Programação Orientada a Objetos

Crítica à Herança

André Santanchè
Instituto de Computação – UNICAMP
Abril 2015

Picture by http://www.flickr.com/photos/fdecomite/1457493536/

Dependency Inversion Principle (DIP)

- “Depender das Abstrações. Não depender das Concretizações.” (Martin, 2000)

Clientes por Herança X Clientes por Instanciação

Herança introduz dois tipos de clientes de uma classe:

- **Clientes por Instanciação:** os usuários criam instâncias da classe e manipulam-as através de sua interface pública. Esses clientes são objetos.
- **Clientes por Herança:** os usuários são as próprias subclasses que herdam os métodos e atributos da classe base. Esses clientes são classes.

(Rubira, 2011)

Usos de Herança

Usos de Herança

O mecanismo de herança permite a construção de duas categorias de hierarquias:

- Hierarquias de implementação
- Hierarquias de comportamento

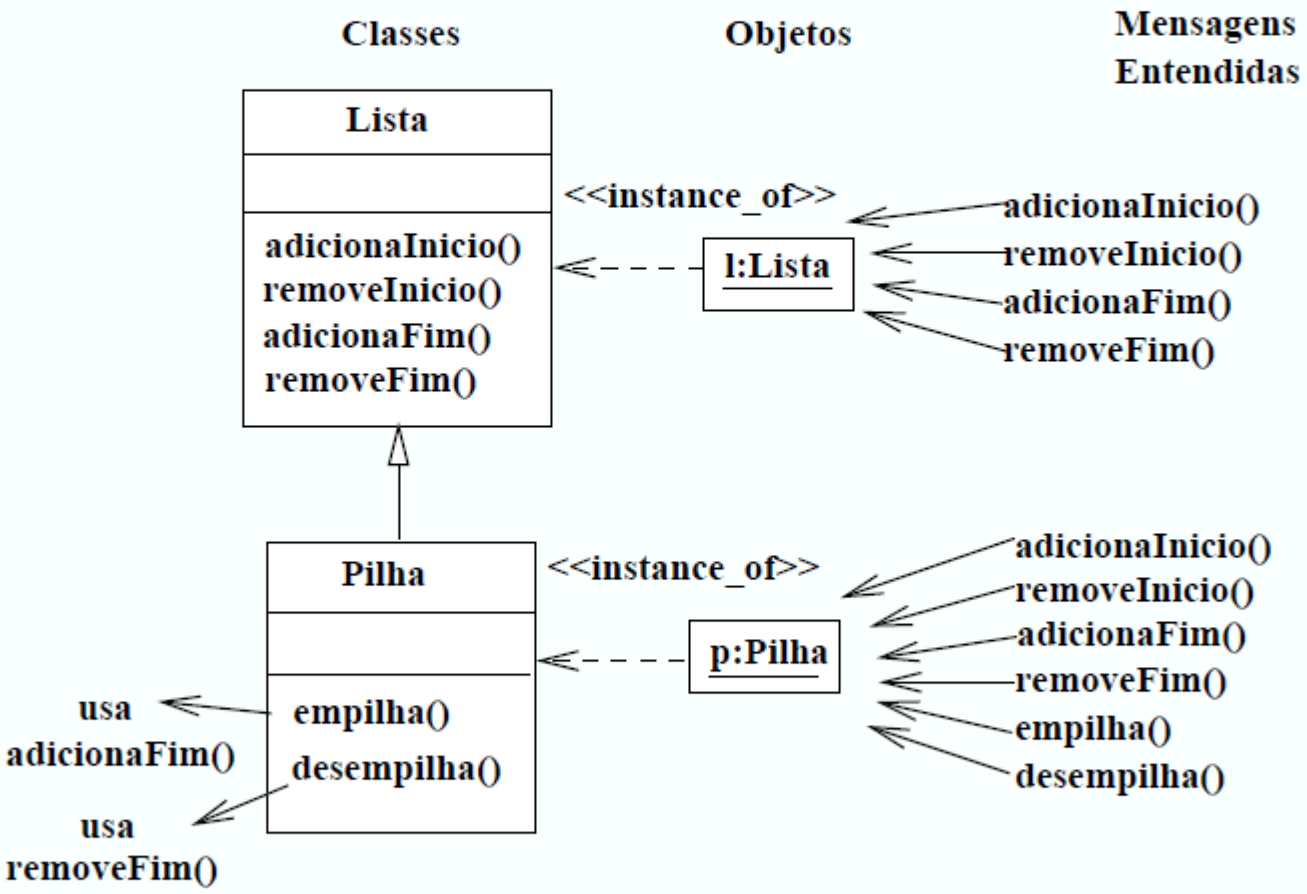
(Rubira, 2011)

Hierarquias de Implementação

- Herança é usada como uma técnica para implementar TADs que são similares a outros já existentes (reutilização de código)
- Nesse caso, o programador usa herança como uma técnica de implementação, com nenhuma intenção de garantir que a subclasse tenha o mesmo comportamento da superclasse
- Você pode herdar comportamento não desejado, implicando num comportamento INCORRETO da subclasse

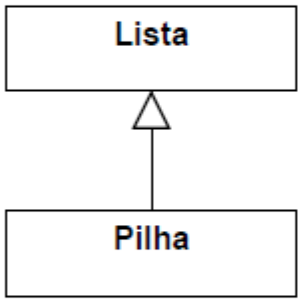
(Rubira, 2011)

Exemplo

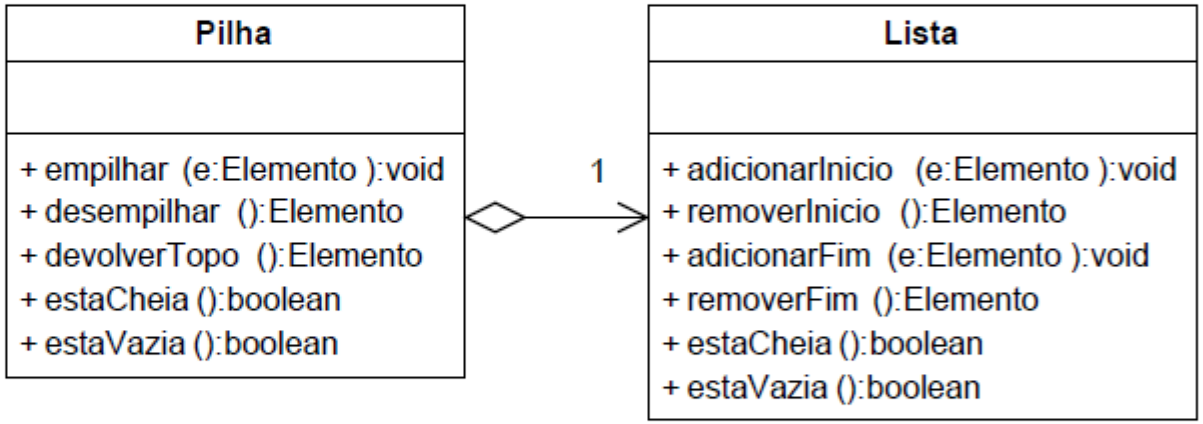


(Rubira, 2011)

Solução Recomendada em Java (I)



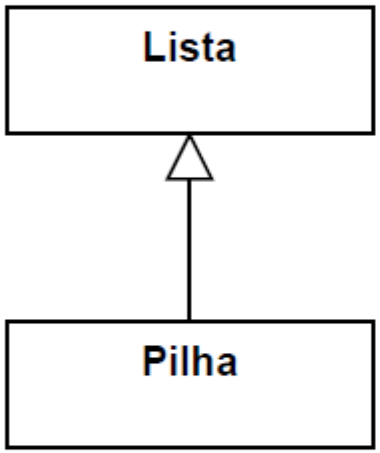
NÃO RECOMENDADO



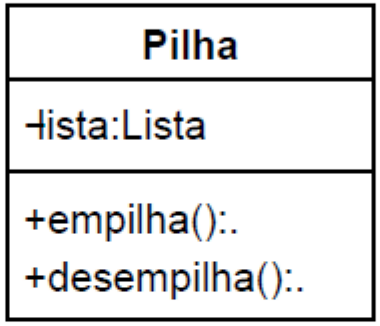
RECOMENDADO

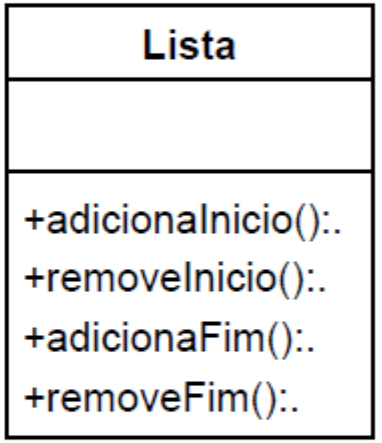
(Rubira, 2011)

Solução Recomendada em Java (II)



NÃO RECOMENDADO

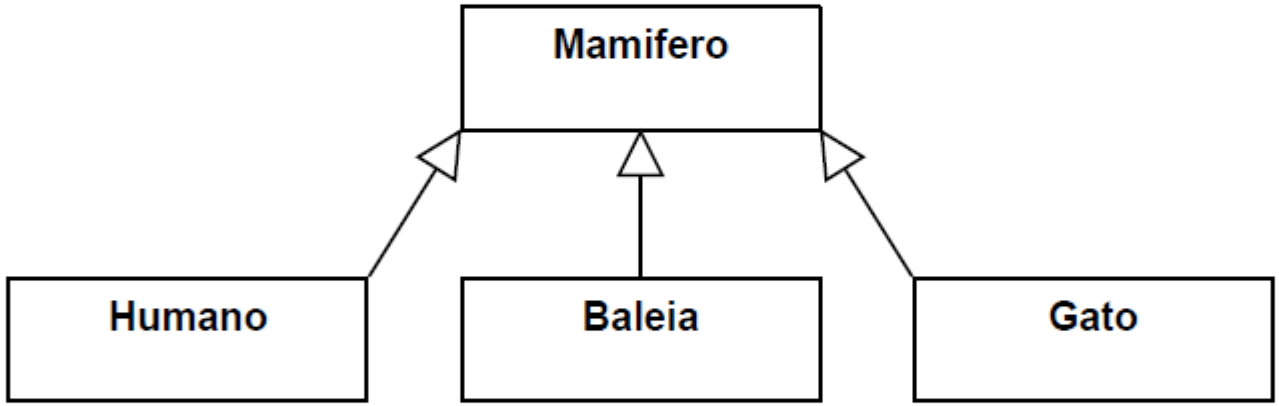




RECOMENDADO

(Rubira, 2011)

Herança de Comportamento (I)



(Rubira, 2011)

Herança de Comportamento (II)

- Herança de comportamento representa uma hierarquia verdadeira de generalização/especialização
- Herança de comportamento equivale ao relacionamento **é-um**, ou **é-subtipo-de**
- No exemplo, podemos dizer que o tipo `Humano` é um tipo especializado de `Mamífero` ou que o tipo `Humano` é um subtipo de `Mamífero`

(Rubira, 2011)

Liskov Substitution Principle (LSP)

- Associado à noção de Tipo Abstrato de Dados – Abstract Data Type (ADT)
- Foi enunciado por Barbara Liskov
- Baseado na noção de subtipo:
  - Dado que um programa P que faz uso de um objeto O1; O2 será subtipo de O1 se for possível substituir O1 por O2 no programa P, sem que P altere seu comportamento (Liskov, 1987).
- Em OO: noção de subclasse equivale a noção de subtipo

O Conceito de Subtipo (I)

- Subtipo preocupa-se com o **compartilhamento de comportamento** ("behaviour sharing")
- Então, se S é um subtipo de T, um objeto do tipo S pode ser usado no lugar de um objeto de tipo T
- A idéia principal é que classes derivadas "comportem-se como se fossem as classes bases"

(Rubira, 2011)

O Conceito de Subtipo (II)

- Dizemos que o subtipo S se "conforma" ao tipo "T".
- No modelo de objetos, o compartilhamento de comportamento é somente justificável quando existe um relacionamento verdadeiro de generalização/especialização entre as classes.
- No caso, hierarquia de generalização/especialização definida por Lista e Pilha não é verdadeira, pois existem operações de Lista que invalidam o comportamento da classe Pilha.
- Portanto, você não deve usar herança. Use o relacionamento de agregação no lugar.

(Rubira, 2011)

Referências

- Martin, R. C. **Design Principles and Design Patterns.** Object Mentor, 2000.
- Rubira, Cecília Mary Fischer (2011). **Introdução à Programação Orientada a Objetos Usando Java.** Slides de aula, IC – Unicamp.

André Santanchè
http://www.ic.unicamp.br/~santanche

License

- These slides are shared under a Creative Commons License. Under the following conditions: Attribution, Noncommercial and Share Alike.
- See further details about this Creative Commons license at: http://creativecommons.org/licenses/by-nc-sa/3.0/