POO (MiEI/LCC)

2021/2022

Ficha Prática #03

Classes

Conteúdo

1 Síntese

Segundo Grady Booch *?What isn't a class? An object is not a class, although, curiously, [...], a class may be an object.*", pelo que em Java (e nas linguagens por objectos) temos duas definições importantes: os objectos *instância* que são aqueles que são utilizados em tempo de execução, e os objectos *classe*, que são objectos que guardam a definição da estrutura e comportamento dos objectos instância (embora possam também ter comportamento, como iremos ver nas próximas aulas).

Ainda de acordo com Booch, *"An object has state, behavior, and identity; the structure and behavior of similar objects are defined in their common class; the terms instance and object are interchangeable."*, o que significa que as classes são os objectos onde se descreve a estrutura e o comportamento das instâncias que a partir dela são criadas.

As instâncias de uma classe são criadas através da invocação de uns métodos especiais, os construtores, que são métodos facilmente identificáveis por terem o mesmo nome da classe. Estes métodos não são métodos de instância, mas são métodos da classe.

Exemplos de criação de objectos;

```
1 Ponto p1 = new Ponto(); Ponto p2 = new Ponto(5, -1); Ponto p3
    = new Ponto(10,0);
2 Turma t = new Turma("POO T"); Aluno al1 = new Aluno("5555",
    "José Dias", "MiEi", 14.3);
3 Lampada l = new Lampada("10kW/h");
```

Para descrevermos uma classe, e de acordo com o que foi apresentado nas aulas teóricas, teremos de prever os seguintes items:

- declaração das variáveis de instância (com o indicador de visibilidade `private`)
- codificação dos construtores
- codificação dos métodos de acesso às variáveis de instância
- codificação dos outros métodos que sejam relevantes para definir o comportamento do objecto que se está a descrever

Note-se que as declarações de métodos anteriormente referidas se forem etiquetadas com o nível de visibilidade `public` permitem a sua invocação externa. Caso tenham o nível de visibilidade `private` serão definições locais, isto é auxiliares, e só poderão ser invocadas a partir de código escrito na própria classe.

2 Um exemplo de uma classe

O código que a seguir se apresenta mostra uma classe, **Ponto**, e apresenta os métodos habituais de construção, acesso e alteração das variáveis de instância,

bem assim como os métodos usuais que se consideram necessários (cf. aulas teóricas).

```
1
2  /**
3   * Classe que implementa um Ponto num plano2D.
4   * As coordenadas do Ponto são inteiras.
5   *
6   * @author  MaterialPOO
7   * @version 20210305
8   */
9  public class Ponto {
10
11   //variáveis de instância
12   private int x;
13   private int y;
14
15   /**
16    * Construtores da classe Ponto.
17    * Declaração dos construtores por omissão (vazio),
18    * parametrizado e de cópia.
19    */
20
21   /**
22    * Construtor por omissão de Ponto.
23    */
24   public Ponto() {
25     this.x = 0;
26     this.y = 0;
27   }
28
29   /**
30    * Construtor parametrizado de Ponto.
31    * Aceita como parâmetros os valores para cada coordenada.
32    */
33   public Ponto(int cx, int cy) {
34     this.x = cx;
35     this.y = cy;
36   }
37
38
39
40   /**
41    * Construtor de cópia de Ponto.
42    * Aceita como parâmetro outro Ponto e utiliza os métodos
43    * de acesso aos valores das variáveis de instância.
44    */
45   public Ponto(Ponto umPonto) {
46     this.x = umPonto.getX();
47     this.y = umPonto.getY();
48   }
49
50   /**
51    * métodos de instância
```

```
52      */
53
54    /**
55     * Devolve o valor da coordenada em x.
56     *
57     * @return valor da coordenada x.
58     */
59    public int getX() {
60      return this.x;
61    }
62
63    /**
64     * Devolve o valor da coordenada em y.
65     *
66     * @return valor da coordenada y.
67     */
68    public int getY() {
69      return this.y;
70    }
71
72    /**
73     * Actualiza o valor da coordenada em x.
74     *
75     * @param novoX novo valor da coordenada em X
76     */
77    public void setX(int novoX) {
78      this.x = novoX;
79    }
80
81    /**
82     * Actualiza o valor da coordenada em y.
83     *
84     * @param novoY novo valor da coordenada em Y
85     */
86    public void setY(int novoY) {
87      this.y = novoY;
88    }
89
90    /**
91     * Método que desloca um ponto somando um delta às
          coordenadas
92     * em x e y.
93     *
94     * @param deltaX valor de deslocamento do x
95     * @param deltaY valor de deslocamento do y
96     */
97    public void deslocamento(int deltaX, int deltaY) {
98      this.x += deltaX;
99      this.y += deltaY;
100   }
101
102   /**
103    * Método que soma as componentes do Ponto passado como
          parâmetro.
```

```
104     * @param umPonto ponto que é somado ao ponto receptor da
            mensagem.
105     */
106    public void somaPonto(Ponto umPonto) {
107      this.x += umPonto.getX();
108      this.y += umPonto.getY();
109    }
110
111    /**
112     * Método que move o Ponto para novas coordenadas.
113     * @param novoX novo valor de x.
114     * @param novoY novo valor de y.
115     */
116    public void movePonto(int cx, int cy) {
117      this.x = cx;  // ou setX(cx)
118      this.y = cy;  // ou this.setY(cy)
119    }
120
121    /**
122     * Método que determina se o ponto está no quadrante
            positivo de x e y
123     * @return booleano que é verdadeiro se x>0 e y>0
124     */
125    public boolean ePositivo() {
126      return (this.x > 0 && this.y > 0);
127    }
128
129    /**
130     * Método que determina a distância de um Ponto a outro.
131     * @param umPonto ponto ao qual se quer determinar a
            distância
132     * @return double com o valor da distância
133     */
134    public double distancia(Ponto umPonto) {
135
136      return Math.sqrt(Math.pow(this.x - umPonto.getX(), 2) +
137                       Math.pow(this.y - umPonto.getY(), 2));
138    }
139
140
141
142    /**
143     * Método que determina se o módulo das duas coordenadas é o
            mesmo.
144     * @return true, se as coordenadas em x e y
145     * forem iguais em valor absoluto.
146     */
147    private boolean xIgualAy() {
148      return (Math.abs(this.x) == Math.abs(this.y));
149    }
150
151
152    /**
153     * Método que implementa a igualdade entre dois pontos.
```

```
154     * Reescrita do método equals que todos os objectos possuem
155     *
156     * @param o objecto que é comparado com o receptor da
            mensagem
157     * @return boolean resultado booleano da comparação do
            parâmetro com o receptor
158     */
159    public boolean equals(Object o) {
160      if (this == o)
161        return true;
162      if ((o == null) || (this.getClass() != o.getClass()))
163        return false;
164      Ponto p = (Ponto) o;
165      return (this.x == p.getX() && this.y == p.getY());
166
167    }
168
169
170    /**
171     * Método que devolve a representação em String do Ponto.
172     * @return String com as coordenadas x e y
173     */
174    public String toString() {
175      return "Cx = " + this.x + " -  Cy = " + this.y;
176    }
177
178
179    /**
180     * Método que faz uma cópia do objecto receptor da mensagem.
181     * Para tal invoca o construtor de cópia.
182     *
183     * @return objecto clone do objecto que recebe a mensagem.
184     */
185
186    public Ponto clone() {
187      return new Ponto(this);
188    }
189  }
```

3 Regras a seguir na escrita de classes

Conforme a abordagem seguida nas aulas teóricas, tenham em atenção que:

- As variáveis de instância devem ser declaradas como `private`. Desta forma garantimos que o estado do objecto fica devidamente encapsulado e que o acesso aos valores destas variáveis é feito por métodos construídos para o efeito (os *getters* e *setters*).
- Devem ser criados três construtores para podermos facilmente proceder às tarefas de criação de objectos. Devem ser fornecidos o

construtor por omissão (o construtor vazio), redefinindo assim o construtor herdado da implementação Java, o construtor parametrizado que nos permite criar um objecto com valores para todas as suas variáveis e o construtor de cópia, para facilmente podermos fazer um clone de um objecto.

- Para cada variável de instância de que queiramos visualizar o seu valor e alterá-lo, devemos ter um método de consulta `getVar()`, e um método de modificação `setVar()`.
- Devem ser sempre incluídos nas definições das classes, a menos que não se justifique por serem demasiado complexas, os métodos complementares: `equals()`, `toString()` e `clone()`;
- O código fonte da classe deverá ser documentado, usando comentários na forma `/** ... */`, com o objectivo de produzir automaticamente documentação no formato oficial da documentação das APIs Java (através do programa `javadoc` ou da opção Documentation no BlueJ). A título de exemplo consulte-se a forma como se documentou a classe da secção anterior.

4 Exercícios

Desenvolva as classes pedidas, não se esquecendo dos construtores e dos métodos `equals(Object o)`, `toString()` e `clone()`. Note que é necessário, cf. a mensagem da Ficha 1, criar uma classe de teste com um método `main` onde se criam objectos e se enviam mensagens e recolhem os resultados.

1. Um **Circulo** representa-se através de um ponto central, definido pelas suas coordenadas em `x` e `y` e por um `raio`. Crie a classe **Circulo** com a declaração de variáveis de instância, os construtores habituais (por omissão, parametrizado e de cópia) e os seguinte métodos:
   (a) método que devolve o valor da variável `x`, `public double getX()`
   (b) método que devolve o valor da variável `y`, `public double getY()`
   (c) método que devolve o valor da variável `raio`, `public double getRaio()`
   (d) métodos que alteram o valor das variáveis de instância `public void setX(double x)`, `public void setY(double y)` e `public void setRaio(double raio)`
   (e) método que altera o posicionamento do círculo, `public void alteraCentro(double x, double y)`
   (f) método que calcula a área do círculo, `public double calculaArea()`
   (g) método que calcula o perímetro do círculo, `public double calculaPerimetro()`

2. Considere que um **Telemóvel** é uma entidade que tem como características:

   - marca
   - modelo
   - display: resolução em X × Y
   - dimensão do armazenamento para as mensagens de texto (vistas como `Strings`)
   - dimensão do armazenamento total para fotos e aplicações
   - dimensão do armazenamento para as fotos
   - dimensão do armazenamento para as aplicações
   - espaço total ocupado (em bytes)
   - número de fotos guardadas
   - número de aplicações instaladas
   - nome das aplicações instaladas

   Para a classe **Telemóvel**, codifique os métodos usuais de acesso e alteração às variáveis de instância, os construtores habituais e ainda:

   (a) método que valida se um determinado conteúdo pode ser carregado para o telefone, `public boolean existeEspaco(int numeroBytes)`

   (b) método que carrega (isto é, instala) uma aplicação nova, `public void instalaApp(String nome, int tamanho)`

   (c) método que recebe e guarda uma mensagem de texto, `public void recebeMsg(String msg)`

   (d) método que determina o tamanho médio das aplicações, `public double tamMedioApps()`

   (e) método que devolve a maior mensagem de texto recebida, `public String maiorMsg()`

   (f) método que desinstala uma aplicação, `public void removeApp(String nome, int tamanho)`

3. Um vídeo no YouTube pode ser descrito como contendo:

   - o nome do vídeo
   - o conteúdo do vídeo (uma sequência de bytes, que são depois reproduzidos)
   - a data do seu carregamento
   - a resolução em que foi gravado
   - a duração do mesmo, medida em minutos e segundos

- os comentários que os utilizadores da plataforma fazem sobre o vídeo
- os contadores de "likes"e "dislikes"

Codifique os métodos:

(a) métodos usuais de acesso e alteração das variáveis de instância

(b) método que insere um comentário ao vídeo, `public void insereComentario(String comentario)`

(c) método que determina quantos dias passaram deste que o vídeo foi publicado, `public long qtsDiasDepois()`

(d) método que faz um *like*, `public void thumbsUp()`

~~(e) método que devolve o conteúdo do vídeo pronto para ser depois enviado para um qualquer *render*, `public String processa()` (no caso da classe de teste o render será o System.out)~~

4. Uma **Lâmpada** pode estar ligada ou desligada ou em modo *eco*. Em qualquer um desses estados tem um consumo por milissegundo que é característica da lâmpada (quando desligada o consumo é zero!). Pretende-se que codifique uma classe que permita criar objectos do tipo **Lâmpada** e que possa, além dos métodos usuais, as seguintes funcionalidades:

   (a) ligar a lâmpada no modo *consumo máximo*, `public void lampON()`

   (b) desligar a lâmpada, `public void lampOFF()`

   (c) ligar a lâmpada em modo ECO, `public void lampECO()`

   (d) devolver a informação de quanto se gastou desde a criação da lâmpada, `public double totalConsumo()`

   (e) devolver a informação de quanto se gastou desde o último reset de consumo, `public double periodoConsumo()`

5. Um **jogo de futebol** disputa-se entre duas equipas: visitado e visitante. O jogo pode estar em 1 de três estados: *por iniciar*, *a decorrer* ou *terminado*. Enquanto o jogo está *por iniciar*, o único evento válido consiste em dar início ao jogo. Enquanto o jogo está a decorrer podem acontecer golos, quer da equipa visitada, quer da visitante. Pode ainda ser consultado o resultado do jogo. Após o fim do jogo, apenas pode ser consultado o resultado do jogo.

   Codifique os métodos:

   (a) método que dá início ao jogo, `public void startGame()`

   (b) método que termina o jogo, `public void endGame()`

   (c) método que adiciona um golo à equipa visitada, `public void goloVisitado()`

(d) método que adiciona um golo à equipa visitante, `public void goloVisitante()`

(e) método que devolve o resultado actual do jogo, `public String resultadoActual()`

6. Um **Carro** é uma classe que representa a informação que todos conhecemos dos veículos. A informação que se guarda para cada instância desta classe é a seguinte:

   - marca do carro
   - modelo
   - ano de construção
   - consumo por km a uma velocidade de 100 km/h (serve de referência para cálculos lineares do consumo)
   - kms totais realizados
   - média de consumo desde o início
   - kms no último percurso
   - média de consumo no último percurso
   - capacidade de regeneração de energia (abate ao consumo) quando se trava durante 1 km (o cálculo é também linear)

   Codifique os métodos:

   (a) métodos usuais de acesso e alteração das variáveis de instância

   (b) método que liga o carro (não é possível andar se o carro não estiver ligado!), `public void ligaCarro()`

   (c) método que desliga o carro, `public void desligaCarro()`

   (d) método que força explicitamente um reset do contador de último percurso (se não se fizer nada este contador tem reset quando se liga o carro), `public void resetUltimaViagem()`

   (e) método que avança X metros a uma velocidade de V *km/h*, `public void avancaCarro(double metros, double velocidade)`

   (f) método que trava o carro durante X metros, `public void travaCarro(double metros)`

7. Uma **Linha de Encomenda** representa uma ordem de encomenda de um único produto. Para representar uma entidade deste tipo guardam-se os atributos:

   - referência produto, que é uma `String` com o código que internamente se atribui ao produto
   - descrição do produto

- preço do produto antes de impostos
- quantidade encomendada
- regime de imposto que se aplica ao produto - é um valor em percentagem (ex: 6%, 13%, 23%, etc.)
- valor (em percentagem) do desconto comercial em relação ao preço antes de impostos

Codifique os métodos:

(a) métodos usuais de acesso e alteração das variáveis de instância

(b) método que determina o valor da venda já considerados os impostos e os descontos , `public double calculaValorLinhaEnc()`

(c) método que determina o valor numérico (em euros) do desconto, `public double calculaValorDesconto()`

8. Considere agora que pretendemos criar uma classe **Encomenda**, que é composta por diversas linhas de encomenda (cf a classe feita anteriormente). Para uma **Encomenda** guardam-se os seguintes atributos:

   - nome do cliente
   - número fiscal do cliente
   - morada do cliente
   - número da encomenda
   - data da encomenda
   - as linhas da encomenda (nesta fase guardadas num *array*)

   Codifique os métodos:

   (a) métodos usuais de acesso e alteração das variáveis de instância

   (b) método que determina o valor total da encomenda, `public double calculaValorTotal()`

   (c) método que determina o valor total dos descontos obtidos nos diversos produtos encomendados, `public double calculaValorDesconto()`

   (d) método que determina o número total de produtos a receber, `public int numeroTotalProdutos()`

   (e) método que determina se um produto vai ser encomendado, `public boolean existeProdutoEncomenda(String refProduto)`

   (f) método que adiciona uma nova linha de encomenda, `public void adicionaLinha(LinhaEncomenda linha)`

   (g) método que remove uma linha de encomenda dado a referência do produto, `public void removeProduto(String codProd)`

9. Um **Triângulo** é definido por três pontos no espaço 2D. Crie a classe correspondente, especificando os métodos:
   (a) métodos de acesso e alteração das variáveis de instância
   (b) método que calcula a área do triângulo, `public double calculaAreaTriangulo()`
   (c) método que calcula o perímetro do triângulo, `public double calculaPerimetroTriangulo()`
   (d) método que calcula a altura do triângulo, definido como sendo a distância no eixo dos `y` entre o ponto com menor coordenada em `y` e o ponto com maior coordenada.