

Programação Centrada em Objetos

2021/2022

Série 3

Construção de classes

Revisões

1. O que é uma instância de uma classe? O que são atributos e métodos de instância?
2. Qual o âmbito de uma variável? E de um parâmetro? E de um atributo `private`?
3. Qual o âmbito de um método `private`? E de um método `public`?
4. O que é um construtor? Como se processa a criação de um objeto?
5. O que é o `this`?
6. O que entende por uma referência `null`? Qual o efeito da invocação de um método sobre tal referência?
7. O que são atributos e métodos de classe (`static`)? Qual a diferença entre um atributo de classe e um atributo de instância?
8. O que é um enumerado? Como posso aceder a todos os valores que define?

Exercícios

9. Considere a classe `Point` que representa pontos do plano, descrita pela seguinte documentação.

Constructor Summary	
<code>Point</code>	<code>(double x, double y)</code> Builds and initializes a point at the specified (x,y) location in the coordinate space.
Method Summary	
boolean	<code>colinear(Point p1, Point p2)</code> Determines whether or not this point and two others are colinear.
<code>Point</code>	<code>copy()</code> Returns a copy of this point.
double	<code>distance(Point p)</code> Returns the distance from this <code>Point</code> to a specified point.
boolean	<code>equalsPoint(Point p)</code> Determines whether or not this point is equal to another point.
<code>String</code>	<code>toString()</code> Returns a string representation of this point and its location in the (x,y) coordinate space.
void	<code>translate(double dx, double dy)</code> Moves this point by dx along the x axis and by dy along the y axis.
double	<code>x()</code> Returns the x (cartesian) coordinate of this point in double precision.
double	<code>y()</code> Returns the y (cartesian) coordinate of this point in double precision.

- a) Programe esta classe de acordo com a documentação.
- b) Escreva uma classe de nome `TestPoint` contendo um método `main` que (i) crie dois pontos, (ii) imprima a representação textual desses dois pontos, (iii) faça a translação do segundo ponto de 5 unidades na horizontal e 4 na vertical, (iv) imprima os dois pontos de novo e (v) imprima a distância entre os dois pontos.

10. Um polígono é uma região do plano fechada, delimitada por segmentos de reta. Defina uma classe `Polygon` cujos objetos representam polígonos e que ofereça as seguintes funcionalidades:

- a) um construtor que crie um polígono a partir dum vetor de pontos com os seus vértices, assumindo que o vetor dado é válido (ver alínea f);
- b) um método com a assinatura `double perimeter()` que retorne o perímetro deste polígono;
- c) um método com a assinatura `Point closestOrigin()` que devolva o vértice deste polígono mais próximo da origem;
- d) um método com a assinatura `void translate(double dX, double dY)` que efetue uma translação deste polígono segundo o vetor definido por (dX, dY) ;
- e) um método com a assinatura `String toString()` que devolva uma representação textual deste polígono.
- f) um método de classe com a assinatura `static boolean valid(Point[] vertex)` que verifique se: 1) o vetor `vertex` tem mais do que 2 vértices; 2) o vetor `vertex` está todo preenchido; 3) o vetor `vertex` não tem 3 pontos consecutivos colineares. Note que estas são propriedades exigidas aos vértices de qualquer polígono, considerando que cada dois pontos consecutivos definem uma aresta, bem como o primeiro e o último (estas propriedades, não sendo suficientes, servem o propósito do exercício).
- g) um método com a assinatura `boolean equalsPolygon(Polygon other)` que verifique se este polígono é igual ao polígono dado (observe que a numeração dos vértices de um polígono não é relevante);
- h) um método com a assinatura `Polygon copy()` que devolva um polígono igual a este polígono;
- i) um método com a assinatura `double longestSide()` que retorne o comprimento da aresta mais comprida deste polígono;
- j) um método com a assinatura `int vertexInQuadrant(int n)` que devolva o número de vértices deste polígono no quadrante n , sabendo que um ponto (x,y) pertence:
 - ao 1º quadrante se $x > 0$ e $y > 0$
 - ao 2º quadrante se $x < 0$ e $y > 0$
 - ao 3º quadrante se $x < 0$ e $y < 0$
 - ao 4º quadrante se $x > 0$ e $y < 0$

11. Escreva uma classe de nome `TestPolygon` contendo um método `main` que:

- a) cria três pontos;
- b) cria um polígono após ter a certeza que os 3 pontos formam um polígono;
- c) imprime a representação textual do polígono;
- d) imprime o perímetro do polígono e qual o vértice mais próximo da origem;
- e) faz a translação do polígono segundo o vetor definido por $(4,5)$ e imprime de novo o polígono.

Assuma que esta classe cliente está na mesma diretoria que as classes `Polygon` e `Point`.

12. Para o desenvolvimento duma aplicação foi identificada a necessidade de ter uma classe cujos objetos representem instantes de tempo. Defina uma classe `Instante` cujos objetos sejam

instantes de tempo definidos por uma hora, um minuto e um segundo (por exemplo, *14 : 37 : 20*). A classe deve oferecer as seguintes funcionalidades:

- a) um método de classe com a assinatura `static boolean legal(int horas, int minutos, int segundos)` que verifique se os argumentos fornecidos definem um instante;
- b) construtores com 0, 1, 2 ou 3 argumentos, correspondendo (por ordem) às horas, minutos e segundos do instante a criar (assuma o valor 0 para argumentos ausentes em cada um dos construtores);
- c) métodos com as assinaturas `void avancaSegundo()`, `void avancaMinuto()` e `void avancaHora()` permitindo avançar este instante respectivamente de um segundo, um minuto ou uma hora (considere que às 23 : 59 : 59 se seguem as 0 : 00 : 00);
- d) um método com a assinatura `void avanca(Instante tempo)` que avança este instante a quantidade de tempo descrita pelo argumento;
- e) um método com a assinatura `boolean equalsInstante(Instante other)` que determina se este instante é igual ao dado;
- f) um método com a assinatura `Instante copy()` que devolva uma cópia deste instante;
- g) um método com a assinatura `String toString()` que devolva uma representação textual deste instante.

Escreva também uma classe cliente para testar esta classe.

13. O jogo do galo joga-se num tabuleiro de 3 x 3. Dois jogadores jogam alternadamente escolhendo uma casa livre e marcando-a com X (primeiro jogador) ou O (segundo jogador). O primeiro jogador que conseguir ocupar as três casas duma linha, coluna ou diagonal vence o jogo.

(1,1)	(1,2)	(1,3)
(2,1)	(2,2)	(2,3)
(3,1)	(3,2)	(3,3)

Considerando que as posições do tabuleiro são identificadas como mostrado acima e tirando partido do tipo enumerado

```
enum SimboloJogo {X, O}
```

desenvolva:

- a) uma classe `Tabuleiro` cujos objetos representem instâncias de tabuleiros do jogo do galo e que ofereça as seguintes funcionalidades:
 - um construtor que crie um tabuleiro no estado inicial (tabuleiro vazio);
 - um método com a assinatura `void efetuarJogada(int linha, int coluna, SimboloJogo s)` que altera o tabuleiro em conformidade com a jogada, assumindo que a posição indicada não está ocupada;
 - um método com a assinatura `boolean ocupada(int i, int j)` que indica se a posição do tabuleiro está ou não ocupada;
 - um método com a assinatura `SimboloJogo ocupante(int i, int j)` que indica o símbolo ocupante da posição do tabuleiro (null se não estiver ocupada);
 - um método com a assinatura `String toString()` que devolva uma representação textual do estado do tabuleiro.
- b) uma classe `Galo` cujos objetos representem instâncias do jogo do galo e que ofereça as seguintes funcionalidades:
 - um construtor que crie um jogo novo;
 - um método com a assinatura `boolean terminado()` que indica se o jogo já está no estado terminado;
 - um método com a assinatura `boolean jogadaValida(int linha, int coluna, SimboloJogo s)` que determina se uma jogada é válida (o jogo não está terminado, a

posição indicada por linha e coluna não está já ocupada e é a vez do jogador que joga com `s` jogar);

- um método com a assinatura `void efetuarJogada(int linha, int coluna, SimboloJogo s)` que, assumindo que a jogada é válida, altera o estado do jogo;

14. Recorrendo à classe `Galo`, escreva um programa que permita que dois jogadores joguem interativamente ao jogo do galo. Não se esqueça que é da responsabilidade dos clientes verificar que todas as invocações dos seus métodos são corretas.
15. Enriqueça o programa anterior de forma a dar também a possibilidade de um jogador jogar contra um jogador automático.
16. O Natal está à porta e os calendários de Natal olham para nós nas lojas e tentam-nos: “*Que chocolates deliciosos eu tenho para tu encontrares!*” À falta de melhor, vamos aqui construir um programa que cria um calendário de Natal virtual e procura as 24 surpresas, dia a dia.

Para isso, vamos criar os seguintes enumerados e classes:

- a) Um enumerado `Chocolate` que representa os vários chocolates que as “caixinhas-surpresa” dos calendários contêm (`BRANCO`, `DE_LEITE`, `PRETO`, `COM_AMENDOAS`, `COM_ARROZ_TUFADO`, `PRALINE`, `DE_GINJA`, `SNICKERS`, `MARS`).
- b) Um enumerado `Likomments` que representa os comentários que nós faríamos ao comer os chocolates, dia após dia (`HMMMM`, `MNHANMNHAN`, `FABULOUS`, `DELICIOUS`, `SCRUMPTIOUS`, `YUMMY`, `NUMMY`).
- c) Uma classe `BlocoDia` cujos objetos representam as “caixinhas-surpresa” do calendário de Natal e oferece as seguintes funcionalidades:
 - um construtor que constrói um bloco, dados o chocolate-surpresa que vai conter e o dia;
 - dois métodos para conhecer o chocolate-surpresa e o dia deste bloco, respetivamente;
 - um método com a assinatura `String toString()` que dá a representação textual deste bloco.
- d) Uma classe `CalendarioDeNatal` cujos objetos representam calendários de Natal e oferece as seguintes funcionalidades:
 - um método de classe com a assinatura `static boolean legal(int[] dias)` que verifique que os inteiros contidos em dias são os números de 1 a 24, sem repetições;
 - um construtor `CalendarioDeNatal (int[] dias)` que constrói um calendário de Natal com 24 “caixinhas-surpresa” cujos dias seguem a ordem dada pelo parâmetro dias; a surpresa de cada uma destas “caixinhas” é definida de forma aleatória;
 - um método com a assinatura `Chocolate encontraSurpresa(int dia)` que, assumindo que o parâmetro dia é um valor entre 1 e 24, devolve o chocolate-surpresa que este calendário de Natal define para esse dia;
 - um método com a assinatura `String toString()` que dá a representação textual deste calendário de Natal.
- e) Uma classe `TestaCalendario` contendo um método `main` que:
 - Constrói um calendário de Natal e imprime-o no ecrã;
 - Para cada dia de 1 a 24, imprime no ecrã o chocolate-surpresa que o calendário define, seguido de um comentário do enumerado `Likomments` escolhido aleatoriamente.

17. Considere que tem ao seu dispor os seguintes tipos de dados, implementados por alguém:

```
public class Microbio
extends java.lang.Object

Classe cujas instancias representam microbios

Author:
in

Constructor Summary

Constructors
Constructor and Description
Microbio(MicrobioTipo tipo, java.lang.String nome)
Inicializa os atributos do novo objeto

Method Summary

All Methods Instance Methods Concrete Methods
Modifier and Type Method and Description
boolean equals(java.lang.Object ob)
Este microbio e' igual a outro?
java.lang.String nome()
O nome deste microbio
MicrobioTipo tipo()
O tipo deste microbio
java.lang.String toString()
Representacao textual deste microbio
```

```
public enum MicrobioTipo
extends java.lang.Enum<MicrobioTipo>

Enumerado cujos elementos representam todos os tipos de microbios existentes

Author:
in

Enum Constant Summary

Enum Constants
Enum Constant and Description
ARCHAEA
BACTERIA
FUNGO
VIRUS

Method Summary

All Methods Static Methods Concrete Methods
Modifier and Type Method and Description
static MicrobioTipo valueOf(java.lang.String name)
Returns the enum constant of this type with the speci
name.
static MicrobioTipo[] values()
Returns an array containing the constants of this enum
type, in the order they are declared.
```

```
public class Par<P,S>
extends java.lang.Object

Representa pares de objetos de tipos quaisquer

Author:
isabel nunes

Constructor Summary

Constructors
Constructor and Description
Par(P prim, S seg)
Inicializa os atributos do novo objeto

Method Summary

All Methods Instance Methods Concrete Methods
Modifier and Type Method and Description
P primeiro()
O primeiro elemento deste par
S segundo()
O segundo elemento deste par
java.lang.String toString()
Representacao textual deste par
```

Construa a classe Pessoa, cujas instâncias representam pessoas que têm um nome e vários micróbios. A classe deve oferecer o construtor:

- public Pessoa(String nome) que inicializa os atributos do novo objeto; o nome do novo objeto ficará igual a nome;

e os seguintes métodos:

- public void adicionaMicrobio(Microbio m) que, assumindo que m não é null, adiciona o micróbio m aos micróbios desta pessoa;
- public boolean temMicrobio(Microbio m) que retorna true se esta pessoa tem o micróbio m;
- public List<Par<String,MicrobioTipo>> todosMicrobios() que retorna uma lista de pares contendo, para cada micróbio que esta pessoa tem, o seu nome e o seu tipo;
- public String toString() que retorna a representação textual desta pessoa (ver formato no exemplo abaixo);
- List<Par<MicrobioTipo,Integer>> quantosPorTipo() que retorna uma lista de pares contendo, para cada tipo de micróbio definido no enumerado MicrobioTipo, o seu nome e o número de micróbios desse tipo que esta pessoa tem.

Sabendo que o método toString() de qualquer implementação de List devolve a representação de todos os seus elementos separados por vírgulas, entre parêntesis retos, o método main da figura ao lado deve produzir o seguinte output:

Nome: LactoBac Tipo: BACTERIA

Nome: Donald

Microbios:

Nome: Gripe Tipo: VIRUS

Nome: LactoRam Tipo: BACTERIA

Nome: Sars Tipo: VIRUS

Nome: Bifido Tipo: BACTERIA

Nome: LactoBac Tipo: BACTERIA

Nome: Bolor Tipo: FUNGO

true

[(Gripe,VIRUS), (LactoRam,BACTERIA), (Sars,VIRUS), (Bifido,BACTERIA), (LactoBac,BACTERIA), (Bolor,FUNGO)]

```
public static void main(String[] args) {

    Microbio[] mArray = new Microbio[6];
    mArray[0] = new Microbio(MicrobioTipo.VIRUS, "Gripe");
    mArray[1] = new Microbio(MicrobioTipo.BACTERIA, "LactoRam");
    mArray[2] = new Microbio(MicrobioTipo.VIRUS, "Sars");
    mArray[3] = new Microbio(MicrobioTipo.BACTERIA, "Bifido");
    mArray[4] = new Microbio(MicrobioTipo.BACTERIA, "LactoBac");
    mArray[5] = new Microbio(MicrobioTipo.FUNGO, "Bolor");

    System.out.println(mArray[4].toString());
    System.out.println();

    Pessoa p = new Pessoa("Donald");
    for(Microbio m : mArray) {
        p.adicionaMicrobio(m);
    }

    System.out.println(p.toString());
    System.out.println(p.temMicrobio(
        new Microbio(MicrobioTipo.VIRUS, "Sars")));
    System.out.println(p.todosMicrobios());

}
```

18. Um campeonato de Fórmula 1 é constituído por vários grandes prémios (Grand Prix) que se realizam ao longo do ano em vários locais do mundo.

Considere que tem ao seu dispor os seguintes tipos de dados, implementados por alguém:

```
public class Driver
extends java.lang.Object

Class whose objects represent drivers
```

Constructor Summary

Constructors

Constructor and Description
Driver (java.lang.String name, Team team) Gives initial values to the attributes of the new object

Method Summary

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
java.lang.String	name () The name of this driver	
Team	team () The team to which this driver belongs	
java.lang.String	toString () The textual representation of this driver	

```
public enum Team
extends java.lang.Enum<Team>

Enumerado que representa todas as equipas de Formula1
```

Enum Constant Summary

Enum Constants

Enum Constant and Description

FERRARI

MC_LAREN

MERCEDES

RED_BULL

WILLIAMS

Method Summary

All Methods

Static Methods

Concrete Methods

Modifier and Type

Method and Description

static Team

valueOf(java.lang.String name)

Returns the enum constant of this type with the specified name.

static Team[]

values()

Returns an array containing the constants of this enum type, in the order they are declared.

```
public class Pair<F,S>
extends java.lang.Object
```

Represents pairs of objects of any types

Constructor Summary

Constructors

Constructor and Description

Pair(F first, S second)

Gives initial values to the attributes of the new object

Method Summary

All Methods

Instance Methods

Concrete Methods

Modifier and Type

Method and Description

F

first()

The first element of this pair

S

second()

The second element of this pair

java.lang.String

toString()

Textual representation of this pair

Construa a classe `GrandPrix`, cujas instâncias representam grandes prémios de Fórmula 1, que se realizam num dado local e têm vários condutores. A classe deve oferecer o construtor:

- `public GrandPrix(String place, List<Driver> drivers)` que inicializa os atributos do novo objeto; os elementos da lista `drivers` estão ordenados de acordo com o *ranking* atual do campeonato;

e os seguintes métodos:

- `public int howManyOfTeam(Team t)` que devolve o número de condutores inscritos no grande prémio que pertencem à equipa `t`;
- `public Team hasMoreDrivers()` que retorna a equipa que tem mais condutores a participar neste grande prémio; considere que existe uma só equipa nessas condições;
- `public Team driverTeam(String name)` que retorna a equipa a que pertence o condutor com o nome `name` e `null` se não existir;
- `List<Pair<Team, Integer>> howManyEachTeam()` que retorna uma lista de pares contendo, para cada equipa, o seu nome e o número de condutores que lhe pertencem.
- `public String toString()` que retorna a representação textual desta pessoa (ver formato no exemplo abaixo).

O método `main` da figura ao lado deve produzir o seguinte *output*:

Name: Hamilton Team: MERCEDES

Place: Portugal

Drivers:

Name: Verstappen Team: RED_BULL

Name: Hamilton Team: MERCEDES

Name: Norris Team: MC_LAREN

Name: Bottas Team: MERCEDES

Name: Leclerc Team: FERRARI

RedBull drivers: 1

Team with more drivers: MERCEDES

Norris' team: MC_LAREN

```
public static void main(String[] args) {
    List<Driver> drivers = new ArrayList<Driver>();
    drivers.add(new Driver("Verstappen", Team.RED_BULL));
    drivers.add(new Driver("Hamilton", Team.MERCEDES));
    drivers.add(new Driver("Norris", Team.MC_LAREN));
    drivers.add(new Driver("Bottas", Team.MERCEDES));
    drivers.add(new Driver("Leclerc", Team.FERRARI));

    System.out.println(drivers.get(1).toString());
    System.out.println();

    GrandPrix gp = new GrandPrix("Portugal", drivers);

    System.out.println(gp.toString());
    System.out.println("RedBull drivers: " +
        gp.howManyOfTeam(Team.RED_BULL));
    System.out.println("Team with more drivers: " +
        gp.hasMoreDrivers());
    System.out.println("Norris' team: " +
        gp.driverTeam("Norris"));
}
```