Herança

Prof. Ms. Peter Jandl Junior
Prof. Ms. Télvio Orru
Prof. Nathan Silva

J12B

Linguagem de Programação Orientada a Objetos

Ciência da Computação - UNIP – Jundiaí

POO::herança & sobreposição

- Os conceitos essenciais da POO - Programação Orientada a Objetos:
  - Sobrecarga e Sobreposição,
  - Herança e
  - Polimorfismo.
- A rigor, o polimorfismo proporciona todos os mecanismos- chave da Orientação a Objetos.

Herança

- Mecanismo que possibilita compartilhamento dinâmico de código.

Sobreposição

- Mecanismo que possibilita substituição de operações em subclasses.

Aplicação

- Como a herança e a sobrecarga podem ser explorados.

polimorfismo

Do grego ***poli morfos***, ou seja, muitas formas.

Característica da orientação a objetos que admite múltiplas formas de suas construções.

Na orientação a objetos o polimorfismo se manifesta na sobrecarga, na herança, na sobreposição e na generalização (tratamento polimórfico).

POO::herança

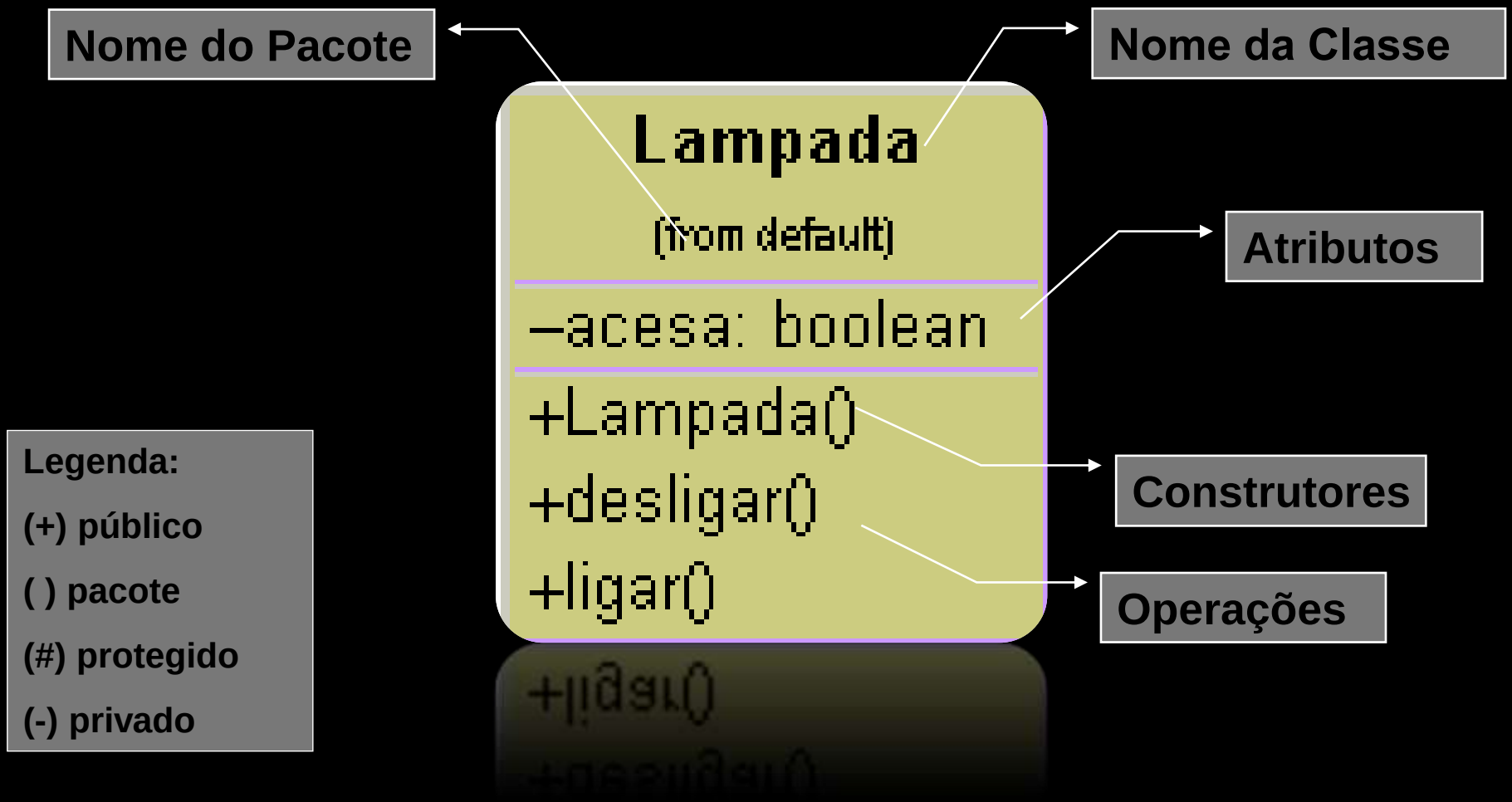
Herança

- Segunda característica mais importante da Orientação a Objetos
- Significa a construção de tipos de dados (classes) baseada em outros tipos já existentes, onde:
  - as características ancestrais são compartilhadas pelos descendentes e
  - novas características são adicionadas nestes.
- Possibilita a ***especialização*** das classes.

Herança::situações-exemplo

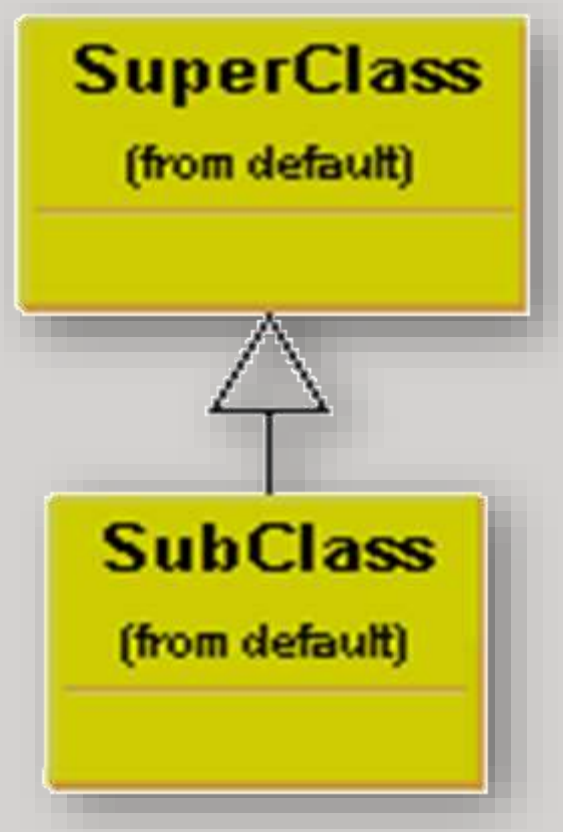
- Uma bola de futebol ***é uma*** bola que por sua vez ***é uma*** esfera.
- Um gerente administrativo ***é um*** administrador que ***é um*** funcionário.
- Um pardal ***é uma*** ave que ***é um*** animal vertebrado.
- Um Fiat Uno ***é um*** Fiat que ***é um*** automóvel que ***é um*** veículo.

Notação UML (*Unified Modeling Language*)



Herança

- A classe de origem é chamada de **classe base** ou **classe-pai** ou **superclasse**.
- A nova classe criada é chamada de **classe derivada** ou **classe-filha** ou **subclasse**.



Herança

- São termos complementares:
  classe-pai e *classe-filha*
  ou
  classe-base e *classe-derivada*.
- Em Java é comum o uso dos termos:
  superclasse e *subclasse*.
- A palavra reservada ***extends*** permite expressar relações de herança no Java. (No C# o : é usado para este fim.)

Herança

```
// SuperClass.java
public class SuperClass {
   :
}
// SubClass.java
public class SubClass extends SuperClass {
   :
}
```

Herança

- Por que usamos?

  Porque simplifica e flexibiliza o projeto do software.

- Para que usamos?

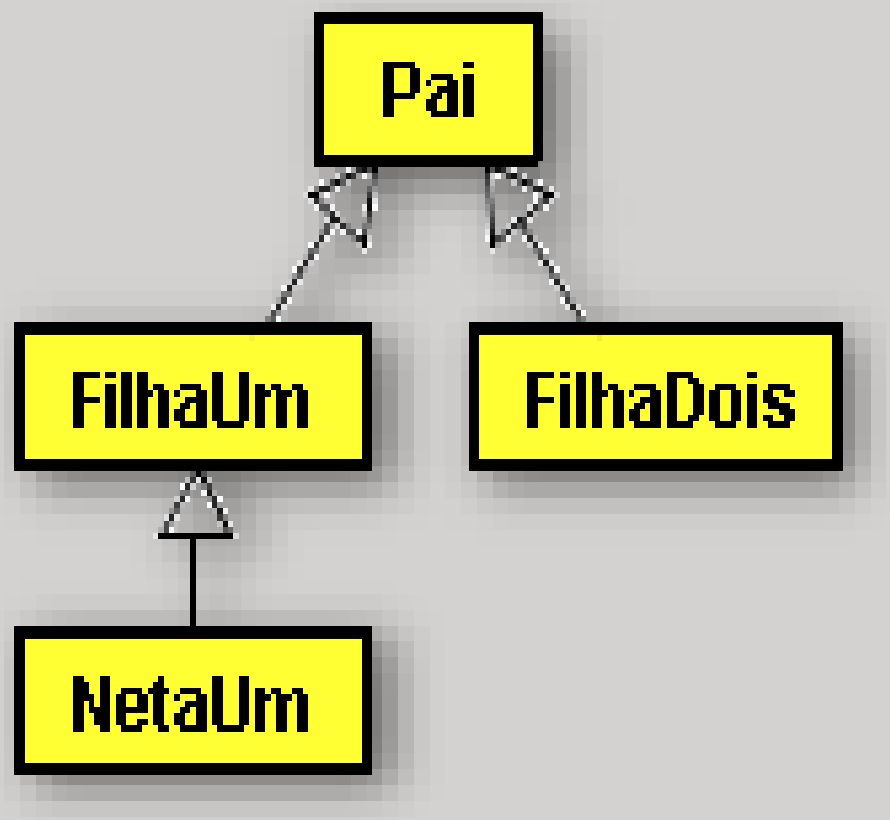
  Para permitir reusabilidade do código e do projeto

- Quando usamos?

  Quando percebemos a existência de uma relação "é um" (*is a*)

Herança::Construção de Hierarquias

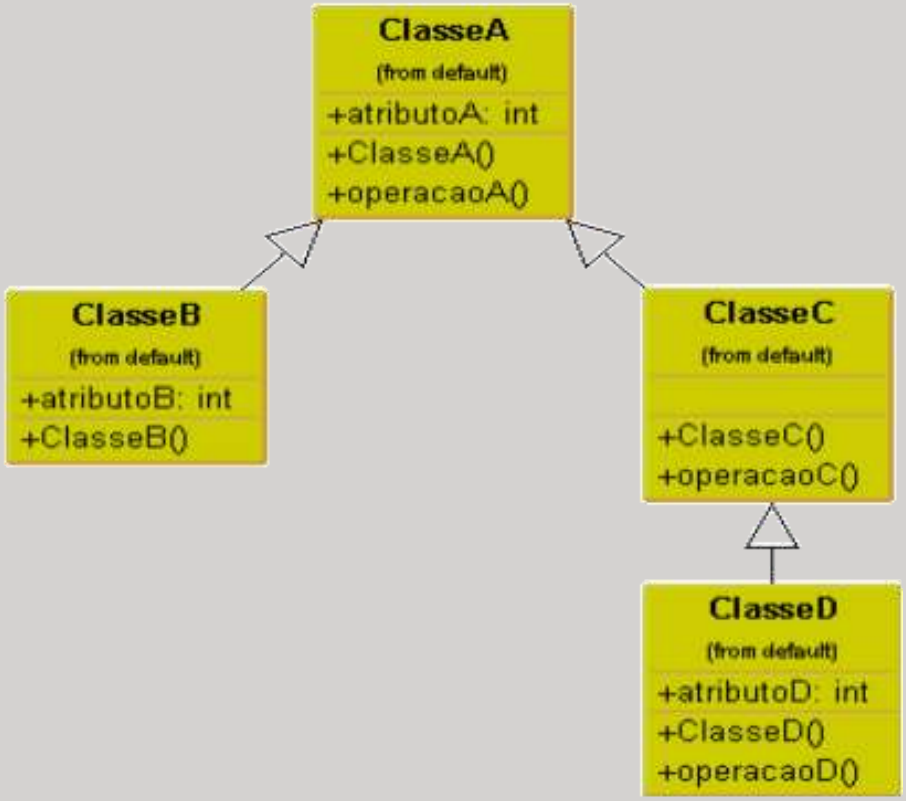
```
// Pai.java
public class Pai {
  :
}
// FilhaUm.java
public class FilhaUm
            extends Pai {
  :
}
// NetaUm.java
public class NetaUm
            extends FilhaUm {
  :
}
```



Herança::Hierarquias de classes

- Hierarquias de classes são tipicamente representadas por árvores de classes onde:
  - raiz é posicionada acima, folhas para baixo (árvore invertida);
  - elementos mais próximos da raiz são ditos de hierarquia mais alta;
  - elementos mais próximos da folhas são ditos de hierarquia mais baixa;
  - o grau hierárquico se refere, portanto, ao número de filhos (classes derivadas) possíveis;
  - o grau hierárquico também se refere as possibilidades de generalização existentes (polimorfismo).

Herança:Hierarquias de classes



- Uma classe pode originar uma hierarquia (ou família) de classes.
- Atributos e operações **públicas** e **protegidas** são compartilhadas entre os descendentes de cada classe.
- Atributos e operações **privadas** são exclusivos de cada classe.
- Os construtores, a despeito de sua acessibilidade, *nunca são compartilhados*.

Herança:construção de hierarquias

ClasseA.java

```
public class ClasseA {

   public int atributoA;


   public ClasseA ( ) { }


   public int operacaoA ( ) { }

}
```

ClasseB.java

```
public class ClasseB
   extends ClasseA {

   public int atributoB;


   public ClasseB ( ) { }

}
```

Herança:construção de hierarquias

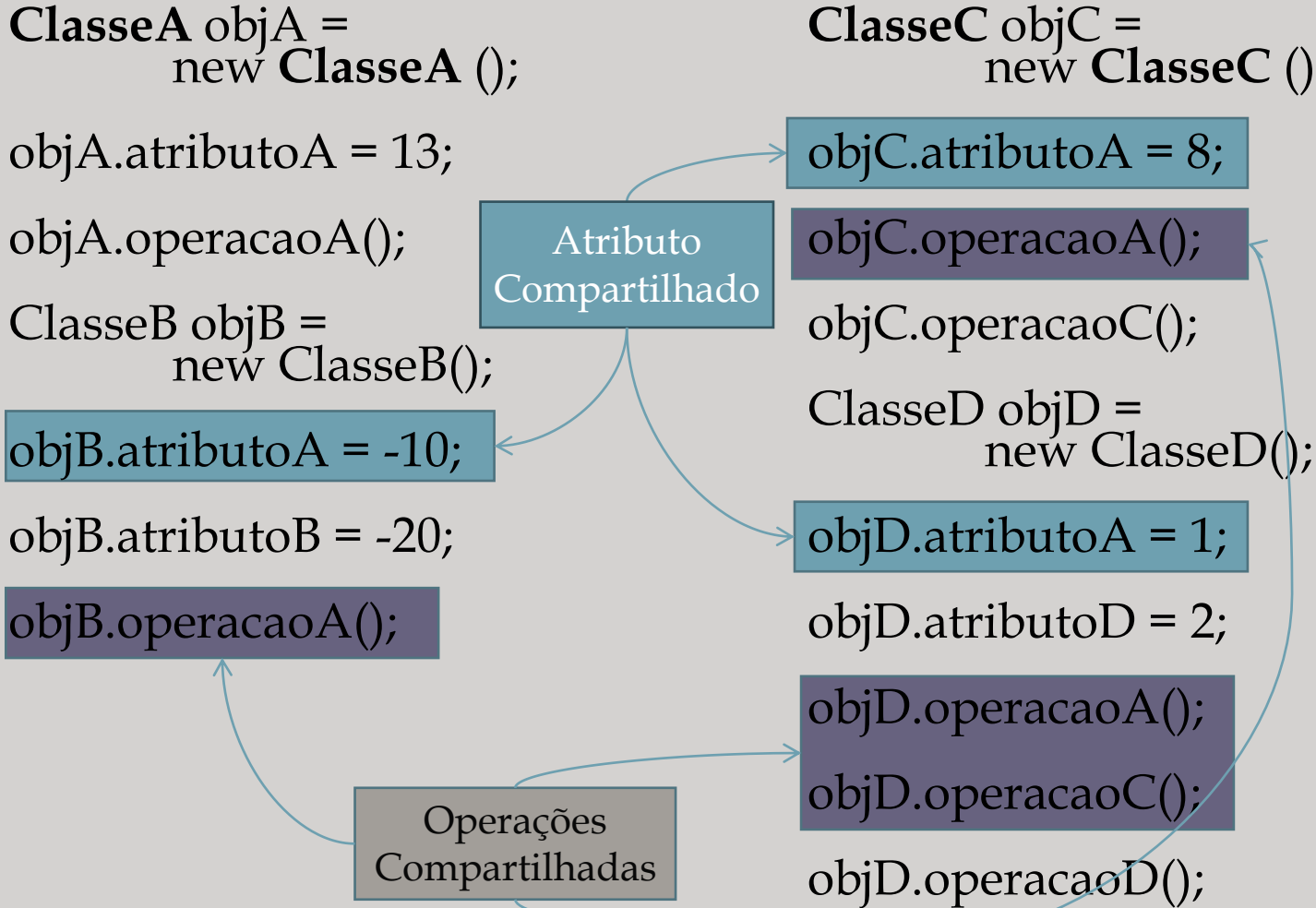
ClasseC.java

```
public class ClasseC
   extends ClasseA {

   public ClasseC ( ) { }

   public int operacaoC ( ) { }
}
```

ClasseD.java

```
public class ClasseD
   extends ClasseC {

   public int atributoD;

   public ClasseD ( ) { }

   public int operacaoD ( ) { }
}
```

Herança::uso desta hierarquia::possibilidades



Herança:acessibilidade de membros

	Acessibilidade de membros			
Especificador	Implementação Superclasse	Instâncias Superclasse	Implementação SubClasse	Instâncias SubClasse
private	sim	não	não	não
protected	sim	não	sim	não
public	sim	sim	sim	sim

Membros protegidos constituem uma espécie de herança de programador, isto é, só está disponível para *implementações* de subclasses.

Herança:aplicações

Extensão

- A Herança permite a criação de novas subclasses que ampliam as operações e atributos existentes na superclasse.

Contração

- A Herança também permite que as funcionalidades da superclasse sejam alteradas ou restritas em subclasses.

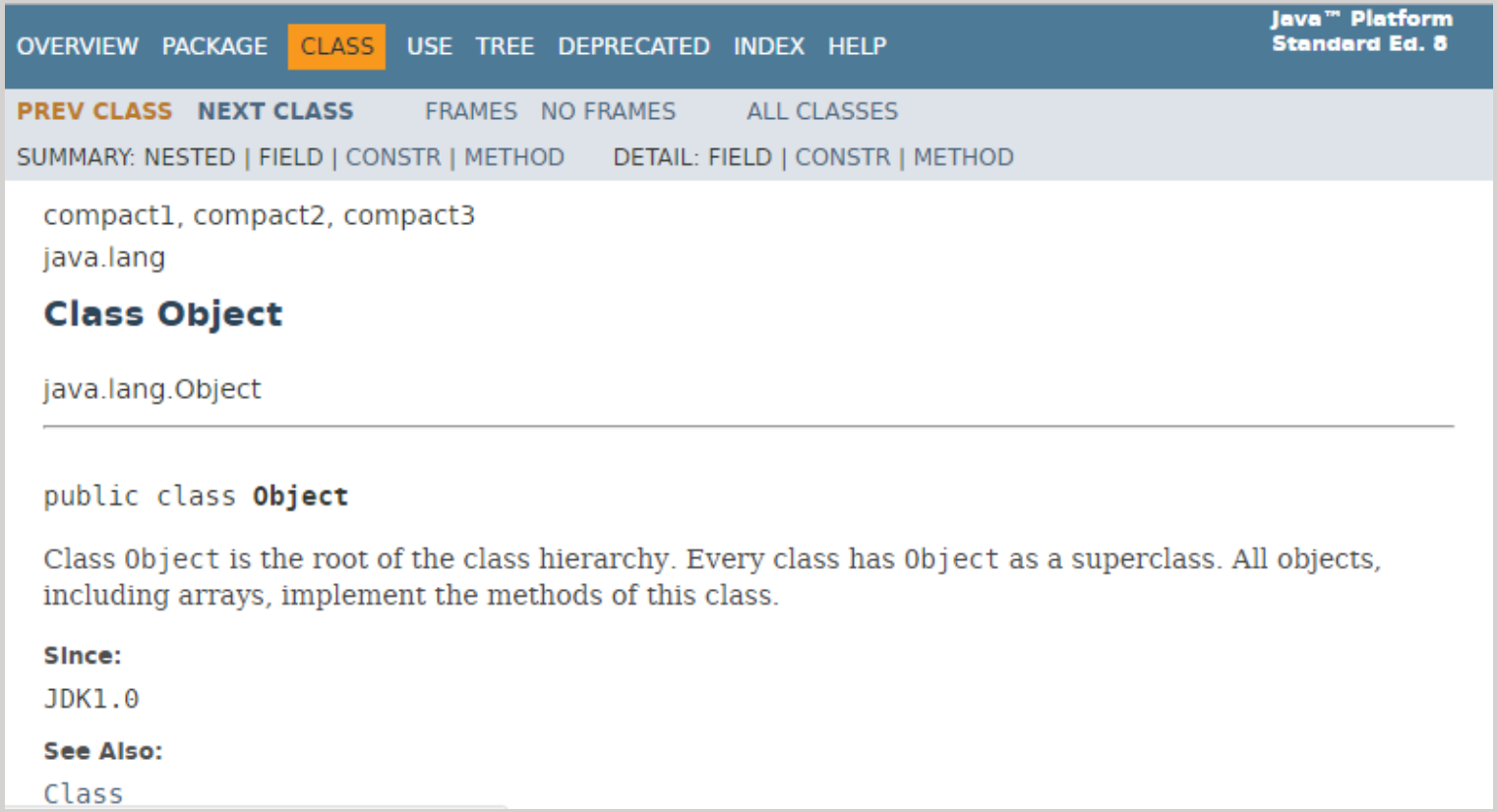
A **Herança** é um mecanismo de *especialização*, ou seja, permite a construção de novas classes derivadas que possuem características especiais em relação a classe base.

A classe java.lang.Object

Herança::raiz da hierarquia Java

- No caso específico do Java:
  - A classe **Object**, do pacote **java.lang**, é a classe raiz de toda hierarquia de objetos.
  - O que significa que TODAS as demais classes são derivadas direta ou indiretamente da classe **Object**.
  - Isto assegura que os mecanismos estruturais de alocação de memória e sincronização sejam disponibilizados para todas as demais classes, além de um conjunto de operações comuns.

Classe java.lang.Object



OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

Class Object

java.lang.Object

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

`JDK1.0`

See Also:

`Class`

Herança explícita e implícita

Explícita

- Indicada diretamente no código:

```
public class Contrato
        extends Object {

}
```

Efeito produzido é exatamente o mesmo!

Implícita

- Não indicada no código:

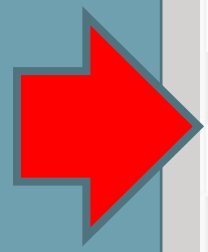
```
public class Contrato
                {

}
```

Quando não indicada a superclasse, ocorre herança implícita de **Object**!

java.lang.Object

Observe a infraestrutura oferecida para controle de:

- Execução (threads);
- Memória e
- Identidade do objeto.



Constructor Summary

Constructors

Constructor and Description
`Object()`

Method Summary

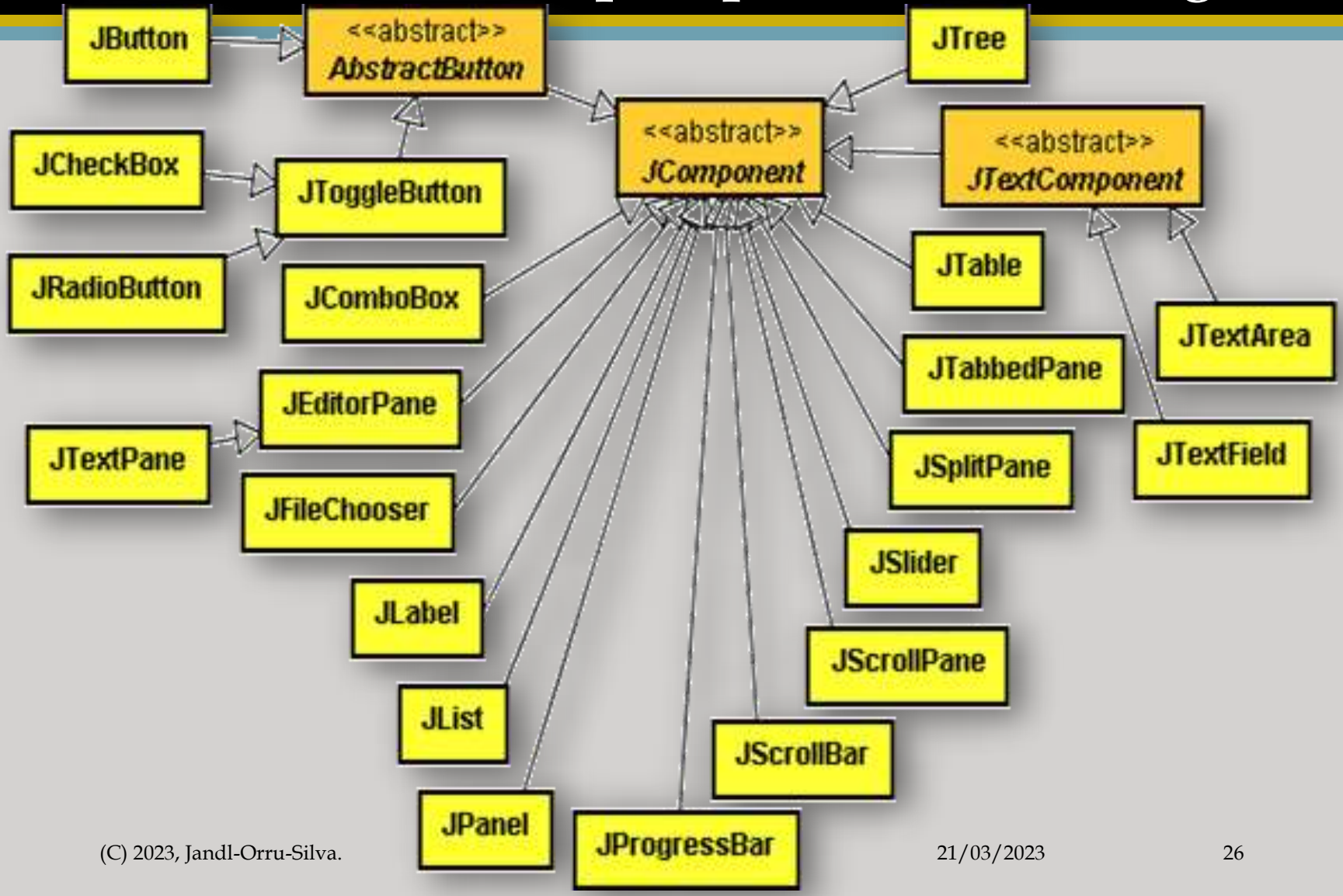
All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
protected Object	**clone**() Creates and returns a copy of this object.
boolean	**equals**(Object obj) Indicates whether some other object is "equal to" this one.
protected void	**finalize**() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	**getClass**() Returns the runtime class of this Object.
int	**hashCode**() Returns a hash code value for the object.
void	**notify**() Wakes up a single thread that is waiting on this object's monitor.
void	**notifyAll**() Wakes up all threads that are waiting on this object's monitor.
String	**toString**() Returns a string representation of the object.
void	**wait**() Causes the current thread to wait until another thread invokes the **notify()** method or the **notifyAll()** method for this object.
void	**wait**(long timeout) Causes the current thread to wait until either another thread invokes the **notify()** method or the **notifyAll()** method for this object, or a specified amount of time has elapsed.
void	**wait**(long timeout, int nanos) Causes the current thread to wait until another thread invokes the **notify()** method or the **notifyAll()** method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Uso de herança na API Java

- Herança é usada extensivamente na API do Java, como visto nos componentes Swing, nas coleções e nos *streams*:
  - java.lang.**Object**
    → java.awt.**Component**
    →→ java.awt.**Conteiner**
    →→→ javax.swing.**JComponent**
  - java.util.**Collection**
    → java.util.**Queue**
    →→ java.util.**Deque**
  - java.io.**Reader**
    → java.io.**InputStreamReader**
    →→ java.io.**FileReader**

Herança::hierarquia parcial do Swing



Herança::organização de hierarquias

- A construção de hierarquias envolve três etapas:

1. Identificação das classes hierarquicamente relacionadas (relação "is a/é um").
2. Fatoração dos elementos comuns nos níveis mais altos possíveis da hierarquia.
3. Inclusão das especificidades (atributos e operações próprios) nos níveis mais baixos da hierarquia.

Exercícios Propostos
&
Recomendações de Estudo

Exercícios Propostos

1. Construa uma classe que possa representar um produto genérico de um supermercado, incluindo elementos que possam existir em qualquer tipo de produto, como:
   - Código, Descrição, Fabricante e Preço Unitário.
2. Crie duas para representar categorias mais específicas de produtos, nas quais atributos (e operações) apropriadas possam ser incluídos
   - Produto Perecível → data de fabricação, prazo de validade
   - Produto Não Perecível → prazo de garantia

Recomendações de Estudo

- **Java – Guia do Programador**,
  3ª Edição,
  P. JANDL Jr,
  Novatec, 2015.
- **Java 6- Guia de Consulta Rápida**,
  P. JANDL Jr,
  Novatec, 2008.
- **Java 5- Guia de Consulta Rápida**,
  P. JANDL Jr,
  Novatec, 2006.
- **Introdução ao Java**,
  P. JANDL Jr,
  Berkeley, 2002.