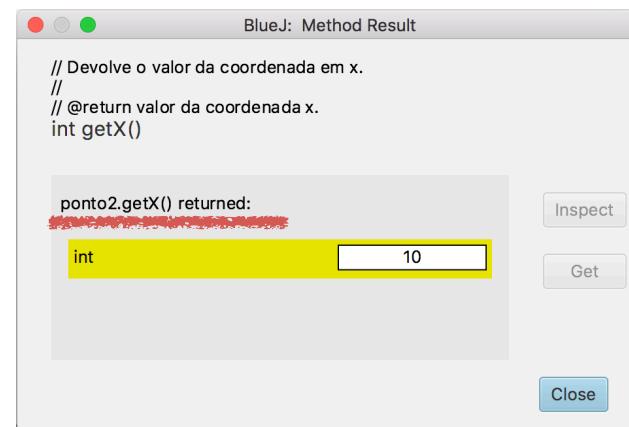
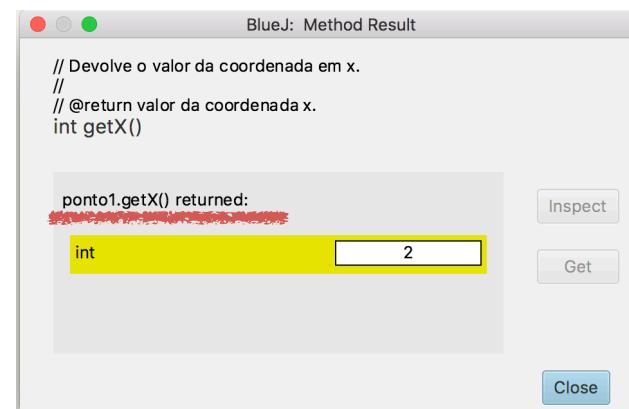


Modelo de execução dos métodos

- quando uma instância de uma classe recebe uma dada mensagem, solicita à sua classe a execução do método correspondente
- os valores a utilizar na execução são os do estado interno do objecto receptor da mensagem

- o envio da mensagem `getX()` a cada um dos pontos, origina a seguinte execução:



- importa referir que existe uma distinção entre mensagem e o resultado de tal envio
- o resultado é activação do método correspondente
- o método é executado no contexto do objecto receptor, utilizando os valores do estado interno do objecto.
- daí, o mesmo método ter resultados diferentes consoante o objecto receptor

Utilização de uma classe: regra geral

- uma classe teste, com um método main(), onde se criam instâncias e se enviam métodos
- um programa em POO é o resultado do envio de mensagens entre os objectos, de acordo com o alfabeto definido anteriormente

```
public class TestePontos {  
    public static void main(String[] args) {  
  
        Ponto p1, p2, p3, p4;  
        int c1, c2, c3, c4;  
  
        p1 = new Ponto(1,1);  
        p2 = new Ponto();  
        p3 = new Ponto(4,-5);  
        p4 = new Ponto(10,15);  
  
        //imprimir a representação interna dos pontos  
        System.out.println("P1 = " + p1.toString());  
        System.out.println("P2 = " + p2.toString());  
        System.out.println("P3 = " + p3.toString());  
        System.out.println("P4 = " + p4.toString());  
  
        //aceder às coordenadas dos pontos através dos métodos de acesso  
        c1 = p2.getX(); c2 = p2.getY();  
        System.out.println("c1 = " + c1 + " c2 = " + c2);  
  
        //alterar os pontos  
        p2.deslocamento(3,5);  
        p1.somaPonto(p3);  
        p3.movePonto(5,5);  
        p4.setY(-1);  
        System.out.println("P1 = " + p1.toString());  
        System.out.println("P2 = " + p2.toString());  
        System.out.println("P3 = " + p3.toString());  
        System.out.println("P4 = " + p4.toString());  
        System.out.println("p1 == p2? : " + p1.iguais(p4));|
```

Documentação (efeito lateral positivo)

- é muito difícil (impossível?) programar em POO sem termos uma documentação bem construída
- o esforço de construção da documentação é maioritariamente feito na escrita dos comentários
- o *look&feel* obtido é consistente com o da documentação oficial

Class Ponto

java.lang.Object
Ponto

```
public class Ponto  
extends java.lang.Object
```

Classe que implementa um Ponto num plano2D. As coordenadas do Ponto são inteiras.

Version:

20180212

Author:

MaterialPOO

Constructor Summary

Constructors

Constructor and Description

[Ponto\(\)](#)

Construtor por omissão de Ponto.

[Ponto\(int cx, int cy\)](#)

Construtor parametrizado de Ponto.

[Ponto\(Ponto umPonto\)](#)

Construtor de cópia de Ponto.

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
void		deslocamento(int deltaX, int deltaY) Método que desloca um ponto somando um delta às coordenadas em x e y.
double		distancia(Ponto umPonto) Método que determina a distância de um Ponto a outro.
boolean		ePositivo() Método que determina se o ponto está no quadrante positivo de x e y
int		getX() Devolve o valor da coordenada em x.
int		getY() Devolve o valor da coordenada em y.
boolean		iguais(Ponto umPonto) Método que determina se dois pontos são iguais.
void		movePonto(int cx, int cy) Método que move o Ponto para novas coordenadas.
void		setX(int novoX) Actualiza o valor da coordenada em x.
void		setY(int novoY) Actualiza o valor da coordenada em y.
void		somaPonto(Ponto umPonto) Método que soma as componentes do Ponto passado como parâmetro.
java.lang.String		toString() Método que devolve a representação em String do Ponto.

sobre classes e instâncias

- “*Classes and object are separate yet intimately related concepts. Specifically, every object is the instance of some class, and every class has zero or more instances. For practically all applications, classes are static; therefore, their existence, semantics, and relationships are fixed prior to the execution of a program. Similarly, the class of most objects is static, meaning that once an object is created, its class is fixed. In sharp contrast, however, objects are typically created and destroyed at a furious rate during the lifetime of an application.*”

Definição do objecto classe

- Estado
 - identificação das variáveis de instância
- Comportamento
 - construtores/destrutores
 - getters e setters
 - outros métodos de instância, decorrentes do que representam (usualmente a parte mais interessante da API...)

A referência *this*

- é usual precisarmos de referenciar o objecto que recebe a mensagem
- mas, no contexto da escrita do código da classe, ainda não sabemos como é que se vai chamar o objecto
- sempre que precisamos de ter acesso a uma variável do objecto podemos usar a referência *this*

- uma utilização normal é querermos desambiguar e identificar as variáveis de instância
- Por exemplo, em

```
/**  
 * Construtor parametrizado de Ponto.  
 * Aceita como parâmetros os valores para cada coordenada.  
 */  
public Ponto(int x, int y) {  
    this.x = x;  
    this.y = y;  
}
```

- a utilização de *this* permite desambiguar a qual das variáveis nos estamos a referir

- a referência *this* pode ser utilizada para identificar um outro método da instância

```
public void movePonto(int cx, int cy) {  
    this.setX(cx);  
    this.setY(cy);  
}
```

- no caso de termos escrito apenas `setX(cx)` o compilador teria acrescentado automaticamente a referência *this*
- também utilizada para invocar os construtores (dentro de outros construtores)

Regras de acesso a variáveis e métodos

- a declaração deve ser complementada com informação sobre o nível de visibilidade das variáveis e métodos.

Tipo de Modificador	Visibilidade no código
public	a partir de qualquer classe
private	apenas acessível dentro da classe
protected	acessível a partir da classe, de classes do mesmo package e de todas as subclasses
default	acessível a partir da classe e classes do mesmo package

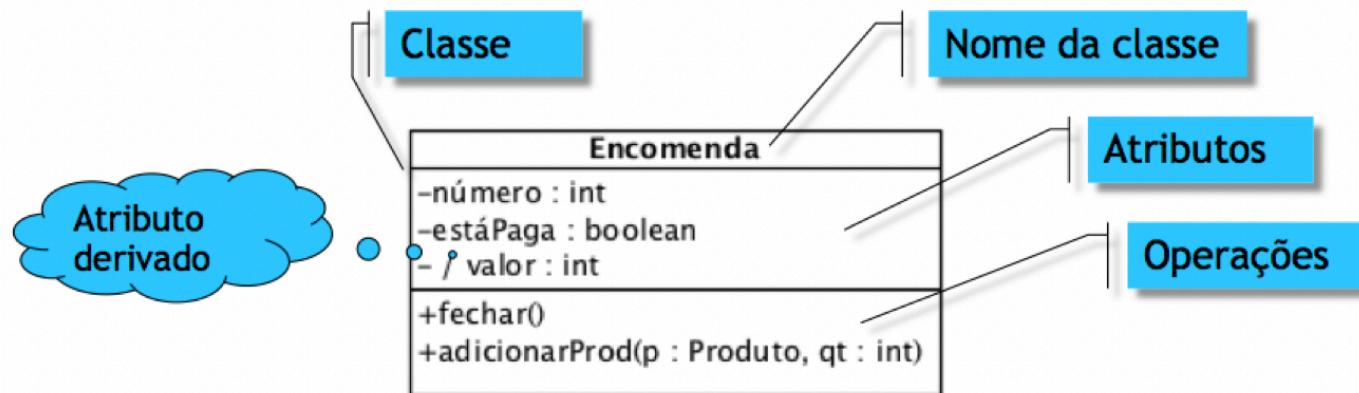
- para garantir o total encapsulamento do objecto as variáveis de instância devem ser declaradas como **private**
- ao ter encapsulamento total é necessário garantir que existem métodos que permitem o acesso e modificação das variáveis de instância.
- os métodos que se pretendem que sejam visíveis do exterior devem ser declarados como **public**

A classe Aluno

- Comecemos por pensar primeiro na arquitectura da classe
 - antes de codificar devemos “modelar” o estado e o comportamento dos objectos
 - vamos recorrer a uma versão simplificada de uma linguagem de modelação, a **UML**
 - modelamos estado, comportamento e visibilidade de v.i. e métodos de instância
 - modelamos o relacionamento entre classes

Breve introdução ao Diagrama Classes UML

- com excertos dos slides de José Campos
(copyrigth DSS 19/20)



- Compartimentos pré-definidos
 - Nome da classe – começa com maiúsculas / substantivo
 - Atributos (de instância) – representam propriedades das instâncias desta classe / começam com minúsculas / substantivos
 - Operações (de instância) – representam serviços que podem ser pedidos a instâncias da classe / começam com minúsculas / verbos

Visibilidade de atributos e operações

- O nível de visibilidade (acesso) que se pretende para cada atributo/operação é representado com as seguintes anotações:
 - privado – só acessível ao objecto a que pertence (cf. encapsulamento)
 - # protegido – acessível a instâncias das sub-classes (atenção: em Java fica também acessível a instâncias de classes do mesmo *package*!)
 - pacote/*package* – acessível a instâncias de classes do mesmo *package* (nível de acesso por omissão)
 - + público – acessível a todos os objectos no sistema (que conhecem o objecto a que o atributo/operação pertence!)

Declaração de atributos

- Atributos

«esterótipo» visibilidade / nome : tipo [multiplicidade] = valorInic {propriedades}

- Exemplos

morada

- morada= “Braga” {addedBy=“jfc”, date=“18/11/2011”}
- morada: String [1..2] {leaf, addOnly, addedBy=“jfc”}



Só o nome é obrigatório!

Declaração de operações

- Operações

Obrigatório!

in | out | inout | return

«esteréotipo» visibilidade nome (direção nomeParam : tipo = valorOmiss) : tipo

{propriedades}

- Exemplos

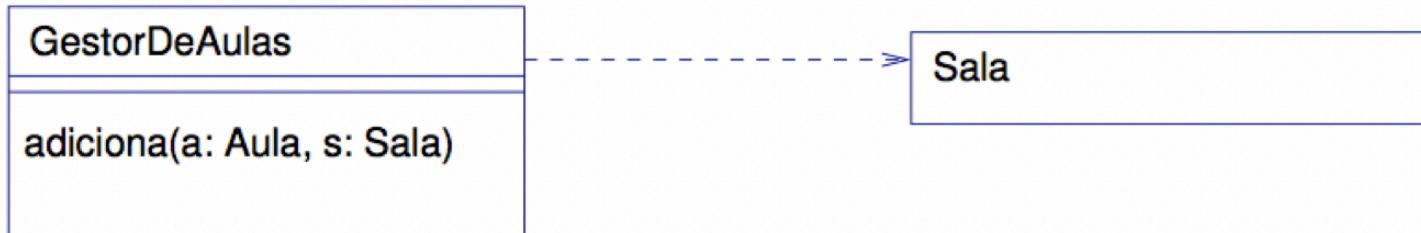
por omissão é “in”

setNome

```
+ setNome(nome = "SCX") {abstract}  
+ getNome() : String {isQuery, risco = baixo}  
# getNome(out nome) {isQuery}  
+ «create» Pessoa()
```

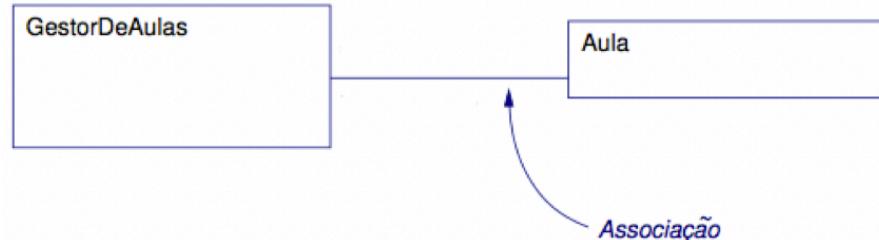
in - parâmetro de entrada
out - parâmetro de saída
inout - parâmetro de entrada/saída
return - operação retorna o parâmetro como um dos seus valores de retorno

Relação entre classes - dependência



- Indica que a definição de uma classe está dependente da definição de outra.
- Utiliza-se normalmente para mostrar que instâncias da origem utilizam, de alguma forma, instâncias do destino (por exemplo: um parâmetro de um método)
- Uma alteração no destino (quem é usado) pode alterar a origem (quem usa)

Relações entre classes - Associação



- Indica que objectos de uma estão ligados a objectos de outra – define uma relação entre os objectos
- Noção de naveabilidade (cf. diagramas E-R)
- Por omissão representam navegação bidireccional – mas pode indicar-se explicitamente o sentido da naveabilidade.



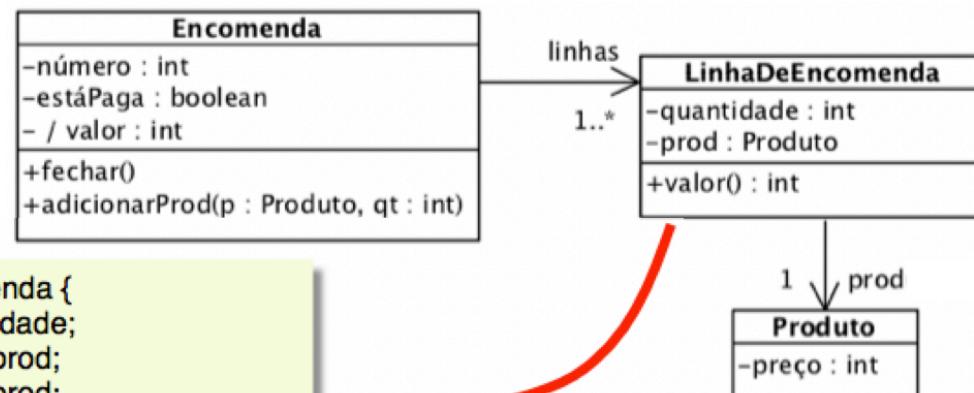
Associações vs Atributos

- Atributos (de instância) representam propriedades das instâncias das classes
 - São codificados como variáveis de instância
- Associações também representam propriedades das instâncias das classes
 - também são codificados como variáveis de instância
- Atributos devem ter tipos simples
 - **utilizar associações para tipos estruturados**

Erro de Principiante!

Repetir as associações nos atributos.

```
class LinhaDeEncomenda {  
    private int quantidade;  
    private Produto prod;  
    private Produto prod;  
  
    public int valor() {...}  
}
```



A classe Aluno: modelo

```
classDiagram
    class Aluno {
        -numero : String
        -nota : int
        -nome : String
        -curso : String
        +Aluno()
        +Aluno(numero : String, nota : int, nome : String, curso : String)
        +Aluno(umAluno : Aluno)
        +getNumero() : String
        +getNota() : int
        +getNome() : String
        +getCurso() : String
        +setNota(novaNota : int) : void
        +setNumero(numero : String) : void
        +setNome(nome : String) : void
        +setCurso(curso : String) : void
        +toString() : String
        +equals(o : Object) : boolean
        +clone() : Object
    }
```

variáveis de
instância

construtores

+

métodos de
instância

- esta abordagem permite-nos:
 - pensar na classe antes de a implementar
 - comunicar qual é o comportamento esperado
 - determinar quais são, e de que tipo, os parâmetros dos métodos
 - permite ao resto da equipa começar a trabalhar noutras classes sabendo qual é a estrutura da classe Aluno.

A classe Aluno: implementação

- declaração das variáveis de instância

```
/**  
 * Classe Aluno.  
 * Classe que modela de forma muito simples a  
 * informação e comportamento relevante de um aluno.  
 *  
 * @author MaterialPOO  
 * @version 20200216  
 */  
public class Aluno {  
  
    private String numero;  
    private int nota;  
    private String nome;  
    private String curso;
```

- construtores: vazio, parametrizado e de cópia

```
public Aluno() {  
    this.numero = "";  
    this.nota = 0;  
    this.nome = "";  
    this.curso = "";  
}
```

```
public Aluno(String numero, int nota, String nome, String curso) {  
    this.numero = numero;  
    this.nota = nota;  
    this.nome = nome;  
    this.curso = curso;  
}
```

```
public Aluno(Aluno umAluno) {  
    this.numero = umAluno.getNumero();  
    this.nota = umAluno.getNota();  
    this.nome = umAluno.getNome();  
    this.curso = umAluno.getCurso();  
}
```

● métodos getters e setters

```
/**  
 * Método que devolve o número de um aluno.  
 *  
 * @return String com o número do aluno  
 */  
public String getNumero() {  
    return this.numero;  
}
```

```
/**  
 * Método que devolve o nome de um aluno.  
 *  
 * @return String com o nome do aluno  
 */  
public String getNome() {  
    return this.nome;  
}
```

```
/**  
 * Método que devolve a nota de um aluno.  
 *  
 * @return int com o número do aluno  
 */  
public int getNota() {  
    return this.nota;  
}
```

```
/**  
 * Método que devolve o curso de um aluno.  
 *  
 * @return String com o número do aluno  
 */  
public String getCurso() {  
    return this.curso;  
}
```

```
public void setNota(int novaNota) {  
    this.nota = novaNota;  
}
```

```
public void setNumero(String numero) {  
    this.numero = numero;  
}
```

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

```
public void setCurso(String curso) {  
    this.curso = curso;  
}
```

Classe Aluno. Classe que modela de forma muito simples a informação e comportamento relevante de um aluno.

Version:

20200216

Author:

MaterialPOO

Constructor Summary

Constructors

Constructor

Description

`Aluno()`

Constructores para a classe Aluno

`Aluno(Aluno umAluno)`

`Aluno(java.lang.String numero, int nota, java.lang.String nome, java.lang.String curso)`

Method Summary

All Methods Instance Methods Concrete Methods

Modifier and Type	Method	Description
<code>Aluno</code>	<code>clone()</code>	Implementação do método de clonagem de um Aluno
<code>boolean</code>	<code>equals(java.lang.Object o)</code>	Implementação do método de igualdade entre dois Aluno Redefinição do método equals de Object.
<code>java.lang.String</code>	<code>getCurso()</code>	Método que devolve o curso de um aluno.
<code>java.lang.String</code>	<code>getNome()</code>	Método que devolve o nome de um aluno.
<code>int</code>	<code>getNota()</code>	Método que devolve a nota de um aluno.
<code>java.lang.String</code>	<code>getNumero()</code>	Método que devolve o número de um aluno.
<code>void</code>	<code>setCurso(java.lang.String curso)</code>	
<code>void</code>	<code>setNome(java.lang.String nome)</code>	
<code>void</code>	<code>setNota(int novaNota)</code>	
<code>void</code>	<code>setNumero(java.lang.String numero)</code>	
<code>java.lang.String</code>	<code>toString()</code>	Implementação do método toString comum na maioria das classes Java.