

**Licenciatura em Engenharia Informática**

***Programação Orientada aos Objetos***

**Docentes:** António Luís Pinto Ferreira Sousa, António Manuel Nestor Ribeiro, João Alexandre Baptista Vieira Saraiva, José Francisco Creissac Freitas Campos

***Racing Manager***

|  |
| --- |
| Grupo 56 |
| |  |  |  | | --- | --- | --- | | C:\Users\marcelo\Pictures\EU\Pedro Faria.jpg | Macintosh SSD:Users:miguelpinto:Desktop:SAM_3128.JPG | Macintosh SSD:Users:miguelpinto:Desktop:975861_3160161781109_752765076_n.jpg | | Pedro Faria - 60998 | Luís Pinto - 61049 | Pedro Lima – 61061 | |
| Braga, Junho de 2013 |

# Resumo:

Neste relatório está explícito todo o desenvolvimento do projecto *“Racing Manager”* desta unidade curricular.

Neste projecto teve-se atenção em reutilizar código, assim ao realizar-se as declarações das classes, organizou-se em hierarquias. Tornando-se mais fácil fazer inserções de novas classes. Por exemplo, as subclasses da classe abstrata Veículo “herdam” todas as variáveis e métodos da classe Veículo. Este procedimento permite reduzir o tamanho do código em cada uma das subclasses e torna possível inserir uma nova subclasse sem qualquer complicação.

Para cada uma das classes, foram definidas as variáveis de instância, os construtores (construtor vazio, construtor por partes e construtor de cópia) e os métodos: *getters, setters*, *equals*, *toString*, *clone*.

Foram feitos outros métodos necessários a cada classe e que permitiram ao utilizador um bom aproveitamento do programa.

Numa última fase foi feito um menu com todas as funcionalidades do programa.

Índice

Resumo: 2

1. Introdução 5

2. Classes 6

2.1 Piloto 6

2.1.1 Métodos 6

2.2 Veículo 6

2.2.1 Métodos 7

2.2.2 Subclasses 8

2.3 Corrida 13

2.3.1 Métodos 13

2.4 Circuito 13

2.4.1 Métodos 14

2.5 Campeonato 14

2.5.1 Métodos 14

2.6 Jogador 15

2.6.1 Métodos 15

2.7 Aposta 16

2.7.1 Métodos 16

3. Manager 17

3.1 Métodos 17

*4.* Interface *“Main”* 18

5. Conclusão 29

Índice Figuras

[Fig. 1 - Classe Manager 17](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323481)

[Fig. 2 - Classe Veículo e Subclasses PC1, PC2, GT, SC 12](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323482)

[Fig. 3 - Classe Circuito 14](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323483)

[Fig. 4 - Classe Piloto 6](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323484)

[Fig. 5 - Classe Jogador 15](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323485)

[Fig. 6 - Classe Campeonato 14](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323486)

[Fig. 7 - Classe Corrida 13](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323487)

[Fig. 8 - Classe Aposta 16](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323488)

[Fig. 9 - Menu Inicial 18](#_Toc358323489)

[Fig. 10 - Criar Jogadores 18](#_Toc358323490)

[Fig. 11 - Inserir Jogador 19](#_Toc358323491)

[Fig. 12 - Carrega Jogo 19](#_Toc358323492)

[Fig. 13 - Grava Jogo 19](#_Toc358323493)

[Fig. 14 - Inserir Jogador 20](#_Toc358323494)

[Fig. 15 - Remove Jogador 20](#_Toc358323495)

[Fig. 16 - Menu Principal 20](#_Toc358323496)

[Fig. 17 - Exemplo de uma corrida 22](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358323497)

[Fig. 18 - Classificação Geral e Menu Principal 22](#_Toc358323498)

[Fig. 19 - Menu Consultas 23](#_Toc358323499)

[Fig. 20 - Exemplo Classificação Geral 24](#_Toc358323500)

[Fig. 21 - Exemplo Trofeu Hibrido 25](#_Toc358323501)

[Fig. 22 - Exemplo Corridas Agendadas 25](#_Toc358323502)

[Fig. 23 - Exemplo lista de jogadores 26](#_Toc358323503)

[Fig. 24 - Menu Apostas 26](#_Toc358323504)

[Fig. 25 - Exemplo lista de corridas 27](#_Toc358323505)

[Fig. 26 - Exemplo lista de carros 27](#_Toc358323506)

[Fig. 27 - Exemplo de quantia a apostar 27](#_Toc358323507)

[Fig. 28 - Exemplo da escolha do 1º Classificado 28](#_Toc358323508)

# Introdução

No âmbito da cadeira de Programação Orientada aos Objetos, perante o problema apresentado iremos desenvolver em Java uma aplicação que visa simular acontecimentos desportivos, neste caso mais direcionado para o desporto automóvel.

O objectivo é criar um programa, em que os utilizadores registados no mesmo fazem apostas relativas à classificação de uma prova automobilística que o software vai simular.

Definiu-se as classes **Campeonato, Circuito, Corrida, Aposta, Piloto, Jogador, Veículo**, assim como as suas subclasses. Assim para **Veículo** definiu-se os seguintes tipos: PC1, PC2, GT e SC. Sendo que cada uma destas subclasses “herda” as variáveis de instância e os métodos da classe **Veículo**, poupando-se código e tornando-se mais fácil a reutilização do código.

Criamos também a classe **Manager,** onde disponibilizamos os métodos de interação com o programa.

Foi também construído um menu para facilitar a utilização dos métodos anteriormente referidos, e tornar o programa mais funcional e interessante.

# Classes

Após a leitura e discussão do enunciado, decidiu-se criar as principais classes para o projeto: Piloto, Veículo, Corrida, Circuito, Campeonato, Jogador, Aposta.

## 2.1 Piloto

A aplicação possuí informações sobre os dotes de condução de cada piloto. Assim a classe Piloto regista o nome, a nacionalidade, o número de provas já vencidas, a qualidade geral do piloto e a capacidade de condução à chuva, que será um factor de incremento da qualidade do piloto quando a corrida se realiza nessas mesmas condições.

Para esta classe, Piloto, foi criado o Construtor Vazio, Construtor de cópia e o Construtor por partes.

### 2.1.1 Métodos

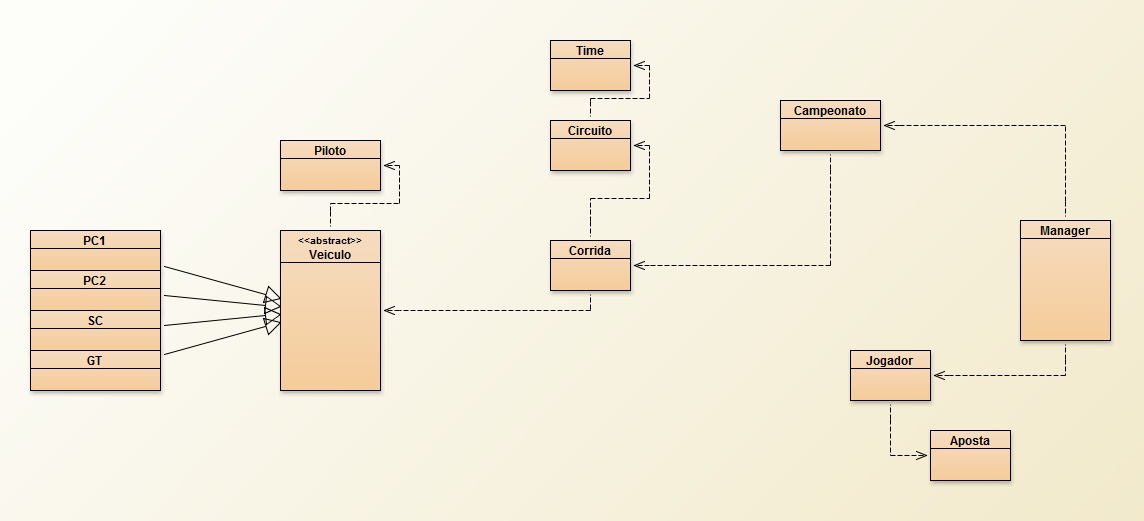
Foram criados os *getters*, *setters*, *equals*, *toString* e *clone*.

Fig. 1 - Classe Piloto

De forma a serem gerados pilotos de forma aleatória foram criados os seguintes métodos com o uso da classe *Random*: “*daNome”* que cria aleatoriamente um nome; “*daNacionalidade”* atribui a um piloto uma nacionalidade; “*geraPiloto”*  gera um piloto através dos métodos indicados atrás e recorrendo aos métodos da classe *Random*, “*nextInt(25)*”, “*nextInt(9)+1*” e “*nextBoolean()*”, gera um possível palmarés, qualidade de condução e capacidade de conduzir à chuva do piloto, assim respectivamente.

## 2.2 Veículo

Num campeonato podem participar veículos de diferentes categorias. Apesar dos veículos serem distintos, todos eles têm algumas características em comum. Sendo assim criou-se a classe Veículo, esta classe é abstracta servindo assim como modelo para as subclasses. Desta forma na classe veículo fica registado: a marca, o modelo, a cilindrada, a potência, os pilotos que conduzem.

A pensar em qual dos dois pilotos está a conduzir o carro criamos um booleano, que nos irá indicar qual dos pilotos está a conduzir o veículo.

Todas estas características são comuns a todos os veículos, assim sempre que for necessário inserir um novo veículo com propriedades extra a estas, basta criar uma subclasse. As já existentes são: PC1, PC2, GT, SC, sendo que as subclasses PC1, PC2, GT são abstractas uma vez serve de modelo para a subclasse PC1Normal e PC1Hibrido estas herdam da classe PC1 os seus métodos, que por sua vez herda da classe veículo e assim sucessivamente com as restantes classes.

### 2.2.1 Métodos

Foram criados os *getters*, *setters*, e abstratamente: *equals*, *toString* e *clone.*

Para além destes, foi também criado um método, designado por “*tempoProximaVolta”* que determina o tempo por volta a num determinado circuito, “*daMarca” e “daModelo”* que gera a respectiva marca e modelo de um carro de forma aleatória. Posteriormente criamos o método “*geraHibrido”* que cria através de um *Random* os veículos que podem ser híbridos(PC1, PC2, GT), criamos o método “*geraVeiculo”* que como o próprio nome indica gera veículos de diferentes categorias atráves do uso dos métodos anteriores e baseado na classe *Random*; “*geraVeiculos”*  que tem como objectivo criar um conjunto de veículos que irão participar no campeonato, assim sempre que um veículo é gerado através da “*geraVeiculo”* irá ser adicionado um *HashSet<Veiculo>.* O número de veículos em prova é gerado aleatoriamente, mas é sempre garantido que estão 15 veículos em prova. E por fim o método “*veHib*” que testa se um veículo tem implementada a interface Híbrida.

### 2.2.2 Subclasses

#### 2.2.2.1 Protótipo Classe 1 (Abstract)

São protótipos feitos especialmente para este campeonato. Nesta subclasse temos duas constantes em que uma é a cilindrada, 6000, e a outra é a fiabilidade, 95%. Decidiu-se alterar a fiabilidade padrão pois em grande partes das voltas existiam demasiadas desistências.

Definimos também para os veículos criados deste tipo algumas regras a quando são gerados, ou seja a potência do motor estará entre os 1300 e 700 cavalos, e a potência do motor elétrico 25 e 175 KW.

Para além do construtores, dos métodos equals, clone e toString, foram declarados abstratamente os método: “*tempoProximaVolta”* que irá determinar o tempo por volta a um determinado circuito; “*calculaFiabilidade”* que devolve a fiabilidade de um PC1; “*hashCode”* que para cada veiculo irá gera um código único.

##### 2.2.2.1.1 PC1Normal

Esta subclasse herda todos os métodos da classe PC1 que por sua vez herda da classe veículo. Sendo assim esta classe terá os mesmos métodos que a classe PC1. Foram então definidos os métodos *equals*, *clone*, *toString e hashCode*.

O “*tempoProximaVolta”* desta classe é calculado com base num valor aleatório, entre o tempo médio por volta dos PC1 e o tempo recorde desta pista, e também pela potência do motor e capacidades dos pilotos, ou seja quanto mais um piloto é “bom” e a qualidade do motor é superior obtém-se melhores tempos por volta, caso contrário o inverso. Dentro desta função existe também um controlo ao piloto que está a conduzir e ao seu número de voltas.

##### 2.2.2.1.1 PC1Hibrido

Os carros híbridos são veículos que possuem as características do PC1, herança, e mais um outro motor, eléctrico, que permite aumentar a potência disponível. Foram então definidos os métodos *equals*, *clone*, *toString e hashCode*.

Como está classe tem implementada a Interface “Híbrida” tem de implementar o método “*getPotenciaMotorElectrico*” que devolve a potência do motor em KW. Em seguida criou-se um método que converte KW para CV, “*converteKWtoCV”* uma vez que o Watt é uma unidade de potência que pode ser convertida para cavalos. Desta forma depois é só somar ambas as potências, a do motor eléctrico com a do motor de combustão para obtermos a potência do veículos híbridos.

A fiabilidade deste protótipo é menor em função da capacidade do motor elétrico instalado, para calcular está usa-se o método “*calculaFiabilidade”.*

A fórmula para calcular o tempo próxima volta deste Híbrido não difere muito da *PC1Normal,*  a única diferença é que este método tem uma potência extra que fará obter melhores tempos.

#### 2.2.2.2 Protótipo Classe 2 (Abstract)

São veículos de alta performance que podem entrar noutros campeonatos. Nesta subclasse temos pré-definido que a cilindrada varia entre os 4000 e 6000 cm3, a potência do motor entre os 950 e 550 cavalos e a fiabilidade deste tipo ser cerca de 85%, mas é ainda em função da cilindrada. Decidiu-se alterar a fiabilidade padrão(70%) pois em grande partes das voltas existiam demasiadas desistências para este tipo de carro.

Para além do construtores deste tipo, foram declarados abstratamente os método: *equals*, *toString,* *clone*, “*tempoProximaVolta”* que irá determinar o tempo por volta a um determinado circuito; “*calculaFiabilidade”* que devolve a fiabilidade de um PC2; “*hashCode”* que para cada veiculo irá gera um código único.

##### 2.2.2.2.1 PC2Normal

Tal como os PC1Normal, esta subclasse para possui todas as características do PC2 que por sua vez possui da classe veículo, desta forma, terá também os mesmos métodos que a classe PC2, assim sendo definiu-se os construtores e métodos: *equals, clone, toString, hashCode, “tempoProximaVolta”* e *“calculaFiabilidade”.*

O método “*tempoProximaVolta”* é calculado da mesma forma que na classe *PC1* mas agora têm-se em atenção a cilindrada do veiculo que é um factor de incremento na diminuição do tempo da volta. Tem-se ainda atenção se está corrida se realiza com o piso molhado/seco sendo que quando chove o tempo por volta sobe numa fórmula em que entra a capacidade do piloto de conduzir à chuva.

O método “*calculaFiabilidade”* baseia-se na cilindrada do veiculo para resultar uma fiabilidade, sendo que quanto melhor é a cilindrada melhor é a fiabilidade. Fórmula da recta (y=mx+b).

##### 2.2.2.2.2 PC2Hibrido

Implementa os mesmos tipos de métodos de PC1Hibrido. Definiu-se por padrão que a potencia do motor elétrico dos veículos deste tipo variam entre os 25 e os 200 Kw.

A fiabilidade vai ser menor que os PC1Hibrido, dependendo da potencia dos motores.

#### 2.2.2.3 Gran Turismo (Abstract)

São veículos desportivos produzidos em massa.

Como nos tipos de veículos definidos anteriormente herda os métodos definidos da classe Veículo e foram definidos e declarados os construtores e métodos: *equals, clone, toString, hashCode, “tempoProximaVolta”* e *“calculaFiabilidade”.*

Para esta classe está predefinido que a cilindrada esteja entre 3000 e 4500 cm3, a potência do motor ronde 400 e 800 cavalos e a fiabilidade inicial de cerca 85%, mas diminui com o desenrolar da corrida e a potência do motor.

##### 2.2.2.3.1 GTNormal

Esta subclasse também herda todos os métodos da classe GT que por sua vez herda da classe veículo. Sendo assim terá os mesmos métodos que a classe GT.

Os métodos “*calculaFiabilidade*” e “*tempoProximaVolta*” são calculados da mesma forma que foram descritos nos tipos de veículos anteriores, neste caso mais propriamente os *PC2* por o valor da cilindrada também variar.

##### 2.2.2.3.2 GTHibrido

Nesta classe, também guarda-se informação relativa à potência.

Como descrito anteriormente nas outras classes que implementam a interface *Híbrida,* a fiabilidade e o tempo das voltas sofrem sempre ligeiras alterações.

#### 2.2.2.4 Stock Cars (SC)

São carros derivados dos automóveis quotidianos. Este tipo de veículo para além das características de todos os veículos possui uma constante relativa à fiabilidade. Sendo assim foi criado um método que calcula a respectiva fiabilidade do GT, *“calculaFiabilidade”*, em que 75% é função do piloto e 25% da cilindrada e no seu geral a fiabilidade deste tipos de veículos nunca é inferior a 60%. Além disto está predefinido que a cilindrada é 2500cm3, e que a potência do motor varia entre os 100 e 200 cavalos.

O método “*tempoProximaVolta”* implementado é parecido com o descrito em PC1, pois também tem a constante cilindrada fixa.

|  |
| --- |
| Fig. 2 - Classe Veículo e Subclasses PC1, PC2, GT, SC |

## 2.3 Corrida

A cada corrida está associado a um circuito, *“crt”*, um conjunto de veículos guardados em um *HashSet, “conjveiculos”*, que pertencem a diferentes categorias, e que são conduzidos por equipas de dois pilotos, e a condição da pista, *“piso”*.

Foi utilizado uma *HashSet* porque é um conjunto de objetos em que não existem elementos repetidos e a ordem não é importante.

Foram criados ainda o construtor vazio, o construtor por partes e o construtor por cópia.

### 2.3.1 Métodos

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 3 - Classe Corrida |

Foram também criados os métodos *equals, toString, clone.* Além deste foram criados outros: “*geraCorrida”* que recebe um conjunto de veículos como parâmetro e cria para uma corrida; *“geraCircuito”* que aleatoriamente atribui a uma corrida um circuito, nunca repetido; *“getTempoMS”* que converte um tempo em milissegundos para minuto, segundos e milissegundos; *“fazCorrida”*  que realiza uma corrida num determinado circuito e no final atribui a classificação a cada veículo, antes disto recorre ao método *“fazVoltas”*  que verifica se um dado veiculo não desistiu da prova e implementa outro método “fazVolta” que a cada volta através do método *“tempoProximaVolta”* tem o tempo que um determinado veículo efetuou a numa volta.

## 2.4 Circuito

A classe Circuito regista a seguinte informação: nome do circuito, distância da pista, número de voltas, tempo médio por volta por categoria de carro, tempo recorde da pista, desvio ao tempo médio em caso de chuva, tempo de paragem nas boxes e o piloto que fez o melhor tempo, recordista.

Foram criados ainda o construtor vazio, o construtor por partes e o construtor por cópia.

### 2.4.1 Métodos

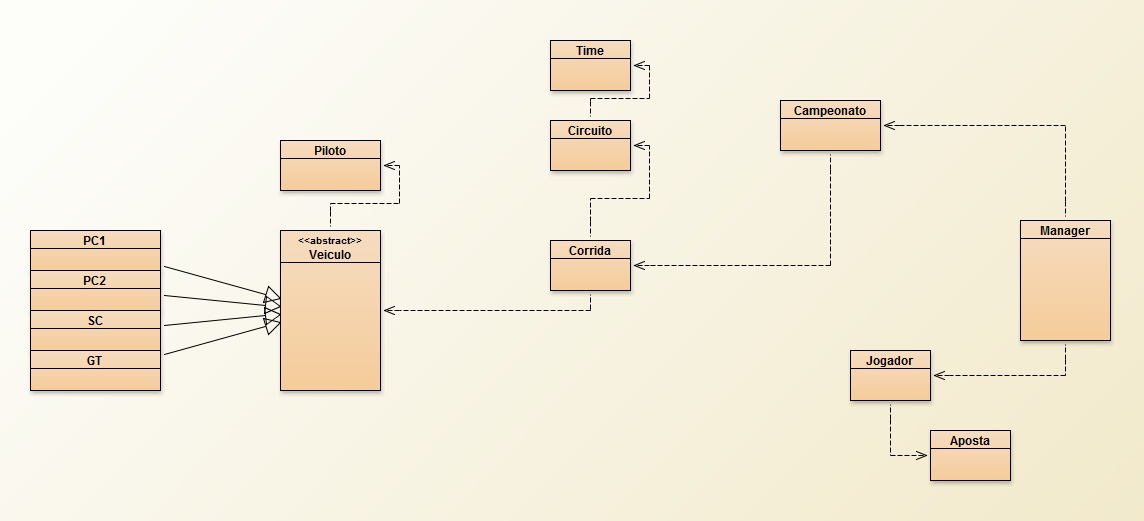
Foram criados os métodos habituais: setters, getters, equals, clone e toString.

Fig. 4 - Classe Circuito

## 2.5 Campeonato

Cada campeonato é composto por um conjunto de corridas para isso utilizamos a coleção *TreeSet<E>* pois implementa conjuntos baseada numa árvore binária ordenada, sendo que assim teremos as corridas ordenadas por um *“compareTo”* e também não vão existir corridas do mesmo tipo, repetidas*.*

Foram criados ainda o construtor vazio, o construtor por partes e o construtor por cópia.

### 2.5.1 Métodos

Foram criados os *getters*, *setters*, *equals*, *toString* e *clone*. Para além destes métodos também foram criados outros: *fazCampeonato()* que como o próprio nome indica irá realizar um campeonato recorrendo a outros métodos, “*fazCorrida”;* *“geraCampeonato”* que através da coleção *Random* gera, *nextInt(15)+15,* o número mínimo de corridas que o campeonato vai ter e a cada corrida associa um conjunto de veículos que foi gerado anteriormente.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 5 - Classe Campeonato |

## 2.6 Jogador

Para cada jogador guardamos as seguintes informações: nome, morada, aposta em vigor, histórico de apostas, conta corrente, o dinheiro investido e o total de ganhos nas apostas.

Os utilizadores registados para cada corrida podem apostar nos três primeiros classificados. Sendo assim, para guardar quer as apostas em vigor quer o histórico de apostas, utilizamos a coleção *ArrayList.* Desta forma temos como suporte um *array* de características dinâmicas, ou seja, capaz de aumentar ou diminuir de dimensão ao longo da execução de um programa.

Foram criados ainda o construtor vazio, o construtor por partes e o construtor por cópia.

### 2.6.1 Métodos

Em relação aos métodos nesta classe foram criados os getters, setters, equals, toString e clone. Criamos também um método designado por *CheckApostas* que é onde verifica em quantas/ quais apostas o jogador acertou e um método chamado *fazAposta* que é onde se gera um nova aposta.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 6 - Classe Jogador |

## 2.7 Aposta

Na aplicação os jogadores podem apostar nos três primeiros classificados. Desta forma nesta classe guardamos informação relativa à quantidade, valor da aposta, e aos veículos a apostar. Porém quando uma corrida começa não é possível efetuar apostas sobre a mesma, mas quando a corrida acaba já é possível efetuar apostas para as corridas seguintes.

Foram criados ainda o construtor vazio, o construtor por partes e o construtor por cópia.

### 2.7.1 Métodos

Para esta classe foram criados os métodos: *getters*, *setters*, *equals*, *toString*, *clone* e também criamos os métodos “*fazAposta”* em que para uma determinada corrida se aposta em três veículos e o método *“checkApostas”* que verifica para cada aposta feita, “*checkAposta”* , se o jogador consegui acertar na aposta realizada. Caso tenha sucesso será decrementado o dinheiro investido, a conta corrente irá sofrer um aumento mediante o valor da aposta e estará disponível ao jogador quanto já ganhou neste campeonato.

Depois uma aposta verificada é removida das apostas em vigor e adicionada ao seu histórico de apostas.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 7 - Classe Aposta |

# Manager

***private*** *Campeonato campeonato;*

***private*** *TreeMap<String, Jogador> apostadores;*

***public******int*** *corrida;*

***public*** *HashMap<Veiculo, Integer> campstatus;*

É necessário que o campeonato, os jogadores, a posição que vão as corridas e a classificação dos veículos sejam guardadas numa estrutura em que sempre que for necessário ter acesso a esta informação.

Foi utilizado um TreeMap<String, Jogador> para armazenar as informações do jogador, ordenado alfabeticamente, onde não há jogadores iguais.

Foi criado uma interface “*Hibrida”* que implementa o método *“getPotenciaMotorElectrico*” nos veículos do tipo PC1, PC2 e GT.

Foram criados ainda o construtor vazio, o construtor por partes e o construtor por cópia.

### 3.1 Métodos

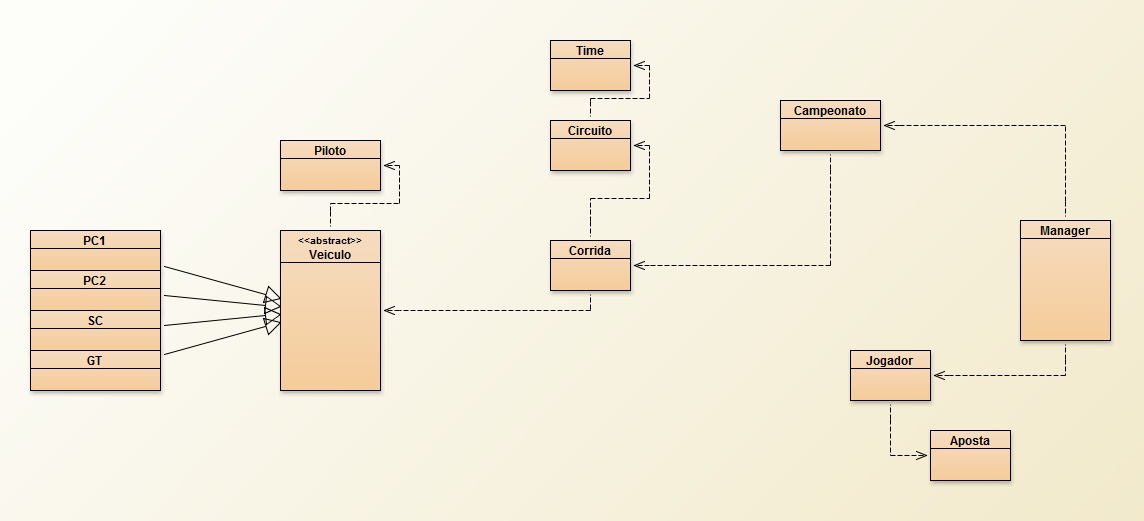
|  |
| --- |
| Fig. 8 - Classe Manager |

Relativamente aos métodos desta classe, foram criados os habituais *getters, setters, equals*, *toString* e *clone.*

Nesta classe encontram-se os métodos de leitura, *“carregaRM”* ,e escrita, *“gravaRM”,* da informação que se encontra no sistema. Mais propriamente, possibilita carregar a informação do campeonato e dos jogadores, para um ficheiro e também alterar essa informação se possível.

Foram ainda criados os métodos que adicionam e removem jogadores da aplicação, assim respectivamente, *“adicionaJogador”* e *“removeJogador”*.

Ainda nesta classe está o método que dá o top Jogadores das apostas na aplicação, *“topJog”,* ou seja os jogadores que já ganharam mais com as suas apostas.



# Interface *“Main”*

Nesta classe estará a estrutura do nosso programa principal, desta forma, encontram-se definidos os métodos de leitura e escrita da informação que se encontra no sistema. Mais propriamente, possibilita carregar a informação das corridas e dos veículos.

O método “*Welcome”* é um menu vertical que apresenta três opções de escolha.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\1.jpg  Fig. 9 - Menu Inicial |

O método “*PedeJogadores”* é também um menu sendo que este pergunta quantos jogadores queremos e após isso pede para inserir os dados relativos a cada jogador, nomeadamente o nome, morada e montante inicial.

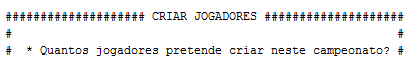


Fig. 10 – Criar Jogadores

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\8.jpg |
| Fig. 11 - Inserir Jogador |

O método “*MenuCarregaJogo”* pede apenas para inserir o nome do ficheiro que o utilizador pretende carregar.

|  |
| --- |
| C:\Users\marcelo\Desktop\cria jogadores.png  Fig. 12 - Carrega Jogo |

O método “*MenuGravaJogo”* tal como o nome indica irá gravar o jogo dessa forma apenas pede ao utilizador o nome com o qual ele quer gravar o jogo.

|  |
| --- |
| C:\Users\marcelo\Desktop\cria jogadores.png  Fig. 13 - Grava Jogo |

O método “*MenuAdicionarJogador”* insere um novo jogador, sendo assim, pede ao utilizador os dados referentes a este novo jogador sendo eles o nome, a morada e o montante inicial.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\8.jpg  Fig. 14 - Inserir Jogador |

O método “*MenuRemoveJogador”* remove um jogador para isso basta o utilizador inserir o nome do jogador que pretende remover.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\9.jpg  Fig. 15 - Remove Jogador |

O método “*MenuPrincipal”* apresenta sete opções de escolha.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\2.jpg  Fig. 16 - Menu Principal |

O método “*MenuCorrida”* é onde se desenrola a “ação”, dada a ordem para simular a corrida, rapidamente se apresenta quem passa em primeiro lugar em cada uma das voltas e quem desiste em cada volta. No fim é apresentada a classificação final e volta novamente ao Menu Principal.

|  |
| --- |
| C:\Users\marcelo\Desktop\cria jogadores.png |
| C:\Users\marcelo\Desktop\cria jogadores.png  Fig. 17 - Exemplo de uma corrida |
| C:\Users\marcelo\Desktop\cria jogadores.png  Fig. 18 - Classificação Geral e Menu Principal |

O método “*MenuConsultas”* apresenta seis opções de escolha. Sendo que cada uma das opções apresenta-nos a respectiva listagem, por exemplo a opção um apresenta-nos a classificação geral.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\3.jpg  Fig. 19 - Menu Consultas |

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\3.jpg  Fig. 20 - Menu Consultas |
| C:\Users\marcelo\Documents\GitHub\POO\print\4.jpg  Fig. 21 - Exemplo Classificação Geral |
| C:\Users\marcelo\Documents\GitHub\POO\print\5.jpg  Fig. 22 - Exemplo Trofeu Hibrido |
| C:\Users\marcelo\Documents\GitHub\POO\print\6.jpg  Fig. 23 - Exemplo Corridas Agendadas |

O método “*jogpos”* exibe a lista dos jogadores criados. Sendo que neste menu tem de escolher o jogador com que pretende apostar.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\10.jpg  Fig. 24 - Exemplo lista de jogadores |

O método “*MenuApostas”* apresenta oito opções de escolha a um jogador.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\11.jpg  Fig. 25 - Menu Apostas |

O método “*MenuFazAposta”* inicialmente exibe a lista de corridas, após a escolha da corrida exibe a lista dos carros (*escolheVeiculo*) e de seguida mostra o saldo atual e pergunta a quantia que se pretende apostar. Após inserir a quantia volta para o menu apostas.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\12.jpg  Fig. 26 - Exemplo lista de corridas |
| C:\Users\marcelo\Documents\GitHub\POO\print\13.jpg  Fig. 27 - Exemplo lista de carros |
| C:\Users\marcelo\Desktop\cria jogadores.png  Fig. 28 - Exemplo de quantia a apostar |

O método “*escolheVeiculo”* neste método é onde se escolhe sobre qual veículo se pretende apostar, sendo que terá que escolher um veículo para o primeiro lugar, outro para o segundo e outro para o terceiro.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\print\13.jpg  Fig. 29 - Exemplo da escolha do 1º Classificado |

# 5. Conclusão

Neste projeto um dos objectivos foi a hierarquia das classes para assim haver reutilização do código e facilitar inserções de novas classes e subclasses.

Com o código feito desta maneira evita-se repetição de código, já que o que é comum a alguns tipos encontra-se na classe principal, por exemplo, a informação comum a todos os veículos encontra-se na classe Veiculo e o que é característicos de cada tipo de veículo é especificado em cada subclasse. Também assim se torna mais fácil a leitura.

Para guardar informação utilizou-se coleções do tipo *Map’s, Set’s e ArrayList.* Esta decisão foi bastante importante pra o grupo, pois uma boa escolha destas coleções permite um dimensionamento bastante superior do programa e tempos inferiores de execução.

Uma etapa realizada com sucesso foi a criação dos métodos que tornaram possível a utilização do programa. Com isso realizado, o programa possuí uma grande variedade de funcionalidades que o utilizador pode tirar proveito.

A interface final, menu, oferece ao utilizador uma utilização fácil e prática fase as funcionalidades que eram esperadas do programa.