FICHA 05 –PROGRAMAÇÃO ORIENTADA A OBJECTOS (HERANÇA)

Um programa em Java é geralmente constituído por vários objectos de várias classes organizadas hierarquicamente.

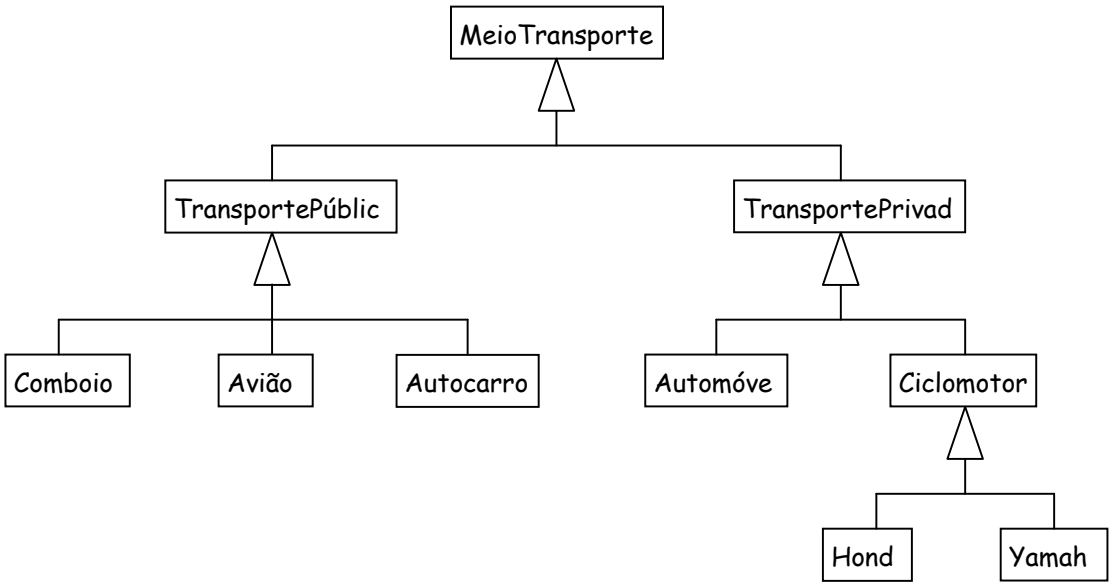
1. Hereditariedade

Se olharmos para o mundo real, os objectos como as pessoas, os animais, as bicicletas, etc., são muitas vezes "agrupados" de acordo com determinadas características e comportamentos. Por exemplo, pessoas e animais são objectos animados, em oposição às bicicletas que são objectos inanimados.
Existem grupos de objectos que além de possuírem as suas características específicas, possuem características comuns a outros grupos. Os humanos, por exemplo, pertencem aos mamíferos, pois exprimem características e comportamentos próprios dos mamíferos, apesar de possuírem as suas características próprias. Dito de outro modo, um humano é um ser humano, mas também é um mamífero.
Em linguagens orientadas a objectos a técnica que nos permite a partir de uma classe utilizá-la na construção de novas classes é denominada hereditariedade, uma vez que a organização das classes é feita hierarquicamente.

1.1. Super classe e classes derivadas

A hereditariedade é a capacidade que uma classe, chamada **classe derivada** ou **classe descendente**, tem de adquirir as características e comportamentos de outra classe, chamada **classe base**, **super classe** ou **classe antecessora**.
O modo mais correcto de usar a hereditariedade é criar uma hierarquia de classes em árvore, que coloca o conceito mais genérico na raiz e os conceitos mais específicos nas folhas.

Na figura mostra-se um exemplo de uma estrutura hierárquica, que descreve vários tipos de transporte:



É importante compreender a relação entre as várias classes:

• Começando das folhas para a raiz, cada subclasse também é um dos elementos da super classe (do mesmo modo que um humano também é um dos elementos dos mamíferos);

• um Autocarro é um TransportePúblico, que, por sua vez, é um meio de Transporte;

• um Autocarro é também, obviamente, um meio de Transporte.

1.2 Implementação em Java

A linguagem Java utiliza a palavra **extends** para exprimir a noção de herança. Assim, se se pretender criar uma classe B que seja derivada de uma classe A, pode fazer-se:

B extends A

Voltando ao exemplo dos meios de transporte, uma implementação das classes MeioTransporte, TransportePublico e Comboio poderia ser:

```
class MeioTransporte {
  String condutor = "Zé";
  void movimenta() {
    System.out.println("Em movimento!!");
  }
}

class TransportePublico extends MeioTransporte {
  String[] passageiros = {"Xico, Manel, João"};
  void picaBilhete() {
    System.out.println("\nA picar um bilhete!!");
  }
}

class Comboio extends TransportePublico {
  int numCarruagens = 5;
  void apita() {
    System.out.println("tutuuu!!");
  }
}
```

1.3 Herança de atributos e métodos

O código acima implementa as classes MeioTransporte, TransportePublico e Comboio. Relativamente aos atributos e métodos que são herdados refere-se:

- Um TransportePublico é um MeioTransporte; por esta razão herda deste um condutor e o método movimenta();
- um Comboio é um TransportePublico, herdando deste a capacidade de transportar passageiros e o método picaBilhete(),
- mas é também um MeioTransporte, pelo que herda ainda um condutor e o método movimenta().

Para testar o mecanismo de herança, apresenta-se o seguinte código:

```
    // (…)
    System.out.println("MeioTransporte:");
    MeioTransporte m = new MeioTransporte();
    System.out.println("condutor " + m.condutor);
    m.movimenta();

    System.out.println("\n\nTransportePublico:");
    TransportePublico t = new TransportePublico();
    System.out.println("condutor " + t.condutor);
    t.movimenta();
    //mas também:
```

```
    System.out.println("passageiros:");
    for (int i = 0; i < t.passageiros.length; i++)
      System.out.print(t.passageiros[i] +  "     ");
    t.picaBilhete();

    System.out.println("\n\nComboio:");
    Comboio c = new Comboio();
    System.out.println("condutor " + c.condutor);
    c.movimenta();
    System.out.println("passageiros:");
    for (int i = 0; i < c.passageiros.length; i++)
      System.out.print(c.passageiros[i] +  "     ");
    c.picaBilhete();
    //.. e ainda:
    c.apita();
    System.out.println("num de carruagens " + c.numCarruagens);
    // (…)
```

A execução do código acima tem o seguinte resultado:

```
MeioTransporte:
condutor Zé
Em movimento!!

TransportePublico:
condutor Zé
Em movimento!!
passageiros:
Xico, Manel, João
A picar um bilhete!!

Comboio:
condutor Zé
Em movimento!!
passageiros:
Xico, Manel, João
A picar um bilhete!!
tutuuu!!
num de carruagens 5
```

2. A palavra reservada super

A palavra reservada super está disponível em todos os métodos não static de uma classe descendente. É utilizada no acesso às variáveis e métodos membro e funciona como uma referência para o objecto corrente enquanto instância da sua super classe.
A chamada super.método invoca sempre a implementação do método da super classe e não a versão do método implementada pela classe corrente.
Por exemplo, acrescentado a cada uma das classes TransportePublico e Comboio um método específico void movimenta() e o método void testaSuper() à classe Comboio:

```
class Comboio extends TransportePublico {
  //(…)
  void testaSuper() {
    System.out.println("vou chamar o movimenta(): ");
    movimenta();
    System.out.println("vou chamar o super.movimenta(): ");
    super.movimenta();
  }
  //(…)
}
```

e agora executando o seguinte código:

```
    Comboio c = new Comboio();
    c.testaSuper();
```

Obtemos o seguinte resultado:

```
vou chamar o movimenta():
Comboio em movimento!!
vou chamar o super.movimenta():
TransportePublico em movimento!!
```

2.1 A palavra reservada super nos construtores

Tal como a palavra reservada **this**, a palavra reservada **super** assume um significado especial nos construtores:

• assim, **super()** serve para invocar explicitamente o construtor por omissão da super classe. Da mesma forma podem-se invocar os restantes construtores, passando os argumentos necessários.

• Caso se queira usar a invocação explícita de um dos construtores da super classe tem que ser a primeira instrução do construtor da classe descendente.

• O Java invoca automaticamente o construtor por omissão da super classe, caso não se invoque explicitamente nenhum dos construtores disponíveis.

Vamos adicionar os seguintes construtores a TransportePublico e a Comboio:

```
class TransportePublico extends MeioTransporte {
  String[] passageiros = {"Xico, Manel, João"};
```

```
  TransportePublico() {
    passageiros = new String[2];
    passageiros[0] = "Josefa";
    passageiros[1] = "Maria";
  }

  TransportePublico(String[] p) {
    passageiros = new String[p.length];
    passageiros = p;
  }
  // (…)
}

class Comboio extends TransportePublico {
  int numCarruagens = 5;

  Comboio() {}
  Comboio(String[] passageiros) {
    super(passageiros);
  }
  // (…)
}
```

Agora vamos executar o seguinte código:

```
    // (…)
    Comboio c = new Comboio();
    System.out.println("passageiros do Comboio():");
    for (int i = 0; i < c.passageiros.length; i++)
      System.out.print(c.passageiros[i] +  "     ");
    System.out.println();

    String[] p = {"Carlo", "Felizberto", "Matias"};;
    Comboio b = new Comboio(p);
    System.out.println("\npassageiros do Comboio(p):");
    for (int i = 0; i < b.passageiros.length; i++)
      System.out.print(b.passageiros[i] +  "     ");
    System.out.println();
    // (…)
```

Obtemos então como resultado:

```
passageiros do Comboio():
Josefa     Maria

passageiros do Comboio(p):
Carlo     Felizberto     Matias
```

3. A palavra reservada final aplicada a classes e métodos

Definir um método como final impede as classes descendentes da classe onde foi definido de fazer a sua própria versão do método. Dito de outro modo, faz com que essa seja a última definição possível para o método.
Uma classe marcada como final não pode ter descendentes. O que faz com que todos métodos nela definidos sejam também implicitamente final.
Para definir uma classe ou método como final, basta colocar a palavra reservada final no início da declaração.
Por exemplo:

```
class Aviao extends TransportePublico {
  final void descola() {
    System.out.println("Avião a descolar!!");
  }
  final void aterra() {
    System.out.println("Avião a aterrar!!");
  }
}

final class Autocarro extends TransportePublico {
  int minutosAtraso;
}
```

tem como consequência:

• que os métodos descola e aterra são considerados final. Assim as classes descendentes da classe Avião podem, na melhor das hipóteses, usar a definição que já existe; podem, no entanto, alterar as definições dos restantes métodos (herdados da classe TransportePublico).

• A classe Autocarro, porque foi tornada final, não pode agora ter descendentes.

Exercícios

1. Futebol

Imagine que pretende desenvolver um programa para gerir os dados das equipas do campeonato nacional de futebol. Cada equipa contém um máximo de 23 jogadores, cada um com um nome e um número de camisola, que poderão ser de 4 tipos diferentes:

- guarda-redes, a que corresponde um número de golos sofridos
- defesa, a que corresponde um número de recuperações de bola e um número de faltas cometidas
- médio, a que corresponde um número de golos marcados e um número de assistências.
- avançado, a que corresponde um número de golos marcados.

a) Construa um diagrama com as classes necessárias à criação de equipas para o campeonato.

b) Elabore uma classe Equipa de acordo com o descrito acima, bem como as classes associadas aos tipos de jogador diferentes.

c) Pretende-se agora que crie uma equipa "maravilha", ou seja, para cada número de camisola, o jogador que tenha a melhor estatística, calculada assim:

1. Guarda-redes – número de golos sofridos
2. Defesa:
    i. (2 * recuperações)/faltas cometidas (se faltas cometidas != 0)
    ii. (4 * recuperações) (se faltas cometidas == 0)
3. Médio – golos marcados * 1,5 + assistências
4. Avançado – número de golos marcados

Construa um método que crie uma equipa "maravilha" com 11 jogadores tendo em consideração as definições anteriores.

2. Banco

Imagine que para a gestão informatizada de um banco está devidamente armazenada informação sobre as contas dos seus clientes. Para cada cliente sabe-se o nome, o número e o tipo de conta (à ordem ou a prazo) e o respectivo saldo. Faça um programa que permita fazer repetidamente as seguintes operações:

a) Levantamentos de uma conta - só se houver saldo;

b) Depósitos numa conta - sempre possíveis;

c) Consulta do saldo de uma conta - sempre possível;

d) Os clientes que têm contas com um saldo superior a x;

e) Quantos dias faltam para vencer o juro de uma conta a prazo (anual).

3. Inscrições

Pretende-se uma aplicação informática para facilitar a gestão das inscrições nas turmas práticas de um curso de uma instituição de ensino superior, como o DEI. Cada aluno é identificado através do seu nome e número de aluno e encontra-se inscrito numa lista de disciplinas (todas do mesmo ano). Cada disciplina pode ter várias turmas práticas. A aplicação deverá permitir:

a) mostrar a lista de disciplinas de um determinado ano,

b) mostrar a lista de todas as turmas de uma disciplina,

c) mostrar a lista de todos os alunos inscritos numa turma de uma disciplina, num determinado momento

d) inscrever um aluno numa turma de uma disciplina (se houver vaga),

e) retirar um aluno de uma turma de uma disciplina.

4. Concurso (problema para avaliação)

Uma cadeia de supermercados decidiu organizar um concurso para comemorar os seus 15 anos de existência. O concurso destina-se a funcionários e clientes e é semelhante ao conhecido Totoloto sem recurso ao número suplementar: uma chave é constituída por 4 números de um universo de 10. Cada funcionário pode fazer 5 apostas, enquanto que cada cliente tem direito a 3 apostas. Um jogador (funcionário ou cliente) é identificado pelo respectivo nome e idade. Enquanto que para um cliente é necessário saber o seu telefone, um funcionário é ainda identificado pelo seu número interno na empresa. Os dados relativos a cada jogador (funcionário ou cliente), bem como as apostas/chaves de todos os jogadores devem ser armazenados de forma adequada. Desenvolva as classes necessárias para utilizar num programa que implemente esta situação.