UPskill – JAVA + .NET

Programação Orientada a Objetos – ArrayLists

Adaptado de Donald W. Smith (TechNeTrain)

Objetivos

- Colecionar elementos usando ArrayLists
- Utilização do ciclo *foreach* para percorrer os elementos de ArrayLists

Conteúdos

- ArrayLists
- Ciclo *for each*

ArrayLists

- Quando é necessária uma estrutura para armazenar valores, nem sempre conhecemos quantos valores serão armazenados
- Nestas situações, um ArrayList apresenta duas vantagens significativas:
  - Um ArrayList pode crescer e diminuir
  - A classe `ArrayList` dispõe de métodos para realizar certas tarefas, como inserção e remoção de elementos

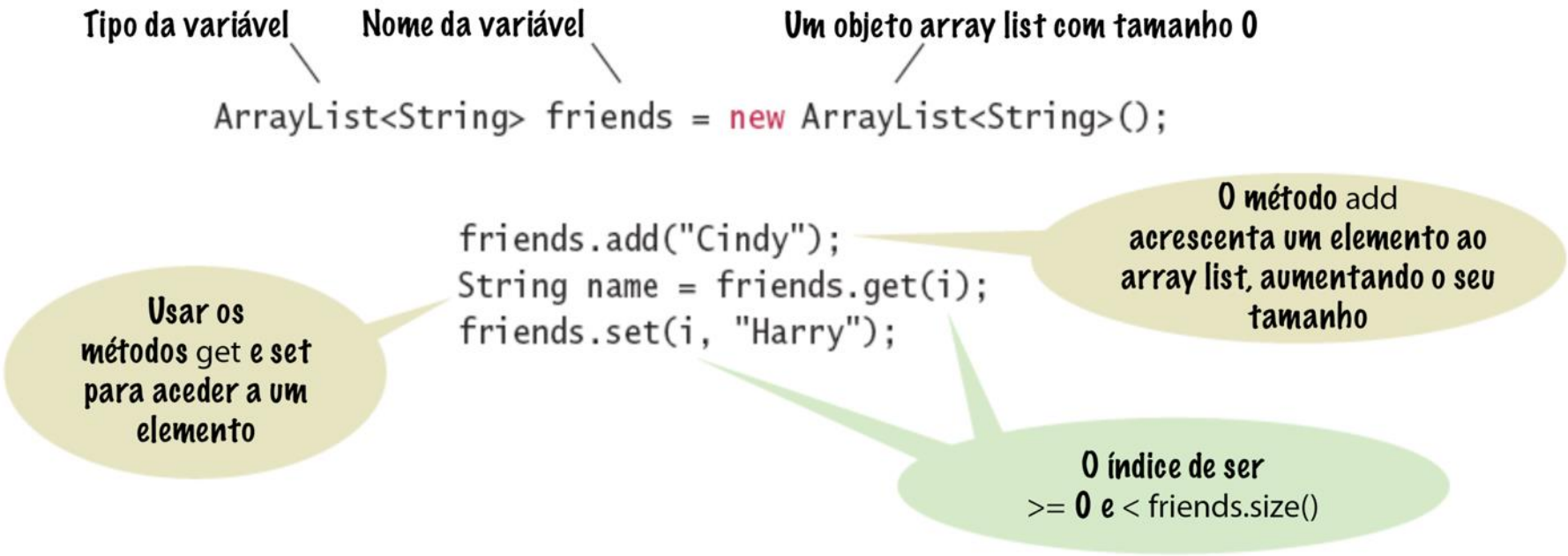
Um ArrayList expande-se para armazenar tantos elementos quanto os necessários

Declaração e Uso de ArrayLists

- A classe `ArrayList` pertence ao package `java.util`
  - É uma classe *generic*
    - Projetada para manter diferentes tipos de objetos
  - O tipo dos elementos é definido na declaração
    - Entre `<` `>` como '*type parameter*':
    - O tipo deve ser uma Classe
    - Não podem ser usados tipos primitivos (`int, double, …`)

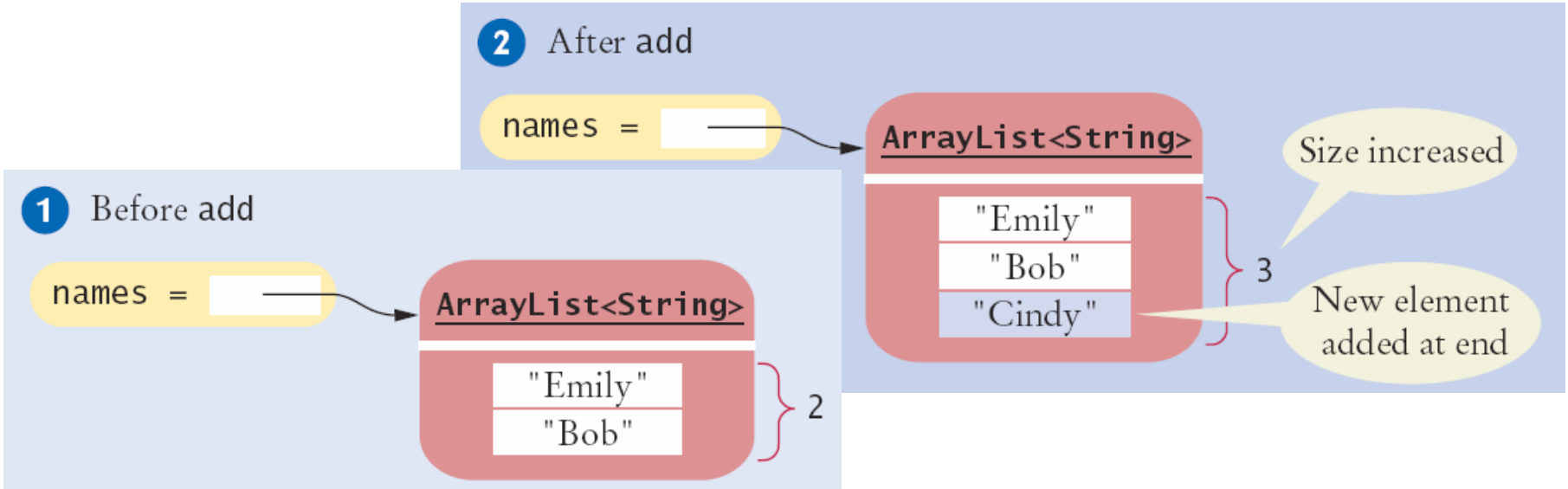
```
ArrayList<String> names = new ArrayList<String>();
```

Sintaxe dos ArrayLists



- A Classe ArrayList dispõe de vários métodos:
  - add: adiciona um elemento
  - get: devolve um elemento
  - remove: remove um elemento
  - set: altera um elemento
  - size: comprimento atual

Juntar um elemento com add()



- O método `add` tem duas versões:
  - Recebe um novo elemento para juntar ao fim

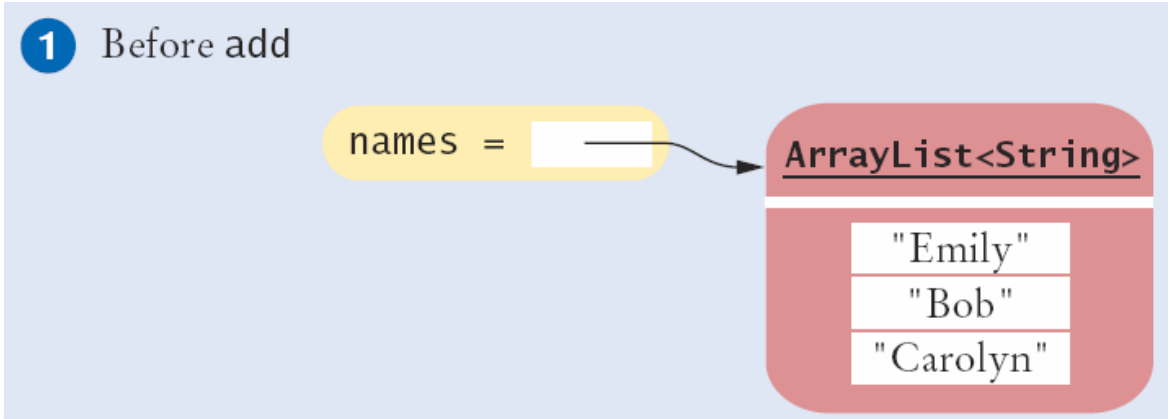
```
names.add("Cindy");
```

  - Recebe uma posição (índice) e o novo valor a adicionar

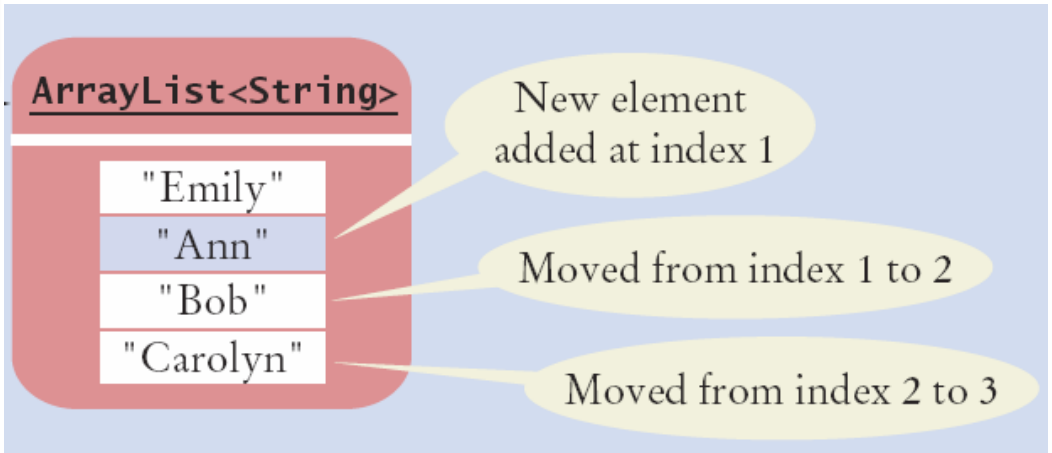
```
names.add(1, "Cindy");
```

Move todos os outros elementos

Adicionar um Elemento a Meio



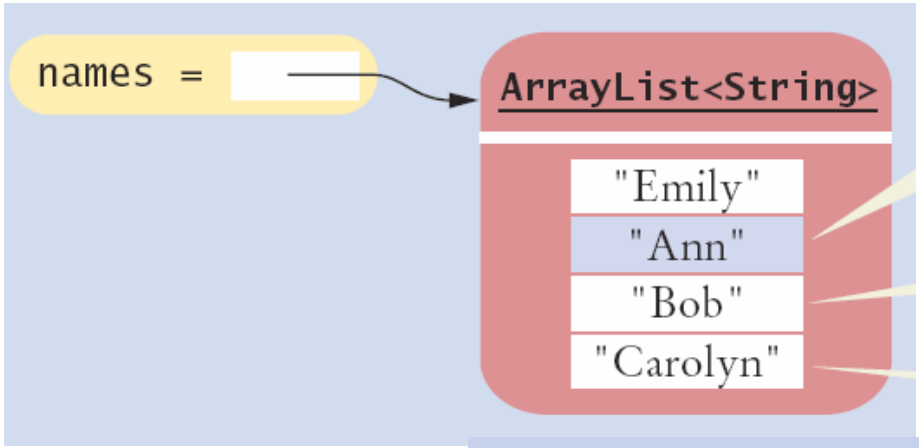

```
names.add(1, “Ann”);
```



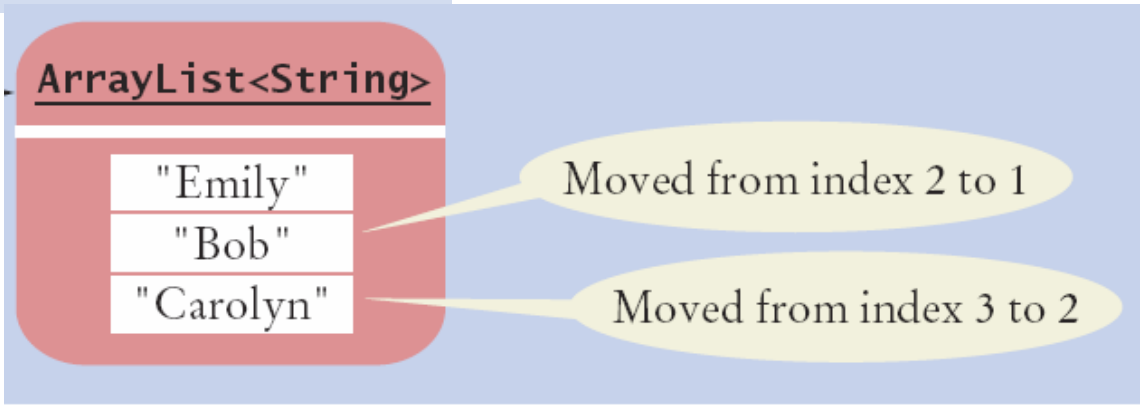
- Definir uma posição (índice) e o novo valor a acrescentar

Move todos os outros elementos

Remover um Elemento




```
names.remove(1);
```



- Definir uma posição (índice) a remover
  - Move todos os outros elementos

Ciclos e ArrayLists

- É possível usar o ciclo *foreach* com ArrayLists:

```
ArrayList<String> names = . . . ;
for (String name : names)
{
  System.out.println(name);
}
```

- Ou ciclos for tradicionais:

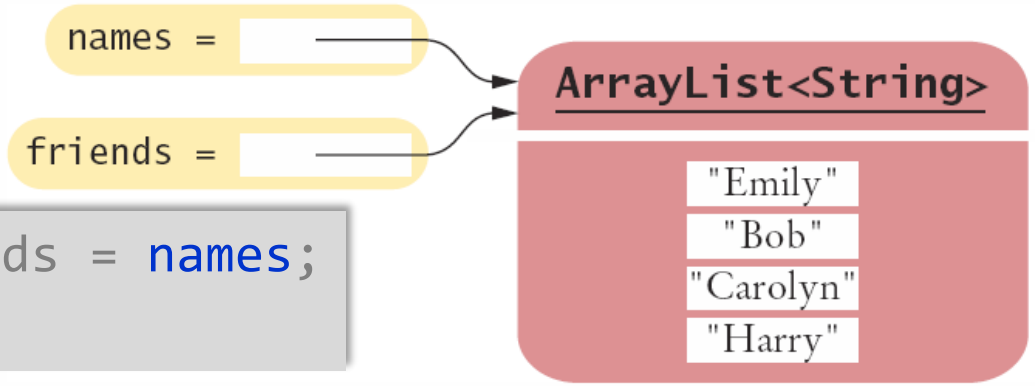
```
ArrayList<String> names = . . . ;
for (int i = 0; i < names.size(); i++)
{
  String name = names.get(i);
  System.out.println(name);
}
```

Utilização de ArrayLists

ArraysList<String> names = new ArrayList<String>();	Constrói um ArrayList vazio que poderá conter strings
names.add("Ann"); names.add("Cindy");	Adiciona elementos no fim
System.out.printl(names);	Imprime [Ann, Cindy]
names.add(1, "Bob");	Insere um elemento na posição 1. *names* contém agora [Ann, Bob, Cindy]
names.remove(0);	Remove o elemento na posição 0. *names* contém agora [Bob, Cindy]
names.set(0, "Bill");	Substitui um elemento por um novo valor. *names* contém agora [Bill, Cindy]
String name = names.get(i);	Obtém um elemento
String last = names.get(names.size() – 1);	Obtém o último elemento

Cópia de um ArrayList

- Uma variável `ArrayList` contém uma referência para um `ArrayList` (tal como os arrays)
- Cópia de uma referência:




```
ArrayList<String> friends = names;
friends.add("Harry");
```

- Para fazer uma cópia, passar a referência do `ArrayList` original para o construtor no novo `ArrayList`:

referência

```
ArrayList<String> newNames = new ArrayList<String>(names);
```

ArrayLists e Métodos

- Tal como os arrays, os ArrayLists podem ser usados como parâmetros e valores de retorno
- Exemplo: um método que recebe uma lista de `Strings` e devolve a lista invertida

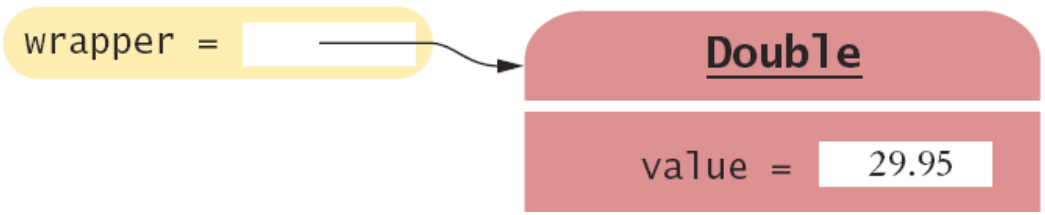
referência

```
public static ArrayList<String> reverse(ArrayList<String> names)
{
  // Allocate a list to hold the method result
  ArrayList<String> result = new ArrayList<String>();
  // Traverse the names list in reverse order (last to first)
  for (int i = names.size() - 1; i >= 0; i--)
  {
    // Add each name to the result
    result.add(names.get(i));
  }
  return result;
}
```

Wrappers Classes

- O Java possui *wrapper classes* para tipos primitivos
  - As conversões são automáticas
    - Primitivo para *Wrapper Class*

```
double x = 29.95;
Double wrapper;
wrapper = x;
```




```
double x;
Double wrapper = 29.95;
x = wrapper;
```

Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Wrappers Classes

- Não é possível usar tipos primitivos num `ArrayList`, mas podemos usar as suas *wrapper classes* correspondentes
- Declarar o `ArrayList` com *wrapper classes* para tipos primitivos
  - Usar `ArrayList<Double>`
    - Adicionar variáveis de tipo primitivo double
    - Ou valores double

```
double x = 19.95;
ArrayList<Double> values = new ArrayList<Double>();
values.add(29.95);
values.add(x);
double x = values.get(0);
```

Array e ArrayList

- Conversão de um Array para ArrayList requer mudar:
  - índice: `[i]`
  - `values.length`

```
double largest = values[0];
for (int i = 1; i < values.length; i++)
{
   if (values[i] > largest)
   {
      largest = values[i];
   }
}
```

- Para
  - metódos: `get()`
  - `values.size()`

```
double largest = values.get(0);
for (int i = 1; i < values.size(); i++)
{
   if (values.get(i) > largest)
   {
      largest = values.get(i);
   }
}
```

Escolher entre Arrays e ArrayLists

- Usar um Array se:
  - O tamanho do array nunca muda
  - É necessário armazenar um grande conjunto de valores de um dos tipos primitivos
    - Por razões de eficiência
- Usar um ArrayList:
  - Para as restantes situações
  - Especialmente se desconhecemos o número de elementos a armazenar

Erro Frequente

UPskill
Digital Skills & Jobs

- *Length* versus *Size*
  - A sintaxe Java para obter o número de elementos num array, num `ArrayList` e numa `String` não é consistente
  - É necessário usar a sintaxe correta para cada um dos tipos:

Tipo	Número de Elementos
Array	a.length
ArrayList	a.size()
String	a.length()

Sumário: ArrayLists

- Um ArrayList armazena uma sequência de valores cujo comprimento pode mudar
  - A classe ArrayList é uma classe genérica: `ArrayList<Type>` coleciona elementos do tipo especificado
  - Usar o método `size` para obter o tamanho atual do ArrayList
  - Usar os métodos `get` e `set` para aceder a um elemento do ArrayList numa dada posição
  - Usar os métodos `add` e `remove` para adicionar e remover elementos do ArrayList
- Para armazenar números num ArrayList será necessário usar uma das *wrapper classes*