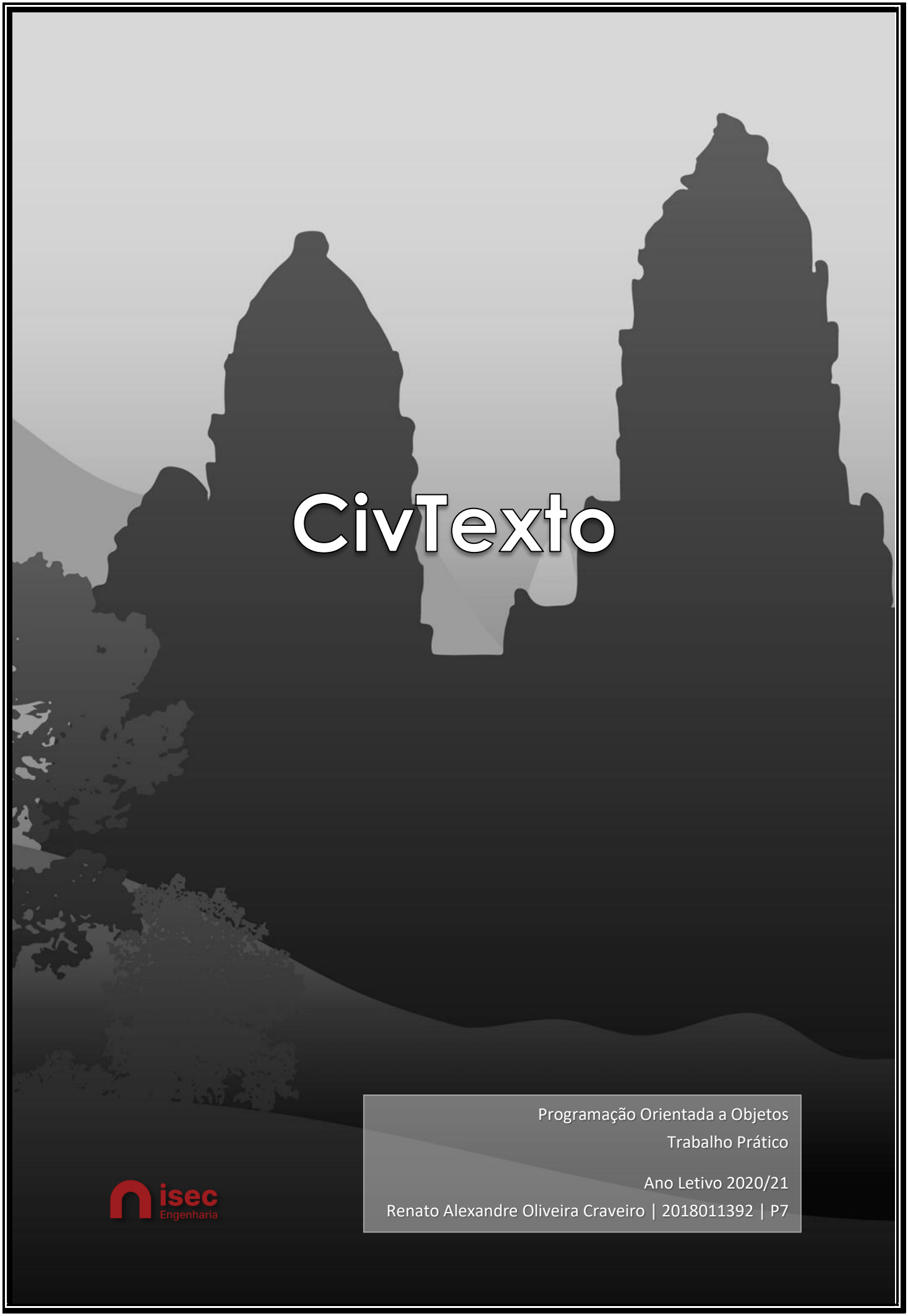

Programação Orientada a Objetos

Trabalho Prático

Ano Letivo 2020/21

Renato Alexandre Oliveira Craveiro | 2018011392 | P7

isec Engenharia

Índice

Organização do código apresentado

1.Quais foram as classes consideradas na primeira versão da aplicação que foi testada?

Na primeira versão testada da aplicação as classes consideradas foram:

- A classe Mundo
- A classe Territorio
- A classe Territorio_Inicial
- A classe Imperio
- E a classe Castelo

Estas classes foram utilizadas de forma a testa como a aplicação iria comunicar entre classes e como seria criado todo o ambiente de execução daí para a frente.

2.Quais os conceitos/classe que identificou ao ler o enunciado?

Os conceitos/classe identificados ao ler o enunciado foram:

- A classe Território
    - As subclasses Ilha e Continente
        - Os tipos de ilha e de continente são subclasses de cada uma, respetivamente
- A classe Imperio
- A classe Interpretador de comandos

3.Relativamente a duas das principais classes da aplicação, identifique em que classes ou partes do programa são criados, armazenados e destruídos os seus objectos.

Os objetos da classe Interpretador são criados e destruídos na função *main()* da aplicação.

Os objetos da classe Território (e das suas subclasses) são criados e destruídos na classe Interpretador, sendo que são posteriormente referenciados para o objeto “mundo” criado na *main()* da aplicação.

4.Indique um exemplo de uma responsabilidade atribuída a uma classe que esteja de acordo com a orientação dada acerca de Encapsulamento.

```
class Mundo
{
        vector<Territorio*> territoriosMundo;
        int turno;
public:

        Mundo() { turno = 0; }
        int getTurno();
        (
…)
};
```

Um exemplo de encapsulamento pode ser verificado na classe Mundo, em que o acesso a todos os dados guardados no objeto é feito através de métodos. Por exemplo o turno em que o mundo se apresenta é devolvido através do método "getTurno()".

5.De entre as classes que fez, escolha duas e justifique por que considera que são classes com objectivo focado, coeso e sem dispersão.

Classe Castelo: esta apenas está focada nas configurações de um território do tipo castelo, em que apenas devolve dados e muda consoante o turno em que se apresenta.

Classe Fortaleza: esta apenas está focada nas configurações de um território do tipo fortaleza, em que apenas devolve dados e muda consoante o turno em que se apresenta.

6.Relativamente à aplicação entregue, quais as classes que considera com responsabilidades de interface com o utilizador e quais as que representam a lógica?

A classe com responsabilidades de interface com o utilizador é:

- A classe Interpretador. Esta é responsável por receber *input* do utilizador, interpretá-lo, chamar a funcionalidade a executar/desejada e devolver uma resposta ao utilizador.

As classes restantes representam a lógica, sendo que têm funcionalidades que não interagem de maneira direta com o utilizador.

7.Identifique o primeiro objecto, para além da camada de interacção com o utilizador, que recebe e coordena uma funcionalidade de natureza lógica?

A primeira classe que recebe e coordena uma funcionalidade de natureza lógica é a classe "mundo". Esta recebe um ponteiro para um território (no caso da primeira interação, para o território inicial) e adiciona-o no mundo criado.

8.A classe que representa a envolvente de toda a lógica executa em pormenor muitas funcionalidades, ou delega noutras classes? Indique um exemplo em que esta classe delega uma funcionalidade noutra classe.

A classe que representa a envolvente da lógica é a classe "Interpretador" (responsável por toda a interação do utilizador com a aplicação). Esta delega funcionalidades de outras aplicações, conforme pedido pelo utilizador. Um exemplo é no avanço de uma fase:

```
void Interpretador::avanca(Mundo* world, Imperio* myImperio) {
   world->novoTurno();
   myImperio->recolheOuro();
   myImperio->recolheProdutos();
}
```

Esta função recebe ponteiros para o mundo e império em questão e executa as funções que cabem a cada um deles, na fase de avanço de turno. O mundo prepara um novo turno e o império faz a recolha de ouro e de produtos.

9.Apresente as principais classes da aplicação:

Classe: Interpretador

- Responsabilidades:

- Permite obter *input* do utilizador (ou *strings*) e executa o comando adequado ao comando introduzido, devolvendo *true* se executou com sucesso e *false* se ocorreu algum erro

- Colaborações: Classes Mundo e Imperio

Classe: Mundo

- Responsabilidades:
  - Permite criar o mundo (atribuindo o valor de turno a 0), devolver o turno em que se encontra, devolver o numero de territórios que ainda não tenham sido conquistados, adicionar um território ao mundo, iniciar um novo turno, devolver um território através do seu nome, devolver uma *string* com os territórios não conquistados, devolver uma *string* com os territórios conquistados, e devolver uma *string* com todos os territórios existentes no mundo.
- Colaborações: Classes Territorio, Continente e suas subclasses, Ilha e suas subclasses e Territorio_Inicial

Classe: Imperio

- Responsabilidades:
  - Permite criar um Imperio ao receber um Territorio_Inicial, retorna uma *string* ao tentar conquistar um território, remove territórios do império, recolhe produtos e ouro dos territórios do império, cria militares para o império, devolve uma *string* com os dados do império, devolve o número de territórios, pontos, o máximo de produtos que pode ter, os produtos atuais que tem, o máximo de ouro que pode ter, o ouro atual, máximo de militares e os militares atuais que o império tem,
- Colaborações: Classes Mundo e Tecnologias (classe ainda não implementada)

Classe: Territorio

- Responsabilidades:
  - Permite criar um Territorio recebendo como parâmetros o seu nome, a sua resistência, os produtos, o ouro e os pontos que dá ao ser conquistado (só pede ao ser criado um território genérico, sendo que se for criado um tipo espec´´ifico de território – por exemplo um Castelo – estes atributos são automaticamente atribuídos). Devolve os dados do território através de uma *string*, de volve o se nome em *string*, o ouro que produz, os pontos que dá ao ser conquistado, os produtos que produz, a resistência que tem e se está ou não conquistado. Permite ainda ser conquistado (recebendo a quantidade de ataque que sofreu) e ser evoluído (sendo que esta última altera de acordo com o tipo de território).

Classe: Continente

- Responsabilidades:
  - Permite as mesmas funcionalidades de Territorio, alterando a devolução dos dados do território, informando que este pertence a um continente.
- Colaborações: é uma subclasse de Territorio

Classe: Continente

- Responsabilidades:
    - Permite as mesmas funcionalidades de Territorio, alterando a devolução dos dados do território, informando que este é uma Ilha.
- Colaborações: é uma subclasse de Territorio~

Classe: Territorio_Inicial

- Responsabilidades:
    - Permite as mesmas funcionalidades de Territorio, atribuindo os valores por defeito do território inicial
- Colaborações: é uma subclasse de Territorio

Classe: Debugger

- Responsabilidades:
    - Permite as funcionalidades de debugging, tendo acesso *friend* às classes Mundo, Imperio e Tecnologias.
- Colaborações: acesso *friend* às classes Mundo, Imperio e Tecnologias.

Classe: savegame

- Responsabilidades:
    - Permite as funcionalidades de gravar, ativar e eliminar um dado jogo da memória
- Colaborações: Precisa da maioria das classes da aplicação para conseguir gravar os dados.

Classe: Tecnlogias

- Responsabilidades:
    - Tem as funcionalidades de tecnologias, como a sua compra/ativação.
- c. Colaborações: é parte da classe Império, pois o Império é que tem tecnologias.

Funcionalidades implementadas

Componente do trabalho	Realizado	Realizado parcialmente	Não Realizado
carrega <nomeFicheiro>	✔		
cria <tipo> <n>	✔		
conquista <nome>	✔		
passa	✔		
maisouro	✔		
maisprod	✔		
maismilitar	✔		

adquire <tipo>	✔		
lista <nome>	✔		
avanca	✔		
grava <nome>	✔		
ativa <nome>	✔		
apaga <nome>	✔		
toma <qual> <nome>	✔		
modifica <ouro|prod> N	✔		
fevento <nome-evento>	✔		