font-weight: bold;
69     text-decoration: none;
70 }
```

- **Linhas 41 e 42:** `max-width` com `margin: 0 auto` é o jeito clássico de centralizar o conteúdo e não deixar o texto esticar demais em telas grandes.
- **Linha 65:** um `border-radius` bem grande deixa o botão com as pontas totalmente redondas.

O CSS, parte 3: cardápio, formulário e rodapé

```
estilo.css

72 .cardapio {
73     list-style: none;
74     padding: 0;
75 }
76
77 .cardapio li {
78     display: flex;
79     justify-content: space-between;
80     padding: 12px 0;
81     border-bottom: 1px dashed var(--laranja);
82 }
83
84 form {
85     display: grid;
86     gap: 8px;
87 }
88
89 input,
90 textarea,
91 button {
92     font: inherit;
93     padding: 12px;
94     border: 1px solid var(--marrom);
95     border-radius: 8px;
96 }
97
98 button {
99     margin-top: 8px;
100     background: var(--marrom);
101     color: var(--creme);
102     cursor: pointer;
103 }
104
105 footer {
106     text-align: center;
107     padding: 24px;
108     font-size: 0.9rem;
109 }
```

- **Linhas 77 a 81:** cada item do cardápio vira um `flex` com `space-between`: o nome fica na esquerda e o preço na direita.
- **Linhas 84 a 87:** `display: grid` com `gap` empilha os campos do formulário, um embaixo do outro, com espaço igual.
- **Linha 92:** `font: inherit` faz os campos usarem a mesma fonte da página. Sem isso, cada navegador usa a dele.

Teste como se fosse um celular

Abra o `index.html` com o Live Server. No computador, aperte `F12` e depois `Ctrl` + `Shift` + `M`: o Chrome mostra a página do tamanho de uma tela de celular, e dá para escolher o modelo do aparelho no topo. Teste também deitando o celular e escolhendo um tablet: a página precisa ficar boa em todos.

DESAFIO

1. Troque as cores das variáveis pelas cores de um negócio que você conhece.
2. Crie uma seção "Encomendas" com uma lista de produtos para festas.
3. Coloque uma foto de verdade no destaque, com um `alt` bem escrito.

// PARTE 2

Java a lógica

Agora o computador começa a pensar: guardar informação, tomar decisões e repetir tarefas. No fim desta parte, você terá um programa de caixa funcionando no terminal.

09 Seu primeiro programa em Java

10 Variáveis: guardando informação

11 Decisões com if e switch

12 Repetições com for e while

13 Listas: arrays e ArrayList

14 Métodos: dando nome às tarefas

15 Classes e objetos

16 Conversando com o usuário

17 Projeto 2: o caixa da padaria

Seu primeiro programa em Java

Todo programador Java do mundo começou com as mesmas cinco linhas. Agora é a sua vez.

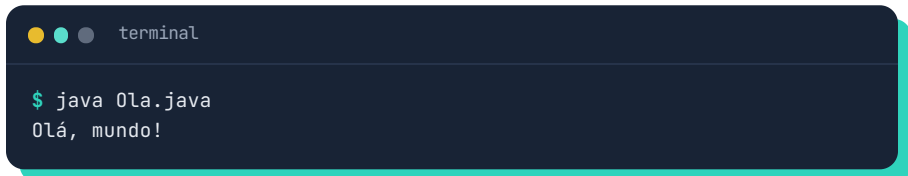
Crie e rode

Na pasta `estudos/java`, crie um arquivo chamado `Ola.java`, com o O maiúsculo, e digite:

A screenshot of a code editor window titled "Ola.java". The editor has a dark background with light blue and green syntax highlighting. The code is as follows:

```
1 public class Ola {  
2     public static void main(String[] args) {  
3         System.out.println("Olá, mundo!");  
4     }  
5 }
```

Salve. Em cima da linha 2 vai aparecer um botão pequeno escrito **Run**. Clique nele e olhe o terminal:

A screenshot of a terminal window titled "terminal". The terminal has a dark background. It shows the command `$ java Ola.java` being entered, followed by the output `Olá, mundo!`.

```
$ java Ola.java  
Olá, mundo!
```

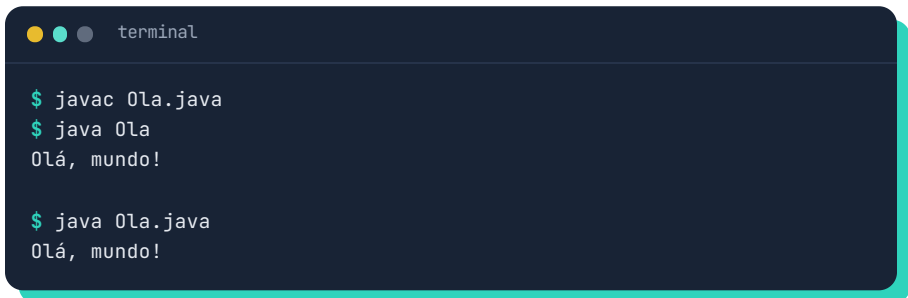
Pronto: você acabou de escrever e rodar um programa em Java.

Linha por linha

- `public class Ola` : todo código Java mora dentro de uma **classe**. O nome da classe tem que ser igual ao nome do arquivo: `Ola` mora em `Ola.java`.
- `public static void main(String[] args)` : é o ponto de partida. Quando você roda o programa, o Java procura exatamente essa linha. Por enquanto, encare como uma fórmula: cada palavra dela vai fazer sentido nos próximos capítulos.
- `System.out.println("Olá, mundo!");` : escreve o texto no terminal e pula para a linha de baixo. Texto vai sempre entre aspas duplas.
- **O ponto e vírgula** termina cada comando, como o ponto final de uma frase.
- **As chaves** `{ }` abrem e fecham blocos. A classe é um bloco, o `main` é outro bloco dentro dela.

Rodando pelo terminal

O botão **Run** é um atalho para o que acontece por baixo. Dá para fazer na mão, pelo terminal, de dois jeitos:

A terminal window with a dark background and a teal border. The title bar says "terminal". It shows two commands being executed. The first command is `javac Ola.java`, followed by `java Ola`, which outputs `Olá, mundo!`. The second command is `java Ola.java`, which also outputs `Olá, mundo!`.

```
terminal

$ javac Ola.java
$ java Ola
Olá, mundo!

$ java Ola.java
Olá, mundo!
```

No primeiro jeito, o `javac` **compila**: traduz o seu código para um arquivo `Ola.class`, que a máquina do Java entende. Depois, o `java Ola` roda esse arquivo. No segundo jeito, o Java compila e roda de uma vez só, sem criar o `.class`. É o que vamos usar daqui para frente.

Os erros mais comuns

Você vai errar, e isso é bom: o compilador avisa o que está errado e em qual linha. As mensagens vêm em inglês. Estas são as que mais aparecem:

```
terminal

$ javac Ola.java
Ola.java:3: error: ';' expected
    System.out.println("Olá, mundo!")
                        ^
1 error
```

MENSAGEM	O QUE ACONTECEU
<code>';' expected</code>	Faltou o ponto e vírgula no fim da linha.
<code>package system does not exist</code>	Você escreveu <code>system</code> com s minúsculo. Java diferencia maiúsculas de minúsculas.
<code>cannot find symbol</code>	Algum nome está escrito errado, como <code>printn</code> no lugar de <code>println</code> .
<code>should be declared in a file named Ola.java</code>	O nome do arquivo é diferente do nome da classe.

DICA

A mensagem mostra o arquivo e a linha (`Ola.java:3` é a linha 3) e põe uma setinha `^` embaixo do lugar do problema. Comece sempre por ali.

ATENÇÃO

Se no seu terminal aparecer `01?` no lugar de `01á`, digite `chcp 65001` e rode de novo. Isso troca o terminal do Windows para um modo que entende acentos.

DESAFIO

Faça o programa escrever três linhas: o seu nome, a sua cidade e o que você quer construir. Depois troque um dos `println` por `print` e descubra a diferença.

Variáveis: guardando informação

Variável é uma caixinha com nome, onde o programa guarda uma informação para usar depois. Em Java, cada caixinha tem um tipo, e o tipo diz o que cabe nela.

Os tipos básicos

TIPO	GUARDA	EXEMPLO
<code>int</code>	Números inteiros	<code>3</code> , <code>-10</code> , <code>2026</code>
<code>double</code>	Números com casas decimais	<code>12.50</code> , <code>0.75</code>
<code>boolean</code>	Verdadeiro ou falso	<code>true</code> , <code>false</code>
<code>char</code>	Um caractere, entre aspas simples	<code>'A'</code>
<code>String</code>	Texto, entre aspas duplas	<code>"Ana"</code>

Repare que `String` começa com letra maiúscula e os outros não. Isso tem um motivo, que você vai entender no capítulo 15.

Na prática

Pedido.java

```
1 public class Pedido {
2     public static void main(String[] args) {
3         String cliente = "Ana";
4         int quantidade = 3;
5         double precoUnitario = 12.50;
6         boolean entrega = true;
7
8         double total = quantidade * precoUnitario;
```

```

9
10     System.out.println("Cliente: " + cliente);
11     System.out.println("Itens: " + quantidade);
12     System.out.println("Entrega? " + entrega);
13     System.out.printf("Total: R$ %.2f%n", total);
14 }
15 }

```

```

terminal

$ java Pedido.java
Cliente: Ana
Itens: 3
Entrega? true
Total: R$ 37,50

```

- **Linhas 3 a 6:** cada variável é criada com **tipo, nome e valor**:
`int quantidade = 3;`
- **Linha 8:** o `*` multiplica. A conta usa o que está guardado nas caixinhas.
- **Linha 10:** o `+` entre texto e variável junta tudo num texto só.
- **Linha 13:** o `printf` formata: `%.2f` mostra o número com duas casas decimais e `%n` pula linha.

DICA

No código, decimal usa **ponto** (`12.50`). Na hora de mostrar, o `printf` usa vírgula se o seu computador estiver em português, ou ponto se estiver em inglês.

Fazendo contas

Java tem as quatro operações (`+`, `-`, `*`, `/`) e mais uma muito útil, o `%`, que dá o resto da divisão. Cuidado com a divisão entre inteiros:

Contas.java

```
System.out.println(7 / 2);    // 3, porque os dois são int
System.out.println(7 / 2.0);  // 3.5
System.out.println(7 % 2);    // 1, o resto da divisão
```

Quando os dois números são inteiros, o resultado também é inteiro, e a parte quebrada some. Basta um deles ter casa decimal para a conta sair completa.

Mudando o valor

Contas.java

```
int estoque = 10;
estoque = estoque - 3; // agora vale 7
estoque += 5;          // o mesmo que estoque = estoque + 5
estoque++;             // soma 1: agora vale 13
```

var e final

Com `var`, o Java descobre o tipo sozinho pelo valor:

`var cidade = "Jales";` vira uma `String`. Já o `final` cria uma constante, um valor que não pode mudar depois:

`final double TAXA_ENTREGA = 5.0;`. Por costume, constantes são escritas em maiúsculas.

ATENÇÃO

O `double` é ótimo para aprender, mas às vezes erra centavos: `0.1 + 0.2` dá `0.30000000000000004`. Por isso, sistemas de dinheiro de verdade usam o tipo `BigDecimal`. Guarde esse nome para quando for trabalhar com pagamentos.

DESAFIO

Faça um programa de troco: o cliente comprou R\$ 37,50 e pagou com R\$ 50,00. Guarde os dois valores em variáveis e mostre o troco com duas casas decimais.

Decisões com **if** e **switch**

Programa de verdade toma decisões: se a loja está aberta, mostra uma coisa; se está fechada, mostra outra.

Na prática

```
Horario.java

1 public class Horario {
2     public static void main(String[] args) {
3         int hora = 10;
4         String dia = "sábado";
5
6         if (dia.equals("domingo")) {
7             System.out.println("Hoje abrimos das 7h às 12h.");
8         } else if (hora >= 7 && hora < 19) {
9             System.out.println("Estamos abertos!");
10        } else {
11            System.out.println("Fechado. Abrimos às 7h.");
12        }
13
14        String turno = switch (hora / 6) {
15            case 0 -> "madrugada";
16            case 1 -> "manhã";
17            case 2 -> "tarde";
18            default -> "noite";
19        };
20        System.out.println("Agora é " + turno + ".");
21    }
22 }
```

```
terminal

$ java Horario.java
Estamos abertos!
Agora é manhã.
```

if, else if e else

O `if` testa uma condição entre parênteses. Se ela for verdadeira, roda o bloco dele. Se não for, o Java testa o `else if`, e assim por diante. O `else` do final é o "se nada disso deu certo". Só um dos blocos roda.

OPERADOR	SIGNIFICA	EXEMPLO
<code>==</code>	igual a	<code>hora == 7</code>
<code>!=</code>	diferente de	<code>hora != 0</code>
<code>></code> <code><</code>	maior, menor	<code>hora < 19</code>
<code>>=</code> <code><=</code>	maior ou igual, menor ou igual	<code>hora >= 7</code>
<code>&&</code>	e (as duas precisam ser verdade)	<code>hora >= 7 && hora < 19</code>
<code> </code>	ou (basta uma ser verdade)	<code>dia == 6 dia == 7</code>
<code>!</code>	não (inverte)	<code>!aberto</code>

ATENÇÃO

Para comparar **textos**, use `.equals()`, como na linha 6, e nunca `==`. O `==` com `String` compara se é o mesmo objeto na memória, e não se as letras são iguais. Às vezes funciona por sorte, e esse é o problema.

switch: quando são muitas opções

Quando você compara a mesma coisa com vários valores, o `switch` fica mais limpo que uma fila de `if`. Na linha 14, `hora / 6` é uma divisão entre inteiros: de 0h a 5h dá 0, de 6h a 11h dá 1, de 12h a 17h dá 2, e daí para frente cai no `default`. Cada `case` devolve um texto, que vai para a variável `turno`.

DESAFIO

Mude os valores de `hora` e `dia` para fazer aparecer cada uma das mensagens. Depois, crie uma regra nova: na sexta-feira, a padaria fica aberta até as 21h.

Repetições com **for** e **while**

Computador não cansa. Repetir uma tarefa mil vezes custa para ele o mesmo que repetir uma vez, e é aí que programar começa a valer a pena.

Na prática

Repeticao.java

```
1 public class Repeticao {
2     public static void main(String[] args) {
3         int numero = 7;
4
5         for (int i = 1; i <= 5; i++) {
6             int resultado = numero * i;
7             System.out.println(numero + " x " + i + " = " + resultado);
8         }
9
10        int segundos = 3;
11        while (segundos > 0) {
12            System.out.println("Abrindo em " + segundos + "...");
13            segundos--;
14        }
15        System.out.println("Loja aberta!");
16    }
17 }
```

```
terminal

$ java Repeticao.java
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
Abrindo em 3...
Abrindo em 2...
Abrindo em 1...
Loja aberta!
```

for: quando você sabe quantas vezes

A linha 5 tem três partes, separadas por ponto e vírgula:

- `int i = 1` : começa com `i` valendo 1.
- `i <= 5` : continua enquanto `i` for menor ou igual a 5.
- `i++` : no fim de cada volta, soma 1 no `i`.

O bloco roda cinco vezes, uma para cada valor de `i`.

while: enquanto for verdade

O `while` da linha 11 repete enquanto a condição for verdadeira. É o mais indicado quando você não sabe de antemão quantas voltas vão ser, como "continue pedindo até a pessoa digitar 0".

ATENÇÃO

Se você esquecer a linha 13 (`segundos--`), o valor nunca muda, a condição nunca fica falsa e o programa repete para sempre. Se isso acontecer, clique no terminal e aperte `Ctrl` + `C` para parar.

DICA

Dentro de qualquer repetição, `break` sai dela na hora e `continue` pula direto para a próxima volta. Você vai usar os dois no Projeto 2.

DESAFIO

Mostre a tabuada completa, do 1 ao 10. Dica: coloque um `for` dentro de outro `for`.

Listas: arrays e ArrayList

Uma variável guarda uma coisa. Uma lista guarda várias: todos os produtos do cardápio, todos os itens de um pedido.

Na prática

```
Cardapio.java

1 import java.util.ArrayList;
2
3 public class Cardapio {
4     public static void main(String[] args) {
5         String[] dias = {"seg", "ter", "qua", "qui", "sex", "sáb"};
6         System.out.println("Primeiro dia: " + dias[0]);
7         System.out.println("Último dia: " + dias[dias.length - 1]);
8
9         ArrayList<String> itens = new ArrayList<>();
10        itens.add("Pão francês");
11        itens.add("Café coado");
12        itens.add("Bolo de milho");
13        itens.remove("Café coado");
14
15        for (String item : itens) {
16            System.out.println("- " + item);
17        }
18        System.out.println("Total de itens: " + itens.size());
19    }
20 }
```

```
terminal

$ java Cardapio.java
Primeiro dia: seg
Último dia: sáb
- Pão francês
- Bolo de milho
Total de itens: 2
```

Array: tamanho fixo

A linha 5 cria um array com seis textos. O tipo tem colchetes (`String[]`) e os valores vão entre chaves. Cada item tem uma posição, chamada **índice**, e a contagem **começa em 0**: `dias[0]` é "seg".

O tamanho fica em `dias.length`. Como a contagem começa em 0, o último item está na posição `length - 1`, como na linha 7.

ArrayList: cresce e diminui

O array nasce com tamanho fixo. Quando você não sabe quantos itens vão entrar, use a `ArrayList` (linha 9), que precisa do `import` da linha 1. O tipo do que ela guarda vai entre os sinais `< >`.

MÉTODO	O QUE FAZ
<code>add(x)</code>	Coloca no fim da lista
<code>get(i)</code>	Pega o item da posição <code>i</code>
<code>set(i, x)</code>	Troca o item da posição <code>i</code>
<code>remove(x)</code>	Tira da lista
<code>size()</code>	Quantos itens tem
<code>contains(x)</code>	Diz se o item está na lista

Passando por todos os itens

O `for` da linha 15 é o "for de cada": ele lê como "para cada `item` em `itens`". Não precisa de contador, e por isso é o jeito mais usado para percorrer listas.

DICA

Para uma lista que nunca vai mudar, use `List.of("a", "b", "c")`. É mais curto, e o Java não deixa ninguém alterar a lista por engano. O cardápio do Projeto 2 é feito assim.

DESAFIO

Monte uma lista de compras com cinco itens, remova um deles e mostre o resultado numerado (`1. Arroz`, `2. Feijão` ...). Dica: use o `for` com contador do capítulo 12 e o `get(i)`.

Métodos: dando nome às tarefas

Quando a mesma conta aparece em vários lugares, você escreve ela uma vez, dá um nome e chama quantas vezes quiser. Isso é um método.

Na prática

Metodos.java

```
1 public class Metodos {
2     static double calcularTotal(double preco, int quantidade) {
3         return preco * quantidade;
4     }
5
6     static void saudar(String nome) {
7         System.out.println("Bom dia, " + nome + "!");
8     }
9
10    public static void main(String[] args) {
11        saudar("Carlos");
12        double total = calcularTotal(0.75, 10);
13        System.out.printf("10 pães: R$ %.2f%n", total);
14        double cafes = calcularTotal(4.50, 2);
15        System.out.printf("2 cafés: R$ %.2f%n", cafes);
16    }
17 }
```

terminal

```
$ java Metodos.java
Bom dia, Carlos!
10 pães: R$ 7,50
2 cafés: R$ 9,00
```

A anatomia de um método

Na linha 2,

`static double calcularTotal(double preco, int quantidade)` diz quatro coisas:

- `static`: o método pertence à classe e pode ser chamado direto do `main`. Você vai ver o outro tipo no próximo capítulo.
- `double`: o tipo do valor que o método **devolve**.
- `calcularTotal`: o nome. Por costume, começa com verbo e letra minúscula.
- `(double preco, int quantidade)`: os **parâmetros**, as informações que o método precisa receber para trabalhar.

O `return` da linha 3 devolve o resultado para quem chamou. Na linha 12, esse resultado vai direto para a variável `total`.

Já o `saudar`, na linha 6, é `void`: ele faz alguma coisa (escreve na tela), mas não devolve valor nenhum. Por isso não tem `return`.

Agora dá para ler o `main`: é um método `public` (qualquer um pode chamar), `static`, `void` (não devolve nada) e que recebe um array de textos chamado `args`.

DICA

Um bom método faz **uma coisa só** e tem um nome que diz qual. Se você precisa escrever "e" no nome dele, provavelmente são dois métodos.

DESAFIO

Crie o método `aplicarDesconto(double valor, double porcentagem)`, que devolve o valor com o desconto. Use ele para mostrar quanto fica um bolo de R\$ 18,00 com 15% de desconto.

Classes e objetos

Até aqui, cada informação era uma variável solta. Com classes, você junta tudo que descreve uma coisa (nome, preço, desconto) num lugar só.

Molde e bolo

Pense numa forma de bolo. A forma é a **classe**: ela diz o formato, mas não dá para comer. Cada bolo que sai dela é um **objeto**: todos têm o mesmo formato, mas cada um tem o seu sabor. Uma classe `Produto` descreve o que todo produto tem; o pão francês e o bolo de milho são dois objetos feitos com ela.

Na prática

```
Loja.java

1 public class Loja {
2     public static void main(String[] args) {
3         Produto pao = new Produto("Pão francês", 0.75);
4         Produto bolo = new Produto("Bolo de milho", 18.00);
5
6         pao.mostrar();
7         bolo.mostrar();
8
9         double promo = bolo.precoComDesconto(10);
10        System.out.printf("Bolo com 10%% off: R$ %.2f%n", promo);
11    }
12 }
13
14 class Produto {
15     String nome;
16     double preco;
17 }
```

```

18  Produto(String nome, double preco) {
19      this.nome = nome;
20      this.preco = preco;
21  }
22
23  void mostrar() {
24      System.out.printf("%s: R$ %.2f%n", nome, preco);
25  }
26
27  double precoComDesconto(int porcentagem) {
28      return preco - preco * porcentagem / 100;
29  }
30 }

```

terminal

```

$ java Loja.java
Pão francês: R$ 0,75
Bolo de milho: R$ 18,00
Bolo com 10% off: R$ 16,20

```

As partes de uma classe

- **Atributos (linhas 15 e 16):** as informações que cada objeto guarda. Todo produto tem um nome e um preço.
- **Construtor (linhas 18 a 21):** tem o mesmo nome da classe e roda quando o objeto nasce. O `this.nome` é o atributo do objeto; o `nome` sozinho é o parâmetro que chegou.
- **Métodos (linhas 23 e 27):** o que o objeto sabe fazer. Repare que eles não têm `static`: cada objeto usa os próprios dados.

Nas linhas 3 e 4, a palavra `new` cria os objetos. Depois, o ponto chama um método de um objeto específico: `pao.mostrar()` mostra o pão, `bolo.mostrar()` mostra o bolo.

E agora o mistério do capítulo 10 se resolve: `String` começa com maiúscula porque é uma classe, e todo texto que você cria é um objeto dela. Por isso o texto tem métodos, como o `.equals()`.

DICA

Quando a classe só guarda dados, existe um atalho, o `record`. Esta linha cria uma classe com os dois atributos, o construtor e os métodos para ler cada um (`p.nome()` e `p.preco()`):

```
Produto.java  
  
record Produto(String nome, double preco) {}
```

O Projeto 2 usa esse atalho.

DESAFIO

Crie uma classe `Cliente` com nome e telefone, e um método `apresentar()` que escreve "Cliente: Ana, telefone: ...". Crie dois clientes no `main` e apresente os dois.

Conversando com o usuário

Até agora, todos os valores estavam escritos no código. Neste capítulo, o programa pergunta, espera a pessoa digitar e usa a resposta.

Na prática

```
Conversa.java

1 import java.util.Scanner;
2
3 public class Conversa {
4     public static void main(String[] args) {
5         Scanner entrada = new Scanner(System.in);
6
7         System.out.print("Qual é o seu nome? ");
8         String nome = entrada.nextLine();
9
10        System.out.print("Quantos anos você tem? ");
11        int idade = Integer.parseInt(entrada.nextLine());
12
13        System.out.println("Prazer, " + nome + "!");
14        int futuro = idade + 10;
15        System.out.println("Daqui a 10 anos: " + futuro + " anos.");
16    }
17 }
```

Rode e responda no terminal. As partes em amarelo são o que a pessoa digitou:

terminal

```
$ java Conversa.java
Qual é o seu nome? Marina
Quantos anos você tem? 27
Prazer, Marina!
Daqui a 10 anos: 37 anos.
```

Como funciona

- **Linha1:** o `Scanner` não vem carregado, então precisa do `import`.
- **Linha5:** cria um `Scanner` que lê o teclado (`System.in`).
- **Linha7:** o `print`, sem o `"\n"`, não pula linha. Assim a resposta é digitada do lado da pergunta.
- **Linha8:** `nextLine()` espera a pessoa digitar e apertar Enter, e devolve o que ela digitou como texto.
- **Linha11:** tudo que vem do teclado é texto. `Integer.parseInt` transforma o texto `"27"` no número `27`, para dar para fazer conta.

DICA

O `Scanner` também tem o `nextInt()`, mas ele deixa o Enter "sobrando", e o próximo `nextLine()` pega esse Enter vazio. É uma das pegadinhas mais famosas do Java. Ler sempre com `nextLine()` e converter depois evita o problema.

E se a pessoa digitar letras?

Experimente responder a idade com `vinte`. O programa para com um erro:

terminal

```
$ java Conversa.java
Qual é o seu nome? Marina
Quantos anos você tem? vinte
Exception in thread "main" java.lang.NumberFormatException: For input
string: "vinte"
```

Para tratar esse erro sem o programa quebrar, existe o `try` e `catch`:
"tente fazer isto; se der esse erro, faça aquilo":

Conversa.java

```
try {
    int idade = Integer.parseInt(entrada.nextLine());
    System.out.println("Daqui a 10 anos: " + (idade + 10));
} catch (NumberFormatException e) {
    System.out.println("Digite a idade só com números.");
}
```

DESAFIO

Faça um programa que pergunta o preço de um produto e a quantidade, e mostra o total com duas casas decimais. Se a pessoa digitar algo que não é número, mostre um aviso em vez de deixar o programa quebrar. Dica: para ler decimais, use `Double.parseDouble`.

O caixa da padaria

Variáveis, decisões, repetições, listas, métodos e classes, tudo trabalhando junto num programa de verdade: um caixa que monta o pedido e imprime o cupom.

O que o programa faz

Ele mostra o cardápio, pergunta o número do produto e a quantidade, repete até a pessoa digitar 0 e então imprime o cupom com o total. Veja uma conta inteira, com o que foi digitado em amarelo:

```
terminal

$ java Caixa.java
=== Padaria Pão Quente ===
1) Pão francês      R$  0,75
2) Café coado       R$  4,50
3) Pão de queijo    R$  3,00
4) Bolo de milho    R$  6,00
Produto (0 fecha a conta): 1
Quantidade: 10
Produto (0 fecha a conta): 2
Quantidade: 2
Produto (0 fecha a conta): 7
Número fora do cardápio.
Produto (0 fecha a conta): 4
Quantidade: 1
Produto (0 fecha a conta): 0
----- CUPOM -----
10x Pão francês      R$   7,50
 2x Café coado       R$   9,00
 1x Bolo de milho    R$   6,00
TOTAL                R$  22,50
Obrigado e volte sempre!
```

Crie uma pasta `projeto2` com um arquivo `Caixa.java`. O código vem em três partes.

Parte 1: o cardápio

```
Caixa.java

1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4
5 public class Caixa {
6     public static void main(String[] args) {
7         List<Produto> cardapio = List.of(
8             new Produto("Pão francês", 0.75),
9             new Produto("Café coado", 4.50),
10            new Produto("Pão de queijo", 3.00),
11            new Produto("Bolo de milho", 6.00));
12        ArrayList<Item> pedido = new ArrayList<>();
13        Scanner entrada = new Scanner(System.in);
14
15        System.out.println("=== Padaria Pão Quente ===");
16        for (int i = 0; i < cardapio.size(); i++) {
17            Produto p = cardapio.get(i);
18            System.out.printf("%d) %-15s R$ %5.2f%n",
19                i + 1, p.nome(), p.preco());
20        }
```

- **Linhas 7 a 11:** o cardápio é uma `List.of`, uma lista que não muda, cheia de objetos `Produto`.
- **Linha 12:** o pedido é uma `ArrayList`, porque vai crescer a cada item.
- **Linha 18:** no `printf`, `%-15s` escreve o texto ocupando 15 espaços, alinhado à esquerda, e `%5.2f` escreve o número em 5 espaços com duas casas. É isso que deixa as colunas alinhadas.
- **Linha 19:** mostra `i + 1` porque a lista começa em 0, mas gente começa a contar do 1.

Parte 2: anotando o pedido

```
Caixa.java

22     while (true) {
23         System.out.print("Produto (0 fecha a conta): ");
24         int escolha = lerNumero(entrada);
25         if (escolha == 0) {
26             break;
27         }
28         if (escolha < 1 || escolha > cardapio.size()) {
29             System.out.println("Número fora do cardápio.");
30             continue;
31         }
32         System.out.print("Quantidade: ");
33         int quantidade = lerNumero(entrada);
34         pedido.add(new Item(cardapio.get(escolha - 1), quantidade));
35     }
```

- **Linha 22:** `while (true)` repete para sempre, até alguém mandar parar.
- **Linha 26:** quem manda parar é o `break`, quando a pessoa digita 0.
- **Linha 30:** se o número não existe no cardápio, o `continue` pula o resto e pergunta de novo.
- **Linha 34:** `escolha - 1` converte o número que a pessoa vê (do 1 em diante) na posição da lista (do 0 em diante).

Parte 3: o cupom

```
Caixa.java

37     System.out.println("----- CUPOM -----");
38     double total = 0;
39     for (Item item : pedido) {
40         String nome = item.produto().nome();
41         System.out.printf("%2dx %-15s R$ %6.2f%n",
42             item.quantidade(), nome, item.subtotal());
43         total += item.subtotal();
44     }
```

```

45     System.out.printf("%-19s R$ %.2f\n", "TOTAL", total);
46     System.out.println("Obrigado e volte sempre!");
47 }
48
49 static int lerNumero(Scanner entrada) {
50     return Integer.parseInt(entrada.nextLine().trim());
51 }
52 }
53
54 record Produto(String nome, double preco) {}
55
56 record Item(Produto produto, int quantidade) {
57     double subtotal() {
58         return produto.preco() * quantidade;
59     }
60 }

```

- **Linhas 39 a 44:** passa por cada item do pedido, escreve a linha do cupom e vai somando no `total`.
- **Linhas 49 a 51:** o método `lerNumero` lê uma linha e converte para número. Como isso acontece duas vezes, virou método. O `trim()` tira espaços que a pessoa digitou sem querer.
- **Linha 54:** o `Produto` é um `record`, o atalho do capítulo 15.
- **Linhas 56 a 60:** o `Item` junta um produto e a quantidade, e sabe calcular o próprio subtotal.

DESAFIO

1. Hoje, se a pessoa digitar uma letra, o programa quebra. Use o `try` e `catch` do capítulo 16 dentro do `lerNumero` para pedir de novo.
2. Dê 10% de desconto quando o total passar de R\$ 50,00.
3. No fim, pergunte a forma de pagamento (dinheiro, Pix ou cartão) e mostre no cupom.

// PARTE 3

Juntando os dois

A página que você fez em HTML vai ser entregue por um programa seu, em Java. E o formulário finalmente vai ter alguém do outro lado para receber as mensagens.

18 Como o navegador conversa com o Java

19 Projeto 3: seu servidor em Java

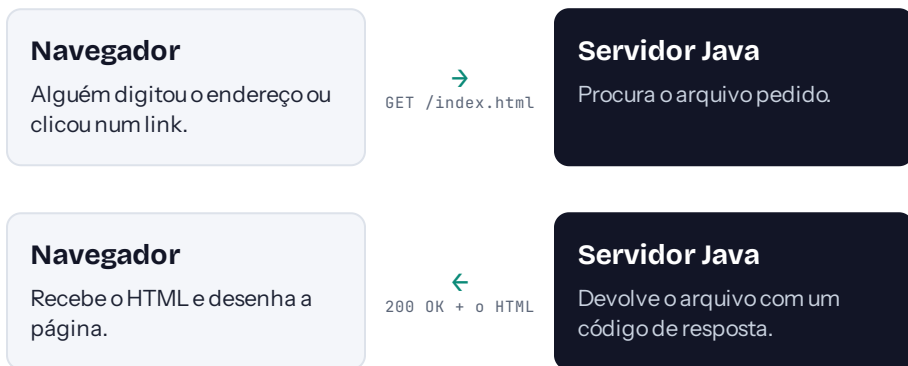
20 Próximos passos

Como o navegador conversa com o Java

Antes de escrever o servidor, vale entender a conversa que acontece toda vez que alguém abre uma página. Ela tem regras simples, e são sempre as mesmas.

Pedido e resposta

A conversa entre navegador e servidor segue um combinado chamado **HTTP**. O navegador sempre começa, com um **pedido**. O servidor sempre termina, com uma **resposta**:



Para cada arquivo da página (o HTML, o CSS, cada imagem), acontece um pedido e uma resposta. Uma página simples como a da padaria gera dois: um para o `index.html` e outro para o `estilo.css`.

As partes de um endereço

`http://localhost:8080/contato`

`http://`
o combinado

`localhost`
qual computador

`:8080`
qual porta

`/contato`
qual caminho

`localhost` quer dizer "este computador aqui". A **porta** é como o número do apartamento num prédio: o mesmo computador pode ter vários programas esperando pedidos, cada um numa porta. E o **caminho** diz o que está sendo pedido.

GET e POST

Todo pedido tem um **método**, que diz a intenção:

- **GET** pede alguma coisa. É o que acontece quando você abre uma página ou clica num link.
- **POST** envia alguma coisa. É o que acontece quando você aperta o botão de um formulário com `method="post"`.

Os códigos de resposta

Toda resposta começa com um número de três dígitos que diz como foi:

CÓDIGO	SIGNIFICA
200	Deu certo, aqui está.
404	Não encontrei o que você pediu.
405	Esse caminho existe, mas não aceita esse método.
500	Deu erro aqui do lado do servidor.

Quando a resposta não é uma página: JSON

Nem toda resposta é HTML. Muitas vezes, o navegador ou um aplicativo de celular só quer um dado: o preço, o horário, a lista de produtos. Para isso existe o **JSON**, um jeito simples de escrever dados que qualquer linguagem entende:

A screenshot of a code editor window with a dark background. The title bar at the top shows three colored circles (yellow, green, grey) followed by the text "resposta.json". The main area of the editor displays a JSON object: {"hora": 10, "aberto": true}. The text is color-coded: "hora" is in light blue, 10 is in light green, "aberto" is in light blue, and true is in light green.

```
resposta.json

{"hora": 10, "aberto": true}
```

Texto entre aspas, número sem aspas, `true` ou `false` para sim e não. Quando você abre o aplicativo do banco e aparece o saldo, quase sempre foi um JSON parecido com esse que chegou.

DICA

No Chrome, aperte **F12** e abra a aba **Network** (Rede). Recarregue a página e você vê cada pedido e cada resposta, com o método, o código e o tempo que levou. É a ferramenta que todo desenvolvedor usa para descobrir por que algo não carregou.

Seu servidor em Java

Um programa Java de pouco mais de cem linhas vai colocar a página da padaria no ar no seu computador, receber as mensagens do formulário e ainda responder em JSON.

Monte a pasta

Crie uma pasta `projeto3`. Dentro dela, crie uma pasta `site` e copie para lá os dois arquivos do Projeto 1:

```
projeto3/  
├── Servidor.java  
└── site/  
    ├── index.html  
    └── estilo.css
```

O código do `Servidor.java` vem em seis partes. Leia com calma: aqui aparecem algumas coisas novas, e cada uma está explicada.

Parte 1: preparando o servidor

```
Servidor.java  
  
1 import static java.nio.charset.StandardCharsets.UTF_8;  
2  
3 import com.sun.net.httpserver.HttpExchange;  
4 import com.sun.net.httpserver.HttpServer;  
5 import java.io.IOException;  
6 import java.io.OutputStream;  
7 import java.net.InetSocketAddress;  
8 import java.net.URLDecoder;
```

```

9  import java.nio.file.Files;
10 import java.nio.file.Path;
11 import java.time.LocalDateTime;
12 import java.util.HashMap;
13 import java.util.Map;
14
15 public class Servidor {
16     static final Path PASTA = Path.of("site");
17
18     public static void main(String[] args) throws IOException {
19         HttpServer servidor =
20             HttpServer.create(new InetSocketAddress(8080), 0);

```

- **Linha 1:** o `import static` deixa escrever só `UTF_8` no lugar de `StandardCharsets.UTF_8`. UTF-8 é o jeito de transformar texto em bytes que aceita acentos.
- **Linhas 3 e 4:** o `HttpServer` já vem dentro do Java. É um servidor simples, perfeito para aprender.
- **Linha 16:** a constante `PASTA` diz de onde o servidor pode entregar arquivos.
- **Linha 18:** o `throws IOException` avisa que o programa pode dar erro de entrada e saída, como a porta já estar ocupada.
- **Linhas 19 e 20:** cria o servidor na porta 8080.

Parte 2: entregando a página

```

● ● ● Servidor.java

22 // 1. Entrega os arquivos da pasta site (HTML, CSS...)
23 servidor.createContext("/", troca -> {
24     String caminho = troca.getRequestURI().getPath();
25     if (caminho.equals("/")) {
26         caminho = "/index.html";
27     }
28     String nomeDoArquivo = caminho.substring(1);
29     Path arquivo = PASTA.resolve(nomeDoArquivo).normalize();
30     boolean dentroDaPasta = arquivo.startsWith(PASTA);
31     if (!dentroDaPasta || !Files.isRegularFile(arquivo)) {

```

```

32     responder(troca, 404, "text/plain", "Não encontrado");
33     return;
34 }
35 byte[] conteudo = Files.readAllBytes(arquivo);
36 responder(troca, 200, tipoDoArquivo(arquivo), conteudo);
37 });

```

O `createContext` da linha 23 diz: "quando chegar um pedido neste caminho, rode este bloco". O `troca -> { ... }` é uma **lambda**, um bloco de código entregue como se fosse um valor. A variável `troca` carrega tudo da conversa: o pedido que chegou e a resposta que vai sair.

- **Linhas 24 a 27:** pega o caminho pedido. Quem pede só `/` recebe o `index.html`.
- **Linhas 28 e 29:** tira a barra do começo e monta o caminho do arquivo dentro da pasta `site`. O `normalize()` resolve os `..` do caminho.
- **Linhas 30 a 34:** se o arquivo pedido fica fora da pasta `site`, ou não existe, a resposta é 404.
- **Linhas 35 e 36:** lê o arquivo e responde com o código 200.

ATENÇÃO

As linhas 30 a 34 são de segurança. Sem elas, qualquer pessoa poderia pedir `../Servidor.java` e baixar o código do seu servidor, ou coisa pior. Esse ataque tem nome, **path traversal**, e é dos mais antigos da internet.

Parte 3: recebendo o formulário

● ● ● Servidor.java

```

39 // 2. Recebe o formulário de contato
40 servidor.createContext("/contato", troca -> {
41     if (!troca.getRequestMethod().equals("POST")) {
42         responder(troca, 405, "text/plain", "Use o formulário.");
43         return;
44     }

```

```

45     byte[] bytes = troca.getRequestBody().readAllBytes();
46     String corpo = new String(bytes, UTF_8);
47     Map<String, String> campos = lerFormulario(corpo);
48     String nome = campos.getOrDefault("nome", "");
49     String mensagem = campos.getOrDefault("mensagem", "");
50     System.out.println("Mensagem de " + nome + ": " + mensagem);
51
52     String html = """
53         <link rel="stylesheet" href="/estilo.css">
54         <main>
55             <h1>Obrigado, %s!</h1>
56             <p>Recebemos a sua mensagem.</p>
57             <a href="/">Voltar para o site</a>
58         </main>
59         """.formatted(escapar(nome));
60     responder(troca, 200, "text/html", html);
61 }

```

- **Linhas 41 a 44:** esse caminho só aceita POST. Se alguém abrir `/contato` direto no navegador (um GET), recebe o código 405.
- **Linhas 45 a 47:** lê o corpo do pedido e separa os campos. Ele chega assim: `nome=Ana&mensagem=Quero+20+por%C3%A3es`. O `+` é um espaço e o `%C3%A3` é o "ã".
- **Linha 50:** escreve a mensagem no terminal. Num sistema de verdade, aqui ela seria salva num banco de dados ou enviada por e-mail.
- **Linhas 52 a 59:** as três aspas abrem um **bloco de texto**, com várias linhas. O `%s` é trocado pelo nome da pessoa no `formatted`.

ATENÇÃO

Repare no `escapar(nome)` da linha 59. Se alguém digitar código HTML no campo do nome, ele vai aparecer como texto, e não vai rodar. Sem isso, um espertinho poderia colocar um script na sua página. Esse ataque se chama **XSS**. Regra de ouro: tudo que vem do usuário é suspeito até que se prove o contrário.

Parte 4: respondendo em JSON e ligando tudo

● ● ● Servidor.java

```
63 // 3. Responde em JSON se a padaria está aberta agora
64 servidor.createContext("/api/horario", troca -> {
65     int hora = LocalTime.now().getHour();
66     boolean aberto = hora >= 7 && hora < 19;
67     String json = "{\"hora\": " + hora
68         + ", \"aberto\": " + aberto + "}";
69     responder(troca, 200, "application/json", json);
70 });
71
72 servidor.start();
73 System.out.println("No ar! Abra http://localhost:8080");
74 }
```

O caminho `/api/horario` não devolve uma página: devolve um JSON dizendo a hora e se a padaria está aberta, usando a mesma regra do capítulo 11. Na linha 72, o `start()` liga o servidor, que fica esperando pedidos até você mandar parar.

Parte 5: as ferramentas

● ● ● Servidor.java

```
76 static String tipoDoArquivo(Path arquivo) {
77     String nome = arquivo.getFileName().toString();
78     if (nome.endsWith(".html")) return "text/html";
79     if (nome.endsWith(".css")) return "text/css";
80     if (nome.endsWith(".js")) return "text/javascript";
81     if (nome.endsWith(".png")) return "image/png";
82     if (nome.endsWith(".jpg")) return "image/jpeg";
83     return "application/octet-stream";
84 }
85
86 static Map<String, String> lerFormulario(String corpo) {
87     Map<String, String> campos = new HashMap<>();
88     for (String par : corpo.split("&")) {
89         String[] partes = par.split("=", 2);
```

```

90     String chave = URLDecoder.decode(partes[0], UTF_8);
91     String valor = partes.length > 1
92         ? URLDecoder.decode(partes[1], UTF_8)
93         : "";
94     campos.put(chave, valor);
95 }
96 return campos;
97 }
98
99 static String escapar(String texto) {
100     return texto.replace("&", "&amp;")
101         .replace("<", "&lt;")
102         .replace(">", "&gt;")
103         .replace("\"", "&quot;");
104 }

```

- **tipoDoArquivo** : avisa o navegador o que está chegando, pelo cabeçalho `Content-Type` . Sem isso, o navegador pode ignorar o CSS.
- **lerFormulario** : quebra o corpo do formulário nos `&` , depois cada pedaço no `=` , e guarda tudo num `Map` , uma estrutura de pares nome e valor, como um dicionário. O `URLDecoder` desfaz os `+` e os `%` .
- **escapar** : troca os caracteres especiais do HTML (`<` , `>` , `&` e aspas) pelos códigos que só aparecem como texto.

Parte 6: mandando a resposta

```

● ● ● Servidor.java
106 static void responder(HttpExchange troca, int status,
107     String tipo, String texto) throws IOException {
108     responder(troca, status, tipo, texto.getBytes(UTF_8));
109 }
110
111 static void responder(HttpExchange troca, int status,
112     String tipo, byte[] conteudo) throws IOException {
113     if (tipo.startsWith("text/") || tipo.endsWith("json")) {
114         tipo += "; charset=utf-8";
115     }
116     troca.getResponseHeaders().set("Content-Type", tipo);

```

```
117     troca.sendResponseHeaders(status, conteudo.length);
118     try (OutputStream saida = troca.getResponseBody()) {
119         saida.write(conteudo);
120     }
121 }
122 }
```

Existem dois métodos `responder`, com o mesmo nome. O Java escolhe qual usar pelo que você passa: com texto, roda o primeiro, que converte o texto em bytes e chama o segundo. Isso se chama **sobrecarga**. O segundo monta o cabeçalho, avisa o código e o tamanho da resposta (linha 117) e escreve os bytes. O `try` com parênteses da linha 118 fecha a resposta sozinho no final.

No ar!

No terminal do VS Code, entre na pasta `projeto3` e rode o servidor:

```
terminal
$ cd projeto3
$ java Servidor.java
No ar! Abra http://localhost:8080
```

Abra `http://localhost:8080` no navegador: é a página da padaria, agora entregue pelo seu programa. Preencha o formulário e clique em **Enviar**:

localhost:8080/contato

Obrigado, Ana!

Recebemos a sua mensagem.

[Voltar para o site](#)

E no terminal, onde o servidor está rodando, aparece a mensagem:

```
terminal

No ar! Abra http://localhost:8080
Mensagem de Ana: Quero 20 pães para amanhã às 7h
```

Agora abra `http://localhost:8080/api/horario`. Em vez de uma página, aparece só o dado:

```
localhost:8080/api/horario

{"hora": 10, "aberto": true}
```

Para desligar o servidor, clique no terminal e aperte **Ctrl** + **C**.

ATENÇÃO

Página com "Não encontrado"? Você provavelmente rodou o servidor de outra pasta. O programa procura a pasta `site` a partir de onde foi iniciado, então rode sempre de dentro da `projeto3`.

Erro dizendo Address already in use ? Já tem um programa usando a porta 8080, talvez o próprio servidor rodando em outro terminal. Feche o outro ou troque o 8080 por 8081 na linha 20.

DICA

No mercado, quase ninguém escreve um servidor do zero assim: usa-se um framework, e o mais famoso do Java é o **Spring Boot**. Mas agora você sabe o que ele faz por baixo dos panos, e isso faz muita diferença na hora de aprender.

DESAFIO

1. Guarde cada mensagem num arquivo `mensagens.txt`. Pesquise por `Files.writeString` com a opção `StandardOpenOption.APPEND`.
2. Crie o caminho `/api/cardapio`, que devolve os produtos e preços em JSON.
3. Crie uma página `404.html` bonita, no estilo da padaria, e entregue ela no lugar do texto "Não encontrado".

Próximos passos

Você fez uma página, um programa e um servidor, e entendeu como os três conversam. Isso é a base de quase tudo na web. Daqui para frente, é escolher um caminho e aprofundar.

Coloque a sua página no ar

A página do Projeto 1 é só HTML e CSS, e esse tipo de site pode ser publicado de graça. Os dois caminhos mais conhecidos:

- **GitHub Pages:** você guarda o projeto no GitHub e ele vira um site com endereço próprio. De quebra, você aprende Git, que todo emprego vai pedir.
- **Netlify:** você arrasta a pasta do projeto para a página deles e em segundos tem um endereço para mandar para os amigos.

O formulário só vai funcionar de verdade com um servidor por trás, como o do Projeto 3. Para colocar um programa Java na internet, você vai precisar de um serviço de hospedagem em nuvem, e esse é um ótimo próximo projeto.

O que estudar agora

CAMINHO	PRÓXIMOS ASSUNTOS
Front-end (a parte que se vê)	CSS a fundo (flexbox, grid, responsividade), depois JavaScript, depois um framework como o React.
Back-end (a parte do servidor)	Java a fundo (coleções, exceções, orientação a objetos), depois SQL e banco de dados, depois Spring Boot.
Para os dois	Git e GitHub, lógica de programação e inglês técnico para ler documentação.

Pratique com projetos seus

A melhor forma de fixar é construir algo que você mesmo usaria. Algumas ideias:

- A página de um negócio da sua família ou do seu bairro.
- O seu portfólio, com os três projetos deste livro.
- Uma calculadora de orçamento em Java para um serviço que você conhece.
- Uma lista de tarefas que salva num arquivo.

Mostra pra gente

Terminou um projeto do livro, ou fez a sua versão de algum? Posta e marca o **@developers_gk** no Instagram. Vou adorar ver o que você construiu.

// para consultar sempre

Colinha de bolso

O essencial deste livro em duas tabelas. Imprima, salve no celular, cole do lado do computador.

HTML

TAG	PARA QUE SERVE
<code><h1></code> a <code><h6></code>	Títulos, do mais importante ao menos
<code><p></code>	Parágrafo
<code></code> <code></code>	Texto importante, texto com ênfase
<code></code>	Link
<code></code>	Imagem, sempre com descrição
<code></code> <code></code> <code></code>	Lista com bolinha, numerada, item
<code><header></code> <code><footer></code>	Topo e rodapé
<code><nav></code> <code><main></code>	Menu e conteúdo principal
<code><section></code> <code><article></code>	Seção com assunto, conteúdo independente
<code><form></code>	Formulário (<code>action</code> e <code>method</code>)
<code><label></code> <code><input></code>	Nome do campo e campo de uma linha
<code><textarea></code> <code><select></code>	Campo de várias linhas, lista de opções
<code><button type="submit"></code>	Botão que envia o formulário
<code><link rel="stylesheet"></code>	Liga o arquivo de CSS

Java

ESCREVA	PARA
<code>int n = 3;</code>	Guardar um inteiro (<code>double</code> , <code>boolean</code> , <code>String</code>)
<code>final int MAX = 10;</code>	Criar uma constante
<code>System.out.println(x);</code>	Mostrar e pular linha
<code>System.out.printf("%.2f", x);</code>	Mostrar com duas casas decimais
<code>if (a > b) { } else { }</code>	Decidir
<code>texto.equals("sim")</code>	Comparar textos (nunca com <code>==</code>)
<code>for (int i = 0; i < 5; i++)</code>	Repetir um número certo de vezes
<code>while (condicao) { }</code>	Repetir enquanto for verdade
<code>for (String s : lista)</code>	Passar por cada item da lista
<code>new ArrayList<String>()</code>	Lista que cresce (<code>add</code> , <code>get</code> , <code>size</code>)
<code>static int dobro(int x)</code>	Criar um método que devolve um valor
<code>new Produto("Pão", 0.75)</code>	Criar um objeto
<code>record Ponto(int x, int y) {}</code>	Classe só de dados, em uma linha
<code>Integer.parseInt("27")</code>	Transformar texto em número
<code>try { } catch (Exception e) { }</code>	Tratar um erro sem o programa quebrar

Precisa de um site, app ou sistema? Chama no direct.

Este livro é gratuito porque eu acredito que todo mundo devia ter a chance de começar. E quando o projeto é grande demais para fazer sozinho, ou não dá tempo, é isso que eu faço todos os dias:

Sites

Páginas de empresa e landing pages feitas para o Google encontrar.

Aplicativos

Apps para o seu cliente usar direto do celular.

Sistemas web

Estoque, pedidos e relatórios, sem planilha.

Bancos digitais

Conta, cartão e app com a marca da sua empresa.

@developers_gk no Instagram



Gregório Karabachian
@developers_gk