



Ciência da **Computação**

Programação Orientada a Objetos
Prof. Luciano Rodrigo Ferretto

Relembrando

- 4 Pilares: Abstração, Encapsulamento, Herança e Polimorfismo
 - Polimorfismo Estático - Sobrecarga de Métodos
 - Polimorfismo Dinâmico - Sobrescrita de Métodos
- Classe Object
- Palavra chave “**super**”
- Palavra chave “**abstract**” – Classes e métodos abstratos
- Palavra chave “**final**” – Classes, Métodos e Atributos
- Palavra chave “**static**” – Métodos e Atributos estáticos

Casting em Java: Dominando a Conversão de Tipos

Conversão de tipos

Casting em Java

- Java é uma linguagem **Fortemente Tipada**, ou seja, uma variável “nasce” com um tipo e permanece fiel a ele até o fim.
- Porém as vezes é preciso converter¹ o valor armazenado/referenciado por essa variável.
- Como dito, **não podemos alterar o tipo da variável**, mas podemos:
 - alterar/acessar o valor para um novo tipo.
 - armazenar/referenciar em uma nova variável.

Esse processo chamamos de **Casting**

1- Aqui leia-se “converter” ou “acessar” o valor de forma diferente

Casting em Java

- Em Java, casting é um conceito fundamental que permite a conversão de um tipo de dado em outro.
- Isso é útil quando você precisa trabalhar com diferentes tipos de dados em seu código.
- Existem duas formas principais de casting em Java:
casting implícito (widening) e casting explícito (narrowing).
- Além disso, precisamos entender que o Casting pode ser realizado tanto em valores do **tipo Primitivo**, quanto valores do **tipo Objeto**

Casting em Java – Tudo tem limite.

- Embora o casting permita a conversão entre diversos tipos, ele não é capaz de realizar milagres impossíveis.
- A compatibilidade entre os tipos de dados envolvidos é fundamental para o sucesso da operação.
- Tentar converter tipos incompatíveis é como tentar encaixar peças de quebra-cabeças que não se encaixam: o resultado será sempre um erro de compilação ou de execução.
- Tipos Primitivos: boolean → int = ERRO
- Tipos Objetos: Somente classes com uma relação de Herança. (e é preciso ter cuidado!)

Casting em Tipos Primitivos

- **Cast Implícito (Widening Cast):** Esse tipo de casting ocorre de forma automática quando você está convertendo um tipo menor para um tipo maior.
- Neste casting **não** há perda de dados.

```
int numeroInteiro = 100;  
long numeroLong = numeroInteiro; // cast implícito de int para long  
double numeroDouble = numeroInteiro; // cast implícito de int para double  
double numeroDouble2 = numeroLong; // cast implícito de long para double
```

Casting em tipos primitivos

- **Cast Explícito (Narrowing Cast):** Esse tipo de casting ocorre quando você está convertendo um tipo maior para um tipo menor.
- Isso pode resultar em perda de dados, pois o tipo de destino pode não ser capaz de representar todos os valores do tipo de origem.
- Um casting explícito deve ser feito pelo programador usando parênteses e especificando o tipo desejado.

```
double numeroDoublePI = 3.14159265359;  
float numeroFloatPI = (float) numeroDoublePI; // cast explícito de double para float  
int numeroInteiroPI = (int) numeroDoublePI; // cast explícito de double para int  
System.out.println("PI em double: " + numeroDoublePI + "\nPI em float: " +  
    numeroFloatPI + "\nPI em int: " + numeroInteiroPI);
```

```
PI em double: 3.14159265359  
PI em float: 3.1415927  
PI em int: 3
```


Casting em Tipos Objeto

- **Cast Implícito (Upcasting)**: Nesse caso, você pode fazer um casting de um objeto da subclasse para a superclasse (upcasting) de forma implícita.
- Casting em Tipos Objeto não altera o Objeto em si, este continua sendo o mesmo, apenas cria uma nova variável que vai referenciá-lo.
- Diferente dos tipos primitivos, neste caso **não** há perda de informações, porém, os métodos e atributos específicos da subclasse ficam inacessíveis.

Casting em Objetos - Upcasting

```
// Upcasting LivroFisico para Livro
// Isso é possível porque LivroFisico é uma subclasse de Livro
Livro varLivro = livroFisico;

// Upcasting LivroDigital para Livro
Livro varLivro2 = new LivroDigital("1984", "George Orwell", 1949, 328, 1.5, "PDF");
```

```
▼ varLivro = LivroFisico@10 "Título:
  anoPublicacao = 1954
  > autor = "J.R.R. Tolkien"
  > dimensoes = "15x23 cm"
  numeroExemplares = 5
  numeroPaginas = 1216
  > titulo = "O Senhor dos Anéis"
```

```
▼ varLivro2 = LivroDigital@11 "Títu
  anoPublicacao = 1949
  > autor = "George Orwell"
  > formatoArquivo = "PDF"
  numeroPaginas = 328
  tamanhoArquivo = 1,500000
```

Casting em Objetos - Upcasting

```

varLivro = LivroFisico@10 "Título:
anoPublicacao = 1954
> autor = "J.R.R. Tolkien"
> dimensoes = "15x23 cm"
numeroExemplares = 5
numeroPaginas = 1216
> titulo = "O Senhor dos Anéis"

```

```
// Exibindo informações do livro físico
System.out.println("Livro Físico:");
System.out.println(varLivro.toString());
System.out.println("Tipo: " + varLivro.getTipoLivro());
System.out.println("Tem " + varLivro.getNumeroExemplares() + " exemplares");
System.out.println("Número de exemplares: " + varLivro.getNumeroExemplares());
```

The method `getNumeroExemplares()` is undefined for the type `Livro` Java(67108964)

Casting em Tipos Objeto

- **Cast Explícito (Downcast):** Casting da superclasse para a subclasse (downcast) de forma explícita.
- Lembre-se: Casting em Tipos Objeto não altera o Objeto em si, este continua sendo o mesmo, apenas cria uma nova variável que vai referenciá-lo.
- O downcast deve ser feito com **cuidado** para evitar exceções em tempo de execução.
- Recomenda-se sempre fazer a verificação do tipo antes.

Casting em Objetos - Downcasting

```
public static void exibirLivros(List<Livro> livros) {  
    for (Livro livro : livros) {
```

```
        System.out.println(livro.toString());
```

```
        System.out.println(livro.toString());  
        System.out.println(livro.toString());  
        // Exception in thread "main" java.lang.ClassCastException:  
        //     class LivroDigital cannot be cast to class LivroFisico  
        //     (LivroDigital and LivroFisico are in unnamed module of l  
        // oader 'app')  
        //         at Main.exibirLivros(Main.java:40)  
        //         at Main.main(Main.java:25)
```

```
        // Então vou fazer o downcasting para acessar atributos específicos
```

```
        LivroFisico livroFisico = (LivroFisico) livro;
```

```
        System.out.println("Número de exemplares: "  
            + livroFisico.getNumeroExemplares());
```

Casting em Objetos - Downcasting

```
// Se for necessário acessar atributos específicos,  
// é necessário fazer downcasting, mas isso deve ser feito com cuidado  
if (livro instanceof LivroFisico) { // Verifica se é um LivroFisico  
    // Downcasting para LivroFisico - Cria uma variável temporária  
    // do tipo LivroFisico para acessar atributos específicos  
    LivroFisico livroFisicoTemp = (LivroFisico) livro;  
    System.out.println("Número de exemplares: "  
        + livroFisicoTemp.getNumeroExemplares());  
    System.out.println("Dimensões: "  
        + livroFisicoTemp.getDimensoes());  
} else if (livro instanceof LivroDigital) {  
    // Downcasting para LivroDigital - Sem criação de variável temporária  
    System.out.println("Tamanho do arquivo: "  
        + ((LivroDigital) livro).getTamanhoArquivo() + " MB");  
}
```

Pacotes – Organizando a Galáxia


Java: Entendendo Packages

Do caos rebelde à ordem imperial com package e import 🚀



O que é um Package?



- Um package em Java é uma forma de organizar classes de forma lógica e hierárquica.
- Evita conflitos de nomes
- Facilita a manutenção e organização
- Controla acesso (visibilidade)
- Segue estrutura de diretórios
-  br.com.yodacode.sabres.Luz



Exemplo sem package

- 📌 Problema:
- Tudo no mesmo lugar
- Dificuldade em projetos grandes

```
EXPLORER
CODES
Livro.java
LivroDigital.java
LivroFisico.java
Main.java

Livro.java
1  import java.time.LocalDate;
2
3  public abstract class Livro {
4      private String titulo;
5      private String autor;
6      private int anoPublicacao;
7      private int numeroPaginas;
8
9      public Livro(String titulo, String autor, int anoPublicacao, int numeroPaginas) {
10         this.titulo = titulo;
11         this.autor = autor;
12         this.anoPublicacao = anoPublicacao;
13         this.numeroPaginas = numeroPaginas;
14     }
```



Exemplo com package

- 🧭 Equivale a:
- br/com/yodacode/starwarsapp/StarWarsDay.java


```
br > com > yodacode > starwarsapp > 📄 StarWarsDay.java > ...  
1  package br.com.yodacode.starwarsapp;  
2  
3  ∨ public class StarWarsDay {  
4  ∨      public static void main(String[] args) {  
5      |         System.out.println("May the 4th be with you!");  
6      |     }  
7  }
```

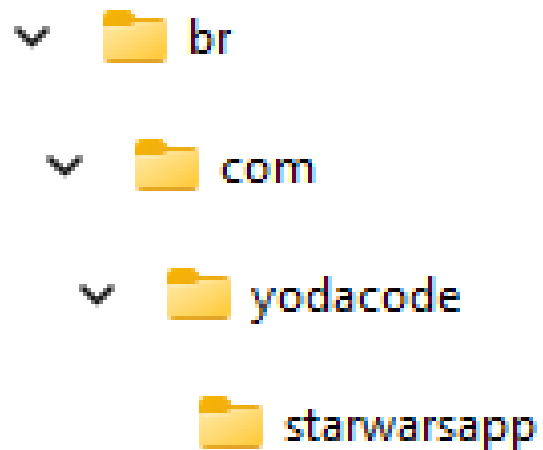



Convenções de nome de package

- Tudo em minúsculas
- Nome inspirado na URL da empresa, invertida
`br.com.yodacode.starwarsapp`
- Sem acentos ou caracteres especiais
- Evite usar nomes genéricos (package1, util, temp...)

Estrutura de pastas

-  Relacionado ao:
- `package br.com.yodacode.starwarsapp;`



Nome	
 StarWarsDay.java	




Importando entre pacotes




- ⚠ *Não se pode importar do pacote default!*

```
1  package br.com.yodacode.starwarsapp;
2
3  import br.com.yodacode.starwarsapp.armas.SabreDeLuz;
4
5  public class StarWarsDay {
6      public static void main(String[] args) {
7          System.out.println("May the 4th be with you!");
8          SabreDeLuz sabreLuke = new SabreDeLuz("azul");
9          sabreLuke.ativar();
10     }
11 }
```

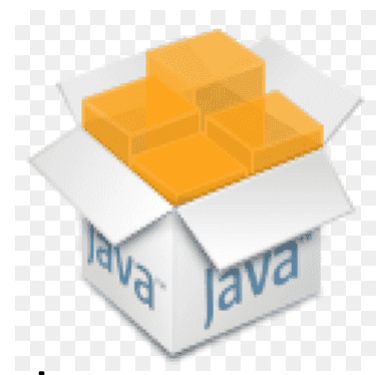
O que não fazer

```
java > util >  MinhaLista.java > ...  
1    package java.util;  
2  
3    ∨ public class MinhaLista {  
4        |    //Não faça isso, perturba a força.  
5    }
```

Frases marcantes (com ajudinha da IA)

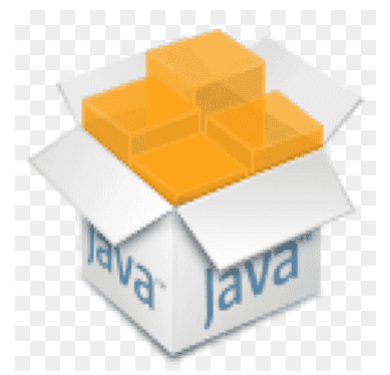
-  "Um projeto Java sem pacotes é como um sabre de luz sem energia."
-  "Mexer no pacote `java.util` é como tentar clonar o Darth Vader: vai dar ruim."
-  "Cada pacote bem organizado é um planeta seguro na galáxia do seu projeto."

Pacotes em Java - Package



- **Java Standard Library:** A plataforma Java possui uma série de pacotes padrão, como **java.util**, **java.io**, **java.lang**, entre outros.
- Você não precisa importar explicitamente o pacote **java.lang**, pois ele é automaticamente disponibilizado para você em seus programas Java.
- No entanto, pacotes como **java.util** e **java.io** precisam ser importados explicitamente.
- **Convenção de Nomenclatura:** Os nomes de pacotes geralmente seguem a convenção de nomenclatura reversa do domínio da empresa para evitar conflitos de nome.
 - Por exemplo, se o site da sua empresa é **minhaempresa.com**, você pode usar **com.minhaempresa** como o início do nome do pacote.

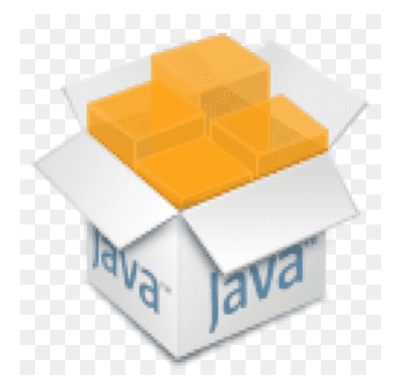
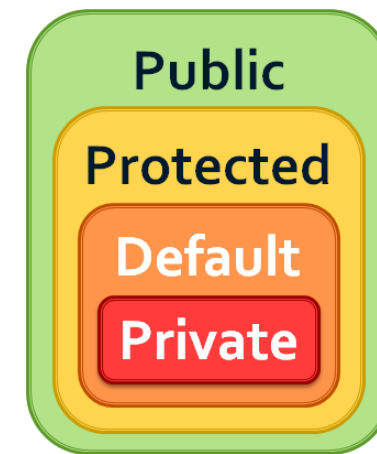
Pacotes em Java - Package



- **Acesso Protegido:** Classes com o modificador de acesso "padrão" (sem public, private ou protected) são acessíveis apenas dentro do mesmo pacote. Isso ajuda a controlar o acesso às classes e manter a encapsulação.

Pacotes em Java - Package

- **Modificadores de Acesso**



Visibilidade	<u>public</u>	<u>protected</u>	default	<u>private</u>
A partir da mesma classe	✓	✓	✓	✓
Qualquer classe no mesmo pacote	✓	✓	✓	✗
Qualquer classe filha no mesmo pacote	✓	✓	✓	✗
Qualquer classe filha em pacote diferente	✓	✓	✗	✗
Qualquer classe em pacote diferente	✓	✗	✗	✗