POO (MiEI/LCC)

2015/2016

Ficha Prática #03

Classes

Conteúdo

1 Síntese teórica

Nas linguagens de PPO os objectos são divididos em duas grandes categorias de entidades: as INSTÂNCIAS (objectos computacionais puros) e as CLASSES (objectos fundamentalmente de definição, mas que também podem ser de definição e até de prestação de serviços).

CLASSES são objectos particulares que, entre outras funções, guardam a descrição da **estrutura** (*variáveis de instância*) e do **comportamento** (*métodos*) que são comuns a todas as instâncias a partir de si criadas.

As instâncias de uma classe são criadas usando a palavra reservada `new` e um método especial para criação de instâncias, designado **construtor**, que tem, obrigatoriamente, o mesmo nome da classe, podendo ter ou não parâmetros. Exemplos:

```
1 Triangulo tri1 = new Triangulo();
2 Ponto p = new Ponto(); Ponto p2 = new Ponto(5, -1);
3 Turma t = new Turma(); Aluno al1 = new Aluno();
```

2 Classe simples com métodos complementares usuais

```
1  /**
2  * Pontos descritos como duas coordenadas reais.
3  */
4  import static java.lang.Math.abs;
5  public class Ponto2D {
6
7  // Variáveis de Instância
8  private double x, y;
9
10 // Construtores usuais
11 public Ponto2D(double cx, double cy) { x = cx; y = cy; }
12 public Ponto2D(){ this(0.0, 0.0); } // usa o outro construtor
13 public Ponto2D(Ponto2D p) { x = p.getX(); y = p.getY(); }
14
15 // Métodos de Instância
16 public double getX() { return x; }
17 public double getY() { return y; }
18
19 /** incremento das coordenadas */
20 public void incCoord(double dx, double dy) {
21     x += dx; y += dy;
22 }
23
24 /** decremento das coordenadas */
```

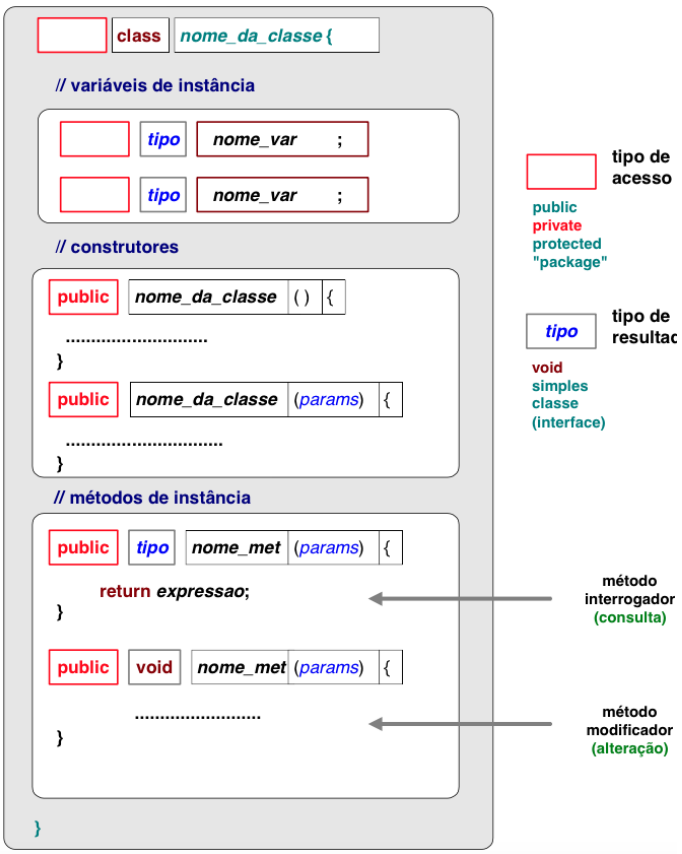

Figura 1: Estrutura de definição de instâncias típica de uma classe.

```
25 public void decCoord(double dx, double dy) {
26     x -= dx; y -= dy;
27 }
28
29 /** soma as coordenadas do ponto parâmetro ao ponto receptor */
30 public void somaPonto(Ponto2D p) {
31     x += p.getX(); y += p.getY();
32 }
33
34 /** soma os valores parâmetro e devolve um novo ponto */
35 public Ponto2D somaPonto(double dx, double dy) {
36     return new Ponto2D(x += dx, y+= dy);
37 }
38
39 /* determina se um ponto é simétrico (dista do eixo dos XX o
       mesmo que do eixo dos YY */
40 public boolean simetrico() {
41     return abs(x) == abs(y);
42 }
43
```

```
44 /** verifica se ambas as coordenadas são positivas */
45 public boolean coordPos() {
46     return x > 0 && y > 0;
47 }
48
49 // Métodos complementares usuais
50 /* verifica se os 2 pontos são iguais */
51 public boolean igual(Ponto2D p) {
52     if (p != null)
53         return (x == p.getX() && y == p.getY());
54     else
55         return false;
56 }
57
58 // outra versão de igual(Ponto2D p)
59 public boolean igual1(Ponto2D p) {
60     return (p == null) ? false : x == p.getX() && y ==
        p.getY();
61 }
62
63 /** Converte para uma representação textual */
64 public String toString() {
65     return new String("Pt2D = " + x + ", " + y);
66 }
67 /** Cria uma cópia do ponto receptor (receptor = this) */
68 public Ponto2D clone() {
69     return new Ponto2D(this);
70 }
71 }
```

3 Regras fundamentais para a criação de classes

- As variáveis de instância devem ser declaradas como `private`, satisfazendo assim o princípio do encapsulamento;
- Pelo menos dois construtores devem ser criados: o que redefine o construtor de JAVA por omissão e o construtor para cópia (recebe um objecto da mesma classe e inicializa as variáveis de instância com os valores deste);
- Por cada variável de instância devemos ter um método de consulta `getVar()` ou equivalente, e um método de modificação `setVar()` ou equivalente;
- 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()`;
- A classe deverá ser bem documentada, em especial os métodos, usando comentários, em particular os comentários `/** ...`

`*/` para que seja produzida automaticamente documentação de projecto através do utilitário `javadoc`.

4 Exercícios

1. Um número Complexo na sua forma rectangular representa-se por $a + bi$, sendo a e b números reais que correpondem aos coeficientes real e imaginário do número, respectivamente. Desenvolva uma classe `Complexo` com métodos de instância para:

   - calcular o conjugado de um complexo, sendo $conjugado(a + bi) = a - bi$. Assinatura:
     ```
     public Complexo conjugado(Complexo complexo);
     ```
   - calcular a soma de dois complexos, dando um novo número complexo como resultado, sabendo que $(a + bi) + (c + di) = (a + c) + (b + d)i$. Assinatura:
     ```
     public Complexo soma(Complexo complexo);
     ```
   - calcular o produto de dois complexos, dando um novo número complexo como resultado, sabendo que $(a+bi)*(c+di) = (ac - bd) + (bc + ad)i$. Assinatura:
     ```
     public Complexo produto(Complexo complexo);
     ```
   - calcular o recíproco de um complexo, sendo $\frac{1}{a+bi} = \frac{a}{a^2+b^2} - \frac{b}{a^2+b^2}i$. Assinatura:
     ```
     public Complexo reciproco(Complexo complexo);
     ```

2. Um pixel é um ponto de coordenadas x e y inteiras mas que tem a si associada uma cor de 0 a 15. Crie uma classe `Pixel` que permita criar "pixels", sabendo-se que, para além das usuais operações de consulta, cada pixel deve responder às seguintes mensagens:

   - Deslocar-se para cima, para baixo, para a esquerda e para a direita um valor real dado. Assinatura:
     ```
     public void desloca(double x, double y);
     ```
   - Mudar a sua cor para uma nova cor (entre 0 e 15). Assinatura:
     ```
     public void mudarCor(int cor);
     ```
   - Sendo o espectro de cores desde o 0 (Preto) a 15 (Branco) segundo a tabela abaixo, escrever um método que devolva uma String correspondente à cor actual do pixel. Assinatura:
     ```
     public String nomeCor();
     ```

0	Preto	4	Castanho	8	Cinza escuro	12	Vermelho
1	Azul marinho	5	Púrpura	9	Azul	13	Fúcsia
2	Verde escuro	6	Verde oliva	10	Verde	14	Amarelo
3	Azul petróleo	7	Cinza claro	11	Azul turquesa	15	Branco

3. Um veículo motorizado é caracterizado pela sua matrícula, quilometragens total e parcial (em Km), capacidade e conteúdo actual do depósito (em litros). Considera-se que quando o conteúdo do depósito baixa dos 10 litros este está na reserva. Sabe-se ainda o consumo médio actual (em litros/100Km – o consumo médio é colocado a zero sempre que a quilometragem parcial é reinicializada). Crie uma classe `Veiculo` que implemente métodos de instância que permitam obter os seguintes resultados:

   - Abastecer L litros de gasolina, ou o máximo possível $< L$, sem transbordar. Assinatura:
     ```
     public void abastecer(int litros);
     ```
   - Reinicializar a contagem parcial da quilometragem, reinicializando também o consumo médio. Assinatura:
     ```
     public void resetKms();
     ```
   - Determinar quantos quilómetros é previsível poder percorrer com o combustível que está no depósito. Assinatura:
     ```
     public double autonomia();
     ```
   - Registar uma viagem de K quilómetros e respectivo consumo, actualizando os dados do veículo. Assinatura:
     ```
     public void registarViagem(int kms, double consumo);
     ```
   - Determinar se o veículo já entrou na reserva. Assinatura:
     ```
     public boolean naReserva();
     ```
   - Dado um valor médio de custo por litro, calcular o valor total gasto em combustível. Assinatura:
     ```
     public double totalCombustivel(double custoLitro);
     ```
   - Dado um valor médio de custo por litro, calcular o custo médio por Km. Assinatura:
     ```
     public double custoMedioKm(double custoLitro);
     ```

4. Um Cartão de Cliente é um cartão de compras que acumula pontos de bonificação à medida que são registadas compras, e que possui o valor total em dinheiro das compras realizadas, um código alfanumérico e um nome de titular.

   Num dado estabelecimento as regras são as seguintes: por cada refeição efectuada, se o valor for inferior a 5 euros, o cliente recebe 1 ponto. Caso o valor seja superior, o cliente

recebe 2 pontos. Ao chegar aos 20 pontos, o cliente recebe uma bonificação de mais 10 pontos. O cliente pode depois descontar esses pontos de duas formas: descontando 10 pontos, adquirir uma sobremesa (menu 1), ou descontando 20 pontos, adquirir uma refeição (menu 2). Escrever uma classe `CartaoCliente` cujas instâncias exibam este comportamento, e permitam ainda:

- Descontar P pontos ao cartão devido a levantamento de um prémio, discriminando qual o menu em causa. Assinatura:
  ```
  public void descontar(int menu);
  ```
- Modificar o titular do cartão. Assinatura:
  ```
  public void setNome(String nome);
  ```
- Modificar o valor de bónus, passando, por exemplo dos 20 para 15 pontos. Assinatura:
  ```
  public void setValorBonus(int valorBonus);
  ```
- Descarregar os pontos de um cartão para o nosso cartão. Assinatura:
  ```
  public void descarregarPontos(CartaoCliente cartao);
  ```
- Inserir no cartão a informação de uma nova compra de certo valor, e actualizar dados. Assinatura:
  ```
  public void efectuarCompra(double valor);
  ```

5. Um Produto de um dado stock de produtos possui as seguintes características de informação: código, nome, quantidade em stock, quantidade mínima, preço de compra e preço de venda a público. Desenvolva uma classe `Produto` e os seguintes métodos de instância:

   - Alterar a quantidade de um produto, ou por saída ou por entrada de uma dada quantidade do produto no stock. Assinatura:
     ```
     public void modificaStock(int valor);
     ```
   - Modificar o código do produto, sabendo que tem de possuir no mínimo 8 caracteres. Assinatura:
     ```
     public void alteraCodigo(String codigo);
     ```
   - Modificar o preço de venda de um dado produto, sendo que o preço de venda nunca poderá ser inferior ao preço de compra. Assinatura:
     ```
     public void setPrecoVenda(double preco);
     ```

- Modificar o preço de compra, dada uma margem de lucro em percentagem. Assinatura:
  ```
  public void defineMargemLucro(double percentagem);
  ```
- Modificar o stock, dado o valor da compra. Assinatura:
  ```
  public void efectuaCompra(double valor);
  ```
- Determinar o lucro actual de tal stock em caso de venda total. Assinatura:
  ```
  public double lucroTotal();
  ```
- Dada uma encomenda de X unidades do produto, determinar o preço total de tal encomenda. Assinatura:
  ```
  public double precoTotal(int encomenda);
  ```
- Verificar se um produto está já abaixo do nível mínimo de stock. Assinatura:
  ```
  public boolean abaixoValor()
  ```

6. Um cronómetro marca as diferenças entre dois tempos registados (início e fim). Um cronómetro "double-split" faz ainda mais: Inicia uma contagem de tempo, faz uma primeira paragem, mas continua a contar o tempo até ser feita a segunda paragem. Ambas as paragens são efectuadas com recurso ao mesmo método, `public void parar()`. Criar uma classe `CronometroDS` que permita calcular:

   - O tempo total em minutos, segundos e milissegundos entre o início e a primeira paragem. Assinatura:
     ```
     public String primeiraParagem()
     ```
   - O tempo total em minutos, segundos e milissegundos entre o início e a segunda paragem. Assinatura:
     ```
     public String segundaParagem()
     ```
   - A diferença de tempos entre a primeira e a segunda paragem de tempo. Assinatura:
     ```
     public String tempoSplit()
     ```
   - Determinar o tempo absoluto em hora-min-seg-miliseg do arranque e de cada paragem. Assinatura:
     ```
     public String tempoAbsoluto()
     ```

   Os resultados dos 3 primeiros métodos devem ser apresentados no formato "*mm*:*ss*'*ms*".

7. Uma Conta Bancária a prazo é criada com um código, um titular, tem uma data de início (dia, mês e ano) de contagem de juros que é actualizada sempre que os juros são calculados e adicionados ao capital, tem um dado montante de capital

depositado, com um prazo para cálculo de juros, e uma taxa de juro de t% para tal prazo, definida aquando da sua criação. Crie uma classe `ContaPrazo` que, para além dos construtores e dos métodos de consulta, permita realizar as seguintes operações:

- Calcular o número de dias passados desde a abertura da conta. Assinatura:
  ```
  public int diasPassados();
  ```
- Alterar o titular da conta ou alterar a taxa de juros. Assinatura:
  ```
  public void alterarTitular(String nome);
  public void alterarTaxa(double taxa);
  ```
- Atingido o prazo para juros, calcular tais juros, juntá-los ao capital, e registar a nova data de cálculo de juros. Assinatura:
  ```
  public void actualizarJuros(GregorianCalendar novaData);
  ```
- Verificar se hoje é o dia de calcular os juros. Assinatura:
  ```
  public boolean verificarDiaJuros();
  ```
- Fechar a conta calculando o valor total a pagar ao titular (capital inicial + juros). Assinatura:
  ```
  public double fecharConta();
  ```