Departamento de Engenharia Informática e de Sistemas

Licenciatura em Engenharia Informática

Licenciatura em Engenharia Informática - PL

Ramo de Desenvolvimento de Aplicações

Trabalho Prático – Primeira Meta

Programação Orientada a Objetos

2020/2021

Diogo Pires, 2018017890, P6

Rui Canas, 2018020744, P6

# Índice

[Índice 1](#_Toc57666524)

[Introdução 2](#_Toc57666525)

# Introdução

Através deste relatório, pretende-se descrever de forma sucinta as estratégias utilizadas para desenvolver a primeira meta deste trabalho prático.

# Classes consideradas na primeira versão da aplicação

Para a primeira meta do trabalho prático que foi proposto foram utilizadas as seguintes classes:

* **GameData –** classe que encapsula todas as classes envolvidas na lógica e vai receber a informação da interface. Vai ser responsável pela execução de ações que foram pedidas pelo utilizador.
* **World –** classe que vai conter os territórios disponíveis para conquista durante o jogo.
* **Territory –** classe que contém a informação detalhada de um território.
* **Empire –** contém toda a informação relativa a territórios conquistados. Também é composta pela força militar, armazém e cofre.
* **Army –** classe que contém toda a informação da força militar do império.
* **SafeBox –** classe que contém informação sobre o ouro que o império possui.
* **Storage –** classe que contém toda a informação sobre os produtos que estão sob a posse do império.
* **FileReader –** classe responsável por tratar da leitura de ficheiros bem como a informação que os mesmo possuem. Esta classe também já faz a filtragem de comandos inválidos que possam estar presentes nos ficheiros de texto.
* **Interface –** classe responsável pela parte gráfica. Através dela, o utilizador consegue interagir com o programa.

# Leitura do enunciado

Ao ler o enunciado, identificamos de imediato classes que seriam cruciais para o funcionamento do programa – **World** e **Empire.**

Também foi identificada uma classe que seria responsável pela **leitura do ficheiro**. O conceito de **conquistar** **territórios** também nos levou a pensar numa classe do tipo **território.**

Mais adiante, foi necessário desenvolver outras classes (cada uma com a sua importância para um bom funcionamento do programa) destacando-se as classes:

* **Interface –** responsável por fazer a interação com o utilizador. Vai fazer a leitura dos comandos inseridos pelo utilizador e assim, fazer com que sejam desencadeadas as ações na parte da lógica do programa – **GameData**.

# Duas principais classes da aplicação

As duas principais classes para o bom funcionamento do programa seriam as seguintes:

**GameData**

No início do programa, é imperativo que seja sempre criado um objeto do tipo **GameData**.



Figura 1 - Construção da classe GameData

Este objeto vai tratar de toda a lógica do jogo sendo assim armazenado na classe Interface para que seja possível a execução de comandos.

Assim que for feito o pedido para o términus do programa, está prevista a sua devida destruição.

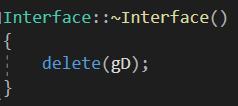


Figura 2 - Destruição da classe GameData

A destruição do **GameData** deverá ocorrer no destrutor da **Interface** uma vez que esta é a classe que vai estar a encapsular a classe referida inicialmente.

**Interface**

Classe que também é construída no início do programa. Necessita de receber como parâmetro um ponteiro do tipo **GameData.**



Figura 3 - Construção da classe interface

É através do parâmetro que recebe que vai conseguir fazer com que seja possível efetuar a execução dos comandos introduzidos pelo utilizador.

Caso esta classe não existisse, não seria possível separar a lógica da interface e o código tornar-se-ia confuso.

O destrutor desta classe também está a ser chamado, como se pode verificar na figura dois.

# Exemplo de uma responsabilidade de encapsulamento

A responsabilidade de “guardar territórios” está atribuído à classe **World**, que vai ter um vetor de ponteiro para **Territory**.

Em todas as classes criadas, é tido em atenção o *private* no ínicio que, apesar de ser feito por *default*, torna o código mais claro no que diz respeito ao encapsulamento de classes.

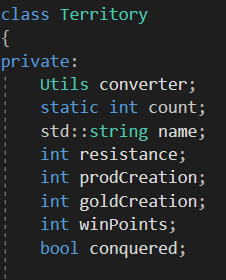


Figura 4 - Classe utilizada como exemplo

# Duas classes com objetivo focado, coeso e sem dispersão

O melhor exemplo para classes de objetivo focado será a classe **Interação** e a classe **FileReader**.

A classe **Interação** é única e exclusiva que age como uma interface de texto, ou seja, apenas recebe e mostra informação (*stdin*/*stdout*), pede a execução de certos comandos introduzidos pelo utilizador e o seu único propósito é somente esse. Assim, teremos código que se encontra organizado e irá tornar o programa mais fácil de manipular para futuras alterações.

A classe **FileReader** tem como único objetivo ler ficheiros de texto e devolver o resultado da sua leitura à classe que o está a utilizar. Para além do que foi descrito, esta classe também irá realizar uma filtragem de conteúdo do ficheiro pelo que só irá retornar conteúdo válido após a leitura de um ficheiro de texto.

Desta forma, temos duas classes diferentes, porém com objetivos específicos e que não dispersão em ocasião alguma.

# Interface vs. Lógica

A classe **Interface** é a classe que vai ser responsável pela interface, como já foi referido neste documento.

A classe **GameData** irá possuir toda a lógica do jogo.

# Primeiro objeto para além da camada de interação com o utilizador que recebe e coordena uma funcionalidade de natureza lógica

As ordens vindas da camada de interação com o utilizador são recebidas e processadas através de um objeto da classe **GameData.**