Wrapper Classes

ICOMP
Instituto de Computação

Horácio Fernandes
horacio@icomp.ufam.edu.br

Slides: http://tinyurl.com/slides-tp

Wrapper Classes
Introdução

- São classes que “empacotam” um tipo primitivo
- Permitem manipular variáveis de tipos primitivos como se fossem objetos de uma classe
  - Importante para poder utilizar diversos métodos em java que aceitam apenas objetos como parâmetros (ou seja, não aceitam tipos primitivos)
- Também possuem uma série de métodos utilitários para manipular os seus respectivos tipos primitivos

Wrapper Classes
Introdução

- Java possui as seguintes *wrapper classes*:

Wrapper Class	Tipo Primitivo
`Integer`	**`int`**
`Short`	**`short`**
`Long`	**`long`**
`Byte`	**`byte`**
`Float`	**`float`**
`Double`	**`double`**
`Character`	**`char`**
`Boolean`	**`boolean`**

Wrapper Classes
Criando Objetos

- *Wrapper classes* podem ser utilizadas como qualquer outra:
  - Utilizando o operador `new` para instanciar um novo objeto

```
Integer numAlunos = new Integer(20);
Float peso = new Float(5.6f);
```

- Por serem classes, possuem outros construtores também
  - Por exemplo, podemos instanciá-los usando `String`

```
Integer numAlunos = new Integer("20");
Float peso = new Float("5.6f");
```

- Ou podemos simplesmente atribuí-los a números diretamente
  - Como fazemos ao criar uma string literal

```
Integer numAlunos = 20;
Float peso = 5.6f;
```

Wrapper Classes
Conversões Boxing

- No último exemplo do slide anterior, um número (constante de tipo primitivo) foi utilizado no lugar de um objeto
- O Java automaticamente converte entre tipos primitivos e *wrapper classes* nos lugares necessários
  - Autoboxing (ou simplesmente *boxing*)
    - *Converte um valor primitivo para um objeto da classe correspondente*

```
Integer numAlunos = 20;
```

O número 20 será convertido para `new Integer(20);`

  - Auto-Unboxing (ou simplesmente *unboxing*)
    - *Converte um objeto de uma wrapper class para o seu tipo primitivo*

```
int numAlunos4 = new Integer(30);
```

O objeto será convertido para o o valor 20 do tipo primitivo `int`

Wrapper Classes
Comportamento das Referências I

- Em geral, objetos das *wrapper classes* terão referências diferentes, mesmo que contenham o mesmo conteúdo:

```
Float f1 = 5.5f;
Float f2 = 5.5f;
// False, pois possuem referências diferentes
System.out.println("f1 == f2 ? " + (f1 == f2));
```

- Note alguns comportamentos do *boxing/unboxing*

```
float f3 = 5.5f;
// True, pois f1 é unboxed para o tipo primitivo float
System.out.println("f1 == f3 ? " + (f1 == f3));
// True, pois f1 é unboxed para o tipo primitivo float
System.out.println("f1 == 5.5f ? " + (f1 == 5.5f));
// True, pois o equals compara conteúdo
System.out.println("f1.equals(5.5f) ? " + f1.equals(5.5f));
// False, pois "5.5" é boxed para a classe Double ao invés de Float
System.out.println("f1.equals(5.5) ? " + f1.equals(5.5));
```

Wrapper Classes
Comportamento das Referências II

- Entretanto, por motivos de performance, objetos da classe `Integer` entre -128 e 127, `Boolean` e `Character` entre 0 e 127, possuem a mesma referência caso possuam o mesmo valor:
  - Isso é detalhe da linguagem, dificilmente notado e comentado na prática

```
Integer i1 = 5;
Integer i2 = 5;
Integer i3 = 129;
Integer i4 = 129;
// True, pois possuem a mesma referência
System.out.println("i1 == i2 ? " + (i1 == i2));
// False, pois possuem referências diferentes
System.out.println("i3 == i4 ? " + (i3 == i4));
```

- *Você pode ver a implementação do método `equals` no código fonte da classe `Integer`*
- *Você pode ver também a implementação do "cache" dos números entre -128 e 127 (classe interna `IntegerCache`)*

Comentários Javadoc

Horácio Fernandes
horacio@icomp.ufam.edu.br

Slides: http://tinyurl.com/slides-tp

Comentários Javadoc
Introdução

- Javadoc permite incluir a documentação do seu sistema diretamente no código dele
- Um aplicativo, chamado `javadoc`, lê os arquivos Java do sistema e gera uma documentação completa do mesmo, em geral no formato HTML
- Comentários Javadoc:
  - O compilador java ignora os comentários de documentação (da mesma forma que ignora os comentários normais

```
/**
 * Classe Livro - Representa um livro na aplicação
 * @author Horacio Fernandes &lt;horacio.fernandes@gmail.com&gt;
 * @version 1.20, 2015-10-21
 */
public class Livro {
  // Declaração da classe ..
}
```

Comentários Javadoc
Exemplo Completo

```java
/**
 * Classe Livro - Representa um livro na aplicação
 * @author Horacio Fernandes &lt;horacio.fernandes@gmail.com&gt;
 * @version 1.20, 2015-10-21
 */
public class Livro {
    /** Nome do autor */
    String autor;
    String nome, editora;
    int anoPublicacao;

    /**
     * Construtor da classe.
     * @param autor autor do livro
     * @param nome nome do autor do livro
     * @param editora editora do livro
     * @param anoPublicacao ano de publicação
     */
    public Livro(String autor, String nome, String editora, int anoPublicacao) {
        this.autor = autor;
        this.nome = nome;
        this.editora = editora;
        this.anoPublicacao = anoPublicacao;
    }

    /**
     * Pega o autor do livro
     * @return String autor do livro
     */
    public String getAutor() {
        return autor;
    }

    // Continuação da classe ..
}
```

Documentação da classe

Documentação dos atributos

Documentação dos construtores

Documentação dos métodos

Comentários Javadoc
Geração da Documentação

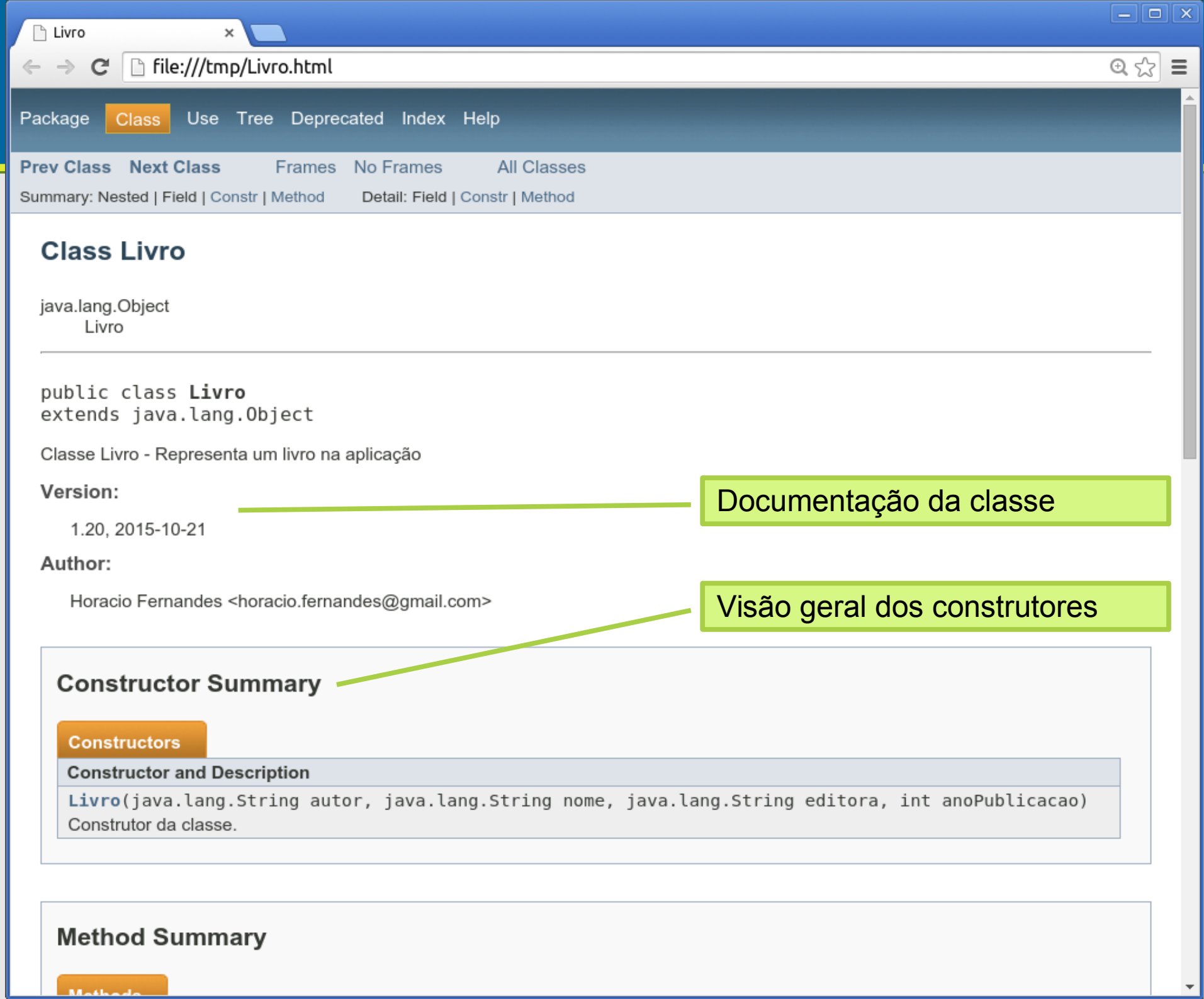
- Uma vez que o código esteja documentado, executa-se o comando `javadoc` para gerar a documentação

```
$ javadoc -charset utf-8 Livro.java
```

- No eclipse, a documentação do sistema todo pode ser gerada
  - Project → Generate Javadoc

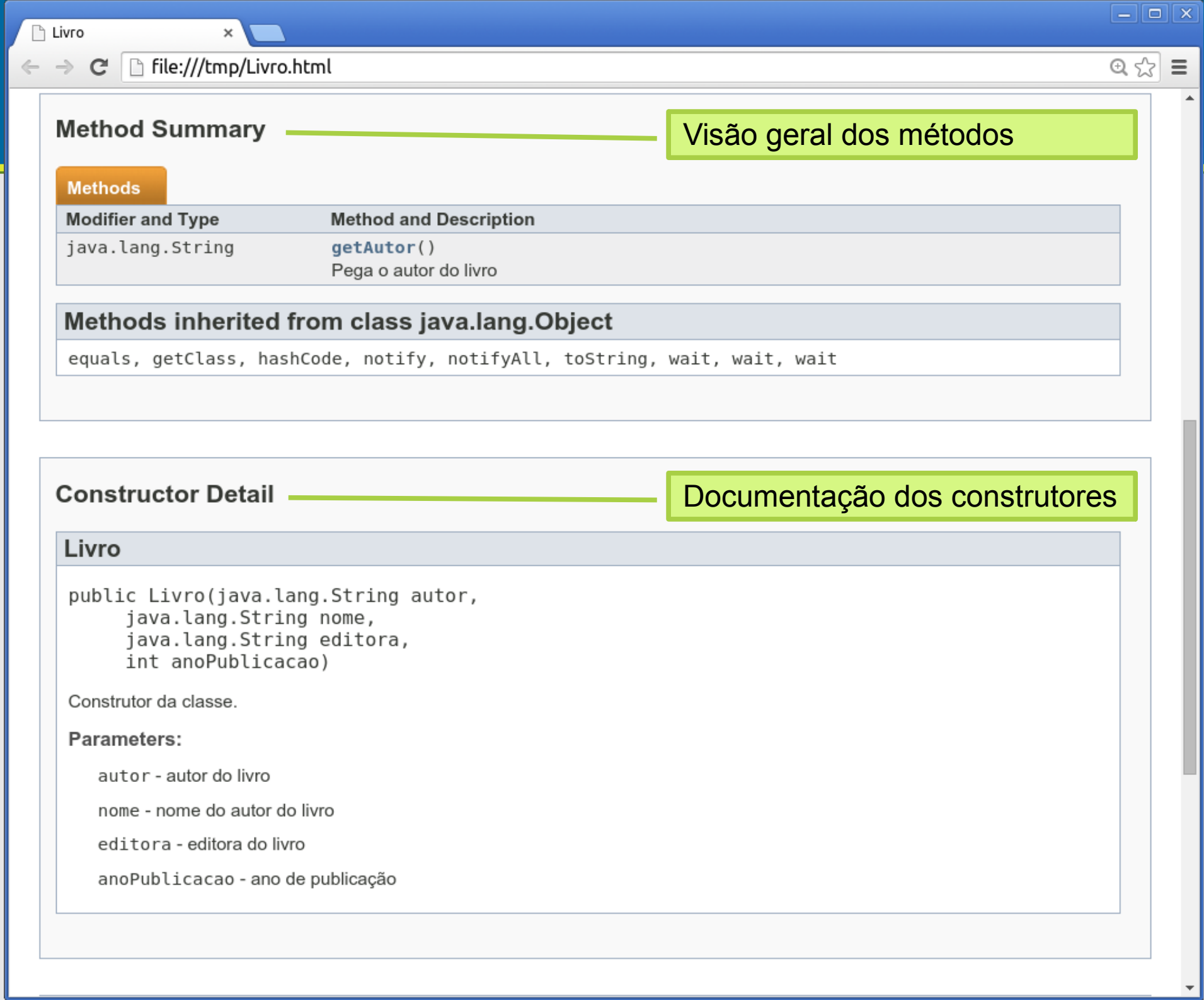
Comentários Javadoc

Resultado do Javadoc I



Comentários Javadoc

Resultado do Javadoc II



Comentários Javadoc
Resultado do Javadoc III

Livro

file:///tmp/Livro.html

```
java.lang.String nome,
java.lang.String editora,
int anoPublicacao)
```

Construtor da classe.

Parameters:

`autor` - autor do livro

`nome` - nome do autor do livro

`editora` - editora do livro

`anoPublicacao` - ano de publicação

Method Detail

Documentação dos métodos

getAutor

```
public java.lang.String getAutor()
```

Pega o autor do livro

Returns:

String autor do livro

Comentários Javadoc
Documentação do Java

- Toda a documentação da API da própria linguagem Java está em formato Javadoc e pode ser facilmente acessado na Internet:
  - No exemplo abaixo, está a documentação da classe `String`

String (Java Platform SE

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

Java™ Platform Standard Ed. 8

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3

java.lang

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

Serializable, CharSequence, Comparable<String>

Abra o código fonte da classe `String`, e compare com o seu Javadoc!

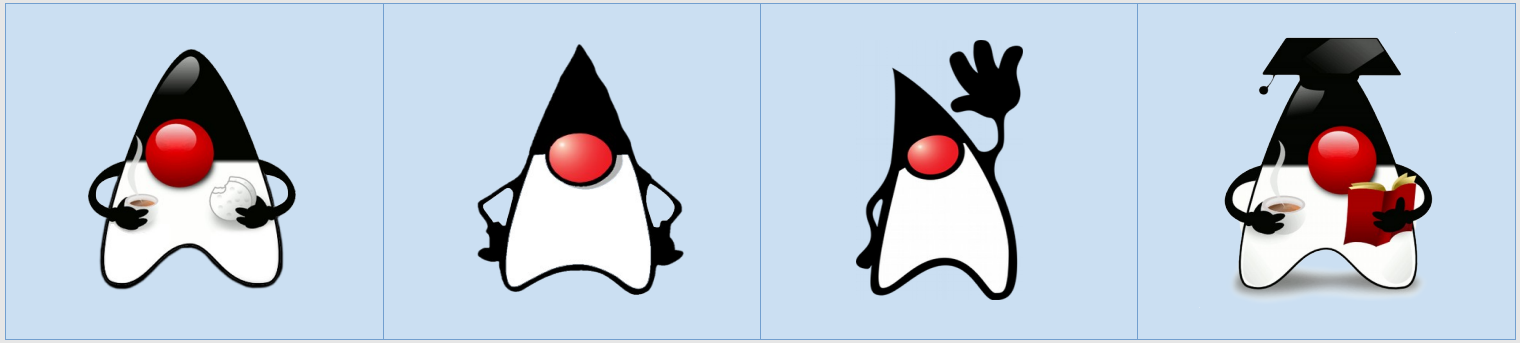
```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class.

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

Vetores em Java

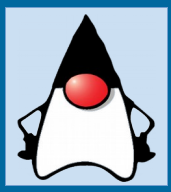


Instituto de Computação

Horácio Fernandes
horacio@icomp.ufam.edu.br

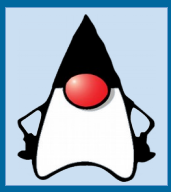
Slides: http://tinyurl.com/slides-tp

Vetores em Java
Introdução



- Em Java, vetores são objetos
  - São criados dinamicamente e alocados em tempo de execução
- Armazenam dados do mesmo tipo
  - Que pode ser de um tipo primitivo (`int`, `float`, etc)
  - Ou de um tipo referência/classe (`String`, `Circulo`, etc)
- Entretanto, não existe uma classe específica para vetores
  - Um “tipo” da classe vetor é referenciado pelo tipo de dado que o vetor armazena, seguido dos colchetes [ ]

Vetores em Java
Declaração e Instanciação



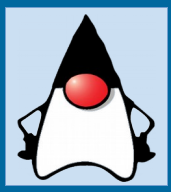
- Declaração de Vetores
  - Os colchetes podem vir depois do tipo ou depois do nome da variável

```
int[]      matriculas; // Vetor de inteiros
int        aulas[];    // Vetor de inteiros
float[][] notas;       // Vetor de vetor de floats (matriz)
String     args[];     // Vetor de objetos da classe String
Circulo[] circulos;    // Vetor de objetos da classe Circulo
```

- Instanciação de Vetores
  - Declarar um vetor não reserva espaço na memória
  - Isso é feito apenas na hora da instanciação
  - Na instanciação, o tamanho do vetor é definido
    - *Uma vez definido, o tamanho não pode ser modificado*

```
matriculas = new int[42];
circulos   = new Circulo[3];
```

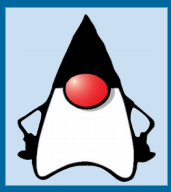
Vetores em Java
Declaração com Atribuição



- Pode-se declarar vetores já atribuindo seus elementos:

```
int[] factorial = { 1, 1, 2, 6, 24, 120, 720, 5040 };

char ac[] = { 'n', 'o', 't', ' ', 'a', ' ',
              'S', 't', 'r', 'i', 'n', 'g' };

String[] aas = { "array", "of", "String", };
```

Vetores em Java
Acessando Dados do Vetor



- Acessando os dados do vetor
  - Igual à linguagem C
  - O índice começa em 0 (zero) e vai até o tamanho do vetor – 1

```
matriculas[0] = 24601;
circulos[2] = new Circulo();

System.out.println(matriculas[0]);
System.out.println(circulos[2].raio);
```

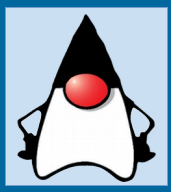
```
24601
0.0
```

  - Acessar um elemento fora dos limites de um vetor resulta em um erro em tempo de execução

```
matriculas[20] = 31337;
```

```
Exception in thread "main"
   java.lang.ArrayIndexOutOfBoundsException: 20
   at Principal.main(Principal.java:64)
```

Vetores em Java
Tamanho de um Vetor



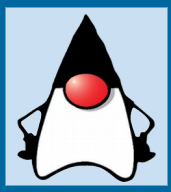
- Todo vetor (que é um objeto) possui um atributo chamado `length` que armazena o tamanho máximo do vetor
  - Nota: é o tamanho máximo do vetor e não a “quantidade” de elementos armazenados/atribuídos

```
System.out.println(matriculas.length);
```

```
20
```

  - Portanto, diferentemente de C, não precisamos criar uma constante para armazenar o tamanho máximo do vetor

Vetores em Java
Vetor em Métodos



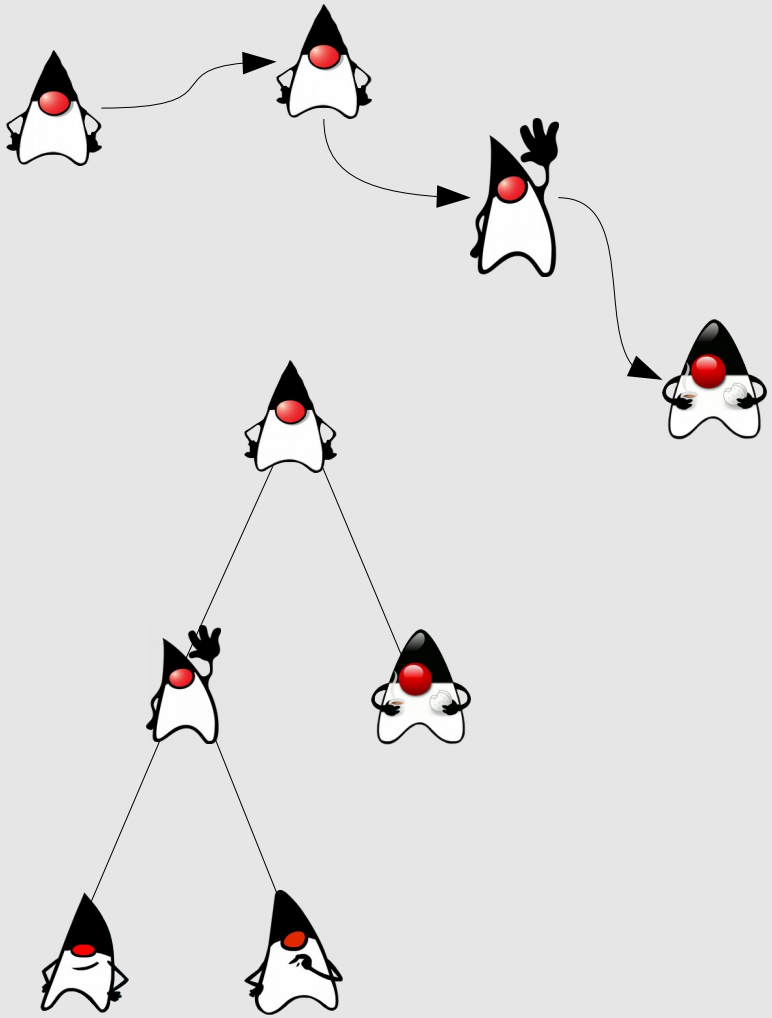
- Um método pode ter um vetor como parâmetro
  - Esse vetor é passado por referência (como todo objeto)
  - Mudar seus valores no método, muda os valores do vetor original
- Um método pode retornar um vetor
  - É retornada a referência para o vetor (que pode ter sido criado dentro do método)

```
/**
 * Duplica um vetor e incrementa os valores.
 */
public int[] incrNumeros(int numeros[]) {
  int[] result = new int[numeros.length];

  for (int i=0; i<resultado.length; i++)
    result[i] = numeros[i] + 1;

  return result;
}
```

Cria um novo vetor com o mesmo tamanho do vetor passado como parâmetro

Preenche o novo vetor com os valores

Retorna o novo vetor



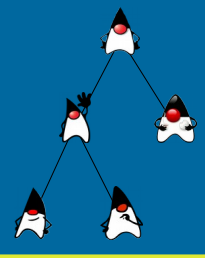
Generic Collections

ICOMP
Instituto de Computação

Horácio Fernandes
horacio@icomp.ufam.edu.br

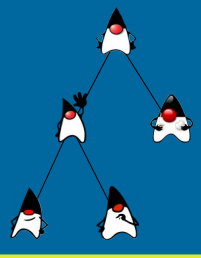
Slides: http://tinyurl.com/slides-tp

Generic Collections
Introdução

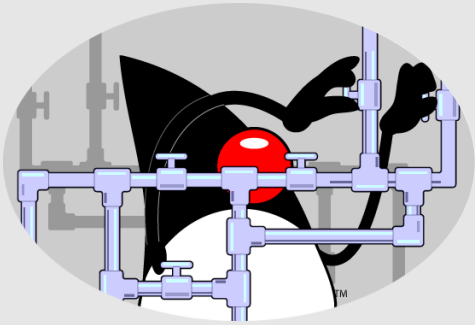


- *Generic Collections* (*Java Collections Framework*)
  - Java possui uma série de implementações de estruturas de dados prontas para serem utilizadas.
  - Em java, um *collection* é uma estrutura de dados que armazena referências para objetos.
    - *Elas usam “Classes e Métodos Genéricos”*
      - *Permitem definir o “tipo exato” de dado armazenado na hora da declaração*
      - *Permitem verificações de tipo em tempo de compilação*
      - *Classes genéricas é um assunto um pouco mais complexo, que não será tratado no curso, mas mostraremos como utilizá-las*

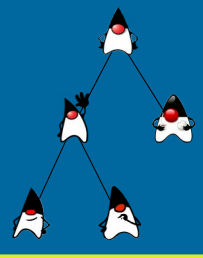
Código Fonte



- O Código Fonte do Java
  - Grande parte do código fonte do Java é livre
    - *GNU GPL v2*
  - Grande parte das bibliotecas, incluindo o *Java Collections Framework* são implementadas na própria linguagem Java e seus códigos fonte podem ser acessados no diretório da disciplina ou no seguinte endereço:
    - *http://hg.openjdk.java.net/*
      - *Página principal*
    - *http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java*
      - *Código fonte das bibliotecas (Java 8)*
  - Todas as classes mencionadas adiante (`ArrayList`, `LinkedList`, `Stack`, `PriorityQueue`, etc) estão implementadas dentro do diretório “util” do código fonte do Java.
    - *Acesse os códigos fonte dessas classes, e compare com os feitos em AED1/AED2! Você irá notar que não há muitas diferenças!*

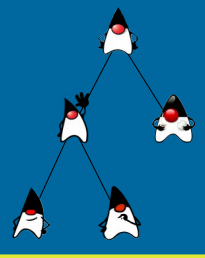


Listas Baseadas em Vetores



- Classe `ArrayList` e Classe `Vector`
  - As duas classes implementam listas usando vetores.
  - Apesar de poder ter um tamanho inicial, este tamanho é aumentado automaticamente quando necessário (lista de tamanho variável).
    - *Nota: por ser internamente implementado usando vetores (que não aumentam de tamanho), aumentar o tamanho da lista tem um custo grande, pois um novo vetor é criado e o conteúdo do anterior é copiado para o atual.*
  - Principais métodos:
    - `int size()` – *Retorna o tamanho da lista*
    - `boolean add(E e)` – *Adiciona um elemento no final da lista*
    - `void add(int index, E e)` – *Insere o elemento na posição especificada*
    - `int indexOf(Object o)` – *Busca o elemento, retornando seu índice*
    - `boolean isEmpty()` – *Retorna se a lista está vazia*
    - `iterator<E> iterator()` – *Retorna um objeto iterator que permite caminhar sequencialmente na lista*
    - `boolean remove(int index)` – *Remove um elemento pelo seu índice*
    - `boolean remove(Object o)` – *Remove um elemento pela sua referência*

Listas Baseadas em Vetores – Exemplo



- Exemplo de uso do `ArrayList`

```
import java.util.*;

public class ListaJava {

  public static void main(String args[]) {

    ArrayList<String> mestres = new ArrayList<String>();
    mestres.add("Obi-Wan Kenobi");
    mestres.add("Qui-Gon Jinn");
    mestres.add("Yoda");

    Iterator<String> iterator = mestres.iterator();
    while (iterator.hasNext()) {
      System.out.println(iterator.next());
    }
  }
}
```

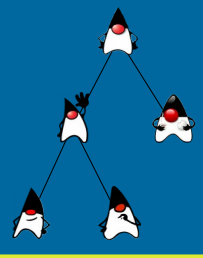
Tipo usado na lista genérica

Usado para iterar nos elementos da lista

Imprime o elemento atual

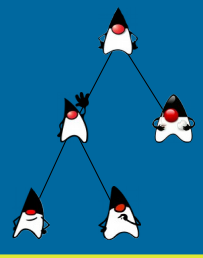
```
$ javac ListaJava.java
$ java ListaJava
Obi-Wan Kenobi
Qui-Gon Jinn
Yoda
```

Lista Encadeada



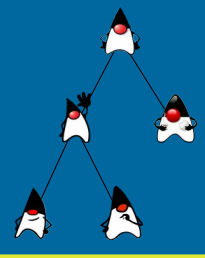
- Classe `LinkedList`
  - Implementa uma lista duplamente encadeada
  - Principais métodos:
    - *Todos citados no `ArrayList` com alguns métodos a mais*
    - `E getFirst()` – *Retorna o primeiro elemento da lista*
    - `E getLast()` – *Retorna o último elemento da lista*
    - `void addFirst(E e)` – *Insere um elemento no início da lista*
    - `void addLast(E e)` – *Insere um elemento no final da lista*
    - `E removeFirst()` – *Remove o primeiro elemento da lista*
    - `E removeLast()` – *Remove o último elemento da lista*

Pilha



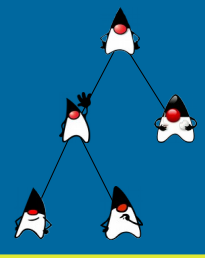
- Classe `Stack`
  - Implementa uma pilha usando um vetor de tamanho variável
  - Principais métodos:
    - `boolean empty()` – *Testa se a pilha está vazia*
    - `E push(E item)` – *Adiciona um item no topo da pilha*
    - `E pop()` – *Remove e retorna o elemento no topo da pilha*
    - `E peek()` – *Retorna o elemento no topo da pilha sem removê-lo*
  - *OBS: a classe `LikedList` (slide anterior) também possui os métodos acima, permitindo a criação de Pilhas usando Listas Encadeadas*

Fila



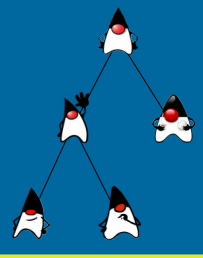
- Interface `Queue`
  - Não existe uma classe em Java para filas. Existe uma interface (`Queue`) que obriga a implementação das operações usadas em filas
  - Principais métodos da interface:
    - `boolean add()` – *Adiciona um elemento no final da fila*
    - `E remove()` – *Remove e retorna o primeiro elemento da fila*
    - `E peek()` – *Retorna o elemento no topo da pilha sem removê-lo*
  - *OBS: Como a classe `LinkedList` implementa a interface `Queue`, a primeira pode usada como uma "Fila implementada por Lista Encadeada"*
- Classe `PriorityQueue`
  - Implementa uma Fila com Prioridades, inserindo o elemento na sua ordem de acordo com o seu conteúdo ou de acordo com um método de comparação.

Tabelas Hash



- Classe `Hashtable`
  - Em Java, Tabelas Hash são implementadas através da classe `Hashtable`
    - *Por ser uma coleção genérica (generic collection), é necessário especificar os tipos de dados (da chave e dos dados) que serão utilizados*
    - *Principais métodos:*
      - `V put(K key, V value)` – *insere um valor com uma determinada chave*
      - `V get(Object key)` – *busca um elemento pela chave*
      - `V remove(Object key)` – *remove um elemento com determinada chave*
      - `int size()` – *quantidade de elementos*
      - `Enumeration<V> elements()` – *retorna uma enumeração dos elementos*
  - A classe `Hashtable` usa *Tabelas Hash com Encadeamento* (listas encadeadas) para armazenar os valores.
    - *Entretanto, quando a tabela atinge um certo fator de uso (loadFactor), indicando que a tabela está ficando cheia (e muitas colisões irão ocorrer), o tamanho da tabela é automaticamente incrementado e todos os elementos são reajustados na tabela. Isso é conhecido como rehash e o fator de uso normalmente é de 75%.*

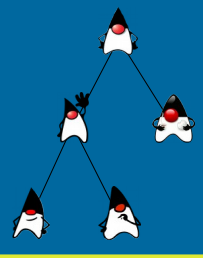
Tabelas Hash – Exemplo



```
import java.util.*;

public class HashJava {

  public static void main(String args[]) {

    Hashtable<String, Integer> mestres = new Hashtable<String, Integer>();

    mestres.put("Obi-Wan Kenobi", 57);
    mestres.put("Qui-Gon Jinn", 92);
    mestres.put("Yoda", 896);

    Integer n = mestres.get("Yoda");

    if (n != null)
      System.out.println("Nascimento de Yoda: " + n + " BBY");

  }

}
```

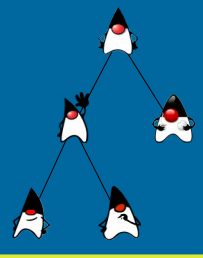
Tabela Hash em que as chaves são strings e valores são inteiros

Árvores



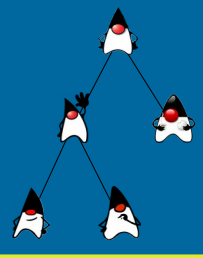
- Classe `TreeMap`
  - Em Java, Árvores são implementadas através da classe `TreeMap`
    - *Por ser uma coleção genérica (generic collection), é necessário especificar os tipos de dados (da chave e dos dados) que serão utilizados*
    - *Principais métodos:*
      - `V put(K key, V value)` – *insere um valor com uma determinada chave*
      - `V get(Object key)` – *busca um elemento pela chave*
      - `V remove(Object key)` – *remove um elemento com determinada chave*
      - `int size()` – *quantidade de elementos*
      - `Set<Map.Entry<K,V>> entrySet()` – *retorna o conjunto de elementos da árvore em ordem ascendente*
      - `NavigableMap<K,V> descendingMap()` – *retorna o conjunto de elementos da árvore em ordem decrescente*
      - `Map.Entry<K,V> firstEntry()` – *retorna o menor elemento*
      - `Map.Entry<K,V> lastEntry()` – *retorna o maior elemento*
      - `Map.Entry<K,V> lowerEntry(K key)` – *retorna o predecessor*
      - `Map.Entry<K,V> higherEntry(K key)` – *retorna o sucessor*

Árvores – Implementação

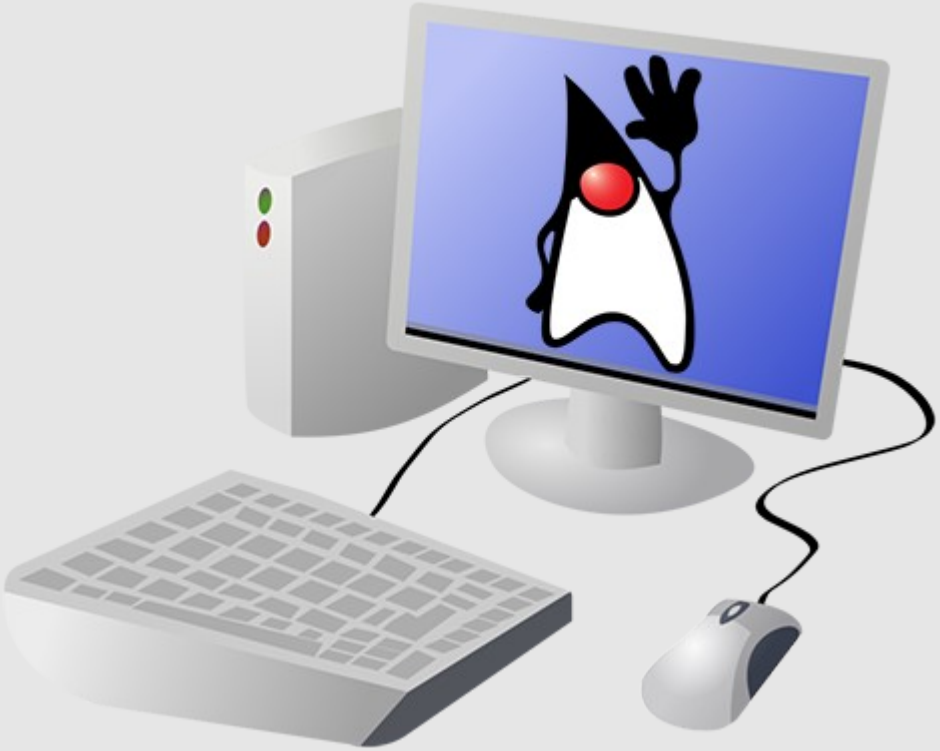


- A classe `TreeMap` usa uma *Árvore Vermelho-Preto* para armazenar os valores.
  - A implementação é baseada no livro do Cormen (Algoritmos, Teoria e Prática)
  - O código-fonte (`TreeMap.java`), pode ser acessado em
    - *`http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/util/TreeMap.java`*
    - *Note como o método de inserção (`put`) primeiro insere o nó como se fosse em uma árvore binária normal e, em seguida, executa o método para rebalancear a árvore (`fixAfterInsertion`)*

Generic Collections
Árvores – Exemplo



```
import java.util.*;
public class ArvoreJava {

  public static void main(String args[]) {

    TreeMap<Integer,String> mestres = new TreeMap<Integer,String>();

    mestres.put(57, "Obi-Wan Kenobi");
    mestres.put(92, "Qui-Gon Jinn");
    mestres.put(896, "Yoda");

    Iterator iterator = mestres.descendingMap().entrySet().iterator();

    System.out.println("Mestres ordenado por idade:");

    while (iterator.hasNext()) {
      Map.Entry<Integer,String> mestre =
                                (Map.Entry<Integer,String>) iterator.next();
      System.out.println("---> " + mestre.getValue() +
                         " tem " + mestre.getKey()   +   " anos");;
    }
  }
}
```



Entrada e Saída em java

ICOMP
Instituto de Computação

Horácio Fernandes
horacio@icomp.ufam.edu.br

Slides: http://tinyurl.com/slides-tp

Entrada e Saída
Fluxos

- Fluxo
  - Sequência de bytes
- Fluxos criados automaticamente
  - `System.out`: da classe `PrintStream`, é o objeto de fluxo de saída padrão
    - *Normalmente é a tela*
  - `System.in`: da classe `InputStream`, é o objeto de fluxo de entrada padrão
    - *Normalmente é o teclado*
  - `System.err`: da classe `PrintStream`, é o objeto de fluxo de saída de erro
    - *Normalmente é a tela também*

Entrada e Saída
Fluxos – Exemplo

```
import java.io.*;
public class TesteES {
  public static void main(String args[]) {
    try {
      int caractere = 0;
      String linha = "";
      while ( (caractere = System.in.read() ) != 10) {
        linha = linha + (char) caractere;
      }
      System.out.println("Linha: " + linha);
      System.err.println("Linha de erro de teste!");
    } catch (IOException e) {}
  }
}
```

Lê um caractere do teclado

Imprime na saída padrão

Imprime na saída de erro

```
$ java TesteES
lalalala
Linha: lalalala
Linha de erro de teste!
$ java TesteES 2> /dev/null
oioioioioi
Linha: oioioioioi
```

Entrada e Saída
Fluxos de Arquivos: Entrada

- Da mesma forma que lemos a partir do `System.in`, podemos ler a partir de um arquivo usando a classe `FileInputStream`

```
import java.io.*;

public class TesteArqEntrada {
  public static void main(String args[]) {
    try {

      FileInputStream arqEntrada = new FileInputStream("/etc/issue.net");
      int caractere = 0;
      String conteudo = "";
      while ( (caractere = arqEntrada.read() ) != -1) {
        conteudo = conteudo + (char) caractere;
      }
      System.out.println("Conteudo do arquivo:\n" + conteudo);
      arqEntrada.close();

    }
    catch (IOException e) {}
  }
}
```

Lê um caractere do arquivo

Imprime o conteúdo

Entrada e Saída
Fluxos de Arquivos: Saída

- Da mesma forma que escrevemos no `System.out`, podemos escrever em um arquivo usando a classe `FileOutputStream`

```
import java.io.*;

public class TesteArqSaida {

  public static void main(String args[]) {
    try {

      String conteudo = "Teste de Saída !!\n";
      FileOutputStream arqSaida = new FileOutputStream("/tmp/Teste.txt");
      arqSaida.write(conteudo.getBytes());
      arqSaida.close();

    }
    catch (IOException e) {}
  }

}
```

Abre o arquivo para saída

Escreve o conteúdo

Recursos do Java
Laboratório 3

- http://tinyurl.com/slides-tp
  - → Laboratórios
  - → TP – 3oLaboratorio.pdf
- Entrega por E-Mail
  - Para: horacio.fernandes@gmail.com
  - Cópia: moyses.lima@icomp.ufam.edu.br
  - Assunto: TP: 3o Lab
- Data Limite:
  - Hoje, às 12hs
  - E-Mails recebidos após 12hs não serão considerados