Atributos e Métodos Estáticos

Programação Orientada a Objetos — QXD0007

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

1º semestre/2022

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Leituras para esta aula

- **Seções 6.1 a 6.5** do Capítulo 6 (Métodos) do livro Java Como Programar, Décima Edição.

- **Capítulo 5** (Modificadores de acesso e atributos de classe) da apostila da Caelum – Curso FJ-11.

Atributos Estáticos

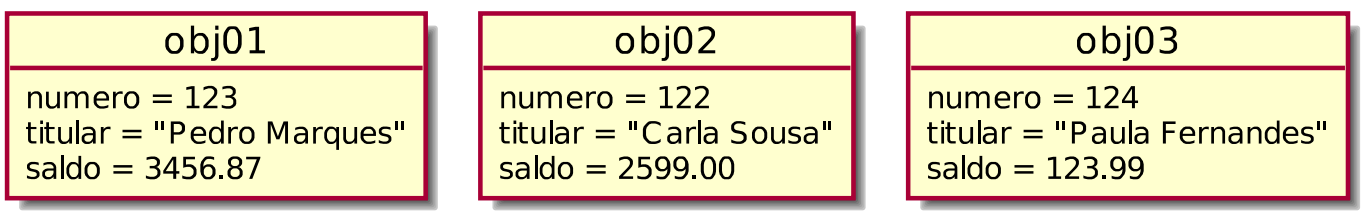
Introdução

- Podemos criar classes contendo atributos de diversos tipos:
  - atributos de tipo nativo
  - atributos de tipo referência

Introdução

- Podemos criar classes contendo atributos de diversos tipos:
  - atributos de tipo nativo
  - atributos de tipo referência

- Cada instância de uma dessas classes terá uma cópia de todos os atributos declarados na classe.
  - Esses atributos são chamados atributos de instância.
  - Modificar o valor de um atributo de instância de um objeto não interfere no valor do mesmo atributo em outro objeto da mesma classe.



Três objetos do tipo Conta

Classe Conta

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  /**
2   * A classe Conta modela uma conta bancária simplificada.
3   */
4  public class Conta {
5      private int numero;
6      private String titular;
7      private double saldo;

8      Conta(int numero, String titular, double saldo) {
9          this.numero = numero;
10         this.titular = titular;
11         this.saldo = saldo;
12     }

13     Conta(int numero, String titular) {
14         // invoca o construtor com três parâmetros
15         this(numero, titular, 0.0);
16     }
```

Classe Conta (cont.)


```
17      public int getNumero() { // getter
18          return numero;
19      }
20
21      public double getSaldo() { // getter
22          return saldo;
23      }
24
25      public String getTitular() { // getter
26          return titular;
27      }

28      public boolean saca(double valor) {
29          if(valor >= 0 && valor <= this.saldo) {
30              this.saldo -= valor;
31              return true;
32          }
33          else return false;
34      }
```

Classe Conta (cont.)


```
35      public boolean deposita(double valor) {
36          if(valor >= 0) {
37              this.saldo += valor;
38              return true;
39          }
40          else return false;
41      }

42      public boolean transfere(Conta3 destino, double valor) {
43          boolean retirou = this.saca(valor);
44          if(retirou == false) {
45              return false; // não deu para sacar
46          }
47          else {
48              destino.deposita(valor);
49              return true;
50          }
51      }
```

Classe Conta (cont.)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
52      /**
53       * Método toString sobrescrito.
54       * @return uma string com a descrição do objeto
55       */
56      public String toString() {
57          String output = this.getClass().getName() + "[" +
58              "número:" + numero + "," +
59              "Titular:" + titular + "," +
60              "Saldo:" + saldo + "]";
61          return output;
62      }
63  }
```

Problema

Suponha que o banco quer controlar a **quantidade de contas** existentes no sistema. Como podemos implementar esse requisito?

Problema

Suponha que o banco quer controlar a **quantidade de contas** existentes no sistema. Como podemos implementar esse requisito?

- Ideia: Uma solução elegante seria se pudéssemos ter uma variável contadora compartilhada por todos os objetos da classe Conta. Quando um objeto modificasse o valor do contador, o seu novo valor estaria visível e acessível para todos os outros objetos da classe.

Atributos estáticos

Atributos estáticos são atributos compartilhados por todas as instâncias de uma classe.

Atributos estáticos

> Atributos estáticos são atributos compartilhados por todas as instâncias de uma classe.

- Em Java, atributos estáticos são declarados com o modificador `static`, que deve ser declarado antes do tipo de dado do atributo.

```
private static int totalDeContas;
```

Atributos estáticos

Atributos estáticos são atributos compartilhados por todas as instâncias de uma classe.

- Em Java, atributos estáticos são declarados com o modificador `static`, que deve ser declarado antes do tipo de dado do atributo.

```
private static int totalDeContas;
```

- Somente um valor é armazenado em um atributo estático, e caso este valor seja modificado por uma das instâncias da classe, a modificação será refletida em todas as outras instâncias da classe.

Atributos estáticos

- Atributos estáticos são também conhecidos como atributos de classes, já que estes atributos poderão ser acessados diretamente usando o nome da classe, sem que seja necessária a criação de uma instância da classe e uma referência para esta instância.

- Em contraste, atributos que podem ter diferentes valores para cada instância da mesma classe (como os vistos nos exemplos anteriores) são conhecidos como atributos de instâncias.

Atividade

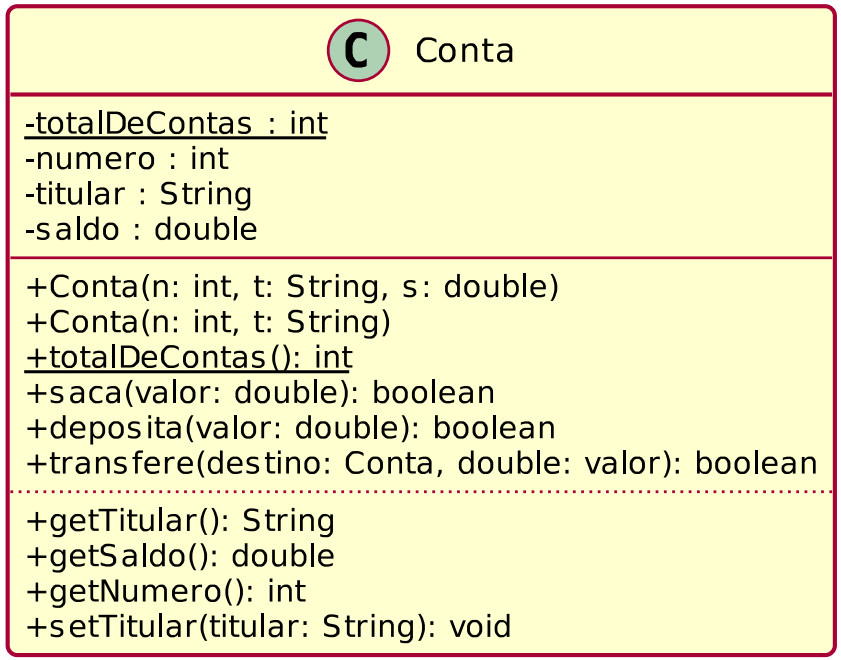
- Modifique a classe `Conta` de modo a solucionar o problema posto anteriormente usando o conceito de atributo estático.

- Codifique uma aplicação Java que teste a sua classe, criando várias contas e imprimindo a quantidade de contas criadas usando como base o atributo estático criado.

`static` e diagramas UML

- Os membros estáticos de uma classe são indicados sublinhando os seus nomes no diagrama de classes UML.
  - atributos estáticos
  - métodos estáticos

- Programadores Java traduzem o sublinhando para a palavra-chave `static`.



A classe `java.lang.Math`

- Java fornece a classe Math do pacote java.lang que possui duas constantes estáticas: Math.PI e Math.E.
  - `Math.PI` (3.141592653589793) é a razão da circunferência do círculo pelo seu diâmetro.
  - `Math.E` (2.718281828459045) é o valor base para o logarítmo natural.
  - Estas classes são declaradas na classe Math com os modificadores public, static e final.

Um exemplo de classe Java

com constantes estáticas públicas

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  public class MathConst {
2      // A raiz quadrada de 2
3      public static final double raizDe2 =
       1.4142135623730950488;
4
5      // A raiz quadrada de 3
6      public static final double raizDe3 =
       1.7320508075688772935;
7
8      // A raiz quadrada de 5
9      public static final double raizDe5 =
       2.2360679774997896964;
10
11     // A raiz quadrada de 6: podemos usar as constantes já
       definidas
12     public static final double raizDe6 = raizDe2*raizDe3;
13 }
```

Um exemplo de Aplicação Java

que faz uso da classe MathConst

```
1  import java.util.Locale;
2
3  class DemoMathConst {
4      public static void main(String[] args) {
5          Locale.setDefault(Locale.US);
6          System.out.println("PI = " + Math.PI);
7          System.out.println("Raiz(2) = " + MathConst.raizDe2);
8          System.out.println("Raiz(3) = " + MathConst.raizDe3);
9          System.out.println("Raiz(5) = " + MathConst.raizDe5);
10         System.out.println("Raiz(6) = " + MathConst.raizDe6);
11         System.out.printf("Raiz(10) = %.3f%n", MathConst.
       raizDe2 * MathConst.raizDe5);
12     }
13 }
```

- **Obs.:** Note que não foi preciso instanciar objetos para usar as contantes, pois elas são estáticas (**constantes de classe**).

Métodos Estáticos

Métodos estáticos

- Vimos até agora que a maioria dos métodos executam em resposta a chamadas de métodos em objetos específicos.

- Contudo, algumas vezes um método realiza uma tarefa que não depende de um objeto. Por exemplo, nos laboratórios de programação, vimos métodos que calculavam o número de Euler e se um número é primo, e que não precisavam de um objeto para serem executados.

- Chamamos de métodos estáticos ou métodos de classe os métodos que não precisam de instâncias de classes para serem executados.

Alguns métodos estáticos da classe Math

- A classe `Math` do pacote `java.lang` é um exemplo de classe em que todos os seus métodos são estáticos (`static`).
- Essa classe fornece uma coleção de métodos que possibilitam a realização de cálculos matemáticos comuns.
- Para uma descrição completa da Classe, consulte a API do Java: `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Math.html`

Métodos estáticos da classe java.lang.Math

Método	Descrição
$abs(x)$	valor absoluto de x
$ceil(x)$	devolve o menor inteiro maior ou igual a x
$exp(x)$	devolve o valor e^x
$floor(x)$	devolve o maior inteiro menor ou igual a x
$log(x)$	logaritmo natural de x (base e)
$pow(x, y)$	devolve o valor x^y
$max(x, y)$	maior valor entre x e y
$min(x, y)$	menor valor entre x e y
$sqrt(x)$	raiz quadrada de x
$sin(x)$	seno de x (x em radianos)
$cos(x)$	cosseno de x (x em radianos)
$tan(x)$	tangente de x (x em radianos)

- x e y são do tipo `double`, assim como o retorno dos métodos.

Métodos estáticos

- Métodos estáticos em uma classe devem ser declarados com o modificador `static`, que deve preceder o tipo de retorno do método e que pode ser combinado com os modificadores de acesso ao método.
  - Exemplo: `public static void main(String[] args)`

Métodos estáticos

- Métodos estáticos em uma classe devem ser declarados com o modificador `static`, que deve preceder o tipo de retorno do método e que pode ser combinado com os modificadores de acesso ao método.
  - Exemplo: `public static void main(String[] args)`
- Métodos estáticos podem ser usados em classes que contenham o método `main` para servirem como sub-rotinas deste.
- A aplicação mais frequente de métodos estáticos é a criação de bibliotecas de métodos — classes que contêm somente métodos estáticos, geralmente agrupados por função, como a classe Math.

O método `main`

Método `main`

- Quando você executa a JVM com o comando `java`, a JVM tenta invocar o método `main` da classe que você especificou.
  - Declarar main como `static` permite à JVM invocar `main` sem criar uma instância da classe.

Método `main`

- Quando você executa a JVM com o comando `java`, a JVM tenta invocar o método `main` da classe que você especificou.
  - Declarar main como `static` permite à JVM invocar `main` sem criar uma instância da classe.

- **Importante:** Se um método for chamado diretamente a partir do método `main`, este método deverá ser obrigatoriamente declarado como estático.

Método `main`

- Quando você executa a JVM com o comando `java`, a JVM tenta invocar o método `main` da classe que você especificou.
  - Declarar main como `static` permite à JVM invocar `main` sem criar uma instância da classe.

- **Importante:** Se um método for chamado diretamente a partir do método `main`, este método deverá ser obrigatoriamente declarado como estático.

- **Importante:** Se o método `main` for acessar atributos declarados na sua classe mas fora do método `main`, estes atributos também deverão ser declarados como `static`.

Método `main`

- Quando você executa a JVM com o comando `java`, a JVM tenta invocar o método `main` da classe que você especificou.
  - Declarar main como `static` permite à JVM invocar `main` sem criar uma instância da classe.

- **Importante:** Se um método for chamado diretamente a partir do método `main`, este método deverá ser obrigatoriamente declarado como estático.

- **Importante:** Se o método `main` for acessar atributos declarados na sua classe mas fora do método `main`, estes atributos também deverão ser declarados como `static`.

- **Importante:** Toda classe em Java pode conter um método `public static void main` – somente a `main` da classe usada para executar a aplicação é chamada.

Exemplo de classe com métodos estáticos


```java
import java.util.Scanner;

class EntradaDados {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print(
            "Digite 3 inteiros separados por espaço: ");
        int num1 = input.nextInt();
        int num2 = input.nextInt();
        int num3 = input.nextInt();

        int resultado = maximo(num1, num2, num3);
        System.out.println("Máximo é: " + resultado);
    }

    public static int maximo(int x, int y, int z) {
        int maior = x;
        maior = (y > maior) ? y : maior;
        maior = (z > maior) ? z : maior;
        return maior;
    }
}
```

Tipos Nativos × Tipos Referência

Tipos nativos

- Como já vimos, o Java possui oito tipos nativos: `char`, `boolean`, `byte`, `short`, `int`, `long`, `float`, `double`.

- Uma variável de tipo nativo guarda exatamente um valor do seu tipo nativo em um dado momento.

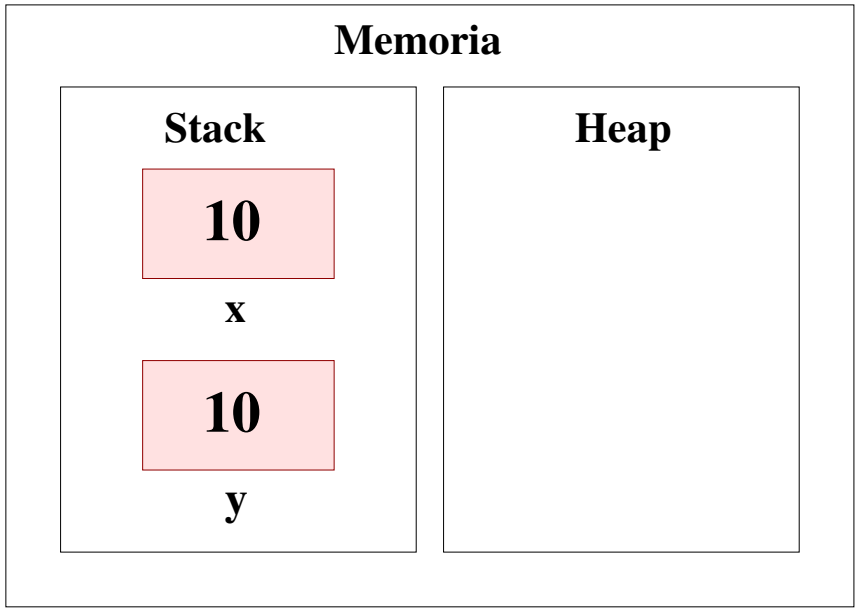
Tipos nativos

- Como já vimos, o Java possui oito tipos nativos: `char`, `boolean`, `byte`, `short`, `int`, `long`, `float`, `double`.

- Uma variável de tipo nativo guarda exatamente um valor do seu tipo nativo em um dado momento.

```
double x , y;
x = 10;
y = x;
```

y recebe uma cópia de x



Classes são tipos referência

- Variáveis cujo tipo são classes, guardam a localização dos objetos na memória.

- Elas devem ser entendidas como ponteiros ou referências.

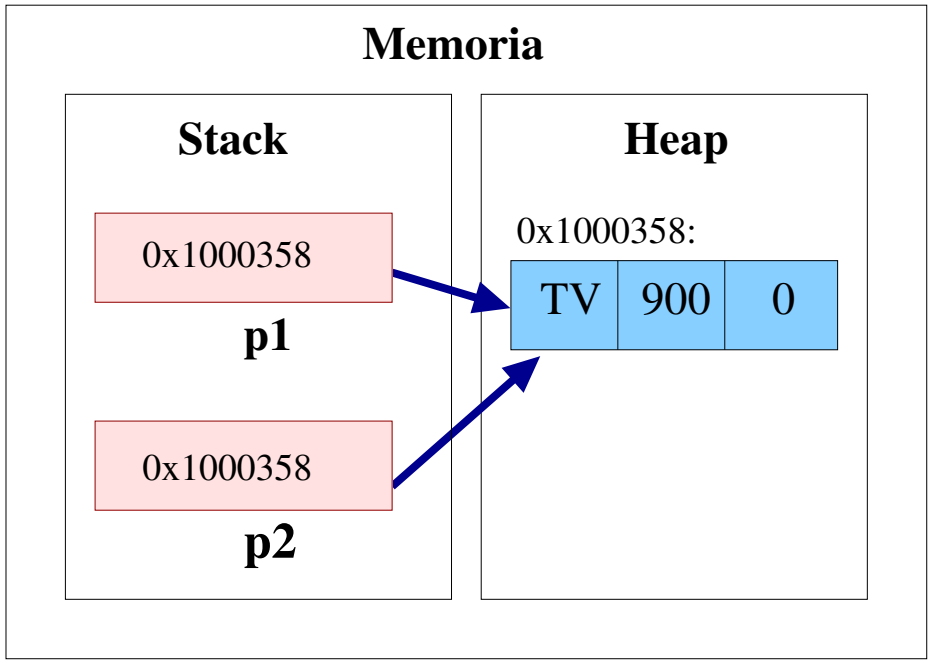
Classes são tipos referência

- Variáveis cujo tipo são classes, guardam a localização dos objetos na memória.

- Elas devem ser entendidas como ponteiros ou referências.

Product p1, p2;
p1 = new Product("TV",900.00,0);
p2 = p1;

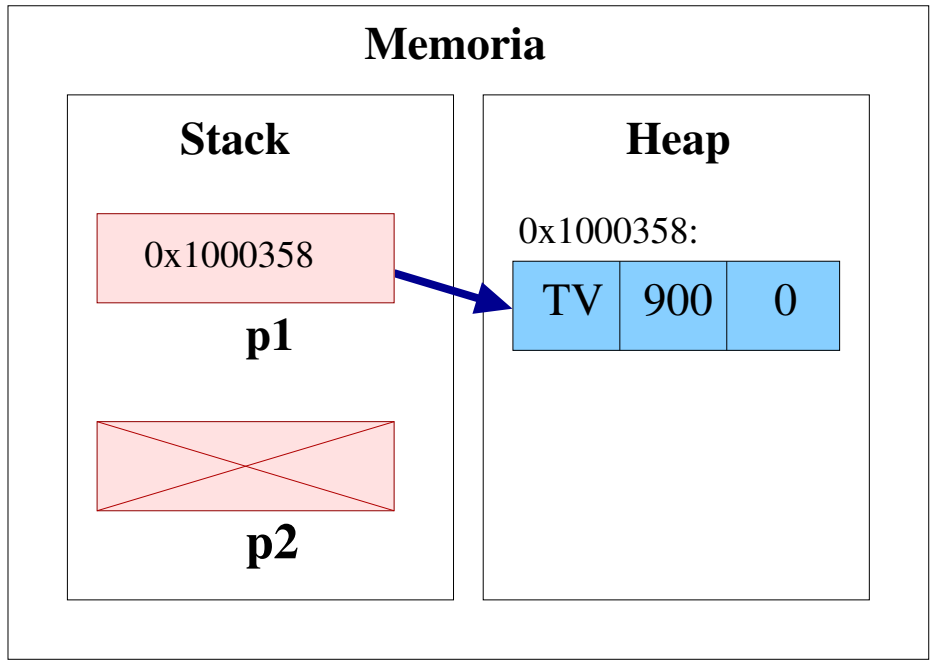
p2 passa a apontar
para onde p1 aponta



Valor `null`

- Variáveis de tipos referência aceitam o valor `null`, indicando que a variável referencia ninguém.

```
Product p1, p2;
p1 = new Product("TV",900.00,0);
p2 = null;
```



Valores-padrão

- Quando alocamos qualquer tipo estruturado (classe ou array), são atribuídos valores padrão aos seus elementos
  - números: `0`
  - boolean: `false`
  - char: `\u0000`
  - objeto: `null`

```
Product p = new Product();
```

Valores-padrão

- Quando alocamos qualquer tipo estruturado (classe ou array), são atribuídos valores padrão aos seus elementos
  - números: `0`
  - boolean: `false`
  - char: `\u0000`
  - objeto: `null`

```
Product p = new Product();
```

Analisar o arquivo Teste.java

Tipo referência vs. Tipo nativo

Tipo referência (classe)	**Tipo Primitivo**
Variáveis são referências	Variáveis guardam o valor
Objetos precisam ser instanciados usando new, ou apontar para um objeto já existente	Não instancia. Uma vez declarados, estão prontos para uso
Aceita valor null	Não aceita valor null
Y = X; "Y passa a apontar para onde X aponta"	Y = X; "Y recebe uma cópia de X"
Objetos instanciados no heap	Variáveis instanciadas no stack
Objetos não utilizados são desalocados em um momento próximo pelo *garbage collector*	Variáveis são desalocadas imediatamente quando seu escopo de execução é finalizado

Classes Empacotadoras e seus atributos e métodos estáticos

Classes empacotadoras (*Wrapper classes*)

- Todos os oito tipos nativos do Java possuem uma classe correspondente denominada classe empacotadora que pertencem ao pacote `java.lang`.

Primitive type	Wrapper type
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

Classes empacotadoras (*Wrapper classes*)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Todos os oito tipos nativos do Java possuem uma classe correspondente denominada classe empacotadora que pertencem ao pacote `java.lang`.

Primitive type	Wrapper type
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

- Classes empacotadoras possuem atributos e métodos estáticos que podem ser úteis em várias aplicações.

Classes empacotadoras são necessárias

- Todas as estruturas de dados do Collection Framework do Java e as classes do pacote java.util trabalham apenas com objetos e, portanto, as classes empacotadoras serão necessárias caso você queira armazenar tipos de dados nativos.

Criando classes empacotadoras

Classes empacotadoras possuem o método estático `valueOf` que recebe como argumento um valor de tipo nativo e retorna um objeto da sua respectiva classe empacotadora.

```
1  import java.util.Scanner;
2
3  public class CreateWrapper {
4      public static void main(String[] args) {
5          Scanner input = new Scanner(System.in);
6
7          Integer myInt = Integer.valueOf(input.nextInt());
8          Integer myInt2 = myInt;
9          Double myDouble = Double.valueOf(5.99);
10         Character myChar = Character.valueOf('A');
11
12         System.out.println(myInt); // chama toString()
13         System.out.println(myInt2); // chama toString()
14         System.out.println(myDouble); // chama toString()
15         System.out.println(myChar); // chama toString()
16     }
17 }
```

Wrappers são imutáveis

- Objetos de classe empacotadora guardam o valor de tipo nativo em um atributo declarado como `private final`.

- Uma vez atribuído um valor a esse atributo, não será possível modificá-lo.

Boxing e unboxing em classes empacotadoras

Boxing (ou empacotamento)

- **Autoboxing:** é o processo de conversão automática de um tipo de dado nativo em um objeto da sua classe empacotadora correspondente.

Boxing (ou empacotamento)

- **Autoboxing:** é o processo de conversão automática de um tipo de dado nativo em um objeto da sua classe empacotadora correspondente.

- Desde o Java 5, não precisamos mais usar o método estático `valueOf` das classes empacotadoras para converter os tipos primitivos em objetos.

- Desde o Java 9, os construtores das classes empacotadoras **estão descontinuados**, o que significa que não devemos usar construtores para criar objetos dessas classes.
  - O mecanismo de autoboxing faz isso automaticamente ou, se você preferir, pode usar o método estático `valueOf`.

Criando classes empacotadoras

usando o mecanismo de boxing


```
1  public class CreateWrapperBoxing {
2      public static void main(String[] args) {
3          Integer myInt = 5; // criando um novo Integer
4          Double myDouble = 5.99; // criando um novo Double
5          Character myChar = 'A'; // criando um novo Character
6
7          System.out.println(myInt); // chama toString()
8          System.out.println(myDouble); // chama toString()
9          System.out.println(myChar); // chama toString()
10
11         Integer intRef = myInt; // intRef = 5
12
13         if(intRef.equals(myInt))
14             System.out.println("possuem o mesmo valor");
15         else
16             System.out.println("possuem valores distintos");
17     }
18 }
```

Unboxing (ou desempacotamento)

- **Unboxing:** é o processo de conversão automática de um objeto de classe empacotadora em seu respectivo tipo nativo.

- O compilador do Java aplica *unboxing* quando um objeto de uma classe empacotadora é:
  - Passado como parâmetro para um método que espera um valor do tipo nativo correspondente.
  - Atribuído a uma variável do tipo nativo correspondente.

Unboxing — Exemplo


```
1  public class WrapperUnboxing {
2      public static void main(String[] args) {
3          Double myDouble = -4567.0; // boxing
4
5          double x = myDouble; // unboxing
6
7          System.out.println("Absolute value of " + myDouble
8              + " = " + Math.abs(x));
9      }
10 }
```

Classe Integer – Métodos não-estáticos

Métodos	**Descrição**
int compareTo(Integer a)	Compara dois valores Integer numericamente. Retorna 0 se x == y, menor que 0 se x <y; e maior que 0 se x >y
int intValue()	Retorna o valor deste Integer como um int
long longValue()	Retorna o valor deste Integer como um long
byte byteValue()	Retorna o valor deste Integer como um byte
short shortValue()	Retorna o valor deste Integer como um short
double doubleValue()	Retorna o valor deste Integer como um double
String toString()	Retorna uma String representando este Integer
boolean equals(Object obj)	Compara este Integer com o objeto obj
int hashCode()	Retorna um hash code para este Integer

- Consulte a API do java para as demais classes Wrappers.
- Analisar o arquivo WrapperExemplo2.java

Classe Integer

Atributos Estáticos

- `int MIN_VALUE`: constante que dá o menor valor que um int pode ter
- `int MAX_VALUE`: constante que dá o maior valor que um int pode ter
- `int BYTES`: o número de bytes usados para representar um int
- `int SIZE`: o número de bits usados para representar um int

Classe Integer

Atributos Estáticos

- `int MIN_VALUE`: constante que dá o menor valor que um int pode ter
- `int MAX_VALUE`: constante que dá o maior valor que um int pode ter
- `int BYTES`: o número de bytes usados para representar um int
- `int SIZE`: o número de bits usados para representar um int

```
1 class IntegerExemplo {
2     public static void main(String[] args) {
3         System.out.println("MIN_VALUE: " + Integer.MIN_VALUE);
4         System.out.println("MAX_VALUE: " + Integer.MAX_VALUE);
5         System.out.println("BYTES: " + Integer.BYTES);
6         System.out.println("SIZE: " + Integer.SIZE);
7     }
8 }
```

Classe Integer

Alguns Métodos Públicos e Estáticos

- `int max(int a, int b)`: retorna o valor do maior parâmetro
- `int min(int a, int b)`: retorna o valor do menor parâmetro
- `int parseInt(String s)`: converte a String `s` para int
- `int parseInt(String s, int radix)`: converte o valor da String `s` na base `radix` para decimal e retorna o valor
- `String toString(int i)`: converte o int i para String

Classe Integer

Alguns Métodos Públicos e Estáticos

- `int max(int a, int b)`: retorna o valor do maior parâmetro
- `int min(int a, int b)`: retorna o valor do menor parâmetro
- `int parseInt(String s)`: converte a String `s` para int
- `int parseInt(String s, int radix)`: converte o valor da String `s` na base `radix` para decimal e retorna o valor
- `String toString(int i)`: converte o int i para String

Como exemplo, veja a classe `IntegerExemplo2.java`

Classe Integer

Alguns Métodos Públicos e Estáticos

- `int max(int a, int b)`: retorna o valor do maior parâmetro
- `int min(int a, int b)`: retorna o valor do menor parâmetro
- `int parseInt(String s)`: converte a String `s` para int
- `int parseInt(String s, int radix)`: converte o valor da String `s` na base `radix` para decimal e retorna o valor
- `String toString(int i)`: converte o int i para String

Como exemplo, veja a classe `IntegerExemplo2.java`

Para mais detalhes sobre a classe `Integer`, consulte a API do Java: `https://cr.openjdk.java.net/~iris/se/17/latestSpec/api/java.base/java/lang/Integer.html`

Demais classes empacotadoras

- Todas as demais classes empacotadoras possuem os atributos estáticos `MIN_VALUE`, `MAX_VALUE`, `BYTES` e `SIZE`, e também o método estático `toString`.

- Com exceção da classe `Character`, todas as demais também têm os métodos estáticos `max` e `min`.

- Do mesmo modo que a classe `Integer` tem o método estático `parseInt`, as demais classes empacotadoras (com exceção da classe `Character`) têm o seu método "parse" equivalente, obtido trocando-se Int pelo nome do tipo nativo correspondente.
  - Veja o programa `WrappersExemplo.java`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM