CONCEITO DE HERANÇA (relembrar e aprofundar)

O conceito de **herança** permite definir uma nova classe, com base numa já existente. A classe criada, designada por subclasse, herda todas as propriedades e métodos da classe já existente (superclasse). O mecanismo de herança permite ainda que a subclasse inclua ou sobreponha novas propriedades e novos métodos.

O mecanismo de herança é recursivo, o que permite criar-se uma hierarquia de classes:

- Nos níveis mais altos da hierarquia estão características comuns a todos os objetos desta classe;
- Nos níveis inferiores estão especializações das classes superiores. As subclasses herdam as características comuns, além de definirem as suas propriedades específicas.

Vejamos o seguinte exemplo: vamos supor que está a desenvolver um sistema informático para uma empresa de produtos alimentares, que possui uma loja de venda ao público e um armazém para revenda. A empresa possui uma carteira de clientes particulares e empresariais.

Tendo em conta o exposto, poderíamos criar uma classe denominada Cliente com os seguintes atributos:

- `Nome;`
- `Morada;`
- `Localidade;`
- `Código Postal;`
- `Localidade;`
- `Contribuinte.`

Em seguida, criaríamos duas subclasses da classe Cliente, denominadas Particulares e Empresas. Ambas as classes herdariam os atributos da classe Cliente, mas poderiam ter alguns atributos a mais, como por exemplo a classe Empresas, que poderia conter mais o atributo CAE (Código de Atividade Económica).

Esquematicamente, teríamos:

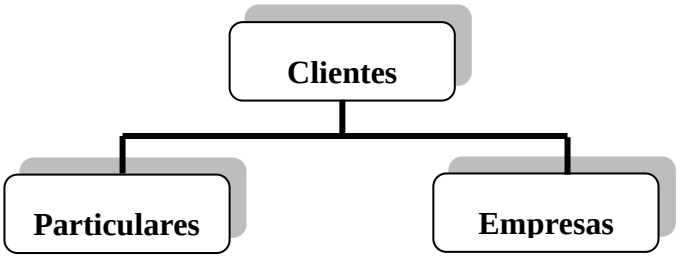


Fig. 1 – Esquema de relações entre classes

Se uma classe for declarada como uma subclasse de outra classe a sintaxe será a seguinte:

```
class nome_da_ classe extends Nome_da_superclasse {
        …
        /*código*/A
        …
```

}

CONCEITO DE POLIMORFISMO

O termo **polimorfismo** o é originário do grego e significa "muitas formas". Aplicado a linguagem orientada por objetos, o polimorfismo é uma técnica de permite implementar código para um objeto em que são criadas várias versões desse objeto para ser utilizado em contextos diferentes, com um modo de funcionar adequado a cada contexto.
O polimorfismo aplica-se apenas aos métodos da classe e, por defeito, exige a utilização da herança. A comunicação é estabelecida apenas na classe mais alta da hierarquia, contudo, como as subclasses estão ligadas por herança, é possível essas classes "receberem" a comunicação.
Com o conceito de polimorfismo, é possível acrescentar novos métodos a classes já existentes, sem a necessidade de recompilar a aplicação.
Por vezes também se usa a palavra **sobrecarga** (em inglês, **overload** ou **override**) para referir esta caraterística.
Pode-se dizer que a execução das versões do método irá depender da instância da classe que for criada.

"Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a mesma identificação, assinatura, mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. "

EXERCÍCIOS

1.No Eclipse cria um projeto de nome **F10ExemploHeranca**, a opção "Create Main Class" deve estar selecionada.

2. Acrescenta um novo ficheiro (opções New \ JavaClass…) com uma nova classe de nome **Animal**.

3. Na classe **Animal** acrescenta o seguinte código:

```
public class Animal {
      String nome;
      Animal( String valorNome ){
      nome = valorNome;
   }
   void Vive()
   {
      System.out.println("Vivo logo existo...");
   }
   void Come()
   {
      System.out.println("Como para sobreviver.");
   }
}
```

4. Na classe **ExemploHeranca**, declara um objeto da classe animal, imprime a sua propriedade nome e invoca os seus métodos, como se mostra de seguida:

```
public class ExemploHeranca {
 public static void main(String[ ] args) {
      Animal a = new Animal("Rei");
      System.out.println(a.nome);
      a.Vive();
      a.Come();
   }
}
```

5. Executa o projeto.

OUTPUT

```
Rei
Vivo logo existo...
Como para sobreviver...
```

6. Acrescenta uma nova classe de nome **Cavalo** no mesmo pacote.

7. Na classe Cavalo coloca o código que se segue, em substituição da classe Cavalo. Estarás a criar uma subclasse de Animal que irá herdar as suas propriedades e métodos.

```
public class Cavalo extends Animal{
   String raca;
   Cavalo(String nomeCavalo, String nomeRaca){
       super(nomeCavalo);
       raca = nomeRaca;
   }
   void Come() {
       System.out.println("Como palha");
   }
   void Salta() {
       System.out.println("Salto barreiras");
   }
}
```

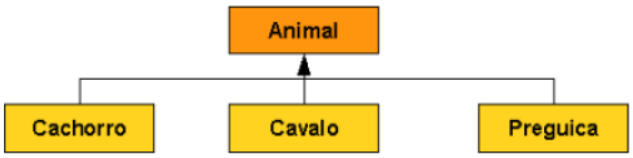
8. Na classe ExemploHeranca, acrescente o código a negrito para declarar um objeto da classe cavalo, imprimir a sua propriedade nome e invocar os seus métodos, como mostra de seguida:

```
public class ExemploHeranca {
public static void main(String[] args) {
      Animal a = new Animal("Rei");
      Cavalo c = new Cavalo("Spirit", "Puro Lusitano");
      System.out.println(a.nome);
      a.Vive();
      a.Come();
      System.out.println(c.nome);
      c.Vive();
      c.Come();
      System.out.println(c.raca);
      c.Salta();
      }}
```

9. Executa o projeto.
OUTPUT

```
Rei
Vivo logo existo...
Como para sobreviver...
Spirit
Vivo logo existo...
Como palha
Puro Lusitano
Salto barreiras
```

10. De acordo com a seguinte hierarquia de classes acrescenta as classes que faltam. Deves criá-las no mesmo pacote.



11. Considera que o(a):

Cavalo	Cachoro	Preguiça
Tem nome	Tem nome	Tem nome
Tem raça	Tem dono	Tem habitat
Tem idade	Tem idade	Tem idade
Vive	Vive	Vive
Come palha	Come carne	Come bagas
Salta	Ladra	Sobe às árvores
Emite som	Emite som	Emite som

Nota:
- As propriedades e métodos comuns devem ser implementados na classe Animal.
(Os métodos são assim implementados de forma **polimórfica**, sendo independentes do tipo de animal).

12. Declara dois novos objetos **cao** da classe Cachorro e **preg** da classe Preguica, invoca os seus métodos e propriedades e executa o projeto.
OUTPUT (com e sem o método toString())

```
Rei
Vivo logo existo...
Como para sobreviver...
Spirit
10
Vivo logo existo...
Como palha
Puro Lusitano
Salto barreiras
Cavalo [raca=Puro Lusitano, nome=Spirit, idade=10]

Nome: Bobby
Idade: 2
Dono: Ana
Cachorro [dono=Ana, nome=Bobby, idade=2]
Vivo logo existo...
Como carne
Faço-me ouvir...
Ladro a quem passa...

Nome: Lentinha
Idade: 3
Habitat: Nva Zelândia
Preguica [habitat=Nva Zelândia, nome=Lentinha, idade=3]
Vivo logo existo...
Como bagas!
Faço-me ouvir...
subo às árvores... lentameeente!
```

13. Implementa uma nova classe **Veterinario** que tem o método Examinar que recebe como parâmetro um objeto da classe Animal. Este método deve listar o nome do animal examinado, o que come e emitir som. Deve colocar no mesmo sourcepackage.

14. Invoca o método examinar para os animais definidos no teu projeto.

15. Executa o projeto.

OUTPUT

```
Examinar Cavalo

Nome:Spirit
Como palha
Faço-me ouvir...

Examinar Cachorro

Nome:Bobby
Como carne
Faço-me ouvir...

Examinar Preguiça

Nome:Lentinha
Como bagas!
Faço-me ouvir...
```

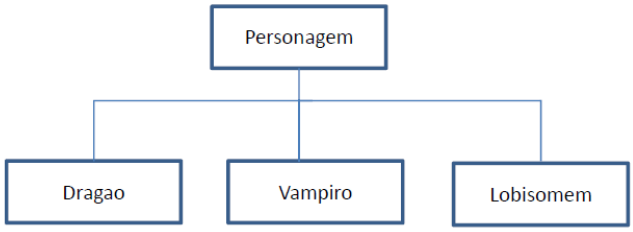
16. Comprime a tua solução (preferencialmente com 7Zip) e envia o ficheiro comprimido para a classroom

---

1. No Eclipse, cria um projeto de nome F10Personagens e uma classe de nome **TestaPersonagens** com o método main.

2. Acrescenta ao projeto uma nova classe de nome **Personagem**.

3. De acordo com a seguinte hierarquia de classes acrescenta as classes que faltam. Deves coloca-las no mesmo pacote.



4. Considera que:

Dragao	Vampiro	Lobisomem
Tem nome	Tem nome	Tem nome
Tem saúde 10	Tem saúde 2	Tem saúde 5
Tem pontos_de_magia 1	Tem pontos_de_magia 8	Tem pontos_de_magia 2
Anda	Anda	Anda
DeitaChamas	Suga	Morde
Voa	FicaMorcego	Transforma
Come bagas	Come espinafres	Come alface
Prende	Amarra	Derruba
Guincha	Grita	Uiva

Nota:
- As propriedades e métodos comuns devem ser implementados na classe Personagem.
- Introduzir os getters, setters e toString
- Vai testando a tua implementação.

5. Na classe **TestaPersonagens** cria dois objetos de cada classe.

6. Declara um método **Listar** na classe **TestaPersonagen** que tem como parâmetro um Personagem e deve imprimir todas as características comuns aos personagens.

7. Declara um método **ListarDetalhes** na classe **TestaPersonagen** que tem como parâmetro um objeto da classe Personagem e deve imprimir todas as características e invocar todos os seus métodos.
Nota: No método ImprimirDetalhes usa, para acederes ao método criado em cada subclasse código semelhante ao seguinte:

```
if (p.getClass().getName().equals("Vampiro"))

((Vampiro) p).Suga();
```

→ método getClasse().getName() - para obter o nome da subclasse associada à variável de instância,
→downCasting - para converter a variável de instância da classe Personagem numa variável de instância da subclasse Vampiro):

8. Executa o projeto.
OUTPUT (para um objeto da classe Vampiro):

```
LISTAR
Personagem [nome=Drácula, saude=2, pontos_magia=8]
LISTAR detalhes
Personagem [nome=Drácula, saude=2, pontos_magia=8]
Eu ando!
Eu como espinafres!
Eu grito!
Eu transformo-mo num morcego!
Eu amarro as minhas vítimas!
Eu sugo as minhas vítimas!
```

9. Para todos os objetos invoca o método Listar.

10. Cria uma nova classe de nome Hibrido subclasse da classe Vampiro. O Hibrido tem o método Correr.
Alteração efetuada no método toString da superclasse:

```
public String toString(String string) {
    return string +"[nome=" + nome + ", saude=" + saude + ", pontos_magia=" + pontos_magia + "]";
}
```

Exemplo da classe TestaPersonagem:

```
void ListarDetalhes(Personagem p) {
    System.out.println(p.toString(p.getClass().getName()));

    if (p.getClass().getName().equals("Vampiro")){

        ((Vampiro) p).Suga();}

    if (p.getClass().getName().equals("Hibrido"))

        ((Hibrido) p).correr();
    p.Anda();
    p.Come();
    p.EmitoSom();
    p.Poder1();
    p.Poder2();
}
```

```java
public static void main(String[] args) {

    TestaPersonagem tp=new TestaPersonagem ();
    Personagem p=new Personagem ("",0,0);
    Vampiro v1=new Vampiro ("Drácula", 2,8);
    Vampiro v2=new Vampiro ("Drácula Júnior", 2,8);
    Hibrido h = new Hibrido("Homidrag", 10, 1);

    System.out.println("LISTAR");
    tp.Listar(v1);

    System.out.println("LISTAR detalhes");
    tp.ListarDetalhes(v1);
    tp.ListarDetalhes(h);

}
```

12. Executa o projeto, já com a inclusão da classe Hibrido.

OUTUP (com objeto da classe Vampiro e da classe Hibrido criados):

```
LISTAR detalhes
Vampiro[nome=Drácula, saude=2, pontos_magia=8]
Eu sugo as minhas vítimas!
Eu ando!
Eu como espinafres!
Eu grito!
Eu transformo-me num morcego!
Eu amarro as minhas vítimas!
Hibrido[nome=Homidrag, saude=10, pontos_magia=1]

Eu corro mais rápido que o Bolt!
Eu ando!
Eu como espinafres!
Eu grito!
Eu transformo-me num morcego!
Eu amarro as minhas vítimas!
```

13. Comprime a sua solução (preferencialmente com 7Zip) e envia o ficheiro comprimido para a classroom.

Bibliografia:
https://www.w3schools.com/java
https://www.tutorialspoint.com/java/
Jesus, C. (2013). Curso Prático de Java. Lisboa: FCA
Coelho, P (2016). Programação em JAVA – Curso Completo. Lisboa: FCA