Java Developer

Java Core

Sumário

Objetivos deste Módulo

Ao final deste módulo, objetiva-se que o aluno adquira:

Conhecimentos:

- Tipos primitivos
- Constantes
- Code conventions
- Criação de métodos com e sem retorno
- Criar projeto, criar pacote/package, criar classe, criar método void, criar método com retorno, invocar ambos os métodos

1. Tipos primitivos

Em Java, existem oito tipos primitivos, também conhecidos como tipos básicos. Eles representam valores simples e são armazenados diretamente na memória, diferentemente dos objetos que são armazenados em referências. Aqui estão os oito tipos primitivos em Java:

- byte: Representa valores numéricos **inteiros** de 8 bits. O intervalo de valores vai de -128 a 127.
- short: Representa valores numéricos **inteiros** de 16 bits. O intervalo de valores vai de -32768 a 32767.
- int: Representa valores numéricos **inteiros** de 32 bits. É um dos tipos primitivos mais comuns e o intervalo de valores vai de -2^31 a 2^31 - 1.
- long: Representa valores numéricos **inteiros** de 64 bits. Pode conter números maiores que o tipo int. O intervalo de valores vai de -2^63 a 2^63 - 1.

Tipo	Tamanho	Valor
byte	8 bits	-128 a 127
short	16 bits	-32.768 a 32.767
int	32 bits	-2.147.483.648 a 2.147.483.647
long	64 bits	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Fonte: https://www.if.ufrgs.br/~betz/jaulas/aula2o.htm

- float: Representa valores de **ponto flutuante** de 32 bits. É usado para armazenar números reais com aproximação. No entanto, devido à natureza de ponto flutuante, ele pode não ser totalmente preciso em todas as situações.
- double: Representa valores de **ponto flutuante** de 64 bits. Assim como o float, é usado para armazenar números reais, mas oferece uma precisão maior devido ao seu tamanho maior.

Tipo	Tamanho	Valor
float	32 bits	-3.40292347E+38 a +3.40292347E+38
double	64 bits	-1.79769313486231570E+308 a +1.79769313486231570E+308

Fonte: https://www.if.ufrgs.br/~betz/jaulas/aula2o.htm

- char: Representa um **único caractere** Unicode de 16 bits. Pode ser usado para armazenar letras, números, símbolos e outros caracteres.
- boolean: Representa um **valor verdadeiro** (true) ou **falso** (false). É usado para lógica e controle de fluxo.

Esses tipos primitivos são a base para a manipulação de dados em Java e são usados para definir variáveis que armazenam valores simples. Além desses tipos, Java também permite a criação de tipos complexos através de classes e objetos, mas os tipos primitivos são fundamentais para a linguagem.

Para **cadeias de caracteres**, utilizamos o tipo **String**. String não é um tipo primitivo.

Os tipos primitivos em Java são eficientes em termos de espaço de memória e tempo de execução, porque eles armazenam diretamente os valores sem a sobrecarga associada aos objetos. Isso os torna ideais para uso em situações onde a performance é crucial, como em sistemas embarcados, cálculos científicos e outras aplicações sensíveis ao desempenho.

Além disso, os tipos primitivos têm algumas características importantes a serem consideradas:

- Não aceitam valores nulos: Ao contrário de objetos, os tipos primitivos não podem ser nulos. Isso significa que eles sempre têm um valor válido, mesmo que seja um valor padrão (por exemplo, 0 para numéricos e false para booleanos).

- Cópia por valor: Quando você atribui um valor de um tipo primitivo a outra variável, uma cópia direta do valor é feita. Isso significa que as alterações em uma variável não afetarão a outra.

- Não possuem métodos: Os tipos primitivos não têm métodos associados a eles, ao contrário dos objetos. Isso ocorre porque eles não são instâncias de classes.

- Operações aritméticas e lógicas: Os tipos primitivos suportam operações aritméticas (adição, subtração, multiplicação etc.) e operações lógicas (AND, OR, NOT) diretamente.

Lembre-se de que, apesar das vantagens dos tipos primitivos, existem situações em que os objetos são mais apropriados, como quando você precisa de funcionalidades mais complexas, como coleções dinâmicas ou métodos específicos. Nesses casos, você pode usar as classes empacotadoras (wrappers) correspondentes para criar objetos que representam os tipos primitivos.

Em resumo, os tipos primitivos em Java são os blocos de construção fundamentais para o armazenamento e manipulação eficiente de valores simples. Combinados com classes e objetos, eles formam a base da linguagem Java.

2. Constantes

Uma constante é um valor que não pode ser alterado após ser definido. Elas são frequentemente usadas para representar valores fixos, como números, Strings ou outros tipos de dados, que são utilizados repetidamente em um programa e não devem ser modificados. A linguagem Java oferece uma maneira de definir constantes usando a palavra-chave **final**.

```java
final int NUMERO_CONSTANTE = 10;
final String MENSAGEM = "Olá, mundo!";
```

Você também pode criar constantes usando a palavra-chave **static final**. Essa combinação cria uma constante que pertence à classe em vez de uma instância específica da classe. Isso é útil quando você deseja compartilhar constantes entre todas as instâncias da classe.

```java
class Constantes {
    static final int VALOR = 100;
    static final String NOME = "Constante";
}
```

3. Comentários

```
// Isso é um comentário de uma linha

/*
* Isso é um comentário de múltiplas linhas
*
*/

/**
 * Isso é um Javadoc
 * @author marcia
 @deprecated
 * @version
 */
```

4. Code conventions

As Java Code Conventions (Convenções de Código Java) são diretrizes estabelecidas pela comunidade de desenvolvedores Java para padronizar a forma como o código Java é escrito.

Seguir essas convenções torna o código mais legível e compreensível para outros desenvolvedores, além de promover consistência em projetos e colaborações. Embora não seja obrigatório, seguir essas convenções é altamente recomendado para manter um padrão de qualidade no código. Aqui estão algumas das principais Java Code Conventions:

- Nomes de Variáveis e Métodos:
  - Nomes de variáveis e métodos devem ser descritivos, usando palavras completas em vez de abreviações obscuras.
  - Nomes de variáveis devem começar com letra minúscula e seguir o estilo CamelCase (exemplo: nomeCompleto).
  - Nomes de métodos também devem começar com letra minúscula e seguir o estilo CamelCase (exemplo: calcularSalario).
  - É uma boa prática usar o nome de métodos no infinitivo. (exemplos: salvar, imprimir, exibir)

- Nomes de Classes e Interfaces:
  - Nomes de classes devem começar com letra maiúscula e seguir o estilo CamelCase (exemplo: PessoaFisica).
  - Nomes de interfaces também devem começar com letra maiúscula e seguir o estilo CamelCase (exemplo: Impressora).

- Constantes:
  - Nomes de constantes devem ser escritos em letras maiúsculas, com palavras separadas por underscores (exemplo: VALOR_MAXIMO).

- Indentação e Espaçamento:
  - Use espaços para indentação (normalmente 4 espaços por nível).
  - Use linhas vazias para separar blocos de código e tornar o código mais legível.
  - Use espaços em torno de operadores (exemplo: a + b, não a+b).
  - Evite espaços em excesso.

- Comentários:
  - Comente o código para explicar trechos complexos, algoritmos ou decisões importantes.
  - Use Javadoc para documentar classes e métodos públicos.
  - Evite comentários desnecessários que não agregam valor.

- Organização do Código:
  - Organize o código em pacotes e diretórios que refletem a estrutura do projeto.
  - Mantenha um estilo consistente em todo o código, mesmo em projetos maiores.

- Uso de Chaves:
  - Use chaves em todos os blocos, mesmo que contenham apenas uma única instrução.
  - Abra chaves na mesma linha do comando que inicia o bloco.

- Imports:
  - Use imports separados para classes do Java padrão, bibliotecas externas e classes do seu projeto.

- Uso de Enums:
  - Defina constantes relacionadas em um enum em vez de usar constantes separadas.

- Tratamento de Exceções:
  - Use exceções para situações excepcionais, não para controle de fluxo normal.

Essas são apenas algumas das principais diretrizes das Java Code Conventions. É uma boa prática ler e seguir o guia oficial de estilo de código da Oracle, chamado de "Code Conventions for the Java Programming Language", para uma compreensão completa das convenções. Isso ajudará você a escrever código mais legível, compreensível e colaborativo.

Oracle:
https://www.oracle.com/java/technologies/javase/codeconventions-contents.html

Google:
https://google.github.io/styleguide/javaguide.html

CamelCase x Snake_case

CamelCase e Snake_case são convenções de nomenclatura para a escrita de identificadores, como nomes de variáveis, métodos, classes e outros elementos em programação.

CamelCase: é uma convenção de nomenclatura em que as palavras são agrupadas sem espaços, e cada palavra subsequente começa com uma letra maiúscula (exceto a primeira palavra). Existem diferentes variações do CamelCase, incluindo:

- camelCase (ou lowerCamelCase): A primeira letra é minúscula. Exemplo: nomeCompleto, calcularSalario.

- PascalCase (ou UpperCamelCase): A primeira letra é maiúscula. Exemplo: NomeCompleto, CalcularSalario.

Essas convenções são frequentemente usadas para nomear variáveis, métodos e classes em linguagens de programação como Java, C#, JavaScript, entre outras.

Snake_case: é uma convenção de nomenclatura em que as palavras são separadas por underscores (_) e todas as letras são minúsculas. É uma escolha comum em linguagens de programação como Python e Ruby. Exemplos: nome_completo, calcular_salario.

As convenções de nomenclatura têm o objetivo de tornar o código mais legível e compreensível, facilitando a identificação de palavras individuais em identificadores compostos. A escolha entre CamelCase e Snake_case geralmente depende da convenção predominante na linguagem que você está usando ou das diretrizes estabelecidas pelo projeto em que você está trabalhando. É importante manter a consistência no estilo de nomenclatura dentro de um projeto para garantir um código claro e de fácil manutenção.

5. Métodos com retorno

Um método com retorno em Java é um método que executa algum tipo de processamento e retorna um valor após a conclusão desse processamento. O tipo do valor retornado é especificado na declaração do método, e o valor é retornado usando a palavra-chave **return**. Aqui está a estrutura básica de um método com retorno em Java

```java
tipoDoRetorno nomeDoMetodo(parametros) {
    // Código do método

    return valor; // Valor do tipo especificado na declaração do método
}
```

Aqui está um exemplo de um método que calcula o quadrado de um número e retorna o resultado:

```java
public class MetodoComRetornoExemplo {
    public static void main(String[] args) {
        int numero = 5;
        int resultado = calcularQuadrado(numero);

        System.out.println("O quadrado de " + numero + " é: " + resultado);
    }

    // Método com retorno que calcula o quadrado de um número
    static int calcularQuadrado(int num) {
        int quadrado = num * num;
        return quadrado;
    }
}
```

Neste exemplo, o método **calcularQuadrado** recebe um número como parâmetro, calcula o quadrado desse número e retorna o resultado como um valor do tipo int. O método é chamado no método main, e o resultado é então impresso.

Lembre-se das seguintes características de métodos com retorno em Java:

- O tipo de retorno especificado na declaração do método deve corresponder ao tipo de valor que você está retornando.
- Um método com retorno deve usar a instrução return para devolver um valor do tipo especificado. O valor retornado deve ser compatível com o tipo de retorno declarado.

No exemplo acima, o método **calcularQuadrado** tem um tipo de retorno int, portanto, ele deve retornar um valor inteiro usando **return**.

6. Métodos sem retorno

Um método sem retorno em Java é um método que executa um determinado processamento, mas não retorna nenhum valor. Isso é indicado pelo uso da palavra-chave **void** na declaração do método, em vez de um tipo de retorno. Aqui está a estrutura básica de um método sem retorno em Java:

```java
void nomeDoMetodo(parametros) {
    // Código do método
}
```

Aqui está um exemplo de um método sem retorno em Java que imprime uma mensagem na tela:

```java
public class MetodoSemRetornoExemplo {
    public static void main(String[] args) {
        exibirMensagem();
    }

    // Método sem retorno que exibe uma mensagem
    static void exibirMensagem() {
        System.out.println("Olá, mundo!");
    }
}
```

Neste exemplo, o método **exibirMensagem** não tem um tipo de retorno especificado (usando void). Ele executa o código para imprimir a mensagem "Olá, mundo!" na tela, mas não retorna nenhum valor.

Lembre-se das seguintes características de métodos sem retorno em Java:

- Um método com a palavra-chave void na declaração não retorna nenhum valor.
- Métodos sem retorno são usados quando você quer executar um certo código ou realizar uma ação, mas não precisa receber um valor de volta.
- Eles podem aceitar parâmetros e realizar qualquer tipo de processamento, mas não têm uma cláusula return.

No exemplo acima, o método exibirMensagem não possui uma instrução return porque ele não retorna um valor. Em vez disso, ele apenas executa a ação de imprimir uma mensagem.

7. Exercícios

a) Crie um projeto/aplicação chamado calculadora.
b) Crie um pacote br.com.targettrust.calculadora
c) No pacote br.com.targettrust.calculadora crie uma classe Operacao
d) Dentro da classe Operacao, crie os métodos a seguir:
    i) somar: recebe dois valores inteiros e retorna o resultado da **soma** destes dois valores
    ii) subtrair: recebe dois valores inteiros e retorna o resultado da **subtração** destes dois valores
    iii) dividir: recebe dois valores inteiros e retorna o resultado da **divisão** destes dois valores
    iv) multiplicar: recebe dois valores inteiros e retorna o resultado da **multiplicação** destes dois valores
e) Criar a classe Tela no package br.com.targettrust.calculadora
f) Na classe Tela criar um método com nome "exibir" que recebe um valor numérico e exibe no seguinte formato: O resultado é xxxx
   Onde xxxx é o valor recebido como argumento

Obs.: Não esquecer de criar uma classe separada que contenha o método main para poder chamar os 5 métodos criados.