# FICHA 07 –PROGRAMAÇÃO ORIENTADA A OBJECTOS (POLIMORFISMO)

## 1. Polimorfismo

Uma das facetas mais importantes da herança é a relação que se estabelece entre uma super classe e uma classe descendente. Uma classe descendente é uma das variantes possíveis para os membros da super classe.

Essa relação especial tem reflexos no modo como os métodos e variáveis existentes na super classe passam para a classe descendente e no modo como um elemento de uma classe descendente se pode transformar num membro da classe ascendente.

Voltando ao exemplo dos mamíferos, humanos e elefantes, sendo mamíferos, têm características semelhantes, como poder amamentar, mas diferem, por exemplo, no modo como andam. O mesmo comportamento (andar) difere nos humanos, que se deslocam na posição vertical, e nos elefantes, que se deslocam com as quatro patas.

**Polimorfismo** quer dizer muitas formas. Quando aplicado à orientação a objectos quer dizer que um mesmo método pode ter muitas formas. Sendo um método reconhecido pelo seu nome e pelo tipo dos seus argumentos (a sua **assinatura**), quer isto dizer que à mesma assinatura podem corresponder várias implementações, dependendo do objecto que está a executar o método.
Se, usando de novo o exemplo dos transportes, redefinirmos em cada uma das classes os métodos herdados, obtemos:

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

class TransportePublico extends MeioTransporte {
  String[] passageiros = {"Xico, Manel, João"};
  //herdado
  void movimenta() {
    System.out.println("TransportePublico em movimento!!");
  }
  void picaBilhete() {
    System.out.println("\nA validar um bilhete de um TransportePublico!!");
```

```
  }
}

class Comboio extends TransportePublico {
  int numCarruagens = 5;
  //herdado
  void movimenta() {
    System.out.println("Comboio em movimento!!");
  }
  void picaBilhete() {
    System.out.println("\nA validar um bilhete de Comboio!!");
  }
  void apita() {
    System.out.println("tutuuu!!");
  }
}
```

Se executarmos o código:

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

    System.out.println("\n\n--------TransportePublico:--------");
    TransportePublico t = new TransportePublico();
    System.out.println("condutor " + t.condutor);
    t.movimenta();
    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\n----------Comboio:----------");
    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();
    c.apita();
    System.out.println("num de carruagens " + c.numCarruagens);
```

Obtemos o resultado:

```
-----------MeioTransporte:-------------
condutor Zé
MeioTransporte em movimento!!

-----------TransportePublico:----------
condutor Zé
TransportePublico em movimento!!
passageiros:
Xico, Manel, João
A validar um bilhete de um TransportePublico!!

-----------Comboio:-----------
condutor Zé
Comboio em movimento!!
passageiros:
Xico, Manel, João
```

```
A validar um bilhete de Comboio!!
tutuuu!!
num de carruagens 5
```

De acordo com a classe do objecto que executa cada um dos métodos a versão correspondente é executada.

Além disso, existe ainda a possibilidade de um mesmo objecto ter várias versões de um método, desde que a assinatura seja diferente. O que quer dizer na prática que podemos ter vários métodos com o mesmo nome, desde que os tipos dos argumentos sejam diferentes (ver overloading de métodos).

## 2. Upcasting e downcasting

O **upcasting** e o **downcasting** são os mecanismos que nos permitem tratar uma instância de uma classe descendente como se fosse uma instância de uma classe ascendente, e vice-versa.

### Upcasting

Já vimos que um membro de uma classe descendente pode ser sempre também visto como um membro da sua super classe. Tomemos como exemplo a classe dos instrumentos musicais e a sua subclasse instrumentos de sopro:

```
class Musica {
  public static void main (String[] args) {
    Sopro flauta = new Sopro();
    // upcasting: flauta é um Sopro, mas é passado ao método afina,
    // que espera um Instrumento
    Instrumento.afina(flauta);
  }
}

class Instrumento {
  static void afina (Instrumento i) { //método da classe
    System.out.println("A afinar um " + i);
    i.toca();
  }
  public void toca () { //método das instâncias
    System.out.println("A tocar um " + this);
  }
}

// Instrumentos de sopro. Têm os mesmos métodos que os outros Instrumentos.
class Sopro extends Instrumento {
}
```

A classe Instrumento possui um método de classe chamado afina que recebe um Instrumento e possui ainda um método de instância toca. Na classe Musica é criado um

instrumento de sopro, uma flauta, e é chamado o método da classe Instrumento, afina(), mas em vez de se passar o **handle** de um Instrumento, passa-se o handle de um Sopro (a flauta). O Java sabendo que um Sopro é um tipo de Instrumento, e portanto possui tudo o que um Instrumento deve possuir, converte automaticamente o handle do Sopro num handle de Instrumento e o código funciona sem problemas.

Chama-se **upcasting** ao acto de converter um handle de uma classe descendente num handle de uma classe superior.

Curiosamente, apesar de ao chamar o afina se converter implicitamente a flauta num handle de um Instrumento, a flauta em si continua a ser um Sopro, e, por isso, quando se executa o código do main de Musica obtém-se:

```
A afinar um Sopro@1ee7f9
A tocar um Sopro@1ee7f9
```

Como um objecto de uma classe descendente possui tudo o que possui um objecto da sua super classe (por herança desta) é sempre possível fazer o upcasting. Está-se simplesmente a passar de um tipo mais específico para um tipo mais geral. O objecto do tipo descendente pode conter mais atributos e mais métodos, mas não pode conter menos. Daí que não seja necessária nenhuma notação especial ou cast explícito para se fazer o upcasting.

### Downcasting

O contrário de upcasting é o **downcasting** e consiste em transformar um handle de uma classe mais genérica (ascendente) numa classe mais específica (descendente).

O downcasting também é possível, mas só nalguns casos, nomeadamente se o handle apontar na realidade para um objecto do tipo mais específico. Por exemplo:

```
    Instrumento f = new Sopro();
    Sopro s = (Sopro) f; //funciona
    Instrumento g = new Instrumento();
    Sopro t = (Sopro) g; //vai dar origem a uma excepção
```

Criamos um Sopro, mas passamos a referi-lo com um handle de um Instrumento (upcasting). Mais à frente voltamos a reconvertê-lo num Sopro, s, (downcasting). Não há problema porque f é na realidade um Sopro, por isso, garantidamente possui tudo o que um Sopro deve possuir. Quando tentamos fazer o mesmo com um Instrumento, no entanto, não funciona do mesmo modo. Na realidade a execução do código dará origem

a uma excepção. Isto porque um objecto da classe Sopro, pertencendo a uma classe descendente, pode possuir mais métodos e atributos do que uma instância da classe Instrumento. Se assim for, como no caso de g, que na realidade aponta para um Instrumento e não possui a informação adicional que um Sopro deveria possuir, dá-se uma excepção. Dito de outra forma, embora seja verdade que um Sopro é sempre um Instrumento, nem sempre um Instrumento é um Sopro.

## Exercícios

### 1. Futebol

Considere a estrutura de classes criada para o exercício da Ficha 05, relativa à gestão de dados das equipas do campeonato nacional de futebol.

a) Certifique-se que cada equipa tem uma tabela de 23 jogadores, dos diferentes tipos (Guarda-redes, Defesa, Médio e Avançado), correspondentes a todos os jogadores da equipa.

b) Acrescente à classe Jogador, um método que devolve a sua pontuação, de acordo com as fórmulas definidas, e outro que devolve o tipo de jogador, de acordo com a seguinte tabela:

| Jogador | Tipo |
|---|---|
| Guarda-redes | 1 |
| Defesa | 2 |
| Médio | 3 |
| Avançado | 4 |

c) Acrescente à classe Equipa, um método que devolva o jogador mais valioso da equipa.

d) Acrescente ainda à classe Equipa, um método que devolva o jogador mais valioso da Equipa, consoante o Tipo solicitado.

e) Crie agora a classe Campeonato, sabendo que este é composto por 16 equipas.

f) Acrescente à classe campeonato os seguintes métodos:

1. Um método que devolva o melhor jogador do campeonato;

2. Um método que devolve o melhor jogador do campeonato, consoante o tipo (tabela da alínea a);

3. Um método que devolve uma equipa "selecção", composta pelos melhores jogadores do campeonato, constituída pelos seguintes jogadores:

| Tipo de Jogador | Quantidade |
|---|---|
| Guarda-redes | 3 |
| Defesas | 8 |
| Médios | 8 |
| Avançados | 4 |

## 2. Biblioteca

Numa biblioteca de um departamento universitário os livros estão organizados por áreas. Cada livro é descrito por um título, autor(es) e uma cota. Os leitores são essencialmente de dois tipos: professores e alunos. Os professores podem requisitar até um máximo de 10 livros por um prazo máximo de 2 meses. Os alunos podem requisitar apenas até 3 livros, por um prazo máximo de 8 dias. Além dos dados usuais (nome, morada, idade, telefone, email) a ficha de cada leitor tem o número e tipo de leitor e ainda informação sobre os livros que tem requisitados num determinado momento - cota e data de requisição. Desenvolva um programa que permita:

a) consultar os livros da biblioteca, por área e por autor;

b) requisitar um livro, usando a cota como chave de acesso;

c) vizualizar os livros que necessitam de ser entregues (cujo prazo de entrega já terminou).

## 3. Empresa (problema para avaliação)

Uma empresa possui dois tipos de empregados: os que trabalham à comissão e os que trabalham à hora. Cada empregado é caracterizado por possuir um nome e um número. Os ordenados dos empregados que trabalham à comissão são calculados adicionando ao ordenado base o total de vendas que efectuaram multiplicado pela sua percentagem

de comissão. Os ordenados dos empregados que trabalham à hora são calculados multiplicando o número de horas que trabalharam pelo preço da hora de trabalho. Pretende-se que desenvolva um programa que permita:

a) actualizar os dados diários de um empregado conforme o seu tipo;

b) vizualizar os dados de um empregado ao longo de um mês;

c) implementar e vizualizar a contabilidade mensal (ordenados a pagar) da empresa.