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**

- 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 enviados para o objecto classe.
  - exemplo: Ponto2D.metodo()
- se uma classe possui variáveis de classe o acesso a essas variáveis deverá ser feito através dos métodos de classe
  - métodos de instância => v. instância
  - métodos de classe => v. 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 na classe Conta Bancária se pretende:

  a) saber quantas contas foram criadas

  b) saber qual é o somatório dos saldos das contas existentes

- se para b) podemos ter outras soluções (quais?), como é que poderemos satisfazer o requisito expresso em a)?

- uma variável que guarde o número de contas não é certamente uma variável de instância
  - actualização do contador em todas as instâncias?
  - redundância?
- teriam de se ter implementados mecanismos de comunicação entre todas as instâncias!!

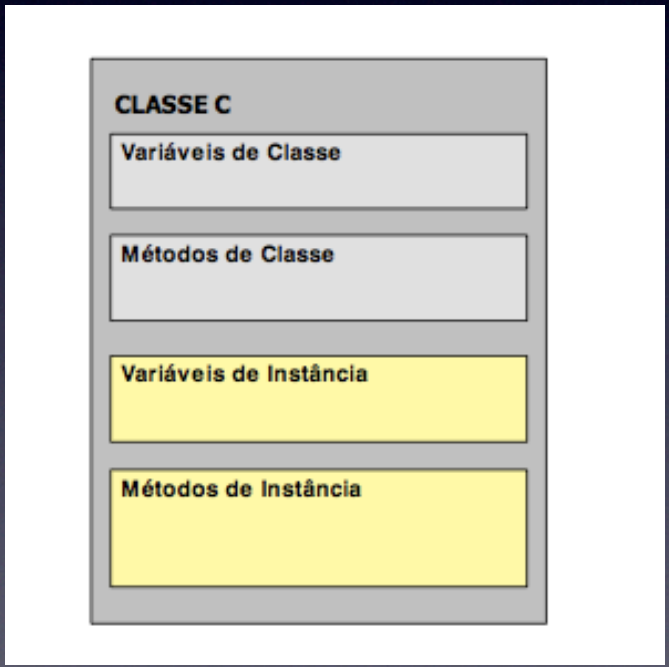
- as variáveis de classe servem para guardar *informação global* a todas as instâncias
- podem também ser utilizadas para guardar constantes que são utilizadas pelos diversos objectos instância
  - exemplo: Math.PI

- 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 varíaveis de instância
  - declaração de métodos de instância

Estrutura de uma classe

- estrutura tipo de uma classe



Classe PMMB

- seja uma classe PMMB (porta moedas multibanco)
  - queremos acrescentar informação sobre o número total de PMMB's criados e sobre o valor total de saldo existente em todos os cartões

- duas variáveis de classe e respectivos métodos de acesso

```
public class PMMB {

   // Variáveis de Classe
   public static int numPMMB = 0;
   public static double saldoTotal = 0.0;

   // Métodos de Classe
   public static int getNumPMMB() {
      return numPMMB;
   }
   public static double getSaldoTotal() {
      return saldoTotal;
   }
   public static void incNumPMMB() {
      numPMMB++;
   }
   public static void actSaldoTotal(double valor) {
      saldoTotal += valor;
   }
```

```
// Variáveis de Instância
private String codigo;
private String titular;
private double saldo;
private int numMovs;  // total de movimentos
```

- onde é que se actualiza a informação do número de cartões existentes?
  - no construtor de PMMB

```
public PMMB() {
   codigo = ""; titular = "";
   saldo = 0.0; numMovs = 0;
   this.actSaldoTotal(0);
   this.incNumPMMB();
}
```

  - no construtor, *this* representa a classe, sempre que o método invocado seja de classe.

```
// Métodos de Instância
. . . .
public void carregaPM(double valor) {
   saldo = saldo + valor;
   numMovs++; actSaldoTotal(valor);
}
// pré-condição
public boolean prePaga(double valor) {
   return saldo >= valor ;
}

public void pagamento(double valor) {
   saldo = saldo – valor;
   numMovs++; actSaldoTotal(-valor);
}
```

- para não confundir quem lê, as invocações anteriores poderiam ter sido feitas na forma:
  - PMMB.actSaldoValorTotal(valor)

- os métodos de classe anteriormente definidos devem ser utilizados como:

```
int pMoedas = PMMB.getNumPMMB();
double saldos = PMMB.getSaldoTotal();
```

```
PMMB.incNumPMMB();        //
PMMB.actSaldoTotal(valor);
```

- para que se consiga perceber o código é necessário que se siga 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?
  - 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 circulo (no fim e numa posição)
  - remover um círculo
  - ver se um circulo existe
  - dar a posição de um circulo na estrutura
  - dar número de elementos existentes

- Documentação com os métodos necessários:

Constructor Summary

`GrowingArray()`	
`GrowingArray(int capacidade)`	

Method Summary

void	`add(Circulo c)` Adiciona o elemento passado como parâmetro ao fim do array
void	`add(int indice, Circulo c)` Método que adiciona um elemento numa determinada posição, forçando a que os elementos à direita no array façam shift.
boolean	`contains(Circulo c)` Método que determina se um elemento está no array.
Circulo	`get(int indice)` Devolve o elemento que está na posição indicada.
int	`indexOf(Circulo c)` Método que determina o índice do array onde está localizada a primeira ocorrência de um objecto.
boolean	`isEmpty()` Método que determina se o array contém elementos, ou se está vazio.
boolean	`remove(Circulo c)` Remove a primeira ocorrência do elemento que é passado como parâmetro.
Circulo	`remove(int indice)` Remove do array o elemento que está na posição indicada no parâmetro.
void	`set(int indice, Circulo c)` Método que actualiza o valor de uma determinada posição do array.
int	`size()` Método que determina o tamanho do array de elementos.

- 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);
  }
```

- 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;
}
```

- 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);
   }
}
```

```java
/**
 * 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++;
  }
}
```

```java
/**
 * 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;
}
```

```java
 * 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;
}
```

```java
/**
 * 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;
}
```

```java
/**
 * 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));

  }
}
```