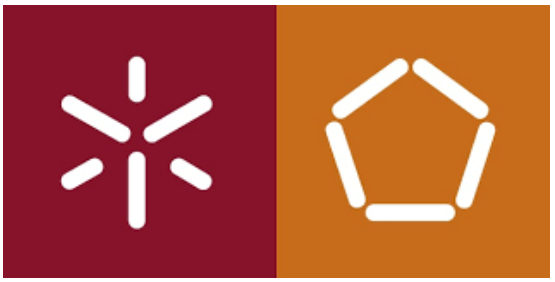


UNIVERSIDADE DO MINHO

DESENVOLVIMENTO DE SISTEMAS DE SOFTWARE

LICENCIATURA EM ENGENHARIA INFORMÁTICA

ENTREGA INTERMÉDIA 2

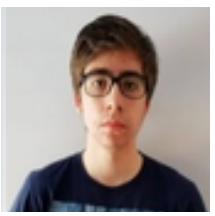
GRUPO 30

Ana Rita Poças
(a97284)

João Pedro Braga
(a97368)

Miguel Pinto
(a96106)

Orlando Palmeira
(a97755)



Pedro Miguel Martins
(a97613)

URL do repositório:
https://github.com/orlandopalmeira/Trabalho-DSS-2022-2023

1. Introdução

Este relatório surge no âmbito da Unidade Curricular de Desenvolvimento de Sistemas de Software, na qual nos foi proposto o desenvolvimento de um sistema que permita simular campeonatos de automobilismo, algo similar a um F1 Manager.

Nesta segunda fase, pretendemos chegar à concepção da solução. Assim, vamos elaborar uma arquitetura conceptual do sistema, capaz de suportar os requisitos identificados na primeira parte. Desenvolveremos, também, os modelos comportamentais necessários para descrever o comportamento pretendido para o sistema.

Para atingirmos estes objetivos, recorremos à concepção de um **Diagrama de Componentes**, **Diagramas de Classes** e **Diagramas de Sequência**.

2. Diagrama de Componentes

O diagrama de componentes tem como objetivo demonstrar o relacionamento existente entre diferentes componentes do sistema e, assim, representar de forma física os componentes de software do nosso sistema. O termo “componente” refere-se aos subsistemas do sistema a desenvolver.

A elaboração de um diagrama de componentes é essencial durante a fase de concepção da solução, uma vez que através dele somos capazes de ter uma visão da estrutura física da arquitetura do sistema.

De forma a termos uma melhor visão dos requisitos associados ao nosso programa, decidimos desenvolver uma folha de cálculo para o processamento de Use Cases :
https://docs.google.com/spreadsheets/d/15se9dV0aw_zHmRDLZ71AlSe6qFxrNQGQOubi_biQqNE/edit?usp=sharing

Assim sendo, projetamos o seguinte diagrama:

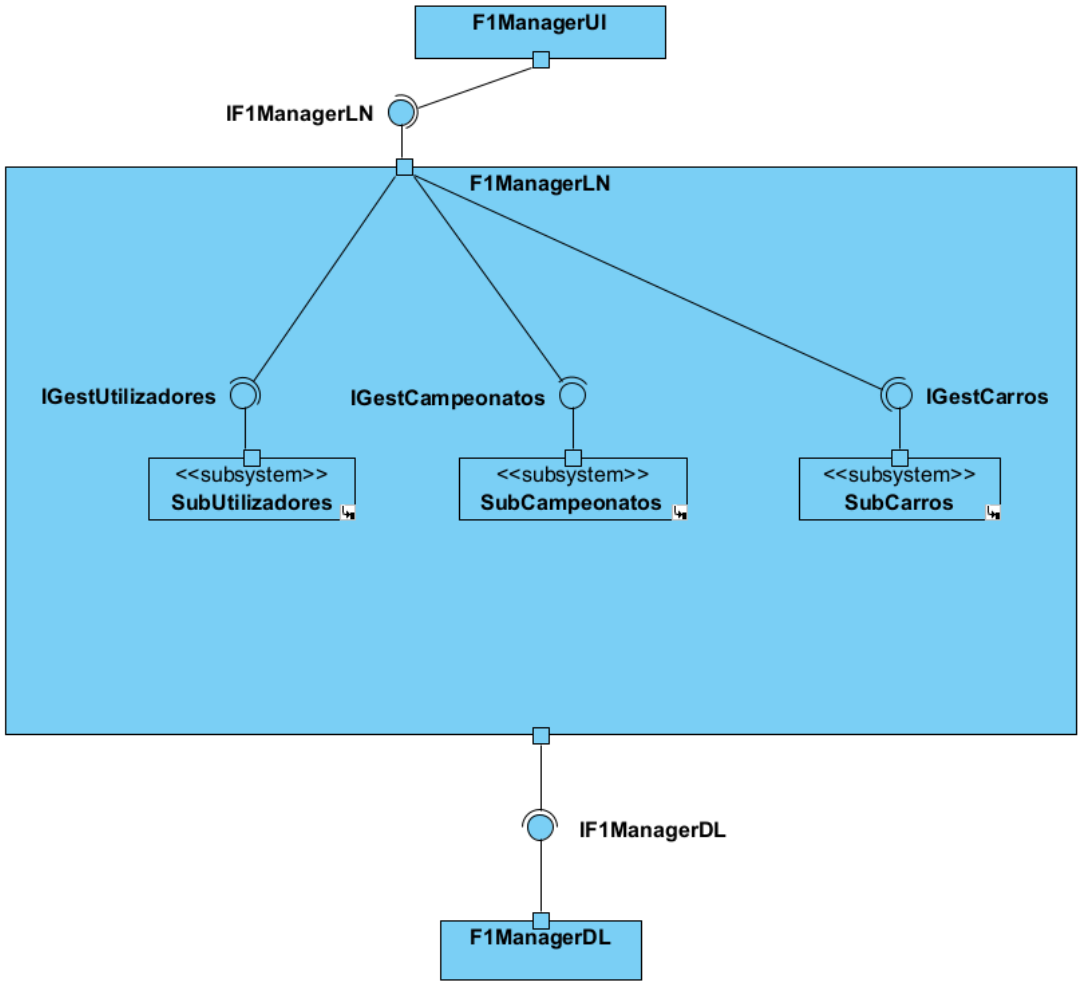


Figura 1 - Diagrama de Componentes

3. Diagramas de Classes

O diagrama de classes tem como objetivo descrever as classes de um sistema e os relacionamentos existentes entre elas. Permite, também, ilustrar a estrutura do sistema, através das operações e atributos das classes.

Desta forma, concebemos os seguintes diagramas de classes para cada subsistema do programa:

3.1. Diagrama de Classes SSUtilizadores:

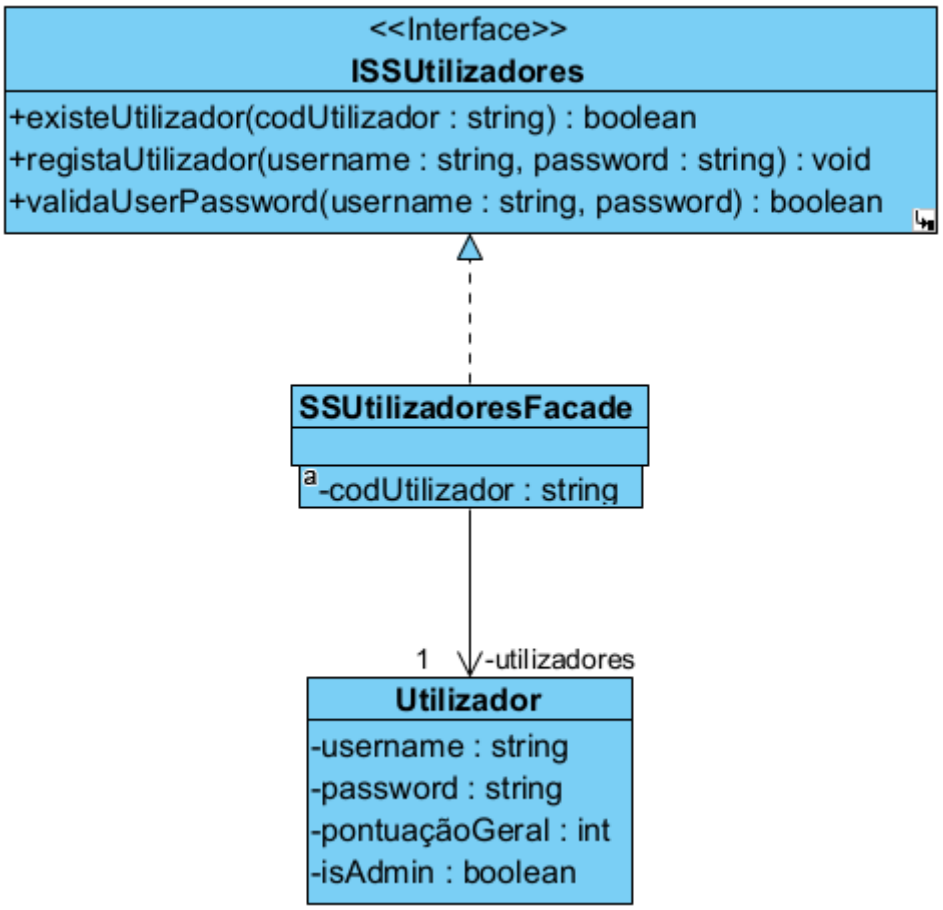


Figura 2 - Diagrama de Classes (SSUtilizadores).

Este diagrama de classes corresponde ao subsistema de utilizadores e servirá como gestor de todos os utilizadores do sistema (Jogadores e Administradores).

O subsistema possui um Facade que contém um Map de todos os utilizadores do programa.

3.2. Diagrama de Classes SSCarros:

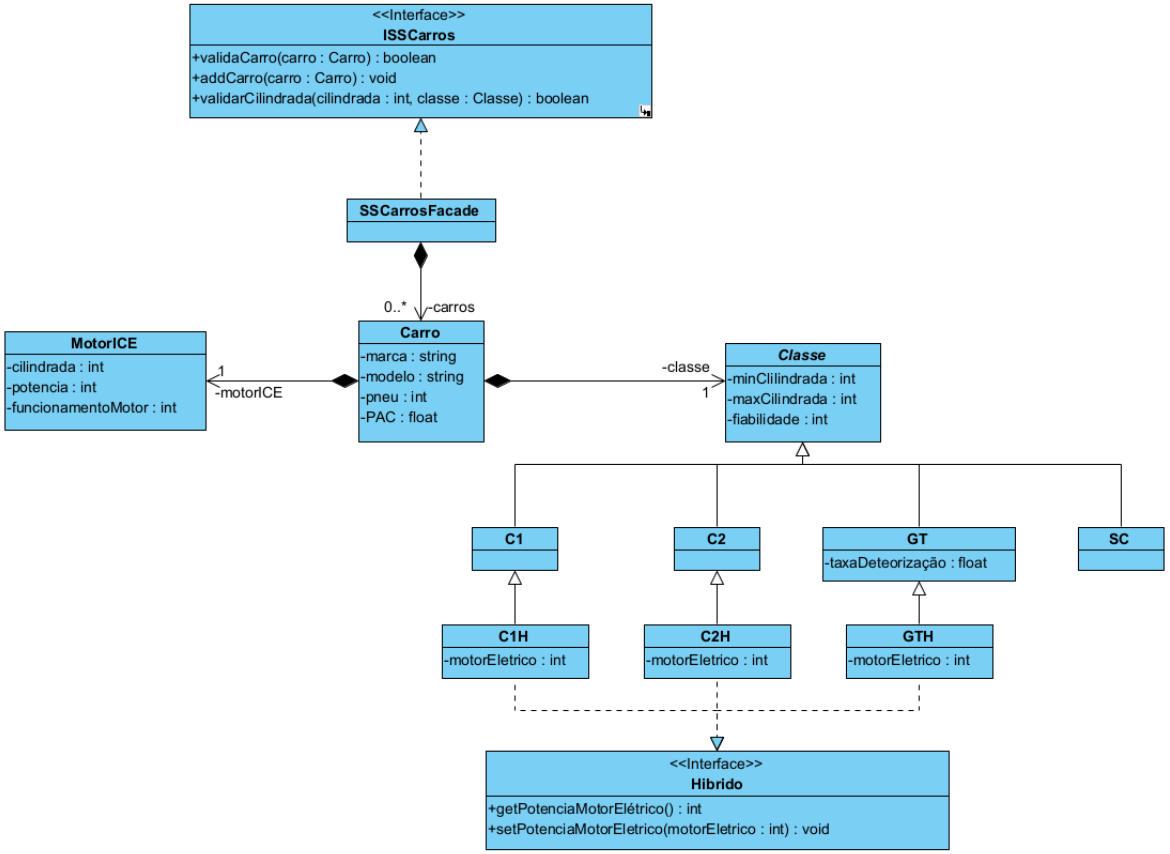


Figura 3 - Diagrama de Classes (SSCarros).

Este diagrama de classes corresponde ao subsistema de carros e servirá como gestor de todos os carros do sistema.

O subsistema possui um Facade que contém uma lista de todos os carros armazenados no sistema. Esses carros possuem dados básicos como **marca**, **modelo**, **pneu** e **PAC**. Todos os carros terão um **MotorICE** que irá armazenar os dados relativos à performance do carro nas corridas. Cada carro irá pertencer a uma **Classe** que determina várias particularidades do carro como:

Mínimo e Máximo de cilindrada do **MotorICE**, o valor da fiabilidade do carro e a possibilidade do carro possuir um motor **Híbrido** ou não. No caso do carro pertencer a uma classe híbrida essa classe terá de armazenar o valor da potência do **motorEletrico**.

3.3. Diagrama de Classes SSCampeonatos:

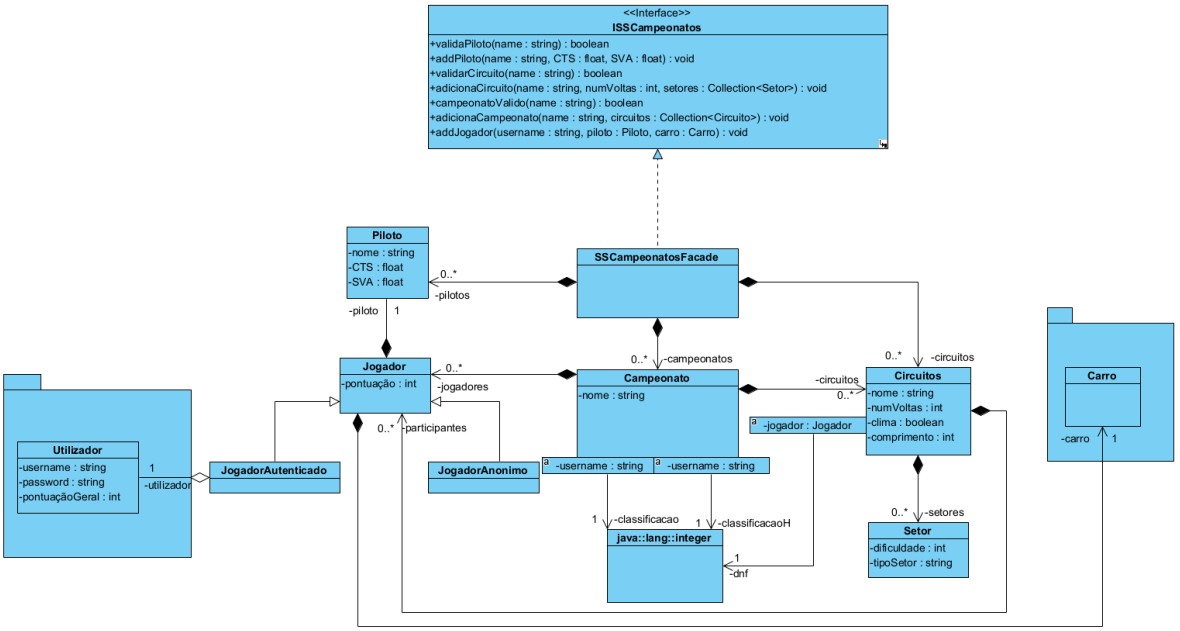


Figura 4 - Diagrama de Classes (SSCampeonatos).

Este diagrama de classes corresponde ao subsistema de campeonatos que é o subsistema principal do programa.

O subsistema possui um Facade que contém uma lista de todos os **campeonatos** disponíveis para jogar, todos os **circuitos** e todos os **pilotos** disponíveis no programa.

Os **campeonatos** são descritos por um **nome** e contêm:

1 lista de **circuitos** que serão jogados.

1 lista de **jogadores** que irão participar no campeonato.

1 map com a pontuação dos jogadores com carros “normais”.

1 map com a pontuação dos jogadores com carros “híbridos”.

Os **circuitos** são descritos por um **nome**, possuem um número fixo de **voltas**, são afetados por um **clima** e têm um **comprimento**.

Cada circuito contém também uma lista de **setores** onde a simulação das ultrapassagens irá ocorrer e uma lista de todos os **jogadores** participantes da corrida.

Existem 3 tipos de setores **reta**, **curva** e **chicane** e cada setor terá uma **dificuldade de ultrapassagem** que afeta o número de ultrapassagens que irão ocorrer naquele setor.

O circuito também guardará um map com os jogadores que não conseguiram terminar a corrida (**DNF**) devido a uma falha mecânica ou erro do piloto e a volta onde ocorreu o incidente. Os jogadores que não conseguiram terminar a corrida não receberão pontuação para essa corrida.

Os **pilotos** são descritos por um **nome** e são caracterizados por um valor de **CTS** (valor entre 0 e 1) que descreve o seu desempenho dependendo do clima (0 será melhor em clima de chuva e 1 será melhor em tempo seco) e um valor de **SVA** (valor entre 0 e 1) que descreve a sua agressividade em pista (0 o piloto é muito seguro e por isso não consegue ultrapassar tão frequentemente e 1 o piloto ultrapassa mais frequentemente mas terá maior probabilidade de se despistar).

Os **jogadores** são divididos em 2 tipos **JogadorAutenticado** e **JogadorAnonimo**.

O **JogadorAutenticado** é um tipo de jogador que está associado a um **Utilizador** e a pontuação obtida no campeonato será adicionada à pontuação total do **Utilizador** enquanto que um **JogadorAnonimo** não estará autenticado logo a pontuação obtida no campeonato não ficará armazenada no sistema.

O **Jogador** também estará associado a um **Piloto** e **Carro** que irá utilizar em todas as corridas do campeonato.

4. Diagramas de Sequência

Os diagramas de sequência têm como objetivo focar no ordenamento temporal da troca de mensagens, permitindo visualizar como é que os objetos comunicam entre si.

4.1. existeUtilizador

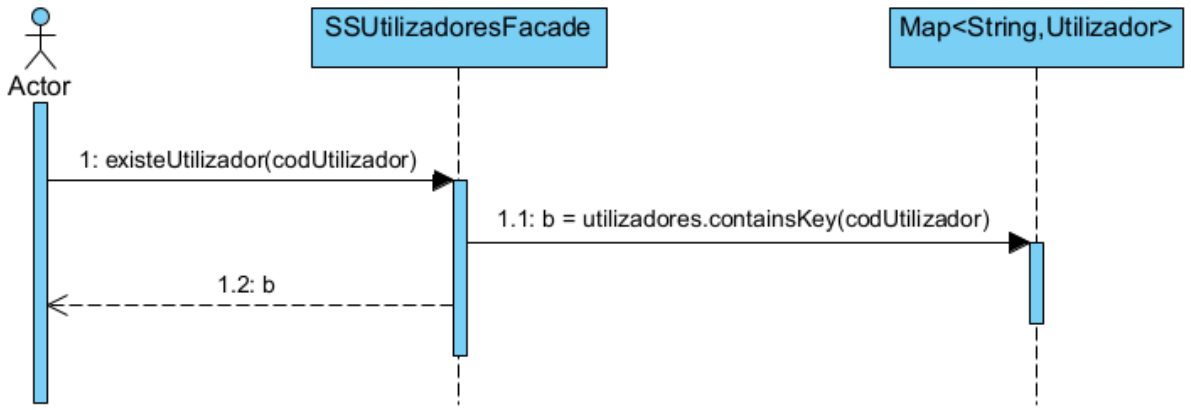


Figura 5 - Diagrama de Sequência "existeUtilizador".

Este diagrama de sequência corresponde ao método existeUtilizador que verifica a existência de um utilizador com o username recebido como argumento. Retorna true caso o utilizador exista e false quando não existe.

4.2. registaUtilizador

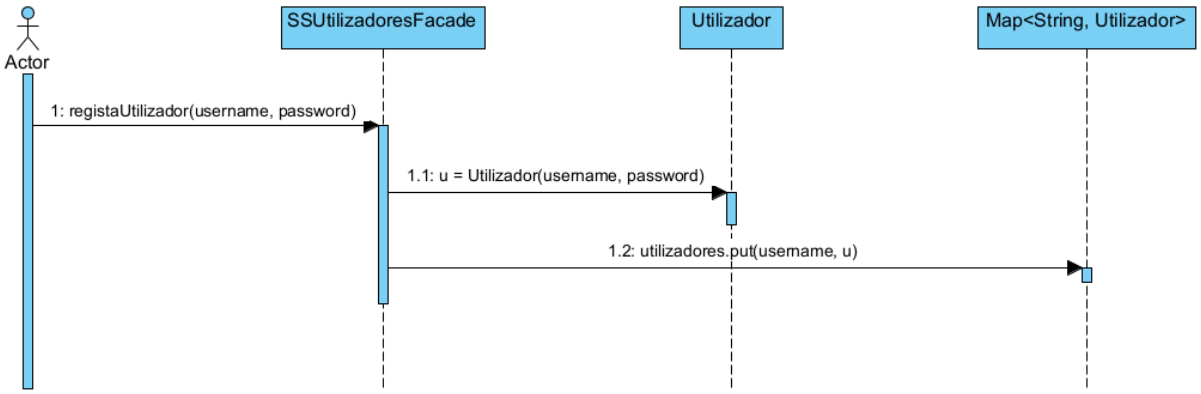


Figura 6 - Diagrama de Sequência "registaUtilizador".

Este diagrama de sequência corresponde ao método registaUtilizador que recebe o username e password de um novo utilizador e armazena-o no sistema

4.3. validaUserPassword

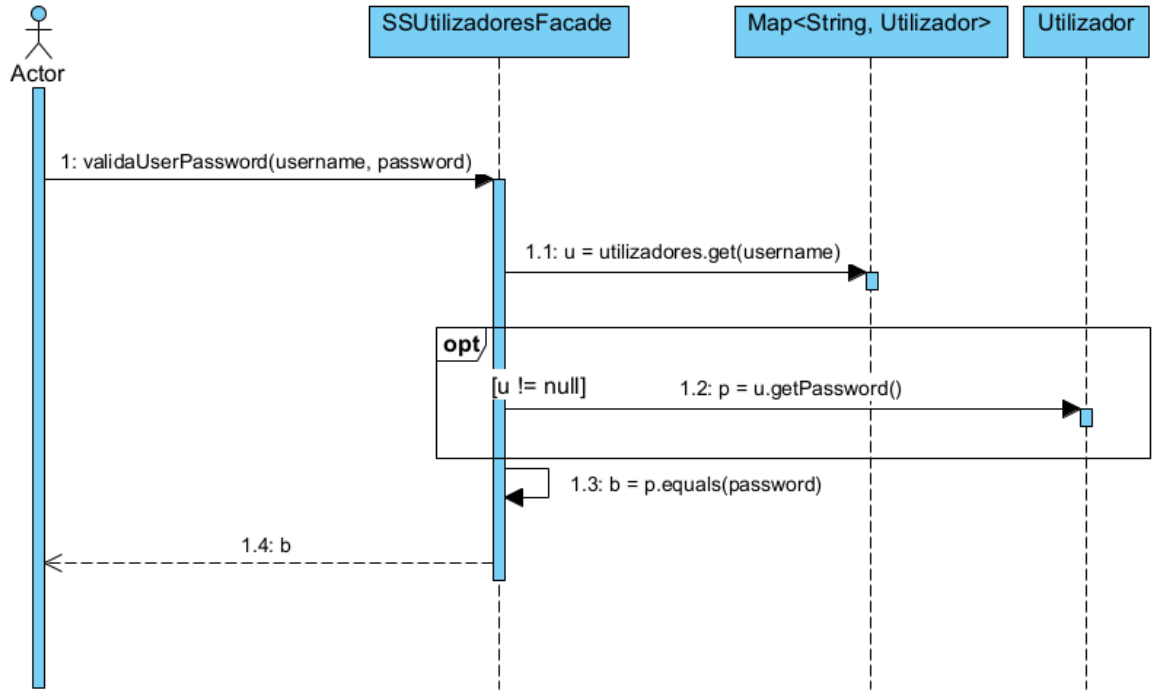


Figura 7 - Diagrama de Sequência "validaUserPassword".

Este diagrama de sequência corresponde ao método validaUserPassword que recebe o username e password de um utilizador e verifica se existe um utilizador com esses dados. Se existir o método devolve true autenticando o utilizador com essa conta senão devolve false e o utilizador não fica autenticado.

4.4. validaCarro

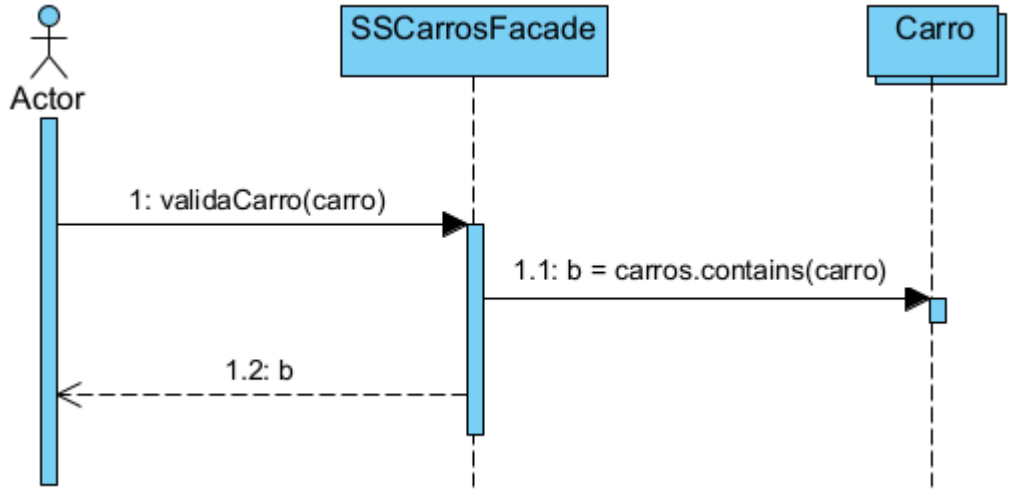


Figura 8 - Diagrama de Sequência "validaCarro".

Este diagrama de sequência corresponde ao método validaCarro que recebe um objeto Carro e verifica se já existe um objeto igual na lista de carros.

Se existir o método devolve true e o carro não deverá ser adicionado à lista de carros.

4.5. addCarro

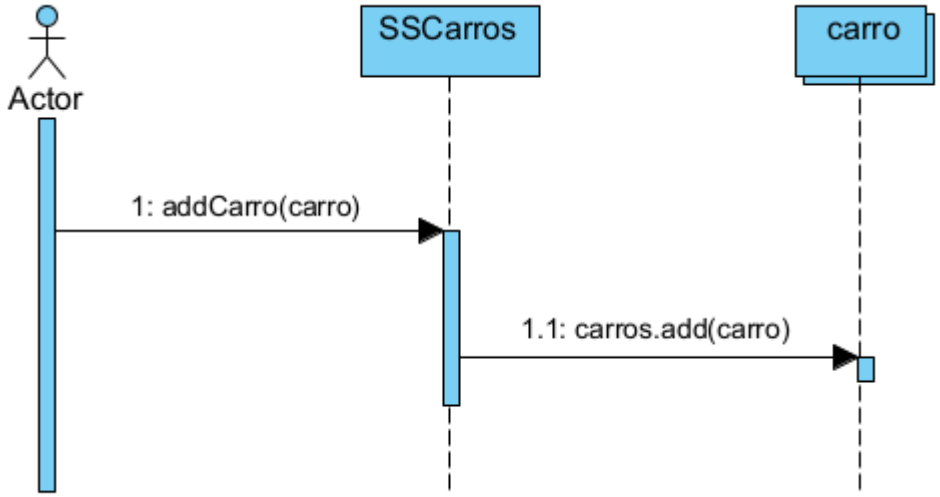


Figura 9 - Diagrama de Sequência "addCarro".

Este diagrama de sequência corresponde ao método addCarro que recebe um objeto Carro e adiciona-o à lista de carros do sistema.

4.6. validaCilindrada

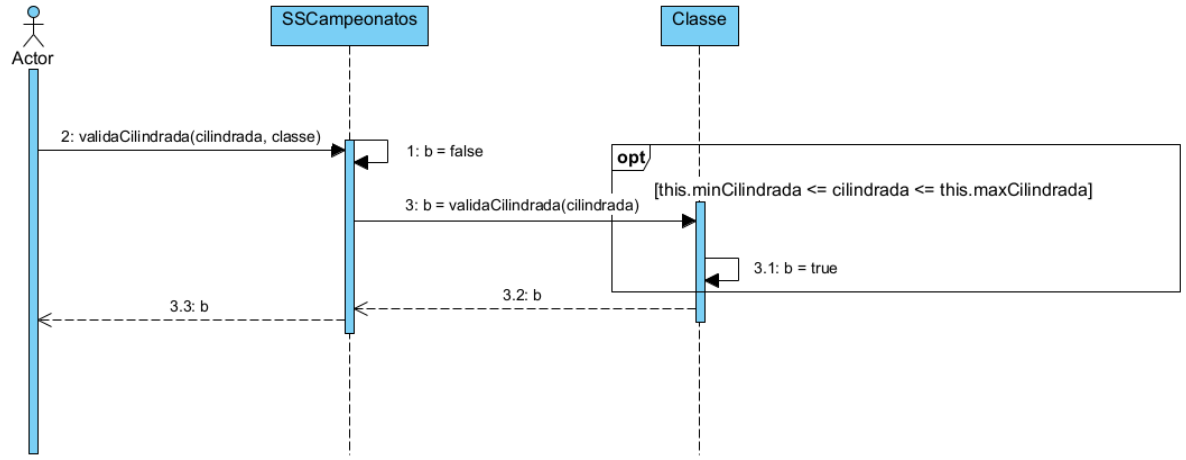


Figura 10 - Diagrama de Sequência "validaCilindrada".

Este diagrama de sequência corresponde ao método validaCilindrada que recebe um valor inteiro e uma classe e verifica se o valor da cilindrada recebido está entre os valores mínimos e máximos definidos.

4.7. validaPiloto

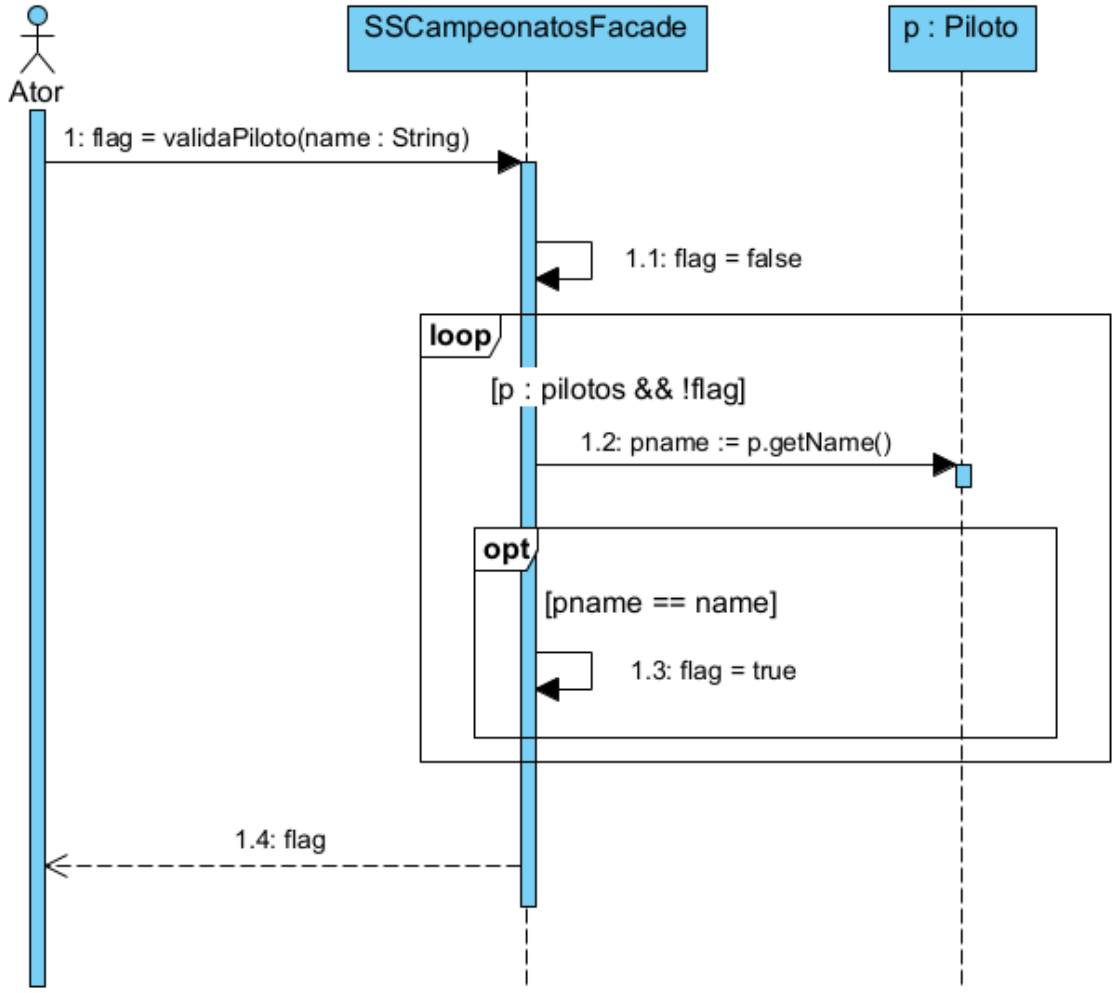


Figura 11 - Diagrama de Sequência "validaPiloto".

Este diagrama de sequência corresponde ao método validaPiloto que recebe uma string com o nome do piloto e verifica se esse piloto já existe nos pilotos existentes.

4.8. addPiloto

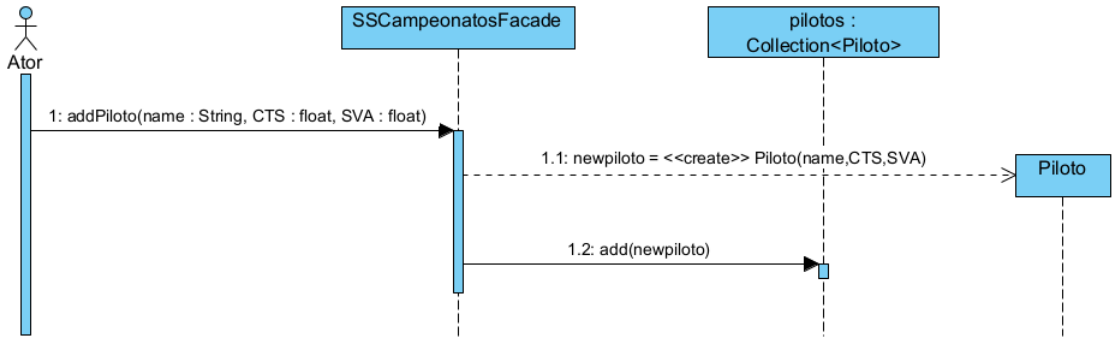


Figura 12 - Diagrama de Sequência "addPiloto".

Este diagrama de sequência corresponde ao método addPiloto que recebe o nome do piloto, o seu critério de Chuva vs Tempo Seco (CTS) e o seu critério Segurança vs Agressividade (SVA), adicionando esse piloto ao sistema.

4.9. campeonatoValido

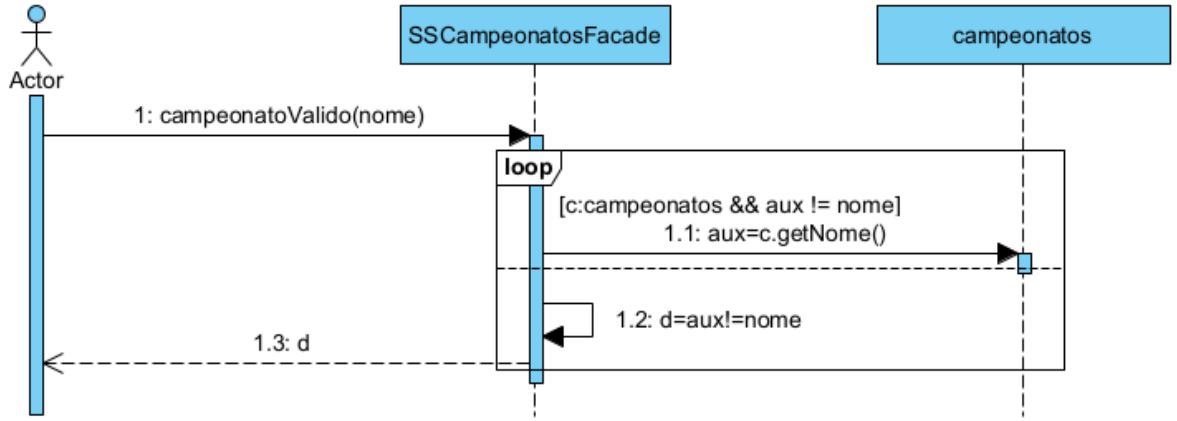


Figura 13 - Diagrama de Sequência "campeonatoValido".

Este diagrama de sequência corresponde ao método campeonatoValido que recebe uma string com o nome do Campeonato e verifica se não existe nenhum Campeonato que contenha esse nome, devolve um Boolean :True se o campeonato é válido - não existe nenhum campeonato com esse nome no sistema), False, caso contrário.

4.10. adicionaCampeonato

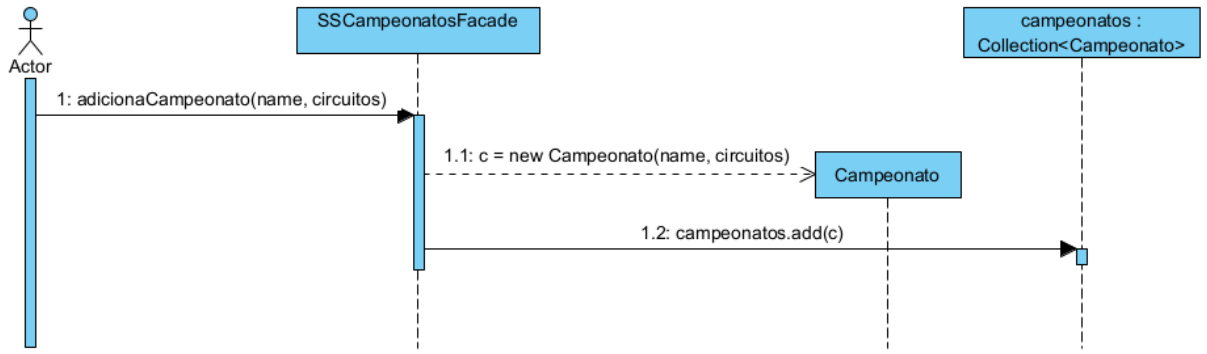


Figura 14 - Diagrama de Sequência "adicionaCampeonato".

Este diagrama de sequência corresponde ao método adicionaCampeonato que recebe uma string com o nome do Campeonato a adicionar e uma coleção de circuitos que serão jogados pelos jogadores do campeonato e adiciona-o à coleção de campeonatos existentes.

4.11. validarCircuito

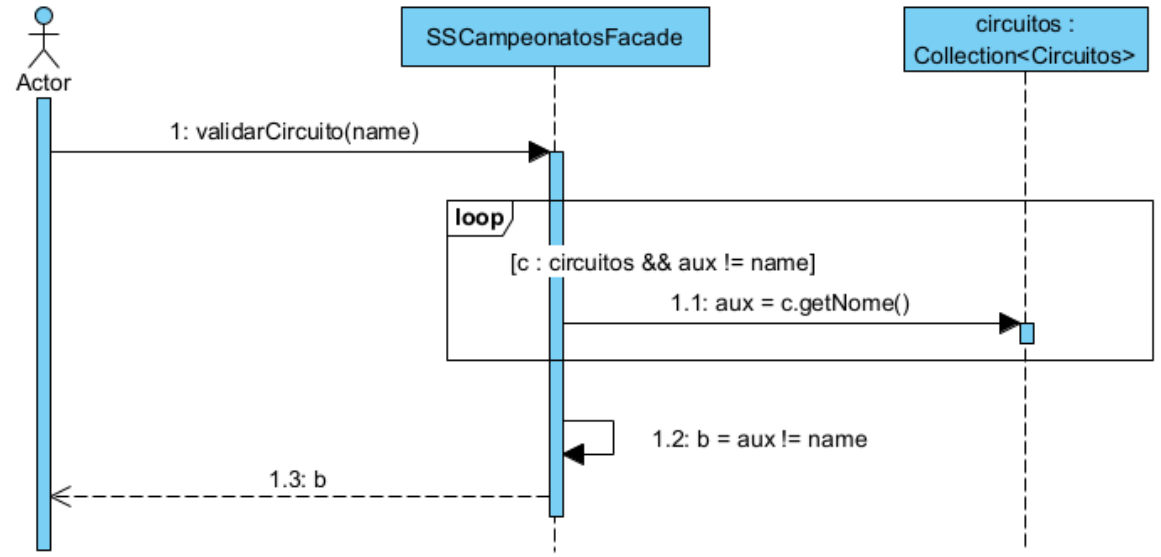


Figura 15 - Diagrama de Sequência "validarCircuito".

Este diagrama de sequência corresponde ao método validarCircuito que verifica se o nome de um circuito já existe na lista de circuitos do sistema.

4.12. adicionaCircuito

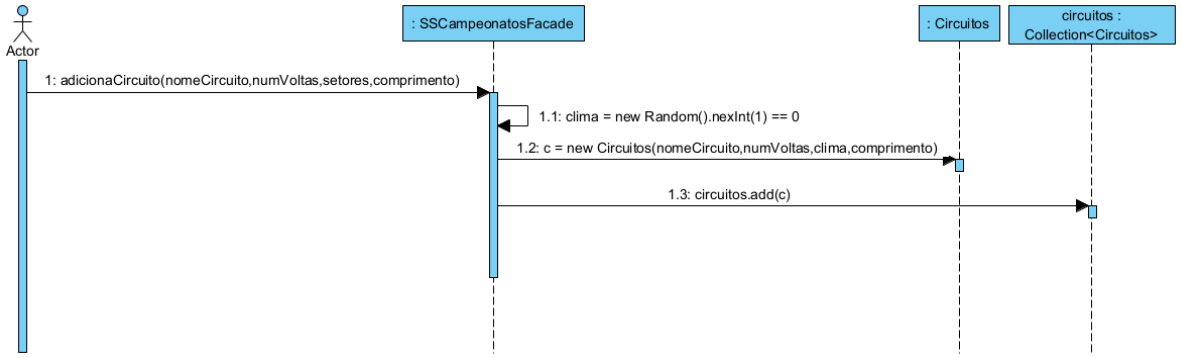


Figura 16 - Diagrama de Sequência "adicionaCircuito".

Este diagrama de sequência corresponde ao método adcionaCircuito que recebe um nome, número de voltas, comprimento e uma lista de setores e cria um circuito para ser adicionado à lista de circuitos do sistema. O clima do circuito será escolhido aleatoriamente.

4.13. addJogador

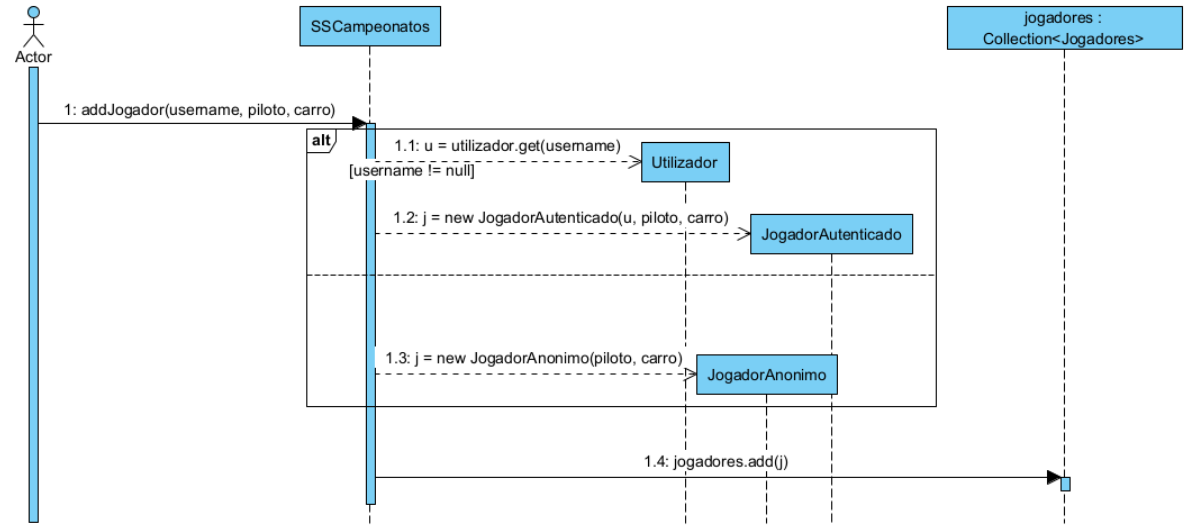


Figura 17 - Diagrama de Sequência "addJogador".

Este diagrama de sequência corresponde ao método addJogador que adiciona um jogador à lista de jogadores do campeonato. Se o método receber um username não nulo o jogador ficará associado a um utilizador senão ficará como um jogador anônimo no campeonato.

5. Conclusão e análise dos resultados obtidos

Depois de termos elaborado o **Diagrama de Componentes**, os **Diagramas de Classes** e os **Diagramas de Sequência**, temos agora, os modelos comportamentais necessários para descrever o comportamento pretendido para o nosso sistema, bem como uma arquitetura conceptual do nosso programa, capaz de suportar os requisitos identificados.

Em primeiro lugar, com o Diagrama de Componentes que desenvolvemos, somos capazes de ter uma visão da estrutura física da arquitetura do nosso sistema.

Os Diagramas de Classes permitiram-nos ilustrar os relacionamentos existentes entre as classes do nosso sistema.

Por último, os Diagramas de Sequência auxiliaram a detalhar como as operações do nosso sistema se desenvolvem, assim como as interações a alto nível entre utilizadores e o sistema e entre subsistemas.

Em suma, através desta concepção da solução do nosso problema, a próxima etapa de desenvolvimento do sistema será realizada de uma maneira mais metódica e organizada, otimizando assim a sua qualidade e futura interpretação.