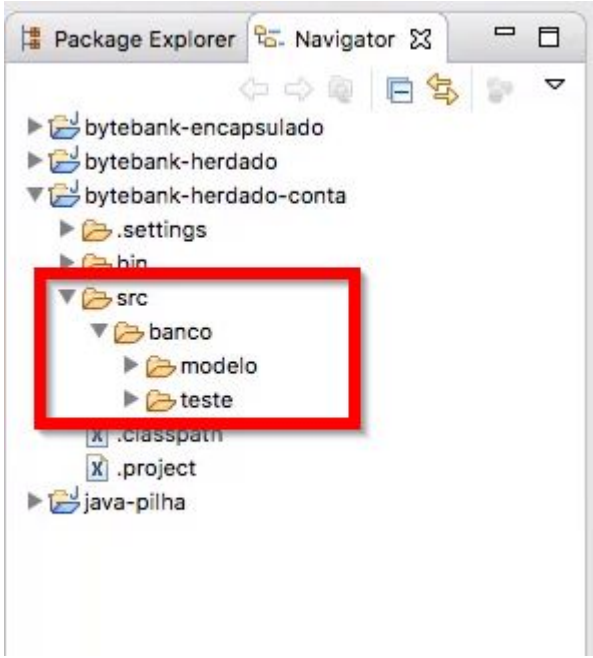
Introdução ao Java

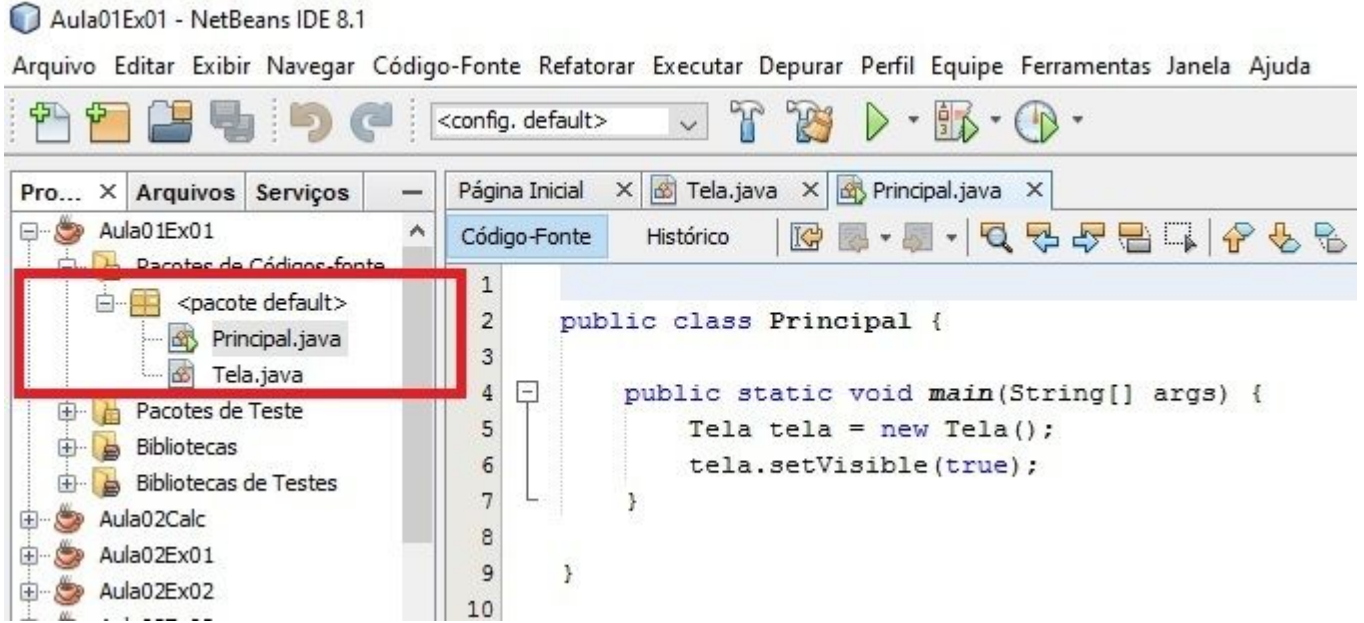
Organização em pacotes

Arquivos Java são armazenados fisicamente em uma hierarquia de pastas, chamadas pacotes (packages):

```
package nomedopacote;
```



Caso tal declaração não esteja presente, as classes farão parte do “pacote default”, que está mapeado para o diretório corrente. O uso do pacote default não é uma boa prática.



Importação de Classes e Interfaces

É possível importar classes e interfaces de outros pacotes incluindo o comando import antes da declaração de sua classe ou interface:

```
import nomedopacote.NomeDaClasse
```

ou

```
import nomedopacote.*
```

onde * indica que todas as classes do pacote serão importadas.

Tipos de Dados

Tipos primitivos:

- Tipos numéricos inteiros
- Tipos numéricos para ponto flutuante
- Tipo caractere
- Tipo lógico
- Tipos enumerados

Tipos Primitivos Inteiros

Permitem a representação de valores inteiros:

Tipo (palavra reservada)	Faixa de valores
byte	-128 a +127
short	-32.768 a +32.767
int	-2.147.483.648 a +2.147.483.647
long	-9.223.372.036.854.775.808 a +9.223.372.036.854.775.807

Tipos Primitivos para Ponto Flutuante

Permitem a representação de um subconjunto de valores reais, além de valores especiais:

Tipo (palavra reservada)	Faixa de valores
float	±1,40129846432481707 × 10^(-45) a ±3,40282346638528860 × 10^(+38)
double	±4,94065645841246544 × 10^(-324) a ±1,79769313486231570 × 10^(+308)

Tipo Primitivo Caractere

Permite a representação de caracteres individuais, delimitado por aspas simples.
Exemplo: ‘A’

Tipo (palavra reservada)	Valores Assumidos
char	Qualquer um dos 32.768 caracteres da tabela UNICODE

Tipo Primitivo Lógico

Permite a representação de valores lógicos, utilizados em operações lógicas

Tipo (palavra reservada)	**Valores Assumidos**
boolean	false *ou* true

Tipos Enumerados

Permite a restrição dos valores de uma variável a um conjunto de valores constantes pré-definidos. Enumeradores não são tipos primitivo, estendem implicitamente a classe java.lang.Enum. Declarados com auxílio da palavra reservada enum. Podem ser declarados separadamente ou no escopo de uma classe, mas nunca dentro de um método.

Regra de Declaração

Sintaxe: modificador enum NomeDoTipo {VALOR1, VALORN};

Exemplo: public enum Direcao {NORTE, SUL, LESTE, OESTE};

Atributo, Variável e Argumento

```
class NomeDaClasse {

      int atributo1; // Declaração de atributos acontece no escopo da classe

      int metodo(int argumento1, int argumento2 ){ // Argumentos fazem parte da assinatura do
método

            int variavel1; // Declaração de variáveis acontece em escopo interno ao método

            int variavel2;

      }

}
```

Operadores

- Operadores aritméticos
- Operadores relacionais
- Operadores lógicos
- Operadores de manipulação binária
- Operador de concatenação
- Operadores de atribuição

Operadores Aritméticos

Símbolo	Operação	Exemplo
+	Adição	a + b
-	Subtração	a - b
*	Multiplicação	a * b
/	Divisão	a / b
%	Resto da divisão inteira	a % b
++	Incremento	++a ou a++
- -	Decremento	- -a ou a- -

Operadores Relacionais

Símbolo	Operação	Exemplo
==	Igual a	a == b
!=	Diferente de	a != b
>	Maior que	a > b
>=	Maior OU igual a	a >= b
<	Menor que	a < b
<=	Menor OU igual a	a <= b

Os operadores operam sobre par de valores de mesma natureza e retornam valor lógico.
Deve-se tomar cuidado com comparações entre objetos, pois os operadores relacionais comparam o endereço na memória e não o conteúdo dos objetos.

Operadores Lógicos

Símbolo	Operação	Exemplo
&&	E lógico (and)	a && b
||	OU lógico (or)	a || b
!	Negação (not)	!a

Operador de Concatenação

Em Java, texto estático é representado pela classe String, presente no pacote java.lang. Existe um operador próprio para auxiliar a concatenação de texto.

Símbolo	**Operação**	**Exemplo**
+	Concatenação de String	“Texto” + a a + “Texto”

Este operador opera sobre uma instância de String e outro tipo qualquer, retornando outra instância de String

Operadores de Atribuição

Símbolo	Operação	Exemplo
=	Atribuição Simples	a = b
+=	Atribuição composta, um atalho para operação a = a + b	a += b
-=	Atribuição composta, um atalho para operação a = a - b	a -= b
*=	Atribuição composta, um atalho para operação a = a * b	a *= b
/=	Atribuição composta, um atalho para operação a = a / b	a /= b

Estruturas de Controle de Fluxo

Desvios condicionais

- if … else if … else
- switch … case … default

Repetições

- while
- do … while
- for …

Desvio Condicional

```
if ( expressão booleana ){
    bloco de comandos
} else if ( expressão booleana ){
    bloco de comandos
}else{
    bloco de comandos
}
```

- Desvio em caso de condição verdadeira:

```
if (expressão lógica){
    bloco de comandos
}
```

- Desvio caso a condição anterior seja falsa:

```
else if (expressão lógica){
    bloco de comandos
}
```

- Desvio caso todas as condições anteriores sejam falsas:

```
    else{
        bloco de comandos
    }
```

Desvio em caso de condição verdadeira

```
public class ExemploIf {

    public static void main(String[] args) {

        int a = 10;

        int b = 20;

        if (a < b) {

            System.out.println(“O menor valor eh: ” + a);

        }

    }

}
```

Desvio caso de condição anterior seja falsa

```
public class ExemploElseIf {
     public static void main(String[] args) {
          int a = 20;
          int b = 10;
          if (a < b) {
               System.out.println("O menor valor eh: " + a);
          }else if (b < a){
               System.out.println("O menor valor eh: " + b);
          }
     }
}
```

Desvio caso todas as condições anteriores sejam falsas

```
public class ExemploElse {
     public static void main(String[] args) {
          int a = 20;
          int b = 20;
          if (a < b) {
               System.out.println(“O menor valor eh: ” + a);
          }else if (b < a){
               System.out.println(“O menor valor eh: ” + b);
          }else{
               System.out.println(“Os valores são iguais!”);
     }
}
```

Desvio Condicional

Desvio condicionado ao caso:

```
switch (expressão) {
     case valor1: bloco de comandos
          [break;]
     case valor2: bloco de comandos
          [break;]
     case valorN: bloco de comandos
          [break;]
     default: bloco de comandos
}
```

Exemplo:

```
public class ExemploSwitch {
     public static void main(String[] args) {
          switch (args.length) {
               case 0: System.out.println(“Nenhum.”);
                    break;
               case 1: System.out.println(“Um.”);
                    break;
               case 2: System.out.println(“Dois.”);
                    break;
               default: System.out.println(“Mais que dois.”);
          }
     }
}
```

Repetição Condicional

Teste de condição antes da execução do bloco de comandos:

```
while (expressão lógica) {
    bloco de comandos
}
```

Teste de condição após a execução do bloco de comandos:

```
do {
    bloco de comandos
} while (expressão lógica);
```

Exemplo:

```
import java.util.Scanner;

public class ExemploWhile {

      public static void main(String[] args) {

            Scanner teclado = new Scanner(System.in);

            String linha = teclado.nextLine();

            while (!linha.equals("fim")) {

                  System.out.println("Linha = " + linha);

                  linha = teclado.nextLine();

            }

      }

}
```

Obs: a palavra reservada “equals” é utilizada para comparar o conteúdo de 2 Strings. Caso o teste fosse executado utilizando o operador lógico “==”, o bloco de comandos do while não seria executado, pois nesse caso estaríamos testando se o endereço de memória são iguais e não o conteúdo.

Exemplo:

```
import java.util.Scanner;

public class ExemploDoWhile {

      public static void main(String[] args) {

            Scanner teclado = new Scanner(System.in);

            String linha;

            do {

                  linha = teclado.nextLine();

                  System.out.println("Linha = " + linha);

            } while (!linha.equals("fim"));

      }

}
```

Repetição Contável

Uma maneira enxuta de fazer repetições que dependem de valores sequenciais

```
for (inicialização; expressão lógica; (in/de)cremento) {
    bloco de comandos
}
```

```
for (Objeto c : Objeto){
    bloco de comandos
}
```

Exemplo:

```
public class ExemploFor {

    public static void main(String[] args) {

        char[] c = {‘F’,’A’,’B’,’I’,’O’}

        for (int i = 0; i < c.length; i++) {

            System.out.println(c[i]);

        }

        System.out.println(“Fim do programa.”);

    }

}
```

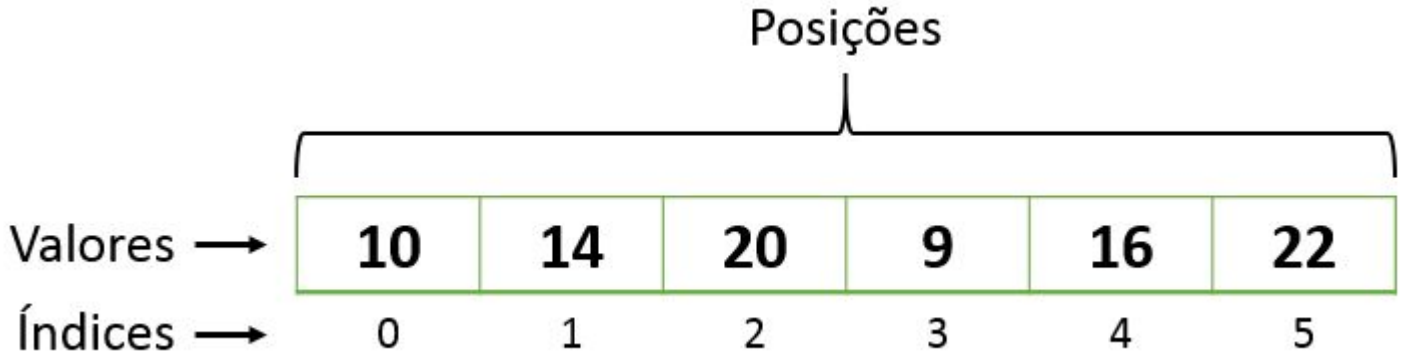
Exemplo:

```
public class ExemploForEach {

     public static void main(String[] args) {

          char[] c = {‘F’,’A’,’B’,’I’,’O’}

          for (char a : c) {

               System.out.println(a);

          }

          System.out.println(“Fim do programa.”);

     }

}
```

Arrays Unidimensionais de Tipos Primitivos

Permitem armazenamento e manipulação de uma mesma quantidade de dados de mesmo tipo.

Mapeia-se um conjunto finito de índices inteiros (de tamanho constante) em um conjunto qualquer de elementos de mesmo tipo.



Arrays Unidimensionais de Tipos Primitivos

Declaração e inicialização de variáveis:

```
tipo[] arrayVazio;

tipo[] array = new tipo[quantidade];

tipo[] array1 = {valor1, valor2, ...};
```

Consulta de tamanho:

- Atributo length

Exemplo:

```
double[] notas = {8.0, 7.0, 9.0};

int qtd = notas.length;
```

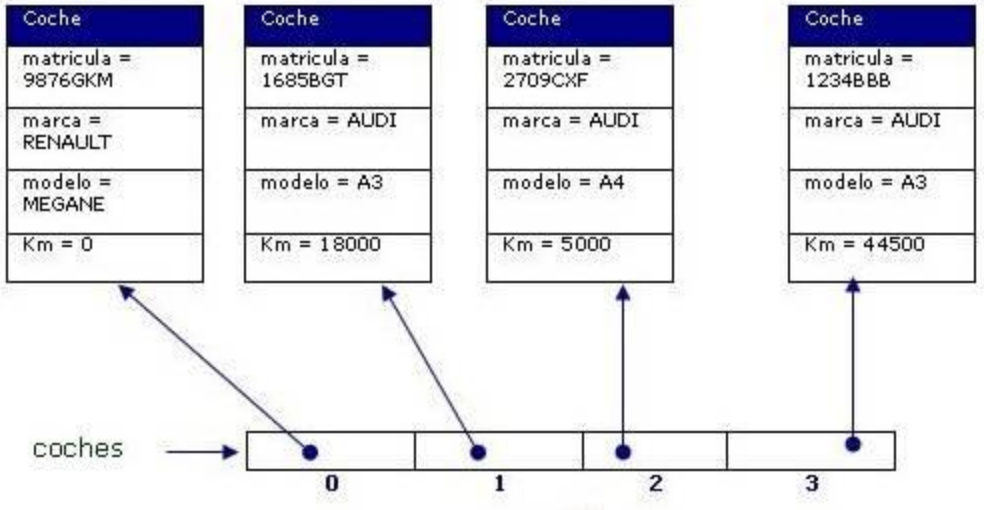
Arrays Unidimensionais de Tipos Primitivos

Percorrer e acessar elementos de um array de double's:

```
public class ExemploArrayPrimitivos {

     public static void main(String[] args) {

          double[] notas = {8.0, 7.0, 9.0};

          for (int i = 0; i < notas.length; i++) {

               System.out.println(notas[i]);

          }

          System.out.println("Fim do programa.");

     }

}
```

Arrays Unidimensionais de Objetos

Utiliza-se o mesmo princípio do array de tipos primitivos, mas o interior do array ‘aponta’ para objetos. Permitem armazenamento e manipulação de uma mesma quantidade de dados de mesmo tipo. Mapeia-se um conjunto finito de índices inteiros (de tamanho constante) em um conjunto qualquer de elementos do mesmo tipo.



Arrays Unidimensionais de Objetos

Declaração e inicialização de variáveis:

```
tipo[] arrayVazio;

tipo[] array = new tipo[quantidade];

tipo[] array1 = {valor1, valor2, ...};
```

Consulta de tamanho:

- Atributo length

Exemplo:

```
Pessoa p1 = new Pessoa(“Fabio”);

Pessoa p2 = new Pessoa(“Maria”);

Pessoa[] pessoas = {p1, p2};

int qtd = pessoas.length;
```

Arrays Unidimensionais de Objetos

Percorrer e acessar elementos de um array de pessoas:

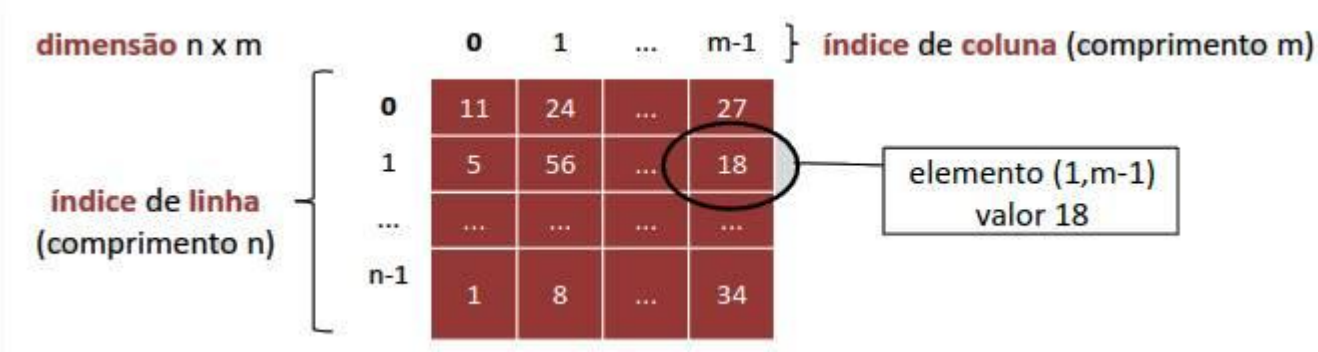
```
public class ExemploArrayObjetos {

      public static void main(String[] args) {

            Pessoa p1 = new Pessoa(“Fabio”);

            Pessoa p2 = new Pessoa(“Maria”);

            Pessoa[] pessoas = {p1, p2};

            for (int i = 0; i < pessoas.length; i++) {

                  System.out.println(pessoas[i].getNome());

            }

            System.out.println(“Fim do programa.”);

      }

}
```

Arrays Bidimensionais (Matrizes)

Uma matriz é um Array de Arrays.

Declaração e inicialização de variáveis:

```
tipo[][] matrizVazia;

tipo[][] matriz = new tipo[quantidadeLinhas][quantidadeColunas];

tipo[] matriz1 = {{valor1, valor2},{valor3,valor4}, ...};
```



Arrays Bidimensionais (Matrizes) de primitivos

Consulta de tamanho:

- Atributo length

Exemplo:

```
 int qtdLinhas = matriz.length;

int qtdColunas = matriz[posLinha].length;
```

Arrays Bidimensionais de Tipos Primitivos

Percorrer e acessar elementos de uma matriz de double's:

```
public class ExemploMatrizPrimitivos {

	public static void main(String[] args) {

		double[][] notas = {{8.0, 7.0, 9.0},{5.0, 6.0, 7.0}};

		for (int i = 0; i < notas.length; i++) {

			for(int j = 0; j < notas[i].length; j++){

				System.out.println(notas[i][j]);

			}

			System.out.println("Fim da nota de indice: " + i);

		}

		System.out.println("Fim do programa.");

	}

}
```

Comentários em Java

Comentar o código é uma boa prática de desenvolvimento. Há três tipos de comentários em Java:

```
// Comentário de uma linha.

/* Comentário de
 * múltiplas linhas.
*/

/** Comentário de documentação. Também pode
  * possuir múltiplas linhas e é utilizado
  * para gerar arquivos hipertexto de ajuda
*/
```

Exemplo:

```
/** Classe que representa a entidade cliente do sistema.
      * <p> É utiliza sempre no contexto de venda de
      * mercadorias para pessoa física.
      *
      * @author O estagiário
      * @see meusistema.Venda
*/
public class Cliente extends PessoaFisica {
      /** Construtor padrão.
      */
      public Cliente() {
            super(); // Invocar construtor padrão da superclasse.
      }
}
```