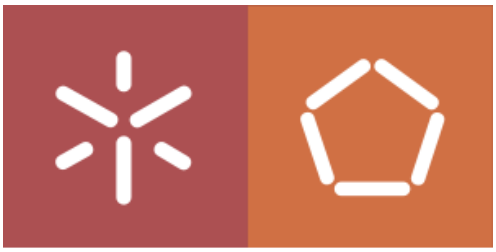


Universidade do Minho
Escola de Engenharia

Desenvolvimento de Sistemas de Software

Trabalho Prático - Fase 1

Grupo 22

2022/2023

A93258 - Bernardo Garcia de Freitas Lima

A91671 - João Manuel Novais da Silva

A91660 - Pedro António Pires Correia Leite Sequeira

A91697 - Luís Filipe Fernandes Vilas

A91677 - Vicente de Carvalho Castro

Conteúdo

Introdução

Neste relatório vai ser apresentada a elaboração do modelo conceptual e os diagramas que irão suportar os Use Cases definidos na fase anterior, assim como o raciocínio dos alunos na realização da fase 2 do trabalho prático de Desenvolvimento de Sistemas de Software. Com o objetivo da criação de uma forte fundação necessária para o desenvolvimento do código para a fase 3 do trabalho prático foram pensados e elaborados os vários diagramas indispensáveis para a realização coerente não só deste trabalho prático, mas também qualquer projeto a desenvolver, posteriormente, na vida profissional dos alunos.

Levantamento de Requisitos

Após uma análise detalhada dos use cases realizados na fase anterior do projeto, com base nos cenários de utilização facultados pelos docentes, foram definidos os seguintes requisitos para a boa funcionalidade da aplicação a ser desenvolvida:

- Verificar se o utilizador as credenciais dum utilizador (Administrador ou Convidado);
- Adicionar um novo utilizador;
- Mostrar ranking gerais (número de campeonatos ganhos por cada jogador);
- Criar campeonatos;
- Criar circuitos;
- Criar carros;
- Criar pilotos;
- Adicionar circuitos aos campeonatos;
- Configurar campeonato;
- Adicionar carros associados a cada jogador ao campeonato;
- Adicionar pilotos associados a cada jogador ao campeonato;
- Simular campeonato;
- Simular corridas;
- Atualizar as classificações de corrida em corrida e de volta em volta;

- Alterar o clima no início de cada corrida;
- Fazer afinações;
- Remover utilizador;
- Remover carro;

Definição de métodos em função dos requisitos

Com os requisitos definidos, podemos agora começar por pensar em fazer um inicial rascunho dos métodos necessários e, posteriormente, a relação que irão ter entre si as classes a que vão pertencer.

(REVER DPS OS CRIA CONSUANTE O Q ELES FIZEREM)

1. verificaCredenciais (user : Utilizador) : void;
2. adicionarUtilizador (user : Utilizador) : void;
3. printRanking () : void;
4. criarCampeonato (camp : Campeonato) : void;
5. criarCircuito (nome : String, voltas : Integer, caminho : List<String>, dificuldadeCaminho : List<Double>, nafinações : Integer) : void;
6. criarCarro () : void;
7. criarPiloto () : void;
8. adicionaCircuito (corrida : Corrida);
9. adicionarCampeonato (camp : Campeonato) : void;
10. adicionarCarro (carro : Carro) : void;
11. adicionarPiloto (piloto : Piloto) : void;
12. simularCampeonato () : void;
13. simularCorriada () : void;
14. atualizaClassificação () : void;
15. setClima(chove : Boolean) : void;
16. afinações () : void;
17. removerUtilizador (user : Utilizador) : void;
18. removerCarro (carro : Carro) : void.

Agora que temos uma ideia dos métodos necessários para o sistema, podemos começar por encaixar estes nas classes respetivas, estruturando, assim a aplicação.

Após esta análise feita o grupo decidiu estruturar as classes numa maneira quase que hierárquica. Será necessária uma interface que estará ligada apenas ao Facade, e este irá por sua ver servir de eixo central para as comunicações para as classes dos objetos que devem ser criados. Para representar estes objetos serão criadas as classes Campeonato, Circuito, Corrida, Piloto, Carro e Utilizador. Estes serão divididos em packages, um package SSCampeonato irá possuir a classes Campeonato, Corrida, Circuito, Piloto e Carro em conjunto com as subclasses que representam todos os tipos de carro e o package SSUtilizador retém a classe Utilizador e as subclasses admin e jogador.

Métodos Categorizados por Classes

Vamos agora organizar os métodos definidos anteriormente pelas respetivas classes.

1. verificaCredenciais (user : Utilizador) : void → RacingManagerFacade
2. adicionarUtilizador (user : Utilizador) : void → Utilizador
3. printRanking () : void → IRacingManager <<Interface>>
4. criarCampeonato (camp : Campeonato) : void → IRacingManager <<Interface>>
5. criarCircuito (nome : String, voltas : Integer, caminho : List<String>, dificuldadeCaminho : List<Double>, nafinações : Integer) : void → IRacingManager <<Interface>>
6. criarCarro () : void → IRacingManager <<Interface>>
7. criarPiloto () : void → IRacingManager <<Interface>>
8. adicionarCircuito (corrida : Corrida) → Circuito
9. adicionarCampeonato (camp : Campeonato) : void → Campeonato
10. adicionarCarro (carro : Carro) : void → Carro
11. adicionarPiloto (piloto : Piloto) : void → Piloto
12. simularCampeonato () : void → IRacingManager <<Interface>>
13. simularCorriada () : void → Corrida
14. ????
15. setClima(chove : Boolean) : void → Corrida

16. afinações () : void → IRacingManager <<Interface>>
17. removerUtilizador (user : Utilizador) : void → Utilizador
18. removerCarro (carro : Carro) : void → Carro

Estrutura dos Packages

Como mencionado em cima, vamos organizar as classes definidas em packages da seguinte forma:

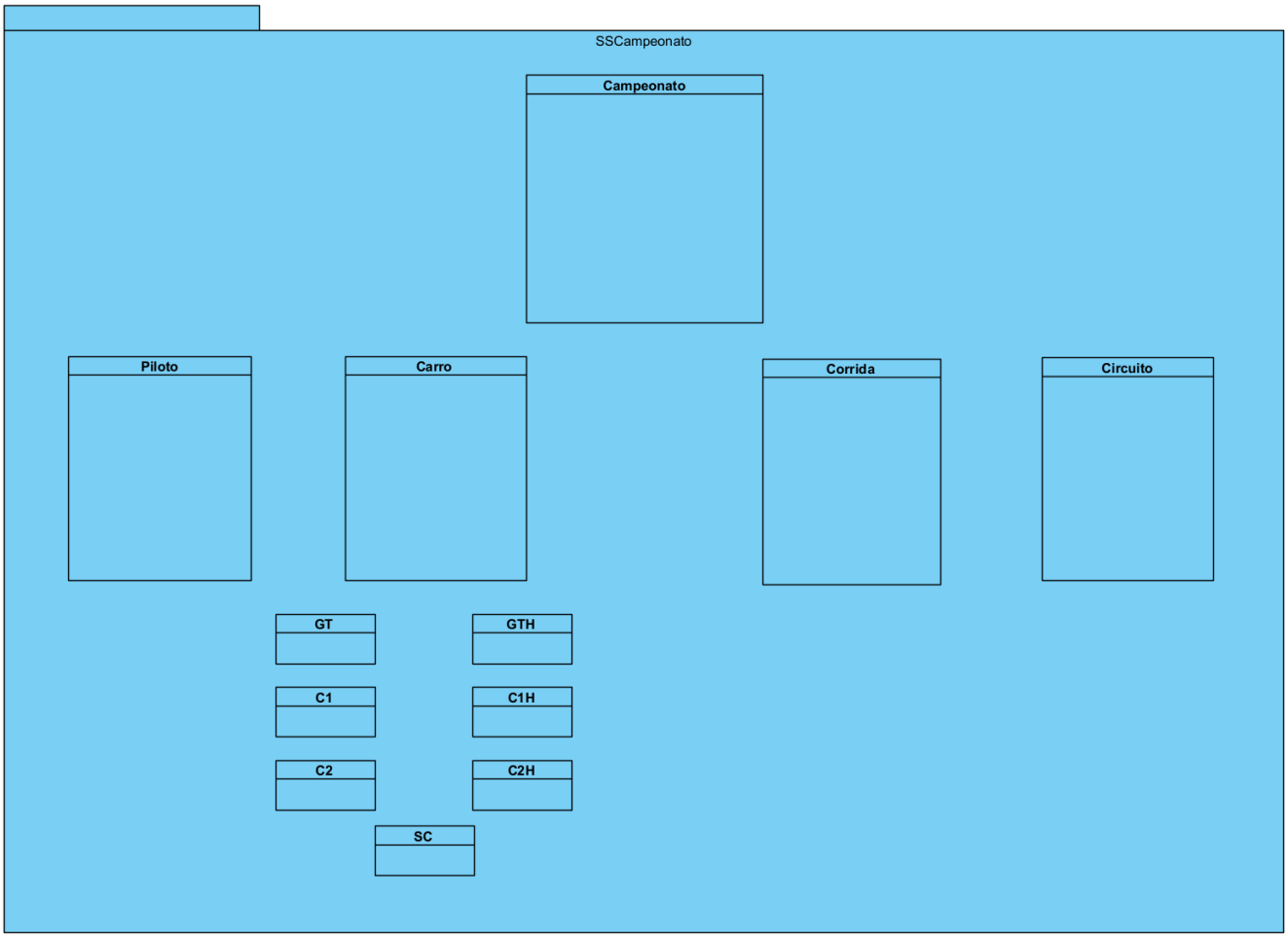


Figura 1- Package SSCampeonato

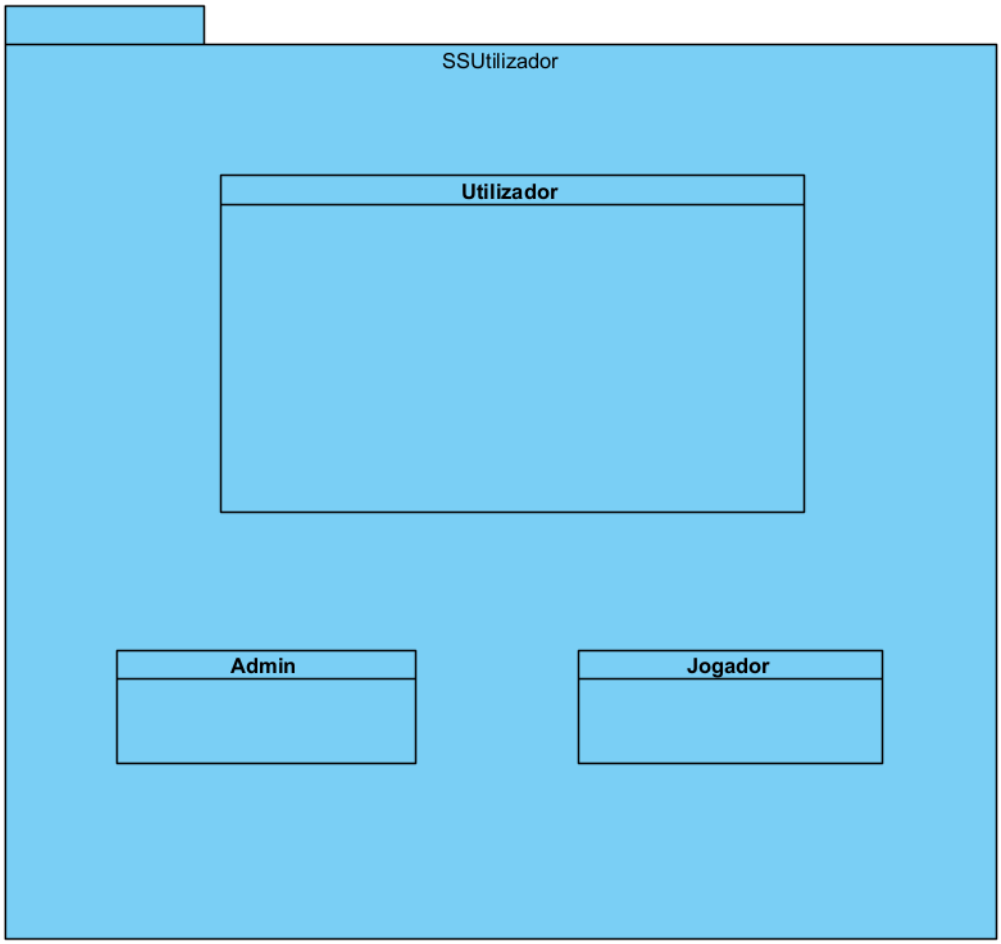


Figura 2 - Package SSUtilizador

Diagrama de Componentes

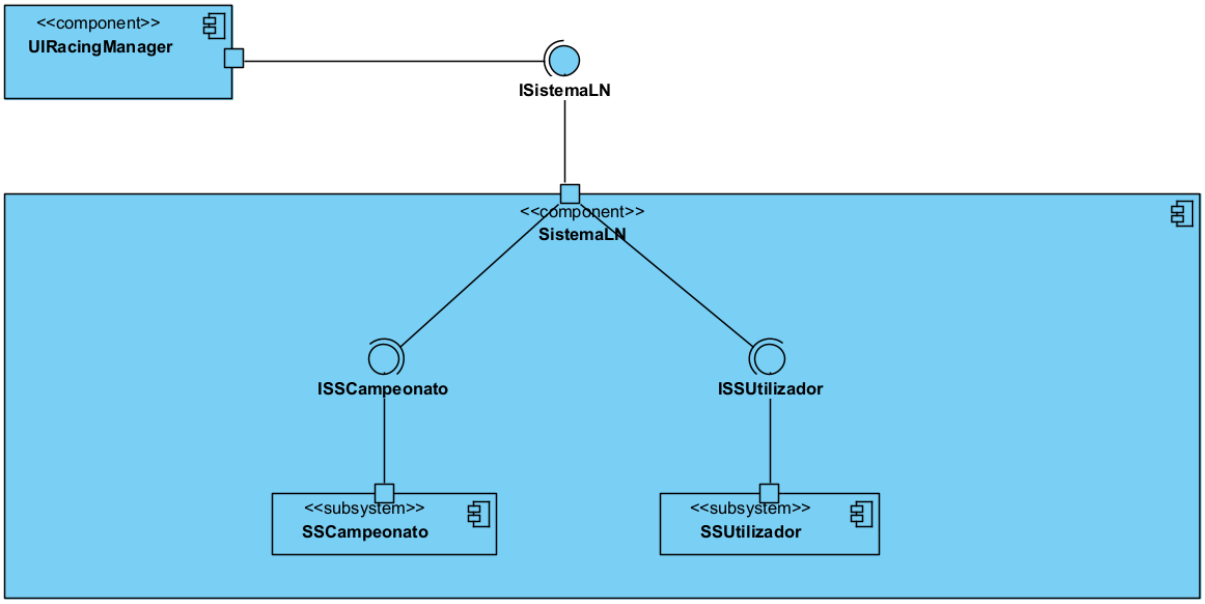


Figura 3 - Diagrama de Componentes

A implementação e formulação dos packages foram feita a pensar na modulação do código, sendo pertinente representar os subsistemas a partir dum diagrama de componentes. Os blocos que combinam para formar o sistema são o SSCampeonato e o SSUtilizador. Esta prática é essencial para obter um encapsulamento do código uma vez que cada subsistema implementa uma interface com os métodos que lhes foram atribuídos.

Inicialmente o grupo tinha pensado em 3 subsistemas, tendo esta extra a classe carro e os seus derivados, mas foi rapidamente concluído que esta arquitetura exigia que o carro, sempre que necessitasse de comunicar com o campeonato (e vice-versa), tivesse de passar pela facade primeiro, fazendo “viagens” desnecessárias.

Com isto, chegamos á conclusão que é apenas necessária a implementação de 2 subsistemas, o subsistema SSCampeonato que implementa a interface ISSCampeonato e o subsistema SSUtilizador que implementa a interface ISSUtilizador. O UIRacingManager aceda às funcionalidades do sistema a partir da interface ISistemaLN que implementa as interfaces anteriores.

Diagrama de Classes

Após o trabalho feito anteriormente será mais acessível fazer o diagrama de classes, uma vez que já temos ideia de que classes serão necessária, assim como os packages. Como já mencionado na Definição de Métodos vamos representar as classes Campeonato, Circuito, Corrida, Piloto, Carro e Utilizador e as subclasses de carro correspondentes aos tipos de carros e de utilizador, admin e jogador. Estas classes serão divididas nos packages SSCampeonato que possuí tudo menos a classe Utilizador e suas derivadas cujas estão inseridas no package SSUtilizador.

Começamos pela classe Campeonato, esta representa, por parte, a ligação de todas as outras, ou seja, todas as classes existem para permitirem a simulação de campeonatos. Esta classe vai necessitar duma lista de circuitos, um map que contém os ids dos utilizadores como keys e os ids dos carros selecionados por estes, outro mapa que, duma maneira parecida, contém como keys os ids dos utilizadores e como values os ids dos pilotos escolhidos pelos jogadores, além destes também será necessária uma variável que guarde o número de afinações possíveis e finalmente o nome do campeonato. Para além destas variáveis a classe campeonato vai ter, mais do que os métodos usuais, os métodos para permitir a configuração do próprio campeonato adicionar pilotos, corridas, jogadores. Também é preciso de ter em conta as classificações dos jogadores durante os campeonatos, por isso será pertinente implementarmos métodos que os apresentam.

A classe circuito tem como variáveis de classe o nome do circuito, o número de voltas do circuito, uma lista caminho que tem a descrição dos elementos do circuito (curvas, chicanes, retas), uma lista de dificuldadesCaminho que tem como elementos doubles que descrevem a dificuldade de cada elemento do circuito e por fim a variável clima que indica se está a chuver ou não no circuito. Por fim, para além dos métodos usuais estão também definidos os métodos adicionarCircuito e removerCircuito, que alteram a estrutura de dados que tem guardados os circuitos.

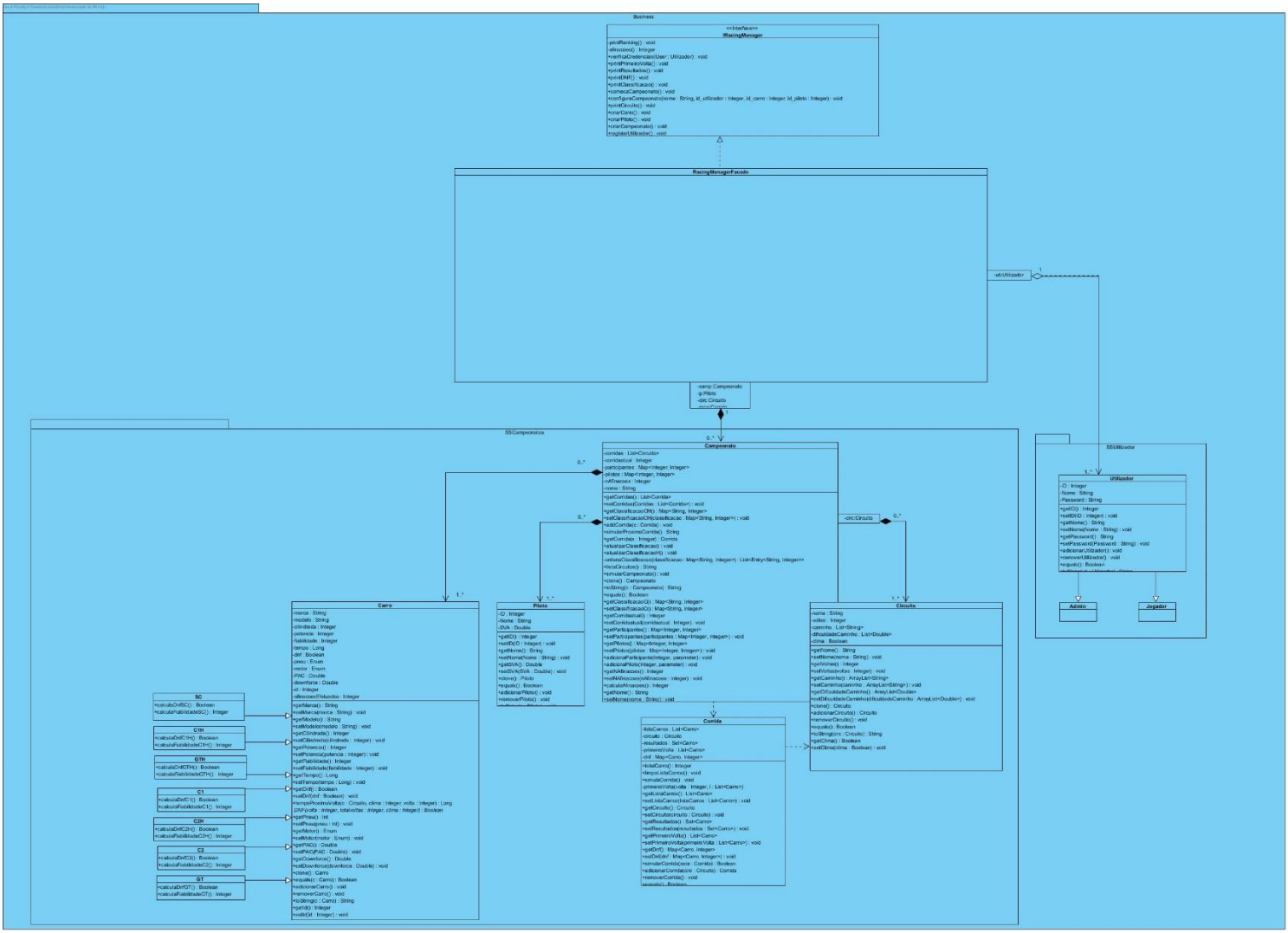
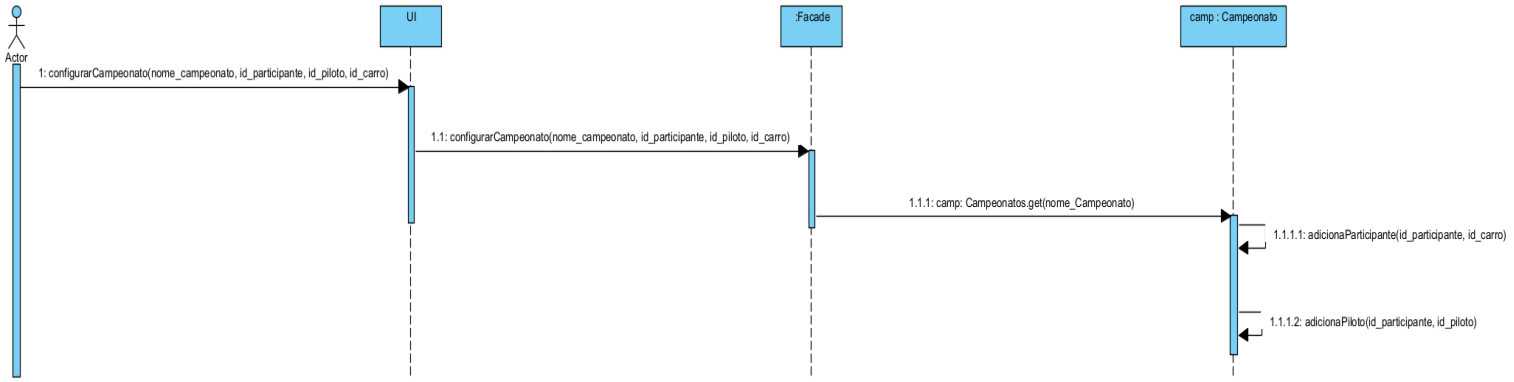


Figura 4 - Diagrama de Classes

Diagrama de Sequência



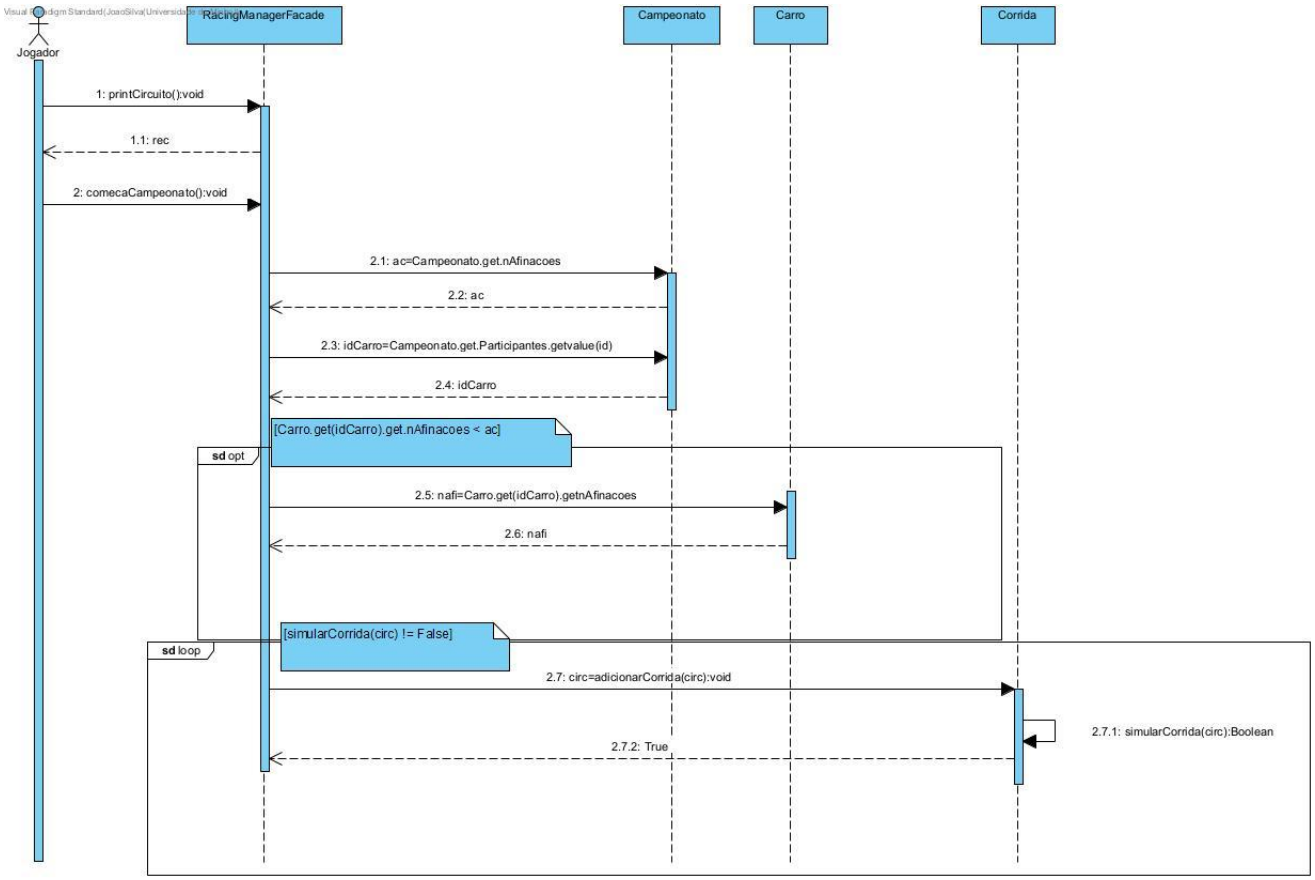


Figura 6 - Diagrama de sequência 2

Conclusão

O trabalho realizado permitiu elaborar os diagramas necessários para, na próxima fase, a elaborarmos do código.

Github: LuisFilipe6/DSS2223 (github.com)