4

ARRAYLISTS e ENUM TYPES

Paradigmas de Programação

LEI - ISEP

Luiz Faria, adaptado de Donald W. Smith (TechNeTrain.com)

Objetivos

- Colecionar elementos usando array lists
- Utilização do ciclo *for each* para percorrer os elementos de array lists
- Utilização de tipos enumerados para predefinição de um conjunto de valores que uma variável poderá assumir

Conteúdos

- Arrays Lists
- Ciclo *for each*
- Enum Types

Array Lists

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

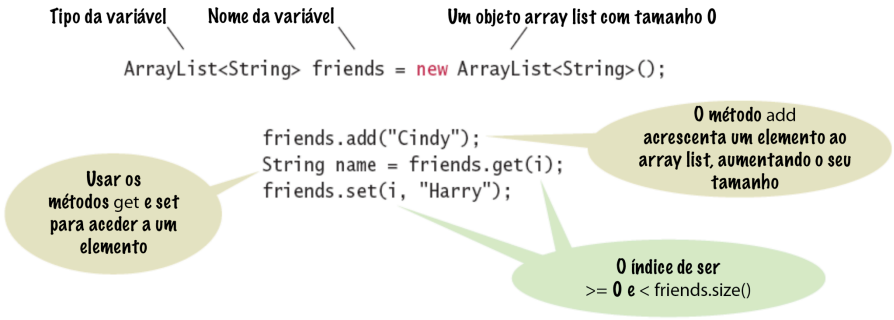
Um Array List expande-se para armazenar tantos elementos quanto os necessários

Declaração e Uso de Array Lists

- 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`, ...)

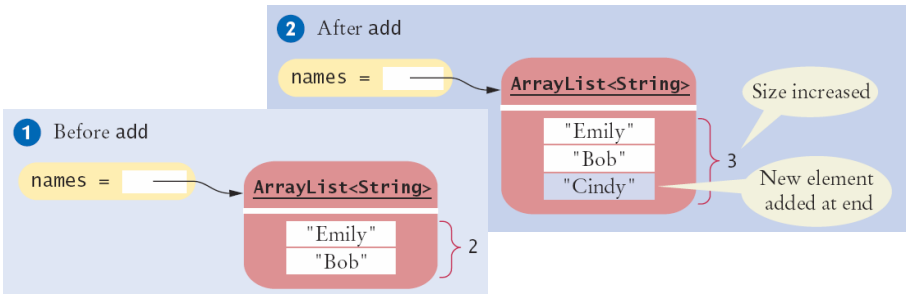
```
ArrayList<String> names = new ArrayList<String>();
```

Sintaxe dos Array Lists



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

Juntar um elemento com add()



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

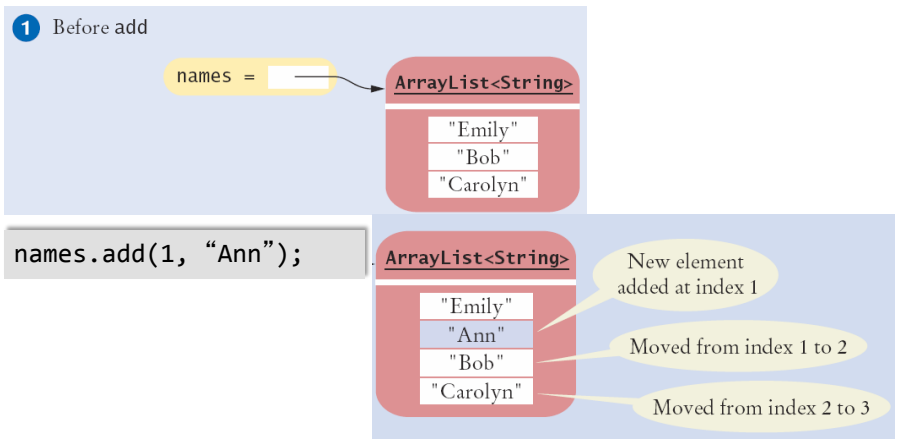
```
names.add("Cindy");
```

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

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

Move todos os outros elementos

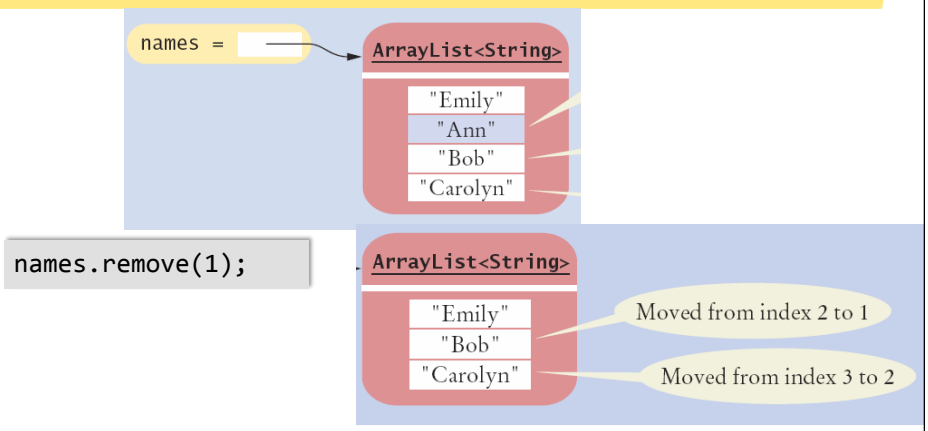
Adicionar um Elemento a Meio



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

Move todos os outros elementos

Remover um Elemento



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

Ciclos e Array Lists

- É possível usar o ciclo *for each* com Array Lists:

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

- Ou ciclos "normais":

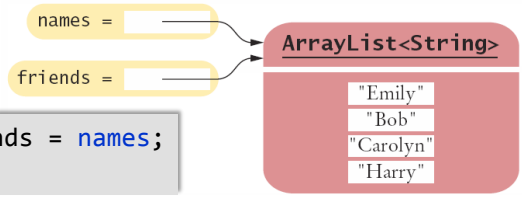
```
ArrayList<String> names = . . . ;
for (int i = 0; i < names.size(); i++)
{
  String name = names.get(i);
  System.out.println(name);
}
```

Utilização de Array Lists

ArraysList<String> names = new ArrayList<String>();	Constrói um array list 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);
```

Array Lists e Métodos

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

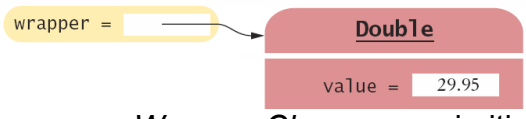
referência

```
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;
```



    - *Wrapper Class* para primitivo

```
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 Array Lists

- 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 Array List:
  - Para as restantes situações
  - Especialmente se desconhecemos o número de elementos a armazenar

Comparação de Operações entre Array e Array List

Operação	Arrays	Array Lists
Obter um elemento	x = values[4];	x = values.get(4);
Substituir um elemento	values[4] = 35;	values.set(4, 35);
Número de elementos	values.length	values.size();
Remover um elemento	(não é direto)	values.remove(4);
Adicionar um elemento, aumentando a dimensão da estrutura	(não é direto)	values.add(35);
Inicializar a estrutura	int[] values = {1, 4, 9};	Chamar *add* três vezes

Erro Frequente

- *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
Array list	a.size()
String	a.length()

Sumário: Array Lists

- Um Array List 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 array list
  - Usar os métodos `get` e `set` para aceder a um elemento do array list numa dada posição
  - Usar os métodos `add` e `remove` para adicionar e remover elementos do array list
- Para armazenar números num Array List será necessário usar uma das *wrapper classes*

Enum Types

- Um tipo enumerado é um tipo especial que permite predefinir um conjunto de valores que poderão ser assumidos por uma variável
- A variável assume um dos valores predefinidos
- Exemplo: variável para representar um dos 4 pontos cardeais (norte, sul, este, oeste), ou os dias da semana

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

- Uma vez que os diferentes valores são constantes, devem ser representados com letra maiúscula

Enum Type: Exemplo 1

```
public class EnumTest {
    Day day;
    public EnumTest(Day day) {
        this.day = day;
    }
    public void tellItLikeItIs() {
        switch (day) {
            case MONDAY:
                System.out.println("Mondays are bad.");
                break;
            case SATURDAY: case SUNDAY:
                System.out.println("Weekends are best.");
                break;
        }
    }
    public static void main(String[] args) {
        EnumTest firstDay = new EnumTest(Day.MONDAY);
        firstDay.tellItLikeItIs();
        EnumTest seventhDay = new EnumTest(Day.SUNDAY);
        seventhDay.tellItLikeItIs();
    }
}
```

Output:
Mondays are bad.
Weekends are best.

Enum Type: uma Classe

- A declaração *enum* define uma classe (designada por *enum type*)
- O corpo da classe *enum* pode incluir métodos e outros campos
- As constantes têm que ser definidas antes dos atributos e dos métodos; quando existem atributos e métodos, a declaração das constantes tem que terminar com ;
- O compilador adiciona automaticamente alguns métodos especiais quando cria um *enum*. Por exemplo, inclui um método estático que devolve um array contendo todos os valores de um *enum* na ordem pelo qual foi declarado

```
for (Planet p : Planet.values()) {
  System.out.printf("Your weight on %s is %f%n”, p,
  p.surfaceWeight(mass));
}
```

Planet é um tipo enumerado. O método value() pode ser usado num ciclo *for each* para iterar sobre todos os valores de um tipo enumerado

Enum Type: Construtor

- Quando as constantes de um tipo enumerado são declaradas com valores, estes valores são passados ao construtor no momento em que a constante é criada
- O construtor de um *enum type* deve ter acesso *package private* ou *private*
- O construtor cria automaticamente as constantes que estão definidas no início do corpo
- Não é possível invocar o construtor

Enum Type: Exemplo 2

```
public enum Planet {
    MERCURY (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    EARTH   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6);
    private final double mass;
    private final double radius;
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    private double mass() { return mass; }
    private double radius() { return radius; }
    public static final double G = 6.67300E-11;

    double surfaceGravity() {
        return G * mass / (radius * radius);
    }
    double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
```

```
    public static void main(String[] args) {
        double earthWeight = 175;
        double mass = earthWeight/
           EARTH.surfaceGravity();
        for (Planet p : Planet.values())
           System.out.printf("Your weight
           on %s is %f%n",
           p, p.surfaceWeight(mass));
    }
}
```

Output:
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
Your weight on MARS is 66.279007