OFICINA DO FUTURO

Conceitos e Fundamentos de Linguagem Java

Fundamentos

Aviso de Propriedade Intelectual

Todo o conteúdo desta obra intelectual é de propriedade intelectual única e exclusiva do Instituto Eldorado, IBM e Flextronics. Esta obra intelectual destina-se única e exclusivamente a ser utilizada por instrutores e alunos de projetos científicos, culturais e educacionais que o Instituto Eldorado, IBM e Flextronics promovam, em conjunto ou separados, e em consonância com o objetivo desses projetos. Qualquer outra forma de uso é estritamente proibida.
Dependem de prévia e expressa autorização dos proprietários, por escrito, usos que impliquem em:

a) Reprodução parcial ou integral desta obra intelectual;
b) Edição desta obra intelectual;
c) Adaptação e quaisquer outras transformações desta obra intelectual;
d) Tradução desta obra intelectual para qualquer idioma;
e) Inclusão desta obra intelectual em fonograma ou produção audiovisual;
f) Distribuição desta obra intelectual, independentemente do meio utilizado;
g) A inclusão desta obra em base de dados, sistema de armazenamento em computador, microfilme e demais formas de arquivamento do gênero.

Licença de uso especial para instrutores: a fim de disseminar o conhecimento contido nesta obra intelectual a terceiros, os instrutores que fizerem parte dos Projetos realizados pelo Instituto Eldorado, IBM e Flextronics terão direito a uma licença de uso especial, nos seguintes termos:

a) A licença é concedida por prazo indeterminado e em abrangência mundial.
b) A licença permite aos instrutores utilizar o conteúdo desta obra intelectual, no todo ou em parte, nas suas apresentações e aulas expositivas, síncronas ou assíncronas, tenham elas ou não finalidade lucrativa.
c) A licença não permite a exclusão de quaisquer avisos de propriedade intelectual, direitos autorais, *copyright* e afins.
d) A licença não permite aos instrutores modificar o conteúdo desta obra intelectual, no todo ou em parte, qualquer que seja o motivo.
e) A licença não permite:
    i. A edição desta obra intelectual;
    ii. A adaptação ou transformação desta obra intelectual;
    iii. A tradução desta obra intelectual para qualquer idioma;
    iv. A inclusão desta obra intelectual em fonograma ou produção audiovisual;
    v. A distribuição desta obra intelectual, independentemente do meio utilizado;
    vi. A inclusão desta obra em base de dados, sistema de armazenamento em computador, microfilme e demais formas de arquivamento do gênero, para propósitos diversos daqueles autorizados pela licença.
f) A licença não pode ser transferida a terceiros.
g) Sempre que houver o uso desta obra intelectual, deverá ser feita uma menção ao Projeto pelo qual o instrutor obteve o seu conteúdo
h) Sempre que houver o uso desta obra intelectual, deverá ser feita uma menção aos direitos de propriedade intelectual do Instituto Eldorado, IBM e Flextronics.

Ambiente de desenvolvimento Java

Para desenvolver aplicações Java, nós precisamos prepara um ambiente para:

- Editar o código fonte
- Compilar e empacotar classes
- Executar e debugar componentes
- Trabalhar em grupo controlando as versões

Ambiente de desenvolvimento Java

Java Platform (JDK) 8u11

Java IDEs

Exemplos das principais IDEs:

- **Eclipse**
- **NetBeans**

Como obter o Eclipse?

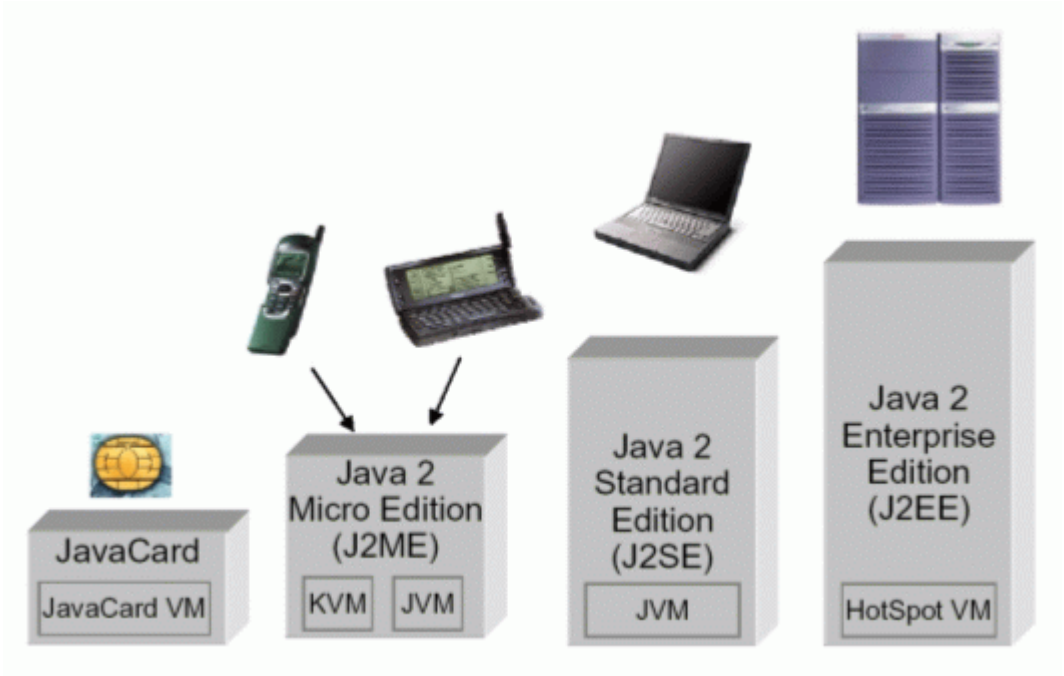
Eclipse é um ambiente de desenvolvimento open-source e pode ser baixado gratuitamente no seguinte endereço:

http://www.eclipse.org

A Linguagem Java - Introdução

Nessa unidade iremos mostrar as características da linguagem Java, a sintaxe e os passos para criar uma aplicação simples em Java utilizando o Eclipse.

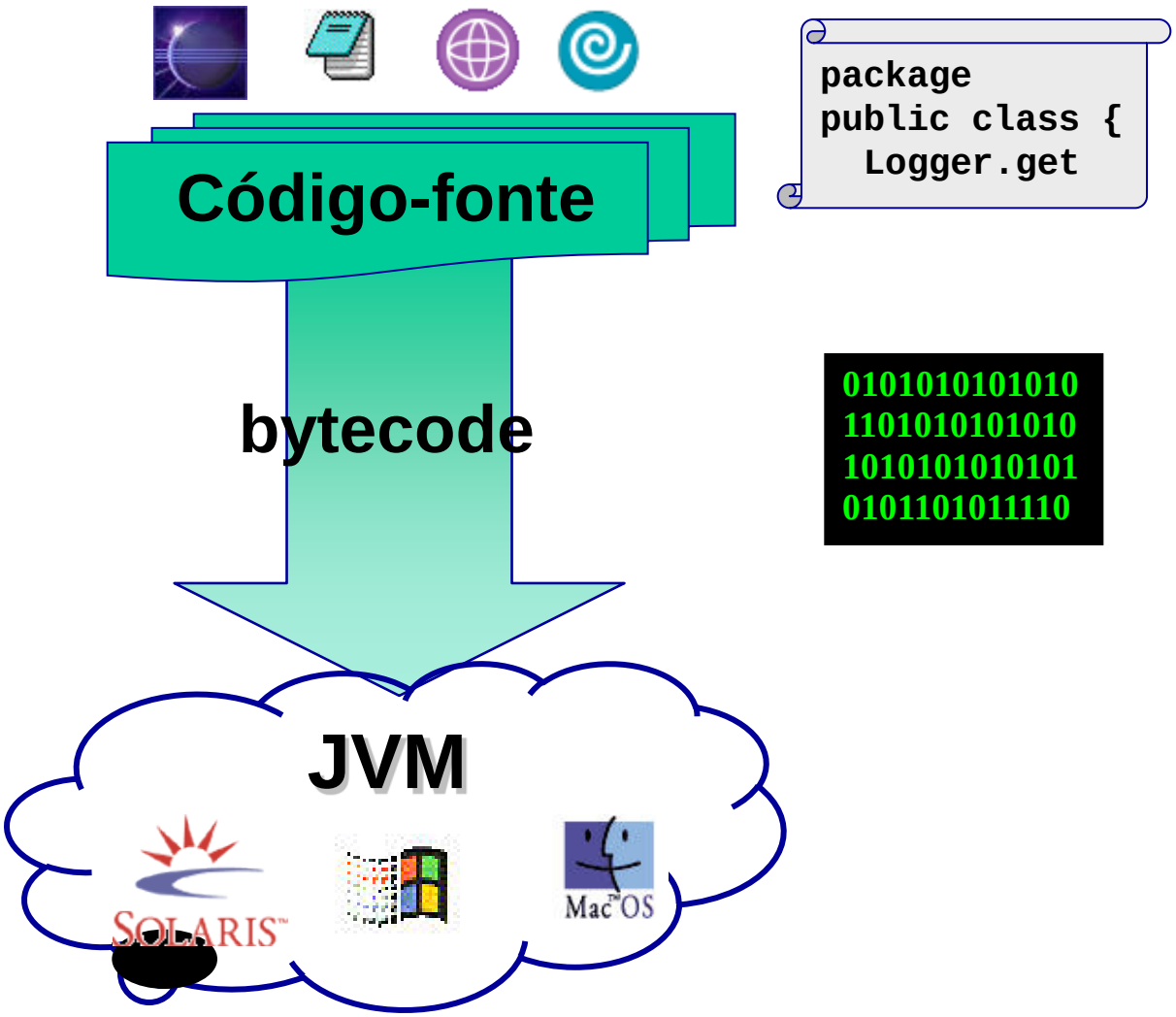
Java em Toda Parte



Características do Java

- Simples
- Orientada a Objetos
- Distribuída
- Suporte a Concorrência
- Dinâmica
- Independente de Plataforma
- Portável
- Alta Performance
- Robusta
- Segura

Entendo o Funcionamento Java



A Linguagem Java

- ByteCode
  - Código intermediário (*.class), entre o código fonte (*.java) e o código de máquina.
- Compilador
  - Responsável pela geração do bytecode (*.class) através da tradução do código fonte.
- JVM
  - Máquina virtual do Java, responsável pela leitura do bytecode, tradução para a linguagem de máquina e execução do programa.
- Classpath
  - Conjunto de caminhos especificado para a JVM encontrar as classes necessárias para a execução do programa.

Os arquivos fontes

- Arquivos fontes devem obrigatoriamente seguir a seguinte estrutura e ordem:
  - Definição do Pacote
  - Lista de import's
  - Declaração de Classe (s)
  - Corpo da Classe
- É permitido que haja uma única classe pública por arquivo fonte
- A classe pública deve conter o mesmo nome que o arquivo fonte.
  - Se o nome da classe é Aluno então o nome do arquivo fonte deve ser Aluno.java

Pacotes

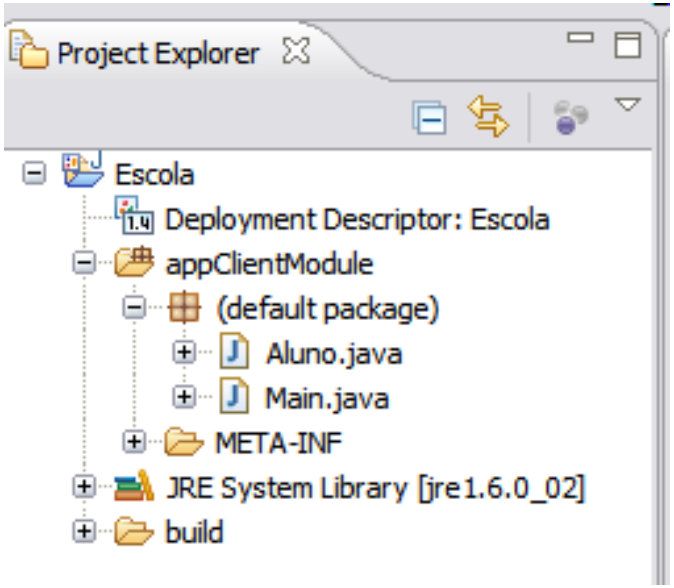
A primeira linha indica o nome do pacote que a sua classe pertence. Se você não especificar, será assumido que sua classe pertence ao pacote default.

O nome do pacote pode ser **com.oficina.app** e será representado por uma estrutura de diretórios como: **com/oficina/app**.

Pacotes

- Forma de organizar grupos de classes em unidades.
- Pode conter qualquer número de classes que se relacionam, seja pelo mesmo objetivo ou por escopo.
- Reduz problemas de conflitos de nome.
  - O nome de uma classe não é apenas aquele usado em sua declaração, mas sim o conjunto: nome do pacote + nome usado na definição da classe.
- Permite a proteção de classes, variáveis e métodos através dos modificadores.

Pacotes



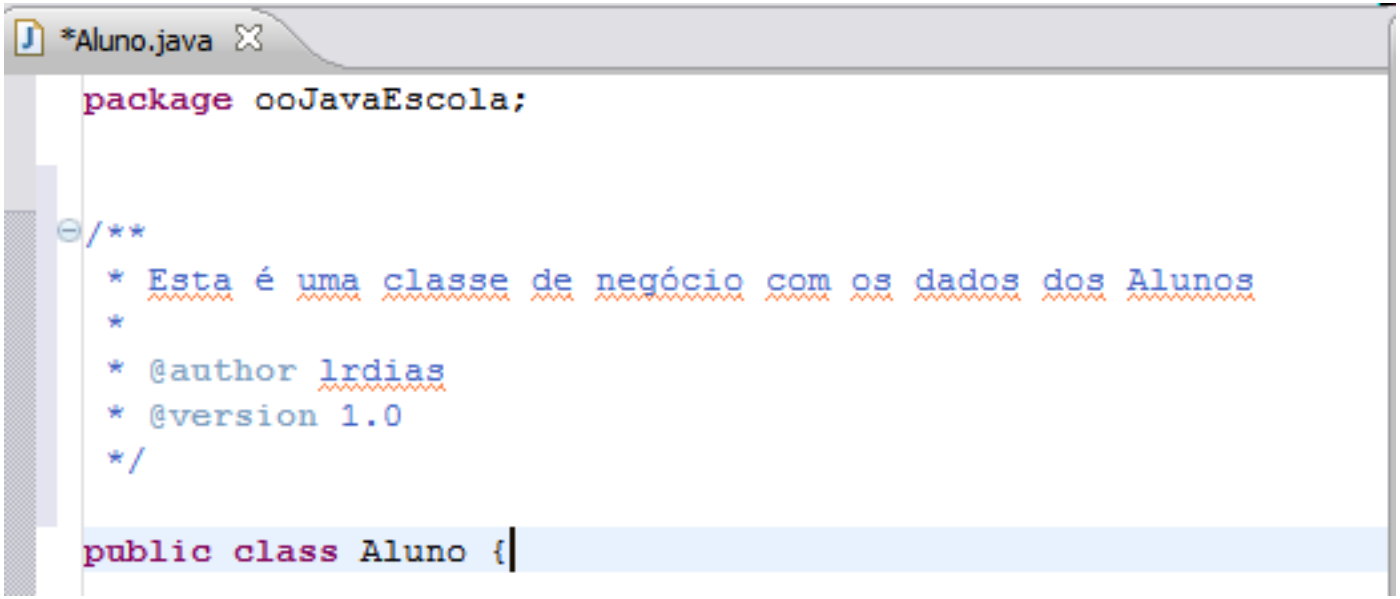
O Eclipse mostra a estrutura dos pacotes dentro do módulo da aplicação.

É uma boa prática **não** deixar as classes no pacote default.

Pacotes são usados para agrupar classes relacionadas.

Adicionando comentários

É uma boa prática adicionar comentários significativos para as classes e métodos.



Sintaxe

- Statement:
  - Uma ou mais linhas terminadas por ‘;’
- Blocos:
  - Conjuntos de statements delimitados por ‘{’ e ‘}’
- Comentários:
  - // → comentário simples, de uma linha
  - /* */ → comentário simples, de uma ou mais linhas
  - /** */ → comentários para documentação (Javadoc)

- Utiliza-se o import para declaração de classes que são referenciadas no arquivo fonte mas que não pertencem ao pacote onde este arquivo se encontra.
- Import's podem referenciar:
  - Outras classes no mesmo projeto
  - Classes da API Java, como java.util.List
  - Classes contidas nas bibliotecas utilizadas pelo projeto, ou seja, nos arquivos *.jar referenciados no classpath do projeto
- Sintaxe:
  - **import java.util.Date;**
  - **import java.util.*;**

Declaração da classe

Todo arquivo Java precisa ter uma classe pública e seu nome precisa ser igual ao nome dessa classe.

```
public class Aluno {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub

	}

}
```

Declarando uma classe

- Definição de uma classe:

```
[modificadores] class NomeDaClasse
{
   ....
}
```

- Exemplos:

```
public class Curso
{
}

public class Turma
{
}

public class Aluno
{
}
```

O corpo da classe

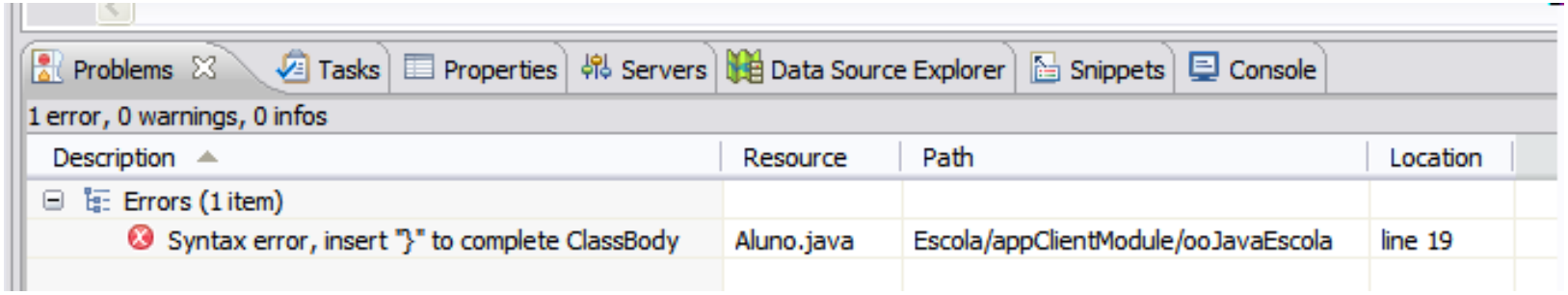
O corpo da classe é definido entre chaves { e }. Se não fecharmos uma chave aberta, um erro de compilação será mostrado.

```
public class Aluno {

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
```

Syntax error, insert "}" to complete ClassBody

O corpo da classe

Podemos ver a descrição dos erros na aba Problems no Eclipse.



O método **main**

Dentro do corpo da classe, nós podemos ter a declaração dos membros da classe, métodos e outras classes. Para o exemplo do Aluno, nós teremos somente o método main.

Repare que temos os modificadores **public**, **static** e **void** junto com os parâmetros em forma de array chamado args.

O método **main**

O modificador **public** significa que qualquer outra classe Java pode chamar esse método.

O modificador **static** significa que este método não faz parte do estado da classe.

O modificador **void** indica que o método não irá retornar nada e o resultado é somente a execução do mesmo.

O método **main**

O String[] args entre parênteses indica os argumentos de entrada para esse método.

Toda classe que tem um método parecido com:

public static void main(String[] args)

é chamada classe executável, porque a JVM sabe o que fazer quando os usuários querem executar essa classe diretamente.

Arquivos fontes

```
// Declaração de Pacote
package de pessoal.meuPacote;

// Declaração import's
import java.util.Random; //Importação de uma classe
import java.sql.*;       //Importação de um pacote inteiro

// Definições de Classes
public class MinhaClasse
{
    ... // Corpo da Classe
}
```

Declarando atributos de uma classe

- Definição de um Atributo:

  [modificadores] tipo nomeDoAtributo [= inicialização];

- Exemplos:

  private static int numero;

  public final String tamanhoMaximo = 15;

  private String nome = “Maria da Silva”;

  double raio = 6.5;

  Object o = new Object();

Declarando métodos

- Definição de um método:

```
[modificadores] retorno nomeDoMetodo ( [Argumentos] ) [ throws Exeções ]
{
    ...
    [ return varRetorno; ]
}
```

- Exemplos:

```
private void obtemNumeroAlunosTurma ( long codigoTurma ) { ... }

public String getNomeAluno( int codigoAluno ) { ... }

public void insereAluno ( String nomeAluno ) throws Exception { ... }

public static long getNumeroInstancias () { ... }
```

Exemplo de classe

```
public class Aluno
{
   String nomeAluno;
   int codigoAluno;

   public String getNomeAluno()
     {
     return nomeAluno;
   }

   public void setNomeAluno( String param )
   {
     nomeAluno = param;
   }

   public int getCodigoAluno()
     {
     return codigoAluno;
   }

   public void setCodigoAluno( int param )
   {
     codigoAluno = param;
   }
}
```

Nessa unidade iremos trabalhar com:

- Tipos de dados
- Operadores
- Estruturas de controle
- Arrays

Tipo de dados

A linguagem Java oferece diversos tipos de dados que podem ser divididos em duas categorias:

- Tipos Primitivos – dados mais simples ou escalares
- Tipos de Referências – Estruturas mais complexas como arrays, classes e interfaces.

Tipos primitivos

Tipo	Conteúdo	Default	Tamanho (bits)	Mínimo	Máximo
boolean	Valor lógico	False	8	--	--
char	Caracter Unicode	\u0000	16	\u0000	\uFFFF
byte	Inteiro com sinal	0	8	-2^7	$2^7 - 1$
short	Inteiro com sinal	0	16	-2^{15}	$2^{15} - 1$
int	Inteiro com sinal	0	32	-2^{31}	$2^{31} - 1$
long	Inteiro com sinal	0	64	-2^{63}	$2^{63} - 1$
float	Ponto flutuante	0.0	32	IEEE 754*	IEEE 754*
double	Ponto flutuante	0.0	64	IEEE 754*	IEEE 754*

Podemos declarar as variáveis dentro da classe (variável Global) ou dentro de um método.

Seguindo a sintaxe: tipo + nomeVariável;

Exemplos:

```
int idade;
boolean menorIdade = false;
char sexo;
float nota1;
double salario  = 1.500;
```

Programando:

Crie as seguintes variáveis na classe Aluno:

- nota1 e nota2 do tipo double;
- idade do tipo int;
- sexo do tipo char;

Atribua valores para essas variáveis e imprima usando o comando:

System.out.println(variavel);

```
public static void main(String[] args) {
	// TODO Auto-generated method stub

	double nota1, nota2;
	int idade;
	char sexo;

	nota1 = 5;
	nota2 = 7.0;
	idade = 10;
	sexo  = 'f';

	System.out.println(nota1);
	System.out.println(nota2);
	System.out.println(idade);
	System.out.println(sexo);

}
```

Identificadores

- São os nomes dados a uma classe, método, atributo, variável ou parâmetro.
- Começam sempre por um caractere Unicode, (_) ou ($).
- Diferenciam maiúsculas e minúsculas
- Não podem coincidir com uma palavra reservada.
- Identificadores Válidos - exemplos:
  - x
  - y
  - América
  - _9_i$to_EH_meio_esquisito
  - total_1+2+3
  - $4outroExemplo
  - exemploCOMmuitasPALAVRAS

Identificadores – Convenções da Linguagem

- Na linguagem Java é utilizada a seguinte convenção para formação de identificadores:
  - Constantes com todas as letras em maiúsculo: **CONSTANTE**;
    - **public static final int QUANTIDADE_MAXIMA = 100;**
  - Variáveis começam com letra minúscula: **variável**;
    - **String nomeUsuario;**
  - Classes começam com letra maiúscula: **Classe**;
    - **public class Usuario { ...}**
  - Métodos começam com letra minúscula: **metodo(), metodo2(int a)**;
    - **public void recuperaUsuario(int codigoUsuario) {...}**
  - Se nome for composto, cada nome começa com letra maiúscula: **variavelComNomeComposto.**
    - **String nomeUsuario;**

Palavras reservadas

abstract	boolean	break	byte	case
catch	char	class	const	continue
default	do	double	else	extends
false	final	finallly	float	for
goto	if	implements	import	instanceof
int	interface	long	native	new
null	package	private	protected	public
return	short	static	super	synchronized
this	throw	throws	transient	true
try	void	volatile	while	

Operadores

Operador	Operação
++ , --	In/decremento unário
+ , -	Mais/menos unário (sinal)
~	Complemento de 1
!	Complemento lógico (not)
(tipo)	“cast”
*, /, %	Multiplicação, divisão, modulo
+ , -	Adição, subtração
+	Concatenação de strings
<<	Shift left
>>	Shift right
>>>	Shift right sem sinal
<, <=	Menor que, menor ou igual a
> , >=	Maior que, maior ou igual a
Instanceof	Comparação de tipos

Operadores - continuação

Operador	Operação
==, !=	Igual/diferente (valores)
==, !=	Igual/diferente (referência ao objeto)
&	E (bits)
&	E (lógico)
^	XOR (bits)
^	XOR (lógico)
|	OU (bits)
|	OU (lógico)
&&	E (lógico)
||	OU (lógico)
?:	Operador condicional (ternário)
=	Atribuição
*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=	Atribuição com operação

Exercício:

Crie uma variável chamada media do tipo double.

Calcule a media na nota1 e nota2 e atribua para a variável media.

Imprima na tela o resultado.

Dica:

```
System.out.println("O resultado da media é: " + media);
```

Estruturas de controle - decisão

- A linguagem Java provê duas estruturas de decisão:
  - if() / else
  - switch
- if() / else

```
if ( expressao_boolean ) {
    ....
} [ else  {
    ...
} ]
```

- switch

```
switch (key) {
    case value:
        <bloco de comandos>
     break;
     default :
        <bloco de comandos>
     break;
}
```

Estruturas de controle - decisão

OFICINA DO FUTURO

Exemplos:

```
if (idade < 18){

     System.out.println("Menor de idade!");
}
else{
     System.out.println("Maior de idade!");
}
```

```
switch (sexo){
case 'f': System.out.println("Sexo feminino!");
break;
case 'm': System.out.println("Sexo masculino!");
break;
default : System.out.println("Não especificado!");
break;
}

System.out.println("O resultado da media é: " + media);
```

Estruturas de controle - laço

- Existem três estruturas de controle em Java
  - **while()**

```
while(expressao_booleana)
{
  <bloco de comandos>
}
```

  - **do/while()**

```
do
{
  <bloco de  comandos>
} while(expressao_booleana)
```

  - **for**

```
for (<statement_inicializacao> [, <statement_inicializacao n>];
        <condicao_parada>;
        <expressao_incremento> [, <expressao_incremento n>])
{
        <bloco de comandos>
}
```

Estruturas de controle - laço

OFICINA DO FUTURO

Exemplos:

```
int aux = 1;
while (aux < 10){
    System.out.println("While: " + aux);
    aux += 1;
}

aux = 1;
do{
    System.out.println("Do: " + aux);
    aux += 1;
}while (aux < 10);


for (aux = 1; aux<10; aux++){
    System.out.println("For: " + aux);
}
```

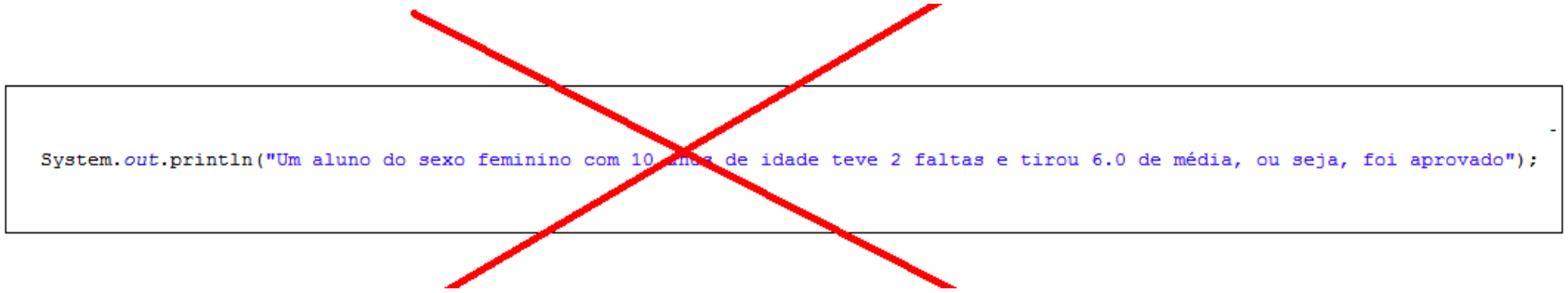
Exercícios:

- Crie uma variável do tipo int para armazenar as faltas.
- Atribua a quantidade de faltas = 12.
- Crie uma estrutura que imprima na tela se o aluno está aprovado ou reprovado com base na média do aluno (deverá ser acima de 5.0) e nas faltas (abaixo de 10).
- Atribua a quantidade de faltas = 4 e verifique novamente se o aluno está aprovado ou não.

Desafio:

- **Programe** para que saia a seguinte saída na tela:
- Um aluno do sexo feminino com 10 anos de idade teve 2 faltas e tirou 6.0 de média, ou seja, foi aprovado.

Obs: Utilize as variáveis existentes e crie novas se achar necessário.



A classe String

- String é uma classe da linguagem java e não um tipo primitivo.
  - Exemplo:

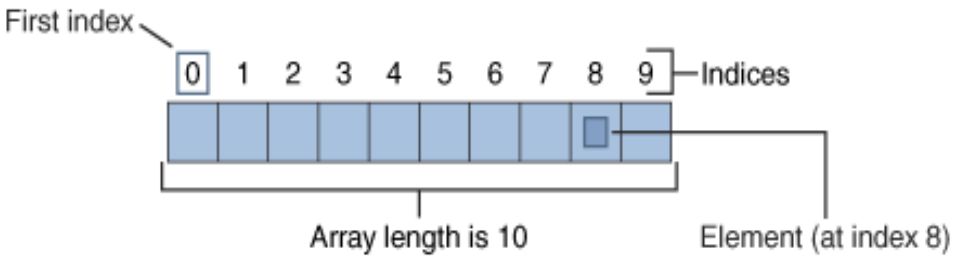
    String nome = “Carolina”;

    String sobreNome = “Silva”;
  - Podemos usar para saída de dados na tela:

```
String mensagem  = "Esta mensagem sera impressa";
System.out.println(mensagem);
```

Arrays

- Array é uma estrutura de tamanho fixo que armazena múltiplos valores do mesmo tipo.
- Qualquer tipo permitido em Java pode ser armazenado em um Array
  - Arrays de tipos primitivos
  - Arrays de referências de objetos
  - Arrays de outros arrays
- O tamanho de um Array precisa ser definido quando este é criado.
- Um elemento um array correspodente a um dos elementos armazenados no array e pode ser acessado por sua posição.



Utilizando Arrays

- Para utilizar um array é necessário seguir os três passos abaixo:
  - Declaração

    tipo [ ] nomeArray
  - Construção

    nomeArray = new tipo[numElementos]
  - Inicialização

    nomeArray[posicao] = valor

Utilizando Arrays

- Exemplos:
  - Declaração

    String[ ] nomes;
  - Construção

    nomes = new String[20];
  - Inicialização

    nomes[0] = “Maria”;

Exercícios:

- Faça um array de Strings com os nomes das disciplinas do aluno (Matematica, Portugues, Ciencias, Historia, Geografia e Artes). O array deverá ter 7 posições.
- Faça um segundo array de faltas, considerando que a posição 0 (zero) será Janeiro, atribua faltas para cada mês do ano.
- Na última posição do Array de disciplinas, concatene todas as displinas.
- Imprima os dois arrays da tela.

Referências

- Java How to Program – Deitel & Deitel
- Thinking in Java – Bruce Eckel
- Core Java Vol. 1 – Hortsman & Cornell
- Java API documentation

http://java.sun.com/j2se/1.5.0/docs/api/

OFICINA DO FUTURO

Conceitos e Fundamentos da Linguagem Java

Conceitos de Orientação a Objetos

História da POO

Nessa unidade, veremos um resumo da História da Programação Orientada a Objetos e seus benefícios.

História da POO

Primeira linguagem orientada a objetos: Simula 67.

Foi a primeira linguagem a incorporar facilidades para definir classes de objetos genéricos na forma de hierarquia de classes e subclasses.

Uma classe em Simula é um módulo que engloba a definicão da estrutura e do comportamento comuns a todas as suas instâncias (objetos)

História da POO

Primeira geração: linguagem de máquina

Segunda geração: linguagem de montagem (assembly)

Terceira geração: linguagem de alto nível

Quarta geração: linguagens para geração de aplicações

Geração Orientada a Objetos: linguagens voltadas para reuso e manutenção.

A quinta geração enfrenta o desafio de efetuar a manutenção e o reuso das milhares de aplicações desenvolvidas pelas gerações anteriores e atual.

Paradigma Orientado a Objetos X Estruturado

OFICINA DO FUTURO

Paradigma Estruturado:

- foco na ação;
- Executa funções ou procedimentos.

Paradigma Orientado a Objetos:

- foco no modelo;
- representa o modelo de entidades agrupando os dados e seus comportamentos correspondentes.

Benefícios:

- Possibilita ao usuário implementar um modelo realista das características e capacidades;
- Facilita a modularização e o reuso de código;
- Provê flexibilidade nas modificações de um sistema existente;
- Facilita a manutenção do código.

Conceitos

Nessa unidade, veremos quais são os conceitos gerais da Programação Orientada a Objetos.

Abstração

Considerar somente a informação essencial, desconsiderando detalhes desnecessários, definindo os limites conceituais de um objeto.

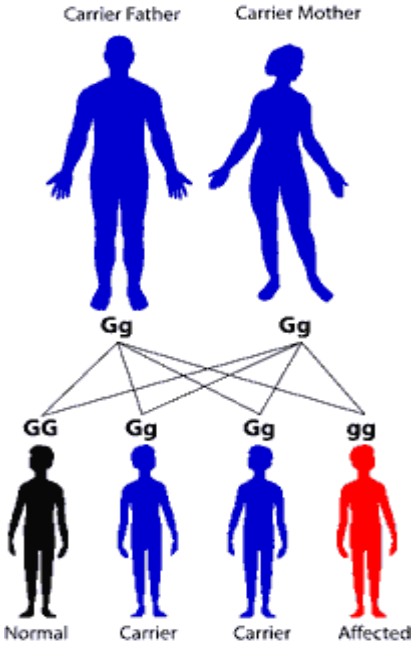
Ou seja, identificar os aspectos relevantes sem se perder nos detalhes.

Encapsulamento

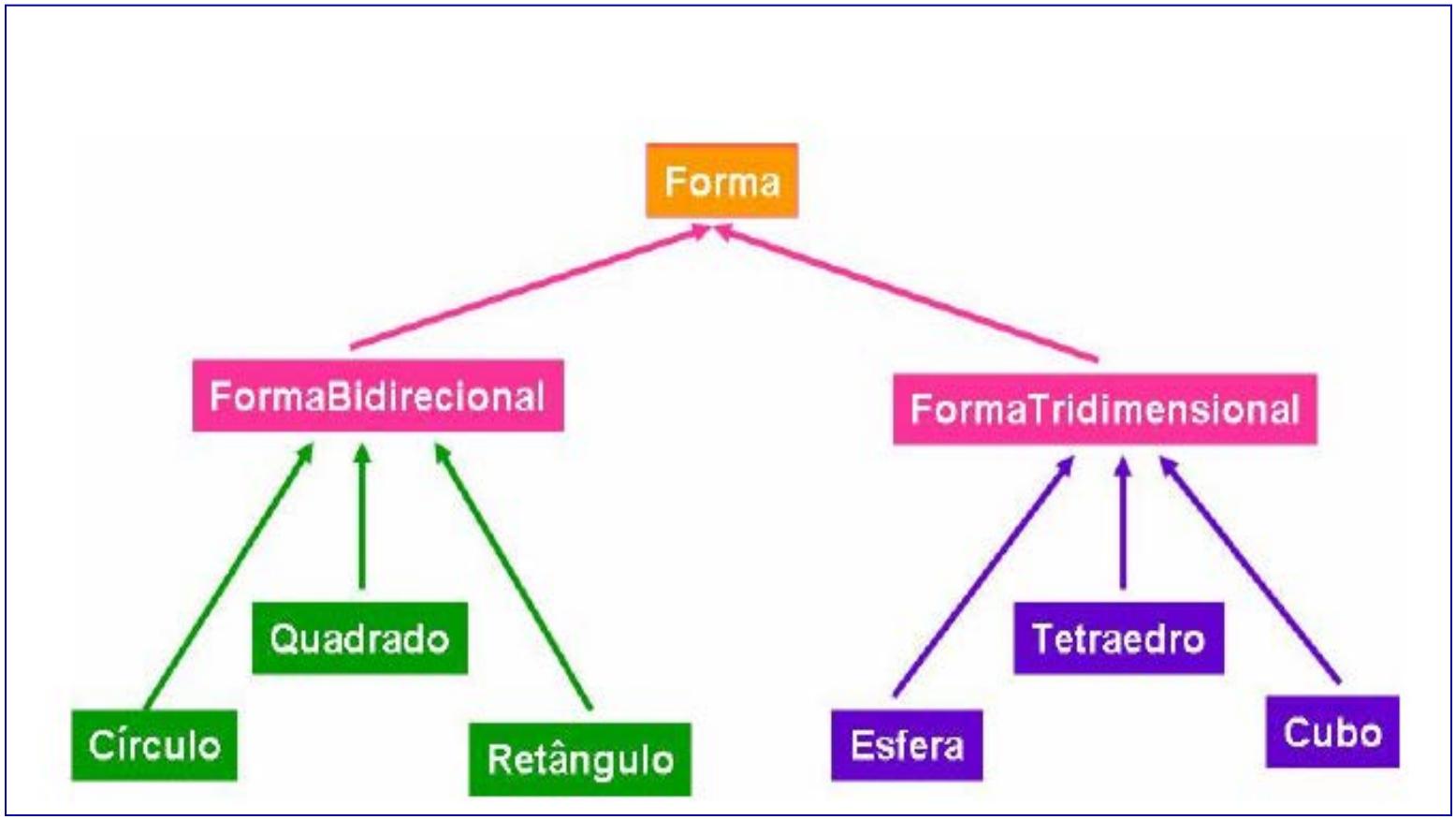
Mostrar somente o necessário, escondendo detalhes da implementação e informação desnecessária.

Herança

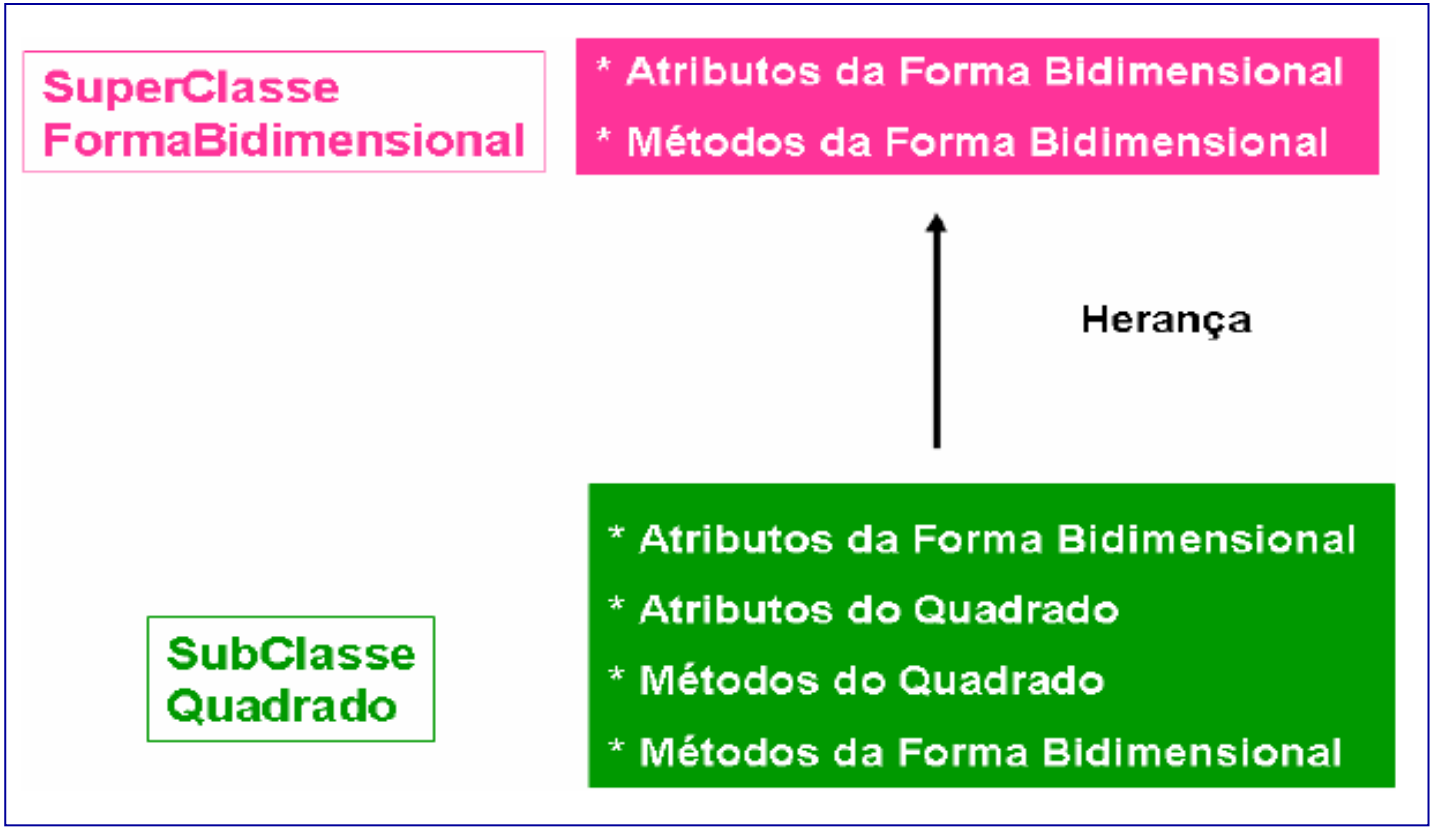
Habilidade de uma classe compartilhar e extender características existentes e comportamentos de uma outra classe, definindo então, uma hierarquia de classes.



Herança



Herança



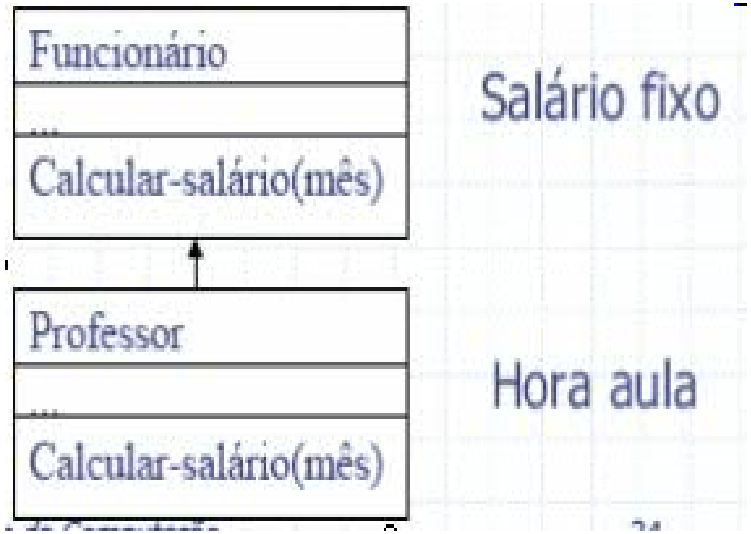
Polimorfismo

Habilidade de provêr diferentes comportamentos para o mesmo método, dependendo do tipo ou classes das quais os métodos foram invocados.

Polimorfismo

Em outras palavras, polimorfismo é a implementação diferenciada de operações nas classes filhas.

Uma operação deve manter a assinatura na hierarquia das classes.

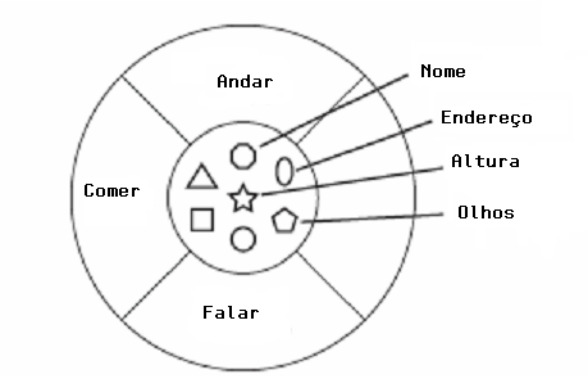


Conceitos chaves da POO:

- **Classe**
  - Um modelo que descreve um objeto
- **Objetos**
  - Representam entidades existentes no mundo real
- **Mensagens**
  - Mensagens são as ações dos objetos. Em outras palavras, são as chamadas para os métodos dos objetos.

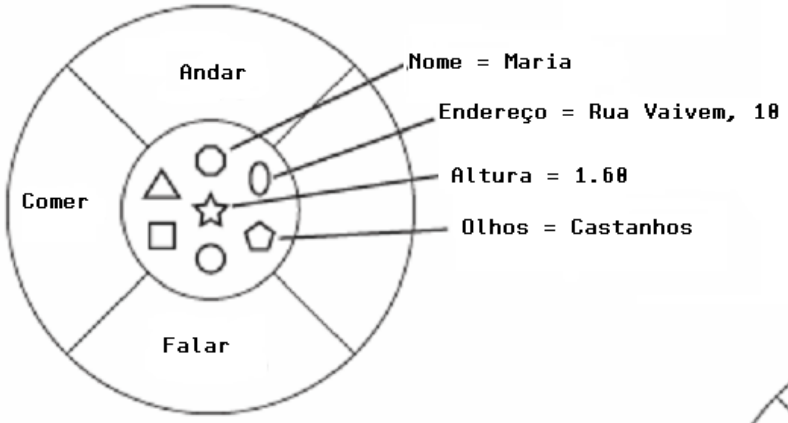
Exemplos:

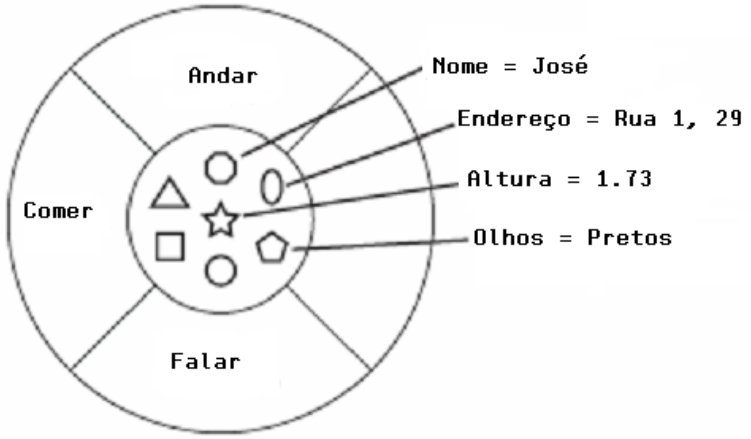
- A classe Pessoa
  - Quais itens descrevem características de uma pessoa?
  - Quais itens descrevem comportamentos de uma pessoa?



Exemplos:

- Objetos da Classe Pessoa





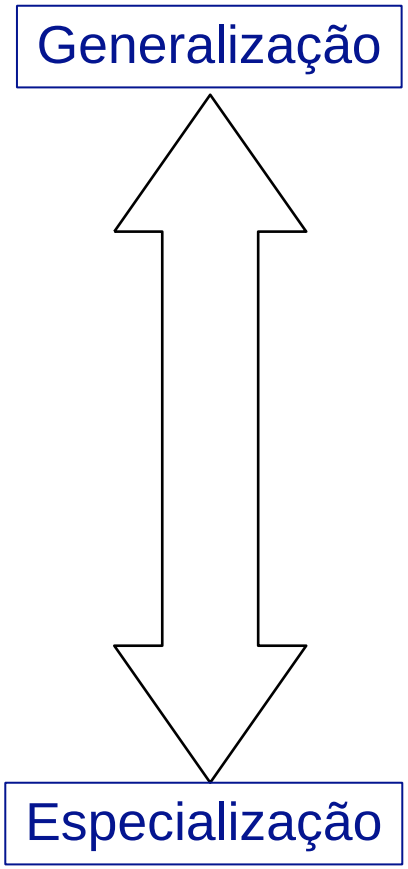
Relacionamentos

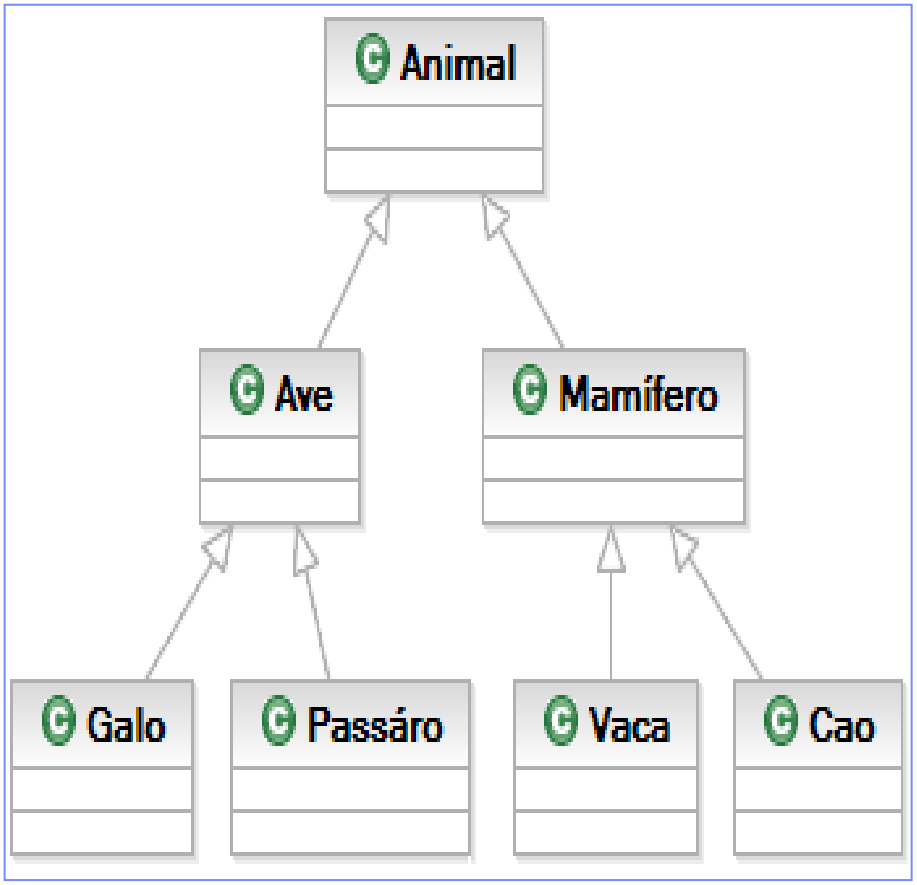
Nessa unidade, veremos quais são os tipos de relacionamentos da Orientação a Objetos.

Relacionamento entre Classes:

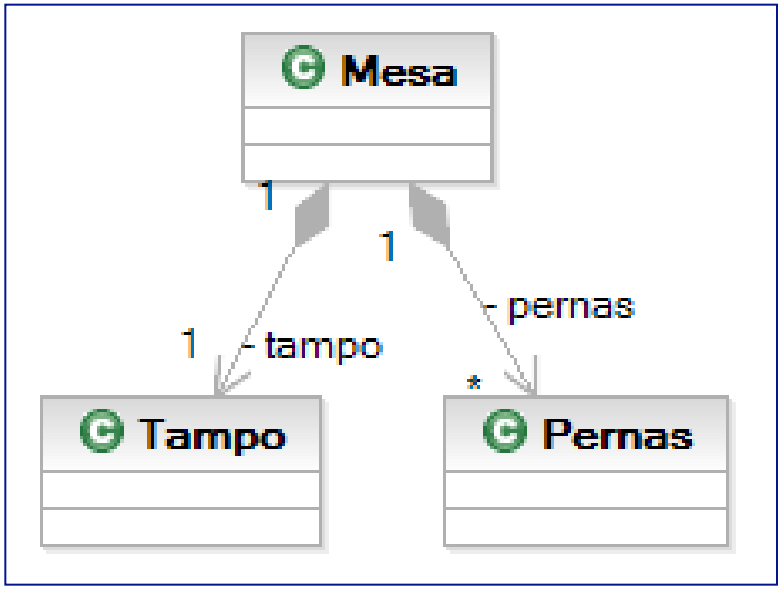
- Herança
  - Classes “filhas” herdam o comportamento e atributos da classe “pai”.
- Composição
  - Formação do todo pelas partes.
- Generalização
  - Comportamento e características generalizados.
- Especialização
  - Particularização do Comportamento das subclasses.

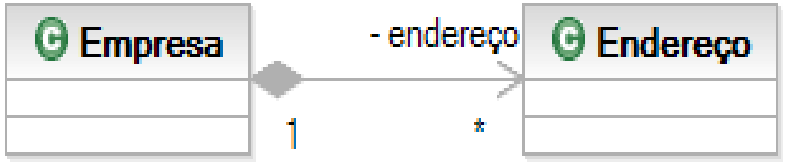
Herança, Generalização, Especialização





Composição





Exercício

Com base nos modelos apresentados anteriormente, desenhe um modelo, com as classes, atributos, métodos e relacionamentos para um sistema que calcula o salário de um funcionário e deposita na sua conta bancária. Sabendo que, existem dois tipos de funcionários: Professores (que ganham por hora) e diretores (sálario fixo).

Exercício

Dica:

Esse diagrama deve conter todos os tipos de relacionamentos aprendidos nessa lição.

Vantagens

Nessa unidade, veremos quais são as vantagens da Programação Orientada a Objetos.

Vantagens

- Reuso

  - Acontece devido aos possíveis relacionamentos entre as classes: Herança e Composição.

- Abstração

  - Atributos e comportamentos bem encapsulados, o que torna o código mais manutenível e robusto.

Reuso

Para entregar um software de qualidade é necessário reuso.

Reuso não é copy & paste!

Abstração

O conceito de abstração pode ser implementado com:

- Interfaces
- Classes Abstratas
- Encapsulamento

Interfaces

- Não são classes
- Não possuem métodos implementados
- Possuem apenas definição de comportamento:
  - Métodos abstratos
  - Constantes
- Não podem ser instanciadas

Para quê servem???

Classes Abstratas

- Classes abstratas são classes que podem possuir métodos implementados, mas que possuem ao menos um método abstrato.
  - Método abstrato é aquele em que não existe implementação, apenas sua definição.
- São úteis quando definem a implementação de métodos comuns a todas as classes que as estendem, mas obrigam que cada uma destas classes definam a implementação dos outros métodos abstratos.
- Classes abstratas não são instanciadas.

Classes Abstratas x Interfaces

- Interfaces possuem somente métodos abstratos
- Classes abstratas possuem métodos abstratos, mas também possuem tantos métodos implementados quantos for necessário.
- Interfaces são usadas para determinar como lidar e não como fazer uma tarefa ou grupo de tarefas.

Exercícios

Identifique nas classes a seguir, que atributos e/ou métodos não pertecem ao escopo do problema.

Cliente	Endereço	Alugar	Carro
nome idade sexo religião	rua cidade estado telefone cep	dataAluguel placaCarro renavam valorAluguel	placa renavam cor cliente
cadastrarCliente( )	alterarEndereço( ) cadastrarEndereço( )	alugarCarro( ) alterarPlaca( )	alterarNome( ) cadastrarEndereço( )

OFICINA DO FUTURO

Conceitos e Fundamentos da Linguagem Java

Java e Programação Orientada a Objetos

Classes e Métodos

Nessa unidade, entenderemos a estrutura de uma classe feita em Java, seus atributos e métodos.

Classes e Métodos

```java
//Declaração do Pacote (Não é mandatório)
package ooJavaEscola;

//Definição dos imports (Não é mandatório)
import java.math.*;

//Declaração da classe
//<nivel de acesso> + <nome da classe>
public class Teste {

    //declaração dos atributos

    //<nivel de acesso> + <tipo> + <nome>
    private boolean estaOK;

    //<nivel de acesso> + <modificador> + <tipo> + <nome>
    private static String texto;


    //declaração dos métodos

    //<nivel de acesso> + <tipo do retorno> + <nome>
    public boolean seraQueEstaOK ( ) {
        return estaOK;
    }

  //<nivel de acesso> + <modificador> + <tipo do retorno> + <nome> + <parametros>
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        System.out.println(texto);

    }

}
```

Declaração da Classe

Baseado na estrutura vista anteriormente, nós podemos identificar os elementos que formam a delcaração:

- Níveis de acesso (veremos na próxima unidade)
- Modificadores
- Tipos de Retorno
- Nomes

Modificadores

O modificador de um método especifica como as subclasses da classe mãe irá interagir com o método. Ou seja, o jeito que os atributos e métodos serão usados pelas outras classes.

Possíveis valores para os modificadores são: abstract, final, native, “nenhum valor”, static, synchronized, etc.

Para classes os únicos aplicados são abstract e final.

Tipos de Retorno

O tipo de retorno de um método determina que tipo de valor é esperado quando o método é chamado.

Possíveis tipos de retorno: todos os tipos básicos do Java, todas as classes e **void**. O retorno **void** representa a ausência de um tipo de retorno para um método.

Sempre que escolhemos um tipo de retorno, o método requere que a palavra **return** seja usada, exceto quando usamos o **void**.

Nomes

Os nomes de classes, atributos e métodos são a critério do programador. Entretanto, é importante atender as Convenções de Código Java (Java Code Conventions).

Maiores informações nesse assunto podem ser encontradas no site:

http://java.sun.com/docs/codeconv

Assinatura dos Métodos

A assinatura de um método o identifica na classe.

É composta pelo seu nome e parâmetros.

public static void falarAlgo **(**String algo **)**
public static void falarAlgo **(**StringBuffer algo **)**

Métodos de Acesso

Os métodos de acesso são utilizados para alterar e pegar informações dos atributos da classe.

Pela convenção Java são dois tipos: **getters** e **setters**.

O método **get** pega o valor do atributo.

O método **set** altera o valor do atributo.

Getters e Setters

Exemplos:

```
public String getAtributo(){
    return atributo;
}

public void setAtributo(String atributo){
    this.atributo = atributo;
}
```

Exercícios

Crie um novo pacote chamado Escola;

Crie uma classe pública chamada Aluno;

Crie os seguintes atributos:

- matricula (String)

- nome (String)

- curso (String)

- idade (int)

Crie os métodos de acesso para esses atributos;

Controlar no método que pega a idade se o usúario entrou com a idade menor que 1, nesse caso, atribuir o valor 0 (zero).

Construtor da Classe

- Para que um objeto exista é necessário contruí-lo, isto é, dizer para a JVM que é necessário espaço de memória para criação do objeto.
- Para contruir um objeto usa-se o construtor da classe.
  - **Aluno o1 = new Aluno();**
  - **Object o2 = new Object();**
- Toda classe possui, por default, um construtor padrão: público e sem argumentos.
- O construtor default somente é criado quando nenhum outro construtor for definido pelo programador.
- Uma classe pode ter quantos contrutores desejar.
- Implicitamente, ou mesmo explicitamente, o construtor sempre chama o contrutor da sua super classe.

Declarando Construtores da Classe

- Definição de um método:

```
[modificador] nomeDaClasse ( [Argumentos] ) [ throws Exeções ]
{
    ...
}
```

- Exemplos:

```
public Turma () { ... }

public Turma ( long codigoTurma ) { ... }

public Curso( int codigoCurso ) throws Exception { ... }

public Curso ( String nomeCurso, int codigoCurso ) { ... }
```

Exemplo de Classe

```
public class Aluno
{
   String nomeAluno;
   int codigoAluno;

    public Aluno()          //construtor vazio
  { }

    public Aluno(String nomeAluno, int codigoAluno)
    {
       this.nomeAluno = nomeAluno;
     this.codigoaluno = codigoAluno;
    }
    public String getNomeAluno()
     {
     return nomeAluno;
   }


   public int getCodigoAluno()
     {
     return codigoAluno;
   }
}
```

Exercício

Para a classe Aluno criada anteriormente, crie um construtor vazio e um cheio (com todos atributos);

Faça um método **main,** crie um objeto aluno usando o construtor vazio;

Imprima na tela os valores dos atributos utilizando o método **get**;

Crie um segundo objeto aluno usando o construtor cheio;

Imprima na tela os valores dos atributos utilizando o método **get**.

Exercício Continuação

Crie um novo objeto aluno com o construtor vazio;

Utilize o método **set** para alterar os valores do atributos;

Imprima na tela os valores dos atributos usando o método **get**.

Controle de Acessos

Nessa unidade, veremos quais são os tipos de controle de acessos que o Java disponibiliza.

Níveis de Acesso

O primeiro componente da declaração de uma **classe** ou **método** é o modificador do nível de acesso.

Os modificadores de nível de acesso determinam se outras classes podem usar os atributos ou invocar um método particular da classe.

Os possíveis valores para os especificadores de acesso são: private, protected, default (nenhum valor) e public.

Modificadores de nível de acesso

- **Public:** visível para qualquer classe sem restrições
  - Pode ser usado por classes, atributos e métodos.
- **Protected:** visível para todas as classes do mesmo pacote e para as subclasses
  - Pode ser usado apenas por atributos e métodos.
- **Nenhum modificador definido (default):** visível para todas as classes do mesmo pacote
  - Pode ser usado por classes, atributos e métodos.
- **Private:** visível somente para as subclasses
  - Pode ser usado por classes internas, atributos e métodos.

Exercício

Crie uma classe chamada TestaAcesso, com o método **main**;

Crie um objeto da classe Aluno utilizando o construtor cheio;

Mude o nível de acesso do método getNome para **private** e veja o que acontece.

Desafio

Crie uma classe Teste no pacote **ooJavaEscola**;

Crie um objeto da classe Aluno do pacote **Escola** usando o construtor cheio e chame os métodos para pegar os valores dos atributos.

Altere a classe Aluno do pacote **Escola** e veja o que acontece.

Variável This

A variável this – Diferenciando Atributos de Variáveis Locais

```
public class Pessoa
{
    private String nome;

    public void setNome( String nome )
    {
      // this.nome -> atributo da classe
      // nome -> variável local ao método
        this.nome = nome;
    }
 }
```

Classe Object e String

Nessa unidade, entenderemos como funcionam as classes String e Object e quais são seus principais métodos.

A Classe String

- String é uma classe da linguagem java e não um tipo primitivo.
  - Exemplo:

    String nome = “Carolina”;

    String sobreNome = “Silva”;
  - Alguns métodos utilitários da classe String:
    - **boolean equals(...)**
    - **equalsIgnoreCase(...)**
    - **int length(...)**
    - **substring(...)**
    - **toLowerCase(...)**
    - **toUpperCase(...)**
  - Referência para a classe String:
    - API Java: http://java.sun.com/j2se/1.4.2/docs/api/

A Classe Object

- Topo da Hierarquia de classes do Java
  - Toda classe em Java é “filha” da classe **Object**.
- Mesmo que um classe não use a palavra reservada extends, o compilador gera a classe extendendo diretamente de **Object**.

Garbage Collector

- Em Java, não é preciso fazer alocação dinâmica explícita para criar objetos, também não é preciso desalocar memória.
- O Garbage Collector é um processo interno da JVM que de tempos em tempos executa seu processo e faz desalocação de objetos que não possuem mais referências.
  - Não é possível saber quando os objetos serão enviados para o Garbage Collector.
  - Algumas classes precisam defnir como suas instâncias devem ser excluídas da memória.
    - Para isto, o GC utiliza o método finalize() da classe **Object.** É possível extender o método para fazer a desalocação personalizada para uma subclasse.

Herança

Nessa unidade, iremos saber como implementar Herança na linguagem Java.

Herança - Conceitos

- A implementação do conceito de herança em Java é feito com a palavra **extends**.
- Em Java é possível extender SOMENTE uma classe.
- Um classe que extende a outra é chamada de sub classe, e a classe extendida é chamada de super classe.
  - A sub classe é uma especialização da super classe.
  - A super classe é uma generalização da sub classe.

Herança - Sintaxe

```
public class Pessoa
{
   public void getNome()
   {
   }
   public void getCPF() {
   }
}
```

```
public class Funcionario
   extends Pessoa
{
   public void getSalario()
   {
   }
}
```

Sobreescrita de Métodos

- Usado quando é preciso modificar o comportamento de um método da classe pai.
- Para sobreescrever um método as condições abaixo devem ser satisfeitas:
  - O nome do método assim como o tipo e a ordem dos paramêtros devem ser idênticos aos do método da classe pai.
  - O tipo de retorno também deve ser idêntico.
  - A visibilidade não deve ser mais restritiva que a visibilidade do método original.
  - O método não deverá lançar “checked exceptions” que não são lançadas pelo método original.

Herança e Super

- Quando um método sobreescreve um método da super classe, o comportamento normal da subclasse é executar o novo método.

- Existe uma maneira de executar o método da super classe através da palavra reservada **super**, que é uma referência à super classe.

```
public void metodoSobreescrito  {

   super.metodoSobreescrito();

    // outras coisas específicas da classe filha

}
```

Sobrecarga de métodos

- Usada quando é preciso vários métodos que desempenham papéis semelhantes em diferentes condições.
- Para sobrecarregar um método as seguintes condições devem ser satisfeitas:
  - A identidade de um método é determinada pelo nome completo da classe a que pertence, pelo seu nome, pelo seu tipo, ordem e quantidade de parâmetros.
  - Dois ou mais métodos na mesma classe (incluindo métodos da super classe) com o mesmo nome mas com uma lista de parâmetros diferentes são métodos sobrecarregados.
  - O tipo de retorno do método, sua visibilidade, e lista de parâmetros pode variar livremente.
  - Métodos sobrecarregados podem chamar um ao outro, utilizando uma chamada comum de método com lista de parâmetros apropriada.

Sobrecarga de métodos

```
public class ExemploSobrecarga()
{
   public void metodoSobrecarregado(int param1){ }
   public void metodoSobrecarregado(String param1){ }
   public void metodoSobrecarregado(int param1,
                    String  param2){ }
   public void metodoSobrecarregado(String param1,
                    int param2){ }
   public void metodoSobrecarregado(double param1,
                    String param2, int param3){ }
}
```

Sobrecarga de métodos

```
class Base {
   public base( String s ) {
      // inicializa o objeto usando s
    }
    public base( int i ) {
      // inicializa o objeto usando i
    }
}

 class Derived extends Base {
    public Derived( String s ) {
       // passa o controle para o construtor de Base na linha 2
       super( s );
    }
    Public Derived( int i ) {
       // passa o controle para o construtor de Base na  linha 5
           super( i );
        }
}
```

Exercício

Codifique as seguintes classes: