

Ainda sobre classes

“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.

a definição de classe que temos vindo a utilizar está incompleta

quer as instâncias, quer as classes são
objectos

Variáveis e métodos de classe

as classes não deixam de ser objectos

objectos que guardam o que é comum a todas as instâncias

apenas um objecto-classe por classe

se objectos possuem estado e comportamento, então podemos extrapolar e dizer que a classe também tem:

variáveis e métodos de classe

os métodos de classe são activados a partir de mensagens que são enviadas para a classe

exemplo: Ponto.metodo()

se uma classe possui variáveis de classe o acesso a essas variáveis deverá ser feito através dos métodos de classe

métodos de instância => var. instância
métodos de classe => var. classe

o que é que se pode guardar como variável de classe?

valores que sejam comuns a todas os objectos instância

não faz sentido colocar estes valores em todos os objectos (repetição)

Imagine-se que numa classe Factura se pretende:

saber quantas facturas (quantas instâncias) foram criadas (é possível saber quantas existem?)

definir uma taxa de imposto (ex: IVA) comum a todas as facturas

como é que poderemos satisfazer os requisitos expressos em a) e b)?

uma variável que guarde o número de facturas criadas não é certamente uma variável de instância

actualização do contador em todas as instâncias?

redundância?

teriam de se ter implementados mecanismos (complexos) de comunicação entre todas as instâncias!!

as variáveis de classe servem para guardar *informação global* a todas as instâncias

podem também ser utilizadas para guardar constantes que são utilizadas pelos diversos objectos instância

como já vimos, por exemplo com `Math.PI` (e outros)

os métodos de classe fazem o acesso às variáveis de classe

aos métodos de classe aplicam-se as mesmas regras de visibilidade que se aplicam aos métodos de instância

os métodos de classe são sempre acessíveis às instâncias, mas métodos de classe **não** tem acesso aos métodos de instância

como se declaram métodos e variáveis de classe?

utilizando o prefixo **static**

a definição de classe passa a ter:

declaração de variáveis de classe

declaração de métodos de classe

declaração de variáveis de instância

declaração de métodos de instância

Estrutura de uma classe

estrutura tipo de uma
classe

para que possa ser
utilizada pelos
potenciais clientes

esta definição irá ser
refinada ao longo do
semestre

Classe

Variáveis de Classe

Métodos de Classe

Construtores

Variáveis Instância

Métodos de Instância

(c:/ getters, setters, equals, toString,
clone)

Classe Factura (exemplo)

seja uma classe Factura, que modela de forma muito simples o conhecido conceito

queremos representar a informação que permitirá ao criar uma factura ajustar o valor ao imposto que na altura está em vigor.

terá de ser uma definição global a todas as instâncias

uma variável de classe e respectivos métodos

```
public class Factura {  
    //variáveis de instância  
    private String nomeEmpresa;  
    private String nif;  
    private String descricaoDespesa;  
    private LocalDate dataFact;  
    private double valorFact;  
    private double valorAPagar; //pelo cliente, depois de aplicado o imposto  
  
    //a taxa de imposto é definida globalmente para todas  
    //as facturas e na altura da emissão deve utilizar-se  
    //a taxa de imposto actual.  
    private static double taxaImposto;  
  
    public static double getTaxaImposto() {  
        return taxaImposto;  
    }  
  
    public static void setTaxaImposto(double tx) {  
        taxaImposto = tx;  
    }  
}
```

o valor a pagar é calculado aquando da invocação do construtor

```
public Factura(String nomeEmpresa, String nif, String descricaoDespesa,
               LocalDate dataFact, double valorFact) {
    this.nomeEmpresa = nomeEmpresa;
    this.nif = nif;
    this.descricaoDespesa = descricaoDespesa;
    this.dataFact = dataFact;
    this.valorFact = valorFact;
    this.valorAPagar = this.valorFact * (1 + getTaxaImposto());
}
```

o método de classe utilizado no construtor poderia ter sido invocado,

enviando o método ao objecto onde está definido, a classe Factura

na forma:

```
this.valorAPagar = this.valorFact * (1 + Factura.getTaxaImposto());
```

Nota: ajuda seguir a convenção que diz que as classes começam por letra maiúscula

Estruturas de Dados

Como já vimos atrás, podemos criar conceitos mais complexos através do mecanismo de composição/agregação de objectos de classes mais simples:

- a Turma a partir de Aluno

- o Stand a partir de Veículo

- etc.

Mas para isso precisamos de ter colecções de objectos...

Até ao momento apenas temos disponível a utilização de arrays

```
Aluno[] alunos = new Aluno[30];  
Veiculo[] carros = new Veiculo[10];  
  
for (int i=0; i<alunos.length && !encontrado; i++)  
    if (alunos[i].getNota() == 20)  
        encontrado = true;  
  
for(Veiculo v: carros)  
    System.out.println(v.toString());
```


Esta é uma solução simples e testada, com a inconveniente de o tamanho da estrutura de dados ser estaticamente definido.

Será que conseguimos ter uma estrutura de dados, baseada em arrays, que pudesse crescer de forma transparente para o utilizador?

(primeira solução) Sim, bastando para tal criarmos uma classe com esse comportamento

Seja essa classe chamada de GrowingArray e, por comodidade, vamos utilizar instâncias de Circulo

Que operações necessitamos:

- adicionar um círculo (no fim e numa posição)

- remover um círculo

- ver se um círculo existe

- dar a posição de um círculo na estrutura

- dar número de elementos existentes

Documentação com os métodos necessários:

Constructor Summary

Constructors

Constructor and Description

`GrowingArray()`

`GrowingArray(int capacidade)`

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type	Method and Description
void	<code>add(MyDynamicArray.Circulo c)</code> Adiciona o elemento passado como parâmetro ao fim do array
void	<code>add(int indice, MyDynamicArray.Circulo c)</code> Método que adiciona um elemento numa determinada posição, forçando a que os elementos à direita no array façam shift.
boolean	<code>contains(MyDynamicArray.Circulo c)</code> Método que determina se um elemento está no array.
<code>MyDynamicArray.Circulo</code>	<code>get(int indice)</code> Devolve o elemento que está na posição indicada.
int	<code>indexOf(MyDynamicArray.Circulo c)</code> Método que determina o índice do array onde está localizada a primeira ocorrência de um objecto.
boolean	<code>isEmpty()</code> Método que determina se o array contém elementos, ou se está vazio.
boolean	<code>remove(MyDynamicArray.Circulo c)</code>

Declarações iniciais:

```
public class GrowingArray {  
  
    private Circulo[] elementos;  
    private int size;  
  
    /**  
     * variável que determina o tamanho inicial do array,  
     * se for utilizado o construtor vazio.  
     */  
    private static final int capacidade_inicial = 20;  
  
    public GrowingArray(int capacidade) {  
        this.elementos = new Circulo[capacidade];  
        this.size = 0;  
    }  
  
    public GrowingArray() {  
        this(capacidade_inicial);  
    }  
}
```

invocação do
construtor anterior

get de um elemento da estrutura de dados

```
/**
 * Devolve o elemento que está na posição indicada.
 *
 * @param indice posição do elemento a devolver
 * @return o objecto que está na posição indicada no parâmetro
 * (deveremos ter atenção às situações em que a posição não existe)
 */

public Circulo get(int indice) {
    if (indice <= this.size)
        return this.elementos[indice];
    else
        return null; // ATENÇÃO!
}
```

set de uma posição da estrutura

```
/**
 * Método que actualiza o valor de uma determinada posição do array.
 *
 * @param indice a posição que se pretende actualizar
 * @param c o circulo que se pretende colocar na estrutura de dados
 *
 */
public void set(int indice, Circulo c) {
    if (indice <= this.size) //não se permitem "espaços vazios"
        this.elementos[indice] = c;
}
```

as estruturas de dados que vamos estudar
nunca chamam o clone

tem de ser feito por quem as utiliza!

adicionar um elemento à estrutura de dados

```
/**
 * Adiciona o elemento passado como parâmetro ao fim do array
 *
 * @param c circulo que é adicionado ao array
 *
 */

public void add(Circulo c) {
    aumentaCapacidade(this.size + 1);
    this.elementos[this.size++] = c;
}
```

método auxiliar que aumenta espaço

```
/**
 * Método auxiliar que verifica se o array alocado tem capacidade
 * para guardar mais elementos.
 * Por cada nova inserção, verifica se estamos a mais de metade
 * do espaço
 * alocado e, caso se verifique, aloca mais 1.5 de capacidade.
 *
 */

private void aumentaCapacidade(int capacidade) {
    if (capacidade > 0.5 * this.elementos.length) {
        int nova_capacidade = (int)(this.elementos.length * 1.5);
        this.elementos = Arrays.copyOf(this.elementos, nova_capacidade);
    }
}
```



```

/**
 * Método que adiciona um elemento numa determinada posição,
 * forçando a
 * que os elementos à direita no array façam shift.
 * Tal como no método de set não são permitidos espaços.
 *
 * @param indice indice onde se insere o elemento
 * @param c circulo que será inserido no array
 *
 */

public void add(int indice, Circulo c) {
    if (indice <= this.size) {
        aumentaCapacidade(this.size+1);
        System.arraycopy(this.elementos, indice, this.elementos,
                        indice + 1, this.size - indice);
        this.elementos[indice] = c;
        this.size++;
    }
}

```

```

/**
 * Remove do array o elemento que está na posição indicada no parâmetro.
 * Todos os elementos à direita do índice sofrem um deslocamento
 * para a esquerda.
 * @param indice índice do elemento a ser removido
 * @return o elemento que é removido do array. No caso do índice não
 * existir devolver-se-á null.
 */
public Circulo remove(int indice) {
    if (indice <= this.size) {
        Circulo c = this.elementos[indice];

        int deslocamento = this.size - indice - 1;
        if (deslocamento > 0)
            System.arraycopy(this.elementos, indice+1, this.elementos,
                             indice, deslocamento);
        this.elementos[--this.size] = null;
        return c;
    }
    else
        return null;
}

```

```

* Remove a primeira ocorrência do elemento que é passado como parâmetro.
* Devolve true caso o array contenha o elemento, falso caso contrário.
*
* @param c círculo a ser removido do array (caso exista)
* @return true, caso o círculo exista no array
*/
public boolean remove(Circulo c) {
    if (c != null) {
        boolean encontrado = false;
        for (int indice = 0; indice < this.size && !encontrado; indice++)
            if (c.equals(this.elementos[indice])) {
                encontrado = true;
                int deslocamento = this.size - indice - 1;
                if (deslocamento > 0)
                    System.arraycopy(this.elementos, indice+1,
                                     this.elementos, indice, deslocamento);
                this.elementos[--this.size] = null;
            }
        return encontrado;
    }
    else
        return false;
}

```

```

/**
 * Método que determina o índice do array onde está localizada a
 * primeira ocorrência de um objecto.
 *
 * @param c círculo de que se pretende determinar a posição
 * @return a posição onde o círculo se encontra. -1 caso não esteja no
 * array ou o círculo passado como parâmetro seja null.
 */

public int indexOf(Circulo c) {
    int posicao = -1;
    if (c != null) {
        boolean encontrado = false;
        for (int i = 0; i < this.size && !encontrado; i++)
            if (c.equals(this.elementos[i])) {
                encontrado = true;
                posicao = i;
            }
    }
    return posicao;
}

```

invocação do
método equals de Circulo

```
/**
 * Método que determina se um elemento está no array.
 *
 * @param c círculo a determinar se está no array
 * @return true se o objecto estiver inserido na estrutura de dados,
 * false caso contrário.
 */
public boolean contains(Circulo c) {
    return indexOf(c) >= 0;
}
```

```
/**
 * Método que determina se o array contém elementos, ou se está vazio.
 *
 * @return true se o array estiver vazio, false caso contrário.
 */
public boolean isEmpty() {
    return this.size == 0;
}
```

```
public class TesteGA {  
    public static void main(String[] args) {  
  
        Circulo c1 = new Circulo(2,4,4.5);  
        Circulo c2 = new Circulo(1,4,1.5);  
        Circulo c3 = new Circulo(2,7,2.0);  
        Circulo c4 = new Circulo(3,3,2.0);  
        Circulo c5 = new Circulo(2,6,7.5);  
  
        GrowingArray ga = new GrowingArray(10);  
        ga.add(c1.clone());  
        ga.add(c2.clone());  
        ga.add(c3.clone());  
  
        System.out.println("Num elementos = " + ga.size());  
        System.out.println("Posição do c2 = " + ga.indexOf(c2));  
    }  
}
```