Programação OO em Java

Baseado nos materiais de

Profa. Andréa S. Charão
Prof. Guilherme D. Juraszek

(rev. de Alexandre G. Silva, 09/2017)

Sumário

- Classes abstratas
- Interfaces
- Tipos genéricos
- Coleções

Classes abstratas

- São classes que **não podem** ser instanciadas, porque representam entidades "incompletas"
- Possuem métodos abstratos que devem ser sobrescritos nas classes derivadas

```
abstract class Bicho
{
  protected String nome;
  public Bicho(String nome)
  {
    this.nome = nome;
  }
  abstract public String som();
}
```

mensagem do compilador:

```
Bicho.java: Bicho is abstract; cannot be instantiated
    Bicho b = new Bicho();
              ^
1 error
```

Exemplo

```
abstract class Bicho
{
  protected String nome;
  public Bicho(String nome)
  {
    this.nome = nome;
  }
  abstract public String som();
}

class Cachorro extends Bicho
{
  public Cachorro(String nome)
  {
    super(nome);
  }
  public String som()
  {
    return "Au Au";
  }
}

class Gato extends Bicho
{
  public Gato(String nome)
  {
    super(nome);
  }
  public String som()
  {
    return "Miau";
  }
}
```

Método abstrato som()

Método concreto som()

Cria array de referências para Bichos

```
class BichoApp
{
  public static void ma (String[] args)
  {
    Bicho[] bs = new Bicho[2];
    bs[0] = new Cachorro("Scooby");
    bs[1] = new Gato("Garfield");
    for (int i = 0; i < bs.length; i++)
      System.out.println(bs[i].som());
  }
}
```

Classes abstratas

Erro: classe não abstrata com método abstrato

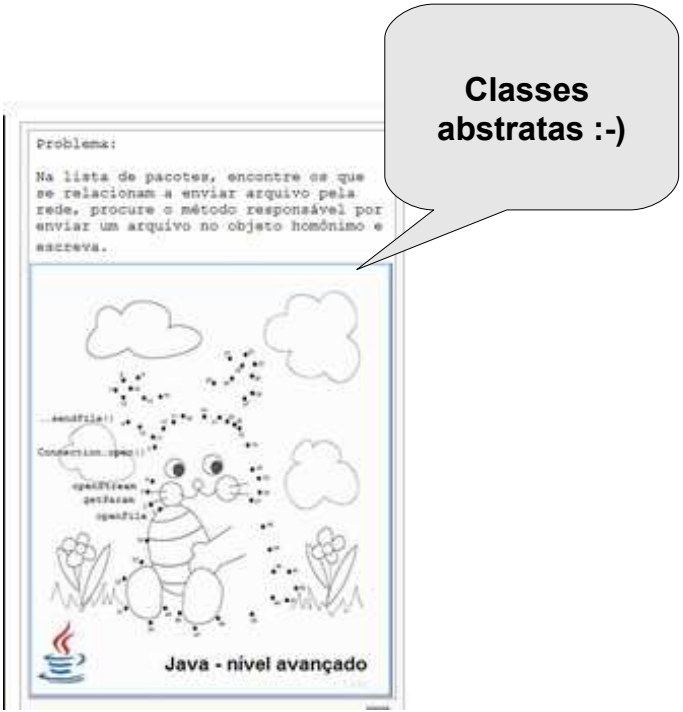
```
class Bicho
{
  protected String nome;
  public Bicho(String nome)
  {
    this.nome = nome;
  }
  abstract public String som();
}
```

- Métodos abstratos só podem ser declarados em classes abstratas
- Em geral, classes abstratas também possuem métodos concretos
- Se uma classe só tem métodos abstratos, é melhor declará-la como **interface**

mensagem do compilador:

```
Bicho.java: Bicho is not abstract and does
not override abstract method som() in
Bicho
class Bicho
^
1 error
```

Java na Desciclopédia :-)



Fonte:
http://desciclopedia.org/wiki/Java_(linguagem_de_programação)

Interfaces

- São um tipo de encapsulamento contendo principalmente métodos
- Definem um conjunto de métodos (comportamento) que devem ser implementados em classes que herdam a interface

```
interface Matricial
{
  public void transpoe();
  public void inverte();
}
```

```
interface Runnable
{
  public void run();
}
```

Implementando interfaces

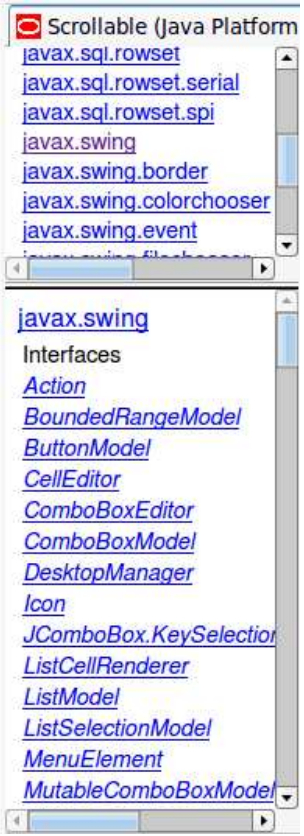
- Usar a palavra-chave **`implements`**

```
class MatrizEsparsa
  implements Matricial
{
  public void transpoe()
  { ... }
  public void inverte()
  { ... }
}
```

```
class Worker
  implements Runnable
{
  public void run()
  { ... }
}
```

Implementando interfaces

- Classes que implementam uma mesma interface garantem que têm um comportamento comum
- A plataforma Java tem diversas interfaces pré-definidas (ActionListener, Scrollable, Runnable, etc.)



Mais sobre interfaces

- Java suporta "herança múltipla" de interfaces, mas não de classes

```
class A {...}
interface B {...}
interface B {...}

class X extends A
implements B,C
{...}
```

Mais sobre interfaces

- Atributos declarados em interfaces são implicitamente `public static final` (constantes)
- Métodos declarados em interfaces são implicitamente `public abstract`

Veja mais sobre interfaces em:
http://download.oracle.com/javase/tutorial/java/concepts/interface.html

Tipos genéricos

- Classes genéricas definidas em função de algum parâmetro (tipos parametrizáveis)
- **Polimorfismo** paramétrico

```
class Box<T> {

    private T t; // T significa "Type"

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
}
```

Usando tipos genéricos

- Para usar o tipo, define-se o parâmetro específico

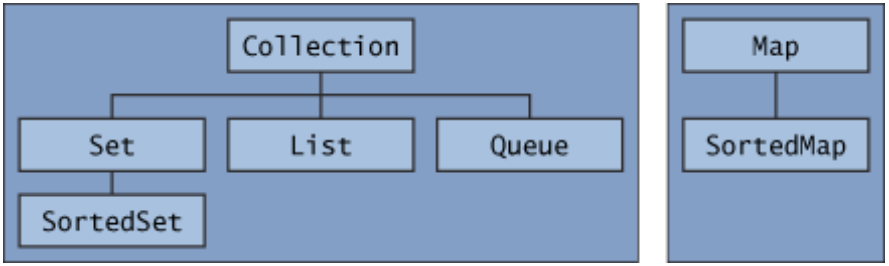
```
class BoxApp {

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        Integer i = integerBox.get();
        System.out.println(i);

        Box<String> stringBox = new Box<String>();
        stringBox.add("Hello");
        String s = stringBox.get();
        System.out.println(s);

    }
}
```

Veja mais em:
http://download.oracle.com/javase/tutorial/java/generics/index.html

Collections em Java

- Um framework com **estruturas de dados** e **algoritmos** reutilizáveis, disponíveis em java.util.
- Componentes
  - **Interface**s: definem como as estruturas podem ser manipuladas (ex.: List)
  - **Implementações**: definem estruturas concretas (ex.: ArrayList, LinkedList)
  - **Algoritmos**: métodos estáticos que se aplicam a diferentes coleções

Collections Interface



Fonte:
http://download.oracle.com/javase/tutorial/collections/interfaces/index.html

Exemplo de implementação: ArrayList

- Representa uma lista que pode ser acessada por índices (0 a size()-1)
- Implementa métodos da **interface List**:
  - add(E e): adiciona elemento
  - size(): número de elementos da lista
  - clear(): remove todos os elementos
  - isEmpty(): verifica se lista é vazia
  - remove(Object o): remove elemento
  - remove(int index): remove elemento
  - etc.

Exemplo

```
import java.util.*;
class ArrayListExemplo {
   public static void main(String args[]) {
      // cria objeto
      ArrayList<String> sl = new ArrayList<String>();
      System.out.println("Tamanho inicial: " + sl.size());
      // adiciona elementos
      sl.add("Fulano");
      sl.add("Beltrano");
      sl.add("Sicrano");
      sl.add("Fulana");
      System.out.println("Novo tamanho: " + sl.size());
      // remove elementos
      sl.remove("Beltrano");
      sl.remove(0);
      System.out.println("Conteudo: " + sl);
   }
}
```

saída:

```
Tamanho inicial: 0
Novo tamanho: 4
Conteudo: [Sicrano, Fulana]
```

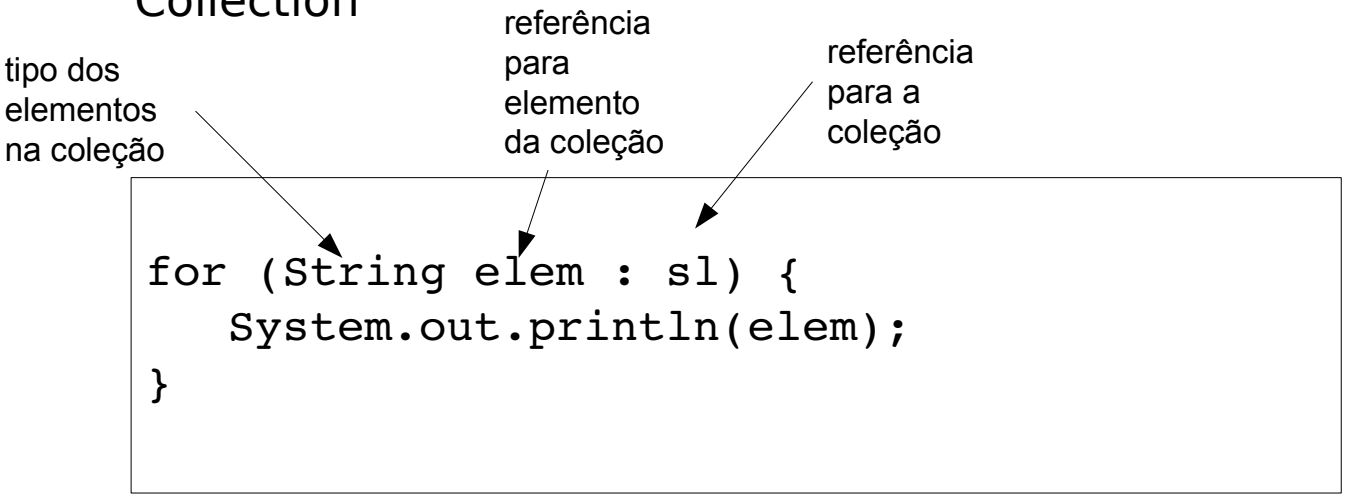
Percorrendo a lista

- Laço **for** tradicional, com índice

```
for (int i = 0; i < sl.size(); i++) {
   String elem = sl.get(i);
   System.out.println(elem);
}
```

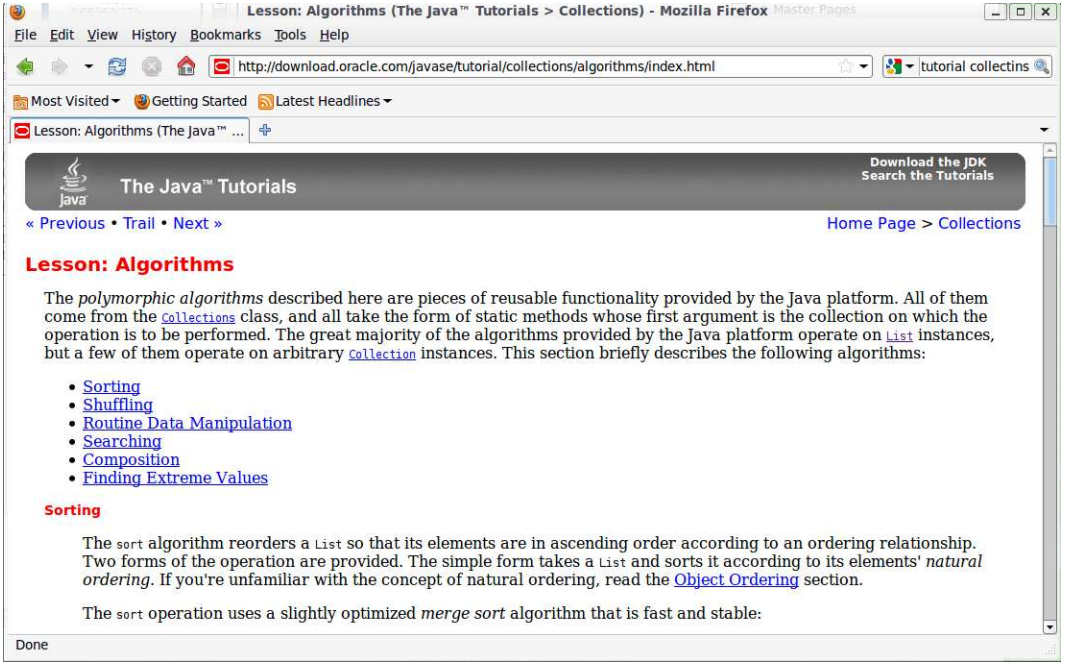
Percorrendo a lista com for-each

- Laço **for** alternativo (**for-each**)
- Inspirado na programação funcional
- Pode ser usado com arrays ou classes Collection



Algoritmos

- Ordenação, busca, embaralhamento, etc.



Algoritmos: sort

```
import java.util.*;

class Sort {
  public static void main(String[] args) {
    String[] array = {"cadabra", "abra"};
    List<String> list = Arrays.asList(array);
    Collections.sort(list);
    System.out.println(list);
  }
}
```

Listas (List e ArrayList)

Ao utilizar vetores é necessário especificar o tamanho do vetor. Mas e se quisermos armazenar uma lista de coisas e ainda não sabemos quantas são?

ArrayList funcionam de forma semelhante aos vetores, porém permitem a inserção e remoção de itens de forma dinâmica

Os valores são armazenados na ordem em que foram inseridos

Listas (List e ArrayList)

```
...
import java.util.ArrayList;
import java.util.List;
...


...
List<String> nomes = new ArrayList<String>();

//Adicionando elementos na lista
nomes.add("Zezinho");
nomes.add("Huguinho");
nomes.add("Luizinho");

//Remove um elemento da lista
nomes.remove("Zezinho");

//Verifica se Huguinho está na lista
boolean huguinhoAdicionado = nomes.contains("Huguinho");
System.out.println("Huguinho está na lista? " + huguinhoAdicionado);

//Verifica se Zezinho está na lista
boolean zezinhoAdicionado = nomes.contains("Zezinho");
System.out.println("Zezinho está na lista? " + zezinhoAdicionado);

//Apaga todos os itens da lista
nomes.clear();

//Recupera um elemento da lista por meio do índice
String nome = nomes.get(0);
...
```

Conjuntos (Set e HashSet)

Os conjuntos (sets) são coleções de dados que não permitem elementos duplicados.

A ordem em que os elementos são armazenados **pode não ser** a ordem na qual foram inseridos.

É representado pela interface *Set* e tem como principais implementações o *HashSet*, **LinkedHashSet* e *TreeSet*.

*o LinkedHashSet mantém a ordem de inserção

Conjuntos (Set e HashSet)

```
...
import java.util.HashSet;
import java.util.Set;
...

...
Set<String> nomes = new HashSet<String>();

//Adicionando elementos no conjunto
nomes.add("Zezinho");
nomes.add("Huguinho");
nomes.add("Luizinho");

//Remove um elemento do conjunto
nomes.remove("Zezinho");

//Verifica se Huguinho está no conjunto
boolean huguinhoAdicionado = nomes.contains("Huguinho");
System.out.println("Huguinho está na lista? " + huguinhoAdicionado);

//Verifica se Zezinho está no conjunto
boolean zezinhoAdicionado = nomes.contains("Zezinho");
System.out.println("Zezinho está na lista? " + zezinhoAdicionado);

//Apaga todos os itens do conjunto
nomes.clear();
...
```

Listas e Conjuntos - Percorrendo

Podemos utilizar o **for** para percorrer as listas e conjuntos. Funciona da mesma forma para Set e List.

```
...
List<String> nomes = new ArrayList<String>();

//Adicionando elementos na lista
nomes.add("Zezinho");
nomes.add("Huguinho");
nomes.add("Luizinho");

//Imprime todos os valores da lista
for(String nome : nomes){
    System.out.println("Nome: " + nome);
}
...
```

Mapas (Map)

Muitas vezes queremos buscar um objeto a partir de alguma informação sobre ele.

Ex:

- Buscar um carro a partir do número da placa
- Buscar um usuário a partir do e-mail
- Buscar uma pessoa a partir do CPF
- Buscar um livro a partir do ISBN

Mapas (Map)

Um mapa é composto por um conjunto de associações entre um objeto **chave** e um objeto **valor**. É equivalente ao conceito de dicionário.

É representado pela interface Map e as principais implementações são **HashMap**, TreeMap e Hashtable.

Mapas (Map)

```
...
import java.util.HashMap;
import java.util.Map;
...

...
Map<String, String> alunos = new HashMap<String,String>();

//Adicionando elementos no mapa
alunos.put("435", "Zezinho");
alunos.put("436", "Huguinho");
alunos.put("437", "Luizinho");

//Remove um elemento do mapa
alunos.remove("437");

//Verifica se Huguinho está no mapa
boolean huguinhoAdicionado = alunos.containsKey("436");
System.out.println("Huguinho está na lista? " + huguinhoAdicionado);

//Verifica se Zezinho está no mapa
boolean zezinhoAdicionado = alunos.containsValue("Zezinho");
System.out.println("Zezinho está na lista? " + zezinhoAdicionado);

//Apaga todos os itens do mapa
alunos.clear();
...
```

Mapa - Percorrendo um mapa

Podemos utilizar o for para extrair as chaves e buscar os valores em um mapa

```
Map<String, String> alunos = new HashMap<String,
String>();

//Adicionando elementos no mapa
alunos.put("435", "Zezinho");
alunos.put("436", "Huguinho");
alunos.put("437", "Luizinho");

//Imprime todos os valores no mapa
for(String matricula : alunos.keySet()){
    String nome = alunos.get(matricula);
    System.out.println("Matricula: " + matricula + " - "
+ nome);
}
```

Coleções - Exercícios

1) Crie um código que insira 30 mil números quaisquer (de preferência, aleatórios) em um ArrayList, pesquise-os e imprima o tempo gasto.

```
...
long inicio = System.currentTimeMillis();
// *** coloque o seu código aqui ***
long fim = System.currentTimeMillis();
long tempoTotal = fim - inicio;
System.out.println("Tempo gasto: " + tempoTotal);
...
```

2) Substitua o uso de ArrayList por HashSet e execute novamente. Qual foi mais rápido?

3) Qual operação foi mais lenta, a inserção ou a consulta?

Tratamento de exceções

Tratamento de exceções (I)

- **Exceção** é evento gerado pela execução de uma ação que a máquina não consegue realizar
- Ou seja, um erro em tempo de execução
- Exemplos:
  - Divisão por zero;
  - Encontrar o valor numérico de um string que não representa um valor;
  - Em um array, acessar um elemento fora da faixa;
  - Solicitar execução de um método para um objeto cuja referência vale null;
  - Memória insuficiente;
  - Abrir um arquivo que não existe no local especificado.

Tratamento de exceções (II)

- Sempre que há uma exceção, Java cria automaticamente um objeto

  que representa a exceção.
- O objeto da exceção verifica se não há algum código definido pelo projetista que deve ser executado
  - Caso exista – tratamento da exceção
  - Caso não exista – o programa será automaticamente abortado

- Exemplo:

```
//...
int a = 10;
int b, c;
String s = JOptionPane.showInputDialog("Digite um valor: ");
b = Integer.parseInt(s); // pode ocorrer erro
c = a / b;  // pode ocorrer erro
```

Tratamento de exceções (III)

- Exemplo com tratamento de exceção:

```
//...
int a = 10;
int b, c;
try
{
    String s = JOptionPane.showInputDialog("Digite um valor: ");
    b = Integer.parseInt(s);
    c = a / b;
}
catch (Exception e)
{
    System.out.println("Ocorreu algum erro");
}
//...
```

Tratamento de exceções (IV)

- Exemplo com tratamento específico de exceção:

```
//...
int a = 10;
int b, c;
try
{
    String s = JOptionPane.showInputDialog("Digite um valor: ");
    b = Integer.parseInt(s);
    c = a / b;
}
catch (NumberFormatException nfe)
{
    System.out.println("Foi digitado valor invalido");
}
catch (DivideByZeroException d)
{
    System.out.println("Ocorreu divisao por zero");
}
//...
```

Tratamento de exceções (V)

- try–catch–finally

```
try {
  /* coloque aqui o codigo que a maquina tentara' executar e que
     podera' causar alguma excecao */
}
catch (ClasseExcecao1 e1) {
  /* coloque aqui as instrucoes que serao executadas caso ocorra
     uma excecao da classe ClasseExcecao1 */
}
catch (ClasseExcecao2 e2) {
  /* coloque aqui as instrucoes que serao executadas caso ocorra
     uma excecao da classe ClasseExcecao2 */
}
//...
finally {
  /* bloco opcional; obrigatorio se nao for colocado nenhum
  catch; estas instrucoes serao executadas ocorrendo ou nao
  uma excecao */
}
```

Tratamento de exceções (VI)

- Algumas classes de exceção
  - Exception *(superclasse)*
    - ★ RuntimeException
      - ArithmeticException
        - DivideByZeroException *(divisão por zero)*
      - IndexOutOfBoundsException
        - ArrayIndexOutOfBoundsException *(índice array fora da faixa)*
        - StringIndexOutBoundsException *(índice string fora do interv.)*
      - NegativeArraySizeException *(tamanho negativo do array)*
      - NullPointerException *(uso de um objeto cuja referência vale null)*
      - IllegalArgumentException
        - NumberFormatException *(valor em formato inadequado)*
      - SecurityException *(acessar dados de um arquivo não permitido)*
    - ★ IOException
      - EOFException *(tentar ler após fim de arquivo)*
      - FileNotFoundException *(arquivo não encontrado)*

Tratamento de exceções (VII)

- Propagação de exceções – **throws**
  - É possível deixar o tratamento de exceções para quem ativou o método
  - Exemplo:

```
import javax.swing.*;
public class Interfaces {
    public int leInteiro() throws NumberFormatException {
        /* Caso ocorra NumberFormatException sem
           tratamento aqui, a excecao sera' propagada
           (throws) para quem ativou este metodo */
        String s = JOptionPane.showInputDialog("Valor: ");
        return Integer.parseInt(s);
} }
```

```
public class Principal {
    public static void main (String arg[]) {
        int i;
        Interface umaI = new Interface();
        try {
            i = umaI.leInteiro();
        }
        catch (NumberFormatException e) {
            i = -1;
} } }
```