Ciências ULisboa

Programação Centrada em Objetos

Licenciatura em Tecnologias da Informação

Projeto – Fase 2

2021/2022

O objetivo final do projeto de PCO, feito em 3 fases, é pôr em prática os conhecimentos que vão sendo adquiridos nas aulas.

Nesta segunda fase do projeto vão exercitar, para além das matérias já exercitadas na Fase1, as seguintes matérias lecionadas em PCO: construção de classes que definem tipos de dados, relação cliente/fornecedor entre classes, o *interface* e a classe genéricos `List` e `ArrayList`, enumerados, métodos/atributos de instância e de classe.

Alguns conceitos importantes

Vamos continuar a trabalhar com viagens interplanetárias. Temos o conceito de planeta e o de sistema solar.

- Cada planeta tem um determinado conjunto de características, ou propriedades, que o definem;
- Um sistema solar contém planetas, representados numa estrutura bi-dimensional;
- Nesta fase do projeto, uma viagem é sempre feita seguindo um percurso em forma de lagarta horizontal através da matriz de planetas do sistema solar; na próxima fase serão considerados também outros tipos de percurso.

O que se pretende de vós nesta 2ª fase do projeto?

Nesta 2ª fase do projeto a vossa tarefa é construir, em Java, dois dos tipos de dados – `SistemaSolar` e `Planeta` – necessários para executar o programa da classe `PCOFase2`, dada por nós.

No método `main` desta classe são criadas 3 listas de propriedades que serão usadas para a criação dos planetas de um sistema solar. É criada uma instância de `Planeta` e invocados dois métodos sobre esse objeto. De seguida são criados dois *arrays* bi-dimensionais de planetas, um que não é uma matriz e outro que é matriz, e é invocado o método *static* `universoValido` da classe `SistemaSolar` para cada um desses *arrays*. De seguida a matriz de planetas é usada para criar uma instância de `SistemaSolar` sobre a qual são invocados vários métodos. Finalmente é criada uma instância de `Verificador` para esse sistema solar, que é usado para verificar se o sistema solar satisfaz alguns requisitos.

Na classe `PCOFase2` que tem o método `main` acabado de descrever, é usada uma instância de outra classe, também dada por nós – a classe `Verificador`:

- **`Verificador`**: os objetos desta classe têm um sistema solar associado e sabem verificar se esse sistema solar satisfaz determinados requisitos. A classe oferece as seguintes funcionalidades:
  - construtor **`Verificador(SistemaSolar sistema)`** que constrói um verificador que fica associado ao sistema solar **`sistema`**;
  - método **`boolean verificaPropriedade(String requisitos)`** que devolve `true` se o sistema solar associado a este verificador satisfaz os requisitos **`requisitos`**; tem como pré-condição que a *string* **`requisitos`** é da forma $X_1{:}s_1;X_2{:}s_2;\ldots;X_n{:}s_n$ em que cada X_i representa um valor inteiro positivo e em que cada s_i e' da forma $p_1,\ldots,p_n$ sendo p_i o nome de uma propriedade;
  - método privado que é invocado pelo método anterior.

NOTA 1: repare que o método `verificaPropriedade` da classe `Verificador` faz essencialmente o mesmo que o método `verificaPropriedade` pedido na Fase 1 do projeto.

- A grande diferença é que nesta Fase 2 o método é um método de instância (não *static*) que trabalha sobre um sistema solar que é atributo da classe `Verificador`.
- Relembre que na Fase 1 o método era *static* (de classe) e recebia como parâmetros, para além dos requisitos a verificar, também o *array* com informação sobre as propriedades dos planetas existentes e o sentido do trajeto (regular ou inverso); Agora, a informação sobre os planetas pertence ao próprio sistema solar que é atributo do verificador e a decisão de qual é o próximo planeta também é tomada pelo mesmo sistema solar;
- Nesta fase 2 não consideramos o sentido do percurso (regular ou inverso);
- Resumindo, nesta Fase 2 criámos a noção de "objetos que sabem fazer uma verificação de propriedades sobre um sistema solar, sempre que isso lhes é pedido" – as instâncias de `Verificador`.

Os tipos de dados a construir pelos alunos

Os tipos de dados a construir pelos alunos incluem necessariamente o seguinte enumerado:

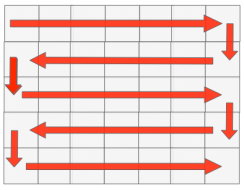
- **`PROPRIEDADE`**, que representa as possíveis propriedades de um planeta:
  - `HAS_WATER`, `HAS_LIGHT`, `FRIENDLY` e `BREATHABLE`;

e as seguintes classes:

- **`Planeta`**: os objetos desta classe representam planetas. A classe deve oferecer as seguintes funcionalidades:

- construtor **`Planeta(String nome, List<Propriedade> props)`** que inicializa um novo Planeta cujo nome é **`nome`**, e cujas propriedades são as contidas na lista **`props`**;
- método **`String nome()`** que devolve o nome deste planeta;
- método **`boolean temPropriedade(Propriedade p)`** que devolve `true` se este planeta tem a propriedade **`p`**;
- método **`boolean temTodas(List<Propriedade> props)`** devolve `true` se este planeta tem todas as propriedades contidas na lista **`props`**;
- método **`String toString()`** que devolve a representação textual deste Planeta (*ver NOTA 2 mais abaixo*).

- **`SistemaSolar`**: os objetos desta classe representam sistemas solares formados por planetas. A classe deve oferecer as seguintes funcionalidades:
  - construtor **`SistemaSolar(String nome, Planeta[][] planetas)`** que inicializa um sistema solar cujo nome é **`nome`** e contém uma matriz de planetas igual a **`planetas`**; tem como pré-condição a condição `universoValido(planetas);`
  - o método (de classe) **`static boolean universoValido(Planeta[][] arrayBi)`** que devolve `true` se o *array* bi-dimensional **`arrayBi`** é uma matriz e não contém elementos a `null`;
  - método **`String nome()`** que devolve o nome deste sistema solar;
  - método **`boolean temPlaneta(String nome)`** que devolve `true` se este sistema solar contém um planeta cujo nome é **`nome`**;
  - método **`List<String> comPropriedades(List<Propriedade> props)`** que devolve uma lista contendo os nomes de todos os planetas deste sistema solar que têm todas as propriedades referidas na lista **`props`**;
  - método **`int[] quantosPorPropriedade()`** que devolve um vetor em que o i-ésimo elemento representa o número de planetas deste sistema solar que tem a i-ésima propriedade do vetor `Propriedade.values();`
  - método **`boolean nEsimoTem(int n, List<Propriedade> props)`** que devolve `true` se o **`n`**-ésimo planeta deste sistema solar tem todas as propriedades referidas na lista **`props`**; a ordem pela qual são considerados os planetas no sistema solar segue a direção de uma lagarta horizontal (ver imagem ao lado) que volta ao início quando termina;



  - método **`Propriedade maisFrequente()`** que devolve a propriedade que aparece mais vezes nos planetas deste sistema solar;
  - método **`String toString()`** que devolve a representação textual desta instituição (*ver NOTA 2 abaixo*).

Com o objetivo de estruturar bem o vosso código, as classes pedidas podem ter mais métodos que os listados acima, desde que sejam métodos **privados**.

Já sabe que para testar as suas classes deve usar a classe `PCOFase2` acessível na página da cadeira. Aí está também acessível a classe `Verificador`.

O *output* resultante de executar o programa `PCOFase2` é apresentado no ficheiro `Anexo-Output`, acessível também na página de PCO.

NOTA 2: Pode ver o formato da representação textual (devolvida pelo método `toString()`) de um *Planeta* logo na primeira linha do ficheiro `Anexo-Output`.

No mesmo ficheiro `Anexo-Output` poderá ver o formato da representação textual de um *SistemaSolar*. Vai desde a linha

```
Tarvos
```

até à linha:

```
Kallichore: BREATHABLE HAS_WATER HAS_LIGHT Cyllene: FRIENDLY BREATHABLE
Eukelade: BREATHABLE HAS_WATER HAS_LIGHT
```

Repare que cada linha da matriz de planetas de um sistema solar é formada pelas representações textuais dos planetas que a formam.

O que entregar?

Não há relatório a entregar porque o vosso *software* é a vossa documentação. Assim, têm que comentar condignamente as vossas classes `Planeta` e `SistemaSolar`: incluir no início de cada classe um cabeçalho Javadoc com `@author` (número do grupo e nome e número dos alunos que compõem o grupo); para cada método definido, incluir um cabeçalho com a sua descrição, e, se for caso disso, `@param`, `@requires` e `@return`.

Para entregar: Um ficheiro *zip* com as classes que compõem a vossa solução e com os ficheiros Javadoc que geraram a partir delas.

O nome do ficheiro *zip* que contém o vosso trabalho deverá ter o formato `PCOxxx.zip` (onde `xxx` é o número do vosso grupo).

Como entregar o trabalho?

Através do Moodle de PCO. Às 23h55 do dia acordado para a entrega, 19 de Novembro, os trabalhos entregues serão recolhidos.

Atenção que ao entregar o trabalho está a comprometer-se com o seguinte:

- O trabalho entregue é atribuível única e exclusivamente aos elementos que constituem o seu grupo;
- Qualquer indício de plágio será investigado e poderá levar ao não aproveitamento dos elementos do grupo e consequente processo disciplinar.