Introdução à Tecnologia Java™

riccioni@univali.br

Tecnologia Java™

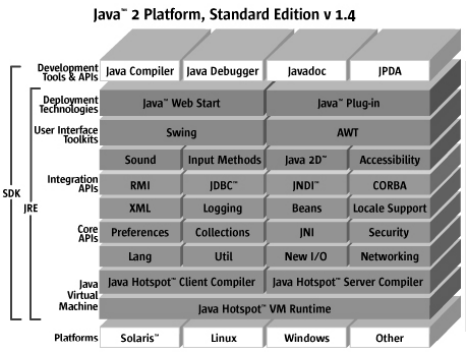
J2EE™ – Tecnologia corporativa com suporte a WEB Services, segurança e interoperabilidade em desenvolvimento

J2SE™ – Oferece ambiente para desenvolvimento de projetos de aplicações corporativas, incluindo a Linguagem de programação Java, compilador, ferramentas, APIs para escrever e desenvolver aplicações.

J2ME™ – oferece ambiente para desenvolvimento em smart cards, pager, wireless,...

O Nome Java™ 2

Nome	Descrição
Plataforma Java™ 2, Standard Edition, v. 1.4 (J2SE)	A plataforma **abstrata** Java™ 2 apenas uma especificação do que toda implementação (concreta) da plataforma deve oferecer.
Java™ 2 SDK, Standard Edition, v. 1.4 (J2SDK)	Produto da SUN que implementa a plataforma J2SE. Kit de desenvolvimento que pode ser usado para construir aplicações para esta plataforma. Inclui tanto o compilador quanto a máquina virtual e a API do Java para a plataforma J2SE.

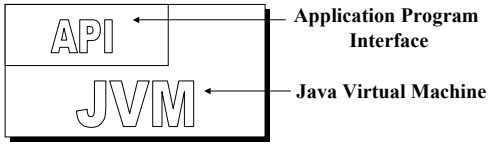


Tecnologia Java™

- Linguagem de programação Java™

```
package principal;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class FrameConsultar extends JIntern

  private JButton botaoOK;
  private JButton botaoCancelar;
```

- Plataforma Java™

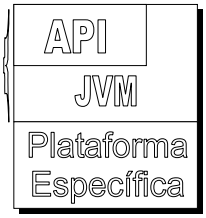


Linguagem Java™

- **Simples**
- **Orientada a Objetos**
- **Distribuída**
- **Robusta (compilação rigorosa)**
  - Fortemente tipada
  - Elimina vários erros de alocação de objetos
- **Portável**
- **Multithreaded(múltiplas linhas de execução)**
- **Compilada E Interpretada**
- **Neutra em relação a arquitetura**
- **Segura**
- **Alto desempenho**
- **Dinâmica**

Plataforma Java™

- Uma plataforma em geral combina software (sistema operacional) + hardware.
- A plataforma Java difere das demais, pois inclui apenas software:
  - Java API e
  - Máquina Virtual Java (JVM)



API

- Formada por vários pacotes de classes, p.ex:
  - java.lang
    - classes fundamentais.
  - java.io
    - classes relacionadas à entrada/saída.
  - java.net
    - classes relacionadas à comunicação de dados.
  - java.awt e javax.swing
    - classes relacionadas à interface com o usuário.
  - java.util
    - classes de estruturas de dados e outros.
  - java.sql
    - classes para acesso a banco de dados.

A Linguagem Java™

- Sintaxe e palavras-reservadas semelhantes as da linguagem C++.

```
5       // ...
6
7       Pessoa a[];
8       a = new Pessoa[20];
9       for(int i=0; i<20; i++) {
10          nome = in.readLine();
11          a[i] = new Pessoa(nome, new Date());
12      }
13
14      // ...
15
16      int i=0;
17      while(i < 20) {
18          a[i] = null;
19          i++;
20      }
21
22      i=0;
23      do {
24          // ...
25      } while(i < 20);
26  }
```

Tipos de Programa Java (Standard)

- Application
  - Equivalente a uma aplicação comum (como fazemos em outras linguagens).
  - O usuário tem a iniciativa de executá-lo.
- Applet
  - Utiliza a máquina virtual de um browser.
  - É executado automaticamente (ao visitar uma página web).
  - Portanto, tem restrições de segurança.
- Java Beans
  - Seguem uma especificação para construção de componentes.

Tipos de Programa Java (Enterprise)

- São executados em servidor:
  - JSP (Java Server Pages)
    - "Equivalente" a PHP e ASP, mas em Java.
    - Representam views no modelo MVC.
  - Servlet
    - Representa o controle no modelo MVC.
  - Enterprise Java Beans
    - Permitem naturalmente encapsular sessão, sincronização, segurança, connection pooling, multi-threading e outros detalhes importantes para o desenvolvimento de aplicações.

Modelo MVC (Model-View-Controller)

- Não deixe um objeto responsável por muito!
- **M**odel
  - Representa o conteúdo. (Java Beans)
- **V**iew
  - Define a apresentação. (JSP)
- **C**ontrol
  - Define reações para controlar eventos. (Servlets)

Model x View

- Um modelo:

Disciplina
nome
ementa
cargaHoraria

- Duas views:

Nome: .....
C/H
Ementa: .....
.....
Carga Horária: .....

Nome Ementa
....

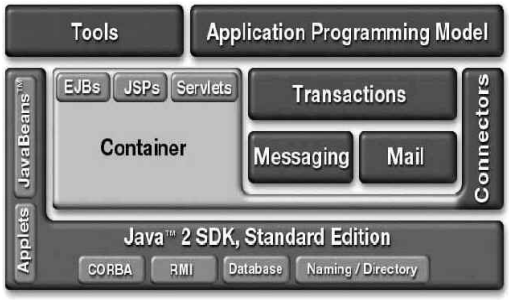
Framework para Desenvolvimento

- Um framework pré-define total ou parcialmente a *estrutura* de uma aplicação.
- Permite desenvolvedores utilizarem design patterns já implementados como o MVC.
- Estamos nos baseando no framework STRUTS:
  *"The **Struts** Web Application **Framework**"*
  http://jakarta.apache.org/struts/

Nesta Disciplina

- Linguagem Java
- Plataforma Java 2 Standard Edition (J2SE)
- Programação de aplicações Java.
- Simulação do STRUTS em ambiente standalone.
  - Programação *standard*, mas pensando em WEB.

Visão Geral



Application

- **Uma classe que possui um método `main`.**

`NomeDaClasse.java` (arquivo)

```
public class NomeDaClasse {

  public static void main(String args[]) {

    // programa principal ...

  }
}
```

- **Ponto de entrada do programa: `main`**
- **Passagem de parâmetros: args[]**
- **Palavras-Reservadas:**
  - **public, class, static, void.**

Application (primeiro exemplo)

`PrimeiroExemplo.java`

```
public class PrimeiroExemplo {

  public static void main(String args[]) {

    System.out.println("Não vou dizer Hello World!");

  }
}
```

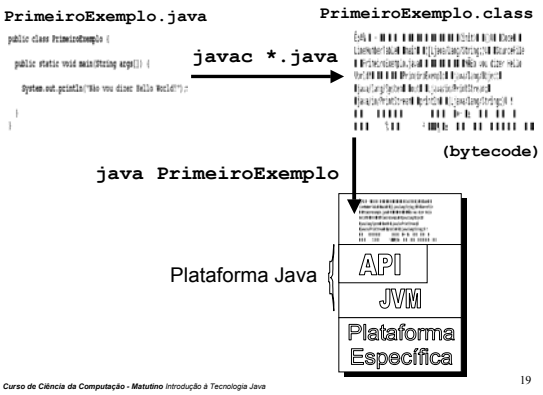
- **Compilação (produz PrimeiroExemplo.class)**

```
javac *.java
```

- **Execução:**

```
java PrimeiroExemplo
```

Application (Compilação e Execução)



Tipos Fundamentais

- Inteiros:

int	4 bytes	-2bi até 2bi
short	2 bytes	-32768 até 32767
long	8 bytes	-9E+18 até 9E+18
byte	1 byte	-128 até 127

Em Java não existe unsigned!

Tipos Fundamentais

- Reais (Ponto-Flutuante):

float	4 bytes	6 dígitos decimais significantes
double	8 bytes	15 dígitos decimais significantes

- Lógicos
  - **`boolean`**
  - palavras-reservadas: **`true`** ou **`false`**
  - impossível converter de boolean para int!

Tipos Fundamentais

- Caracteres
  - Denotados por aspas simples: ‘a’, ‘b’, ‘c’, ‘1’, ‘2’,...
  - **`char`** denota o tipo caracter em Java
  - Representação em Unicode: (0 a 65535)
    - Extensão do ISO 8859-1 ou Latin-1 (apenas 256)
    - caracteres estão na faixa de ‘\u0000’ a ‘\uFFFF’.
    - \u denota um caracter Unicode.
    - Para os 256 primeiros, podemos usar ‘a’,’b’,’c’,...
    - Para os demais, utilizamos a notação ‘\uXXXX’.
    - p. ex. o caracter ™ é denotado por ‘\u2122’.

Hexadecimal!

Tipos Fundamentais

- Caracteres Especiais:

‘\b’	backspace	‘\u0008’
‘\t’	tab	‘\u0009’
‘\n’	nova linha	‘\u000a’
‘\r’	retorno	‘\u000d’
‘\”’	aspas duplas	‘\u0022’
‘\’’	aspas simples	‘\u0027’
‘\\’	barra	‘\u005c’

Operadores Lógicos

- Não há palavras-reservadas (idênticos a C++):

Operador	Significado
&&	E
||	OU
!	NÃO

Operadores Relacionais

- Idênticos aos do C++:

Operador	Significado
==	igual a
!=	diferente de
<	menor que
>	maior que

Estruturas Fundamentais

- Repetição:
  - for(...) {...}
  - while(...) {...}
  - do {...} while(...);
- Desvio:
  - if(...) {...} else {...}
  - switch(...) { case ... : ...; break; ... }

Exemplos

- for

```
int i;

for(i=0; i<10; i++)
{
    System.out.println("exemplo");
}
```

ou (declarando variável interna)

```
for(int i=0; i<10; i++)
{
    System.out.println("exemplo");
}
```

Exemplos

- if

```
int i = 10, j = 20;

if(i == j) {
    System.out.println("iguais");
} else {
    System.out.println("diferentes");
}


int i = 10, j = 20;

if((i != j) && (i < 15)) {
    System.out.println("diferentes e i < 15");
} else {
    System.out.println("iguais ou i >= 15");
}
```


Exemplos

- switch-case

```
char opcao;

// ... obtém opção do usuário.

switch(opcao) {
    case 'I' : // incluir...
               System.out.println("incluído.");
               break;

    case 'E' : // excluir...
               System.out.println("excluído.");
               break;

    default  : System.out.println("opção inválida!");
}
```

  - opção deve ser char ou int (valor ordinal!)

Atribuição e Inicialização

- Toda variável deve ser inicializada:
  - Exemplo: não podemos imprimir o valor de i

```
int i;

System.out.println(i);
```

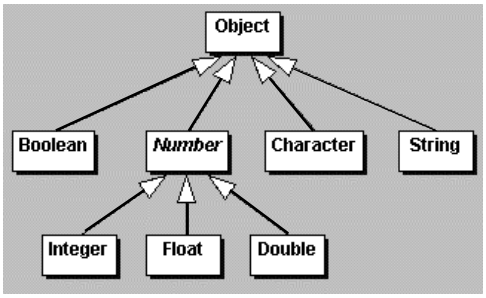
Erro: variable i might not have been initialized!

  - Correto:

```
int i = 0;

System.out.println(i);
```

Classes Wrapper

- Substituem tipos primitivos:



(estão no pacote java.lang)

A Classe String

- Qualquer sequência de caracteres entre aspas é uma instância desta classe.

```
String a = "";  // string vazia.
String b;
b = "01234";  // string de 5 caracteres.

String teste;  // não inicializada.

System.out.println(teste);

Erro: variable teste might not have been
initialized!
```


Concatenação de Strings

```
String a = "Ja";
String b = "va!";
String c = a + b;   // "Java!"

String c = "Java " + 2; // "Java 2"
```

- **concatenação é associativa à esquerda!**

```
String s;
s = "Java " + 1 + 1;    // "Java 11", mas
s = "Java " + (1 + 1);  // "Java 2"
```

Substrings

```
String a = "Java!";
String b = a.substring(0, 4);   // "Java"
```

posição inicial da substring,

primeira posição que **não** será copiada

Comparando Strings

```
String a = "Java";
String b = "Java";
```


```
if(a == b) // incorreto!
   System.out.println("Talvez!");

if(a.equals(b))
   System.out.println("Ok!");
```


Métodos de java.lang.String

- **public char** charAt(**int** index)
  - retorna o caracter da posição especificada.
- **public int** compareTo(String other)
  - retorna um valor negativo, zero ou positivo se a outra string está antes, é igual ou vem depois na ordem alfabética, respectivamente.
- **public boolean** endsWith(String suffix)
  - retorna true se a string termina com o sufixo especificado.
- **public boolean** equals(Object anObject)
  - retorna true se a string é igual a outra.

Métodos de java.lang.String

public boolean `equalsIgnoreCase(String other)`

- retorna true se a string é igual a outra desprezando diferenças de maiúsculas/minúsculas.

public int `indexOf(String str)` ou
public int `indexOf(String str, int fromIndex)`

- retorna a posição inicial da primeira substring igual a str, começando no índice 0 ou em fromIndex.

public int `lastIndexOf(String str)` ou
public int `lastIndexOf(String str, int fromIndex)`

- retorna a posição inicial da última substring igual a str, começando no índice 0 ou em fromIndex.

Métodos de java.lang.String

- **public int** `length()`
  - retorna o tamanho da string.
- **public** `String replace(char oldChar, char newChar)`
  - retorna uma nova string obtida pela substituição de todo caracter oldChar pelo caracter newChar.
- **public boolean** `startsWith(String prefix)`
  - retorna **true** se a string inicia com `prefix`.
- **public** `String substring(int beginIndex)`
  ou
  public `String substring(int beginIndex, int endIndex)`
  - retorna uma nova string formada pela seqüência de caracteres de beginIndex até, mas não incluindo, endIndex.

Métodos de java.lang.String

- **public** `String toLowerCase()`
  - retorna uma nova string formada pela mesma sequência de caracteres original, mas com todas as letras minúsculas.
- **public** `String toUpperCase()`
  - retorna uma nova string formada pela mesma sequência de caracteres original, mas com todas as letras maiúsculas.
- **public** `String trim()`
  - retorna uma nova string formada pela mesma sequência de caracteres original, exceto os espaços em branco no início e no final da string.

As Classes Integer e Double

- Convertendo de String para int. Exemplo:

```
int n = Integer.parseInt("20");
for(i=0; i < n; i++) {
    // ...
}
```

- Convertendo de String para double:

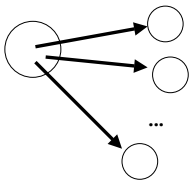
```
double nota = Double.parseDouble("9.5");
nota = nota + 0.5;
// ...
```


Orientação a Objetos em Java™

- Herança
  - Apenas herança simples (~~herança múltipla~~)
  - Conceito de **interface**
- Encapsulamento
  - Modificadores de acesso: **private**/**protected**/**public**
  - Pacote: visibilidade também aplicada a classes
- Polimorfismo
  - Automaticamente (sem palavras-reservadas)
  - Conceitualmente correto (Java é fortemente tipada)

Orientação a Objetos em Java™

- Associação e Agregação
  - Implementadas como em C++ ou Delphi.
  - Ponto crítico para o gerenciamento de memória efetuado pelo coletor de lixo!



Classes e Objetos

- Criando um objeto: operador **new**.

```
objeto = new NomeDaClasse(...);
```

Construtor possui o mesmo nome da classe!

Parâmetros, se houver.

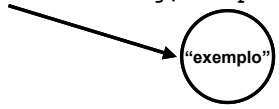
- Exemplo:

```
String s;
s = new String("exemplo");
```


Classes e Objetos

- Destruindo um objeto:
  - Não existe destrutor como em outras linguagens.
  - Garbage Collector (Coletor de lixo): um objeto é destruído tão logo não seja mais referenciado:

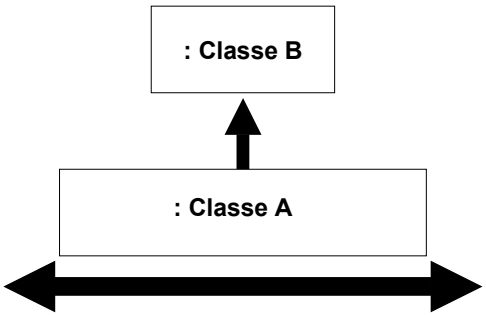
```
String s;
s = new String("exemplo");
```




```
s = null;
   // pode ser destruído caso não haja outra referência!
```


Vazamento de Memória

- Sem uso mas ainda referenciado!

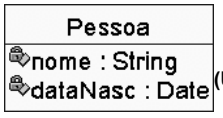


Encapsulamento

- Modificadores de visibilidade:

	Atributo ou Método	Classe
private	invisível para o mundo exceto para a classe	idem para classes internas
protected (default)	visível no pacote	
public	visível para o mundo	

Encapsulamento



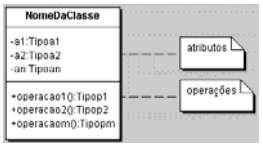
(UML)

```
public class Pessoa {

   private String nome;
   private Date dataNasc;

   //...
}
```

(Java)

Implementando Diagramas de Classes

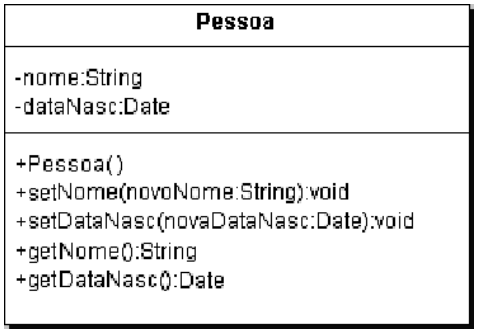
- Classe em UML ⟶ Classe em Java




```
public class NomeDaClasse {
    private Tipoa1 a1;
    private Tipoa2 a2;
    private Tipoan an;

    public Tipop1 operacao1() {
    }

    public Tipop2 operacao2() {
    }

    public Tipopm operacaom() {
    }
}
```

Classe em UML ⟶ Classe em Java

- Exemplo:



Classe em UML ⟶ Classe em Java

```
public class Pessoa {
   private String nome;
   private Date dataNasc;

   public Pessoa() {
       nome = null;
       dataNasc = null;
   }

   public String getNome() {
       return nome;
   }

   public Date getDataNasc() {
       return dataNasc;
   }

   public void setNome(String nome) {
       this.nome = nome;
   }

   public void setDataNasc(Date dataNasc) {
       this.dataNasc = dataNasc;
   }
}
```


Implementando um Construtor

- Parâmetros servem para definir um estado inicial para o objeto:

```
public class Pessoa {
    private String nome;
    private Date dataNasc;

    public Pessoa(String novoNome, Date novaData) {
        nome = novoNome;
        dataNasc = novaData;
    }

    //...
}

Pessoa alguem;
alguem = new Pessoa("Ana", new Date(...));
```

Implementando Getters (Métodos de Acesso)

```
public class Pessoa {
   private String nome;
   private Date dataNasc;

   public String getNome() {
       return nome;
   }

   public Date getDataNasc() {
       return dataNasc;
   }

   //...
}
```

```
String s;
s = alguem.getNome();
```


Implementando Setters

```
public class Pessoa {
   private String nome;
   private Date dataNasc;

   //...

   public void setNome(String nome) {
       this.nome = nome;
   }

   public void setDataNasc(Date dataNasc) {
       this.dataNasc = dataNasc;
   }
}
```

```
alguem.setNome("Maria");
```

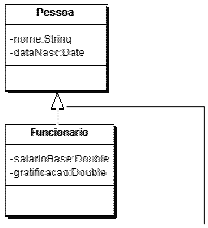

Variável de Classe

- Um atributo cujo valor é o mesmo para todos os objetos da classe. (**`static`**)
- Geralmente são constantes. (**`final`**)
- Exemplo: a idade mínima de uma pessoa.

```
public class Pessoa {
    private String nome;
    private Date dataNasc;

    public static final int IDADE_MINIMA = 0;

    //...
}
```

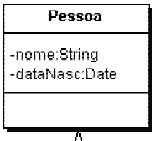
Herança em Java™

- Herança simples: uma classe possui exatamente uma superclasse.
- Exemplo: todo funcionário é uma pessoa.

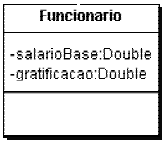



```
class Funcionario extends Pessoa
```


Herança em UML ⟶ Herança em Java™



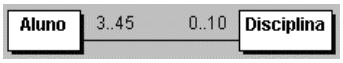

```
class Pessoa {

    private String nome;
    private Date dataNasc;

    // ...
}
```



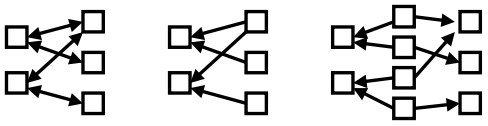

```
class Funcionario extends Pessoa {

    private Double salarioBase;
    private Double gratificacao;

    // ...
}
```


Implementando Diagramas de Classes

- Associação em UML

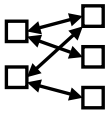


  - Há vários mapeamentos possíveis.
  - Cada um apresenta vantagens/desvantagens relacionadas a desempenho de processamento ou memória.



Associação em UML → Associação em Java

1. **Variáveis de instância em ambos os extremos**:



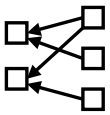
```
public class Aluno
{
  private Disciplina disciplinas[10];

  //...

}

public class Disciplina
{
  private Aluno alunos[45];

  //...
}
```

↑↑ Memória!

↓↓ Processamento

Associação em UML → Associação em Java

2. **Variáveis de instância em apenas um extremo**:



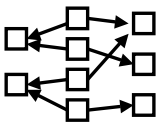
```
public class Aluno
{

  // sem referências.

  //...
}

public class Disciplina
{
  private Aluno alunos[45];

  //...
}
```

↑ Memória!

↓ Processamento

Associação em UML → Associação em Java

3. **Usando um objeto de associação:**



```
public class Aluno
{

  // sem referências.

  //...
}

public class Disciplina
{

  // sem referências.

  //...
}
```

↓ Memória

↓ Processamento

Associação em UML → Associação em Java

3. Usando um objeto de associação:

```
public class AssocAlunoDisciplina {
   private Aluno aluno;
   private Disciplina disciplina;

   public AssocAlunoDisciplina() {
       aluno = null;
       disciplina = null;
   }

   public Aluno getAluno() {
       return aluno;
   }

   public Disciplina getDisciplina() {
       return disciplina;
   }

   public void setAluno(Aluno aluno) {
       this.aluno = aluno;
   }

   public void setDisciplina(Disciplina disciplina) {
       this.disciplina = disciplina;
   }
}
```


Ferramenta javadoc

- Produz documentação HTML de uma classe.
- Por exemplo, a documentação em HTML da API do Java foi gerada utilizando javadoc.
- Os comentários inseridos no código fonte são extraídos para produzir um documento HTML.
- Comentários:

```
/**
   Bla bla <code>bla</code> bla ...
*/
```

devem preceder cada elemento da classe.

Comentários da Classe

- Devem ser inseridos imediatamente antes da definição da classe:

```
/**
 * Um objeto da classe <code>Pessoa</code>
 * representa alguém com nome e data de nascimento.
 */
public class Pessoa {
   private String nome;
   private Date dataNasc;

   //...
}
```

Comentários de Métodos

- Podemos utilizar tags adicionais para documentar parâmetros e valores de retorno:

```
public class Pessoa {
    //...

    /**
     * Obtém o nome da pessoa.
     * @return o nome da pessoa
     */
    public String getNome() {
        return nome;
    }

    /**
     * Altera o nome da pessoa.
     * @param nome o novo nome
     */
    public void setNome(String nome) {
        this.nome = nome;
    }
}
```


Outros Comentários

- Os atributos (privados) não fazem parte da documentação do usuário.
- Portanto, não é recomendável comentá-los na sintaxe utilizada pelo javadoc.
- Apenas as variáveis de classe públicas são importantes:

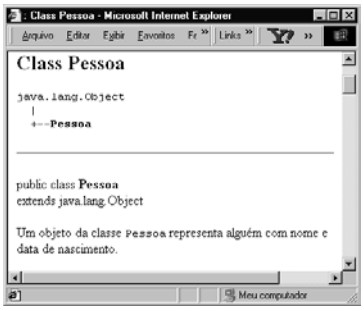
```
public class Pessoa {
    private String nome;
    private Date dataNasc;

    /**
     * A idade mínima de uma pessoa.
     */
    public static final int IDADE_MINIMA = 0;
}
```


Outras Tags

- Em comentários de classe:
  - @author nome
  - @version texto
- Em qualquer comentário:
  - @since texto
    - Ex: @since versão 2.1.3
  - @deprecated texto
    - Indica uma forma melhor de utilizar um método.
    - Ex: A classe Date foi praticamente substituída pela classe GregorianCalendar.
    - @deprecated Use <code>Calendar.get(...)</code>.

Compilando Comentários

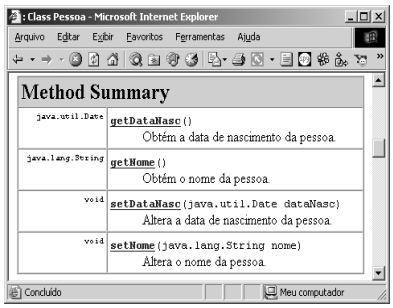
- javadoc Pessoa.java ou
- javadoc –author –version Pessoa.java



Pessoa.html

A Documentação em HTML

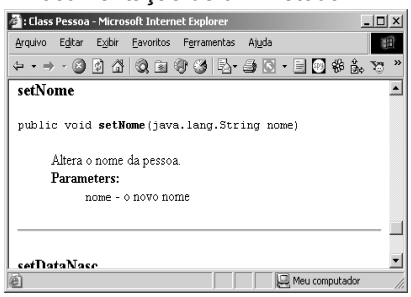
- Resumo dos métodos (fácil navegação):



Pessoa.html

A Documentação em HTML

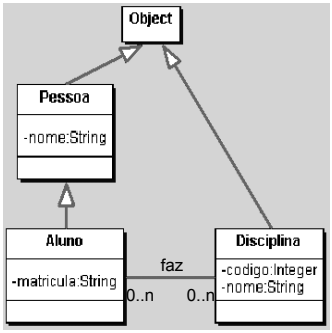
- Documentação de um método:



Pessoa.html

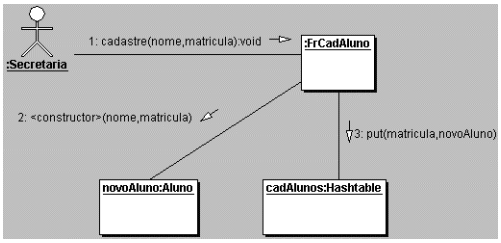
Estudo de Caso: Sistema de Matrículas

- Diagrama de Classes



Estudo de Caso : Sistema de Matrículas

- Diagrama de Colaboração
  - para o Cadastro de um Aluno.



Sistema de Matrículas

- Implementação da classe Pessoa

```
public class Pessoa {
   private String nome;

   public Pessoa(String nome) {
       this.nome = nome;
   }

   public String getNome() {
       return nome;
   }

   public void setNome(String nome) {
       this.nome = nome;
   }
}
```

resolve a ambiguidade!

or default, da classe n Java é ubclasse e Object.

Sistema de Matrículas

- Implementação da classe Aluno

```
public class Aluno extends Pessoa {
    private String matricula;

    public Aluno(String nome, String matricula) {
        super(nome);
        this.matricula = matricula;
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
}
```

Chama o construtor da **superclasse** Pessoa(nome)

Sistema de Matrículas

- Implementação da classe Disciplina

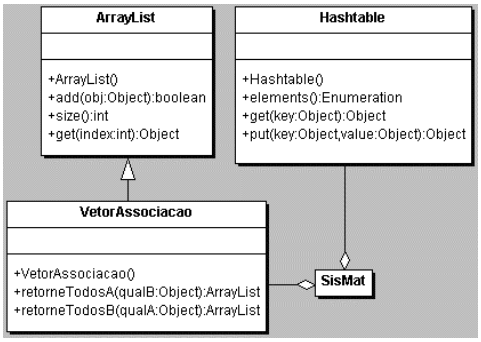
```
public class Disciplina {
    private Integer codigo;
    private String nome;

    public Disciplina(Integer codigo, String nome) {
        this.codigo = codigo;
        this.nome = nome;
    }

    public Integer getCodigo() {
        return codigo;
    }

    public void setCodigo(Integer codigo) {
        this.codigo = codigo;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome)    {
        this.nome = nome;
    }
}
```


Projeto do Sistema de Matrículas

- Estruturas de Dados:
  - 1 VetorAssociacao (ArrayList):
    - para armazenar as associações (matriculas de alunos em disciplinas).
  - 2 Hashtable:
    - para armazenar alunos, pois precisamos saber qual o aluno, a partir de sua matricula (chave).
    - para armazenar disciplinas, pois precisamos saber qual a disciplina, a partir de seu código (chave).
- API do Java oferece as classes ArrayList e Hashtable do pacote java.util

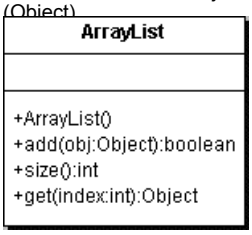
Projeto do Sistema de Matrículas

- Estruturas de Dados



A Classe ArrayList (java.util)

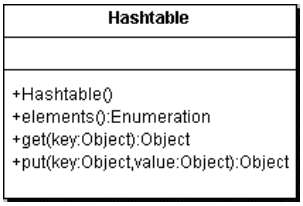
- Genérica
  - Uma lista armazena objetos de qualquer classe (Object)



- Mais eficiente do que a antiga Vector

A Classe Hashtable (java.util)

- Genérica para a chave e para o valor:
  - chave pode ser um objeto de qualquer classe



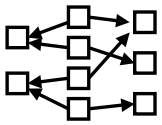
- Exemplo:
  - **cadAlunos.put(matricula, new Aluno(nome, matricula);**
  - **Aluno alguem = (Aluno)cadAlunos.get(matricula);**

Associação em UML → Associação em Java

3. Usando um objeto de associação:

- Matrícula = associação entre 1 aluno e 1 disciplina.

```
public class Aluno
{

  // sem referências.

  //...
}

public class Disciplina
{

  // sem referências.

  //...
}
```



Memória

Processamento

Associação em Java

- AssocAlunoDisciplina serve apenas para Aluno e Disciplina.

```
public class AssocAlunoDisciplina
{
  private Aluno oAluno;
  private Disciplina aDisciplina;

  //...
}
```

- Para a associação entre Professor e Disciplina, seria necessário outra classe AssocProfessorDisciplina?!
- Interessante definir uma classe genérica.

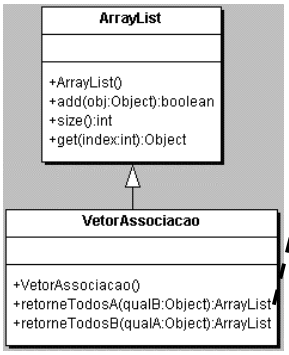
Uma Classe Genérica para Associação

- Associa dois objetos de quaisquer classes:

```
public class Associacao {
    private Object objetoA;
    private Object objetoB;

    public Associacao(Object a, Object b) {
        objetoA = a;
        objetoB = b;
    }

    public Object getObjetoA() {
        return objetoA;
    }

    public Object getObjetoB() {
        return objetoB;
    }
}
```

matrícula = associação entre 1 aluno (A) e 1 disciplina (B)

A Classe Genérica VetorAssociacao



Num relatório, é necessário saber quais são os alunos matriculados numa disciplina.

Implementação da Classe VetorAssociacao

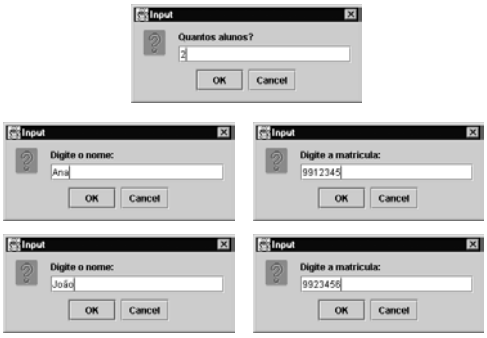
```
public class VetorAssociacao extends ArrayList {

  //...

  public ArrayList retorneTodosA(Object qualB) {

    ArrayList todosA;

    todosA = new ArrayList();

    int n = this.size();

    for(int i=0; i < n; i++) {
      if(qualB == ((Associacao)this.get(i)).getObjetoB())
        todosA.add(((Associacao)this.get(i)).getObjetoA())
      }
    }

    return todosA;
  }
}
```


☹ Sistema de Matrículas em 1 Camada

- Application em Java

```
import javax.swing.*;  // interface
import java.util.*;

public class SisMat {
    private Hashtable cadAlunos;
    private Hashtable cadDisciplinas;
    private VetorAssociacao cadMatriculas;

    public SisMat() {
        cadAlunos = new Hashtable();
        cadDisciplinas = new Hashtable();
        cadMatriculas = new VetorAssociacao();

        //...
    }

    public static void main(String args[]) {
        new SisMat();

        System.exit(0);
    }
}
```

Cadastrando Alunos

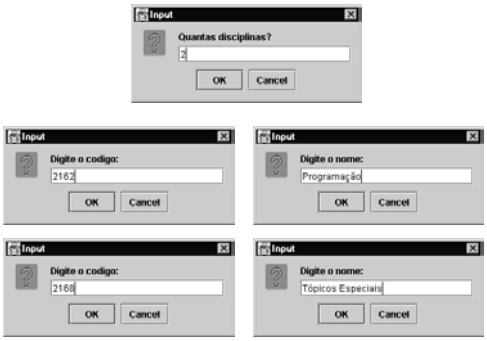


Cadastrando Alunos

```
public class SisMat {
  //...

  public SisMat() {
    //...

    String entrada;
    entrada = JOptionPane.showInputDialog("Quantos alunos?");

    int n = Integer.parseInt(entrada);

    for(int i=0; i < n; i++)
    {
      String nome;
      nome = JOptionPane.showInputDialog("Digite o nome: ");
      String matricula;
      matricula = JOptionPane.showInputDialog("Digite a matricula: ");
      cadAlunos.put(matricula, new Aluno(nome, matricula));
    }

    //...
  }
  //...
}
```

hashtable **chave** **objeto**

Cadastrando Disciplinas

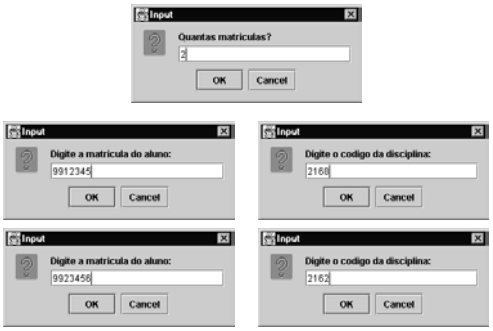


Cadastrando Disciplinas


```
public class SisMat {
  //...

  public SisMat() {
    //...

    entrada = JOptionPane.showInputDialog("Quantas disciplinas?");

    n = Integer.parseInt(entrada);

    for(int i=0; i < n; i++)
    {
        entrada = JOptionPane.showInputDialog("Digite o codigo: ");
        Integer codigo = new Integer(entrada);
        String nome;
        nome = JOptionPane.showInputDialog("Digite o nome: ");
        cadDisciplinas.put(codigo, new Disciplina(codigo, nome));
    }

    //...
  }
  //...
}
```

hashtable chave objeto

Fazendo Matrículas



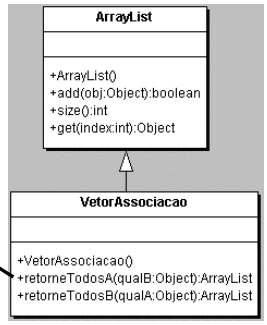
Fazendo Matrículas

```
//...

entrada = JOptionPane.showInputDialog("Quantas matriculas?");

n = Integer.parseInt(entrada);

for(int i=0; i < n; i++)
{
  String matricula;
  matricula = JOptionPane.showInputDialog("Matricula aluno: ");

  Aluno qualAluno = (Aluno)cadAlunos.get(matricula);

  String codigo;
  codigo = JOptionPane.showInputDialog("Codigo disciplina: ");

  Disciplina qualDiscip;

  qualDiscip = (Disciplina)cadDisciplinas.get(new Integer(codigo));

  cadMatriculas.add(new Associacao(qualAluno, qualDiscip));
}
```

faz a matrícula

Relatório de Matrículas

- Para cada disciplina, listar todos os alunos nela matriculados:
- Disciplina
  1 – Primeiro Aluno
  2 – Segundo Aluno
  ...
  N – Último Aluno



Retorna uma lista de todos os alunos de uma disciplina

Relatório de Matrículas

- cadDisciplinas é uma Hashtable. Como listar todas as disciplinas? R: método **`elements()`**

```
//...

System.out.println("--------RELATÓRIO----------");
Enumeration elementos = cadDisciplinas.elements();
while(elementos.hasMoreElements()) {
  Disciplina discip = (Disciplina)elementos.nextElement();
  System.out.println(discip.retorneNome());

  ArrayList alunos = cadMatriculas.retorneTodosA(discip);
  int total = alunos.size();
  for(int j=0; j < total; j++) {
    System.out.print(""+(j+1)+" - ");
    System.out.println(((Aluno)alunos.get(j)).retorneNome());
  }
}

//...
```

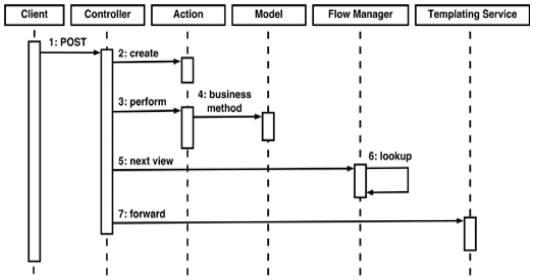
typecast de Object para Aluno

Resultado

- ---------- RELATÓRIO ----------
- Programação
  1 – João
  ...
- Tópicos Especiais
  1 – Ana
  ...
- ...

☺ Aplicando MVC

- Diagrama de sequência (um evento na WEB):

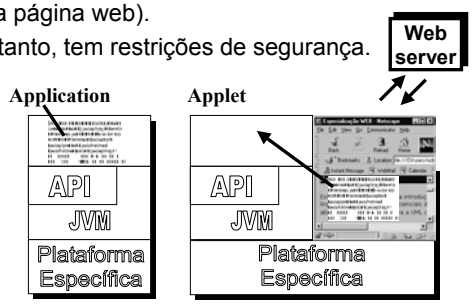


Java Standard direcionado para a WEB

- Implementar MVC sempre pensando na natureza *stateless* da WEB.
- Desenvolver uma classe *AplicacaoServidor* equivalente a um servlet. (Control)
- Desenvolver os frames de interface gráfica equivalentes à JSP. (View)
- Implementar Java Beans para representar o modelo. (Model)
- Uma implementação neste framework pode ser facilmente traduzida para o contexto da WEB!

Applets

- Utilizam a máquina virtual de um browser.
- São executadas automaticamente (ao visitar uma página web).
- Portanto, tem restrições de segurança.



Estrutura de uma Applet

```
public class Exemplo extends Applet {
  public void init() { /* primeiro executado */ }

  public void start() { ... }
  public void stop() { ... }

  public void destroy() { /* último executado */ }

  public void paint(Graphics g) { ... }
}
```


A Tag Applet em HTML

- **arquivo.html**

```
<html>
<head>
  <title>
    Exemplo de HTML com Applet
  </title>
</head>
<body>
  <applet code="Exemplo.class"
          width="200" height="200">
  </applet>
</body>
</html>
```


Aparência de uma Applet

- arquivo.html aberto num browser:

Height = 200

Width =200

Desvantagens do uso de Applets

- Restrições de segurança:
  - Comunicação apenas com o servidor de onde se originou a applet.
  - Acesso ao sistema de arquivos somente remoto.
- Uso limitado da API:
  - Classes devem estar na API utilizada pelo browser.
  - Ex: javax.swing não pode ser utilizada em geral.
- Applets ainda são recomendadas para aplicações pequenas e interativas, onde se conhece *a priori* o browser utilizado.
- Exemplo: na intranet de uma empresa.
- Para a web, em geral, utiliza-se JSP.

Restrições de Segurança em Applets

- Applications têm acesso a banco de dados, mas Applets têm restrições de segurança.
- Uma Applet pode apenas ter acesso a um banco de dados que estiver na máquina remota de onde se originou a Applet.
- Neste caso, tanto o servidor de web quanto o banco de dados devem estar na mesma máquina (ou um proxy deve ser usado).