```java
/*
 Nome do pacote (package) onde esta classe está implementada.
 Nomes de pacote funcionam de modo similar a uma estrutura de diretórios
 de um sistema operacional, permitindo que as classes tenham um nome único.

 No exemplo abaixo, o nome completo da classe Person é:

 br.univali.kob.poo1.p01_simple_class.Person

 Isso permite a coexistência de duas classes com o mesmo nome, desde que em
 pacotes diferentes. Isso é particularmente útil quando você utiliza
 bibliotecas de terceiros. Estas bibliotecas podem ter utilizado um mesmo
 nome que você utilizou para alguma classe sua. Entretanto, o espaço de nome
 (namespace) gerado pelo pacote garante a individualidade. É por isso que,
 como boa prática, as organizações utilizam o início do nome do pacote com a
 URL invertida da organização:

 br.univali.[nome do pacote].[nome do pacote].[...]

 Deste modo, o espaço de nomes pode ser gerenciado localmente sem a
 preocupação com interferências externas.
*/

package br.univali.kob.poo1.p01_simple_class;

/*
 Como LocalDate não está no mesmo pacote da classe Pessoa, você precisa
 utilizar a cláusula import. Tipicamente, IDEs (ambiente integrado de
 desenvolvimento) como NetBeans, IntelliJ ou Eclipse já indicam a
 falta do import e oferecem mecanismos de inclusão automática (você
 precisa confirmar).

 Experimente comentar esta linha depois e veja que a declaração de
 LocalDate ficará marcada. Se você clicar no indicador (na margem do
 editor), haverá uma opção para adicionar a importação do pacote
 correto.
*/

import java.time.LocalDate;

/*
 Implementação de uma classe simples em Java.

 Uma classe é uma abstração. Abstrair é identificar os aspectos
 essenciais de um contexto qualquer, ignorando características menos
 importantes ou acidentais. Abstração é o resultado deste processo.

 A seleção de quais aspectos são essenciais depende do observador e
 do fenômeno observado (problema a ser resolvido).

 A classe representa um molde, a partir do qual objetos são instanciados.
 Ela define dados (atributos) e comportamentos (operações) comuns a todos
 os objetos instanciados a partir dela.

 Note que há uma mudança de paradigma em relação à abordagem estruturada.
 Na estruturada, temos procedimentos e funções externos às estruturas para
 manipulá-las. Na orientação a objetos, os dados e as rotinas que manipulam
 estes dados estão ENCAPSULADOS em uma mesma estrutura (classe). A partir da
 classe, podemos instanciar (criar) quantos objetos precisarmos.

 Por exemplo, vamos considerar que queremos ordenar uma lista.
```

```
62
63  Na abordagem estruturada: sort(list)
64
65  Na abordagem orientada a objetos: list.sort();
66
67  Você notou a diferença? Em OO, se você quer ordenar uma lista, então
68  peça a ela. Ela é que deve saber como fazer isso.
69 */
70
71 /**
72  * @author Marcello Thiry
73  *
74  * Classe base para hierarquia de pessoas.
75  */
76 public class Person {
77
78 /*
79  Você notou que os atributos são PRIVADOS. Na OO, o conceito de
80  ocultamento da informação (information hiding) estabelece que o
81  acesso aos dados de um objeto deve ser realizado através das
82  operações públicas que ele disponibiliza. Portanto, trataremos os
83  atributos SEMPRE como PRIVADOS.
84
85  OO = encapsulamento + information hiding: NUNCA acessaremos o
86  estado (valor atual dos atributos) de um objeto diretamente.
87 */
88
89     /**
90      *
91      * Nome da pessoa.
92      */
93     private String name;
94     /**
95      *
96      * Data de nascimento da pessoa.
97      */
98     private LocalDate dateOfBirth;
99
100 /*
101  Construtores são operações especiais que permitem a criação
102  (instanciação) de um objeto. Na literatura são consideradas
103  operações gerenciadoras (manager), juntamente com os destrutores.
104  Lembre-se que em Java, não implementamos destrutores.
105  A destruição de um objeto que não possui mais uma referência
106  válida (ninguém aponta para ele) é destruído pelo garbage collector.
107  O garbage collector (coletor de lixo) é um programa que roda em
108  background e varre constantemente a memória, identificando
109  e liberando a memória de objetos que não possuem mais referência.
110
111  Nesta classe, o construtor foi implementado apenas para demonstrar
112  como ele é declarado. Em Java, se o construtor explícito não for
113  declarado, um construtor default NomeDaClasse() é definido
114  implicitamente. Note que a sintaxe é diferente das demais
115  operações. Um construtor em Java deve sempre ter o mesmo nome da
116  classe. Além disso, o tipo de retorno não é explicitado. Ele deve
117  ser invocado sempre em combinação com o comando "new".
118
119  Ex: Person someone = new Person();
120
121  Você pode utilizar o construtor para inicializar o estado de um
122  objeto.
```

```
*/

    /**
     *
     * Construtor default da classe Pessoa.
     */
    public Person() {
        // use este espaço para inicializações
    }

/*
 String é uma classe imutável. Ou seja, não é possível alterar o estado
 de um objeto String depois que ele é criado. Sua documentação pode ser
 encontrada em:

 https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

 Se você fizer algo como:

 String s1 = "teste";
 String s2 = s1;
 s2 = "alterei"

 O valor de s1 continuará sendo "teste".

 Setters são operações modificadoras (mutators), uma vez que elas
 modificam o estado do objeto. Operações modificadoras não são apenas
 Setters. Por exemplo, uma operação depositar em uma classe ContaBancaria
 também altera o estado do objeto (neste caso, o saldo).

 Abaixo, você pode observar annotations javadoc (@xxx). O javadoc é um
 programa de documentação que varre o seu código e gera uma documentação
 em formato html. Ele utiliza annotations predefinidas para reconhecer
 informações de modo diferenciado. Annotation é uma forma de metadados
 que descrevem dados sobre o programa, mas sem fazer parte dele.

 Além disso, você deve ter notado comentários anteriores que iniciam
 com "/**". Qualquer comentário iniciado desta forma é tratado pelo
 javadoc. No roteiro disponibilizado, você aprenderá como gerar a
 documentação desta classe.

 Este comentário, por exemplo, não será considerado pelo javadoc. Você
 saberia dizer por que?
*/

     /**
     *
     * @param name nome da pessoa
     */
    public void setName(String name) {
        /*
         Em Java, utilizamos a palavra reservada "this" para referenciar o
         objeto corrente (o objeto pelo qual este método está sendo chamado.
         "this" permite referenciar qualquer membro (atributo ou método) da
         instância.

         No exemplo abaixo, note que o nome do parâmetro é igual ao nome do
         atributo. Sem a utilização do "this", o programa faria o argumento
         recebido receber ele mesmo (name = name).
         Com o uso de "this", o compilador assumirá que você está se referindo
         ao atributo "name".
```

```
184         */
185         this.name = name;
186     }
187 
188 /*
189  Getters são operações de acesso (accessors), uma vez que elas
190  retornam o estado (mesmo que parcial) do objeto.
191 */
192 
193     /**
194      *
195      * @return nome da pessoa
196      */
197     public String getName() {
198         return name;
199     }
200 
201 /*
202  LocalDate é uma nova classe imutável introduzida a partir do Java 8
203  Sua documentação pode ser encontrada em:
204 
205  https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html
206 */
207 
208     /**
209      *
210      * @param dateOfBirth data de nascimento da pessoa
211      */
212     public void setDateOfBirth(LocalDate dateOfBirth) {
213         this.dateOfBirth = dateOfBirth;
214     }
215 
216     /**
217      *
218      * @return data de nascimento da pessoa
219      */
220     public LocalDate getDateOfBirth() {
221         return dateOfBirth;
222     }
223 
224 }
225 
```