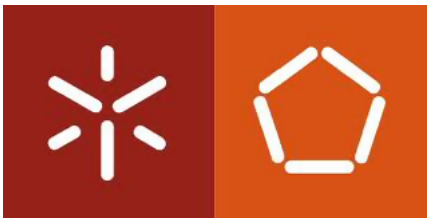


Escola de Engenharia

Departamento de Informática

Licenciatura em Engenharia Informática

Programação Orientada aos Objectos

Projecto Java - Racing Manager

A69854
Jo„o Mano

A66822
Miguel Guimar„es

A
Bruno Torres

Braga, Junho de 2014

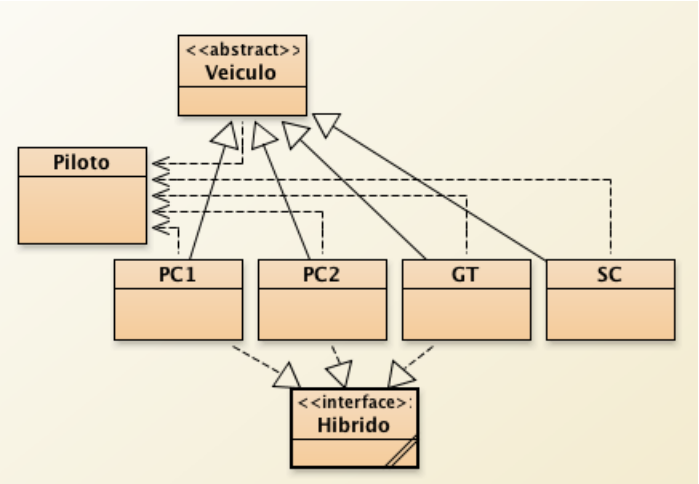
Conteúdo

1 Estrutura da aplicação

1.1 Classe abstracta Activity

Esta classe È a fundaÁ„o de todas as actividades da aplicaÁ„o, contÈm as vari·veis, *String name*, *GregorianCalendar date*, *double timeSpent* e *double calories*; e mÈtodos, *public String getName()*, *public GregorianCalendar getDate()*, *public double getTimeSpent()*, *public double getCalories()*, *public void setName(String name)*, *public abstract void setCalories(double peso)*, *public void setActivityCalories(double calories)*, comuns a todas.

1.2 Classe Abstracta Veiculo



Optou-se por uma classe abstracta pois, para além de permitir criar um "esqueleto" para as suas subclasses, reutilizando código, permite também obrigar as suas subclasses a implementar vários métodos necessários ao funcionamento da aplicação. Isto faz com que a aplicação seja desde logo extensível no toca a novas classes de veículos.

Se a opção fosse uma interface, as variáveis e métodos de instância de *Veículo* não podiam ser reutilizadas. Se não fosse uma classe abstracta, não obrigava as subclasses a implementar os métodos pretendidos.

Para além das variáveis de instância pedidas no enunciado, tem uma *String idVeiculo*, que serve como identidade e ordem dentro da aplicação, um *Piloto condutor* e um *int voltas* que representam o condutor actual do veículo e o número de voltas que esse condutor irá fazer numa dada corrida.

Para além dos construtores, *getters*, *setters* e *equals()*, obriga as suas subclasses a implementar os métodos:

- *clone()* e *toString()*;
- *int tempoProximaVolta()*, que calcula um desvio ao tempo médio por volta, em milisegundos;
- *boolean terminouVolta()*, que determina se um carro termina, ou não, uma dada volta de acordo com a sua fiabilidade e um factor aleatório;
- *int quantasVoltas()*, que determina o número de voltas a fazer pelo primeiro condutor do veiculo, de acordo com a qualidade dos dois pilotos;
- *void trocaCondutor()*, que troca para o segundo piloto, somando o tempo de paragem ao tempo da volta.

1.2.1 Classe PC1

Para além das variáveis e métodos herdados de *Veiculo*, tem um *static final int cilindrada = 6000*, pois os PC1 têm sempre $6000cm^3$ de cilindrada, um *int hibridoCv* para o motor híbrido (= 0 caso não tenha) e um *Random rand* para uniformizar a distribuição de probabilidade em *terminouVolta()*.

1.2.2 Classe PC2

Para além das variáveis e métodos herdados de *Veiculo*, tem um *int hibridoCv* para o motor híbrido (= 0 caso não tenha), um *int preparacao* que é a preparação mecânica do carro e que afecta a sua fiabilidade, e um *Random rand* para uniformizar a distribuição de probabilidade em *terminouVolta()*.

1.2.3 Classe GT

Para além das variáveis e métodos herdados de *Veiculo*, tem um *static final int fiabilidadeInicial = 100* que é a fiabilidade no início de cada corrida, um *int hibridoCv* para o motor híbrido (= 0 caso não tenha), uma *double taxaDecrescimo* que é a taxa a que a fiabilidade desce em cada volta, e um *Random rand* para uniformizar a distribuição de probabilidade em *terminouVolta()*.

1.2.4 Classe SC

Para além das variáveis e métodos herdados de *Veiculo*, tem um *static final int cilindrada = 2500*, pois os SC têm sempre $2500cm^3$ de cilindrada, e um *Random rand* para uniformizar a distribuição de probabilidade em *terminouVolta()*.

1.3 Interface Hibrido

As classes PC1, PC2 e GT implementam a interface *Hibrido*, que as obriga a implementar o método *int getPotenciaMotorElectrico()*, que retorna a potência do motor eléctrico em KW. Isto não faz com estas classes sejam obrigadas a ter uma variável de instância para o motor híbrido mas, como não é permitida herança múltipla de classes, não se pode criar uma superclasse *Hibrido*.

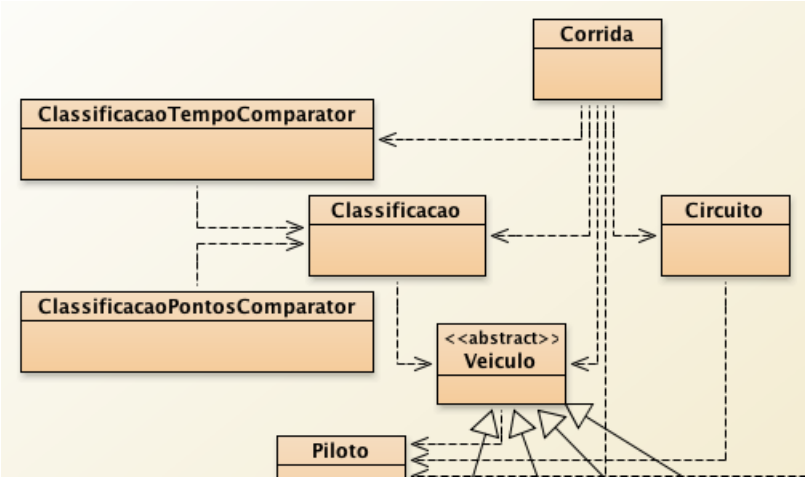
1.4 Classe Circuito

Para além das variáveis de instância pedidas no enunciado, esta classe apenas tem uma *String idCircuito*, que serve como identidade e ordem dentro da aplicação e um *String nome* do circuito. Todos os tempos estão em milisegundos, a resolução dos tempos na aplicação.

Para além dos construtores, *getters*, *setters*, *equals()*, *clone()* e *toString()*, implementa os métodos *void printCircuito()*, que imprime a informção do circuito e *String toTime(int millis)*, que recebe um tempo em milisegundos e retorna uma *String* em formato mins:segs:milis.

Os tempos médios por volta das categorias PC1, PC2, GT e SC são respectivamente, +2s, +3.5s, +5s e +6s do que o tempo record da pista.

1.5 Classe Corrida



Tem como variáveis de instância uma *String idCorrida*, que serve como identidade e ordem dentro da aplicação, um *Circuito circuito*, um *Map<String,Veiculo> veiculos* e uma *List<Classificação>classificacoes*.

Os veículos do campeonato são replicados para cada corrida para não terem de ser passados como parâmetro na aplicação. Como a dimensão dos dados é relativamente pequena isto não tem impacto no tempo de execução do programa.

Em relação ás classificações, optou-se por um List por forma a usar o *sort*, utilizando um *Comparator*.

Para além dos construtores, *getters*, *setters*, *equals()*, *clone()* e *toString()*, implementa os métodos *void efectuaCorrida(boolean chuva)*, que efectua a corrida, *void printClassificacao()*, que imprime a classificação da corrida, *String toTime(int millis)*, que recebe um tempo em milisegundos e retorna uma *String* em formato mins:segs:milis e *void reiniciarCorrida()*, que faz *reset* da corrida.

1.6 Classe Classificação

Como a aplicação só tem uma classificação geral, optou-se por adicionar uma variável para guradar os pontos dos veículos em vez de fazer uma nova classe para o efeito.

Tem como variáveis de instância um *Veiculo veiculo*, um *int tempo*, em milisegundos, um *boolean dnf*, que indica se terminou a corrida, um *int voltaDnf* que, em caso de não terminação da corrida, guarda a volta de desistência e um *int pontos*, para guardar os pontos do veículo.

Para além dos construtores, *getters*, *setters*, *equals()*, *clone()* e *toString()*, implementa apenas o método *void incrementaPontos(int pontos)*, que serve para ir somando os pontos do veículo ao longo do programa.

Como esta classe serve dois propósitos, classificar dentro de uma corrida, em função do tempo, e classificar no campeonato, em função dos pontos, são implementados dois *Comparators*, *ClassificacaoTempoComparator*, que ordena por ordem ascendente de tempo, e *ClassificacaoPontosComparator*, que ordena por ordem descendente de pontos.

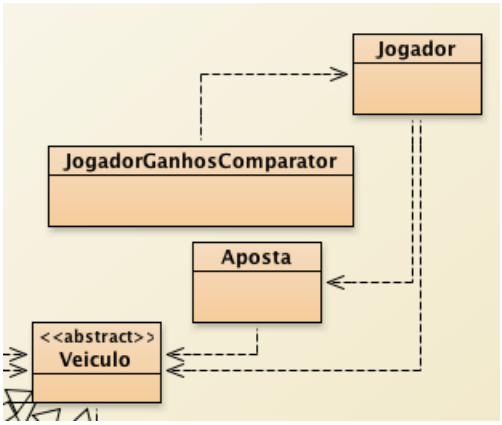
1.7 Classe Aposta

Tem como variáveis de instância uma *String idCorrida*, que serve como identidade e ordem dentro da aplicação, um *double montante*, três *Veiculo ouro, prata, bronze*, uma *double odd* e um *double ganho*.

Para além dos construtores, *getters*, *setters*, *equals()*, *clone()* e *toString()*, implementa apenas o método *double calculaOdd()*, que calcula a odd que a aposta deve ter de acordo com seus veículos.

O método *calculaOdd()* é chamado dentro do construtor da aposta, ficando logo definida a odd no momento da criação da aposta.

1.8 Classe Jogador



Para além das variáveis de instância pedidas no enunciado, esta classe apenas tem uma *String idJogador*, que serve como identidade e ordem dentro da aplicação.

Para além dos construtores, *getters*, *setters*, *equals()*, *clone()* e *toString()*, implementa apenas o método *void fazAposta(Aposta a)*, que coloca a aposta nas apostas correntes, desde que não haja uma aposta na mesma corrida nas apostas correntes ou nas apostas passadas.

Implementa ainda o método *actualizaAposta(String idCorrida, Veiculo ouro, Veiculo prata, Veiculo bronze)*, que processa a aposta.

Como a aplicação deve registar um *scoreboard* de apostadores, foi implementado um *JogadorGanhosComparator*, que ordena por ordem descendente de ganhos.

1.9 Classe Campeonato

Tem como variáveis de instância uma *String corridaActual*, que é a idCorrida de uma das corridas, um *Map<String,Corrida> corridas*, um *Map<String,Jogador> jogadores* e um *List<Classificacao> classificacaoGeral*.

Para além dos construtores, *getters*, *setters*, *equals()*, *clone()* e *toString()*, implementa os métodos:

- *void actualizaApostas(String idCorrida)*, que actualiza as apostas de uma corrida;
- *void printClassificacaoGeral()*, que imprime a classificacao geral do campeonato;
- *void printClassificacaoHibrido()*, que imprime a classificacao geral do campeonato, mas apenas dos carros híbridos;
- *void printScoreBoard()*, que imprime os três apostadores com mais ganhos no campeonato;
- *void reiniciarCampeonato()*, que faz *reset* do campeonat.

A estrutura de dados das corridas é um TreeMap, pois é necessário unicidade e ordem. A ordem é dada pela identidade da corrida.

1.10 Classe Simulação

Tem como variáveis de instância um *Campeonato campeonato*, um *Map<String,Piloto> pilotos*, um *Map<String,Veiculo> veiculos*, um *Map<String,Circuito> circuitos* e um *Map<String,Jogador> jogadores*.

Os menus têm todos o formato [n] <opção>, tendo o utilizador que premir o número da opção que pretende escolher. O menus são muito interactivos pois pode-se sempre voltar a um menu anterior e têm uma interface muito simples.

Por exemplo, Carregar Campeonto -> Retomar Campeonato -> Menu Principal -> Sair:

```
------ Racing Manager -------


[1] Novo Campeonato
[2] Carregar Campeonato
[3] Sair
2

Nome do ficheiro: campeonato.ser

[1] Retomar Campeonato
[2] Reiniciar Campeonato
[3] Menu Principal
1


[1] Classificação Geral                 [4] Efectuar Corrida           [7] Apostar
[2] Classificação Híbrido               [5] Ver Jogadores              [8] Gravar Campeonato
[3] Classificação da corrida N (1..20)  [6] ScoreBoard Apostadores     [9] Menu Principal
9


------ Racing Manager -------


[1] Novo Campeonato
[2] Carregar Campeonato
[3] Sair
3
```

Execução de uma corrida, aqui sem o primeiro classificado e desistências de cada volta:

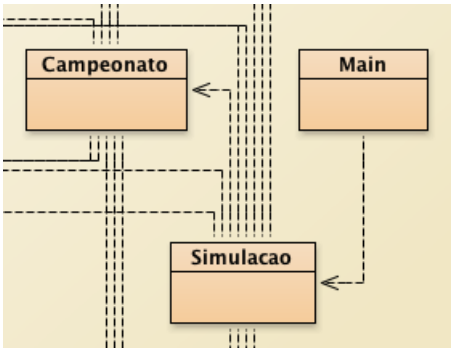
```
Circuito: Interlagos || Distância: 4.309 || Voltas: 10
Record: 1:10:229 por Rubens Barrichello (Brasil)

[1] Piso Seco    [2] Piso Molhado
1
```

	Classe	Marca	Modelo	Piloto1	Piloto2	Tempo	
1	PC1	Jaguar	XJR-9 LM	Keiichi Tsuchiya	Jim Clark	11:23:727	
2	PC2	Nissan	R90CP	Jacky Ickx	Juan Manuel Fangio	11:39:940	
3	GT	Ferrari	F2004	Michael Schumacher	Rubens Barrichello	11:56:293	
4	GT	Ferrari	312T2/B	Gilles Villeneuve	Ronnie Peterson	11:57:188	
5	SC	McLaren	MP4/4	Ayrton Senna	Alain Prost	11:59:470	
6	GT	Nissan	R92CP	Alan Jones	Giuseppe Farina	12:2:680	
7	SC	Ferrari	F2008	Kimi Räikkönen	Felipe Massa	12:8:529	
8	PC1	Sauber	C9	Stefan Bellof	Clay Regazzoni	12:27:650	
9	GT	Toyota	GT-ONE SE	Phil Hill	Carlos Reutemann	12:36:148	
10	GT	Lancia	LC2	Derek Bell	Frank Biela	12:56:990	
11	GT	Matra	MS670	Alberto Ascari	Lewis Hamilton	13:0:229	
12	GT	Ferrari	312PB	Juichi Wakisaka	Damon Hill	13:0:832	
13	SC	Toyota	Eagle MkIII	Dan Gurney	P.J. Jones	13:10:603	
14	SC	Renault-Alpine	A442B	Graham Hill	John Surtees	13:11:431	
15	SC	Mazda	78B7	Riccardo Patrese	Tom Kristensen	13:11:513	
16	SC	Porsche	956/962	James Hunt	Jose Froilan Gonzalez	13:12:607	
17	PC2	Porsche	936	Emerson Fittipaldi	Nelson Piquet	DNF	1
18	PC1	Audi	R10 TDI	André Lotterer	Benoît Tréluyer	DNF	2
19	PC1	Ford	GT40 MK4	Woolf Barnato	Marcel Fässler	DNF	2
20	PC1	Porsche	917	Francois Cevert	Jenson Button	DNF	3
21	PC2	Red Bull	RB7	Sebastian Vettel	Mark Webber	DNF	4
22	PC2	Audi	R8C	Jackie Stewart	Jim Clark	DNF	4
23	PC1	McLaren	F1	Henri Pescarolo	Yannick Dalmas	DNF	4
24	PC1	BMW	V12 LMR	Garhard Berger	Jean Behra	DNF	5
25	PC2	Ferrari	330 P4	Jack Brabham	Mario Andretti	DNF	5
26	PC2	Porsche	911 GT1-98	Loïc Duval	Marco Werner	DNF	6
27	GT	Peugeot	908 HDi FAP	Marc Gené	Jacques Villeneuve	DNF	6
28	PC1	Saleen	S7	Emanuele Pirro	Olivier Gendebien	DNF	7
29	SC	McLaren	MP4/2C	Nigel Mansell	Keke Rosberg	DNF	7
30	PC2	Bentley	SPEED 8	Fernando Alonso	The Stig	DNF	8
31	SC	Porsche	935	Jochen Rindt	Mika Hakkinen	DNF	9
32	PC2	Toyota	Minolta CV-88	Niki Lauda	Stirling Moss	DNF	10

```
[1] Classificação Geral                 [4] Efectuar Corrida          [7] Apostar
[2] Classificação Híbrido               [5] Ver Jogadores             [8] Gravar Campeonato
[3] Classificação da corrida N (1..20)  [6] ScoreBoard Apostadores    [9] Menu Principal
```

1.11 Classe Main

Apenas cria uma simulação e chama o menu principal da aplicação.



2 Decisões a Destacar

2.1 Fiabilidades

As fiabilidades inicialmente propostas eram, de facto, baixas, o que resultava em corridas com três ou quatro finalistas. Assim sendo, tiveram um aumento à volta dos 10%.

PC1 - 95% e, se tiver um motor híbrido, 90%.

PC2 - 85% a 95% em função da cilindrada e da preparação mecânica do carro. Se tiver um motor híbrido, 80% a 90%.

GT - 100% de fiabilidade inicial, que decresce em cada volta de acordo com uma taxa definida pelo utilizador. Se tiver um motor híbrido, a fiabilidade inicial passa a ser 95%.

SC - 80% a 95%, influenciada em 25% pela cilindrada (constante, igual a $2500cm^3$) e em 75% pela qualidade dos pilotos.

2.2 Motores híbridos

Optou-se por um limite de cavalagem nos motores hibridos de 300 CV. A razão desta escolha deve-se a uma pesquisa sobre este tipo de motores, que revelou valores máximos próximos deste valor.

2.3 Tempos de Voltas

Optou-se por fazer com que cada um de 4 atributos - cilindrada, cavalagem, fiabilidade e qualidade do condutor - influencie em até 1s no desvio resultante. A isto soma-se 1s e ainda um factor aleatório que aumenta/diminui estes 5s em até 20%.

O desvio ao tempo médio retornado pelo método *tempoProximaVolta(boolean chuva)* está no intervalo [-6,6] segundos, sem contar com uma possível redução do tempo causada pelos motores híbridos, na ordem dos 10%.

Para determinar se o desvio ao tempo médio é positivo ou negativo, olha-se para a cilindrada, cavalagem e qualidade do condutor: se mais metade destes atributos for maior do que o ponto médio dos valores possíveis destes atributos, então o desvio será negativo. Casso contrário, será positivo.

O facto de os desvio estarem neste intervalo faz com que todas as classes possam bater o record das pistas.

2.4 Odds

O cálculo da odd de uma aposta é feito no momento da sua criação. Os atributos que contribuem para a odd são: a média das qualidades dos pilotos, a média das fiabilidades dos veículos, a média das cilindradas e a média das cavalagens.

Cada um destes atributos tem um peso de até 2.5 e as odds ficam assim compreendidas no intervalo [1.15,10].

2.5 Limitações

Limitaram-se certas variáveis de instância a gamas de valores realistas e que, ao serem usados em métodos da aplicação, produzirem valores com sentido. Por isso, grandezas como distâncias e tempos têm que ser não-negativos.

Um exemplo é o da cavalagem, que está compreendida entre 0 e 1200 *CV*. Isto tem o objectivo de não deixar o utilizador inserir valor absurdos na aplicação.

Outra medida com o mesmo sentido foi o de fazer com a simulação fosse "à prova de bala- embora os menus tenham todos como opções valores inteiros, a aplicação recebe uma *String*. Assim, o utilizador pode escrever valores diferentes dos pretendidos e a aplicação não gerar erro.

3 Extensionabilidade

Com a classe abstracta *Veiculo* a aplicação é extensível a outro tipo de veículos. Mas na prática não é extensível, pelo menos totalmente.

Quando se quer criar um campeonato, ao criar um veículo é preciso especificar a sua classe por forma a chamar o seu construtor. Ora, se alguém que não tenha acesso ao módulo *Simulação* quiser uma nova classe de veículo, não a pode construir em ambiente de consola.

Outro caso está na classe *Circuito*: no enunciado foi pedido um tempo médio para cada classe de carro, que é usado no método *efectuaCorrida()* para somar, em cada volta, ao desvio dado por *tempoProximaVolta()*.

Mais uma vez, se um utilizador sem acesso ao módulo *Circuito* quisesse uma nova classe de veículos, não poderia especificar o seu tempo médio nos circuitos.