Igualdade de objectos

- Como implementar os métodos
  - `public boolean existeAluno(Aluno a)`
  - `public void removeAluno(Aluno a)`
- como é que determinamos se o objecto está efectivamente dentro do array de alunos?

- A solução
- `alunos[i] == a`, não é eficaz porque compara os apontadores (e pode ter havido previamente um clone)
  - `(alunos[i]).getNumero() == a.getNumero()`, assume demasiado sobre a forma como se comparam alunos
- Quem é a melhor entidade para determinar como é que se comparam objectos do tipo Aluno?

- através da disponibilização de um método, na classe Aluno, que permita comparar instâncias de alunos
  - é importante que esse método seja universal, isto é, que tenha sempre a mesma assinatura
  - é importante que todos os objectos respondam a este método
- **public boolean equals(Object o)**

- dessa forma o método `existeAluno(Aluno a)` da classe Turma, assume a seguinte forma:

```
public boolean existeAluno(Aluno umAluno) {
  boolean resultado = false;

  if (umAluno != null) {

    for(int i=0; i< this.ocupacao && !resultado; i++)
      resultado = this.alunos[i].equals(umAluno);

    return resultado;
  }
  else
    return false;
}
```

- Em resumo:
  - o método de igualdade é determinante para que seja possível ter colecções de objectos
  - o método de igualdade entre objectos de uma classe não pode ser codificado a não ser pela classe: abstração de dados
  - existe um conjunto de regras básicas que todos os métodos de igualdade devem respeitar

O método equals

- a assinatura é:

```
public boolean equals(Object o)
```

- é importante referir, antes de explicar em detalhe o método, que:

O método equals

- a relação de equivalência que o método implementa:
- é **reflexiva**, ou seja x.equals(x) == true, para qualquer valor de x que não seja nulo
- é **simétrica**, para valores não nulos de x e y se x.equals(y) == true, então y.equals(x) == true

- é **transitiva**, em que para x,y e z, não nulos, se x.equals(y) == true, y.equals(z) == true, então x.equals(z) == true

- é **consistente**, dado que para x e y não nulos, sucessivas invocações do método equals (x.equals(y) ou y.equals(x)) dá sempre o mesmo resultado

- para valores nulos, a comparação com x, não nulo, dá como resultado false.

- quando os objectos envolvidos sejam o mesmo, o resultado é true, ie, x.equals(y) == true, se x == y
  - dois objectos são iguais se forem o mesmo, ie, se tiverem o mesmo apontador
- caso não se implemente o método equals, temos uma implementação, por omissão, com o seguinte código:

```
public boolean equals(Object object) {
        return this == object;
}
```

- template típico de um método equals

```
public boolean equals(Object o) {

    if (this == o)
      return true;

    if((o == null) || (this.getClass() != o.getClass()))
      return false;

    <CLASSE> m = (<CLASSE>) o;
    return ( <condições de igualdade> );

}
```

- o método equals da classe Aluno

```java
/**
* Implementação do método de igualdade entre dois Aluno
* Redefinição do método equals de Object.
*
* @param  umAluno   aluno que é comparado com o receptor
* @return      booleano true ou false
*/
public boolean equals(Object o) {
  if (this == o)
    return true;

  if((o == null) || (this.getClass() != o.getClass()))
    return false;

  Aluno umAluno = (Aluno) o;
  return(this.nome.equals(umAluno.getNome()) && this.nota == umAluno.getNota()
        && this.numero.equals(umAluno.getNumero())
        && this.curso.equals(umAluno.getCurso()));
}
```

- como é que será o método equals da classe Turma?

- quais as consequências de não ter o método equals implementado??
  - consideremos que Aluno "não tem" equals (tem apenas o que é fornecido por omissão)
  - o que acontece neste método de Turma?

```
public boolean existeAluno(Aluno umAluno) {
  boolean resultado = false;

  if (umAluno != null) {

    for(int i=0; i< this.ocupacao && !resultado; i++)
      resultado = this.alunos[i].equals(umAluno);

    return resultado;
  }
  else
    return false;
}
```

O método toString

- a informação deve ser concisa (sem *acuçar de ecran*), mas ilustrativa
- todas as classes devem implementar este método
- caso não seja implementado a resposta será:

getClass().getName() + '@' + Integer.toHexString(hashCode())

O método toString

- implementação *normal* de toString na classe Aluno

```
/**
 * Implementação do método toString
 * comum na maioria das classes Java
 *
 * @return     uma string com a informação textual do objecto aluno
 */
public String toString() {

    return("Numero:" + this.numero + "Nome:"+ this.nome + "Nota:" + this.nota);

}
```

- o operador "+" é a concatenação de Strings, sempre que o resultado seja uma String

- Strings são objectos imutáveis, logo não crescem, o que as torna muito ineficientes
  - Para tornar a construção de Strings mais simples (e legível) pode recorrer-se à utilização da classe StringBuilder

```
/**
 * Implementação do método toString
 * comum na maioria das classes Java
 *
 * @return     uma string com a informação textual do objecto aluno
 */
public String toString() {
    StringBuilder sb= new StringBuilder();

    sb.append("Numero: ");
    sb.append(this.numero+"\n");
    sb.append("Nome: ");
    sb.append(this.nome+"\n");
    sb.append("Nota: ");
    sb.append(this.nota+"\n");

    return sb.toString();
}
```

...completar a classe Turma

- equals

```java
/**
 * Método equals standard do Java.
 * Utiliza o método privado getAlunos para efectuar a comparação entre
 * duas instâncias de turma.
 */

public boolean equals(Object umaTurma) {
  if (this == umaTurma)
    return true;

  if((umaTurma == null) || (this.getClass() != umaTurma.getClass()))
    return false;
  else {
    Turma turma = (Turma) umaTurma;
    return (this.designacao.equals(turma.getDesignacao())
           && this.capacidade == turma.getCapacidade()
           && this.ocupacao == turma.getOcupacao()
           && Arrays.equals(this.alunos,turma.getAlunos()));

  }
}
```

- nesta versão recorreu-se ao método equals da classe Arrays
- é necessário garantir que a remoção de alunos não deixa "lixo" no *array* alunos

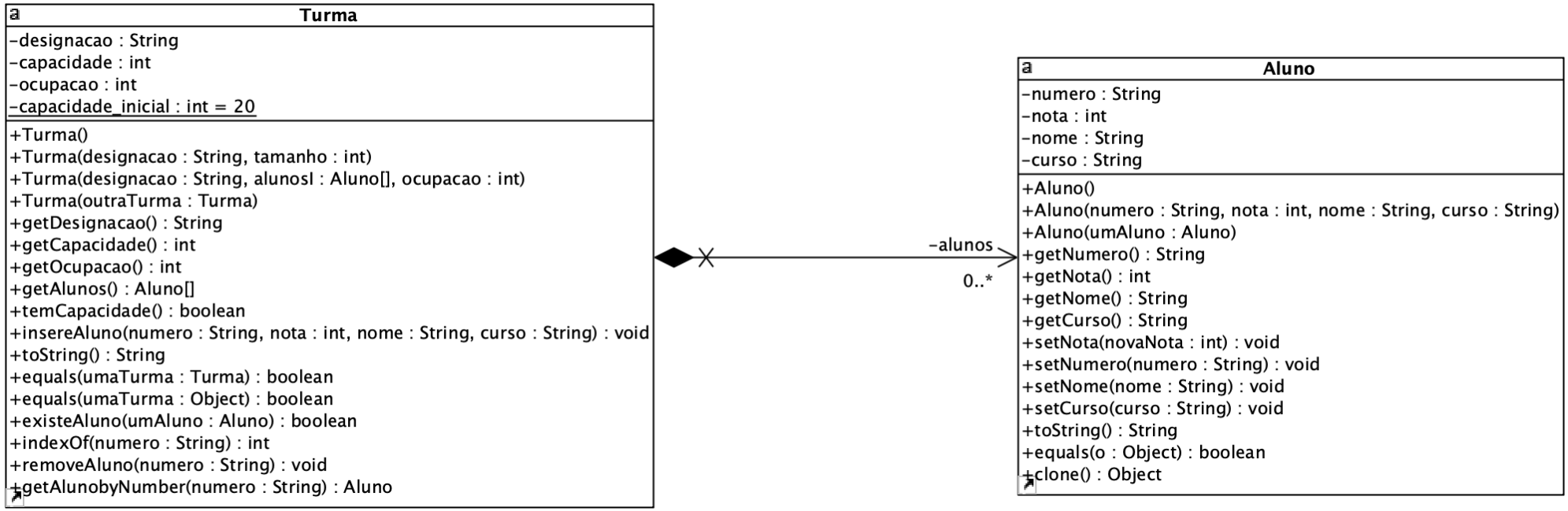
- toString

```
/**
 * Método toString por questões de compatibilização com as restantes
 * classes do Java.
 *
 * Como o toString é estrutural e a classe Aluno tem esse método
 * implementado o resultado é o esperado.
 *
 */
public String toString() {
  StringBuffer sb = new StringBuffer();

  sb.append("Designação: "); sb.append(this.designacao+"\n");
  sb.append("Capacidade: "); sb.append(this.capacidade+"\n");
  sb.append("Alunos: "+"\n"); sb.append(this.alunos.toString());

  return sb.toString();
}
```

- clone

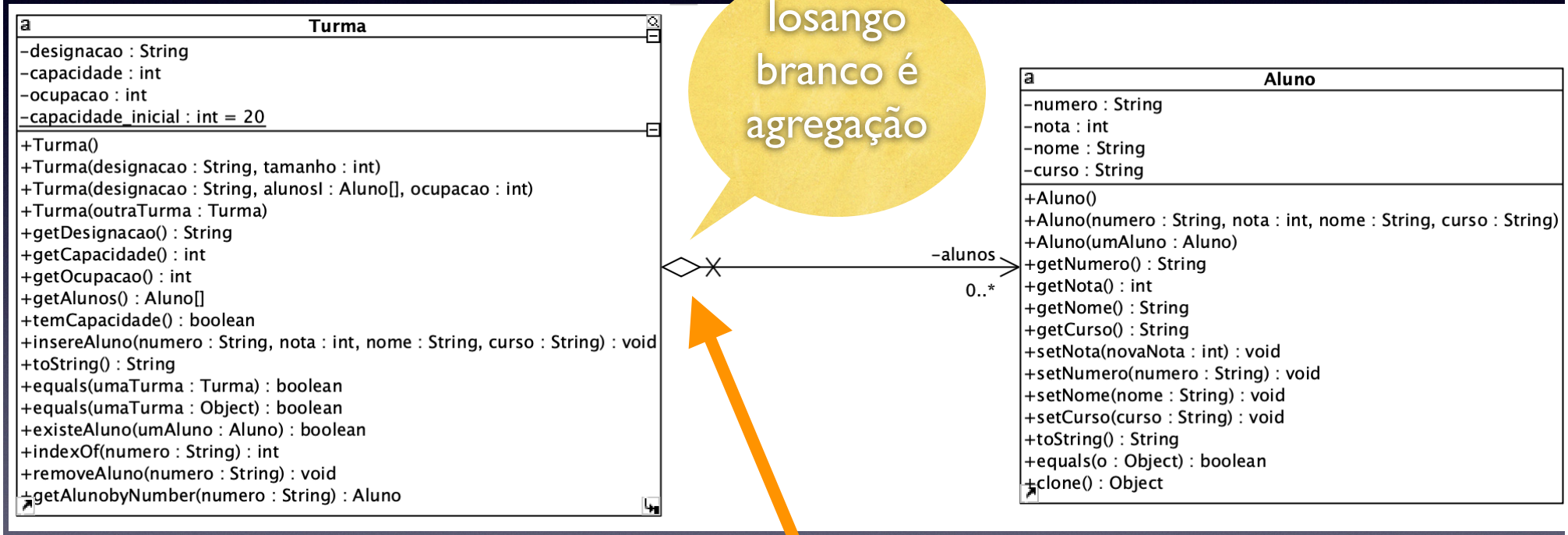
```
public Turma clone() {
  return new Turma(this);
}
```

A arquitectura com composição



A arquitectura com agregação

losango branco é agregação



neste caso as instâncias de Aluno já não são internas à Turma - partilha-se o apontador

… em resumo

- Se o diagrama de classes indicar uma associação de **composição**:
  - faz-se uma cópia (clone) dos objectos quando são guardados internamente
  - devolve-se sempre uma cópia dos objectos e, caso seja necessário, da estrutura de dados que os guarda

… em resumo

- Se o diagrama de classes indicar uma associação de **agregação**:
  - guarda-se internamente o apontador dos objectos passados como parâmetro
  - devolve-se sempre o apontador dos objectos e, caso seja solicitado, uma cópia da estrutura de dados que os guarda

… em resumo

- Quando o diagrama de classes não explicitar se a associação é de composição ou de agregação, parte-se do princípio que é de **composição**!
- O mesmo se aplica quando não se fornece o diagrama de classes.

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.metodoClasse()
- 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:

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

  b) 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!!

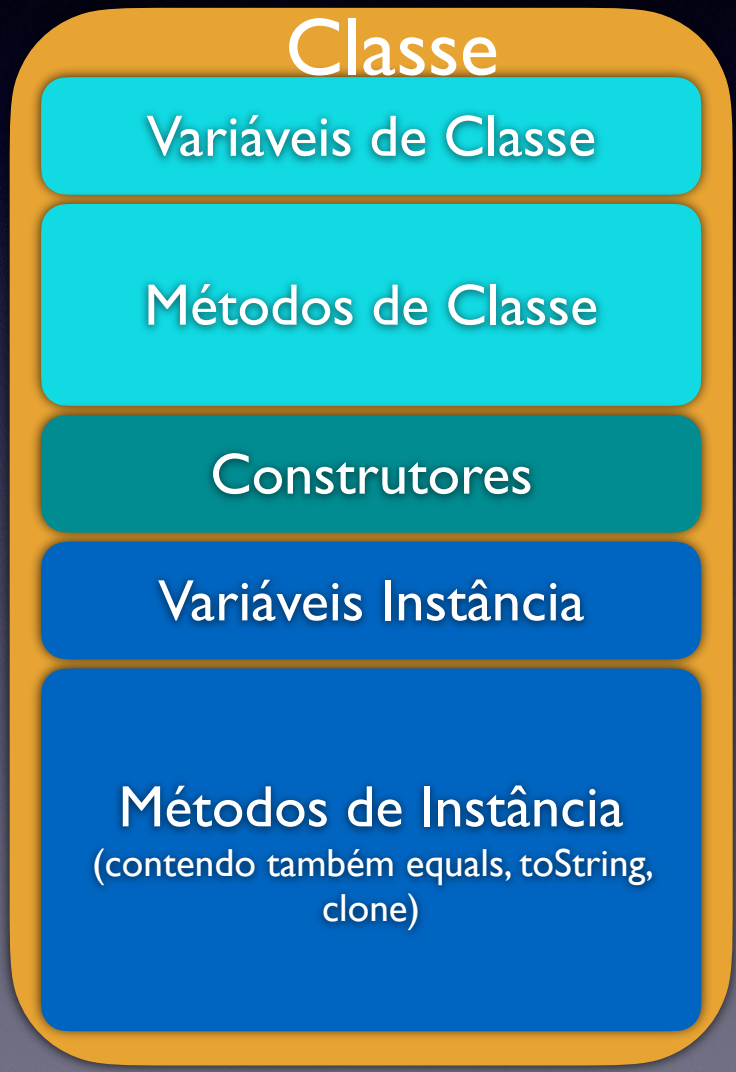
- as variáveis de classe servem para guardar *informação global* a todas as instâncias
- podem também ser utilizadas para guardar constantes que são utilizadas pelos diversos objectos instância
  - como já vimos, por em exemplo com Math.PI (e outros)

- 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

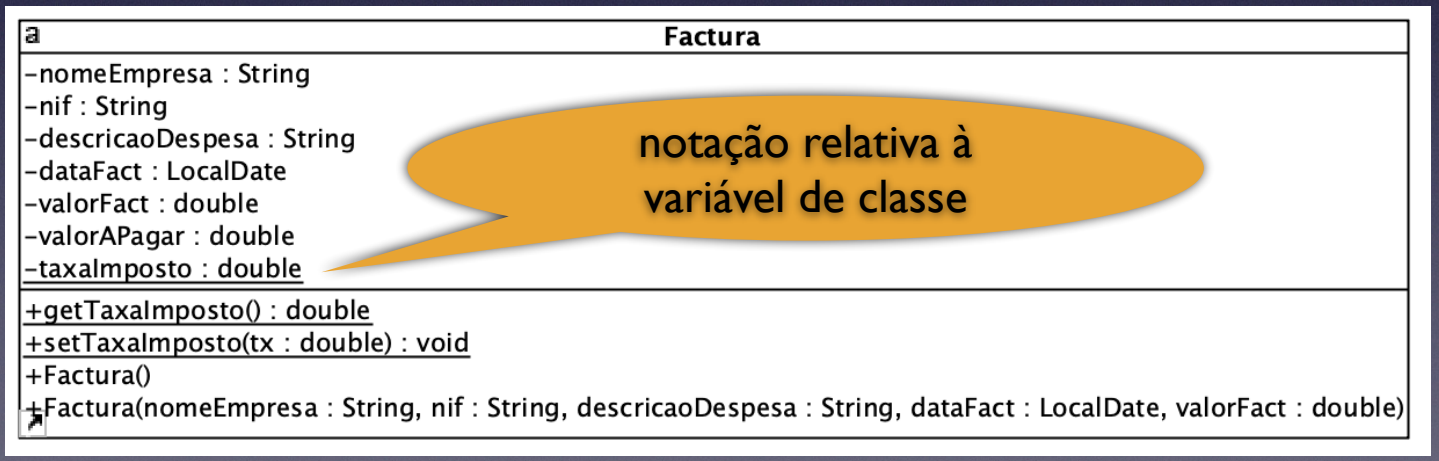


Classe Factura (exemplo)

- seja uma classe Factura, que modela de forma muito simples uma factura
  - 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

Classe Factura

- Representação em Diagrama de Classe



- 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