

PROGRAMAÇÃO ORIENTADA A OBJETOS

Referência da linguagem Java

Atualizado para Java 8



3º período

Prof. Marcello Thiry <marcello.thiry@gmail.com>

JRE e JDK

3

❑ JRE (*Java Runtime Environment*)

- ▣ Máquina Virtual Java (JVM - Java Virtual Machine ou VM)
- ▣ Inclui também, plugins para navegadores executarem Applets

❑ JDK (*Java Development Kit*)

- ▣ Inclui o JRE, além de compiladores e ferramentas (ex: JavaDoc, Java Debugger)

 **Nossa escolha!!**

Onde baixar o JDK?

4

- <http://www.oracle.com/technetwork/java/javase/downloads/index.html>



Nossa escolha!!

Após a instalação

5

- ❑ set **JAVA_HOME**="C:\Program Files\Java\jdk<versão>"
 - ❑ Considerando a versão 8 instalada em C:\Java\jdk1.8.0:
 - set JAVA_HOME="C:\Program Files\Java\jdk1.8.0"

- ❑ set **PATH**=%JAVA_HOME%\bin;%PATH%

Se você quiser saber mais: <http://docs.oracle.com/javase/tutorial/essential/environment/paths.html>
http://www.java.com/pt_BR/download/help/path.xml

CLASSPATH

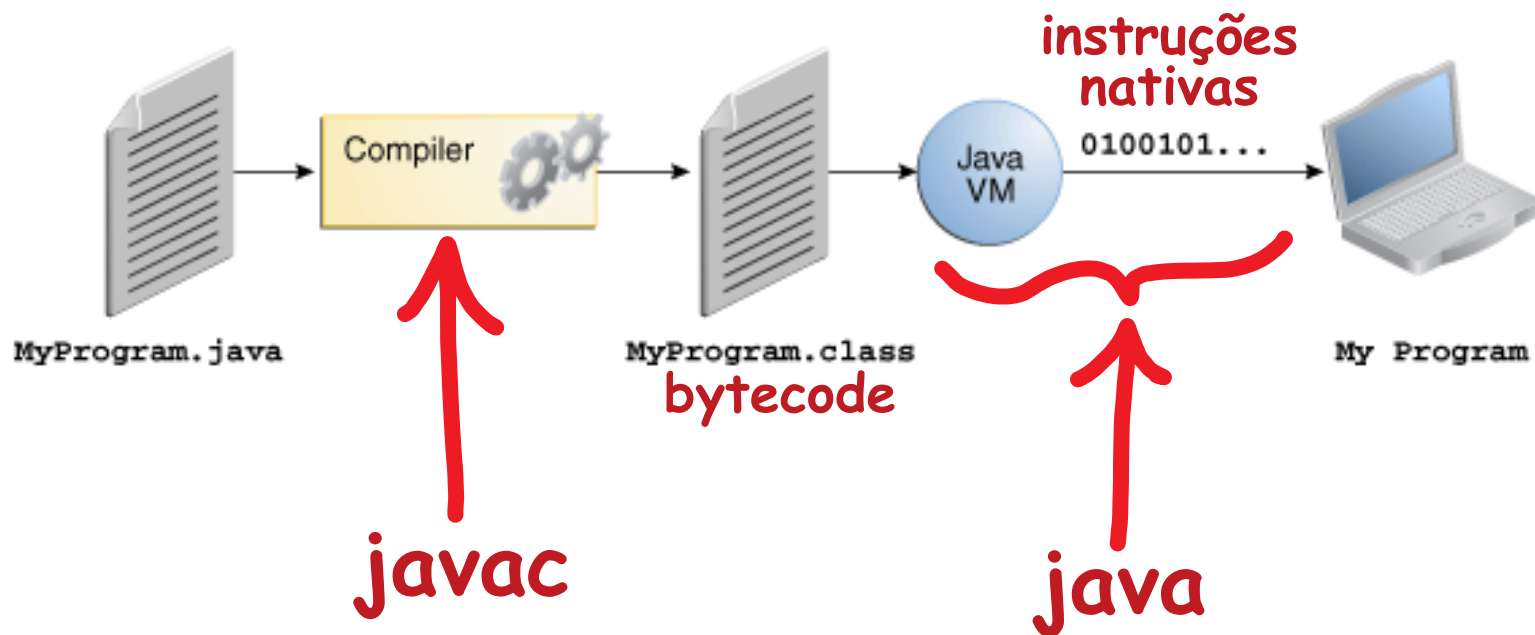
6

- ❑ Variável que informa para o JDK o local default para classes desenvolvidas pelos usuários
- ❑ Utilize preferencialmente a opção “**-cp**” via linha de comando
 - ▣ `javac -cp .;C:\users\classes;C:\tools\java\classes prog.java`
 - ▣ Permite que a CLASSPATH seja definida para cada aplicação sem afetar outras aplicações
 - ▣ O valor default para CLASSPATH é “.” (diretório atual)

Compilando e executando



7



Máquina Virtual



8

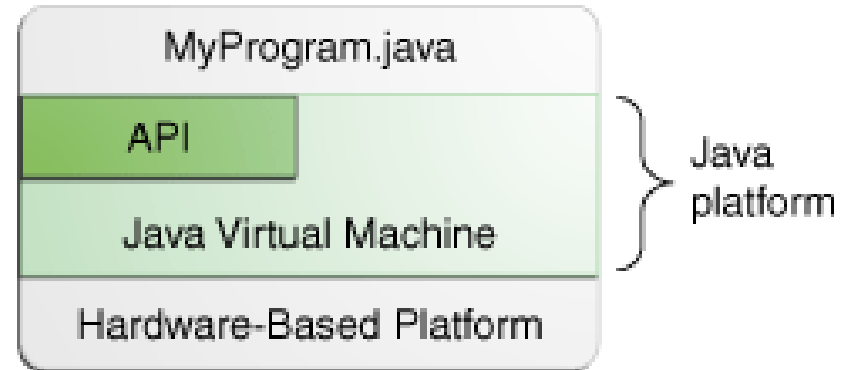
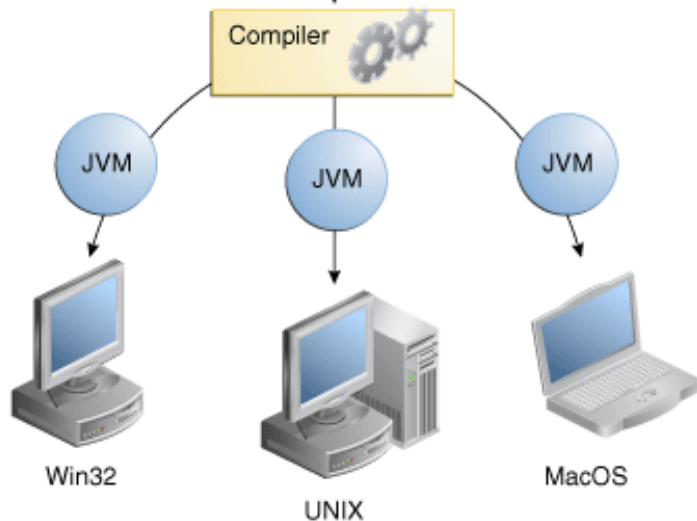
- Java VM (*Virtual Machine*)
 - ▣ Software que carrega e executa bytecode
 - ▣ Torna o Java independente de hardware
 - ▣ Oferece **portabilidade**

Plataforma Java

Java Program

```
class HelloWorldApp {  
    public static void main(string [] args) {  
        System.out.println("Hello World!");  
    }  
}
```

HelloWorldApp.java



Principais programas (executáveis)

10

Programa	Nome	Descrição
javac	Compilador Java	Compila programas java
java	Interpretador Java	Executa programas java
javadoc	Documentador	Gera documentação pelos comentários
Appletviewer	Java Applet Viewer	Visualiza applets sem navegador
jar	Compactador	Compacta fontes em um único arquivo
javap	Disassembler Java	Mostra o código de um arquivo .class
jdb	Java Debugger	Depura programas java

Extensões comuns

11

- “.java”
 - ▣ Código fonte Java
- “.class”
 - ▣ Código “compilado” → código objeto → bytecode
- “.jar”
 - ▣ Java Archive
 - Coleção de arquivos .class e outros recursos (formato ZIP)

```

C:\Users\Thiry>java
Use: java [-options] class [args...]
      (para executar uma classe)
ou java [-options] -jar jarfile [args...]
      (para executar um arquivo jar)
em que as opções incluem:
  -d32      usar um modelo de dados de 32 bits, se estiver disponível
  -d64      usar um modelo de dados de 64 bits, se estiver disponível
  -server   para selecionar a VM "server"
  -hotspot  é um sinônimo da VM "server" [obsoleto]
            A VM default é server.

  -cp <caminho de pesquisa da classe dos diretórios e arquivos zip/jar>
  -classpath <caminho de pesquisa da classe dos diretórios e arquivos zip/jar>

            Uma lista separada por ; de diretórios, archives JAR
            e archives ZIP nos quais serão procurados os arquivos de class
e.
  -D<nome>=<valor>      define uma propriedade do sistema
  -verbose[:<classe>]  ativa a saída detalhada
  -version             imprime a versão do produto e sai do programa
  -version:<valor>     requer a execução da versão especificada
  -showversion         imprime a versão do produto e continua
  -jre-restrict-search [-no-jre-restrict-search]
                        inclui/exclui JREs privados do usuário na pesquisa de versão
  -? -help             imprime esta mensagem de ajuda
  -X                  imprime a ajuda sobre opções não padronizadas
  -ea[:<nome do pacote>...[:<nome da classe>]]
  -enableassertions[:<nome do pacote>...[:<nome da classe>]]
                        ativa asserções com granularidade especificada
  -da[:<nome do pacote>...[:<nome da classe>]]
  -disableassertions[:<nome do pacote>...[:<nome da classe>]]
                        desativa asserções com granularidade especificada
  -esa : -enablesystemassertions
                        ativa asserções do sistema
  -dsa : -disablesystemassertions
                        desativa asserções do sistema
  -agentlib:<nome da biblioteca>[=<opções>]
                        carrega a biblioteca de agentes nativa <nome da biblioteca>, p
ou exemplo: -agentlib:hprof
                        consulte também: -agentlib:jdwp=help e -agentlib:hprof=help
  -agentpath:<nome do caminho>[=<opções>]
                        carrega a biblioteca de agentes nativa com base no nome do cam
inho completo
  -javaagent:<caminho do arquivo jar>[=<opções>]
                        carrega o agente da linguagem de programação Java; consulte ja
va.lang.instrument
  -splash:<caminho da imagem>
                        mostra a tela de abertura com a imagem especificada
Consulte http://www.oracle.com/technetwork/java/javase/documentation/index.html
para obter mais detalhes.

```

java

javac

```

C:\Users\Thiry>javac
Usage: javac <options> <source files>
where possible options include:
  -g          Generate all debugging info
  -g:none     Generate no debugging info
  -g:<lines,vars,source> Generate only some debugging info
  -nowarn     Generate no warnings
  -verbose    Output messages about what the compiler is doing
  -deprecation Output source locations where deprecated APIs are used
  -classpath <path> Specify where to find user class files and annotations processors
  -cp <path> Specify where to find user class files and annotations processors
  -sourcepath <path> Specify where to find input source files
  -bootclasspath <path> Override location of bootstrap class files
  -extdirs <dirs> Override location of installed extensions
  -endorseddirs <dirs> Override location of endorsed standards path
  -proc:<none,only> Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses default discovery process
  -processorpath <path> Specify where to find annotation processors
  -d <directory> Specify where to place generated class files
  -s <directory> Specify where to place generated source files
  -implicit:<none,class> Specify whether or not to generate class files for implicitly referenced files
  -encoding <encoding> Specify character encoding used by source files
  -source <release> Provide source compatibility with specified release

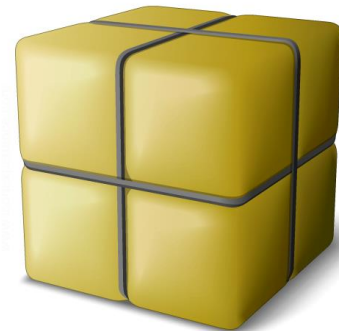
  -target <release> Generate class files for specific VM version
  -version        Version information
  -help          Print a synopsis of standard options
  -Akey[=value] Options to pass to annotation processors
  -X            Print a synopsis of nonstandard options
  -J<flag>       Pass <flag> directly to the runtime system
  -Werror        Terminate compilation if warnings occur
  @<filename>    Read options and filenames from file

```

Package (Pacote)

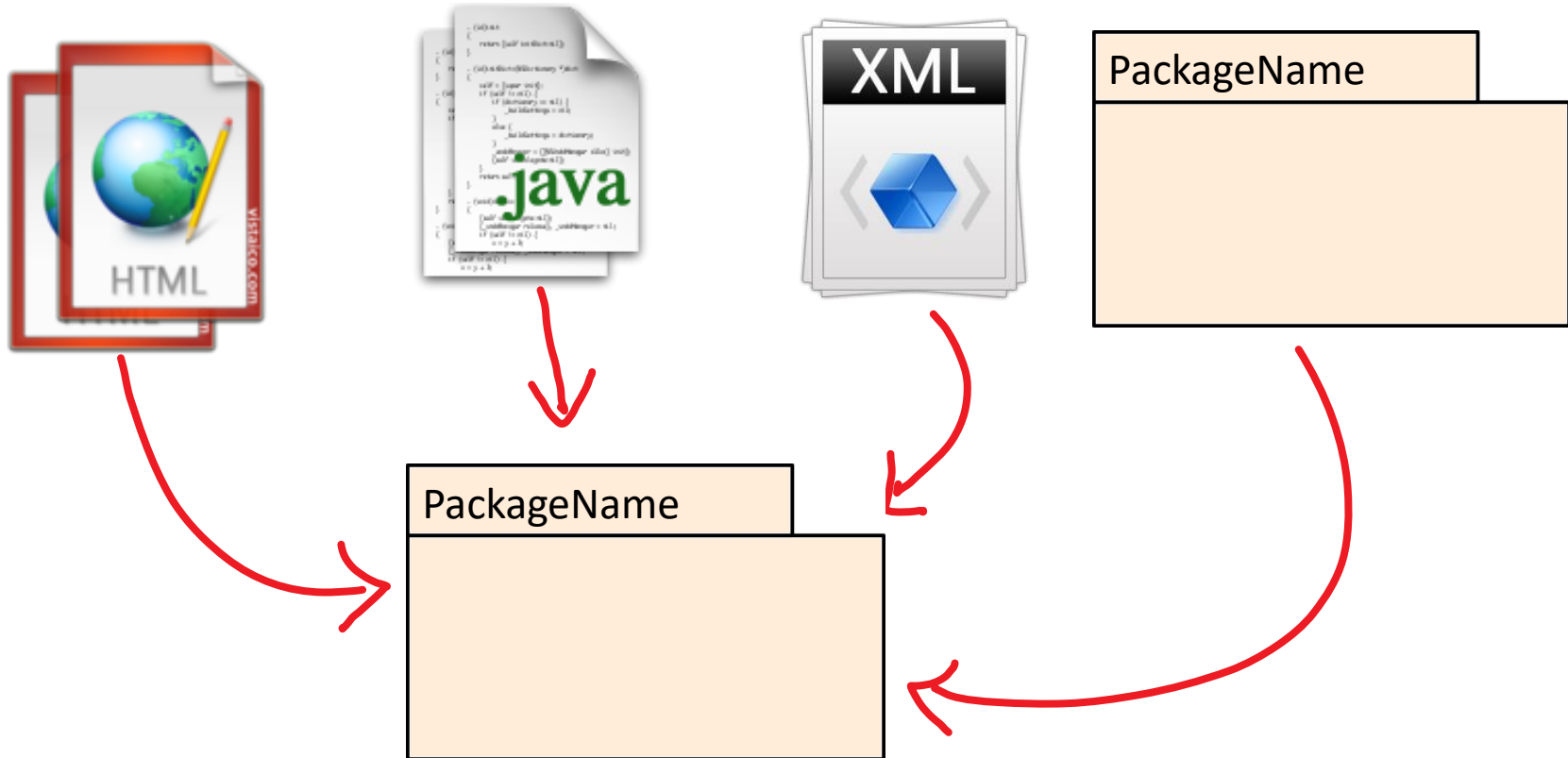
13

- ❑ Coleção de classes relacionadas e outros recursos
 - ▣ Similar a um diretório
- ❑ Permite proteger o acesso e gerenciar *namespaces*
 - ▣ Um namespace é um identificador único (caminho formado pela concatenação de nomes de pacote):
 - EX: br.univali.poo.kob.atividade1.MinhaClasse



Package (Pacote)

14



Package (Pacote) em Java

15

- ❑ Permite a organização do código-fonte em vários arquivos físicos
- ❑ Similar a estrutura de diretório usada no sistema operacional
- ❑ Um pacote pode conter códigos-fonte, bibliotecas, etc., além de outros pacotes

Package (Pacote) em Java

16

```
package br.univali.kob.pool.aula02a;
```

```
import java.time.LocalDate;
```

```
import java.time.Period;
```

```
public class Person {
```

```
    private String name;
```

```
    private LocalDate dateOfBirth;
```

```
    public Person(String name, LocalDate dateOfBirth) {
```

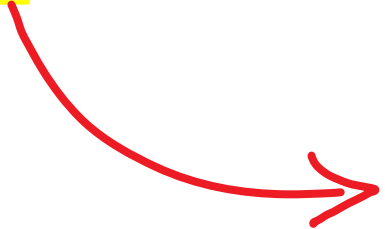
```
        this.name = name;
```

```
        this.dateOfBirth = dateOfBirth;
```

```
    }
```

```
    public void setName(String name) {
```

```
        this.name = name;
```



Pacote da
classe Person

Package (Pacote) em Java

17

```
package br.univali.kob.pool.aula02a;
```

```
import java.time.LocalDate;
```

```
import java.time.Period;
```

```
public class Person {
```

```
    private String name;
```

```
    private LocalDate dateOfBirth;
```

```
    public Person(String name, LocalDate dateOfBirth) {
```

```
        this.name = name;
```

```
        this.dateOfBirth = dateOfBirth;
```

```
    }
```

```
    public void setName(String name) {
```

```
        this.name = name;
```

Namespace da
classe LocalDate



Package (Pacote) em Java

18

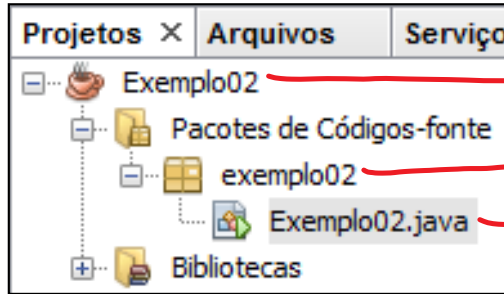
- ❑ Para pequenas aplicações
 - ▣ Pode ser viável usar um único pacote
- ❑ Para aplicações maiores
 - ▣ Usar vários pacotes
 - ▣ Evita problemas com nomes duplicados de classes e permite localizar o código da classe de forma eficiente
 - ▣ Uso de pacotes de terceiros (ex: bibliotecas)

Package (Pacote) em Java

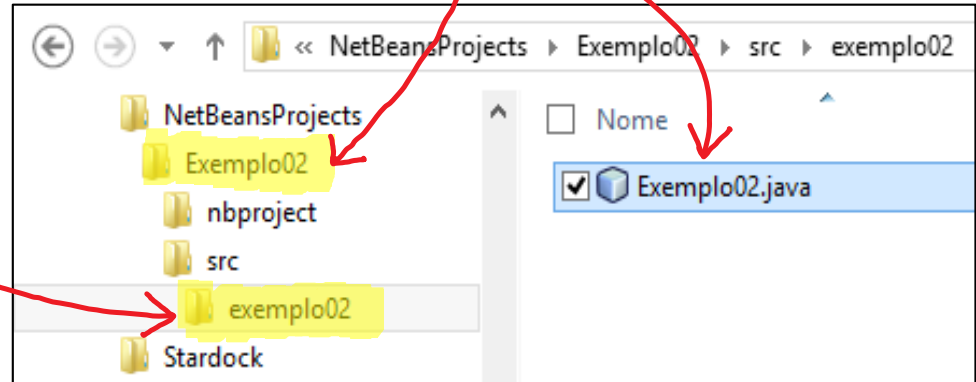
- ❑ Se uma classe **A** que pertence a um pacote **nome.do.pacote**
 - ❑ Seu nome completo (namespace) é **nome.do.pacote.A**
 - ❑ O compilador Java espera encontrar o arquivo **A.java** em um subdiretório **nome/do/pacote**
 - ❑ Este diretório deve estar localizado na variável de ambiente CLASSPATH ou deve ser informado na linha de comando da compilação

Package (Pacote) no NetBeans

20



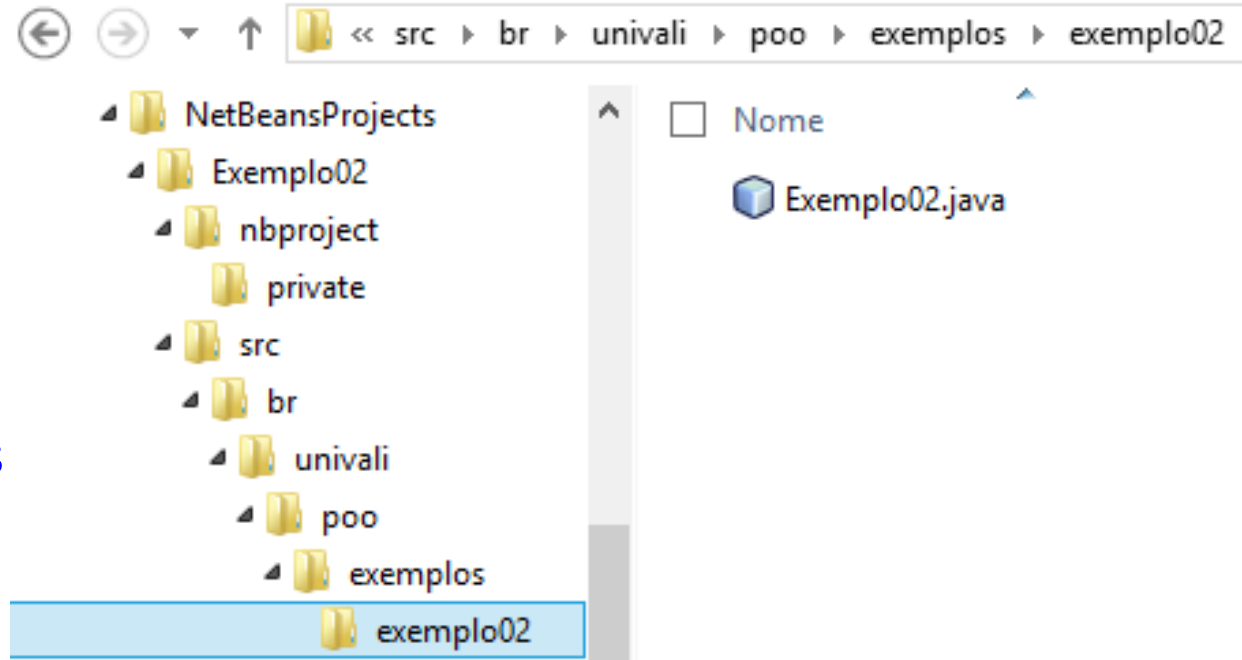
- Se a classe **Exemplo02** está no pacote **exemplo02**, então ela deve estar num diretório com o mesmo nome: **exemplo02**



Package (Pacote) no NetBeans

21

- Se a classe **Exemplo02** está no pacote `br.univali.poo.exemplos.exemplo02`, então ela está no diretório:
`br/univali/poo/exemplos/exemplo02`



Package (Pacote)

22

□ Visão modular

▣ Manter alta **coesão**

- Todas as classes agrupadas no pacote contribuem para uma tarefa em comum
- As classes compartilham um objetivo em comum

▣ Manter baixo **acoplamento**

- Depender de poucos
- Ser usado por muitos

Package (Pacote)







23

- ❑ O pacote deve ser criado com classes reutilizáveis
 - ▣ Ou todas as classes dentro do pacote são reutilizáveis ou então, nenhuma é
 - ▣ Classes que tendem a ser reutilizadas juntas devem pertencer ao mesmo pacote
- ❑ Classes que podem ser modificadas pelo mesmo motivo devem pertencer ao mesmo pacote

Roteiros

24

n > 2017-2 > Roteiro 01 > Roteiro 01 - Material do aluno







Nome	Data de
 br.univali.kob.poo1.aula01.Person.pdf	06/02/2
 br.univali.kob.poo1.p01_simple_class.Person.pdf	22/12/2
 PersonTest.java	01/01/2
 POO1 - Roteiro 01 - Atividades.pdf	02/08/2
 POO1 - Roteiro 01 - Exercícios de fixação.pdf	02/08/2
 POO1 - Slides de referência Java (Marcello Thir...	02/08/2

Orientações
passo a passo

Roteiros

25

n > 2017-2 > Roteiro 01 > Roteiro 01 - Material do aluno







Nome	Data de
 br.univali.kob.poo1.aula01.Person.pdf	06/02/2
 br.univali.kob.poo1.p01_simple_class.Person.pdf	22/12/2
 PersonTest.java	01/01/2
 POO1 - Roteiro 01 - Atividades.pdf	02/08/2
 POO1 - Roteiro 01 - Exercícios de fixação.pdf	02/08/2
 POO1 - Slides de referência Java (Marcello Thir...	02/08/2

Exploram os principais
conceitos trabalhados
no roteiro

Roteiros

26

n > 2017-2 > Roteiro 01 > Roteiro 01 - Material do aluno

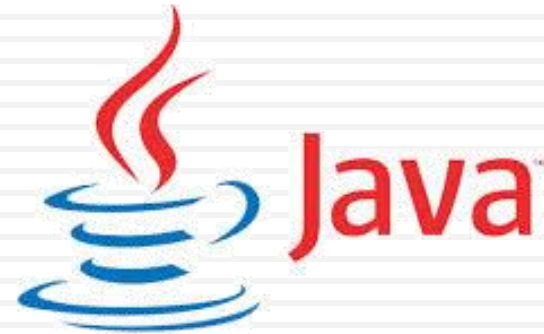
Nome	Data de
 br.univali.kob.poo1.aula01.Person.pdf	06/02/2
 br.univali.kob.poo1.p01_simple_class.Person.pdf	22/12/2
 PersonTest.java	01/01/2
 POO1 - Roteiro 01 - Atividades.pdf	02/08/2
 POO1 - Roteiro 01 - Exercícios de fixação.pdf	02/08/2
 POO1 - Slides de referência Java (Marcello Thir...	02/08/2

Utilize como
referência em
todas as aulas

27

Linguagem Java (básico)

Elementos da linguagem, comentários (javadoc), tipos primitivos, sintaxe das principais estruturas e exemplos



Meu primeiro programa Java

28

```
package br.univali.poo;
```

```
/**  
 * @author Marcello Thiry  
 *  
 */
```

```
public class Main {
```

```
    /**  
     * @param args - lista dos parâmetros recebidos quando o programa é  
     *               executado via comando de linha (SO)  
     */
```

```
    public static void main(String[] args) {  
        System.out.println("meu primeiro programa Java");  
    }
```

```
}
```

Um programa Java inicia pelo
método **main(...)**

Palavras reservadas

29

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

- ❑ **const** e **goto** são reservadas, mas não utilizadas
- ❑ Qualquer palavra reservada devem estar em caixa baixa
- ❑ **null**, **false** e **true** parecem ser reservadas, mas são literais

Identificadores

30

- ❑ Um identificador é utilizado para nomear variáveis, atributos, parâmetros e métodos
 - ▣ Esclarecendo, o nome de uma variável é um identificador
- ❑ Compilador é “case-sensitive”
 - ▣ Existe diferenciação entre letras maiúsculas e minúsculas

Identificadores

31

```
package br.univali.kob.pool.aula02a;
```

```
import java.time.LocalDate;
```

```
import java.time.Period;
```

```
public class Person {
```

```
    private String name;
```

```
    private LocalDate dateOfBirth;
```

```
    public Person(String name, LocalDate dateOfBirth) {
```

```
        this.name = name;
```

```
        this.dateOfBirth = dateOfBirth;
```

```
    }
```

```
    public void setName(String name) {
```

```
        this.name = name;
```

Lembre-se do estilo de escrita
que adotamos para os
identificadores: CamelCase

<http://pt.wikipedia.org/wiki/CamelCase>

Atribuição

32

- `<identificador> = <valor>`

- ▣ Para tipos primitivos, comparações são feitas com "=="

- Exemplos:

- ▣ `int qtdeMaximaAlunos = 10;`

- ▣ `float umValor = 10.0;`

- ▣ `int outroValor = 20;`

Comentários

33

```
// Comentário que termina ao final da linha  
// Este comentário pode ser usado desde o  
// início da linha ou colocado ao lado do  
// código
```

```
comando; // exemplo de comentário
```


Comentários

```
/* Comentário que pode utilizar várias  
 * linhas são iniciados com /* e  
 * terminados com o símbolo abaixo  
 */
```

```
/* É uma boa prática organizar os  
 * comentários iniciando com um  
 * asterisco  
 */
```

javadoc



335

- javadoc é um gerador de documentação a partir de tags em comentários
 - ▣ As tags são chamadas **annotations** (metadados)
 - `@return`
 - `@param`
 - `@author`
 - `@throws`
 - ...

javadoc

36

```
/**  
 * Primeiro parágrafo (sempre iniciando com /**).  
 * <p>  
 * Segundo parágrafo.  
 * Que pode ter múltiplas linhas.  
 * <p>  
 * Terceiro parágrafo.  
 */  
public ...
```

javadoc

37

```
/**
 * Primeiro parágrafo.
 * <p><ul>
 * <li>o primeiro ítem
 * <li>o segundo ítem
 * <li>o terceiro ítem
 * </ul><p>
 * Segundo parágrafo.
 */
public ...
```

<http://blog.joda.org/2012/11/javadoc-coding-standards.html>

javadoc

```
/**
 * Texto javadoc.
 *
 * @param foo  o parâmetro foo
 * @param bar  o parâmetro bar
 * @return o conteúdo baz
 */
public String process(String foo, String bar) {...}
```

javadoc

39

```
/**  
 * O símbolo # permite criar um link para o membro  
 * foo, seja ele um atributo ou uma operação  
 *  
 * @see #foo  
 * ...  
 */
```

<https://pt.wikipedia.org/wiki/Foobar>

javadoc

```
/**  
 * {@inheritDoc}. Você pode herdar a documentação  
 * feita na superclasse e completá-la aqui.  
 *  
 * Outra boa prática é declarar todas as exceções  
 * disparadas por um método  
 *  
 * @throws IOException se não foi possível...  
 */
```

javadoc

```
/**  
 * Se você está redefinindo uma operação  
 * ({@literal @}Override) e não há nenhuma  
 * informação adicional que complete  
 * a documentação, você pode deixar a operação  
 * sem javadoc. A ferramenta utilizará o javadoc  
 * da superclasse.  
 */
```


javadoc

42

```
/**
 * Se você deseja incluir um trecho de código, utilize a tag
 * {@literal <pre>} para texto preformatado e {@literal {@code texto}}.
 * Note que a annotation literal faz com que o javadoc desconsidere
 * caracteres especiais e tags.
 * <pre>
 * {@code
 * se a > b então // mantida a indentação e quebra de linha
 *     mostra a;
 * senão
 *     mostra b;
 * }
 * </pre>
 */
```

Uma classe em Java

```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

Visibilidade e modificadores de acesso

44

Notação visual	Modificador de acesso	A parte é visível...
+	public	dentro da própria classe e para qualquer outra classe
-	private	somente dentro da própria classe
#	protected	somente dentro do próprio pacote e das subclasses em outros pacotes
~	package	somente dentro da própria classe e das classes dentro do mesmo pacote

Visibilidade “package” em Java

45

- ❑ Para representar a visibilidade “package”, você não deve especificar um modificador de acesso
 - ▣ Quando não é especificado um modificador de acesso, o Java assume que a visibilidade é do tipo “package”

```
package br.univali.poo.calculo;
```

```
class ClasseVisibilidadePackage {  
    int atributoVisibilidadePackage;  
    ...  
}
```

Visibilidade “package” em Java

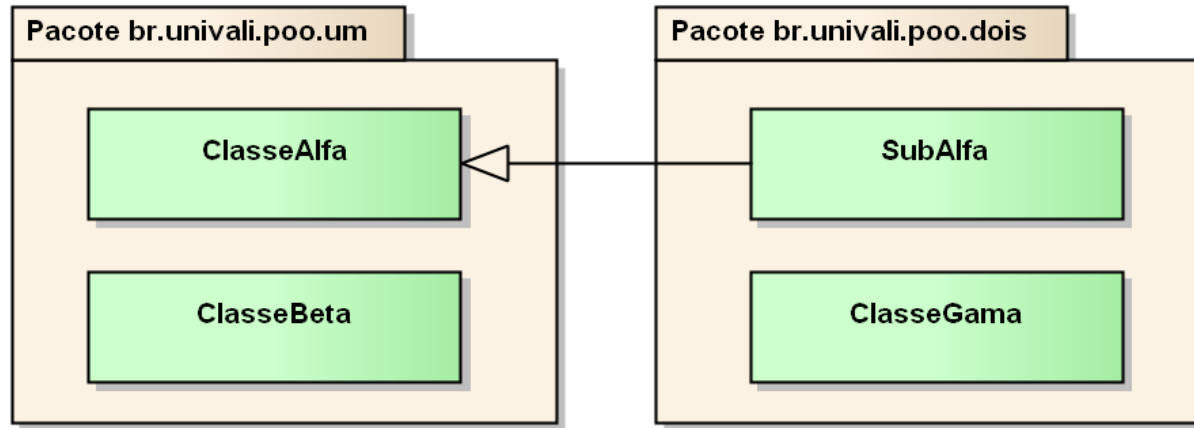


46

- ❑ Para cada arquivo “.java” deve haver, pelo menos, uma classe “public” com o mesmo nome do arquivo
- ❑ As demais classes declaradas no mesmo arquivo devem ser “package”
 - ▣ Ou seja, não devem especificar um modificador de acesso

Visibilidade de atributos e operações

47



Modificador na ClasseAlfa	ClasseAlfa	ClasseBeta	SubAlfa	Gama
public	S	S	S	S
protected	S	S	S	N
sem modificador	S	S	N	N
private	S	N	N	N

Atributos em Java

48

```
public class Pessoa {  
    private String nome;
```

→ Atributo privado

```
    public void setNome(String nome) {  
        this.nome = nome;  
    }
```

```
    public String getNome() {  
        return this.nome;  
    }
```

```
}
```

Método sem retorno

49

```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

**Método público
sem retorno
(void)**



Método com retorno

50

```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getNome() {  
        return this.nome;  
    }  
}
```

Método público
com retorno
(ex: String)

Construtor em Java

51

```
public abstract class Figura2D {  
  
    private PontoCartesiano posicao;  
  
    public abstract void desenhar();
```

```
    public Figura2D(PontoCartesiano posicao) {  
        this.posicao = posicao;  
    }
```

```
    // continuação...
```

```
}
```

Construtor da
classe Figura2D



Classe e Operação Abstrata em Java

52

```
public abstract class Figura2D {
```

Definição de uma
classe abstrata

```
    private PontoCartesiano posicao;
```

```
    public abstract void desenhar();
```

Operação abstrata

```
    public Figura2D(PontoCartesiano posicao) {  
        this.posicao = posicao;  
    }
```

```
    // continuação...
```

```
}
```

Generalização

53

```
public class Retangulo extends Figura2D {  
  
    public Retangulo(PontoCartesiano posicao) {  
        super(posicao);  
    }  
  
    @Override  
    public void desenhar() {  
        // código omitido...  
    }  
  
}
```

Retangulo é
uma subclasse
de Figura2D

Chamando o construtor herdado

54

```
public class Retangulo extends Figura2D {  
  
    public Retangulo(PontoCartesiano posicao) {  
        super(posicao);  
    }  
  
    @Override  
    public void desenhar() {  
        // código omitido...  
    }  
  
}
```

Invoca o método da superclasse: quando é um construtor não indica o nome do método

Redefinição do método de uma operação

55

```
public class Retangulo extends Figura2D {
```

```
    public Retangulo(PontoCartesiano posicao) {  
        super(posicao);  
    }
```

```
@Override
```

```
    public void desenhar() {  
        // código omitido...  
    }
```

Redefinição do método da operação
desenhar() herdada de Figura2D

```
}
```

Classe Genérica (Generics em Java)

```
package br.univali.poo.exemplos.pilha;
```

```
public class Pilha<TipoItem> {
```

```
    private final int tamanho;
```

```
    private int topo;
```

```
    private TipoItem[] elementos;
```

```
    public Pilha() {
```

```
        this(10);
```

```
    }
```

```
    public Pilha(int tamanho) {
```

```
        this.tamanho = tamanho > 0 ? tamanho : 10;
```

→ Parâmetro de tipo

Exceções: try...catch

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
} catch (TipoExcecao2 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
}
```

1. O código controlado pela cláusula **try** é executado
2. Se ocorrer uma exceção, o controle é desviado para o código dentro da cláusula **catch**
3. Se a exceção ocorrida estiver sendo considerada pela cláusula **catch**, o código de tratamento da exceção é executado (caso contrário, a exceção é repassada ao ambiente de execução)
4. Se não ocorrer uma exceção, o tratamento da exceção (**catch**) não é executado

Exceções: try...catch

58

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
} catch (TipoExcecao2 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
}
```


- ❑ Você pode adicionar quantas cláusulas **catch** forem necessárias
 - ▣ Permite tratamento específico para cada tipo de exceção considerado
- ❑ **objExcecao** é o objeto de exceção criado quando a exceção ocorre
 - ▣ Ele possui várias informações sobre a exceção

Exceções: try...catch

59

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
}  
catch (TipoExcecao2 | TipoExcecao3 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
}
```

Você pode considerar
mais de uma exceção
em uma mesma
cláusula **catch**



Exceções: try...finally

60

```
try {  
    <bloco de código>  
} finally {  
    <código sempre executado>  
}
```

1. O código controlado pela cláusula **try** é executado
2. Se ocorrer uma exceção, o controle é desviado para o código dentro da cláusula **finally** que é executado
3. Se não ocorrer uma exceção, após o último comando dentro da cláusula **try**, o código dentro da cláusula **finally** é executado (ou seja, o código dentro da cláusula **finally** é sempre executado)

Combinando **try...catch...finally**

61

```
try {  
    <bloco de código>  
} catch (TipoExcecao1 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao1>  
} catch (TipoExcecao2 objExcecao) {  
    <código para tratar>  
    <a exceção TipoExcecao2>  
} finally {  
    <código sempre executado>  
}
```

Este bloco é
executado
mesmo quando uma
exceção ocorrer!!

Tipos primitivos

62

- ❑ Somente 8 tipos primitivos em Java, todos os demais são **objetos**
- ❑ Estes tipos ficaram na linguagem pela sua velocidade de interação, e por isso não foram transformados em objetos
- ❑ Como não são objetos a sua declaração já cria a variável em memória não necessitando de construtores

Tipos primitivos

63

- ❑ Números inteiros em notação de complemento de dois
 - ❑ **byte** (8 bits)
 - ❑ **short** (16 bits)
 - ❑ **int** (32 bits)
 - ❑ **long** (64 bits)
- ❑ Números em notação de ponto flutuante
 - ❑ **float** (32 bits)
 - ❑ **double** (64 bits)
- ❑ **boolean**
 - ❑ Pode assumir o valor true ou o valor false
- ❑ **char**
 - ❑ Caractere em notação Unicode de 16 bits (alfanumérico)

Números inteiros

64

- O valor default é **0**
- No código fonte, qualquer **constante numérica** será um **int** a não ser que o caractere "**L**" esteja no final do número, indicando que ela é um **long**

Números inteiros

65

- **byte** (8 bits com sinal)

 - ▣ -128 a +127

- **short** (16 bits com sinal)

 - ▣ -32.768 a +32.767

- **int** (32 bits com sinal)

 - ▣ -2.147.483.648 a +2.147.483.647

- **long** (64 bits com sinal)

 - ▣ -9.223.372.036.854.775.808 a +9.223.372.036.854.775.807

Números inteiros

66

□ Exemplos:

□ 100L → long

□ 100 → int

□ 0L → long

□ 0 → int

Pontos flutuantes

67

- ❑ O valor default é **0.0**
- ❑ No código fonte, qualquer número com decimais será um **double** a não ser que o caractere "**f**" ou "**F**" estejam no final do número, indicando que ele é um **float**
- ❑ Um **double** pode, opcionalmente, terminar com o caractere "**d**" ou "**D**"

Pontos flutuantes

68

- **float** (32 bits com sinal)

- ▣ -1.40239846E-45 a +3.40282347E+38

- **double** (64 bits com sinal)

- ▣ -4.94065645841246544E-324 a
+1.79769313486231570E+308

Pontos flutuantes

69

□ Exemplos:

- 100.0f → float
- 100.00d → double
- 0.5 → double
- 0.5F → float
- 0.3D → double

Notação científica

70

□ 5.9736×10^{24}

▣ `double massaPlanetaTerra = 5.9736e24;`

□ $0,0049 = 4.9 \times 10^{-3}$

▣ `double valor = 4.9e-3;`

Problemas com precisão

71

```
double d1 = 0.1;  
double d2 = 0.2;  
System.out.println(d1+d2);
```

run:

0.30000000000000004

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Problema na forma de
representação binária
do tipo double na JVM
(segue a norma IEEE 754)

A representação binária de alguns valores (como 0.1) vira uma
dízima periódica

Solução para precisão

72

- ❑ `java.math.BigDecimal`
 - ▣ Pontos flutuantes de precisão arbitrária
 - Você pode definir o nível de precisão
 - ▣ Utilização:
 - Valores financeiros
 - Cálculos complexos (ex: construção civil)

<http://www.devmedia.com.br/java-bigdecimal-trabalhando-com-mais-precisao/30286>

<https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html>

java.math.BigDecimal

73

```
BigDecimal big1 = new BigDecimal("0.1");
BigDecimal big2 = new BigDecimal("0.2");
BigDecimal sum = big1.add(big2);
BigDecimal mult = big1.multiply(big2);
sum = sum.setScale(2, RoundingMode.HALF_EVEN);
mult = mult.setScale(2, RoundingMode.CEILING);
System.out.println(sum.toString());
System.out.println(mult.toString());
```

BigDecimal
é imutável

run:

0.30

0.02

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

java.text.NumberFormat



7/24

- Diversas possibilidades de formatação para números
 - ▣ Utiliza configurações locais
 - Pontos decimais, dinheiro, separador de milhares, ...

```
BigDecimal valorFinanceiro = new BigDecimal("325.34");  
valorFinanceiro = valorFinanceiro.setScale(2, BigDecimal.ROUND_HALF_EVEN);  
System.out.println(NumberFormat.getCurrencyInstance().format(valorFinanceiro));
```

run:

R\$ 325,34

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Caracteres

75

- ❑ **char** (16 bits sem sinal)
 - ❑ Representa um valor da tabela Unicode
 - Unicode é uma tabela que permite representar caracteres de qualquer idioma
 - Valores 0 a 255 correspondem aos valores da tabela ASCII
 - ❑ O valor default é o código zero: “\0”
 - ❑ Pode representar um número inteiro de 16 bits sem sinal
 - Uma variável **char** pode ser atribuída com valores de 0 a 65535

Valores booleanos

76

- **boolean** (8 bits)

- true ou false
- Podem ser utilizados diretamente em expressões de retorno para testes como **if**, **switch**, **while** e **for**
- Não podem ser comparados com variáveis ou constantes numéricas

Resumindo: valores default

77

- ❑ **byte** = 0
- ❑ **short** = 0
- ❑ **int** = 0
- ❑ **long** = 0L
- ❑ **float** = 0.0f
- ❑ **double** = 0.0
- ❑ **boolean** = false
- ❑ **char** = '\u0000'
- ❑ **Object** = null
 - ▣ Objetos são inicializados com **null**, pois necessitam que a instância seja criada explicitamente, o que não ocorre em tipos primitivos

Literais

78

- Um literal é um valor escrito no código fonte e identificado como de um tipo primitivo

```
int i = 10;           // literal int
char c = 'a';         // literal char
boolean b = true;     // literal boolean
double d = 002.11d;   // literal double
int quinze = 0xF;     // literal int → valor = 15
```

Literais caracteres (escapes)

79

```
❑ char a = 'A';           // letra A
❑ char b = '&';           // caractere &
❑ char c = '\u004D';      // letra M (Unicode)
❑ char d = '\n';          // quebra de linha
❑ char e = '\t';          // tabulação
❑ char f = '\r';          // retorno de carro
❑ char g = '\b';          // backspace
❑ char h = '\f';          // quebra de página
❑ char i = '\\';          // barra
❑ char j = '\'';          // apóstrofo
❑ char k = '\"';          // aspas
❑ char l = 65;            // char recebe um valor int
```

A classe Object

80

- ❑ Qualquer classe Java é uma subclasse de Object
- ❑ Operações herdadas:
 - ❑ `public final Class<?> getClass()`
 - ❑ `public long hashCode()`
 - ❑ `public boolean equals(Object)`
 - ❑ `public String toString()`
 - ❑ `protected Object clone()`
 - ❑ ...

A classe Object

81

□ E ainda outras operações para uso com threads

- `public final void notify()`
- `public final void notifyAll()`
- `public final void wait()`
- `public final void wait(long)`
- ...

A classe System

82

- ❑ Contem vários atributos e métodos úteis
 - ▣ Entrada padrão (in)
 - ▣ Saída padrão (out)
 - ▣ Saída de erro padrão (err)
 - ▣ Acesso a variáveis de ambiente
 - ▣ ...
- ❑ Esta classe não pode ser instanciada

A classe System

83

□ System.in

```
System.in.read(); // lê um caractere do console
```

□ System.out

```
System.out.println("mensagem para console padrão");
```

□ System.err

```
System.err.println("mensagem de erro");
```

A classe String

84

- Representa cadeias (strings) de caracteres
- Todas as literais em Java, como “abc” são implementadas como instâncias desta classe

```
String texto = "Abc";  
  
System.out.println(texto);  
System.out.println(texto.toUpperCase() + " --> ABC");  
System.out.println(texto.toLowerCase() + " --> abc");
```

A classe String

85

- ❑ A classe **String** inclui operações para:
 - ❑ Examinar caracteres individuais da sequência
 - ❑ Comparar com outras strings
 - ❑ Buscar strings
 - ❑ Extrair substrings
 - ❑ Criar uma cópia de uma string com todos os caracteres convertidos para caixa baixa ou caixa alta
 - ❑ ...

Literais String

86

- Um literal string consiste de 0 ou mais caracteres dentro de aspas

```
"" // uma string vazia
```

```
"\" // uma string contendo "
```

```
"Isto eh uma string" // uma string de tamanho 18
```

```
"Isto eh uma string " +  
"em duas linhas" // uma string em 2 linhas
```

Comparando e manipulando Strings

```
String hello = "Hello";
```

```
String lo = "lo";
```

```
System.out.println(hello.equals("Hello") + " --> true");
```

```
System.out.println(hello.equals("Hel" + "lo") + " --> true");
```

```
System.out.println(hello.equals("Hel" + lo) + " --> true");
```

```
System.out.println(hello.equals(("Hel" + lo).intern()) + " --> true");
```

```
System.out.println(hello.startsWith("h") + " --> false");
```

```
System.out.println(hello.equalsIgnoreCase("hello") + " --> true");
```

```
System.out.println(hello.endsWith("lo") + " --> true");
```

```
System.out.println(hello.length() + " --> 5");
```

```
System.out.println(hello.substring(2) + " --> llo");
```

```
System.out.println(hello.substring(2, 4) + " --> ll");
```

Comparando Strings

88

- ❑ Evite comparar objetos String com “==”
 - ❑ Você pode utilizar “==” entre tipos primitivos ou entre literais
 - ... (“**umaLiteralString**” == “**outraLiteralString**”)
- ❑ Utilize a operação “equals” ou “equalsIgnoreCase”

StringBuilder

```
StringBuilder output = new StringBuilder();  
    output.append("Primeira Linha\n");  
    output.append("Segunda linha");  
    output.append("...continuação da segunda linha\n");  
    output.append("Terceira linha\n");  
    System.out.print(output.toString());
```

run:

Primeira Linha

Segunda linha...continuação da segunda linha

Terceira linha

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

Conversões entre tipos primitivos

90

□ Conversão implícita

- ▣ O tamanho de uma variável é maior que o tamanho da variável ou que o valor que está sendo atribuído

- ▣ Exemplo:

```
int y = 10;
```

```
long x = y; // o tamanho de int é menor que o de long, logo ok
```

Conversões entre tipos primitivos

91

□ Conversão explícita

▣ O tamanho de uma variável é menor que o tamanho da variável ou o que valor que está sendo atribuído

▣ Exemplo:

```
long y = 10L;
```

```
int x = (int)y; // type casting1, pois long > int
```

1. Type casting = conversão de tipo

Conversões entre tipos primitivos

92

□ Type Casting

- ▣ Quando uma variável de um tipo menor está sendo atribuída com uma variável de um tipo maior, é necessário explicitar a conversão

```
int x = (int)y; // indica o tipo para qual o valor de y
               // será convertido
```

- Se esta informação não for colocada, ocorrerá um erro de compilação indicando tipos incompatíveis

Operadores aritméticos unários

variavel++	Pós-Incremento	Utiliza o valor atual na expressão avaliada e depois incrementa em 1
++variavel	Pré-incremento	Incrementa em 1 e então utiliza o novo valor na expressão avaliada
variavel--	Pós-decremento	Utiliza o valor atual na expressão avaliada e depois decrementa em 1
--variavel	Pré-decremento	Decrementa de 1 e então utiliza o novo valor na expressão avaliada

Operadores aritméticos (duas variáveis)

94

□ $x + y$

□ $x - y$

□ $x * y$

□ x / y

□ $x \% y$ *// resto da divisão de x por y*

Operadores lógicos e relacionais

95

- ❑ $x > y$ *// x é maior que y?*
- ❑ $x \geq y$ *// x é maior ou igual a y?*
- ❑ $x < y$ *// x é menor que y?*
- ❑ $x \leq y$ *// x é menor ou igual a y?*
- ❑ $x == y$ *// x é igual a y?*
- ❑ $x != y$ *// x é diferente de y?*
- ❑ $x || y$ *// x OU y*
- ❑ $x \&\& y$ *// x E y*
- ❑ $!x$ *// NÃO x*

Operadores bitwise

- $z = \sim x$ // Inverte os bits de x
- $z = x \& y$ // AND bit a bit entre x e y
- $z = x | y$ // OR bit a bit entre x e y
- $z = x \wedge y$ // XOR bit a bit entre x e y
- $z = x \ll y$ // Desloca bits de x para esquerda, y vezes
- $z = x \gg y$ // Desloca bits de x para direita, y vezes
- $z = x \ggg y$ // Preenche zeros a esquerda de x , y vezes

Precedência dos operadores

97

- ++ -- ! (cast)
- * / %
- + -
- << >> >>>
- <> <= >=
- == !=
- &
- ^
- |
- &&
- ||
- = *= /= %= += -= <<= >>= >>>= &= |=

Precedência dos operadores

98

1. Unário ++ -- ! ~ (cast)
2. Aritmético * / % + -
3. Shift << >> >>>
4. Comparação < <= > >= instanceof == !=
5. Bitwise & ^ |
6. Lógico && ||
7. Condicional ?:
8. Atribuição = "op="

Operadores compostos

99

<code>x += 5</code>	<code><==></code>	<code>x = x + 5</code>	<code>==></code>	<code>x = 5</code> // x valia 0
<code>x *= 2</code>	<code><==></code>	<code>x = x * 2</code>	<code>==></code>	<code>x = 10</code>
<code>x /= 2</code>	<code><==></code>	<code>x = x / 2</code>	<code>==></code>	<code>x = 5</code>
<code>x -= 3</code>	<code><==></code>	<code>x = x - 3</code>	<code>==></code>	<code>x = 2</code>
<code>x %= 5</code>	<code><==></code>	<code>x = x % 5</code>	<code>==></code>	<code>x = 2</code> // Resto da divisão
<code>x = 4</code>	<code><==></code>	<code>x = x 4</code>	<code>==></code>	<code>x = 6</code> // OR
<code>x ^= 1</code>	<code><==></code>	<code>x = x ^ 1</code>	<code>==></code>	<code>x = 7</code> // XOR
<code>x &= 255</code>	<code><==></code>	<code>x = x & 255</code>	<code>==></code>	<code>x = 7</code> // AND
<code>x <<= 2</code>	<code><==></code>	<code>x = x << 2</code>	<code>==></code>	<code>x = 28</code> // Shift para direita
<code>x >>= 4</code>	<code><==></code>	<code>x = x >> 4</code>	<code>==></code>	<code>x = 1</code> // Shift para esquerda
<code>x >>>= 7</code>	<code><==></code>	<code>x = x >>> 7</code>	<code>==></code>	<code>x = 0</code> // Coloca 0's a esquerda

A classe Scanner (entrada de dados)



100

- ❑ Permite ler tipos primitivos e strings usando expressões regulares
 - ▣ Quebra um texto em tokens usando um delimitador padrão
 - O delimitador default é o espaço em branco
 - ▣ Existem métodos “next” para converter os tokens em valores dos vários tipos primitivos

A classe Scanner (entrada de dados)

101

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

System.out.print("entre com um texto qualquer: ");
String palavra = s.next(); // pega token/palavra
System.out.printf("A primeira palavra é \"%s\\n\"", palavra);
String texto = s.nextLine(); // pega o resto da linha
System.out.printf("O resto da linha é \"%s\\n\"", texto);
```

A classe Scanner (entrada de dados)

102

```
Scanner s = new Scanner(System.in);  
  
s = new Scanner(System.in);  
System.out.print("entre com um número inteiro: ");  
int numeroInt = s.nextInt();  
System.out.printf("o número informado foi: \"%5d\\n\"", numeroInt);
```

A classe Scanner (entrada de dados)

103

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

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

```
System.out.print("entre com um número real "+  
                "(utilize vírgula para o "+  
                "ponto decimal): ");
```

```
float numeroFloat = s.nextFloat();
```

```
System.out.printf("o número informado foi: \"%5.2f\"%n", numeroFloat);
```

print, println, printf, ...

```
System.out.print("print não quebra linha");  
System.out.println("... mas, o println quebra!");  
System.out.println("estou em uma nova linha!");
```

```
System.out.printf("O resto da linha é \"%s\\n\"", texto);  
System.out.printf("o número informado foi: \"%5d\\n\"", numeroInt);  
System.out.printf("o número informado foi: \"%5.2f\\n\"", numeroFloat);
```

System.out.format

105

```
System.out.format("Valor inteiro (%d): %d\n", 100);
System.out.format("Valor float(%f): %f\n", new Float(100.3));
System.out.format("Valor float (%3.2f) 999.99: %3.2f\n", new Float(100.3));
System.out.format("Valor date (%td/%<tm/%<tY): %td/%<tm/%<tY\n",
    new GregorianCalendar(1543, Calendar.JANUARY, 25));
System.out.format("Notação científica (%2.4e): %2.4e\n", 5.9736e24f);
```

run:

```
Valor inteiro (%d): 100
Valor float(%f): 100,300003
Valor float (%3.2f) 999.99: 100,30
Valor date (%td/%<tm/%<tY): 25/01/1543
Notação científica (%2.4e): 5,9736e+24
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```


Condicionais (if)



106

```
if (<condição>) {  
    comando;  
    comando;  
    comando;  
} else {  
    comando;  
    comando;  
    comando;  
};
```

Condicionais (if)

107

```
if (variavel == valor) {  
    fazerAlgo();  
}
```

```
if (variavel == valor) {  
    fazerAlgo();  
} else {  
    fazerOutraCoisa();  
}
```

Condicionais (if)



108

```
if (variavel == valor) {  
    fazerAlgo();  
} else if (outraVariavel == outroValor) {  
    fazerOutraCoisa();  
} else {  
    fazerUmaTerceiraCoisa();  
}
```

Condicionais (switch)

109

```
switch (<expressão cardinal>) {  
    case <valor1>:  
        <comando>;  
        [break;]  
    case <valor2>:  
        <comando>;  
        [break;]  
    ...  
    default:  
        <comando default>;  
}
```

Condicionais (switch)

110

```
switch (variavel) {  
    case 0:  
        fazerAlgo();  
        break;  
    case 1:  
        fazerOutraCoisa();  
        break;  
    case 2:  
        fazerUmaTerceiraCoisa();  
        break;  
    default:  
        fazerAlgoDefault();  
}
```

Condicionais (operador ternário “?:”)

111

- Similar a uma instrução “if... else”
 - ▣ <expressão booleana> ? <ação *true*> : <ação *false*>

```
float media;  
media = 6.0f;  
System.out.println(media>=6.0f ? "aprovado" : "reprovado"); // aprovado  
media = 5.0f;  
System.out.println(media>=6.0f ? "aprovado" : "reprovado"); // reprovado
```

Laços (for)

112

```
for (<expr. inicial>; <condição>; <incremento>) {  
    <comandos>;  
}
```

```
// 0 até 9
```

```
for (int contador = 0; contador < 10; contador++) {  
    System.out.println("Contador é " + contador);  
}
```

Laços (for)

113

```
// 0 até 9  
for (int contador = 0; contador < 10; ++contador) {  
    System.out.println("Contador é " + contador);  
}
```

```
// cuidado: laço infinito  
for (;;) {  
    System.out.println("Laço infinito.");  
}
```


Laços (for)

114

```
// 0, 2, 4, 6, 8
for (int i = 0; i < 10; i += 2) {           // ou i=i+2
    System.out.println("Contador é " + i);
}
```

```
// 10, 8, 6, 4, 2, 0
for (int i = 10; i >= 0; i -= 2) {         // ou i=i-2
    System.out.println("Contador é " + i);
}
```

Laços (for)

115

```
for (String temp : list) {  
    System.out.println(temp);  
}
```

Percorre a lista “list” iniciando no primeiro elemento.

A cada passada do for, a variável “temp” é atribuída com o elemento da posição atual

“list” pode ser, por exemplo, do tipo ArrayList.

Alternativa para um laço (Java 8+)

116

```
ArrayList<Integer> vetor = new ArrayList<>();  
vetor.add(1); // autoboxing, conversão int para Integer  
vetor.add(2); // autoboxing, conversão int para Integer  
  
vetor.stream().forEach((element) -> {  
    System.out.println(element.intValue());  
});
```

Utilizando programação funcional (*stream* e *lambda*)

Laços (while)

117

```
while (<condição>) {  
    <comandos>;  
}
```

```
// contador = 1, 2, ..., 10  
int contador = 0;  
while (contador++ < 10) {  
    System.out.println("Contador é " + contador);  
}
```

Laços (while)

118

```
// contador = 1, 2, ..., 9
int contador = 0;
while (++contador < 10) {
    System.out.println("Contador é " + contador);
}

// cuidado: laço infinito
while (true) {
    System.out.println("Laço infinito.");
}
```

Laços (while)

119

```
Iterator<String> iterator = list.iterator();  
while (iterator.hasNext()) {  
    System.out.println(iterator.next());  
}
```

Percorre a lista “list” utilizando o padrão iterator
iterator.next() => elemento atual

“list” pode ser, por exemplo, do tipo ArrayList

Laços (do-while)

120

```
do {  
    <comandos>;  
} while (<condição>);
```

```
// 0 até 9  
int contador = 0;  
do {  
    System.out.println("Contador é " + contador);  
} while (++contador < 10);
```

Laços (do-while)



121

```
// 0 até 10
int contador = 0;
do {
    System.out.println("Contador é " + contador);
} while (contador++ < 10);

// cuidado: laço infinito
do {
    System.out.println("Laço infinito.");
} while (true);
```


Desvio em laços (break)

122

```
// 0 até 5
// break: força a saída da construção de um laço
//          (for, do, while ou switch)

int cont = 0;
while (cont < 10) {
    System.out.println("Contador é " + cont);
    if (cont++ == 5) { // testa antes e incrementa depois
        break;
    }
}
```

Desvio em laços (continue)

123

```
// -10 até 9 pulando o número zero  
// continue: desvia para o início do laço, na próxima  
//          iteração (for)
```

```
for (int cont = -10; cont < 10; cont++) {  
    if (cont == 0) {  
        continue;  
    }  
    System.out.println(cont) ;  
}
```

Vetores



124

```
int vetor[] = new int[3];  
vetor[0] = 1;  
vetor[1] = 2;  
vetor[2] = 3;  
for (int i = 0; i < 3; i++) {  
    System.out.println(vetor[i]);  
}
```

Matrices

125

```
String matriz[][] = new String[3][2];  
matriz[0][0] = "0,0";  
matriz[1][0] = "1,0";  
matriz[2][0] = "2,0";  
matriz[0][1] = "0,1";  
matriz[1][1] = "1,1";  
matriz[2][1] = "2,1";  
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 2; j++) {  
        System.out.println(matriz[i][j]);  
    }  
}
```

Inicializando vetores e matrizes

126

```
int vetor[] = {0, 1, 2};  
int matriz[][] = {{0, 1, 2},  
                  {3, 4, 5}};  
  
for (int i = 0; i < 3; i++) {  
    System.out.println(vetor[i]);  
}  
for (int linha = 0; linha < 2; linha++) {  
    for (int coluna = 0; coluna < 3; coluna++) {  
        System.out.println(matriz[linha][coluna]);  
    }  
}
```

Wrappers

127

- ❑ *Wrapper*
 - ▣ Empacotar, embrulhar, encapsular

- ❑ Uma **classe wrapper** modifica o comportamento de outra classe
 - ▣ Camada fina que fica sobre a classe encapsulada, a qual é responsável pelo trabalho real



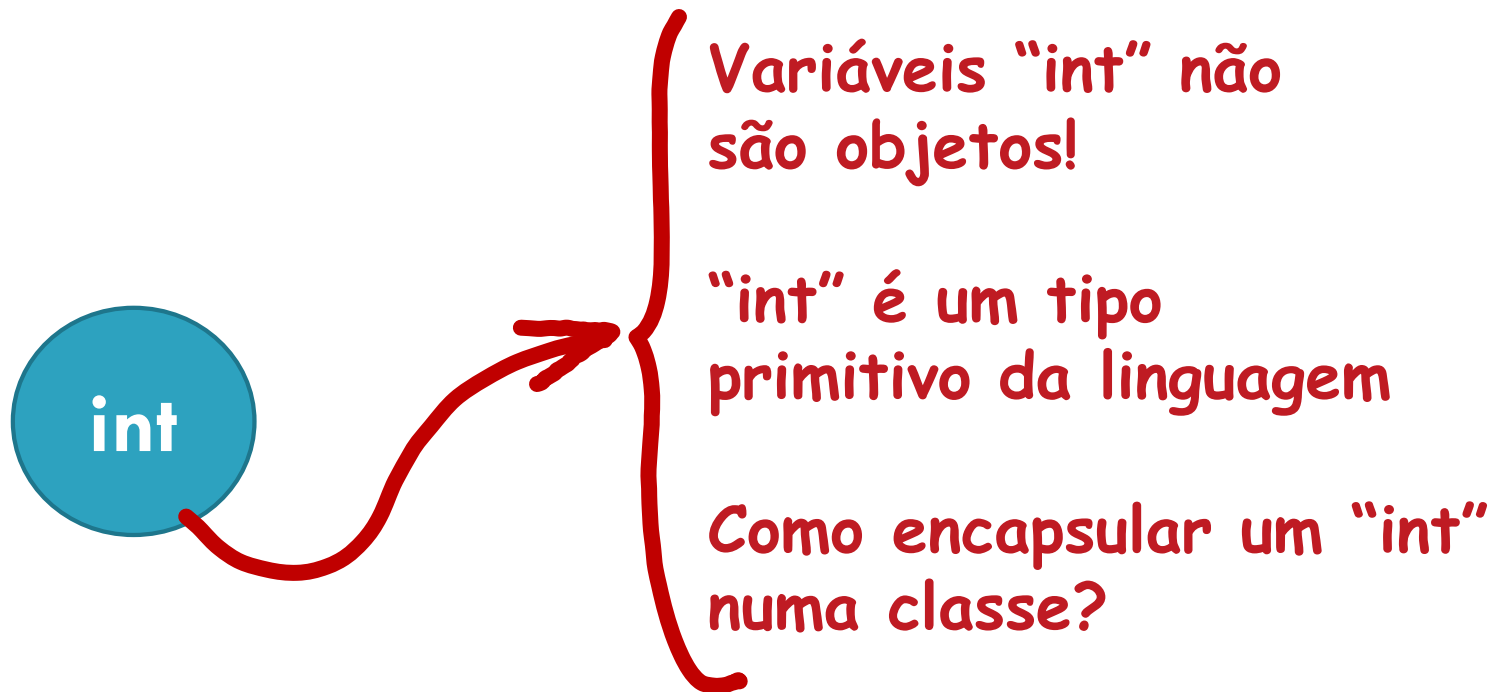
Wrappers

128

- As operações da classe *wrapper* podem ser usadas para adaptar uma classe que tenha uma interface diferente
 - ▣ A classe *wrapper* encapsula a funcionalidade de outra classe
 - ▣ Pode também simplificar o uso de um determinado objeto, reduzindo a quantidade de operações disponíveis na interface

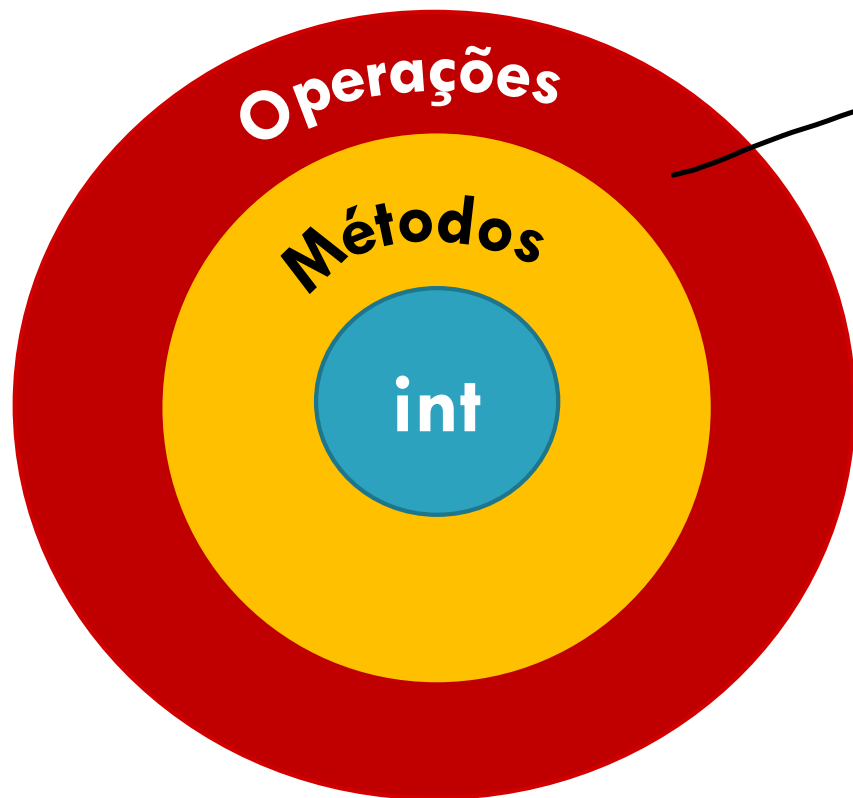
Wrappers para tipos primitivos

129



A classe java.lang.Integer

130



`Integer(int valor)`
`int intValue()`

...

`int compareTo(Integer outroInteger)`
`boolean equals(Object obj)`

...

`float floatValue()`

...

`static int rotateLeft(int i, int distancia)`
`static int rotateRight(int i, int distancia)`

...

Wrappers para tipos primitivos

131

- ❑ Se, por exemplo, você quer armazenar um conjunto de valores **int** nos elementos de um **ArrayList**
- ❑ **OBS** → Valores em um ArrayList devem ser objetos e não tipos primitivos
- ❑ Podemos utilizar a classe *wrapper* Integer

Autoboxing e Unboxing

132

```
ArrayList<Integer> vetor = new ArrayList<>();  
vetor.add(1); // autoboxing, conversão int para Integer  
vetor.add(2); // autoboxing, conversão int para Integer  
for (Integer i: vetor) {  
    System.out.println(i.intValue());  
}  
int num = vetor.get(0); // unboxing, conversão Integer para int  
System.out.println(num);
```

Autoboxing = do tipo primitivo para a classe *Wrapper*

Unboxing = da classe *Wrapper* para o tipo primitivo

Wrappers para tipos primitivos

133

- A linguagem Java oferece classes **wrappers** que encapsulam valores primitivos em classes que oferecem operações utilitárias para manipular estes valores

Tipo primitivo	Classe wrapper	Argumentos do construtor
byte	Byte	byte or String
short	Short	short or String
int	Integer	int or String
long	Long	long or String
float	Float	float, double or String
double	Double	doublebyte or String
char	Character	char
boolean	Boolean	boolean or String

Wrappers para tipos primitivos

134

```
System.out.println(Integer.parseInt("321"));
System.out.println(Integer.toBinaryString(4));
System.out.println(Integer.toHexString(16));
System.out.println(Float.parseFloat("2345.33"));
System.out.println(new Float(832.23).floatValue());
Integer num = 12; // autoboxing
String texto = num.toString();
```

Interface em Java

135

```
package br.univali.kob.poo1.selling;

import java.math.BigDecimal;

/**
 * Interface para objetos que podem (estão aptos --> "able") ser vendidos.
 */
public interface Sellable {

    /**
     * @return a descrição do objeto
     */
    public String getDescription();

    /**
     * @return o preço do objeto em reais
     */
    public BigDecimal getPrice();
}
```

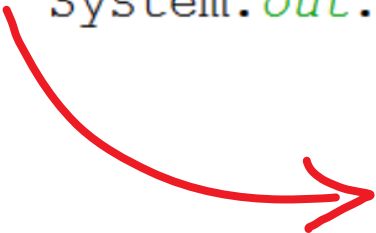
A codificação do método é delegada para as classes que Implementarem a interface

Interface com método default (Java 8+)

136

```
package br.univali.kob.pool.metodosdefault;  
  
public interface InterfaceA {  
    default void metodoDefault() {  
        System.out.println("Método default da Interface A");  
    }  
}
```


Não precisa ser
implementado
pela classe



Implementando a Interface...

137

```
package br.univali.kob.pool.metodosdefault;  
  
public class ExemploImplementacao implements InterfaceA {  
  
}
```



Herda o comportamento dos métodos default (quando houver)

Tipo “enum”

```
public enum DiaSemana {  
    SEGUNDA,  TERCA,  
    QUARTA,   QUINTA,  
    SEXTA,    SABADO,  
    DOMINGO  
}
```

Como utilizar “enum”

139

• • •

```
DiaSemana diaSemana;  
diaSemana = DiaSemana.QUARTA;  
if (diaSemana == DiaSemana.QUARTA) {  
    System.out.println("Funciona! Hoje é " + diaSemana);  
} else {  
    System.out.println("Algo deu errado!!");  
}
```

• • •

“enum” com valores

140

```
public enum Titulacao {  
  
    GRADUACAO(1, "Graduação"),  
    ESPECIALIZACAO(2, "Especialização"),  
    MESTRADO(3, "Mestrado"),  
    DOUTORADO(4, "Doutorado");  
  
    private final int codigo;  
    private final String descricao;  
  
    private Titulacao(int codigo, String descricao) {  
        this.codigo = codigo;  
        this.descricao = descricao;  
    }  
    ...  
}
```

“enum” com valores

141

```
public enum Titulacao {  
    ...  
    public int getCodigo() {  
        return codigo;  
    }  
  
    public String getDescricao() {  
        return descricao;  
    }  
}
```

“enum” com valores

142

• • •

```
Titulacao titulacao;  
titulacao = Titulacao.DOUTORADO;  
System.out.println("Código = " + titulacao.getCodigo());  
System.out.println("Descrição = " + titulacao.getDescricao());  
if (titulacao.equals(Titulacao.DOUTORADO)) {  
    System.out.println("Funcionou!!");  
} else {  
    System.out.println("Algo deu errado!!");  
}
```

• • •

Referências

143

- ❑ Java Tutorial (Oracle), disponível em:
 - ❑ <http://docs.oracle.com/javase/tutorial/index.html>
- ❑ Java API Specification (Oracle), disponível em:
 - ❑ <http://docs.oracle.com/javase/8/docs/api/overview-summary.html>
- ❑ Tutorial rápido para Netbeans, disponível em:
 - ❑ <https://netbeans.org/kb/docs/java/quickstart.html>

Referências

144

- ❑ Java Free Course (Home & Learn), disponível em:
 - ❑ <http://www.homeandlearn.co.uk/java/java.html>
- ❑ Outros links:
 - ❑ <http://www.javapractices.com>
 - ❑ <http://javafree.uol.com.br/>
 - ❑ http://www.java.com/pt_BR/
 - ❑ <http://www.guj.com.br/>