

# As origens do Paradigma dos Objectos

- a maioria dos conceitos fundamentais da POO aparece nos anos 60 ligado a ambientes e linguagens de simulação
- a primeira linguagem a utilizar os conceitos da POO foi o SIMULA-67
  - era uma linguagem de modelação
  - permitia registar modelos do mundo real, nomeadamente para aplicações de simulação (trânsito, filas espera, etc.)

- o objectivo era representar entidades do mundo real:
  - identidade (única)
  - estrutura (atributos)
  - comportamento (acções e reacções)
  - interacção (com outras entidades)

- Introduz-se o conceito de “classe” como a entidade definidora e geradora de todos os “indivíduos” que obedecem a um dado padrão de:
  - estrutura
  - comportamento
- Classes são fábricas/padrões/formas/templates de *indivíduos*
- a que chamaremos de “objectos”!

# Passagem para POO

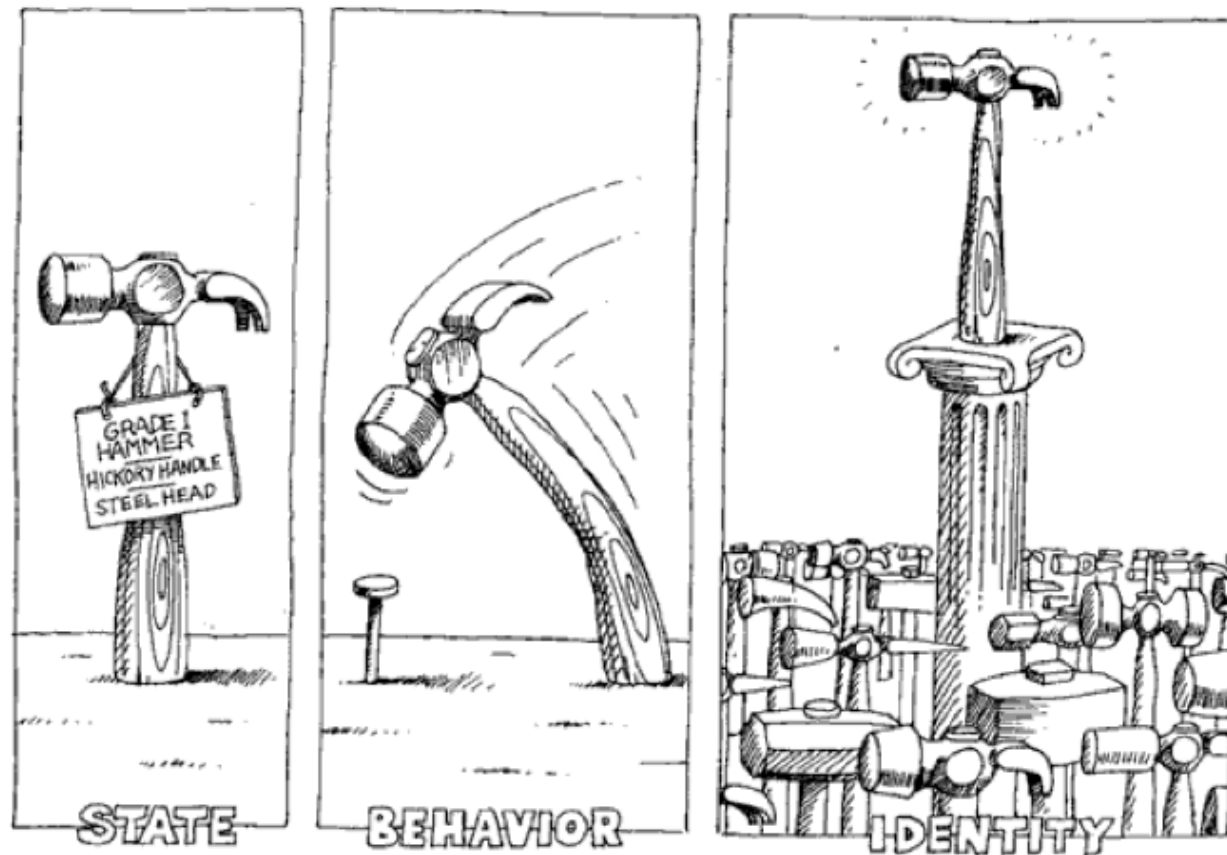
- Um objecto é a representação computacional de uma entidade do mundo real, com:
  - atributos (necessariamente) privados
  - operações
- **Objecto** = Dados Privados (variáveis de instância) + Operações (métodos)

# 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



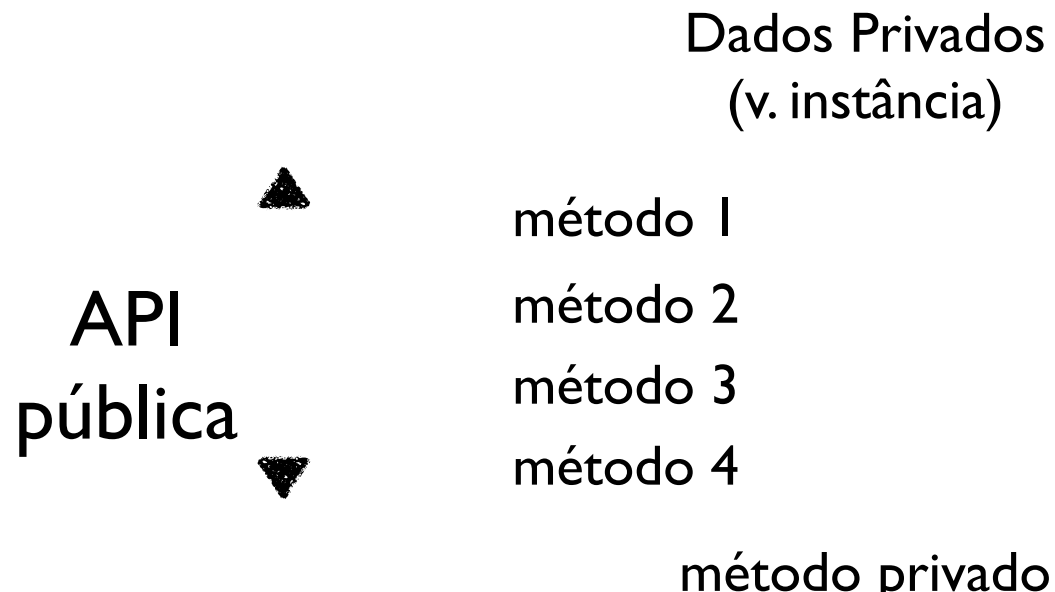
(OOAD with Applications, Grady Booch)

- 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



- Um objecto que representa um ponto (com coordenadas no espaço 2D)
- o método `xIgualAy()` 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)  
...  
...
```

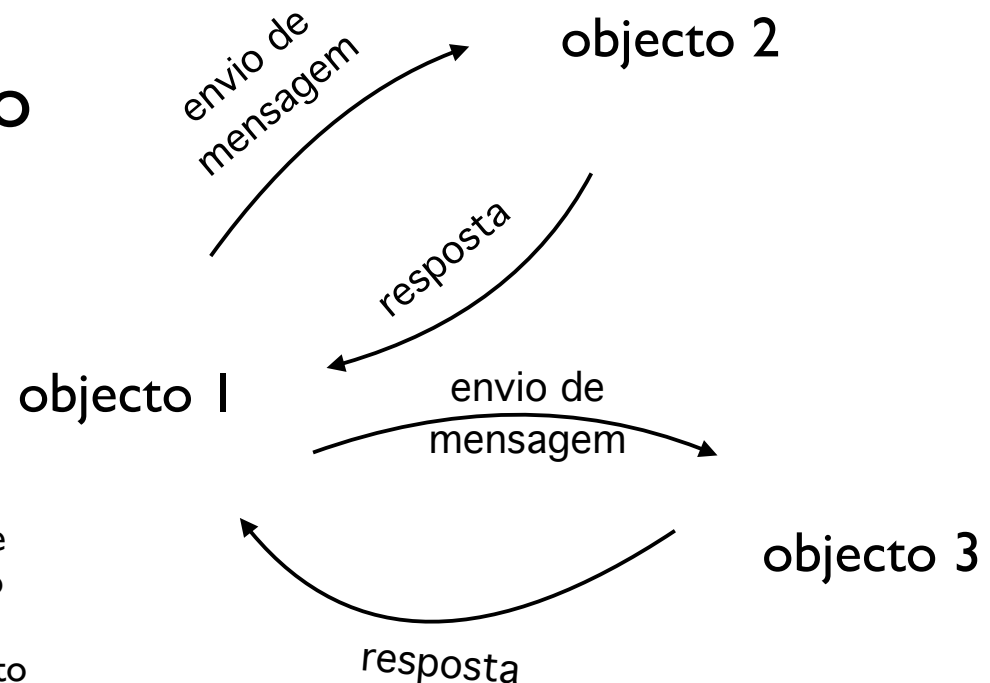
métodos instância privados

```
boolean xIgualAy()
```

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

- Se **ag** for um objecto que represente uma agenda, então poderemos fazer:
- `ag.inserEvento("TestePOO",20,6,2022)`, para inserir um novo evento na agenda
- `ag.libertaEventosDia(10,6,2022)`, para remover todos os eventos de um dia
- `String[] ev = ag.getEventos(6,2022)`, 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
-x : int -y : int
+Ponto() +Ponto(cx : int, cy : int) +Ponto(umPonto : Ponto) +getX() : int +getY() : int +setX(novoX : int) : void +setY(novoY : int) : void +deslocamento(deltaX : int, deltaY : int) : void +somaPonto(umPonto : Ponto) : void +movePonto(cx : int, cy : int) : void +ePositivo() : boolean +distancia(umPonto : Ponto) : double +iguais(umPonto : Ponto) : boolean -xIguaiAy() : boolean +toString() : String +equals(o : Object) : boolean +clone() : Object

ponto1 : Ponto

private int x	<input type="text" value="2"/>	Inspect
private int y	<input type="text" value="-5"/>	Get

Show static fields Close

ponto2 : Ponto

private int x	<input type="text" value="10"/>	Inspect
private int y	<input type="text" value="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));
}
```

diagrama UML