

Definição de Objecto

a noção de objecto é uma das definições essenciais do paradigma

assenta nos seguintes princípios:

independência do contexto (reutilização)

abstracção de dados (abstracção)

encapsulamento (abstracção e privacidade)

modularidade (composição)

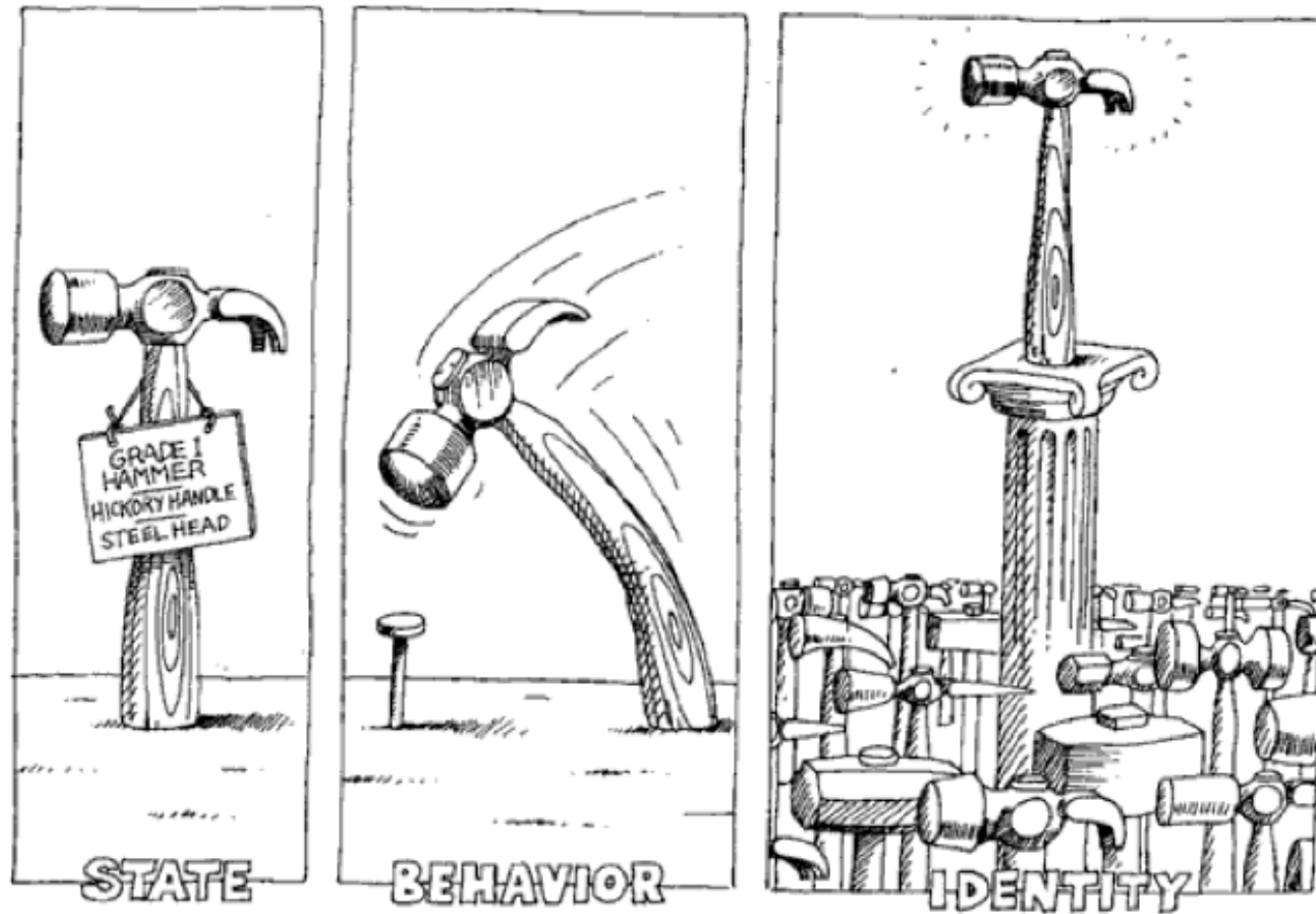
um objecto é o módulo computacional básico e único e tem como características:

identidade única

um conjunto de atributos privados (o **estado** interno)

um conjunto de operações que acedem ao estado interno e que constituem o **comportamento**. Algumas das operações são públicas e visíveis do exterior (a API)

Estado, Comportamento e Identidade



(OOAD with Applications, Grady Booch)

Objecto = “black box” (fechado, opaco)

apenas se conhecem os pontos de acesso
(as operações)

desconhece-se a implementação interna

Vamos chamar

aos dados: **variáveis de instância**

às operações: **métodos de instância**

Encapsulamento

Um objecto deve ser visto como uma “cápsula”, assegurando a protecção dos dados internos

Dados Privados
(v. instância)

API
pública

método 1
método 2
método 3
método 4

método privado

Um objecto que
representa um ponto
(com coordenadas no
espaço 2D)

o método

`xIguaiAy()` é privado
e não pode ser invocado
por entidades externas
ao objecto

ponto

var. instância

```
int coordx;  
int coordY;
```

métodos instância públicos

```
int getX()  
int getY()  
void setX(int cx)  
void setY(int cy)  
void deslocamento(int cx, int cy)  
...  
...
```

métodos instância privados

```
boolean xIguaiAy()
```

Um objecto é:

uma unidade computacional fechada e autónoma

capaz de realizar operações sobre os seus atributos internos

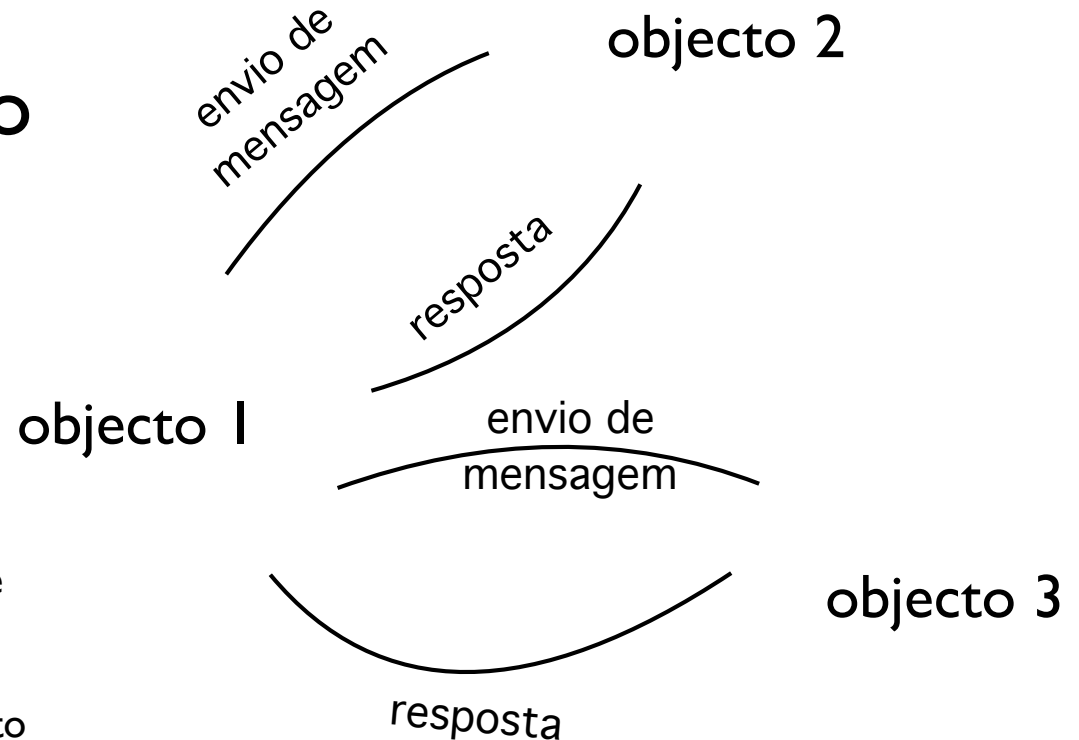
capaz de devolver respostas para o exterior, sempre que estas lhe sejam solicitadas

capaz de garantir uma gestão autónoma do seu espaço de dados interno

Mensagens

a interacção
entre objectos
faz-se através do
envio de
mensagens

dependendo da invocação de
método especificada, o envio
da mensagem origina uma
resposta por parte do objecto
receptor.



Novo alfabeto

o facto de termos agora um alfabeto de mensagens a que cada objecto responde, condiciona as frases válidas em POO

objecto.m()

objecto.m(arg1,...,argn)

r = objecto.m()

r = objecto.m(arg1,...,argn)

propositadamente, estão fora deste alfabeto
as frases:

r = o.var

o.var = x

em que se acede, de forma directa e
não protegida (não encapsulada), ao
campo **var** da estrutura interna do
objecto **o**

Se **ag** for um objecto que represente uma agenda, então poderemos fazer:

`ag.inserEvento("TestePOO",23,5,2019)`,
para inserir um novo evento na agenda

`ag.libertaEventosDia(25,4,2019)`, para
remover todos os eventos de um dia

`String[] ev = ag.getEventos(6,2019)`, para
obter as descrições de todos os eventos
do mês de Junho

...definição de Objecto

"An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable.", Grady Booch, 1993

Definição de Classe

numa linguagem por objectos, tudo são objectos, logo uma classe é um objecto “especial”

uma classe é um objecto que serve de padrão (molde, forma, template) para a criação de objectos similares (uma vez que possuem a mesma estrutura e comportamento)

aos objectos criados a partir de uma classe chamam-se *instâncias*

uma classe é um *tipo abstracto de dados* onde se especifica, quer a estrutura quer o comportamento das instâncias, que são criadas a partir dela

uma vez que todos os objectos criados a partir de uma classe respondem à mesma interface, i.e. o mesmo conjunto de mensagens, são exteriormente utilizáveis de igual forma

a classe pode ser vista como um *tipo de dados*

“The concepts of a class and an object are tightly interwoven, for we cannot talk about an object without regard for its class. However, there are important differences between these two terms. Whereas an object is a concrete entity that exists in time and space, a class represents only an abstraction, the “essence” of an object, as it were.

*Thus, we may speak of the class **Mammal**, which represents the characteristics common to all mammals. To identify a particular mammal in this class, we must speak of "this mammal" or "that mammal.", Grady Booch, 1993*

...e ainda sobre classes

“What isn't a class? An object is not a class, although, curiously, [...], a class may be an object. Objects that share no common structure and behavior cannot be grouped in a class because, by definition, they are unrelated except by their general nature as objects.”, Grady Booch, 1993

Construção de uma classe

para a definição do objecto classe é necessário

- identificar as variáveis de instância

- identificar as diferentes operações que constituem o comportamento dos objectos instância

Exemplo: Classe Ponto

um ponto no espaço 2D inteiro (X,Y)
possui como estado interno as duas
coordenadas

um conjunto de métodos que permitem
que os objectos tenham comportamento

métodos de acesso às coordenadas

métodos de alteração das coordenadas

outros métodos, que pertençam à lógica
de negócio expectável dos pontos

declaração da estrutura:

```
/**
 * Classe que implementa um Ponto num plano2D.
 * As coordenadas do Ponto são inteiras.
 *
 * @author anr (MaterialP00)
 * @version 20180212
 */
public class Ponto {

    //variáveis de instância
    private int x;
    private int y;
```

as variáveis de instância são privadas!

respeita-se o princípio do encapsulamento

declaração do comportamento:

métodos de construção de instâncias

métodos de acesso/manipulação das variáveis de instância

Construtores -
métodos que são
invocados quando
se cria uma
instância

não são métodos
de instância, são
métodos da
classe!

```
/**  
 * Construtor por omissão de Ponto.  
 */  
public Ponto() {  
    this.x = 0;  
    this.y = 0;  
}
```

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

```
/**  
 * Construtor de cópia de Ponto.  
 * Aceita como parâmetro outro Ponto e utiliza os métodos  
 * de acesso aos valores das variáveis de instância.  
 */  
public Ponto(Ponto umPonto) {  
    this.x = umPonto.getX();  
    this.y = umPonto.getY();  
}
```

a classe Ponto e duas instâncias (ponto1 e ponto2):

ponto

var. instância

```
int coordx;  
int coordY;
```

métodos instância públicos

```
int getX()  
int getY()  
void setX(int cx)  
void setY(int cy)  
void deslocamento(int cx, int cy)  
...  
...
```

métodos instância privados

```
boolean xIguarAy()
```

ponto1 : Ponto

private int x	2	Inspect
private int y	-5	Get

Show static fields Close

ponto2 : Ponto

private int x	10	Inspect
private int y	12	Get

Show static fields Close

ponto1 e *ponto2* são instâncias diferentes,
mas possuem o mesmo comportamento

não faz sentido replicar o comportamento
por todos os objectos

cada instância apenas tem de ter a
representação dos valores das suas
variáveis de instância

a informação do comportamento, os
métodos, está guardada no objecto classe

métodos de acesso e alteração do estado interno

getters e setters

por convenção, tem como nome `getX()` e `setX()`.

`getX()`, nas definições de Ponto

`getNota()`, nas definições de Aluno

`getReal()`, nas definições de
NúmeroComplexo

etc.

```
/**
 * Devolve o valor da coordenada em x.
 *
 * @return valor da coordenada x.
 */
public int getX() {
    return this.x;
}
```

```
/**
 * Devolve o valor da coordenada em y.
 *
 * @return valor da coordenada y.
 */
public int getY() {
    return this.y;
}
```

```
/**
 * Actualiza o valor da coordenada em x.
 *
 * @param novoX novo valor da coordenada em X
 */
public void setX(int novoX) {
    this.x = novoX;
}
```

```
/**
 * Actualiza o valor da coordenada em y.
 *
 * @param novoY novo valor da coordenada em Y
 */
public void setY(int novoY) {
    this.y = novoY;
}
```


outros métodos - decorrentes do domínio da entidade, isto é, o que representa e para que serve!

```
/**
 * Método que desloca um ponto somando um delta às coordenadas
 * em x e y.
 *
 * @param deltaX valor de deslocamento do x
 * @param deltaY valor de deslocamento do y
 */
public void deslocamento(int deltaX, int deltaY) {
    this.x += deltaX;
    this.y += deltaY;
}
```

```
/**
 * Método que soma as componentes do Ponto passado como parâmetro.
 * @param umPonto ponto que é somado ao ponto receptor da mensagem.
 */
public void somaPonto(Ponto umPonto) {
    this.x += umPonto.getX();
    this.y += umPonto.getY();
}
```

```
/**
 * Método que move o Ponto para novas coordenadas.
 * @param novoX novo valor de x.
 * @param novoY novo valor de y.
 */
public void movePonto(int cx, int cy) {
    this.x = cx; // ou setX(cx)
    this.y = cy; // ou this.setY(cy)
}
```

```
/**
 * Método que determina se o ponto está no quadrante positivo de x e y
 * @return booleano que é verdadeiro se x>0 e y>0
 */
public boolean ePositivo() {
    return (this.x > 0 && this.y > 0);
}
```

```
/**
 * Método que determina a distância de um Ponto a outro.
 * @param umPonto ponto ao qual se quer determinar a distância
 * @return double com o valor da distância
 */
public double distancia(Ponto umPonto) {

    return Math.sqrt(Math.pow(this.x - umPonto.getX(), 2) +
        Math.pow(this.y - umPonto.getY(), 2));
}
```

```
/**
 * Método que devolve a representação em String do Ponto.
 * @return String com as coordenadas x e y
 */
public String toString() {
    return "Cx = " + this.x + " Cy = " + this.y;
}
```

```
/**
 * Método que determina se dois pontos são iguais.
 * @return booleano que é verdadeiro se os valores das duas
 * coordenadas forem iguais
 */
public boolean iguais(Ponto umPonto) {
    return (this.x == umPonto.getX() && this.y == umPonto.getY());
}
```

```
/**
 * Método que determina se o módulo das duas coordenadas é o mesmo.
 * @return true, se as coordenadas em x e y
 * forem iguais em valor absoluto.
 */
private boolean xIgualAy() {
    return (Math.abs(this.x) == Math.abs(this.y));
}
```

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:

ponto1 : Ponto

private int x	2
private int y	-5

Inspect
Get

Show static fields Close

BlueJ: Method Result

```
// Devolve o valor da coordenada em x.  
//  
// @return valor da coordenada x.  
int getX()
```

ponto1.getX() returned:

int	2
-----	---

Inspect
Get

Close

ponto2 : Ponto

private int x	10
private int y	12

Inspect
Get

Show static fields Close

BlueJ: Method Result

```
// Devolve o valor da coordenada em x.  
//  
// @return valor da coordenada x.  
int getX()
```

ponto2.getX() returned:

int	10
-----	----

Inspect
Get

Close

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

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)	Atualiza o valor da coordenada em x.
void	setY (int novoY)	Atualiza 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

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 queremos 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)