Polimorfismo

- O polimorfismo é um outro conceito central em programação orientada a objectos
- Uma referência polimórfica é aquela que se pode referir a um de vários possíveis métodos
  - Imaginemos que a classe Ferias tem um método chamado celebrar e a classe Verao (descendente de Ferias) lhe sobrepõe um outro método (com o mesmo nome e parâmetros, claro)
  - A instrução dia.celebrar(); qual das duas versões invocará?
  - Se dia referenciar um objecto da classe Ferias será a versão dessa classe, mas se dia referenciar um objecto da classe Verao será a versão respectiva

Polimorfismo

- Em geral, é o tipo do objecto (e não o tipo da referência) que define qual o método que é invocado
- Considerando de novo as classes Pensamento e Conselho já utilizadas anteriormente:

```
class Pensamento {
public void mensagem() {
     System.out.println ("Aproxima-se a Queima das Fitas...");
     }
} // class Pensamento
class Conselho extends Pensamento {
  public void mensagem() {
     System.out.println ("Atenção, Junho está à porta....");
     }
} // class Conselho
```

Polimorfismo

```
class Mensagens {
public static void main (String[] args) {
      Pensamento pensa = new Pensamento();
      Conselho exame = new Conselho();
      pensa.mensagem();
      exame.mensagem();
      pensa = exame;
      pensa.mensagem();
   }
} // class Mensagens
```

- Escreverá:

```
Aproxima-se a Queima das Fitas...
Atenção, Junho está à porta....
Atenção, Junho está à porta....
```


Polimorfismo

- É de notar que, caso a invocação polimórfica de um método esteja dentro de um ciclo, é possível que a mesma linha de código invoque métodos diferentes em momentos (iterações do ciclo) diferentes
- Assim, as referências polimórficas são definidas no momento da execução e não no momento da compilação

Exemplo

- Imaginemos que queríamos implementar um programa de gestão de stocks de uma loja de material fotográfico
- Vamos imaginar que esta loja vende lentes, filmes e câmaras fotográficas
- O nosso sistema necessita de guardar informação sobre os diversos itens:
  - As lentes têm uma distância focal e podem ou não ter zoom
  - Os filmes têm uma sensibilidade e um número de fotos
  - As câmaras podem vir ou não com lente, têm uma dada velocidade máxima e uma dada cor

Exemplo

- Todos os elementos necessitam ainda de:
  - Descrição do item
  - Um código identificador
  - A quantidade em stock
  - O preço do item
- Precisamos de criar classes que representem cada tipo de item, guardando as informações respectivas nas suas variáveis de instância

Exemplo

```
class Lente {
    private String descricao;
    private int numInvent;
    private int quantidade;
    private int preco;
    private boolean temZoom;
    private double distFocal;

    public Lente (...) {…}; //Construtor
    public String getDescricao () {…};
    public int getQuantidade () {…};
    public int getPreco() {…};
    ...
    //Métodos específicos de Lente
    ...
}
```


Exemplo

```
class Filme {
    private String descricao;
    private int numInvent;
    private int quantidade;
    private int preco;
    private int sensibilidade;
    private int numFotos;

    public Filme (...) {…}; //Construtor
    public String getDescricao () {…};
    public int getQuantidade () {…};
    public int getPreco() {…};
    ...
    //Métodos específicos de Filme
    ...
}
```

Exemplo

```
class Camara {
    private String descricao;
    private int numInvent;
    private int quantidade;
    private int preco;
    private boolean temLentes;
    private int velMaxima;
    private String cor;

    public Camara (...) {...}; //Construtor
    public String getDescricao () {...};
    public int getQuantidade () {...};
    public int getPreco() {...};
    ...
    //Métodos específicos de Camara
    ...
}
```


Exemplo

- Como se pode ver há uma grande sobreposição entre as classes, já que têm diversas variáveis e métodos comuns
  - Se for necessário guardar mais alguma informação comum aos três itens será necessário adicioná-la às três classes
- Cada uma das classes modela dois comportamentos relacionados, mas distintos:
  - Um elemento de inventário
  - Um elemento específico
- A herança pode ajudar a simplificar esta situação
- Podemos criar uma super classe que contenha os elementos comuns, derivando depois as classes específicas a partir desta

Exemplo

```
class ItemInventario {
    public ItemInventario (...) {…}; //Construtor
    public String getDescricao () {…};
    public int getQuantidade () {…};
    public int getPreco() {…};
    ...
    private String descricao;
    private int numInvent;
    private int quantidade;
    private int preco;
}
```

- Esta classe não "sabe" nada sobre as especificidades de cada item, mas é responsável pelo comportamento comum a todos os itens existentes no inventário

Exemplo

- Podemos agora definir as sub classes mais específicas:

```
class Lente extends ItemInventario{
    private boolean temZoom;
    private int distFocal;

    public Lente (...) {…}; //Construtor
    ...
    //Métodos específicos de Lente
    public String getDescricao () {
      return super.getDescrição ()+ "Dist Focal = "+distFocal;
    };
}
```

Exemplo

```
class Filme extends ItemInventario{
    private int sensibilidade;
    private int numFotos;

    public Filme (...) {...}; //Construtor

    public String getDescricao () {
      return super.getDescrição ()+ "Num. Fotos = "+numFotos;
    };
}
```


Exemplo

```
class Camara extends ItemInventario{
    private boolean temLentes;
    private int velMaxima;
    private String cor;

    public Camara (...) {...}; //Construtor

    public String getDescricao () {
      return super.getDescrição ()+ "Cor = "+cor;
    };
}
```

Exemplo

- Estas três classes modelam as especificidades e herdam o comportamento comum a partir de ItemInventario
- Esta opção é mais correcta do ponto de vista do design de classes e apresenta ainda algumas vantagens ao nível da manipulação dos objectos
- Dado que Filme é-um ItemInventario (tal como Lente e Camara), podemos fazer:

```
ItemInventario item;
item = new Lente (...); ou
item = new Filme (...); ou
item = new Camara (...);
```


Exemplo

- Um resultado interessante deste tipo de organização pode ser visto no código seguinte:

```
ItemInventario[] invent = new ItemInventario[5];
invent [0] = new Lente (...);
invent [1] = new Filme (...);
invent [2] = new Camera (...);
invent [3] = new Filme (...);
invent [4] = new Camera (...);
...
System.out.println ("Listagem de material em stock:");
for (int i = 0; i < invent.length; i++)
    System.out.println (invent[i].getDescricao());
```