

**Licenciatura em Engenharia Informática**

***Programação Orientada aos Objectos***

**Docentes:** Nestor.

***Racing Manager***

Pedro Miguel de Oliveira Faria - 60998

Luís Miguel Carvalho Pinto - 61049

Pedro Rites Lima – 61061

Braga, 01 de 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. Assim tornou-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* e outros métodos necessários a cada classe.

Foram feitos outros métodos 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: 6

Classes 7

Manager 7

Métodos 7

Veiculo 8

Métodos 8

Subclasses 9

 Protótipo Classe 1 (Abstract) 9

 Protótipo Classe 2 (Abstract) 9

 Grande Turismo (Abstract) 9

 Stock Cars (SC) 10

 PC1Normal 10

 PC2Normal 10

 GTNormal 10

 PC1Hibrido 11

 PC2Hibrido 11

 GTHibrido 11

 Hibrida(Interface) 11

Circuito 13

Métodos 13

Piloto 14

Métodos 14

Jogador 15

Métodos 15

Campeonato 16

Métodos 16

Corrida 17

Métodos 17

Aposta 18

Métodos 18

Diagrama 19

Main 20

Conclusão 23

Índice Figuras

[Fig. 1 - Classe Manager 6](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155990)

[Fig. 2 - Classe Veículo e Subclasses PC1, PC2, GT, SC 11](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155991)

[Fig. 3 - Classe Circuito 12](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155992)

[Fig. 4 - Classe Piloto 12](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155993)

[Fig. 5 - Classe Jogador 13](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155994)

[Fig. 6 - Classe Campeonato 14](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155995)

[Fig. 7 - Classe Corrida 15](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155996)

[Fig. 8 - Classe Aposta 15](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358155997)

# Introdução

No âmbito da cadeira de Programação Orientada aos Objectos, perante o problema 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, Veiculo, Piloto, Jogador, Aposta**, 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.

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 do projecto, decidiu-se criar as principais classes : ***??????????????????????????*** Testa Manager, Manager, Campeonato, Circuito, Corrida, Veiculo, Piloto, Jogador e Aposta.

Para cada uma das classes mencionadas anteriormente, criamos s seguintes construtores: o construtor vazio, construtor de cópia e o construtor por partes.

## Manager

Na classe Manager, são guardados os jogadores, através de uma Hashmap ***????????????????????????????? Falar do CAMPSTATUS***

É nesta classe que disponibilizamos os métodos de interacção com o programa. Métodos estes que vão dar funcionalidades ao utilizador, onde os utilizadores registados fazem apostas relativas à classificação de uma prova.

### Métodos

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 1 - Classe Manager |

Relativamente aos métodos nesta classe, foram criados os getters, setters, equals, toString e clone ***Falar do carregam e gravaRM—Não faço ideia do que sejam***

## Veiculo

Em cada corrida podem participar carros de diferentes categorias. Apesar dos veículos serem distintos, todos eles têm algumas características em comum. Sendo assim criou-se a classe Veiculo, 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 e 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.

### Métodos

Nesta classe foram criados os getters, setters, equals, toString e clone.

Para alem destes, foi também criado um método, designado por *tempoProximaVolta()* que determina o tempo por volta a um 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 todos os veículos de diferentes categorias e por fim criamos o método *geraVeiculos()* através de um *HashSet* implementando assim um conjunto de veículos.

### Subclasses

### Protótipo Classe 1 (Abstract)

São protótipos feitos especialmente para este campeonato. Nesta subclasse temos duas constantes em que uma é a cilindrada e a outra é a fiabilidade Para além dos métodos : equals, clone e toString, foi criado um método, designado por *tempoProximaVolta()* que irá determinar o tempo por volta a um determinado circuito, um método designado por *calculaFiabilidade()* que irá calcular a respectiva fiabilidade do PC1 normal e por fim o método *hashCode()* que para cada veiculo irá gerar um código único.

### Protótipo Classe 2 (Abstract)

São veículos de alta performance que podem entrar noutros campeonatos. Nesta subclasse também utilizamos uma constante para a fiabilidade.

Foi criado um método, designado por *calculaFiabilidade()* que irá calcular a respectiva fiabilidade do PC2 e também o método *tempoProximaVolta()* que irá determinar o tempo por volta a um determinado circuito e também os métodos : equals, clone e toString.

### Grande Turismo (Abstract)

São veículos desportivos de produção em massa. Nesta foi criado um método, designado por *calculaFiabilidade()* que calculará a respectiva fiabilidade do GT, sendo que decrescerá com o desenrolar da corrida a uma determinada taxa e com a cilindrada, para cada carro foi criado também o método *tempoProximaVolta()* que irá determinar o tempo por volta a um determinado circuito e os métodos :

### 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, *calculaFiabilidad()*, em que 75% é função do piloto e 25% da cilindrada. E o método *tempoProximaVolta()* que determina o tempo por volta a um determinado circuito.

### 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.

### 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

### 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.

### PC1Hibrido

Os carros híbridos são veículos que possuem além do motor de combustão, um outro motor, eléctrico, que permite aumentar a potência disponível.

Nesta subclasse apenas se guarda informação relativa à potência. Para além dos métodos que esta classe herda da subclasse PC1 também criamos o método *converteKWtoCV()* que converte de Quilowatt para Cavalos, 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.

### PC2Hibrido

Nesta classe, tal como na PC1Hibrido, guarda-se informação relativa à potência. Para além dos métodos que esta classe herda da classe PC2 foi também criamos o método *converteKWtoCV()* que converte de Quilowatt para Cavalos.

### GTHibrido

Nesta classe, também guarda-se informação relativa à potência. Esta classe herda da classe GT sendo que foi também criamos o método *converteKWtoCV()* que converte de Quilowatt para Cavalos.

### Hibrida(Interface)

Foi criada uma interface *Hibrida*, que é implementada pelas classes PC1Hibrido, PC2Hibrido e GTHibrido, ou seja, à excepção dos SC, todos os veículos em competição podem ser híbridos.

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

## Circuito

Cada corrida realizar-se-á num determinado circuito, sendo assim a classe circuito guarda a informação relativa à distância da pista, tempo médio por volta, por categoria de carro, números de voltas a realizar na corrida, desvio ao tempo médio em caso de chuva, tempo recorde da pista, tempo na paragem das boxes e o piloto recordista.

### Métodos

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

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

## Piloto

Cada piloto possui informação sobre os seus dotes de condução, sendo assim, a informação guardada relativa a cada piloto é o nome, a nacionalidade o número de provas já vencidas, a qualidade geral do piloto, a capacidade de condução à chuva, que será um factor de incremento da qualidade do piloto quando a corrida se realiza nessas mesmas condições.

### Métodos

Relativamente aos métodos nesta classe foram criados os getters, setters, equals, toString e clone. Também foram criados os métodos *daNome()* que através de um *Random* irá dar um nome de um piloto, o método *daNacionalidade()* que através da mesma forma, *Random*, irá dar a nacionalidade e por fim temos o método *geraPiloto()* que ira dar um Piloto.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 4 - Classe Piloto |

## Jogador

Para cada jogador guardamos as seguintes informações: nome, morada, aposta em vigor, histórico de apostas e conta corrente.

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 o *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

### 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. 5 - Classe Jogador |

## Campeonato

Cada campeonato é composto por um conjunto de corridas para isso utilizamos a colecção TreeSet<E> pois implementa conjuntos baseada numa árvore binária ordenada, sendo que assim teremos as corridas ordenadas.

### Métodos

Foram criados os getters, setters, equals, toString e clone. Para além destes métodos também foram criados método *fazCampeonato()* que como o próprio nome indica irá conceber um campeonato e criamos também o método *geraCampeonato()* que através de um *Random gera o campeonato* ***????????????????????????????????****.*

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

## Corrida

Cada corrida realiza-se num circuito, em cada corrida participam um conjunto de carros, guardados em um *HashSet*, que pertencem a diferentes categorias, e que são conduzidos por equipas de dois pilotos.

A classe corrida desta forma terá que guardar as informações relativas a um circuito e relativamente ao piso.

### Métodos

Para além dos métodos habituais: setters, getters, equals, clone e toString, também foram criados outros métodos *fazVolta* ???????????, *fazVoltas* ????????????????? e *fazCorrida* ??????????? o método geraCircuito() que através de *Random* vai gerar um circuito, o método *geraCorrida()* que irá gerar da mesma forma uma corrida, porém este *Random* não será de inteiros mas sim de booleanos devido ao facto de ocorrer chuva ou não.

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

## Aposta

Na aplicação os jogadores podem apostar nos três primeiros classificados. Desta forma nesta classe guardamos informação relativa à quantidade, aos carros, sendo que, quando uma corrida começa não é possível efectuar apostas sobre a mesma, mas quando a corrida acaba será possível efectuar apostas para as corridas seguintes.

### Métodos

Para esta classe foram criados os métodos: getters, setters, equals, toString e clone e também criamos o método *checkAposta()*.???????????

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  Fig. 8 - Classe Aposta |

# Diagrama

|  |
| --- |
|  |

# 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.

|  |
| --- |
|  |

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.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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.

|  |
| --- |
|  |

O método *MenuAdicionarJogador(Manager m)* 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.

|  |
| --- |
|  |

O método *MenuRemoveJogador(Manager m)* remove um jogador para isso basta o utilizador inserir o nome do jogador que pretende remover.

|  |
| --- |
|  |

O método *MenuPrincipal(Manager m)* apresenta sete opções de escolha.

|  |
| --- |
|  |

O método *MenuCorrida(Manager m)* é onde a acção se desenrola, 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. De seguida é apresentada a classificação final e no fim expõe-se novamente o Menu Principal.

|  |
| --- |
|  |

O método *MenuConsultas(Manager m)* 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.

|  |
| --- |
|  |

O método *jogpos(TreeMap<String, Jogador> treeMap)* exibe a lista de todos os jogadores anteriormente criados, e seleccionamos o jogador que pretendemos.

|  |
| --- |
|  |

O método *MenuApostas(Manager m, int i)* apresenta sete opções de escolha.

|  |
| --- |
|  |

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

|  |
| --- |
|  |

O método *escolheVeiculo(Corrida corr,int classi)* 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.

|  |
| --- |
|  |

# Conclusão

Neste projecto 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 HashMaps,HashSet, TreeMap e o ArrayList. Esta decisão foi bastante importante pra o grupo, pois uma boa escolha permite uma dimensão bastante grande do programa.

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 oferece ao utilizador uma utilização fácil e prática fase as funcionalidades que eram esperadas do programa.