Ciências ULisboa

Programação Centrada em Objetos

Licenciatura em Tecnologias da Informação

Projeto – Fase 3

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 terceira fase do projeto vão exercitar, para além das matérias já exercitadas nas fases 1 e 2, as seguintes matérias lecionadas em PCO: interfaces, herança, classes abstratas, princípio "*programar para interfaces*".

Alguns conceitos importantes

Neste trabalho os alunos vão ter que construir vários *interfaces* e classes (abstratas e concretas) tal como indicado no diagrama de classes em UML apresentado em anexo a este enunciado.

O princípio de desenho/programação "*Programar para interfaces*" está bem refletido na estrutura de classes e interfaces:

- As 2 classes `Sistema1DSeguro` e `Sistema2D` (bem como `AbstractSistemaSolar`, a superclasse abstrata destas duas últimas), todas implementam o interface `SistemaSolar`;
- A classe `Sistema2D` tem um atributo do tipo `Direcionador` o qual irá referenciar, em tempo de execução, uma instância de uma das suas quatro possíveis implementações – `LinhaALinha`, `LagartaHorizontal`, `LagartaVertical` e `Espiral` (esta última é opcional).

Isto permite que (como poderão ver na classe `PCOFase3`)

- se usem objetos de qualquer daquelas primeiras 2 classes em todas as situações em que se pretende usar um sistema solar;
- se use carregamento dinâmico de classes para criar um objeto do tipo `Direcionador` sem conhecer o seu tipo concreto (o nome do tipo desejado é pedido ao utilizador).

O que se pretende de vós nesta 3ª fase do projeto?

Nesta 3ª fase do projeto a vossa tarefa é construir, em Java, vários dos tipos de dados apresentados no diagrama de classes fornecido, e que são necessárias para executar o programa da classe `PCOFase3`, dada por nós.

No método `main` desta classe:

- É lido um ficheiro de texto contendo a informação necessária para construir uma matriz de corpos celestes;
- De seguida, são mostrados ao utilizador os nomes dos subtipos de `Direcionador` existentes (lidos do ficheiro `configurações.properties`);
- É pedido ao utilizador que escolha um desses nomes;
- Com o nome escolhido pelo utilizador, é criada uma instância da classe correspondente, usando carregamento dinâmico de classes (se não existir acessível uma classe com o nome escolhido, por defeito é criada uma instância de `LinhaALinha`, fornecida aos alunos); este objeto vai ser usado na criação de uma instância de `Sistema2D`;
- É construída também uma instância de `Sistema1DSeguro`, baseada na mesma matriz de corpos celestes;
- De seguida são invocados vários métodos sobre estes dois sistemas solares e os resultados são apresentados no *standard output*;
- São criadas duas instâncias de `Sistema2D` e uma de `GrandePremioSideral` e esta última é usada para realizar várias jogadas com valores pedidos ao utilizador (até um dos viajantes escolher 0, para terminar); No fim da prova são anunciados os vencedores;
- De seguida é criada outra matriz de corpos celestes a partir de outro ficheiro de texto, outros dois viajantes e um novo grande prémio com um sistema 2D;
- São feitas várias jogadas nesta nova prova e no fim anunciados os vencedores;
- Finalmente é criada uma instância de `Sistema1DSeguro` baseada na mesma matriz e um novo grande prémio e são feitas algumas jogadas.

Os seguintes enumerados e classes já são dados:

- **`Par<P,S>:`** classe genérica que representa pares de elementos de tipos que podem ser diferentes;
- **`Ponto3D:`** classe cujas instâncias representam pontos a três dimensões;
- **`Viajante:`** classe cujas instâncias representam viajantes;
- **`LinhaALinha:`** classe que implementa o *interface* `Direcionador` e que usa uma ordem linha a linha para obtenção de um dado elemento da matriz universo;
- **`ConstroiSistemas:`** classe com um método `main` que permite criar um ficheiro de texto contendo informação gerada de forma aleatória, para ser usado para criar uma matriz de corpos celestes; os ficheiros `InfoSistema1.txt` e `InfoSistem2.txt` foram criados através da execução deste `main`; os alunos podem usá-lo se quiserem criar mais universos.

Para que o método `main` da classe `PCOFase3` funcione como descrito acima, os alunos terão que construir os seguintes tipos de dados.

Os *interfaces*:

- **`SistemaSolar`**, que define os métodos:
  - `String nome()` que devolve o nome do sistema solar;

- `boolean podeVisitar(List<Integer> aVisitar)` que devolve *true* se é possível visitar todos os elementos do sistema solar correspondentes aos números de ordem contidos na lista `aVisitar`;
- `int quantosElementos()` que devolve o número de elementos que este sistema solar define;
- `CorpoCeleste getElemento(int n)` que devolve o corpo celeste deste sistema solar correspondente ao número de ordem `n`;
- `BuracoNegro buracoNegroMaisPerto(CorpoCeleste c)` que devolve o buraco negro deste sistema solar que se encontra mais perto do corpo celeste `c`;

- **`Direcionador`**, que define os métodos:
  - `void defineUniverso(CorpoCeleste[][] m)`, que define o universo sobre o qual o direcionador vai trabalhar como sendo `m`;
  - `CorpoCeleste nEsimoElemento(int n)`, que devolve o elemento na posição `n` da matriz universo, de acordo com a estratégia de direcionamento implementada pelo direcionador;

A *classe abstrata*:

- **`AbstractSistemaSolar`**, que define o que é comum a vários tipos de sistemas solares. A classe implementa o *interface* **`SistemaSolar`**;

  Oferece o seguinte construtor:

  - `public AbstractSistemaSolar(String nome)` que inicializa um novo objeto com o nome `nome`;

  os seguintes métodos concretos:

  - `String nome()` que devolve o nome do sistema solar;
  - `boolean podeVisitar(List<Integer> aVisitar)` que devolve *true* se todos os inteiros contidos na lista `aVisitar` são maiores que zero e menores ou iguais a `quantosElementos()`;
  - `public String toString()` que devolve a representação textual correspondente ao nome do sistema solar;

  deixando por implementar os restantes métodos definidos no *interface* `SistemaSolar` (por isso é que esta classe tem que ser abstrata).

As *classes concretas*:

- **`CorpoCeleste`**: classe cujas instâncias representam corpos celestes ou seja, corpos que têm uma massa e uma posição num espaço tridimensional.

  Oferece o seguinte construtor:

  - `public CorpoCeleste (double massa, Ponto3D pos)` que inicializa um novo objeto com uma massa igual a `massa` e uma posição igual a `pos`;

  e os seguintes métodos concretos:

  - `public double massa()` e `public Ponto3D posicao()` que retornam a massa e a posição deste corpo celeste, respetivamente;

- `public double distancia(CorpoCeleste c)` que retorna a distância deste corpo celeste a `c`;
- `public boolean equals(Object other)` que, se `other` for um corpo celeste (usar o operador `instanceof` para verificar), retorna *true* se este corpo celeste têm uma massa igual à massa de `other` (com uma aproximação de 0.0001) e uma posição igual à posição de `other`;

- **`BuracoNegro`**: subclasse de **`CorpoCeleste`**, cujas instâncias representam corpos celestes com uma massa tal, que faz com que tenham uma grande força de atração que pode provocar a destruição de outros corpos nas suas proximidades.

  Oferece o seguinte construtor:

  - `public BuracoNegro(double massa, Ponto3D pos)` que inicializa um novo objeto com uma massa igual a `massa` e uma posição igual a `pos`;

  e o seguinte método concreto:

  - `public double distanciaMinimaSeguranca(CorpoCeleste c)` que retorna o valor da distância mínima a que um outro corpo celeste tem que estar para não ser muito afetado pela força de atração deste buraco negro; para efeitos deste projeto, este valor é igual à raiz quadrada do produto das massas do buraco negro e de `c`;

- **`Sistema2D`**: subclasse de **`AbstractSistemaSolar`**, cujas instâncias representam sistemas solares em que os corpos celestes estão organizados numa matriz e a sua ordem depende da forma como esta matriz é "percorrida". Um sistema 2D auxilia-se de um `Direcionador` para definir essa ordem. Na matriz podem existir buracos negros, corpos celestes vulgares e elementos a *null*, representando ausência de corpo celeste.

  Oferece o seguinte construtor:

  - `public Sistema2D(String nome, CorpoCeleste[][] m, Direcionador d)` que inicializa um novo objeto com nome `nome`, uma matriz de corpos celestes igual a `m` e um direcionador igual a `d`;

  os seguintes métodos concretos:

  - `int quantosElementos()` que devolve o número de elementos (linhas x colunas) da matriz deste sistema 2D;
  - `CorpoCeleste getElemento(int n)` que devolve o corpo celeste da matriz deste sistema solar correspondente ao número de ordem `n`, tal como calculado pelo direcionador associado a este sistema 2D;
  - `BuracoNegro buracoNegroMaisPerto(CorpoCeleste c)` que devolve o buraco negro deste sistema solar que se encontra mais perto do corpo celeste `c`;

  e redefine os seguintes métodos:

- `boolean podeVisitar(List<Integer> aVisitar)` que devolve *true* se todos os inteiros contidos na lista `aVisitar` são maiores que zero e menores ou iguais a `quantosElementos()` e se todos os elementos da matriz correspondentes aos números de ordem em `aVisitar` são corpos celestes vulgares (não são buracos negros nem *null*);
- `public String toString()` que devolve a representação textual do sistema solar (ver NOTA1 mais à frente neste texto);

- **`Sistema1DSeguro`**: subclasse de **`AbstractSistemaSolar`**, cujas instâncias representam sistemas solares em que os corpos celestes estão organizados linearmente, sem apresentarem buracos negros nem espaços *null*. Um sistema deste tipo tem por base um sistema 2D e funciona como que uma "versão filtrada" do universo deste, representando somente os elementos que são corpos celestes vulgares.

  Oferece o seguinte construtor:

  - `public Sistema1DSeguro(String nome, CorpoCeleste[][] m, Direcionador d)` que inicializa um novo objeto com nome `nome`, tendo por base um sistema 2D com o mesmo nome, a matriz `m` e um direcionador `LinhaALinha`; a lista linear de corpos celestes fica a conter todos os elementos do sistema 2D associado, por ordem crescente, que não são nem *null* nem buracos negros;

  os seguintes métodos concretos:

  - `int quantosElementos()` que devolve o número de elementos da lista de corpos celestes deste sistema;
  - `CorpoCeleste getElemento(int n)` que devolve o (`n`-1)-ésimo corpo celeste da lista deste sistema solar;
  - `BuracoNegro buracoNegroMaisPerto(CorpoCeleste c)` que devolve o buraco negro do sistema 2D associado que se encontra mais perto do corpo celeste `c`;

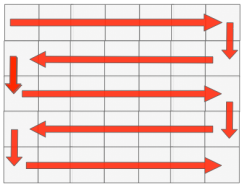
  e redefine o seguinte método:

  - `public String toString()` que devolve a representação textual do sistema solar (ver NOTA1 mais à frente neste texto);

- **`LagartaHorizontal`**, que implementa o *interface* **`Direcionador`**.

  Não define construtor explícito e implementa os métodos:

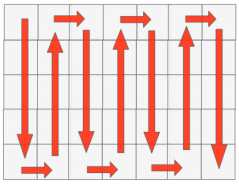
  - `void defineUniverso(CorpoCeleste[][] m)`, que define o universo sobre o qual o direcionador vai trabalhar como sendo `m`;
  - `CorpoCeleste nEsimoElemento(int n)`, que devolve o elemento na posição `n` da matriz universo, de acordo com a estratégia de lagarta horizontal (já vossa conhecida da Fase 2);



- **`LagartaVertical`**, que implementa o *interface* **`Direcionador`**.

  Não define construtor explícito e implementa os métodos:

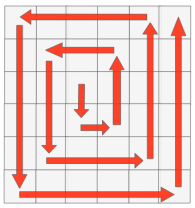
  - `void defineUniverso(CorpoCeleste[][] m)`, que define o universo sobre o qual o direcionador vai trabalhar como sendo `m`;
  - `CorpoCeleste nEsimoElemento(int n)`, que devolve o elemento na posição `n` da matriz universo, de acordo com a estratégia de lagarta vertical como ilustrado na figura ao lado;



- **`Elipse`**, que implementa o *interface* **`Direcionador`**. (Esta classe é opcional, só para quem quer mais um desafio ☺)

  Não define construtor explícito e implementa os métodos:

  - `void defineUniverso(CorpoCeleste[][] m)`, que define o universo sobre o qual o direcionador vai trabalhar como sendo `m`;
  - `CorpoCeleste nEsimoElemento(int n)`, que devolve o elemento na posição `n` da matriz universo, de acordo com a estratégia de elipse como ilustrado na figura ao lado;



- **`GrandePremioSideral`**: cujas instâncias representam grandes prémios siderais que se realizam sobre um dado sistema solar e em que vários viajantes vão fazendo jogadas (escolhas dos destinos para viagens no sistema solar). Define um prémio base que cada viajante recebe por visitar um dado corpo celeste. O prémio é maior em viagens de risco (aquelas para planetas a uma distância de risco de um buraco negro) e mais pequeno em determinadas condições (ver mais adiante).

  Oferece o seguinte construtor:

  - `public GrandePremioSideral(SistemaSolar ss, List<Viajante> jogs, int premioBase)` que inicializa um novo grande prémio a realizar no sistema solar `ss`, com os viajantes contidos em `jogs` (deve construir um `Map` em que as chaves são os nomes dos viajantes e os valores são os viajantes) e um prémio base igual a `premioBase`;

  e os seguintes métodos concretos:

  - `int premioBase()` que devolve o prémio base definido para este grande prémio;
  - `void fazJogada(List<Par<String,Integer>> jogadas)` que regista as jogadas dos vários viajantes que participam no grande prémio; neste método, para cada par `<nome,numero>` da lista `jogadas`, devem ser dados os seguintes passos:
    - obter o elemento de ordem `numero` do sistema solar (chamemos-lhe `c`);
    - se `c` não for *null*,

- registar viagem:
  - verificar se o viajante de nome `nome` (chamemos-lhe `v`) pode viajar para a posição de `c`; caso possa, e caso a posição de `c` seja diferente daquela onde `v` se encontra na altura, mudar a posição global de `v`;
- registar nova pontuação:
  - se `v` não chegou a mover-se (porque não tinha combustível ou porque escolheu o mesmo sítio onde já se encontrava) deverá ser penalizado retirando-lhe o equivalente a um quinto da pontuação que ele tem nesse momento;
  - se `v` se moveu, então:
    - se `c` é um buraco negro, deverá ser penalizado retirando-lhe o valor `PONTOS_BURACO_NEGRO`;
    - caso contrário, deverá ser premiado com o valor do prémio base, o qual deverá ser multiplicado pela `TAXA_RISCO` se `c` estiver a uma distância menor que a distância mínima de segurança do buraco negro mais próximo;
  - se `c` for *null*, deverá ser penalizado retirando-lhe o equivalente a metade da pontuação que ele tem nesse momento;
- `List<String> vencedores()` que devolve o(s) nome(s) do(s) viajante(s) que obteve(obtiveram) a maior pontuação;
- `public String toString()` que devolve a representação textual do grande prémio (ver NOTA1 mais à frente neste texto);

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 `PCOFase3`.

NOTA 1: Pode ver vários exemplos do formato da representação textual (devolvida pelo método `toString()`) de um *Sistema2D,* de um *Sistema1DSeguro* e de um *GrandePremioSideral* nos vários ficheiros de *output* dados por nós.

A representação do *Sistema2D*:

```
Nome: Sirius
Direcionador: LagartaHorizontal
   (0,0,6)      (0,1,13)     (0,2,4)      (0,3,26)       null
     null         null         null         null         null
   (2,0,13)     (2,1,13)       null       (2,3,22)     (2,4,22)
 B(3,0,13)      (3,1,26)       null         null         null
```

A representação do *Sistema1DSeguro*:

```
Nome: Sirius
Planetas:
(0,0,6) (0,1,13) (0,2,4) (0,3,26) (2,0,13) (2,1,13) (2,3,22) (2,4,22) (3,1,26)
```

A representação do *GrandePremioSideral*:

```
============ GRANDE PREMIO ============
Premio base: 500
Sistema solar:
Nome: Sirius
Direcionador: LagartaHorizontal
   (0,0,6)      (0,1,13)     (0,2,4)      (0,3,26)       null
     null         null         null         null         null
   (2,0,13)     (2,1,13)       null       (2,3,22)     (2,4,22)
 B(3,0,13)      (3,1,26)       null         null         null
Viajantes:
Nome: Luke Skywalker  Pontuacao: 0  Posicao: (0,0,6)  Combustivel: 30.0
Nome: Darth Vather  Pontuacao: 0  Posicao: (0,0,6)  Combustivel: 30.0
```

Material fornecido

Um *zip* contendo:

- Um ficheiro `Outputs.zip` contendo ficheiros de texto exemplificativos do *output* que o `main` da classe `PCOFase3` deverá produzir para várias escolhas do utilizador quando o programa pede um tipo de direcionador;
- Ficheiro `DiagClasses.jpeg` contendo o diagrama de classes desta aplicação;
- Um zip de uma pasta de nome `ProjetoFase3Alunos` que contém:
  - Classe genérica `Par`;
  - Classes `Ponto3D`, `Viajante`, `LinhaALinha` e `ConstroiSistemas`;
  - Classe `PCOFase3`;
  - Ficheiros de texto a serem usados no `main` da classe `PCOFase3`:
    - `InfoSistema1.txt` e `InfoSistema2.txt`;
    - `configuração.properties` (contém os nomes das classes que implementam o *interface* `Direcionador`)

O que entregar?

Não há relatório a entregar porque o vosso *software* é a vossa documentação. Assim, têm que comentar condignamente as vossas classes: 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 incluindo a sua descrição, e, se for caso disso, `@param`, `@requires` e `@return`.

Para entregar: Um ficheiro *zip* com as classes e interfaces que compõem a vossa solução (somente com as classes que vos pedimos para fazerem).

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, 15 de Dezembro, 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.