Licenciatura em Engenharia Informática
2019/2020
Programação Orientada a Objetos
Ramo Comum

Relatório
Simulador de corridas em C++
TRABALHO PRÁTICO
Meta 2

Turma Prática nº 1

Rui Filipe Tavares Mota – 2017012857
Ruben António de Jesus Marques - 2017016022

Coimbra, janeiro de 2020

Índice

1. Introdução

Este trabalho prático foi realizado no âmbito da disciplina de Programação Orientada a Objetos. O objetivo do trabalho era implementar um simulador de corridas em C++ onde os requisitos se encontram no enunciado dado. O trabalho foi divido em duas metas e este relatório refere-se à meta 1 e meta 2 ao mesmo tempo. A meta 1 do trabalho encontra-se completa e a meta 2 encontra-se quase toda implementada com exceção de algumas funcionalidades / requisitos mencionados no ponto 3. O relatório é divido em duas secções: organização do código e funcionalidades implementadas.

2. Organização do código apresentado

Como previsto no documento .pdf sobre a organização do relatório vamos responder às perguntas que são colocadas no que diz respeito à organização do código deste trabalho prático.

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

- UserInterface
- SimulatorLogic
- GeneralTravelOffice
- Pilot
- Car
- RaceTrack
- Championship
- Race

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

- GeneralTravelOffice
- Pilot
- CrazyDriver
- FastPilot
- SurprisePilot
- Car
- RaceTrack
- Championship
- Race

Estas foram as classes identificadas à primeira vista ao ler o enunciado.

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

Estas duas classes principais são a classe *SimulatorLogic* e *GeneralTravelOffice.*

Na *SimulatorLogic* são criados e destruídos objetos dinâmicos da classe *GeneralTravelOffice* que representam a Direção Geral de Viação consoante a execução dos comandos “loaddgv”, “savedgv” e “deldgv”.

Na *GeneralTravelOffice* são criados e destruídos objetos dinâmicos do tipo *Pilot, Car e RaceTrack* consoante a execução dos comandos “carregaP”, “carregaC”, “carregaA”, “criaObjeto”, “apaga” e “destroi”.

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

Um exemplo de uma responsabilidade atribuída a uma classe que esteja de acordo com a orientação dada acerca de Encapsulamento é a classe *GeneralTravelOffice*. Ela é responsável pela gestão dos objetos que vão ser usados para tornar as simulações possíveis (carros, pilotos e autódromos). É também esta classe que tem o dever de saber construir esses objetos e de os destruir.

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

Duas classes que são classes com objetivo focado, coeso e sem dispersão são as classes *Car* e *Pilot*.

A classe *Car* tem como objetivo representar um carro e encapsular a lógica de funcionamento do mesmo. Fornece um interface de utilização do para poder ser manipulado por um piloto. Esse interface consiste em funções que representam os pedais do carro para permitir a travagem e a aceleração do mesmo.

A classe *Pilot* tem como objetivo representar um piloto robô e encapsular o comportamento de cada tipo de robô. Tem como interface uma função que manda agir o piloto a cada segundo.

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

Classes com responsabilidades de interface com o utilizador:

- UserInterface
  - Command
  - ErrorException
  - Consola

Classes que representam a lógica

- SimulatorLogic
  - GeneralTravelOffice
    - Pilot
      - CrazyDriver
      - FastPilot
      - SurprisePilot
    - Car
    - RaceTrack
      - Race
        - RaceInfo
  - Campeonato
    - Score

2.7. Identifique o primeiro objeto para além da camada de interação com o utilizador que recebe e coordena uma funcionalidade de natureza lógica?

O primeiro objeto para além da camada de interação com o utilizador que coordena a lógica é um objeto da classe *SimulatorLogic*. Este objeto é criado na função main() e passado por referência para o construtor da classe *UserInterface*. Este será o único objeto coordenador da lógica que a *UserInterface* irá manipular.

2.8. 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 de toda a lógica (*SimulatorLogic*) delega a maior parte das funcionalidades noutras classes e serve maioritariamente de ponte entre a classe *UserInterface* e a restante lógica do simulador. Um exemplo concreto disto a acontecer é visto na execução do comando "passatempo" que vai ser descrito a seguir:

i. Introdução do comando "passatempo" por parte do utilizador

ii. *UserInterface* chama o método correspondente na *SimulatorLogic*

iii. *SimulatorLogic* invoca o método correspondente na classe *Campeonato*

iv. *Campeonato* manda o autódromo (classe *RaceTrack*) onde está a decorrer a corrida fazer passar o tempo.

v. *RaceTrack* manda a classe *Race* passar o tempo

vi. *Race* chama método no *Pilot* que manda o piloto agir

vii. *Pilot* chama métodos acelera() ou trava() consoante o seu tipo da classe *Car* para o carro se movimentar.

2.9. Dê um exemplo de uma funcionalidade que varia conforme o tipo do objeto que a invoca. Indique em que classes e métodos está implementada esta funcionalidade.

Um exemplo e caso único onde isto acontece é na classe *Pilot*. A classe *Pilot* tem 3 classes que derivam dela (*CrazyDriver, FastPilot, SurprisePilot).* Dependendo do tipo do objeto guardado numa referência ou ponteiro *Pilot*, serão executadas diferentes ações no método act().

2.10. Apresente as principais classes da aplicação através da seguinte informação:

1. **Classe**: UserInterface

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar todas as informações relativas à simulação de forma muito encapsulada e abstraida da lógica.

  O que permite fazer?

  - Permite manipular a simulação invocando os métodos abstratos da *SimulatorLogic* de forma muito encapsulada.

- **Colaborações**:

  Colabora com a classe *Command* que usa como auxiliar na hora de verificar commandos e com a classe *SimulatorLogic* para manipular o simulador.

2. **Classe**: SimulatorLogic

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar todas as informações da DGV (carros, pilotos e autódromos) e do campeonato (corrida, autódromo atual e scores) a decorrer no momento.

  O que permite fazer?

  - Permite a manipulação da DGV através de pedidos de criação, edição e eliminação de objetos.
  - Permite manipular o campeonato através de pedidos para passar o tempo, mandar parar pilotos, acidentar carros e etc.

- **Colaborações**:

  Colabora com as classes *GeneralTravelOffice e Championshit* a nível de composição porque constrói e destrói objetos das mesmas.

3. **Classe**: GeneralTravelOffice

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar todas as informações dos carros, pilotos e autódromos

  O que permite fazer?

  - Permite criar, eliminar e editar pilotos, carros e autódromos.

- **Colaborações**:

  Colabora a nível de composição com as classes *Pilot, Car* e *RaceTrack.* É responsável pelo ciclo de vida dos objetos dessas classes

4. **Classe**: Championship

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar informações sobre a corrida e o autódromo onde decorre essa corrida.

  O que permite fazer?

  - Permite iniciar novas corridas e mandar o tempo passar nas corridas.
  - Permite também mandar parar pilotos e acidentar carros.

- **Colaborações**:

  Colabora com as classes *Pilot, Car e Racetrack* a nível de agregação.

5. **Classe**: RaceTrack

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar informações do autódromo e informações da corrida a decorrer no autódromo.

  O que permite fazer?

  - Permite manipular a corrida a decorrer no autódromo, por exemplo mandar passar o tempo.

- **Colaborações**:

  Colabora apenas com a classe *Race* a nível de composição. É responsável por criar e destruir objeto *Race.* Colabora com as classes *Pilot e Car* a nível de agregação.

6. **Classe**: Race

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar informações da corrida tais como o mapa da pista, informações dos corredores.

  O que permite fazer?

  - Permite passar o tempo e mandar os pilotos agir em conformidade

- **Colaborações**:

  Colabora com a classe *Pilot* a nível de agregação.

7. **Classe**: Pilot super de CrazyDriver, FastPilot, SurprisePilot

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar toda a informação relativa a um piloto (carro onde está, nome, etc.)

  O que permite fazer?

  - Permite que mande agir o piloto conforme o seu tipo e mandar o piloto entrar ou sair de um carro.

- **Colaborações**:

  Colabora apenas com a classe *Car* a nível de agregação.

8. **Classe**: Car

- **Responsabilidades**:

  O que permite consultar?

  - Permite consultar toda a informação relativa a um carro.

  O que permite fazer?

  - Permite meter o carro em aceleração, travagem ou acidentá-lo.

- **Colaborações**:

  Nenhuma

3. Funcionalidades implementadas

Neste programa, a maior parte dos requisitos enunciados estão implementados e a funcionar. Sendo assim, nesta tabela em baixo estao apresentados os requisitos / funcionalidades que não funcionam ou não foram corretamente implementadas:

Garagem	Não implementada
Acidentar carro atrás do CrazyDriver quando este se despista	Não implementada
Scores	Implementado, mas não funcionam como esperado
Autódromos com nomes únicos	Não implementado