Pacotes de classes em Java

Programação Orientada a Objetos — QXD0007

UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2021

Pacotes

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Java fornece um mecanismo de agrupamento de classes em pacotes (em inglês, packages), com o qual podemos criar grupos de classes que mantêm uma relação entre si.

Pacotes

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Java fornece um mecanismo de agrupamento de classes em pacotes (em inglês, packages), com o qual podemos criar grupos de classes que mantêm uma relação entre si.

> Um **pacote** é uma coleção de classes relacionadas que provê acesso protegido e gerenciamento de espaço de nomes.

- Conjuntos de classes relacionadas são organizadas em pacotes para:
  - facilitar a localização e uso de tipos
  - evitar conflitos de nomes;
  - fazer controle de acesso.

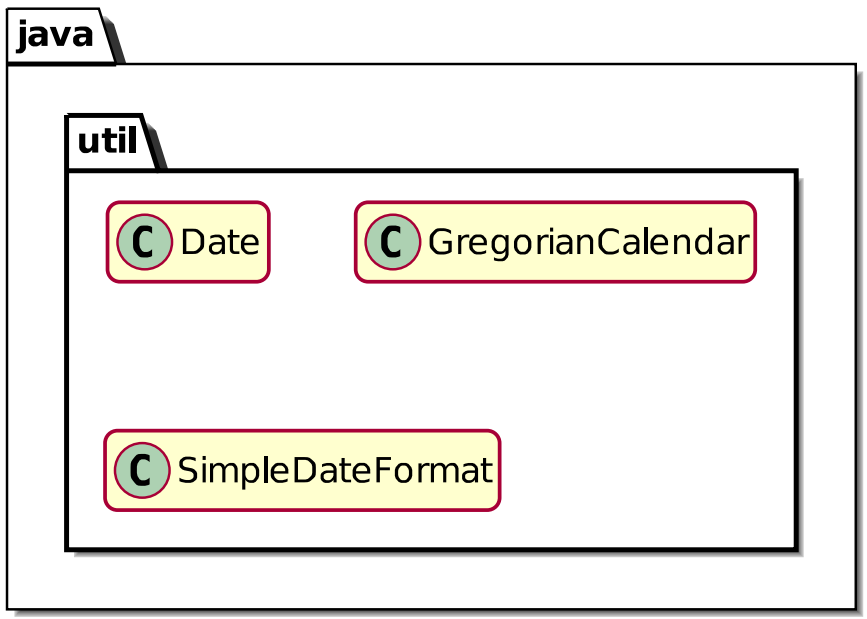
Exemplo

- Os diretórios estão diretamente relacionados aos pacotes e costumam agrupar classes de funcionalidades similares ou relacionadas.

Exemplo

- Os diretórios estão diretamente relacionados aos pacotes e costumam agrupar classes de funcionalidades similares ou relacionadas.

- Por exemplo, no pacote `java.util` temos as classes `Date`, `SimpleDateFormat` e `GregorianCalendar`.
  Todas elas trabalham com datas de formas diferentes.



Criando pacotes de classes

- A maneira mais simples de criar um pacote de classes é criar um diretório e colocar lá todos os códigos-fonte das classes que serão consideradas pertencentes àquele pacote.

Criando pacotes de classes

- A maneira mais simples de criar um pacote de classes é criar um diretório e colocar lá todos os códigos-fonte das classes que serão consideradas pertencentes àquele pacote.

- **Importante:** Cada classe pertencente a um pacote deve ter, no início do arquivo, antes de qualquer outra declaração, a palavra-chave `package` seguida do nome do pacote (caminho de diretórios) ao qual esta classe deverá pertencer.

Criando pacotes de classes

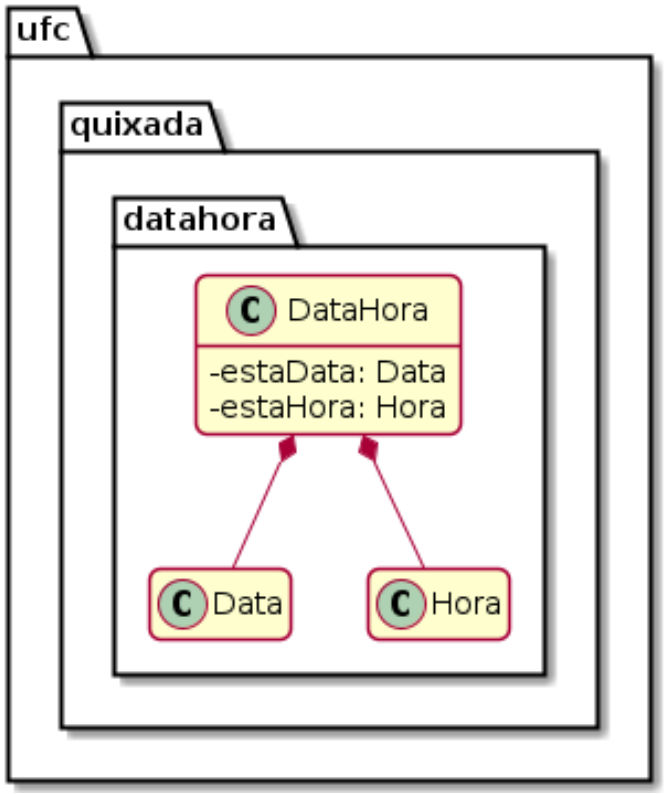
- A maneira mais simples de criar um pacote de classes é criar um diretório e colocar lá todos os códigos-fonte das classes que serão consideradas pertencentes àquele pacote.

- **Importante:** Cada classe pertencente a um pacote deve ter, no início do arquivo, antes de qualquer outra declaração, a palavra-chave `package` seguida do nome do pacote (caminho de diretórios) ao qual esta classe deverá pertencer.

- Todas as classes criadas sem declaração de pertinência em pacotes pertencem ao chamado pacote *default*.
  - As classes criadas assim pertencem ao mesmo pacote e não são necessárias declarações adicionais quando usamos instâncias de uma classe dentro da outra.
  - O compilador e a máquina virtual se encarregarão de chamar os métodos dessas classes.

Criando pacotes de classes – Exemplo

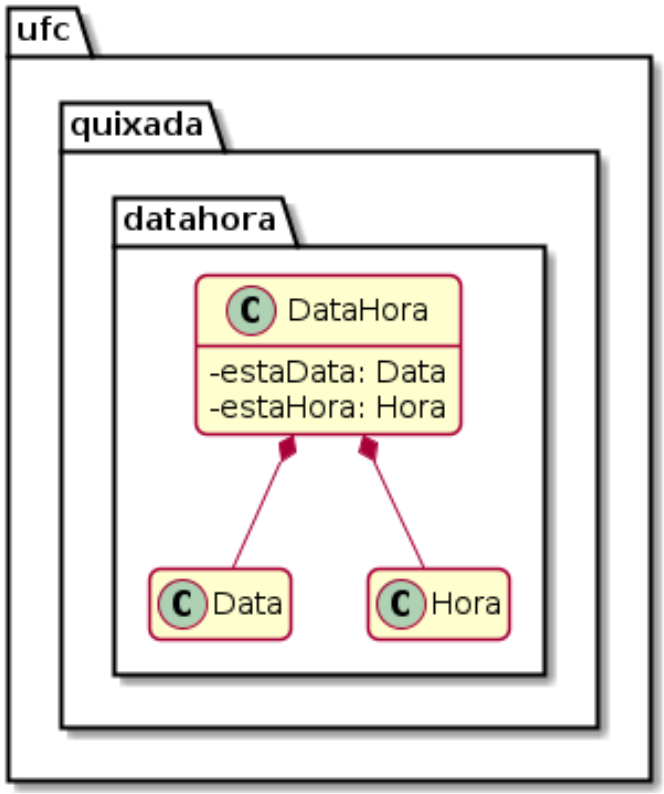
- Considere as classes `Data`, `Hora` e `DataHora`, que encapsula uma data e uma hora através do mecanismo de composição.
  - Vamos criar o pacote `ufc.quixada.datahora` com as três classes acima armazenadas em um diretório `ufc/quixada/datahora`.



Criando pacotes de classes – Exemplo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Considere as classes `Data`, `Hora` e `DataHora`, que encapsula uma data e uma hora através do mecanismo de composição.
  - Vamos criar o pacote `ufc.quixada.datahora` com as três classes acima armazenadas em um diretório `ufc/quixada/datahora`.



- Analisar o projeto DataHora

Padrão da nomenclatura dos pacotes

- As empresas usam seu nome de domínio na Internet de forma reversa para nomear seus pacotes, por exemplo: `br.com.nomedaempresa.nomedoprojeto.subpacote` para um pacote denominado `subpacote` criado por um programador em `nomedoprojeto.nomedaempresa.com.br`

- As colisões de nomes que ocorrem dentro de uma única empresa precisam ser tratadas por convenção dentro dessa empresa, talvez incluindo a região ou o nome do projeto após o nome da empresa.

- Os pacotes só têm letras minúsculas, não importa quantas palavras estejam contidas neles.

Imports

Importação de Classe

- Uma classe pode usar todas as classes do seu pacote e todas as classes públicas de outros pacotes.

Importação de Classe

- Uma classe pode usar todas as classes do seu pacote e todas as classes públicas de outros pacotes.

- Podemos acessar uma classe pública em outro pacote de duas formas:
  - escrevendo o nome completo do pacote antes do nome da classe:

```
java.util.Scanner input;
```

Importação de Classe

- Uma classe pode usar todas as classes do seu pacote e todas as classes públicas de outros pacotes.

- Podemos acessar uma classe pública em outro pacote de duas formas:
  - escrevendo o nome completo do pacote antes do nome da classe:

    ```
    java.util.Scanner input;
    ```

  - Usando a palavra-chave `import`. A declaração `import` permite que façamos referência à classe usando apenas o nome dela:

    ```
    import java.util.Scanner;
    Scanner input;
    ```

Importação de Classe

- Uma classe pode usar todas as classes do seu pacote e todas as classes públicas de outros pacotes.

- Podemos acessar uma classe pública em outro pacote de duas formas:
  - escrevendo o nome completo do pacote antes do nome da classe:

    ```
    java.util.Scanner input;
    ```

  - Usando a palavra-chave `import`. A declaração `import` permite que façamos referência à classe usando apenas o nome dela:

    ```
    import java.util.Scanner;
    Scanner input;
    ```

- Podemos importar todos as classes de um pacote (exceto os subpacotes) usando o símbolo $*$:

  Exemplo: `import java.util.*;`

Observações

A sintaxe `import java.util.*` não tem efeito negativo no tamanho do código.

Porém, importar as classes uma por uma é considerada boa prática de programação em Java, pois facilita a leitura para outros programadores.

Observações

A sintaxe `import java.util.*` não tem efeito negativo no tamanho do código.

Porém, importar as classes uma por uma é considerada boa prática de programação em Java, pois facilita a leitura para outros programadores.

- É muito importante manter a ordem:
  - primeiro aparece uma (ou nenhuma) vez o `package`
  - depois aparecem os `import`s
  - e, por último, as declarações de classe

Cuidados com nomes conflitantes

- Ambos os pacotes `java.util` e `java.sql` possuem uma classe `Date`.
- Se você escrever um programa que importe ambos os pacotes e use a classe `Date`, você obterá um erro de compilação:

```
import java.util.*;
import java.sql.*;
...
Date today; // Erro de compilação
```

Cuidados com nomes conflitantes

- Ambos os pacotes `java.util` e `java.sql` possuem uma classe `Date`.
- Se você escrever um programa que importe ambos os pacotes e use a classe `Date`, você obterá um erro de compilação:

```
import java.util.*;
import java.sql.*;
...
Date today; // Erro de compilação
```

- Isso pode ser resolvido adicionando um import específico:

```
import java.util.*;
import java.sql.*;
import java.util.Date;
```

Cuidados com nomes conflitantes

- Se você realmente quiser usar as duas classes, então terá que usar o nome completo do pacote juntamente ao nome da classe.

```
java.util.Date deadline = new java.util.Date();
java.sql.Date.today = new java.sql.Date(...);
```

Cuidados com nomes conflitantes

- Se você realmente quiser usar as duas classes, então terá que usar o nome completo do pacote juntamente ao nome da classe.

```
java.util.Date deadline = new java.util.Date();
java.sql.Date.today = new java.sql.Date(...);
```

- **Obs.:** Os bytecodes nos arquivos .class usam os nomes completos dos pacotes para se referirem a outras classes.

Cuidados com nomes conflitantes

- Se você realmente quiser usar as duas classes, então terá que usar o nome completo do pacote juntamente ao nome da classe.

```
java.util.Date deadline = new java.util.Date();
java.sql.Date.today = new java.sql.Date(...);
```

- **Obs.:** Os bytecodes nos arquivos .class usam os nomes completos dos pacotes para se referirem a outras classes.

- **Observação** O `import` do Java não tem nada em comum com o `#include` do C++

Cuidados com nomes conflitantes

- Se você realmente quiser usar as duas classes, então terá que usar o nome completo do pacote juntamente ao nome da classe.

```
java.util.Date deadline = new java.util.Date();
java.sql.Date.today = new java.sql.Date(...);
```

- **Obs.:** Os bytecodes nos arquivos .class usam os nomes completos dos pacotes para se referirem a outras classes.

- **Observação** O `import` do Java não tem nada em comum com o `#include` do C++
  - Em C++, um análogo ao mecanismo de pacotes do Java é o `namespace`.
  - Assim, `package` e `import` no Java teriam como análogos no C++ o `namespace` e a diretiva `using`, respectivamente.

Organização do código e das classes

- Se houverem múltiplas classes em um mesmo arquivo, somente uma delas pode ser `public`, e deve ter o mesmo nome do arquivo-fonte.

- Uma boa prática consiste em colocar uma classe por arquivo.
  - Uma vez que outros programadores irão utilizar essa classe, quando precisarem olhar o código da mesma, fica mais fácil encontrá-la sabendo que ela está no arquivo de mesmo nome.

- Somente os membros `public` de um pacote são visíveis fora do pacote, isso inclui as classes, seus construtores, atributos e métodos.

Static imports

Static imports

- Também é possível importar métodos e atributos estáticos (`static`)
- Conhecemos a classe `Math`, do pacote `java.lang`, que possui vários métodos estáticos como, por exemplo:
  - Math.sqrt(), Math.sin(), Math.cos(), Math.abs(), Math.pow(), etc.

Static imports

- Também é possível importar métodos e atributos estáticos (`static`)
- Conhecemos a classe `Math`, do pacote `java.lang`, que possui vários métodos estáticos como, por exemplo:
  - Math.sqrt(), Math.sin(), Math.cos(), Math.abs(), Math.pow(), etc.
- É possível usar os métodos estáticos da classe `Math` sem ter de colocar o nome da classe antes do nome do método. Para isso, basta colocar um `static import` no início do arquivo que usa esses métodos:

```
import static java.lang.Math.*;
```

Static import — Exemplo


```
1  import static java.lang.Math.*;
2
3  public class StaticImport {
4      public static void main(String[] args) {
5          System.out.println("sqrt(900.0) = " + sqrt(900.0));
6          System.out.println("ceil(-9.8) = " + ceil(-9.8));
7          System.out.println("PI = " + PI);
8          System.out.println("E = " + E);
9      }
10 }
```

Pacotes e modificadores de acesso

O pacote `default`

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Se nenhum nome de pacote for utilizado, seus tipos serão membros de um pacote default, que é um pacote sem nome.

- Caso as classes sejam declaradas sem serem pertencentes a pacotes, serão consideradas parte do pacote default, e campos e métodos declarados sem modificadores serão públicos para todas as outras classes do pacote default.

- Esta prática só faz sentido em aplicações muito pequenas, de caráter temporário, ou em uma fase inicial da programação.

Modificadores de acesso e pacotes

	Atributos, construtores e métodos com visibilidade:			
Classes que têm acesso	**private**	**protected**	**default**	**public**
A mesma classe	sim	sim	sim	sim
Classes no mesmo pacote	não	não*	sim	sim
Classes em outro pacote	não	não*	não	sim

* Quando virmos herança, vamos atualizar essas células em cor laranja

Modificadores de acesso e pacotes

	Atributos, construtores e métodos com visibilidade:			
Classes que têm acesso	**private**	**protected**	**default**	**public**
A mesma classe	sim	sim	sim	sim
Classes no mesmo pacote	não	não*	sim	sim
Classes em outro pacote	não	não*	não	sim

* Quando virmos herança, vamos atualizar essas células em cor laranja

- Em Java, classes NÃO podem ser declaradas com os modificadores `private` ou `protected`.

Modificadores de acesso e pacotes

Construtores

- Em Java, construtores podem ser `protected`.
  - Tornar um construtor protegido previne que usuários possam criar uma instância da classe fora do pacote (o mesmo efeito do package access)
  - Um construtor protegido só pode ser acessado dentro do pacote em que ele foi definido ou fora do pacote por uma classe herdeira.
  - Analisar o código do Projeto Animal

Modificadores de acesso e pacotes

Construtores

- Em Java, construtores podem ser `protected`.
  - Tornar um construtor protegido previne que usuários possam criar uma instância da classe fora do pacote (o mesmo efeito do package access)
  - Um construtor protegido só pode ser acessado dentro do pacote em que ele foi definido ou fora do pacote por uma classe herdeira.
  - Analisar o código do Projeto Animal

- Em Java, construtores podem ser `private`.
  - Definir todos os construtores de uma classe como privado, impossibilita instanciar objetos da classe. Isso pode ser útil se você tiver uma classe que apenas tem métodos e atributos `static` (uma biblioteca).

Modificadores de acesso e pacotes

Construtores

- Em Java, construtores podem ser `protected`.
  - Tornar um construtor protegido previne que usuários possam criar uma instância da classe fora do pacote (o mesmo efeito do package access)
  - Um construtor protegido só pode ser acessado dentro do pacote em que ele foi definido ou fora do pacote por uma classe herdeira.
  - Analisar o código do Projeto Animal

- Em Java, construtores podem ser `private`.
  - Definir todos os construtores de uma classe como privado, impossibilita instanciar objetos da classe. Isso pode ser útil se você tiver uma classe que apenas tem métodos e atributos `static` (uma biblioteca).
  - Construtores privados podem ser utilizados também para limitar o número de objetos instanciados. Exemplo: Ver os arquivos ClasseLimitada.java e ClasseLimitadaTeste

Introdução à documentação de classes em Java

Documentação de classes

- Java possui um mecanismo de geração de documentação que cria documentos em HTML a partir de comentários escritos no próprio código.
  - Isso facilita a tarefa do programador — em um mesmo documento ele escreve o que o código faz e quais informações pertinentes devem ser incluídas na documentação.
- Tudo o que for escrito entre os conjuntos de caracteres `/**` e `*/` é considerado comentário. Se o conteúdo entre `/**` e `*/` seguir certas regras, poderemos documentar classes e métodos usando os próprios comentários.

Documentação da classe

- Para documentar uma classe, devemos criar um único comentário imediatamente antes da declaração da classe.

- Exemplo:

```
1 package ufc.quixada.datahora;
2
3 /**
4  * A classe data encapsula os dados de uma data qualquer e
5  * faz parte do pacote ufc.quixada.datahora
6  * @author Atilio Gomes
7  * @version 1.0
8  */
9 public class Data {
```

Marcadores

- Comentários de documentação de classe podem conter marcadores (*tags*) especiais, que devem estar em linhas separadas. Exemplos:
  - `@author`: pode ser seguido de um nome de autor. Podemos ter múltiplos marcadores `author`, se houver vários autores por classe.

Marcadores

- Comentários de documentação de classe podem conter marcadores (*tags*) especiais, que devem estar em linhas separadas. Exemplos:
  - `@author`: pode ser seguido de um nome de autor. Podemos ter múltiplos marcadores `author`, se houver vários autores por classe.
  - `@version`: pode ser seguido de um identificador de versão.

Marcadores

- Comentários de documentação de classe podem conter marcadores (*tags*) especiais, que devem estar em linhas separadas. Exemplos:

  - `@author`: pode ser seguido de um nome de autor. Podemos ter múltiplos marcadores `author`, se houver vários autores por classe.

  - `@version`: pode ser seguido de um identificador de versão.

  - `@see`: pode ser usado em qualquer lugar para referenciar uma outra classe ou um método de outra classe que podem ser de interesse para o programador que utiliza essa classe. Esse marcador geralmente assume uma das duas forma a seguir:
    - `@see nome-da-classe`
    - `@see nome-da-classe#nome-do-método`

    Os comentários de documentação podem conter múltiplos tags `@see`.

Documentação de atributos

- Comentários de atributos devem estar imediatamente antes da declaração dos atributos.

- Por default, somente os atributos protegidos e públicos de uma classe serão incluídos na documentação, e devem ser comentados separadamente.

Documentação de atributos

- Comentários de atributos devem estar imediatamente antes da declaração dos atributos.

- Por default, somente os atributos protegidos e públicos de uma classe serão incluídos na documentação, e devem ser comentados separadamente.

```
1  public class Data {
2      /**
3       * Esse campo encapsula o dia
4       */
5      protected byte dia;
6      /**
7       * Esse campo encapsula o mês
8       */
9      protected byte mes;
10     /**
11      * Esse campo encapsula o ano
12      */
13     protected short ano;
```

Documentação de métodos e construtores

- Comentários de métodos e construtores também devem ser escritos imediatamente antes da declaração dos métodos e construtores.

Documentação de métodos e construtores

- Comentários de métodos e construtores também devem ser escritos imediatamente antes da declaração dos métodos e construtores.

- Comentários de métodos e construtores também podem conter alguns marcadores especiais:
  - `@param`: deve ser seguido de um nome de parâmetro e de uma descrição simples deste parâmetro. Devemos ter um marcador destes para cada argumento passado para o método, e cada marcador deve estar em uma linha do comentário de documentação.

    O marcador `@param` pode ser usado apenas com métodos e construtores. Os comentários de documentação podem conter múltiplos tags `@param`.

Documentação de métodos e construtores

- Comentários de métodos e construtores também devem ser escritos imediatamente antes da declaração dos métodos e construtores.

- Comentários de métodos e construtores também podem conter alguns marcadores especiais:
  - `@param`: deve ser seguido de um nome de parâmetro e de uma descrição simples deste parâmetro. Devemos ter um marcador destes para cada argumento passado para o método, e cada marcador deve estar em uma linha do comentário de documentação.

    O marcador `@param` pode ser usado apenas com métodos e construtores. Os comentários de documentação podem conter múltiplos tags `@param`.

  - `@return`: deve ser seguido de uma descrição do que o método retorna, se o método for declarado como retornando algo que não seja `void`.

Documentação de métodos e construtores

- Comentários de métodos e construtores também devem ser escritos imediatamente antes da declaração dos métodos e construtores.

- Comentários de métodos e construtores também podem conter alguns marcadores especiais:
  - `@param`: deve ser seguido de um nome de parâmetro e de uma descrição simples deste parâmetro. Devemos ter um marcador destes para cada argumento passado para o método, e cada marcador deve estar em uma linha do comentário de documentação.

    O marcador `@param` pode ser usado apenas com métodos e construtores. Os comentários de documentação podem conter múltiplos tags `@param`.

  - `@return`: deve ser seguido de uma descrição do que o método retorna, se o método for declarado como retornando algo que não seja `void`.

  - `@throws`: especifica as exceções lançadas pelo método. Deve ser fornecido para cada tipo de exceção lançado pelo método.

Documentação de métodos – Exemplo

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1       /**
2        * O Construtor da classe, que recebe argumentos para
3        * inicializar os atributos da classe. Note que o
4        * construtor é declarado como sendo público, caso
5        * contrário ele não poderá ser chamado de fora do pacote.
6        * @param dia o dia a ser encapsulado por uma instância
7        *  dessa classe
8        * @param mes o mês a ser encapsulado por uma instância
9        *  dessa classe
10       * @param ano o ano a ser encapsulado por uma instância
11       *  dessa classe
12       * @throws IllegalArgumentException no caso de uma data
13       *  inválida
14       */
15      public Data(byte dia, byte mes, short ano) {
```

Outros marcadores

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Tag javadoc	Descrição
@deprecated	Adiciona uma nota **Deprecated**. Essas são notas para os programadores indicando que eles não devem utilizar os recursos especificados da classe. Notas **Deprecated** normalmente aparecem quando uma classe foi aprimorada com novos e melhores recursos e os recursos mais antigos são mantidos para retrocompatibilidade.
{@link}	Permite que programadores insiram um hyperlink explícito em outro documento de HTML.
@since	Adiciona uma nota **Since:** Essas notas são utilizadas para novas versões de uma classe para indicar quando um recurso foi introduzido primeiro. Por exemplo, a documentação da Java API utiliza esse tag para indicar os recursos que foram introduzidos no Java 1.5.
@version	Adiciona uma nota **Version**. Essas notas ajudam a manter o número de versão do software contendo a classe ou método.

javadoc

- Para gerar a documentação HTML, basta acessar, via terminal, o diretório onde estão os arquivos .java e digitar o comando:

```
javadoc -version -author -d doc *java
```

- A documentação será então gerada e os arquivos resultantes serão colocados dentro do subdiretório `doc` do diretório atual. O nome desse diretório pode ser alterado mudando-se a palavra que segue a flag `-d`.

- Podemos forçar o javadoc a gerar documentação também para a parte privada. Basta acrescentar a flag `-private` na linha de comando acima.

- Uma de várias outras possibilidades é abrir o terminal na pasta do projeto e digitar o comando:

```
javadoc -version -author -private -d doc
ufc.quixada.datahora
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM