

JAVA DEVELOPMENT

# ADICIONANDO LÓGICA AO PROGRAMA JAVA



# 3

## LISTA DE FIGURAS

|  |    |
|--|----|
| Figura 1 – Fluxo para a instrução if .....                       | 9  |
| Figura 2 – Fluxo para a instrução if – else .....                | 10 |
| Figura 3 – Fluxo para a instrução if – else alinhados .....      | 11 |
| Figura 4 – Ajuste do exercício anterior .....                    | 11 |
| Figura 5 – Exemplo de leitura de dados .....                     | 14 |
| Figura 6 – Ajuste do exercício anterior .....                    | 15 |
| Figura 7 – Resultado da execução do programa do exercício .....  | 15 |
| Figura 8 – Exercício do par ou ímpar .....                       | 15 |
| Figura 9 – Exercício das laranjas .....                          | 16 |
| Figura 10 – Execução do programa do exercício das laranjas ..... | 17 |
| Figura 11 – Exemplo de operador ternário .....                   | 18 |
| Figura 12 – Exemplo de menu com if-else .....                    | 19 |
| Figura 13 – Exemplo de menu com switch case .....                | 20 |

## LISTA DE TABELAS

|  |    |
|--|----|
| Tabela 1 – Operadores de comparação.....                               | 5  |
| Tabela 2 – Resumo dos operadores de comparação e negação.....          | 7  |
| Tabela 3 – Métodos da classe Scanner para ler os dados primitivos..... | 13 |

EXEMPLO

## SUMÁRIO

|   |    |
|---|----|
| ADICIONANDO LÓGICA AO PROGRAMA JAVA.....      | 5  |
| 1 OPERADORES DE IGUALDADE E RELACIONAIS ..... | 5  |
| 2 OPERADORES LÓGICOS.....                     | 6  |
| 3 FLUXO DE CONTROLE.....                      | 8  |
| 4 ENTRADA E SAÍDA DE DADOS .....              | 12 |
| 5 OPERADOR TERNÁRIO .....                     | 17 |
| 6 SWITCH CASE .....                           | 18 |
| REFERÊNCIAS.....                              | 22 |

## ADICIONANDO LÓGICA AO PROGRAMA JAVA

### 1 OPERADORES DE IGUALDADE E RELACIONAIS

Podemos comparar variáveis, valores ou outros tipos de informações utilizando os operadores relacionais, que retornam um valor booleano, verdadeiro (*true*) ou falso (*false*). Verificaremos, por exemplo, se um aluno passou de ano, caso a sua média seja maior ou igual a 6. Ou se o valor inserido por um usuário é válido para sacar dinheiro em uma conta bancária.

Apresentamos os operadores de comparação da linguagem Java (Tabela “Operadores de comparação”).

Tabela 1 – Operadores de comparação

| Operador | Significado    | Exemplo |
|----------|----------------|---------|
| ==       | Igual          | x == y  |
| !=       | Diferente      | x != y  |
| >        | Maior          | x > y   |
| <        | Menor          | x < y   |
| >=       | Maior ou igual | x >= y  |
| <=       | Menor ou igual | x <= y  |

Fonte: Elaborado pelo autor (2022)

Aqui, temos um exemplo para verificar se o aluno passou de ano:

```
float media = 7;
```

```
boolean passouAno = media >= 6;
```

Primeiro, a variável **media** foi inicializada com um valor, no caso 7. Depois, foi declarada uma variável denominada **passouAno**, que recebe o resultado da expressão **media >= 6**, ou seja, analisa se a média é maior ou igual a 6, o que é verdadeiro no exemplo.

## 2 OPERADORES LÓGICOS

Operadores lógicos são utilizados para formar expressões de comparação com mais de um termo, resultando também em um valor booleano. Por exemplo, para um aluno passar de ano, ele deve ter a média maior ou igual a 6 e uma frequência das aulas maior do que 75%.

Os operadores principais no Java utilizados para combinar as comparações são "&&" (e) e o "||" (ou). Existe também a negação "!" (Não).

O operador lógico **AND** (e) é representado pelo símbolo ( && ). Quando duas expressões booleanas utilizam o operador &&, o resultado é verdadeiro (true) somente se as duas expressões forem verdadeiras. Exemplo:

```
boolean passouAno = media >=6 && presenca > 75;
```

O exemplo acima combina duas comparações: a primeira testa se a **média** é maior ou igual a 6 (`media >= 6`) e a segunda verifica se a **presença** é maior do que 75 (`presenca > 75`). Se as duas comparações forem verdadeiras, a variável **passouAno** recebe verdadeiro (true), em qualquer outra situação, o valor será falso (false).

O operador **OR** (ou) é representado pelo símbolo ( || ). Essas expressões combinadas retornam como verdadeiro caso uma das condições seja verdadeira. Por exemplo, uma cidade permite que as pessoas com menos de 5 anos e mais de 65 anos utilizem o transporte público gratuitamente. Uma expressão que pode solucionar esse problema é:

```
boolean gratuito = idade < 5 || idade > 65;
```

Dessa forma, a expressão acima verifica se a idade é menor do que 5 ou se é maior do que 65 anos. Se uma das verificações for verdadeira, a pessoa pode utilizar o transporte público de forma gratuita.

O operador **NOT** (não) é representado pelo símbolo de exclamação ( ! ). Ele reverte o valor booleano, ou seja, se o valor ou a expressão resultar em verdadeiro, a negação o tornará falso. A mesma coisa ocorre para o falso, que reverte para o verdadeiro.

Por exemplo, é permitido tirar carta de motorista somente se a pessoa tem 18 anos ou mais. Assim, uma forma de criar a expressão é:

```
boolean podeDirigir = !(idade < 18);
```

Ou seja, primeiro, é realizada a expressão (idade < 18), verificando se a idade é menor do que 18 anos. Depois, o resultado é negado, tendo o seu valor invertido. Assim, se a pessoa tem 10 anos, por exemplo, o resultado da expressão (idade < 18) é verdadeiro. Depois, o resultado é invertido com o operador de negação (!), atribuindo à variável **podeDirigir** o valor **falso**. Claro que a expressão idade >=18 anos é mais simples e resolve o mesmo problema. Porém, o operador de negação pode ser útil em várias outras situações.

Observe um resumo dos resultados dos operadores, considerando as variáveis x e y do tipo booleano (Tabela “Resumo dos operadores de comparação e negação”).

Tabela 2 – Resumo dos operadores de comparação e negação

| X     | Y     | x && y | x    y | !x    |
|-------|-------|--------|--------|-------|
| True  | True  | True   | True   | False |
| True  | False | False  | True   | False |
| False | True  | False  | True   | True  |
| False | False | False  | False  | True  |

Fonte: Elaborado pelo autor (2022)

### 3 FLUXO DE CONTROLE

Acabamos de ver as expressões que fazem as comparações. Agora, precisamos controlar o fluxo do código com seleções e loops. Pois, por exemplo, caso o aluno tenha a nota e a frequência para passar de ano, iremos executar um conjunto de instruções para permitir que o aluno se matricule para o próximo ano, caso contrário, devemos executar os comandos para permitir que o aluno faça a prova de recuperação. E isso será implementado com os famosos `if (se)`.

Mas, antes, vamos lembrar os blocos. Vimos que as chaves `{ }` delimitam o início e o fim das classes e dos métodos. Assim, é possível identificar os comandos que pertencem à classe e ao método. Os blocos podem ser aninhados dentro de outro bloco, como, por exemplo, a classe que possui um método. As chaves serão importantes também para delimitar os comandos que fazem parte do comando de seleção e outros que veremos mais à frente.

A instrução condicional no Java é realizada pela palavra reservada **`if`**. A sintaxe básica é:

**`if ( condição ) instrução;`**

A condição deve ser colocada entre os parênteses e pode conter várias comparações combinadas com os operadores lógicos, que, no final, resultam em um valor booleano.

Somente quando a condição for verdadeira é que a instrução após o **`if`** será executada. Caso a condição seja falsa, a instrução após o **`if`** não será executada, seguindo o fluxo do código normalmente.

As chaves para delimitar as instruções que fazem parte da estrutura de seleção são necessárias caso haja mais de uma instrução para ser executada. Sintaxe:

```
if (condição) {  
    instrução 1;  
    instrução 2;  
}
```



Exemplo do aluno que passou ou não de ano:

```
if (media >=6 && frequencia >75) {  
    System.out.println("Passou de ano");  
    System.out.println("Pode realizar a matrícula");  
}
```

Observe o fluxo para a instrução de seleção (Figura “Fluxo para a instrução if”).

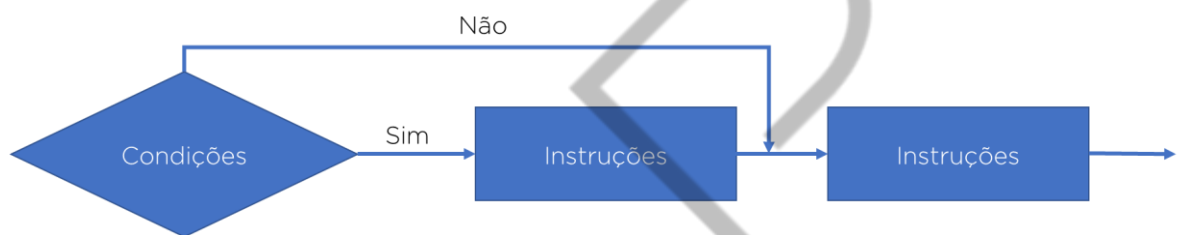


Figura 1 – Fluxo para a instrução if  
Fonte: Elaborado pelo autor (2022)

Existe também o comando **else**, que é opcional, junto do **if**. Se a condição do **if** for falsa, o bloco de código do **else** será executado. Sintaxe:

```
if ( condição ) {  
    Instruções do if;  
} else {  
    Instruções do else;  
}
```

Exemplo:

```
if (media >=6 && frequencia >75) {  
    System.out.println("Passou de ano");  
    System.out.println("Pode realizar a matrícula");  
} else {  
    System.out.println("Deve realizar a prova de recuperação");  
}
```

Observe o fluxo para a instrução if – else (Figura “Fluxo para a instrução if – else”).

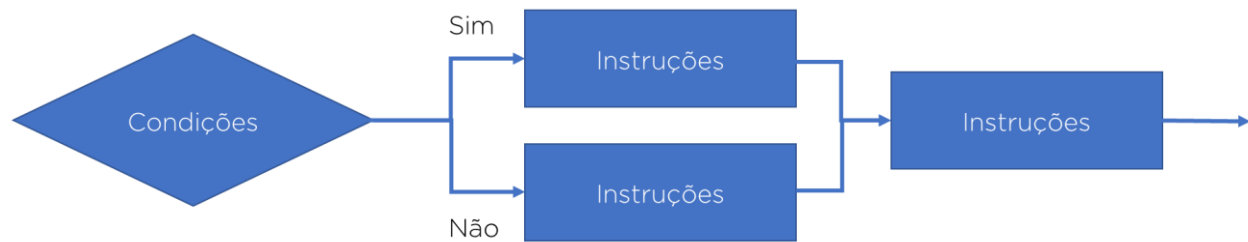


Figura 2 – Fluxo para a instrução if – else  
Fonte: Elaborado pelo autor (2022)

Também é possível encadear as instruções de seleção, deixando uma instrução de seleção dentro de outra. Como, por exemplo:

```
if (media > 6) {  
    System.out.println("A média é maior do que 6");  
} else if (media == 6) {  
    System.out.println("A média é igual a 6");  
} else {  
    System.out.println("A média é menor do que 6");  
}
```

Um if fica dentro de outro if (neste caso, do else). Sempre um else pertence ao if que estiver mais próximo dele. No exemplo, primeiro é verificado se a média é maior do que 6. Se não for, verifica-se se a média é igual a 6. Se não for, a instrução do último else é executada.

## Adicionando lógica ao programa Java

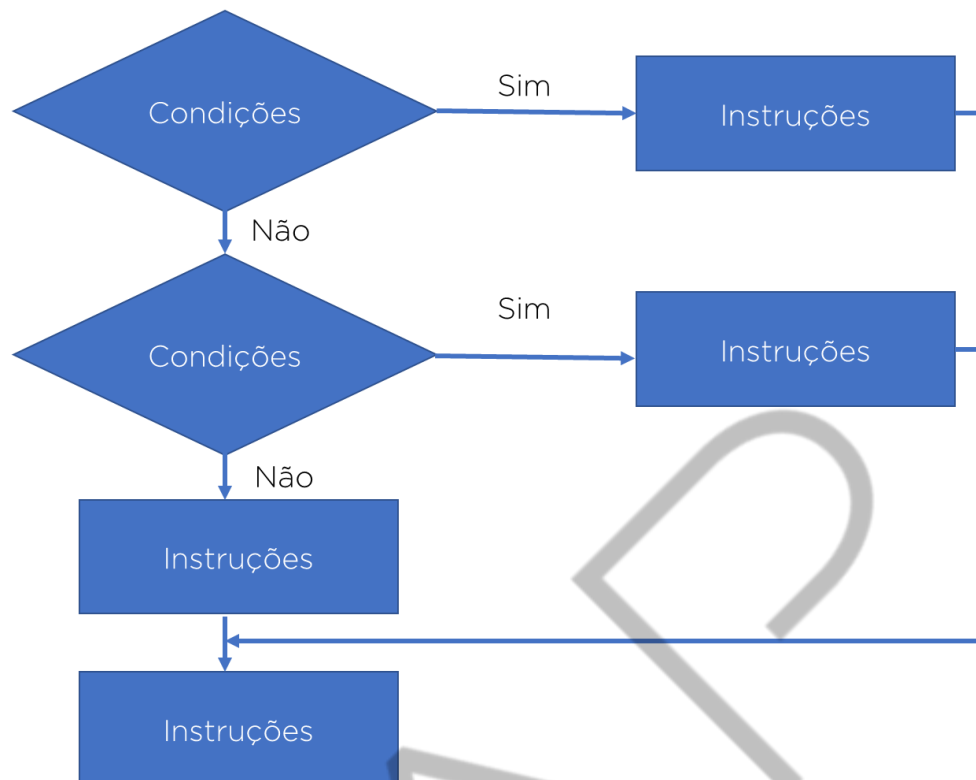


Figura 3 – Fluxo para a instrução if – else alinhados  
Fonte: Elaborado pelo autor (2022)

Agora, vamos ajustar o código anterior para mostrar se o aluno passou ou não de ano, dependendo da média final. Caso a média seja maior ou igual a 6, o aluno passou de ano. Se a média for menor do que 3, infelizmente, o aluno repetiu de ano. E se a média for maior ou igual a 3 e menor do que 6, ele está de exame.

```
package br.com.flap;

public class Exercício01 {

    public static void main(String[] args) {
        float ps = 7; //prova semestral
        float tcc = 5; //trabalho de conclusão de curso
        float a11 = 10; //avaliação intermediária 1
        float a12 = 4; //avaliação intermediária 2

        float media1 = (a11 + a12)/2; //média das avaliações intermediárias
        double mediaFinal = ps + 0.1 * tcc + 0.3 * media1 + 0.2;

        System.out.println("Média final é " + mediaFinal);

        if (mediaFinal >= 6){
            System.out.println("Passou de ano");
        } else if (mediaFinal <= 3 && mediaFinal < 6){
            System.out.println("Falta de exame");
        } else {
            System.out.println("Repetiu de ano");
        }
    }
}
```

A captura de tela mostra o IDE IntelliJ IDEA com o código Java acima. Abaixo do código, a janela de execução mostra a saída: "Média final é 6.0" e "Passou de ano". O status da execução indica que o processo terminou com o código de saída 0.

Figura 4 – Ajuste do exercício anterior  
Fonte: Elaborado pelo autor (2022)

Repare que não é necessário validar a última condição, já que se a média não for maior ou igual a 6 nem maior ou igual a 3 e menor do que 6, automaticamente, a média será menor do que 3, e o estudante não passou de ano. Talvez fosse interessante colocar mais uma condição para saber se a média não é maior do que 10, pois aí algo está errado, já que a média deve ser, no máximo, 10.

## 4 ENTRADA E SAÍDA DE DADOS

Até agora está fácil para este aluno. As notas dele nunca mudam e ele está sempre passando de ano. Brincadeiras à parte, está na hora de pedir para o usuário inserir as informações no programa Java.

O ideal de um programa, não importando em qual linguagem está implementado, é receber os dados do usuário, processá-los e depois mostrar o resultado. Até o momento, já sabemos como processar, com os operadores e seletores. Vimos também como exibir os dados para o usuário, por meio do comando **System.out.println();**,

Existe outra instrução, o **System.out.print();**, que mostra os dados para o usuário e não pula uma linha no final. Se olhar com cuidado, o comando se diferencia pelo **In** no final do **print**. O **In** quer dizer **line**, em inglês, de linha em português. Ou seja, com o **In**, o comando pula de linha no final do dado exibido. Sem ele, os dados serão mostrados na mesma linha.

Agora só falta conhecer o comando para ler os dados que serão inseridos pelo usuário. Para isso, precisamos da ajuda da classe **Scanner**.

**Scanner leitor = new Scanner(System.in);**

Neste momento, nós declaramos uma variável do tipo **Scanner** chamada **leitor** (veremos os tipos de variáveis de referência em breve). Depois, atribuímos um objeto do tipo do **Scanner** à variável, isso é a orientação a objetos, que vamos trabalhar nos próximos capítulos. Mas não se preocupe, o importante é entender que declaramos uma variável do tipo **Scanner** e colocamos um objeto desse tipo na variável, assim, podemos utilizá-lo para ler os dados inseridos pelo teclado do usuário.

A classe `Scanner` permite a leitura de dados que podem ser provenientes de várias fontes, como um arquivo ou dados digitados pelo usuário. Assim, o **`System.in`** determina que queremos ler os dados digitados pelo usuário.

Com a variável pronta, vamos utilizar os métodos da classe `Scanner` para ler os dados do usuário. Existem vários métodos que são utilizados para ler cada um dos tipos de dados possíveis:

```
Scanner leitor = new Scanner(System.in);
```

```
int idade = leitor.nextInt();
```

```
float nota = leitor.nextFloat();
```

No exemplo acima, o método **`nextInt()`** lê um número inteiro. E o método **`nextFloat()`**; lê um número do tipo `float`.

Tabela 3 – Métodos da classe `Scanner` para ler os dados primitivos

| Tipo primitivo | Método do Scanner          |
|----------------|----------------------------|
| Byte           | <code>nextByte()</code>    |
| Short          | <code>nextShort()</code>   |
| Int            | <code>nextInt()</code>     |
| Long           | <code>nextLong()</code>    |
| Float          | <code>nextFloat()</code>   |
| Double         | <code>nextDouble()</code>  |
| Boolean        | <code>nextBoolean()</code> |

Fonte: Elaborado pelo autor (2022)

A classe `Scanner` está localizada no pacote **`java.util`**. Assim, quando utilizamos classes que estejam em pacotes diferentes e não sejam do pacote básico (`java.lang`), é necessário utilizar a instrução **`import`**.

## Adicionando lógica ao programa Java

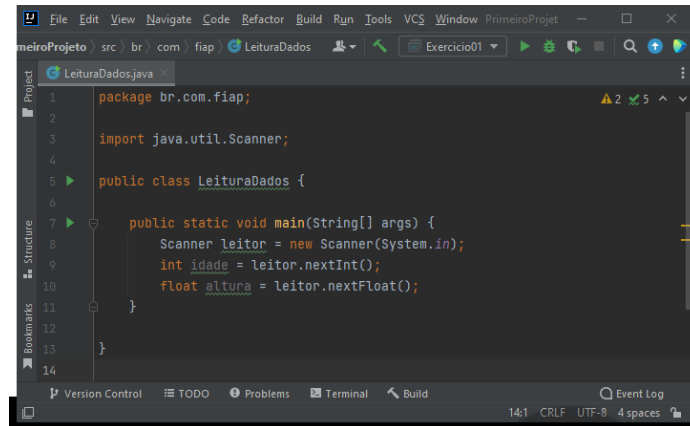
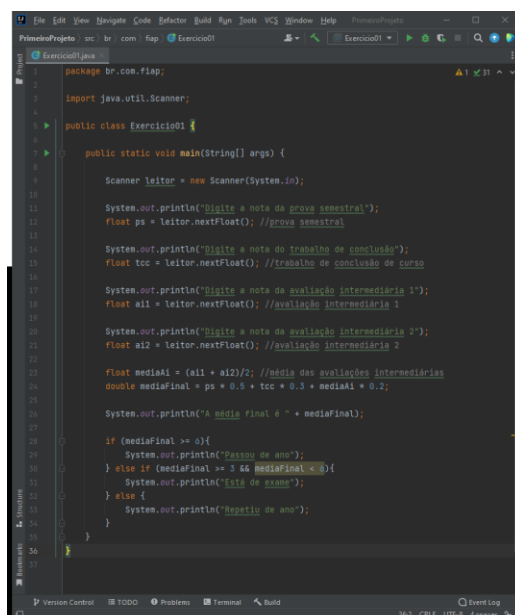


Figura 5 – Exemplo de leitura de dados  
Fonte: Elaborado pelo autor (2022)

Observe a instrução **import** logo após a definição do pacote da classe. Sempre a definição do pacote deve ser a primeira instrução da classe. Depois da palavra **import**, devemos dizer qual o pacote e a classe que queremos utilizar. Neste caso, pacote **java.util** e a classe **Scanner**.

Na classe, declaramos a variável **Scanner** na linha 8 e, depois, a utilizamos para ler um número inteiro (linha 9) e um número do tipo float (linha 10), armazenando os dados digitados pelo usuário em suas respectivas variáveis: **idade** e **altura**.

Sabendo disso, vamos ajustar o programa que calcula a média do aluno para que o usuário possa inserir todas as informações. Bora lá!



## Adicionando lógica ao programa Java

Figura 6 – Ajuste do exercício anterior  
Fonte: Elaborado pelo autor (2022)

Observe que, para que o usuário saiba o que deve digitar, nós utilizamos o **System.out.println()** para exibir a informação ao usuário antes de lê-lo. Quando executar, perceba que o programa fica esperando o usuário digitar o dado.

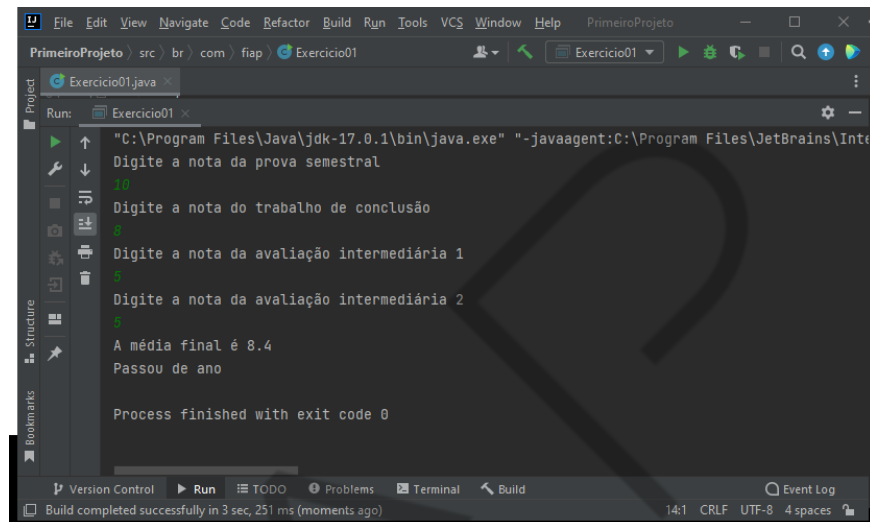


Figura 7 – Resultado da execução do programa do exercício

Agora, é a sua vez! Crie um programa Java para ler um número inteiro para determinar se o número é par ou ímpar.

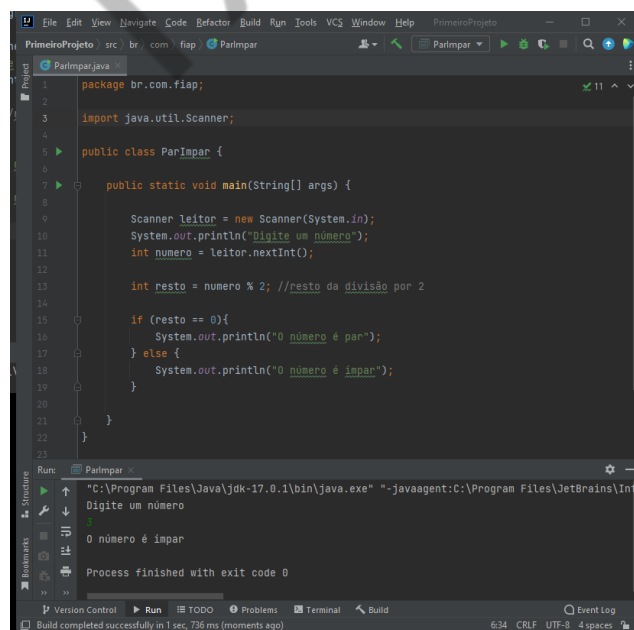
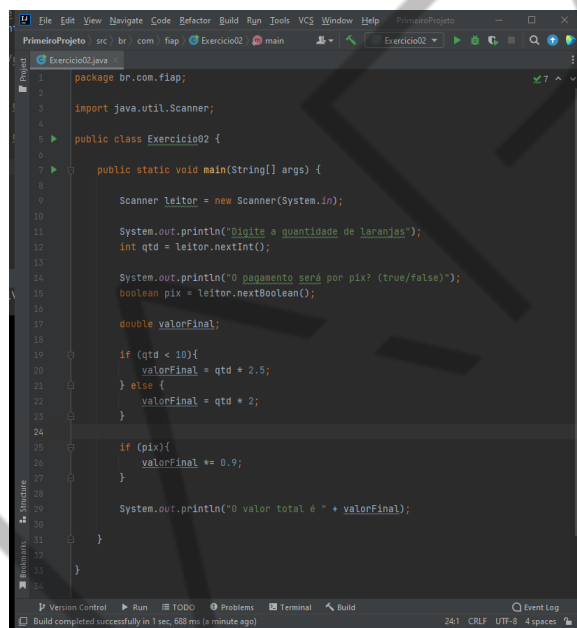


Figura 8 – Exercício do par ou ímpar  
Fonte: Elaborado pelo autor (2022)

## Adicionando lógica ao programa Java

Agora, para finalizar, implemente um programa que recebe a quantidade de laranjas que o cliente deseja comprar e se o pagamento será por pix ou não (boolean). Caso a quantidade de laranjas seja menor do que 10, o valor de cada laranja será R\$ 2,50. Caso seja maior ou igual a 10, o valor será R\$ 2,00. Se o pagamento for por pix, será adicionado um desconto de 10%.

Lembre-se de que para ser bom em algo você deve praticar! E na programação, não é diferente. É como andar de bicicleta, você não vai aprender vendo outra pessoa andando de bike.

The image shows a screenshot of an IDE window titled 'Exercicio02.java'. The code is as follows:

```
package br.com.fiap;

import java.util.Scanner;

public class Exercicio02 {

    public static void main(String[] args) {

        Scanner leitor = new Scanner(System.in);

        System.out.println("Digite a quantidade de laranjas");
        int qtd = leitor.nextInt();

        System.out.println("O pagamento será por pix? (true/false)");
        boolean pix = leitor.nextBoolean();

        double valorFinal;

        if (qtd < 10){
            valorFinal = qtd * 2.5;
        } else {
            valorFinal = qtd * 2;
        }

        if (pix){
            valorFinal *= 0.9;
        }

        System.out.println("O valor total é " + valorFinal);
    }
}
```

Figura 9 – Exercício das laranjas  
Fonte: Elaborado pelo autor (2022)

O programa lê o número de laranjas e, depois, um valor booleano. O usuário deve digitar o valor **true** ou **false**, caso contrário, o programa irá dar erro. Outro detalhe é no **if** do pix. Como a variável pix já é um valor booleano, não é preciso realizar nenhuma comparação. Observe um resultado possível da execução (Figura “Execução do programa do exercício das laranjas”).



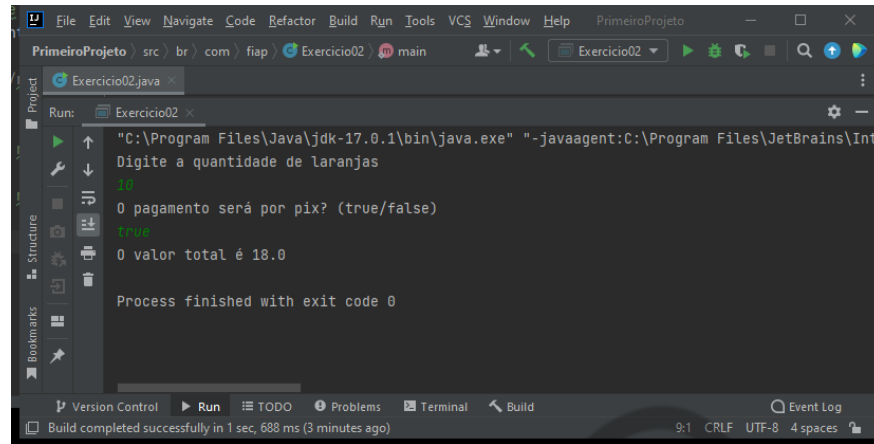


Figura 10 – Execução do programa do exercício das laranjas  
Fonte: Elaborado pelo autor (2022)

## 5 OPERADOR TERNÁRIO

Neste capítulo, vimos como utilizar uma estrutura de seleção com if – else. Durante o desenvolvimento de uma aplicação, é comum a necessidade de validar o código, uma ação simples que retorna um valor para duas possibilidades.

No exemplo da venda de laranjas, temos duas situações: a primeira na quantidade de laranjas que serão vendidas e a outra na forma de pagamento:

- Se a quantidade de laranjas for maior ou igual a 10, o valor da laranja é R\$ 2,00, senão, é R\$ 2,50.
- Se o pagamento for por pix, o cliente tem 10% de desconto.

Uma solução utilizando if – else foi vista na resolução do exercício, na Figura “Execução do programa do exercício das laranjas”. No código, é possível notar que a estrutura de seleção é bem simples, com uma única linha de código dentro do if ou do else.

Nesta situação, podemos utilizar o operador ternário, que funciona de uma forma muito parecida com o if – else, porém, com a diferença de que é preciso retornar um valor após o teste, em uma única linha.

A estrutura do operador ternário é:

**Condição ? se verdadeiro : se falso;**

## Adicionando lógica ao programa Java

A primeira parte do operador ternário (três partes) é a condição, seguida do caractere “?”; a segunda parte é o resultado utilizado caso a condição seja verdadeira, depois do caractere “:” já a terceira parte é o valor utilizado caso a condição seja falsa.

Dessa forma, há outra maneira para resolver o exercício anterior (Figura “Exemplo de operador ternário”).

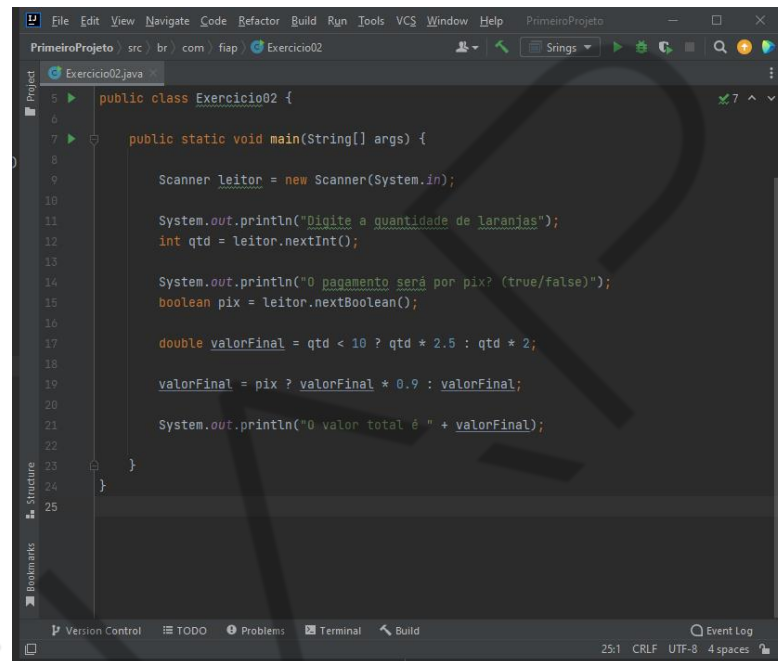


Figura 11 – Exemplo de operador ternário  
Fonte: Elaborado pelo autor (2022)

Na linha 17, está a condição ( $qtd < 10$ ) e, depois do caractere de interrogação (“?”), aparece o resultado caso a condição seja verdadeira, ( $qtd * 2.5$ ); depois do caractere dois pontos (“:”), foi definido o valor para a condição falsa ( $qtd * 2$ ) e o resultado é atribuído para a variável “valorFinal”.

A linha 19 é outro ponto do código no qual o operador ternário foi utilizado. Neste caso, a condição foi o valor da variável booleana pix. Se for verdadeiro, o valor final será 90% do valor final, caso contrário, será o próprio valor final.

## 6 SWITCH CASE

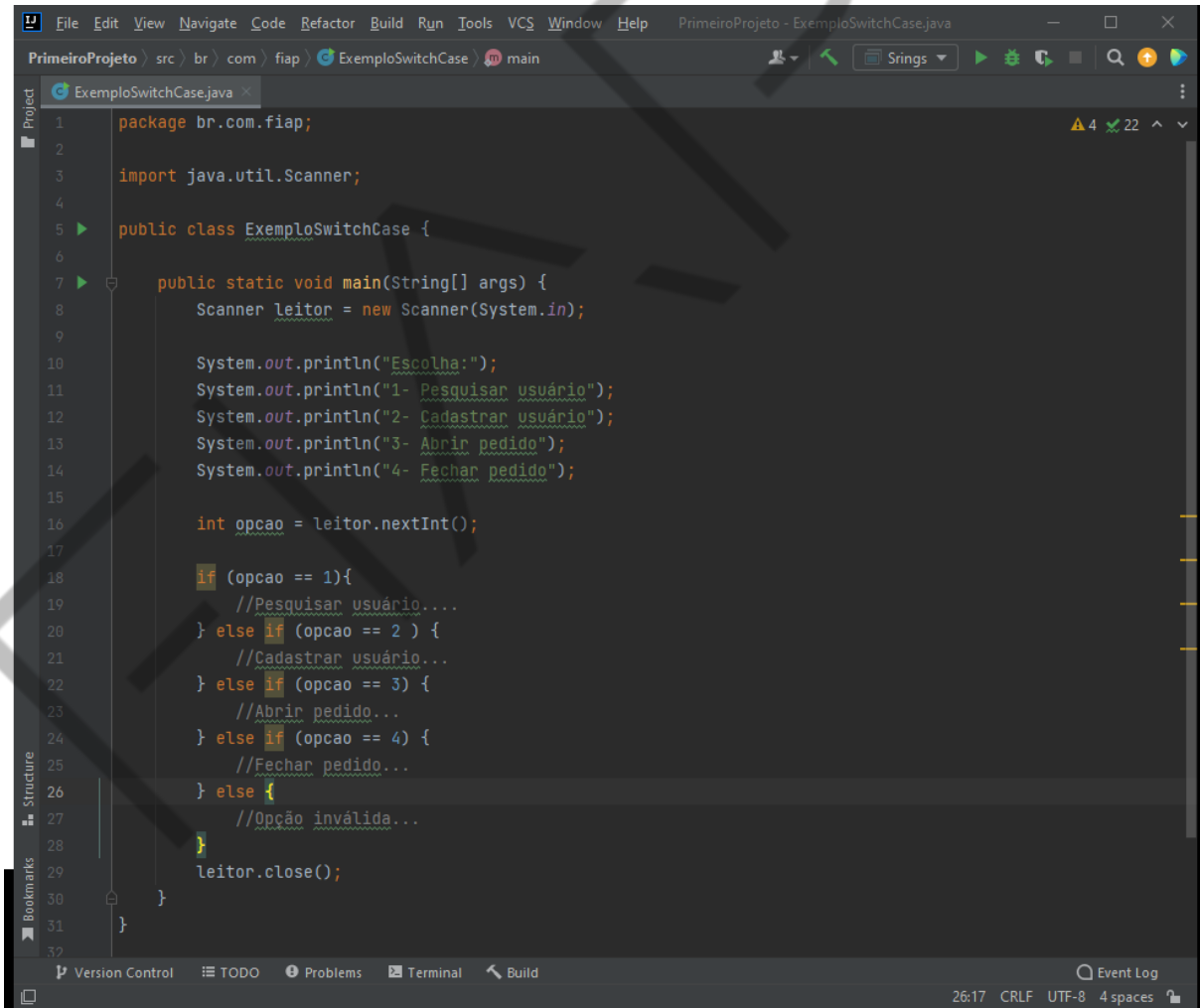
A última estrutura condicional que veremos será o switch case. Imagine uma aplicação com interface texto que apresente um menu com várias opções para o usuário escolher, como, por exemplo:

## Adicionando lógica ao programa Java

Escolha:

- 1- Pesquisar usuário;
- 2- Cadastrar usuário;
- 3- Abrir pedido;
- 4- Fechar pedido.

Você pode pensar: consigo resolver esse problema com a estrutura if-else. E você está certo! Sim, é possível implementar a lógica com várias instruções if-else (Figura “Exemplo de menu com if-else”).



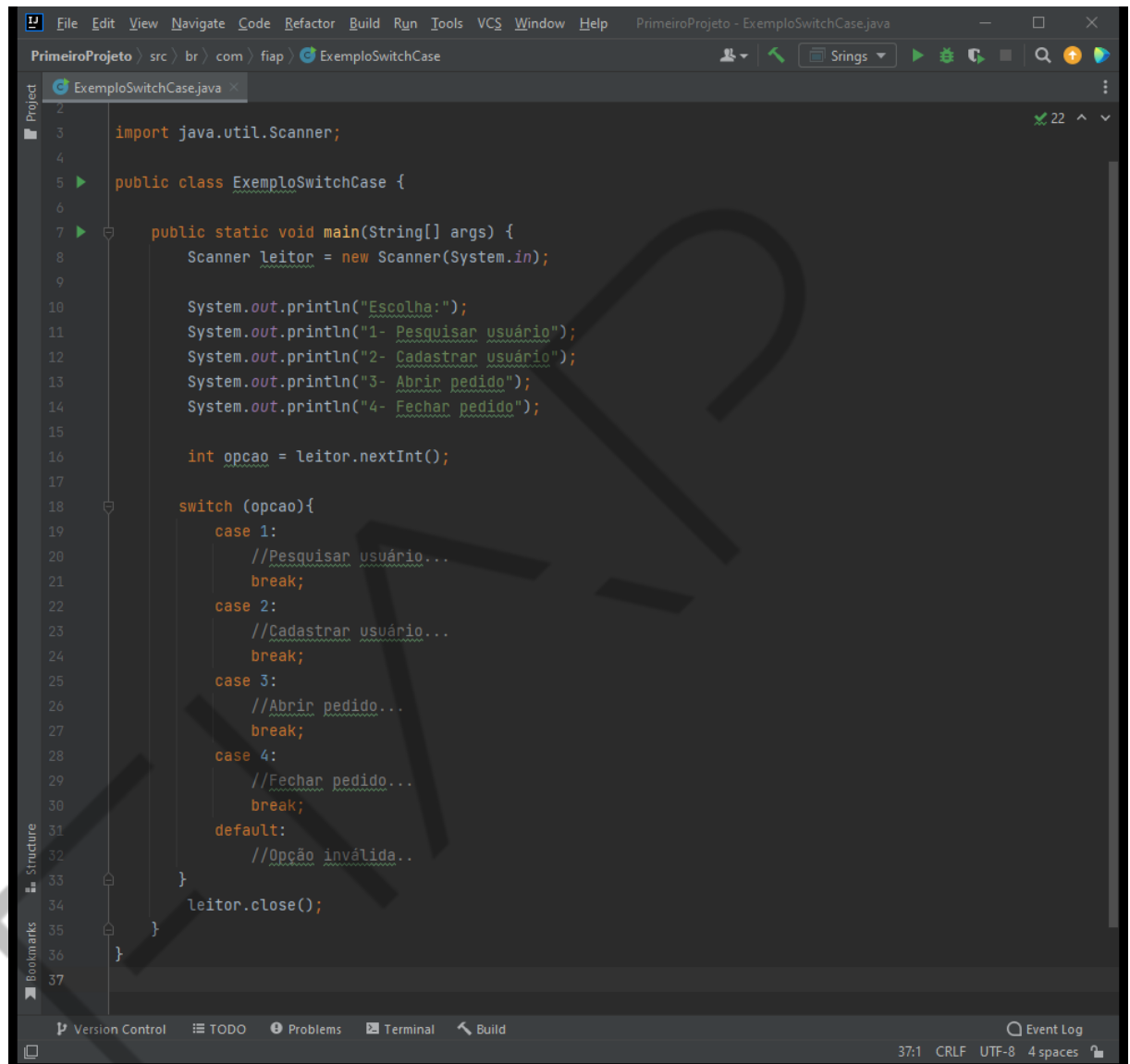
```
1 package br.com.fiap;
2
3 import java.util.Scanner;
4
5 public class ExemploSwitchCase {
6
7     public static void main(String[] args) {
8         Scanner leitor = new Scanner(System.in);
9
10        System.out.println("Escolha:");
11        System.out.println("1- Pesquisar usuário");
12        System.out.println("2- Cadastrar usuário");
13        System.out.println("3- Abrir pedido");
14        System.out.println("4- Fechar pedido");
15
16        int opcao = leitor.nextInt();
17
18        if (opcao == 1){
19            //Pesquisar usuário...
20        } else if (opcao == 2) {
21            //Cadastrar usuário...
22        } else if (opcao == 3) {
23            //Abrir pedido...
24        } else if (opcao == 4) {
25            //Fechar pedido...
26        } else {
27            //Opção inválida...
28        }
29        leitor.close();
30    }
31 }
```

Figura 12 – Exemplo de menu com if-else  
Fonte: Elaborado pelo autor (2022)

Dessa forma, podemos implementar várias estruturas de seleção com if-else para testar o valor da variável **opcao** para realizar a operação correta.

## Adicionando lógica ao programa Java

Porém, existe uma estrutura na linguagem Java que permite testar o valor de uma variável e compará-lo com os valores predeterminados. Analise o código apresentado (Figura “Exemplo de menu com switch case”).

The image is a screenshot of an IDE window titled 'PrimeiroProjeto - ExemploSwitchCase.java'. The code is as follows:

```
1 import java.util.Scanner;
2
3 public class ExemploSwitchCase {
4
5     public static void main(String[] args) {
6         Scanner leitor = new Scanner(System.in);
7
8         System.out.println("Escolha:");
9         System.out.println("1- Pesquisar usuário");
10        System.out.println("2- Cadastrar usuário");
11        System.out.println("3- Abrir pedido");
12        System.out.println("4- Fechar pedido");
13
14        int opcao = leitor.nextInt();
15
16        switch (opcao){
17            case 1:
18                //Pesquisar usuário...
19                break;
20            case 2:
21                //Cadastrar usuário...
22                break;
23            case 3:
24                //Abrir pedido...
25                break;
26            case 4:
27                //Fechar pedido...
28                break;
29            default:
30                //Opção inválida..
31        }
32        leitor.close();
33    }
34 }
```

The IDE interface includes a menu bar (File, Edit, View, etc.), a toolbar with icons for running and debugging, and a status bar at the bottom showing '37:1 CRLF UTF-8 4 spaces'.

Figura 13 – Exemplo de menu com switch case  
Fonte: Elaborado pelo autor (2022)

O condicional **switch** testa o valor contido em uma variável. Neste exemplo, está testado o valor contido na variável **opcao**. Cada **case** representa o valor que está sendo testado. É possível adicionar quantos **cases** forem necessários, e quando um dos valores corresponder ao da variável testada, o código do case será executado.

A estrutura switch compara o valor de cada caso com o valor da variável de forma sequencial. Assim, quando encontrar um valor correspondente, executa o

código associado ao caso. Porém, é preciso adicionar o comando **break;**, no final de cada caso, para evitar que os casos abaixo não sejam executados. O comando **break;** é utilizado para encerrar a execução da estrutura onde ele se encontra, neste caso, a estrutura do switch.

É possível adicionar um valor-padrão, o **default**, caso todos os **cases** sejam testados e nenhum deles corresponda ao valor da variável de teste. Como o comando **default** sempre será a última parte da estrutura, não é necessário adicionar o comando **break**.



## REFERÊNCIAS

BARNES, David J. **Programação Orientada a Objetos com Java: uma introdução prática** utilizando Blue J. São Paulo: Pearson, 2004.

COLHO, Alex. **Java com Orientação a Objetos**. Rio de Janeiro: Ciência Moderna, 2012.

DEITEL Paul; DEITEL Harvey. **Java – como programar**. 10. ed. Pearson, 2016.

HORSTMANN, Cay; CORNELL, Gary. **Core Java – Volume I – Fundamentos**. 8. ed. São Paulo: Pearson 2009.

SIERRA, Kathy; BATES, Bert. **Use a cabeça! Java**. Rio de Janeiro: Alta Books, 2010.