Programação Orientada a Objetos

Herança

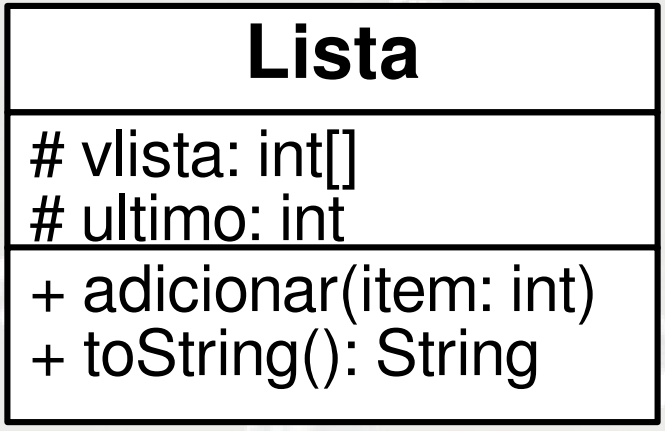
André Santanchè
Laboratory of Information Systems – LIS
Instituto de Computação – UNICAMP
Abril 2020

Herança

- Capacidade das classes expandirem-se a partir das classes existentes.
- Classe herdeira (subclasse)
  - possui os mesmos atributos da superclasse
  - herda acesso aos métodos desta superclasse
  - pode acrescentar novos atributos e métodos (extensão)

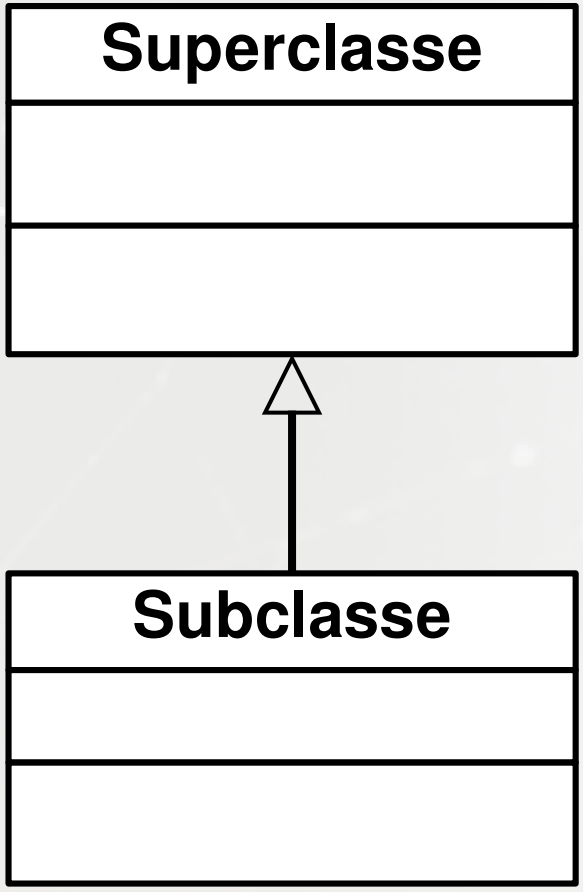
`Lista` em UML

- Classe `Lista` que gerencia uma lista de números simples:



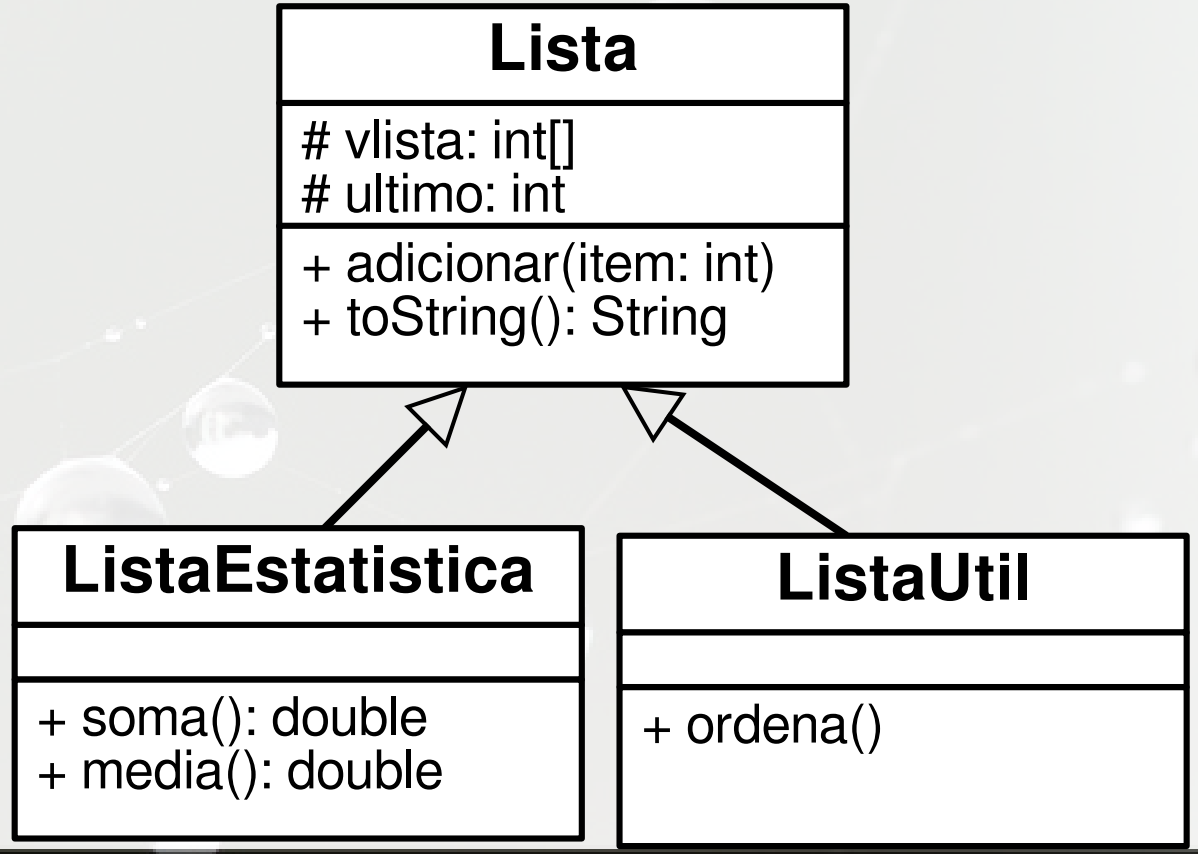
Herança em UML

- Subclasse (herdeira) aponta para superclasse
- Triângulo vazado indicando a herança
- Subclasse só precisa declarar métodos que acrescenta



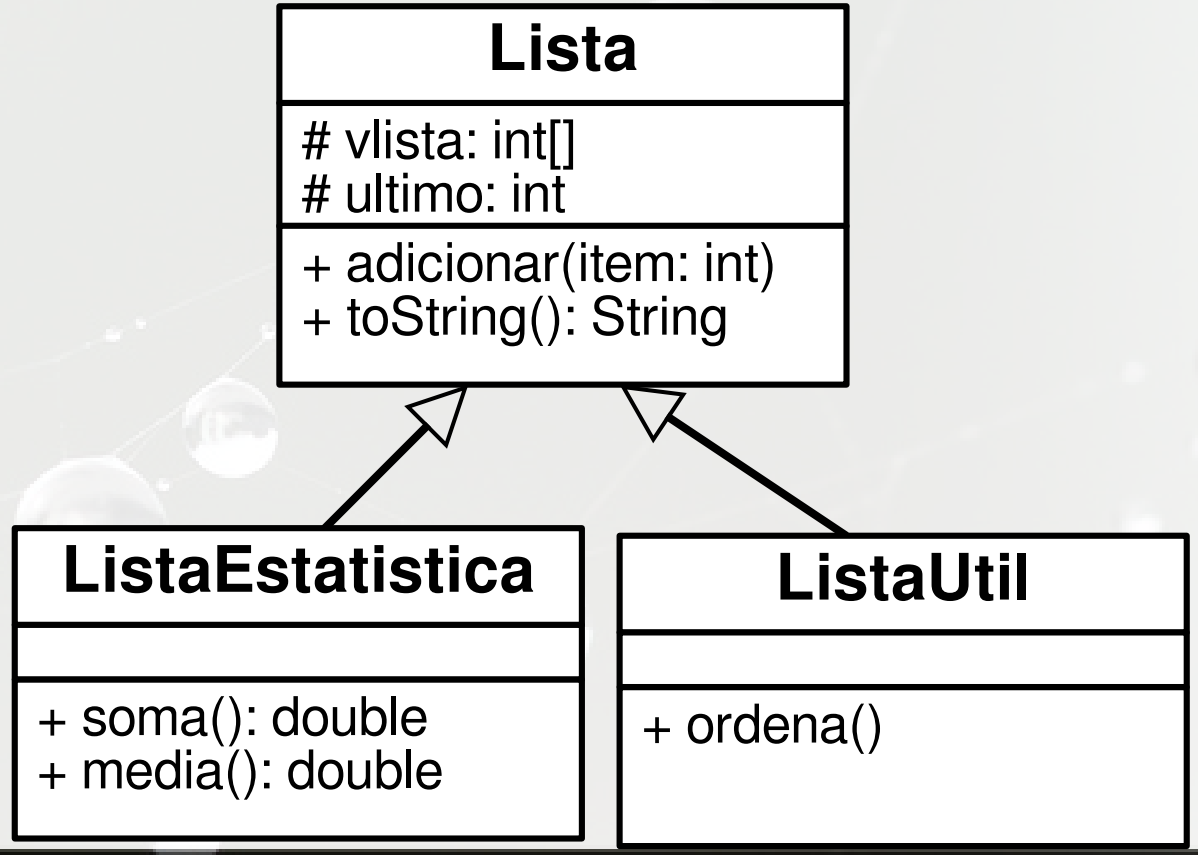
Herdeiros da classe `Lista` em UML

- Classes `ListaEstatistica` e `ListaUtil` herdeiras de `Lista`
- Estendem a classe `Lista` acrescentando métodos



Encapsulamento em UML

- Visibilidade:
  - + público
  - - privado
  - # protegido



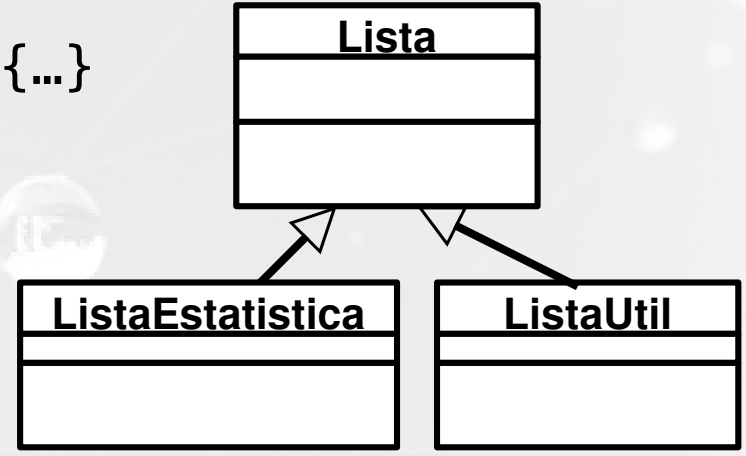
Encapsulamento
Níveis de Acesso

- **Privada**: não visível por classe/objetos externos; visível apenas dentro da classe ou objetos da classe onde é definido.
- **Pública**: completamente visível para qualquer classe/objeto interno ou externo.
- **Protegida**: visível apenas dentro da classe/objetos da classe e para seus herdeiros; não visível a classes/objetos externos.

Herança em Java
extends

- Para se estabelecer que uma classe é herdeira de outra, após o nome da classe coloca-se a cláusula `extends` e o nome da superclasse:

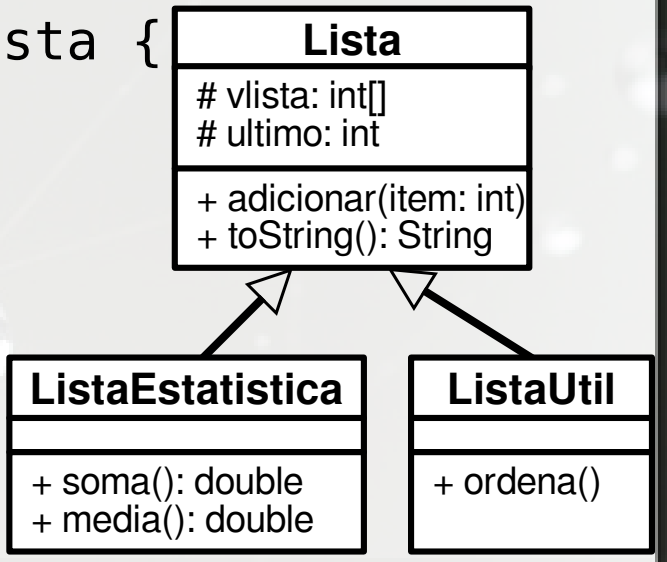
```
public class Lista {…}

public class ListaEstatistica extends Lista {…}

public class ListaUtil extends Lista {…}
```



Herança em Java

- Para se estabelecer que uma classe é herdeira de outra, após o nome da classe coloca-se a cláusula `extends` e o nome da superclasse:

```
public class Lista {…}
public class ListaEstatistica extends Lista {
    public double soma() {…}
    public double media() {…}
}
public class ListaUtil extends Lista {
    public void ordena() {…}
}
```



Usando as Classes Herdeiras

- Métodos herdados são ativados na superclasse de forma transparente, como se pertencessem à subclasse

```
ListaEstatistica le = new ListaEstatistica();
le.adicionar(10);
le.adicionar(20);
le.adicionar(5);
le.adicionar(35);
le.adicionar(15);
System.out.println("Soma: " + le.soma());
System.out.println("Media: " + le.media());
```

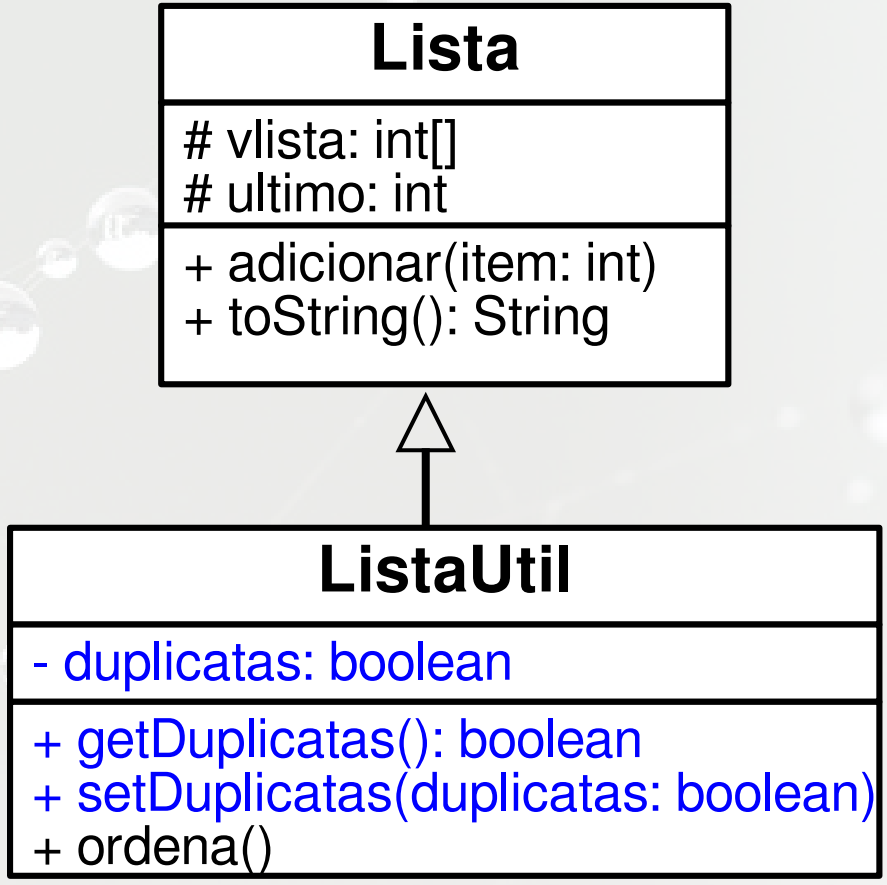
Usando as Classes Herdeiras

- Métodos herdados são ativados na superclasse de forma transparente, como se pertencessem à subclasse

```
ListaUtil lu = new ListaUtil();
lu.adicionar(10);
lu.adicionar(20);
lu.adicionar(5);
lu.adicionar(35);
lu.adicionar(15);
lu.ordena();
System.out.println("Ordenado: " + lu);
```

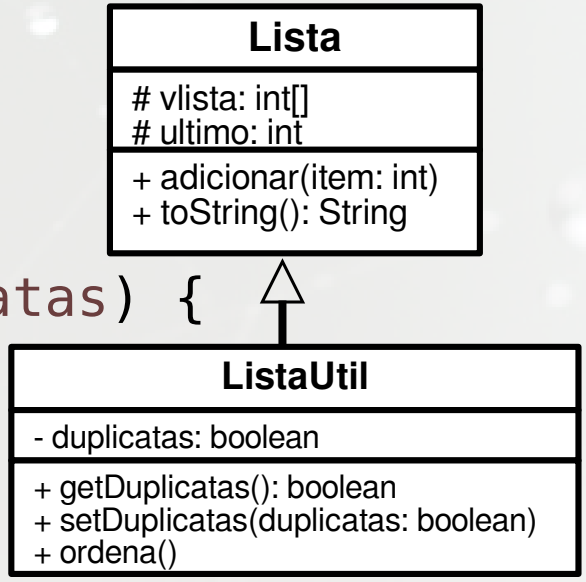
Acrescentando Atributos

- Classe ListaUtil acrescentando um atributo (`duplicatas`)
- Também foram acrescentados os respectivos métodos `get` e `set` para transformá-lo em uma propriedade



Acrescentando Atributos

```
public class ListaUtil extends Lista {
   private boolean duplicatas = true;

   public boolean getDuplicatas() {
      return duplicatas;
   }

   public void setDuplicatas(boolean duplicatas) {
      this.duplicatas = duplicatas;
   }

   public void ordena() {…}
```

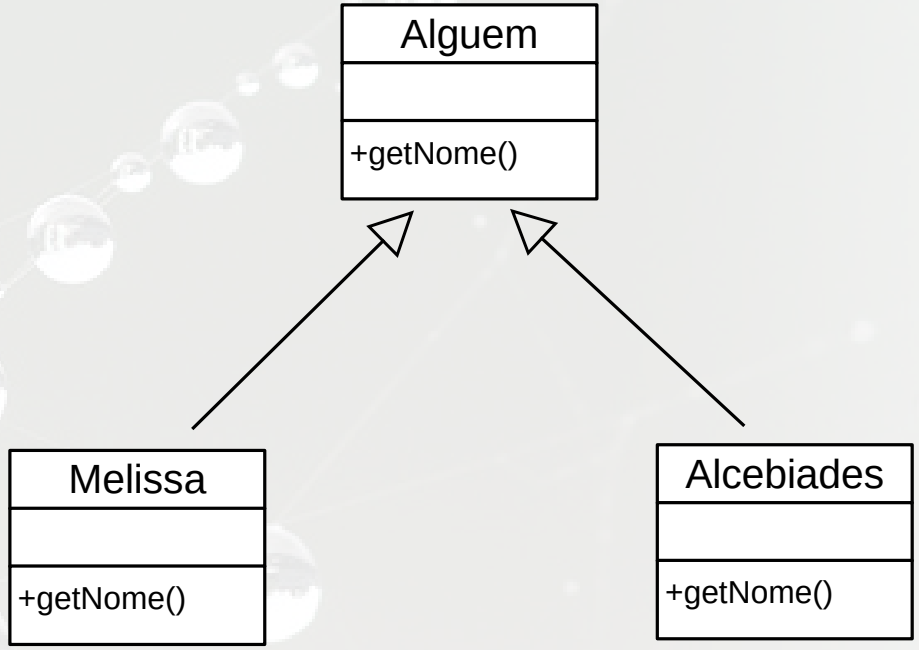


Sobrescrevendo Métodos

- Método na subclasse sobrescreve o da superclasse quando tem a mesma assinatura
  - para um objeto da subclasse, o método da subclasse (que sobrescreve) será chamado no lugar daquele da superclasse
  - não afeta objetos declarados na superclasse
- Assinatura do método
  - composta do nome do método mais os tipos dos argumentos, considerando a ordem

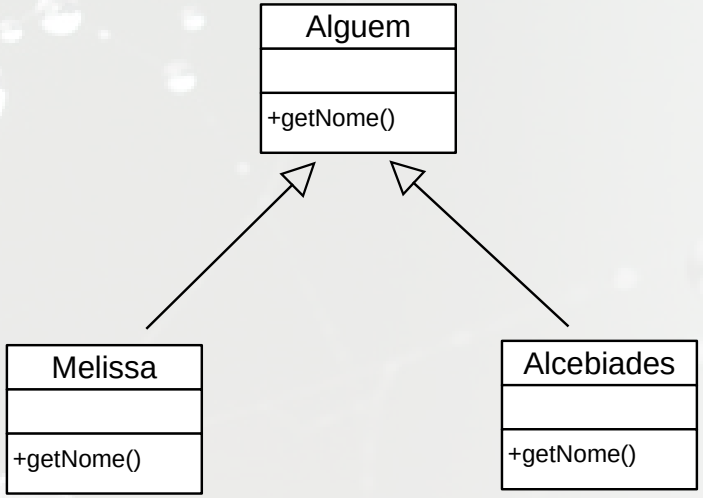
Tarefa
Sobrescrita de Métodos

- Considera a hierarquia de classes
  - `Melissa` e `Alcebiades` sobrescrevem o método `getNome` de `Alguem`



Tarefa
Sobrescrita de Métodos

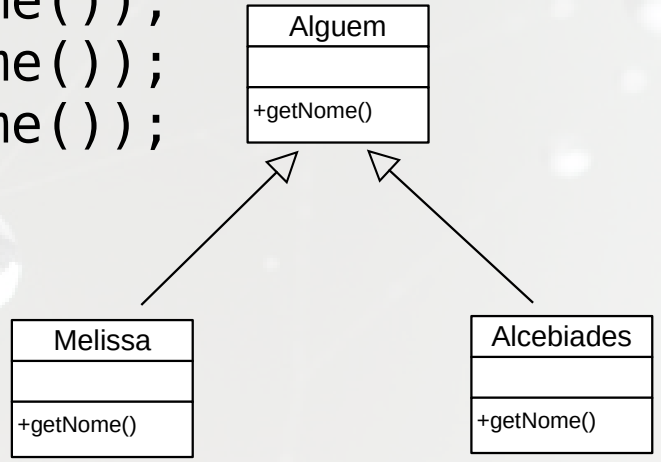
```
public class Alguem {
    public String getNome() {
       return "alguem (genericamente)";
    }
}
public class Alcebiades extends Alguem {
    public String getNome() {
        return "Alcebiades";
    }
}
public class Melissa extends Alguem {
    public String getNome() {
        return "Melissa";
    }
}
```



Tarefa
Sobrescrita de Métodos

- O que aparecerá nesta sequência de instruções?

```
Alguem a = new Alguem();
Alcebiades b = new Alcebiades();
Melissa c = new Melissa();

System.out.println("Nome a: " + a.getNome());
System.out.println("Nome b: " + b.getNome());
System.out.println("Nome c: " + c.getNome());
```



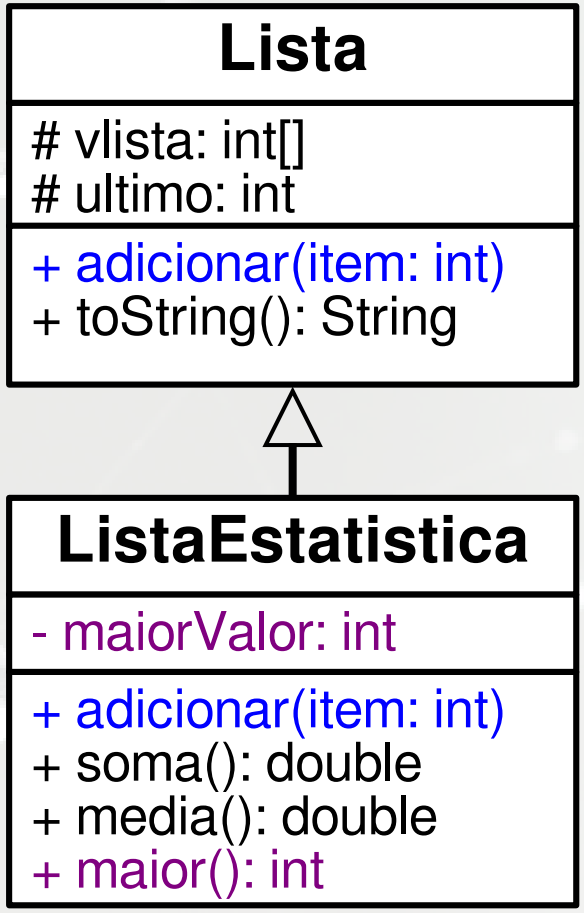
Referência à Superclasse
`super`

- Referência explícita à superclasse
- Só é necessária quando há sobrescrita de método
- Uso comum: método da classe herdeira sobrescreve um método da superclasse mas quer estendê-lo
- A cláusula `super` representa a superclasse:

```
super.metodo(...)
```

Estendendo Métodos

- Método `adicionar` de `ListaEstatica` sobrescreve o mesmo método de `Lista`
- Objetivo: estender o método adicionar para guardar maior valor em cada adição

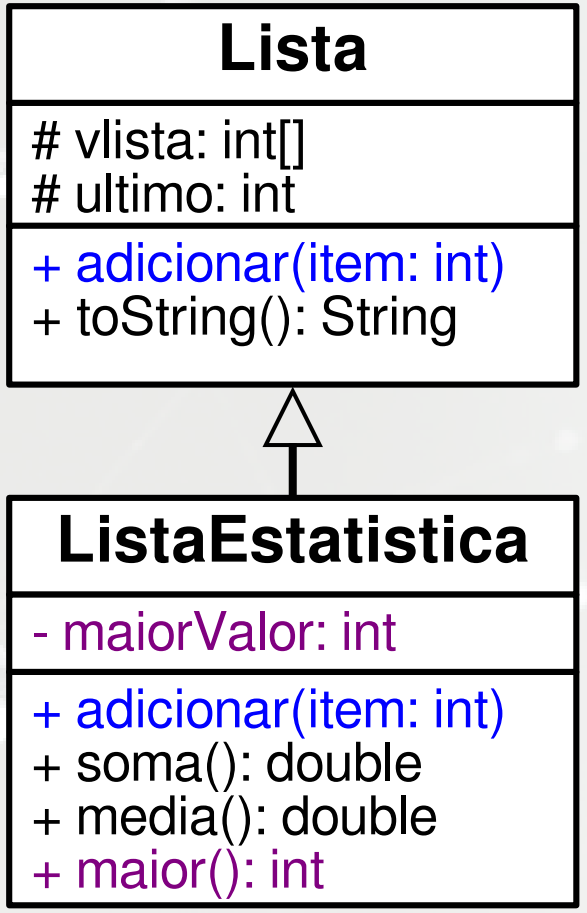


Sobrescrevendo Métodos

- Chamada de método da superclasse com a cláusula `super`:

```
public void adicionar(int item) {
    super.adicionar(item);
    maiorValor = (item > maiorValor)
        ? item : maiorValor;
}
```

- Se não fosse usado o `super` faria uma chamada recursiva



Construtor da Superclasse
`super`

- Chamando o construtor da superclasse:
  - apenas a cláusula `super` na forma de método, com parâmetros (se houver):

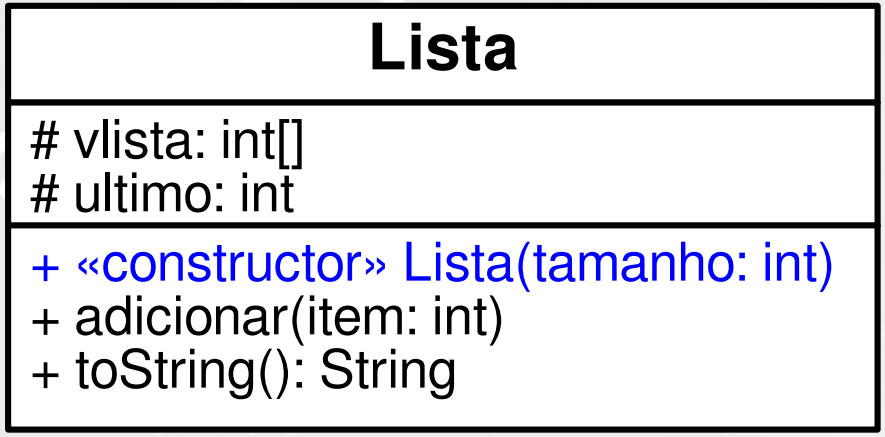
```
super(...)
```

- Só pode ser chamado do construtor da classe
- Se for chamado, tem que ser a primeira instrução do construtor

Construtor para `Lista`

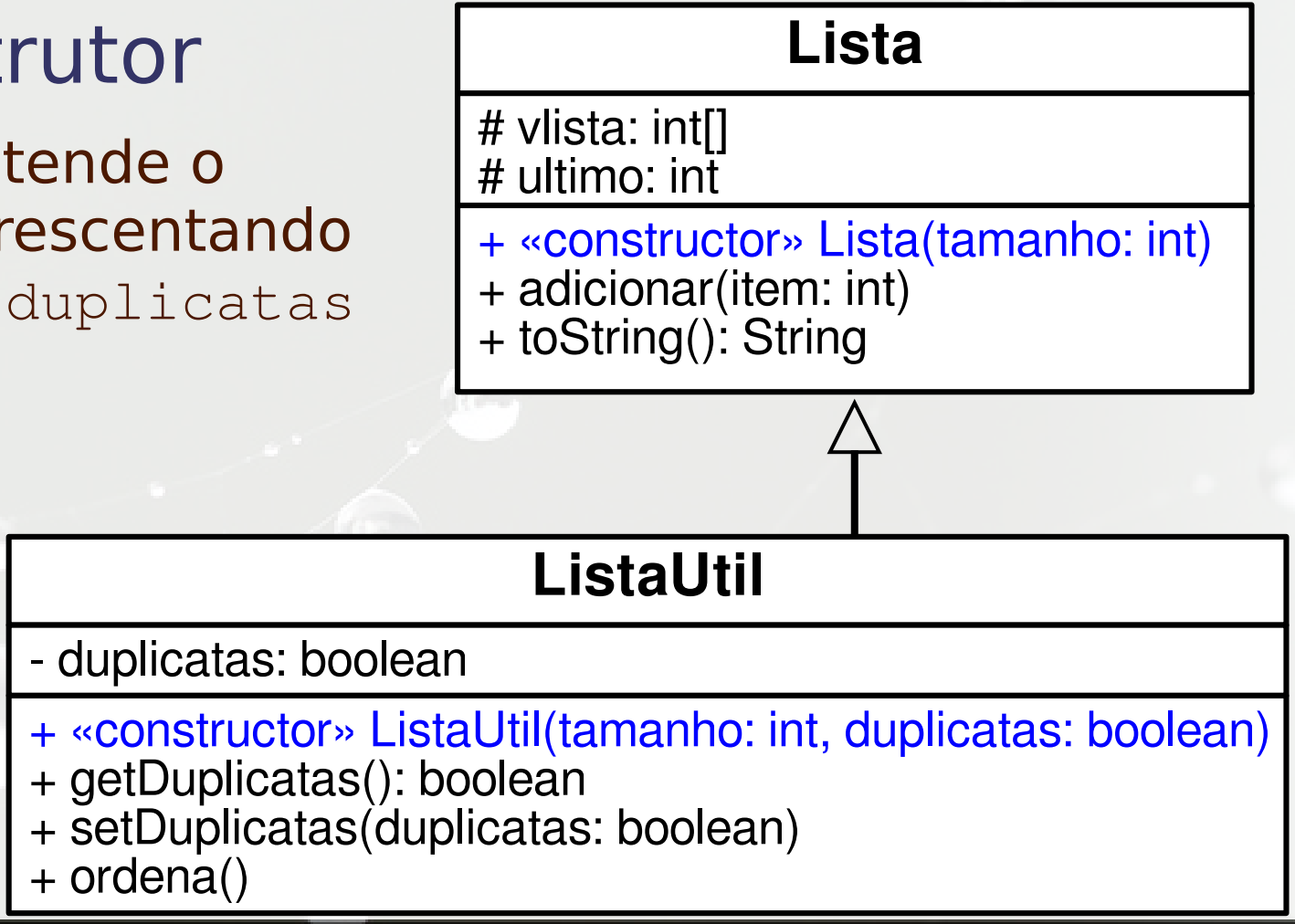
- Acrescentando um construtor com tamanho para Lista

```
protected int vlista[];


public Lista(int tamanho) {
   vlista = new int[tamanho];
}
```



Estendendo o Construtor

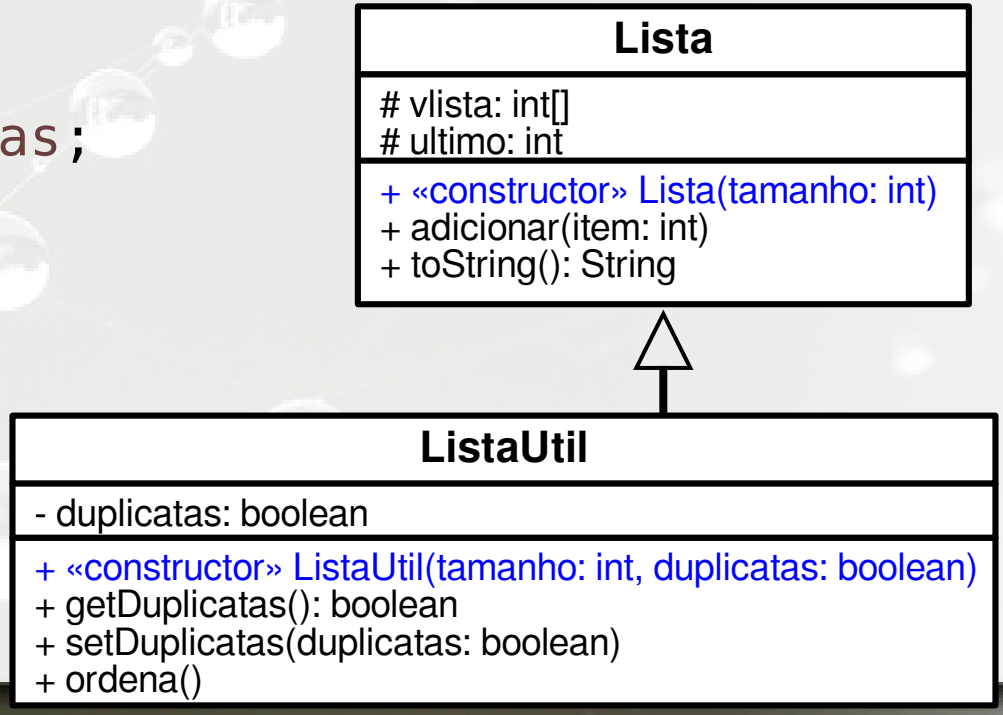
- `ListaUtil` estende o construtor acrescentando o controle de `duplicatas`



Estendendo o Construtor

- ListaUtil estende o construtor acrescentando o controle de duplicatas

```
public ListaUtil(int tamanho, boolean duplicatas) {
    super(tamanho);
    this.duplicatas = duplicatas;
}
```



Construtor sem argumentos implícito e Herdeiros

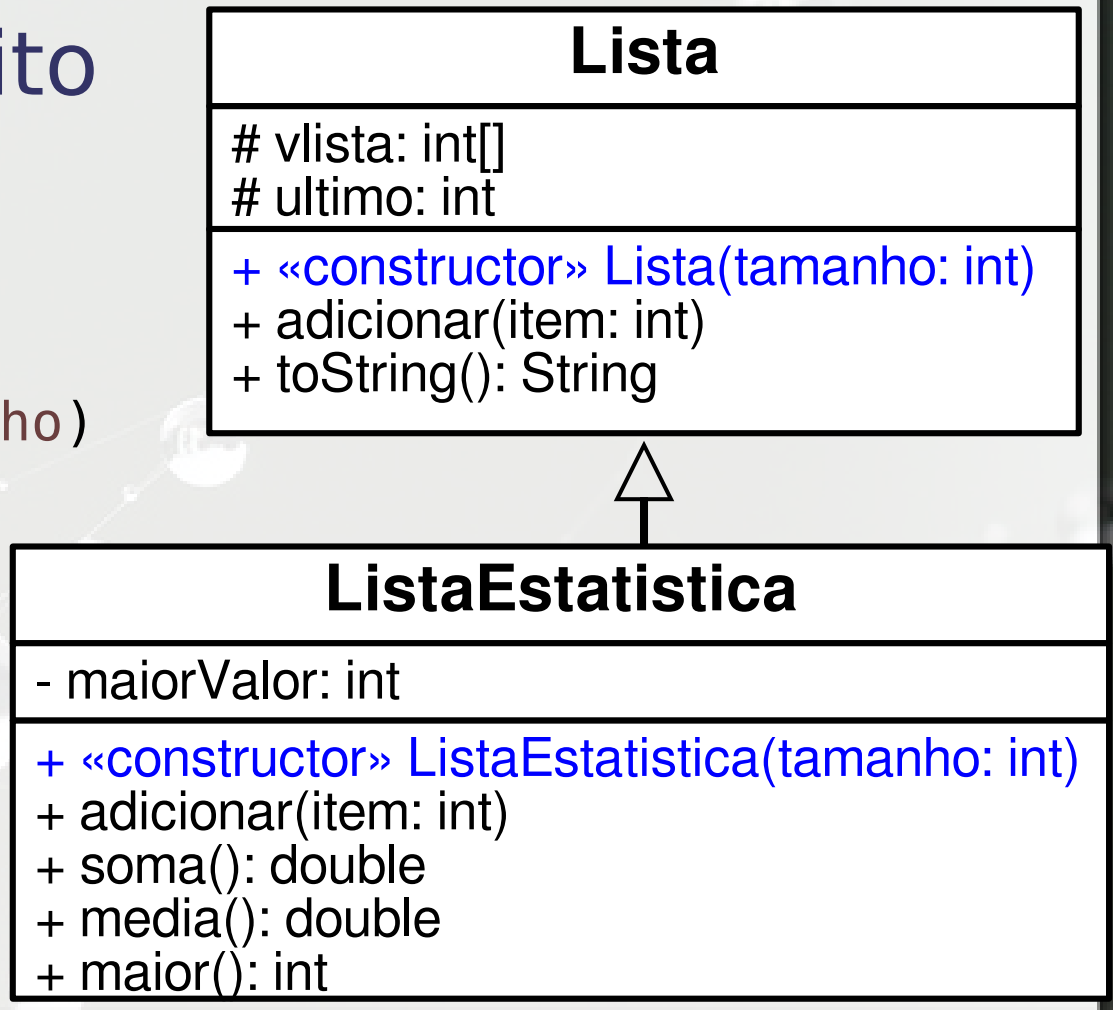
- O construtor sem argumentos é implícito (não precisa ser declarado)
- Quando uma classe declara somente construtor com argumentos:
  - o construtor sem argumentos deixa de existir
  - subclasses deixam de ter construtor implícito e precisam declará-lo

Exigência de Construtor Explícito

- `ListaEstatistica` é obrigada a declarar construtor explícito

```
ListaEstatistica(int tamanho)
{
    super(tamanho);
}
```

- Repassa chamada para superclasse



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

Licença

- Estes slides são concedidos sob uma Licença Creative Commons. Sob as seguintes condições: Atribuição, Uso Não-Comercial e Compartilhamento pela mesma Licença.
- Mais detalhes sobre a referida licença Creative Commons veja no link: http://creativecommons.org/licenses/by-nc-sa/3.0/
- Agradecimento a James Ratcliffe [http://www.flickr.com/photos/jamie/1762955591/] por sua fotografia “A spider web after a misty morning” usada na capa e nos fundos, disponível em [http://www.flickr.com/photos/jamie/1762955591/]

  vide licença específica da fotografia.