|  |  |
| --- | --- |
|  | **Escola de Engenharia**  Departamento de Informática  Licenciatura em Engenharia Informática |

Projecto P.O.O.

***Racing Manager***

Braga, 01 de Junho de 2013

Pedro Miguel de Oliveira Faria - 60998

Luís Miguel Carvalho Pinto - 61049

Pedro Rites Lima – 61061

# Resumo:

**REESCREVER TA MT IGUAL AO DA 41**

No presente relatório está descriminado todo o desenvolvimento do projecto *“Racing Manager”* desta unidade curricular.

Neste projecto teve-se em atenção a reutilização do código, desta forma ao realizar-se as declarações das classes, organizou-se em hierarquias. Assim tornou-se mais fácil fazer inserções de novas classes.

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

Foram feitos métodos base como inserções, remoções, procuras e as listagens. E foram feitos todos os métodos mais complexos e mais importantes. Estes últimos serão os métodos que permitiram ao utilizador um bom aproveitamento do programa.

Numa última etapa foi feita um menu com todas as funcionalidades do programa.

Índice

[Resumo: 2](#_Toc358028758)

[Introdução: 5](#_Toc358028759)

[Classes 6](#_Toc358028760)

[Manager 6](#_Toc358028761)

[Métodos 6](#_Toc358028762)

[Veiculo 7](#_Toc358028763)

[Métodos 7](#_Toc358028764)

[Subclasses 7](#_Toc358028765)

[Circuito 9](#_Toc358028766)

[Métodos 9](#_Toc358028767)

[Piloto 9](#_Toc358028768)

[Métodos 10](#_Toc358028769)

[Jogador 10](#_Toc358028770)

[Métodos 11](#_Toc358028771)

[Campeonato 11](#_Toc358028772)

[Métodos 11](#_Toc358028773)

[Corrida 12](#_Toc358028774)

[Métodos 12](#_Toc358028775)

[Aposta 13](#_Toc358028776)

[Métodos 13](#_Toc358028777)

Índice Figuras

[Fig. 1 - Classe Manager 6](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028778)

[Fig. 2 - Classe Veículo e Subclasses PC1, PC2, GT, SC 9](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028779)

[Fig. 3 - Classe Circuito 9](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028780)

[Fig. 4 - Classe Piloto 10](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028781)

[Fig. 5 - Classe Jogador 11](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028782)

[Fig. 6 - Classe Campeonato 12](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028783)

[Fig. 7 - Classe Corrida 12](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028784)

[Fig. 8 - Classe Aposta 13](file:///C:\Users\marcelo\Desktop\Relatorio%20POO.docx#_Toc358028785)

# Introdução:

No âmbito da cadeira de Programação Orientada aos Objectos, perante o projecto apresentado vai-se desenvolver em Java uma aplicação que visa simular acontecimentos desportivos.

O objectivo é criar um jogo, 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 ***??????????????????????????*** **Testa Manager, Manager, Campeonato, Circuito, Corrida, Veiculo, Piloto, Jogador e Aposta**, assim como as suas subclasses da classe Veículo. Neste caso para **Veículo** definiu-se os tipos de veículos: 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

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***

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

## 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: CP1, CP2, GT, SC.

### 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 (PC1Normal)

São protótipos feitos especialmente para este campeonato. E possuem as características de todos os veículos. Para além dos métodos habituais: setters, getters, equals, clone e toString, foi criado um método, designado por *tempoProximaVolta()* que determina 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()****????????????????????????????????****?*

#### Protótipo Classe 2 (PC2Normal)

São veículos de alta performance que podem entrar noutros campeonatos. Tal como os PC1, este tipo de veículo para possui todas as características de todos os veículos.

Foi criado um método, designado por *calculaFiabilidade* que calcula a respectiva fiabilidade do PC2 e também o método *tempoProximaVolta()* que determina o tempo por volta a um determinado circuito e o método *hashCode()****????????????????????????????????****?*

#### Grande Turismo (GTNormal)

São veículos desportivos de produção em massa. Nesta subclasse foi criado um método, designado por *calculaFiabilidade()* que calcula a respectiva fiabilidade do GT, que decresce com o desenrolar da corrida a uma determinada taxa e com a cilindrada, para cada carro foi criado também o método *tempoProximaVolta()* que determina o tempo por volta a um determinado circuito e o método *hashCode()****????????????????????????????????****?*

#### Stock Cars (SC)

São carros derivados dos automóveis quotidianos. Este tipo de veículo para alem das características de todos os veículos possui informação relativa à fiabilidade. Sendo assim foi criado um método que calcula a respectiva fiabilidade do GT, *calculaFiabilidad()e*, 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.

|  |
| --- |
| C:\Users\marcelo\Documents\GitHub\POO\BLJPOO.jpg  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 |

# Main

Nesta classe encontram-se os métodos de leitura e escrita da informação que se encontra no sistema. Mais propriamente, possibilita carregar a informação das corridas e dos veículos. ??????????? ??????????? ??????????? ??????????? ??????????? ??????????? ??????????? ??????????? ??????????? ??????????? ???????????

# Conclução:

Neste projecto o principal objectivo 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.