```java
/*
 Comentário não javadoc

 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.aula01;

/*
 Comentário não javadoc

 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;

/*
 Comentário não javadoc

 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.

 Na abordagem estruturada: sort(list)

 Na abordagem orientada a objetos: list.sort();
```

```
72
73  Você notou a diferença? Em OO, se você quer ordenar uma lista, então
74  peça a ela. Ela é que deve saber como fazer isso.
75 */
76
77 /**
78  * Classe base para hierarquia de pessoas do sistema acadêmico.
79  *
80  * @author Marcello Thiry
81  */
82 public class Person {
83
84 /*
85  Comentário não javadoc
86
87  Você notou que os atributos são PRIVADOS. Na OO, o conceito de
88  ocultamento da informação (information hiding) estabelece que o
89  acesso aos dados de um objeto deve ser realizado através das
90  operações públicas que ele disponibiliza. Portanto, trataremos os
91  atributos SEMPRE como PRIVADOS.
92
93  OO = encapsulamento + information hiding: NUNCA acessaremos o
94  estado (valor atual dos atributos) de um objeto diretamente.
95 */
96
97     /**
98      * Nome da pessoa.
99      */
100    private String name;
101    /**
102     * Data de nascimento da pessoa.
103     */
104    private LocalDate dateOfBirth;
105
106 /*
107  Comentário não javadoc
108
109  Construtores são operações especiais que permitem a criação
110  (instanciação) de um objeto. Na literatura são consideradas
111  operações gerenciadoras (manager), juntamente com os destrutores.
112
113  Em Java, não implementamos destrutores. A destruição de um objeto
114  que não possui mais uma referência válida (ninguém aponta para ele)
115  é destruído pelo garbage collector.
116
117  O garbage collector (coletor de lixo) é um programa que roda em
118  background e varre constantemente a memória, identificando
119  e liberando a memória de objetos que não possuem mais referência.
120
121  Nesta classe, o construtor foi implementado apenas para demonstrar
122  como ele é declarado. Em Java, se o construtor explícito não for
123  declarado, um construtor default NomeDaClasse() é definido
124  implicitamente.
125
126  A sintaxe é diferente das demais operações. Um construtor em Java
127  deve sempre ter o mesmo nome da classe. Além disso, o tipo de
128  retorno não é explicitado. Ele deve ser invocado sempre em combinação
129  com o comando "new".
130
131  Ex: Person someone = new Person();
132
133  Você pode utilizar o construtor para inicializar o estado de um
134  objeto.
135 */
136
137    /**
138     * Construtor default da classe Person. Pode ser redefinido
139     * pelas subclasses (só precisa ser explicito se você pretende
140     * fazer alguma coisa). Foi colocado aqui apenas para que vocês
141     * conhecerem a sintaxe.
142     */
```

```
143     public Person() {
144         // use este espaço para inicializações
145     }
146 
147 /*
148  Comentário não javadoc
149 
150  String é uma classe imutável. Ou seja, não é possível alterar o estado
151  de um objeto String depois que ele é criado. Sua documentação pode ser
152  encontrada em:
153 
154  https://docs.oracle.com/javase/8/docs/api/java/lang/String.html
155 
156  Se você fizer algo como:
157 
158  String s1 = "teste";
159  String s2 = s1;
160  s2 = "alterei"
161 
162  O valor de s1 continuará sendo "teste".
163 
164  Setters são operações modificadoras (mutators), uma vez que elas
165  modificam o estado do objeto. Operações modificadoras não são apenas
166  Setters. Por exemplo, uma operação depositar em uma classe ContaBancaria
167  também altera o estado do objeto (neste caso, o saldo).
168 
169  Abaixo, você pode observar annotations javadoc (@xxx). O javadoc é um
170  programa de documentação que varre o seu código e gera uma documentação
171  em formato html. Ele utiliza annotations predefinidas para reconhecer
172  informações de modo diferenciado. Annotation é uma forma de metadados
173  que descrevem dados sobre o programa, mas sem fazer parte dele.
174 
175  Além disso, você deve ter notado comentários anteriores que iniciam
176  com "/**". Qualquer comentário iniciado desta forma é tratado pelo
177  javadoc. No roteiro disponibilizado, você aprenderá como gerar a
178  documentação desta classe.
179 
180  Este comentário, por exemplo, não será considerado pelo javadoc. Você
181  saberia dizer por que?
182 */
183 
184      /**
185       * Setter.
186       *
187       * @param name o nome da pessoa
188       */
189     public void setName(String name) {
190         /*
191          Comentário não javadoc
192 
193          Em Java, utilizamos a palavra reservada "this" para referenciar o
194          objeto corrente (o objeto pelo qual este método está sendo chamado.
195          "this" permite referenciar qualquer membro (atributo ou método) da
196          instância.
197 
198          No exemplo abaixo, note que o nome do parâmetro é igual ao nome do
199          atributo. Sem a utilização do "this", o programa faria o argumento
200          recebido receber ele mesmo (name = name).
201          Com o uso de "this", o compilador assumirá que você está se referindo
202          ao atributo "name".
203         */
204         this.name = name;
205     }
206 
207 /*
208  Comentário não javadoc
209 
210  Getters são operações de acesso (accessors), uma vez que elas
211  retornam o estado (mesmo que parcial) do objeto.
212 */
213 
```

```java
    /**
     * Getter.
     *
     * @return o nome da pessoa
     */
    public String getName() {
        return name;
    }

/*
 Comentário não javadoc

 LocalDate é uma nova classe imutável introduzida a partir do Java 8
 Sua documentação pode ser encontrada em:

 https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html
*/

    /**
     * Setter.
     *
     * @param dateOfBirth a data de nascimento da pessoa
     */
    public void setDateOfBirth(LocalDate dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    /**
     * Getter.
     *
     * @return a data de nascimento da pessoa
     */
    public LocalDate getDateOfBirth() {
        return dateOfBirth;
    }

}
```