Ciências ULisboa | Informática

Programação Centrada em Objetos

2021/2022

Série 5

Herança, interfaces, classes abstratas

Exercícios

1. Com a finalidade de construir uma aplicação geral que suporte a realização de concursos para contratação de pessoas em empresas, vamos desenvolver alguns tipos de dados Java. Numa fase inicial existem os conceitos de *concurso (contest)*, *candidato* e *critério* de avaliação de candidatos. Considere já implementados os tipos Java abaixo e o `main` da classe `TextClient` que ilustra a criação e manipulação de alguns desses candidatos e concursos.

```
/** Criterios usados em avaliacao de candidatos a concursos */
public enum Criterio {

    COURSE_RANKING, ACADEMIC_GRADE, PROFESSIONAL_EXPERIENCE,
    SOFT_SKILLS, COMMUNITY_WORK, LEADERSHIP_SKILLS

}
```

```
public interface IContest
```

Method Summary

All Methods | Instance Methods | Abstract Methods

Modifier and Type	Method and Description
void	**addCandidate**(java.lang.String cand, int[] evals) Adiciona um candidato a este concurso se o tempo limite deste concurso nao foi ultrapassado Pre-condicao: inTime() && cand != null && evals != null && evals.length == Criterio.values().length
int	**contestEvaluation**(java.lang.String nome) A avaliacao de um dado candidato neste concurso Pre-condicao: nome != null && isEligible(nome)
java.util.List<java.lang.String>	**eligibleCandidates**() Lista de nomes de candidatos elegiveis para este concurso
boolean	**estahEmConcurso**(java.lang.String cand) Um dado individuo estah em concurso? Pre-condicao: cand != null
int	**grade**(java.lang.String cand, Criterio c) A nota que um dado candidato tem num dado criterio Pre-condicao: cand != null && c != null
java.util.List<java.lang.String>	**higherRanked**(java.lang.String cand) Lista de nomes de candidatos cuja avaliacao neste concurso eh superior ah avaliacao de elem Pre-condicao: cand != null
boolean	**inTime**() Ainda nao passou a data limite do concurso?
boolean	**isEligible**(java.lang.String nome) Um dado candidato eh elegivel para concorrer neste concurso? Pre-condicao: nome != null
int	**maxGrade**() Valor maximo de avaliacao dos candidatos nos varios criterios
int	**minGrade**() Valor minimo de avaliacao dos candidatos nos varios criterios

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

Um candidato a concurso

Constructor Summary

Constructors

Constructor and Description
Candidate(java.lang.String name, int[] evaluations) Construtor Pre-condicao: name != null && evaluations != null

Method Summary

All Methods | Instance Methods | Concrete Methods

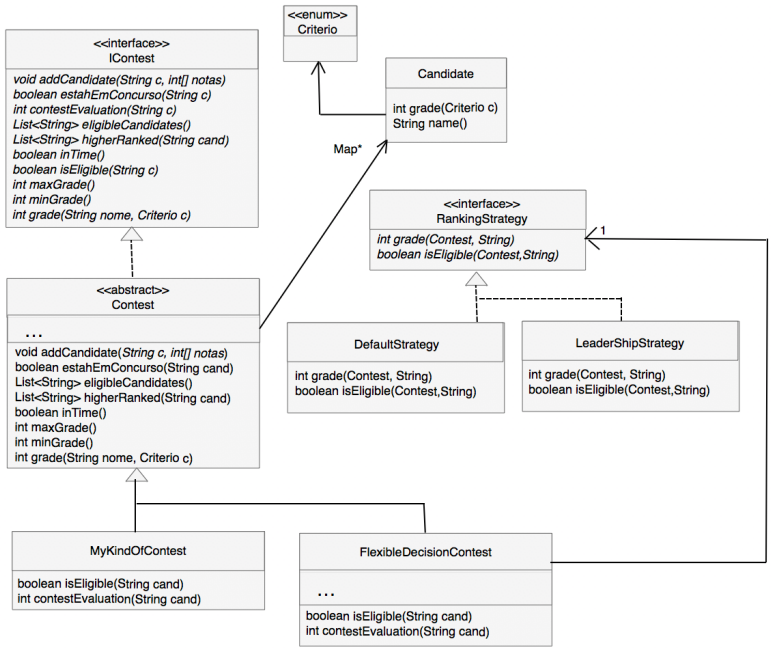
Modifier and Type	Method and Description
int	**grade**(Criterio cri) A classificacao deste candidato em relacao a um determinado criterio de avaliacao Pre-condicao: cri != null
java.lang.String	**name**() O nome deste candidato

```java
import java.time.LocalDateTime;

public class TextClient {

    public static void main(String[] args) {

        IContest myCont = new MyKindOfContest(
                          LocalDateTime.of(2021, 11, 2, 16, 30),1,5);
        if(myCont.inTime()) {
            myCont.addCandidate("Rita", new int[]{3,4,3,5,3,5});
            myCont.addCandidate("Pedro", new int[]{5,5,5,4,4,3});
        }
        System.out.println("Avaliacao da Rita: " +
                           myCont.contestEvaluation("Rita"));
        System.out.println("Rita eh elegivel? " +
                           myCont.isEligible("Rita"));
        System.out.println("Avaliacao do Pedro: " +
                           myCont.contestEvaluation("Pedro"));
        System.out.println("Pedro eh elegivel? " +
                           myCont.isEligible("Pedro"));

        System.out.println("Elegiveis:");
        for(String cand : myCont.eligibleCandidates()) {
            System.out.println(cand);
        }

        System.out.println("Candidatos melhores que a Rita:");
        for(String cand : myCont.higherRanked("Rita")) {
            System.out.println(cand);
        }

        System.out.println("Candidatos melhores que o Pedro:");
        for(String cand : myCont.higherRanked("Pedro")) {
            System.out.println(cand);
        }

        System.out.println("===  BYE  ===");
    }
}
```

O *output* do `main` da classe `TextClient` é o seguinte:

```
Avaliacao da Rita: 3
Rita eh elegivel? false
Avaliacao do Pedro: 5
Pedro eh elegivel? true
Elegiveis:
Pedro
Candidatos melhores que a Rita:
Pedro
Candidatos melhores que o Pedro:
===========  BYE  ==============
```

Um concurso aceita vários candidatos durante o tempo em que está "aberto"; alguns desses candidatos podem não ser elegíveis para o concurso; a condição de elegibilidade depende de cada tipo de concurso; aos candidatos elegíveis é atribuída uma avaliação no concurso que, tipicamente, depende das notas individuais que cada candidato teve a cada critério.



A. Implemente a classe <u>abstrata</u> `Contest` que representa concursos no geral, contendo informação sobre os candidatos ao concurso, sobre a data limite de candidaturas ao concurso e sobre os valores mínimo e máximo, para cada critério, que as notas dos candidatos, nos vários critérios, podem ter. Deve implementar o construtor:

- `Contest(LocalDateTime dayLimit, int minGrade, int maxGrade)` onde `dayLimit` é a data limite para candidaturas ao concurso, e `minGrade` e `maxGrade` são os valores mínimo e máximo que as avaliações dos candidatos em cada critério podem ter;

e os seguintes métodos (alguns serão eventualmente *template*, ou seja, implementações esqueleto):

- `boolean inTime()` que devolve `true` se este concurso ainda está "aberto", ou seja, se ainda aceita candidatos. Use os métodos `public boolean isAfter(LocalDateTime d)` que retorna `true` se o objeto alvo da invocação é posterior a `d` e `public LocalDateTime now()` que retorna a data corrente, da classe `LocalDateTime` para comparar a data limite do concurso com a data corrente;
- `void addCandidate(String cand, int[] notas)` que adiciona um candidato a este concurso, que tenha o nome `cand` e as notas aos vários critérios dadas pelo *array* `notas`;
- `boolean estahEmConcurso(String cand)` que retorna `true` se um candidato de nome `cand` está neste concurso e `false` caso contrário;
- `List<String> eligibleCandidates()` que devolve uma lista com os nomes dos candidatos considerados elegíveis para este concurso;
- `List<String> higherRanked(String cand)` que devolve uma lista com os nomes dos candidatos cuja avaliação neste concurso é superior à do candidato de nome `cand` independente/ de serem elegíveis ou não;
- `int grade(String cand, Criterio c)` que devolve a nota que o candidato de nome `cand` tem no critério `c` ;

e ainda `int maxGrade()` e `int minGrade()` que devolvem os valores máximo e mínimo que as notas dos candidatos nos vários critérios podem ter.

Deve deixar a implementação dos seguintes métodos para as sub-classes:

- `boolean isEligible(String candidate)` que devolve `true` se o candidato de nome `candidate` é

elegível para este concurso;

- `int contestEvaluation(String candidate)` que devolve a avaliação dada ao candidato de nome `candidate` neste concurso (que depende, de alguma forma, das avaliações desse candidato a cada critério de avaliação).

B. Implemente a classe concreta `MyKindOfContest` que representa um concurso para o qual:

- um candidato é elegível se a média das notas que tem nos critérios `ACADEMIC_GRADE`, `PROFESSIONAL_EXPERIENCE` e `COURSE_RANKING` for igual ou superior ao valor máximo que as notas dos candidatos podem ter nos vários critérios, menos 1;
- a avaliação de um candidato é igual à média das notas nesses três critérios, arredondada às unidades.

Esta classe deve implementar o construtor:

```
MyKindOfContest(LocalDateTime dayLimit, int minGrade, int maxGrade);
```

e os métodos públicos, ainda não implementados, do *interface* `IContest`.

C. Queremos agora construir um novo tipo de concurso – `FlexibleDecisionContest` –, que pretende ser mais abrangente, ou seja, em que é possível definir em tempo de execução qual a estratégia para determinar se um concorrente é elegível e qual a avaliação de um candidato no concurso.

Esta classe deve implementar o construtor:

```
FlexibleDecisionContest (LocalDateTime dayLimit, int minGrade, int maxGrade);
```

em que, para além de inicializar os atributos definidos na superclasse, inicializa também a estratégia usada com uma instância de `DefaultStrategy` (ver abaixo).

Deve ainda implementar os métodos públicos, ainda não implementados, do *interface* `IContest` e um novo método de instância:

```
void chooseRankingStrategy(RankingStrategy strat) –
```

que permite modificar a estratégia usada para cálculo da elegibilidade e a avaliação dos candidatos.

D. A ideia é existirem várias estratégias concretas possíveis, que implementam o interface `RankingStrategy` mas aqui só queremos definir duas: `DefaultStrategy` e `LeaderShipStrategy`.

Na classe `DefaultStrategy` a avaliação de um candidato `ca` num concurso `cont` é igual à média de notas que `ca` tem nos vários critérios; um candidato é elegível se a sua avaliação está no intervalo [*min*, *max*] em que *min* e *max* são os valores de `minGrade()` e `maxGrade()` do concurso `cont`.

`public interface RankingStrategy`

Method Summary

All Methods	Instance Methods	Abstract Methods

Modifier and Type	Method and Description
`int`	**`grade`**`(Contest cont, java.lang.String cand)` A avaliacao de um candidato num concurso Pre:condicao: cont != null && cand != null
`boolean`	**`isEligible`**`(Contest cont, java.lang.String cand)` Um dado candidato eh elegivel num dado concurso? Pre:condicao: cont != null && cand != null

Na classe `LeaderShipStrategy` a avaliação de um candidato `ca` num concurso `cont` é igual a 1 se `ca` tem uma nota no critério `LEADERSHIP_SKILLS` igual ao valor máximo definido em `cont` para as notas dos critérios dos candidatos, e zero caso contrário; um candidato é elegível se a sua avaliação é igual a 1.

Veja um exemplo de utilização destas classes no `main` da classe `TextClient2` (figuras na próx. página).

E. Implemente os métodos necessários para que as classes `Contest` e `MyKindOfContest` definam uma noção de igualdade apropriada e sejam clonáveis.

```java
import java.time.LocalDateTime;

public class TextClient2 {

    public static void main(String[] args) {

        FlexibleDecisionContest yourCont = new FlexibleDecisionContest(
                   LocalDateTime.of(2021, 11, 2, 16, 30),
                   1,5);
        yourCont.addCandidate("Rita", new int[]{3,4,3,5,3,5});
        yourCont.addCandidate("Pedro", new int[]{5,5,5,4,4,3});
        System.out.println("Leadership skills da Rita " +
                           yourCont.grade("Rita",Criterio.LEADERSHIP_SKILLS));
        System.out.println("Leadership skills do Pedro " +
                           yourCont.grade("Pedro",Criterio.LEADERSHIP_SKILLS));

        avaliacoes("Rita", "Pedro", yourCont);

        RankingStrategy strat = new LeaderShipStrategy();
        yourCont.chooseRankingStrategy(strat);
        avaliacoes("Rita", "Pedro", yourCont);
    }

    /*
     * Imprime informação sobre dois candidatos dados num concurso dado
     */
    private static void avaliacoes(String cand1, String cand2, Contest contes) {
        System.out.println("==================================");
        if(contes instanceof FlexibleDecisionContest) {
            System.out.println("Estrategia de ranking: " +
                     ((FlexibleDecisionContest)contes).rankingStrategy().getClass());
        }
        System.out.println("Avaliacao de " + cand1 + ": " +
                           contes.contestEvaluation(cand1));
        System.out.println(cand1 + " e' elegivel? " + contes.isEligible(cand1));
        System.out.println("Avaliacao de " + cand1 + ": " +
                           contes.contestEvaluation(cand2));
        System.out.println(cand2 + " e' elegivel? " + contes.isEligible(cand2));
        System.out.println("Elegiveis:");
        for(String cand : contes.eligibleCandidates()) {
            System.out.println(cand);
        }
    }
}
```

O *output* do `main` da classe `TextClient2` é o seguinte:

```
Leadership skills da Rita 5
Leadership skills do Pedro 3
==================================
Estrategia de ranking: class DefaultStrategy
Avaliacao da Rita: 3
Rita eh elegivel? true
Avaliacao do Pedro: 4
Pedro eh elegivel? true
Elegiveis:
Rita
Pedro
==================================
Estrategia de ranking: class LeaderShipStrategy
Avaliacao da Rita: 1
Rita eh elegivel? true
Avaliacao do Pedro: 0
Pedro eh elegivel? false
Elegiveis:
Rita
```

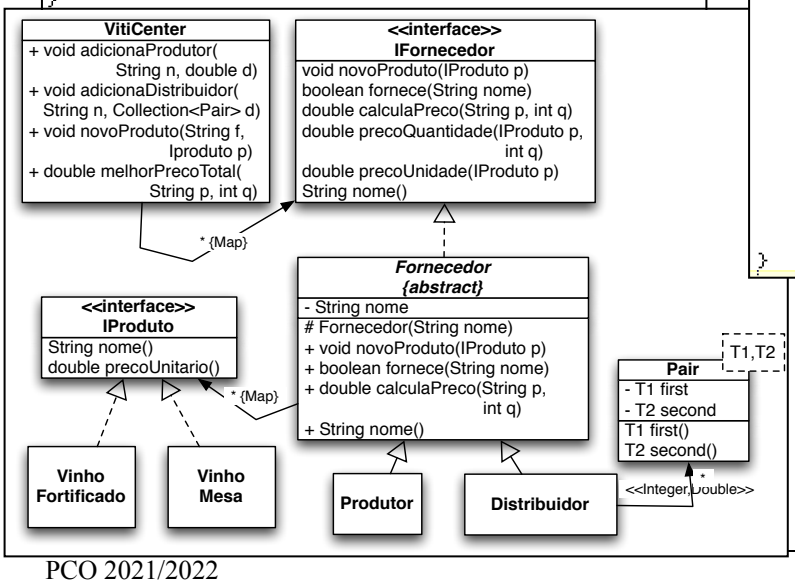
2. A aplicação *VitiCenter* faz a ponte entre fornecedores de produtos vinícolas e clientes interessados em conhecer preços e as melhores ofertas para dados produtos. Numa fase inicial existe o conceito de *fornecedor*. Os fornecedores vendem *produtos vinícolas*, a preços por eles determinados, aos quais podem ainda aplicar descontos de quantidade. Considere os tipos Java abaixo e o `main` que ilustra a simulação de consulta de preços a dois fornecedores em concreto. Tenha também em atenção o diagrama de classes apresentado.

```java
public class VitiCenter {

    // os fornecedores inscritos na VitiCenter
    private Map<String,IFornecedor> fornecedores;

    /** Construtor */
    public VitiCenter(){
        fornecedores = new HashMap<>();
    }

    /**
     * Adicionar um fornecedor produtor
     * @param nome - O nome do produtor
     * @param descPercent - Valor em percentagem para desconto em
     *                      compras superiores a uma garrafa
     * @requires nome != null && descPercent >= 0
     */
    public void adicionaProdutor(String nome, double descPercent){
        Produtor prod = new Produtor(nome,descPercent);
        fornecedores.put(nome, prod);
    }

    /**
     * Adicionar um fornecedor distribuidor
     * @param nome - O nome do distribuidor
     * @param escaloesDescontos - define os valores de desconto
     *            (entre 0 e 1) que este distribuidor aplica
     *            consoante as quantidades de produto vendidas
     * @requires nome != null && escaloesDescontos != null
     */
    public void adicionaDistribuidor(String nome,
                Collection<Pair<Integer,Double>> escaloesDescontos){
        Distribuidor dist = new Distribuidor(nome,escaloesDescontos);
        fornecedores.put(nome, dist);
    }

    /**
     * Adiciona um dado produto ah gama de produtos que um dado
     * fornecedor fornece
     * @param fornecedor - O nome do fornecedor
     * @param produto - O produto a acrescentar ah gama de produtos
     *                  fornecida pelo fornecedor em questão
     * @requires nome != null && escaloesDescontos != null
     */
    public void novoProduto(String fornecedor, IProduto produto){
        fornecedores.get(fornecedor).novoProduto(produto);
    }

    /**
     * O melhor preco para uma dada quantidade de um dado produto,
     * de entre os precos que os varios fornecedores fazem
     * @param prod - O nome do produto desejado
     * @param quant - O numero de garrafas desejado
     * @return O valor considerado como o melhor preco a pagar
     * por quant unidades do produto prod
     */
    public double melhorPrecoTotal(String prod, int quant){
        double melhor = Double.MAX_VALUE;
        for(IFornecedor f : fornecedores.values()){
            if(f.fornece(prod)){
                double precoTudo = f.calculaPreco(prod, quant);
                if(precoTudo < melhor){
                    melhor = precoTudo;
                }
            }
        }
        return melhor;
    }
}
```

```java
public class TextClient {
    public static void main(String[] args) {
        VitiCenter centro = new VitiCenter();

        IProduto p1 = new VinhoMesa("Arco-Iris",4);
        IProduto p2 = new VinhoFortificado("Tempestade", 35);

        centro.adicionaProdutor("Joao", 0.2);
        centro.novoProduto("Joao", p1);
        centro.novoProduto("Joao", p2);

        IProduto p3 = new VinhoMesa("Arco-Iris",2.5);
        IProduto p4 = new VinhoFortificado("Tempestade", 29);
        List<Pair<Integer,Double>> descsQuants = new ArrayList<>();
        descsQuants.add(new Pair(10,0.1));
        descsQuants.add(new Pair(100,0.3));
        descsQuants.add(new Pair(500,0.5));

        centro.adicionaDistribuidor("Vinnus", descsQuants);
        centro.novoProduto("Vinnus", p3);
        centro.novoProduto("Vinnus", p4);

        double preco = centro.melhorPrecoTotal("Arco-iris",11);
        System.out.println("Preco total = " + preco);

        preco = centro.melhorPrecoTotal("Tempestade",1);
        System.out.println("Preco total = " + preco);
    }
}
```

```java
public interface IFornecedor {

    /**
     * Regista que este fornecedor tambem fornece o produto
     * indicado
     * @param p - O novo produto
     */
    public void novoProduto(IProduto p);

    /**
     * O preco total de um dado numero de garrafas de um
     * dado produto vinicola
     * @param prod - O nome do produto
     * @param quant - Numero de garrafas desejadas
     */
    public double calculaPreco(String prod, int quant);

    /**
     * O preco total de um dado numero de garrafas de um
     * dado produto vinicola
     * @param prod - O produto desejado
     * @param quant - Numero de garrafas desejadas
     * @return
     */
    public double precoQuantidade(IProduto prod, int quant);

    /**
     * O preco por uma garrafa de um dado produto
     * @param prod - O produto desejado
     * @return
     */
    public double precoUnidade(IProduto prod);

    /**
     * Este fornecedor fornece um dado produto?
     * @param nome - O nome do produto
     * @return - true se fornece; false se nao fornece
     */
    public boolean fornece(String nome);

    /**
     * O nome deste fornecedor
     * @return
     */
    public String nome();
}
```

```java
/**
 * Tipo que representa um
 * produto vinicola
 */
public interface IProduto {
    /** O nome do produto */
    String nome();

    /**
     * O preco por garrafa
     * de produto
     */
    double precoUnitario();
}
```



Temos dois tipos concretos de fornecedor – o *produtor*, que define um único valor de desconto para vendas acima de uma unidade de produto, e o *distribuidor*, que define valores de desconto diferentes para intervalos de quantidades diferentes. Atenção que um produto de nome X pode ser representado no sistema por várias instâncias, cada uma com o seu preço, uma para cada fornecedor que o comercializa.

A. Implemente a classe abstrata `Fornecedor`. Esta classe deve implementar o construtor:

- `Fornecedor(String nome)` onde `nome` é o nome do fornecedor;

e os métodos públicos indicados no diagrama de classes, de acordo com a documentação do interface `IFornecedor`. O método `double calculaPreco(String prod, int quant)` é um método *template* (ou implementação esqueleto) que identifica o produto cujo nome é `prod` e, consoante a quantidade `quant`, invoca um dos métodos que calculam o preço.

A implementação dos métodos `precoUnidade` e `precoQuantidade` é deixada para as sub-classes.

B. Implemente a classe concreta `Produtor` que representa um fornecedor que produz os seus próprios produtos vinícolas. Deve implementar o construtor:

- `Produtor (String nome, double desconto)` onde `nome` é o nome do fornecedor e `desconto` é o valor (entre 0 e 1) que o produtor faz em vendas superiores a uma unidade (garrafa);

e os métodos `precoUnidade(IProduto prod)` e `precoQuantidade(IProduto prod, int quant)`; o primeiro deve devolver o preço unitário definido para o produto `prod` por este produtor, e o segundo deve devolver o valor total de `quant` unidades (garrafas) do produto `prod`, com o desconto deste produtor já aplicado ao preço unitário desse produto.

C. Implemente a classe concreta `Distribuidor` que representa um fornecedor que vende produtos vinícolas produzidos por outrem, nacionais e estrangeiros. Deve implementar o construtor:

- `Distribuidor (String nome, Collection<Pair<Integer,Double>> limites)` onde `nome` é o nome do fornecedor e `limites` define os valores de desconto (entre 0 e 1) que este distribuidor aplica consoante as quantidades de produto vendidas;

e os métodos `precoUnidade(IProduto prod)` e `precoQuantidade(IProduto prod, int quant)`; tanto um como o outro devem devolver o valor de uma ou `quant` unidades do produto `prod` tendo em conta os escalões de descontos a aplicar por este produtor (que foram determinados no construtor, através da coleção `limites`); (exemplo: se a coleção `limites` contiver os pares <10, 0.1> e <100, 0.3>, isso significa que uma venda de x unidades, em que $x \in [10, 100[$, terá 10% de desconto sobre o valor calculado usando o preço unitário do produto; uma venda superior ou igual a 100 unidades terá 30% de desconto sobre o valor calculado usando o preço unitário do produto; neste exemplo, uma venda de 500 unidades de um produto cujo preço unitário é 2 euros, teria um valor total de $500 \times 2 \times (1 - 0.3) = 700$ euros.

D. Implemente os métodos necessários para que a classe `Distribuidor` defina uma noção de igualdade apropriada e seja clonável.