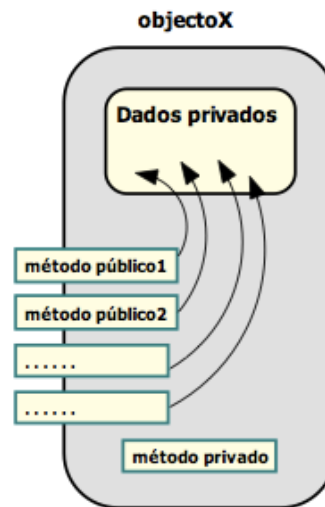
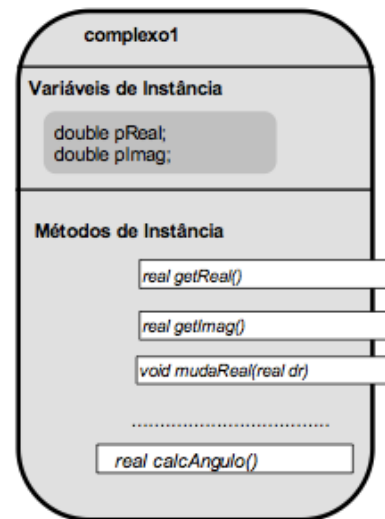


Encapsulamento

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



- Um objecto que representa um número complexo:

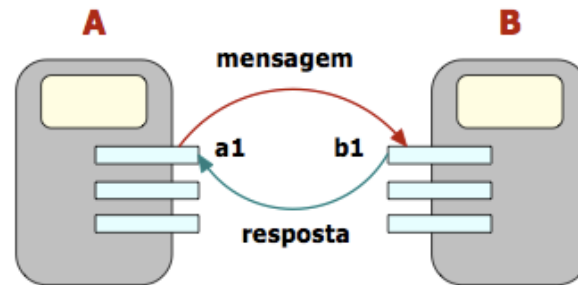


- O método `calcAngulo()` é interno e não pode ser invocado por entidades externas

- 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



Novo alfabeto

- o facto de termos agora um alfabeto de mensagens a que cada objecto responde, altera 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, ao campo **var** da estrutura interna do objecto **o**

- Se **ag** for um objecto que represente uma agenda, então poderemos fazer:
- `ag.inserEvento("TestePOO",9,6,2015)`, para inserir um novo evento na agenda
- `ag.libertaEventosDia(25,4,2015)`, para remover todos os eventos de um dia
- `String[] ev = ag.getEventos(6,2015)`, para obter a descrição de todos os eventos do mês de Junho

...noção 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) 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*

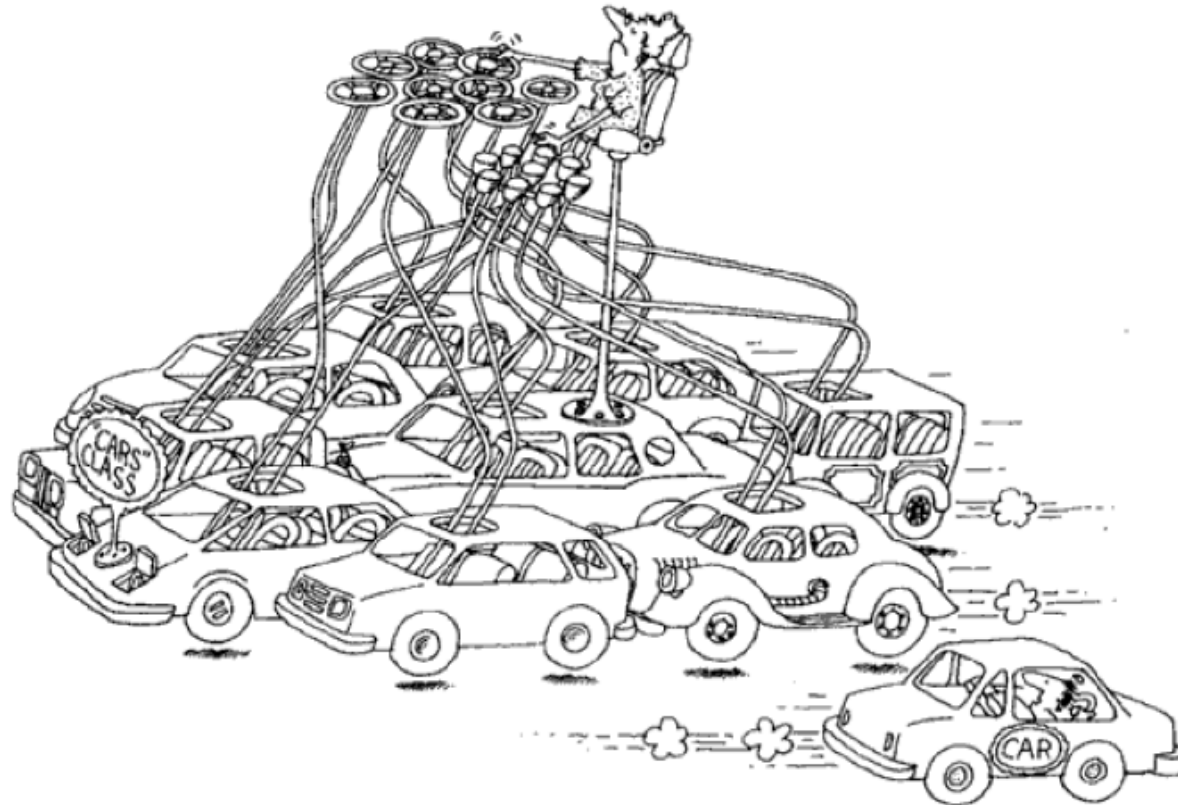
- uma classe é um *módulo* onde se especifica, quer a estrutura, quer o comportamento das instâncias, que a partir dela criamos
- uma vez que todos os objectos criados a partir de uma classe respondem à mesma interface, i.e. o mesmo conjunto de mensagens, são exteriormente utilizáveis de igual forma
- 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*

Uma classe e as suas instâncias

- *A class is a set of objects that share a common structure and a common behavior, Grady Booch, 1993*



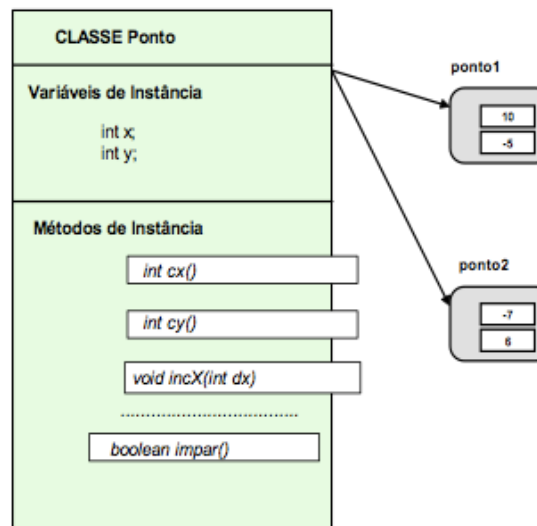
...e ainda sobre classes

- *“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*

Classe Ponto 2D

- 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

- a classe Ponto2D e duas instâncias:

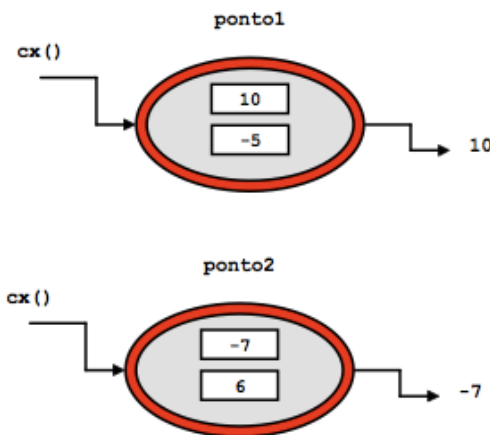


- *ponto1* e *ponto2* são instâncias diferentes, mas possuem o mesmo comportamento
- não faz sentido replicar o comportamento por todos os objectos

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 `cx()` a cada um dos pontos 2D, 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

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

- declaração da estrutura:

```
/**
 * Write a description of class Ponto here.
 *
 * @author anr
 * @version 2010/11
 */
import static java.lang.Math.abs;

public class Ponto2D {

    // Variáveis de Instância
    private double x, y;

    // Constantes variáveis
```

- as variáveis de instância são privadas!
- respeita-se o princípio do encapsulamento
- declaração do comportamento:
 - métodos de construção de instâncias
 - métodos de acesso às v. instância

- Construtores - métodos que são invocados quando se cria uma instância
- não são métodos de instância, são métodos da classe!

```
// Construtores usuais
public Ponto2D(double x, double y) { this.x = x; this.y = y; }
public Ponto2D(){ this(0.0, 0.0); } // usa o outro construtor
public Ponto2D(Ponto2D p) { x = p.getX(); y = p.getY(); }
```

- Utilização na criação de instâncias:
 - Ponto2D p = new Ponto2D(2.0,3.0);
 - Ponto2D r = new Ponto2D();

- métodos de acesso e alteração do estado interno
- *getters* e *setters*
- por convenção tem como nome getX() e setX(). Ex: getNotaAluno()

```
// Métodos de Instância
public double getX() { return this.x; }
public double getY() { return this.y; }

public void setX(double x) {this.x = x;}
public void setY(double y) {this.y = y;}
```

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

```
/** incremento das coordenadas */
public void incCoord(double dx, double dy) {
    this.x += dx; this.y += dy;
}

/** decremento das coordenadas */
public void decCoord(double dx, double dy) {
    this.x -= dx; this.y -= dy;
}

/** soma as coordenadas do ponto parâmetro ao ponto receptor */
public void somaPonto(Ponto2D p) {
    this.x += p.getX(); this.y += p.getY();
}

/** soma os valores parâmetro e devolve um novo ponto */
public Ponto2D somaPonto(double dx, double dy) {
    return new Ponto2D(this.x += dx, this.y += dy);
}

/* determina se um ponto é simétrico (dista do eixo dos XX o
mesmo que do eixo dos YY) */
public boolean simetrico() {
    return abs(this.x) == abs(this.y);
}

/** verifica se ambas as coordenadas são positivas */
public boolean coordPos() {
    return this.x > 0 && this.y > 0;
}
```

Utilização de uma classe

- 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 TestePonto {
    // Classe de teste da Classe Ponto.
    public static void main(String args[]) {
        // Criação de Instâncias
        Ponto pt1, pt2, pt3;
        pt1 = new Ponto();
        pt2 = new Ponto(2, -1);
        pt3 = new Ponto(0, 12);

        // Utilização das Instâncias
        int cx1, cx2, cx3;    // variáveis auxiliares
        int cy1, cy2, cy3;    // variáveis auxiliares
        cx1 = pt1.getx();
        cx2 = pt2.getx();

        // saída de resultados para verificação
        System.out.println("cx1 = " + cx1);
        System.out.println("cx2 = " + cx2);

        // alterações às instâncias e novos resultados

        pt1.incCoord(4,4); pt2.incCoord(12, -3);
        cx1 = pt1.getx(); cx2 = pt2.getx();
        cx3 = cx1 + cx2;
        System.out.println("cx1 + cx2 = " + cx3);

        pt3.decCoord(10, 20); pt2.decCoord(5, -10);
        cy1 = pt2.gety(); cy2 = pt3.gety();
        cy3 = cy1 + cy2;
        System.out.println("cy1 + cy2 = " + cy3);
    }
}

```


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 muito normal é quando queremos desambiguar e identificar as variáveis de instância

- Por exemplo, em

```
// Métodos de Instância  
public double getX() { return this.x; }  
public double getY() { return this.y; }  
  
public void setX(double x) {this.x = x;}  
public void setY(double y) {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 método da classe

```
// modificador - decrementa Coordenada X
void decCoordX(int deltaX) {
    this.decCoord(deltaX, 0);    // invoca decCoord() local
}
```

- no caso de não termos escrito apenas `decCoord(deltaX,0)`
- o compilador teria acrescentado automaticamente a referência *this*