**Java: trabalhando com listas e coleções de dados**

**Declarando variáveis com var**

A partir da versão 10 do Java, foi adicionada uma nova funcionalidade para a declaração de variáveis chamada **var**. Essa nova palavra-chave permite que o compilador infira automaticamente o tipo da variável com base no valor atribuído a ela. Isso pode tornar o código mais limpo e legível, além de reduzir a digitação de código redundante.

**Sintaxe básica**

A sintaxe básica para declarar uma variável com var é a seguinte:

**var** nomeDaVariavel = valorInicial;

Onde **nomeDaVariavel** é o nome que você quer dar à variável e **valorInicial** é o valor que você quer atribuir a ela. O tipo da variável será inferido automaticamente pelo compilador com base no valor atribuído. Exemplo:

**var** numero = 10;

Nesse exemplo, a variável **numero** será inferida como sendo do tipo **int**, já que o valor atribuído é um número inteiro.

**Limitações**

A declaração de variáveis com var possui algumas limitações:

1. O tipo da variável deve ser inferido automaticamente pelo compilador. Isso significa que não é possível utilizar **var** em variáveis cujo tipo não possa ser inferido automaticamente.
2. Não é possível usar **var** em variáveis sem valor inicial. É necessário atribuir um valor à variável na mesma linha em que ela é declarada.

A declaração de variáveis com **var** é uma funcionalidade relativamente nova no Java, mas que pode trazer diversos benefícios para o código, como a redução de digitação e melhor legibilidade do código. No entanto, é importante lembrar que existem limitações e que nem sempre é possível utilizar esse recurso.

**Arrays no Java**

Em Java, **arrays** são estruturas de dados que permitem armazenar uma coleção de elementos do mesmo tipo. Eles são muito utilizados para manipulação de dados em projetos de programação.

Para declarar um **array** em Java, é preciso definir seu tipo e tamanho. Por exemplo, para criar um **array** de inteiros com tamanho 5, podemos escrever o seguinte código:

**int**[] numeros = **new** **int**[5];

Aqui, estamos declarando um **array** chamado "**numeros**" do tipo "**int**" e com tamanho 5. É importante lembrar que o índice dos elementos de um array começa em 0 e vai até o tamanho do ***array menos 1***.

Após declarar um **array**, podemos inicializá-lo com valores. Por exemplo, podemos preencher o **array** "**numeros**" com os números de 1 a 5 da seguinte forma:

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

numeros[i] = i + 1;

}

Aqui, estamos percorrendo o **array** "**numeros**" utilizando um loop for e preenchendo cada posição com seu respectivo índice mais 1.

Também é possível criar **arrays** de objetos e não apenas de tipos primitivos. Por exemplo:

Filme[] filmes = **new** **Filme**[2];

**Filme** filme1 = **new** **Filme**("Avatar", 2009);

**Filme** filme2 = **new** **Filme**("Dogville", 2003);

filmes[0] = filme1;

filmes[1] = filme2;

Embora os **arrays** sejam úteis, eles possuem algumas limitações que podem causar problemas em projetos. Alguns desses problemas incluem:

1. **Tamanho fixo**: o tamanho de um **array** é fixo e não pode ser alterado após a sua criação. Isso pode ser problemático em situações em que o tamanho dos dados a serem armazenados é desconhecido ou variável.
2. **Ausência de métodos**: **arrays** não possuem métodos que permitam a inserção, remoção ou pesquisa de elementos de forma eficiente. Isso pode levar a soluções de código complicadas e ineficientes para tarefas simples.

Justamente por conta desses problemas e dificuldades é que não devemos utilizar **arrays** para representar uma coleção de elementos, mas sim alguma classe do Java, como a **ArrayList**, que encapsula e abstrai um **array**, facilitando a sua utilização via métodos e deixando o código do projeto mais simples de entender e evoluir.

**JavaDoc do ArrayList**

Para conhecer mais detalhes da classe, bem como conhecer todos os métodos que ela possui, você pode consultar o [JavaDoc de ArrayList](https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/ArrayList.html).

**Carrinho de compras**

Sua amiga Iasmin trabalha em uma aplicação Java de uma loja virtual e precisa criar uma classe que representa um carrinho de compras. Ela pensou em fazer da seguinte maneira:

**public** **class** **Carrinho** {

ArrayList<Produto> produtos = **new** ArrayList<>();

**public** **void** **adicionaProduto**(Produto produto) {

produtos.**add**(produto);

}

}

Porém, a loja tem uma restrição de permitir uma quantidade máxima de apenas 10 produtos no carrinho de compras. Como fazer a mudança correta no código anterior para atender a essa restrição?

* É preciso alterar o método adicionaProduto para:

**public** **void** **adicionaProduto**(Produto produto) {

**if** (produtos.size() < 10) {

produtos.**add**(produto);

} **else** {

System.**out**.prinln("Carrinho cheio!");

}

}

O método **size**, da classe **ArrayList**, serve justamente para indicar a quantidade atual de elementos da lista.

**ArrayList e toString**

Faça o mesmo procedimento feito na aula, sobrescrevendo o método toString() na classe Filme, e alterando a classe Principal para criar uma lista de filmes.

@Override

**public** String toString() {

**return** "Filme: " +**this**.getNome() + " (" +**this**.getAnoDeLancamento() + ")";

}

E no método main da classe Principal, crie um ArrayList, adicione os filmes e imprima a lista:

**public** **class** **Principal** {

**public** **static** **void** **main**(String[] args) {

//codigo anterior omitido

ArrayList<Filme> listaDeFilmes = **new** ArrayList<>();

listaDeFilmes.**add**(favorito);

listaDeFilmes.**add**(outro);

System.**out**.println(“Tamanho da lista: “ +listaDeFilmes.size());

System.**out**.println(“Primeiro Filme: “ +listaDeFilmes.**get**(0));

System.**out**.println(listaDeFilmes);

}

}

**Construtor padrão**

Em Java, um construtor é um método especial usado para criar e inicializar um objeto recém-criado. Quando uma classe é definida, ela pode ter um ou mais construtores, sendo que se nenhum construtor for definido explicitamente, o Java criará um construtor default (padrão) automaticamente.

Um construtor default é um construtor que não possui parâmetros e não executa nenhuma instrução. Ele é chamado sempre que um objeto da classe é criado sem argumentos. Por exemplo:

**public** **class** **Pessoa** {

**private** String nome;

**private** String email;

**public** **Pessoa**() {

}

//metodos getters/setters

}

No exemplo de código anterior, a classe **Pessoa** possui um construtor default, que será exatamente o mesmo construtor que o Java criará automaticamente, caso nenhum construtor tivesse sido definido na classe.

Se uma classe define explicitamente um ou mais construtores, mas não define um construtor sem parâmetros, então não há construtor default. Nesse caso, se um objeto é criado sem argumentos, um erro de compilação será gerado.

É importante ressaltar que mesmo que um construtor default possa ser útil em alguns casos, é sempre recomendável definir explicitamente os construtores da classe, especialmente se a classe tiver atributos que precisam ser inicializados com valores específicos ou obrigatórios. Isso também torna o código mais claro e fácil de entender.

**Problemas com construtores**

Você está analisando o código de uma aplicação Java e encontra a seguinte classe:

**public** **class** **Produto** {

**private** String nome;

**private** double preco;

**public** **Produto**() {

}

**public** **Produto**(String nome, double preco) {

nome = nome;

preco = preco;

}

**public** **void** **setNome**(String nome) {

**this**.nome = nome;

}

**public** **void** **setPreco**(double preco) {

**this**.preco = preco;

}

**public** String **getNome**() {

**return** nome;

}

**public** double **getPreco**() {

**return** preco;

}

}

Em relação ao uso de construtores, qual o problema no código acima?

Como os atributos nome e preço são do tipo private, o segundo construtor precisa trocar `nome = nome;`por `this.setNome(nome);` e `preco = preco;` por `this.setPreco(preco);`