

FCT/Unesp – Presidente Prudente
Departamento de Matemática e Computação

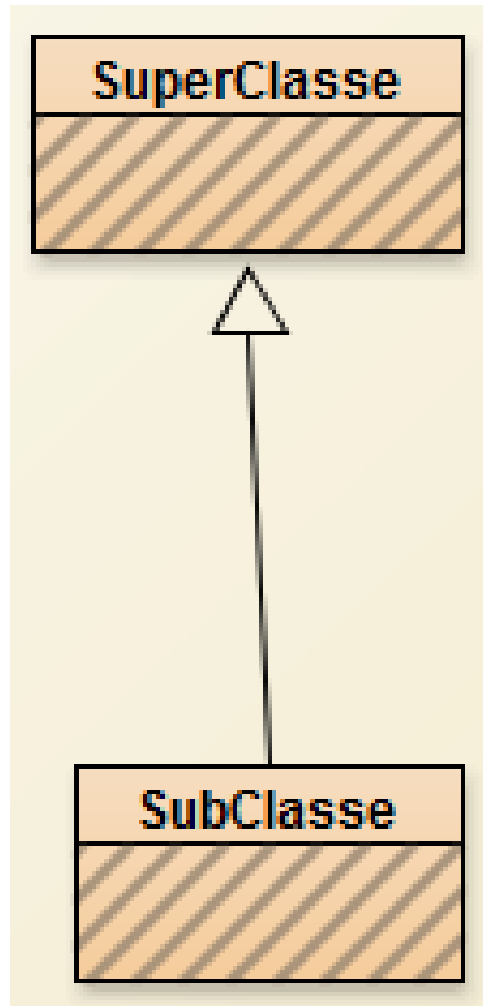
Programação Orientada a Objetos

Polimorfismo

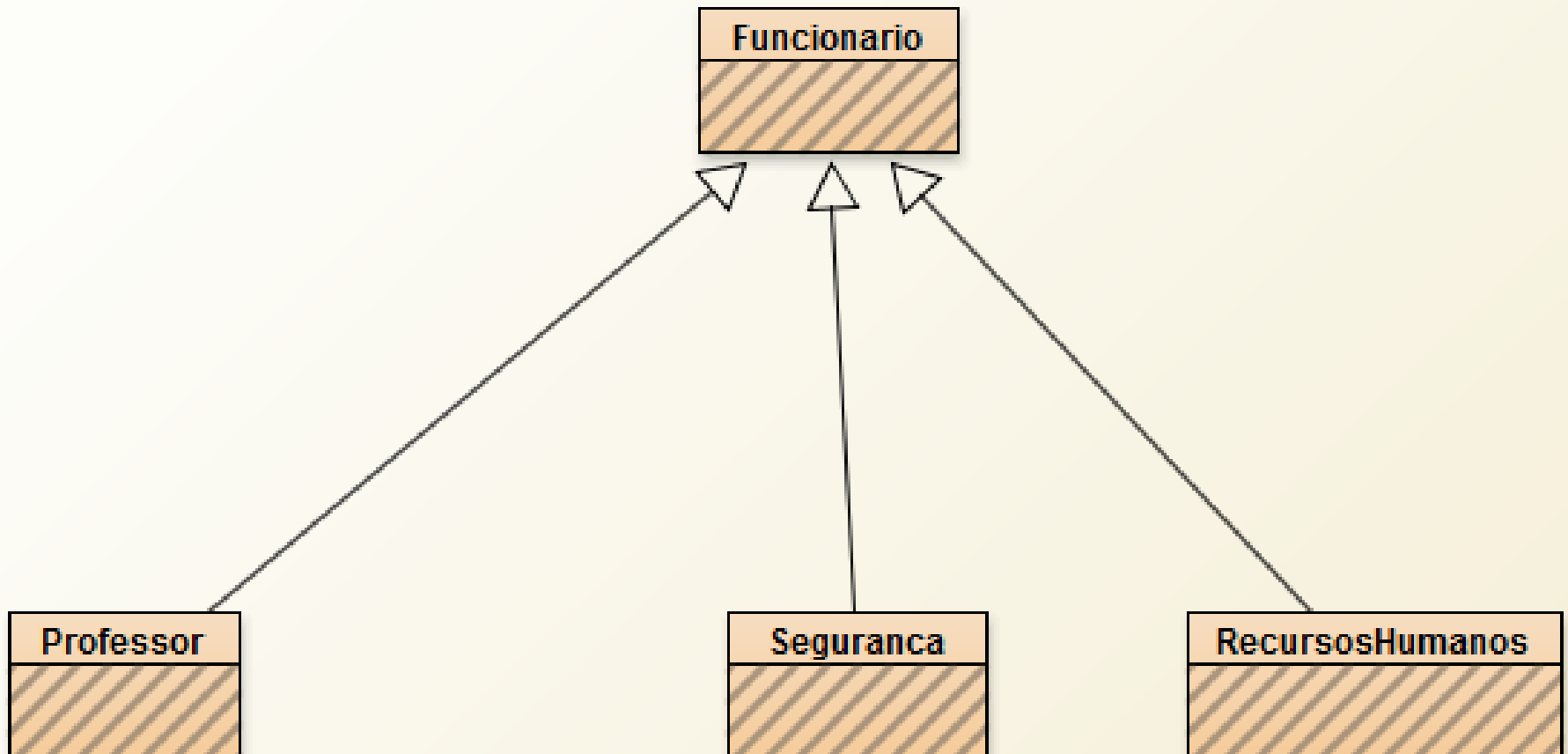
Prof. Danilo Medeiros Eler
danilo.eler@unesp.br

Herança

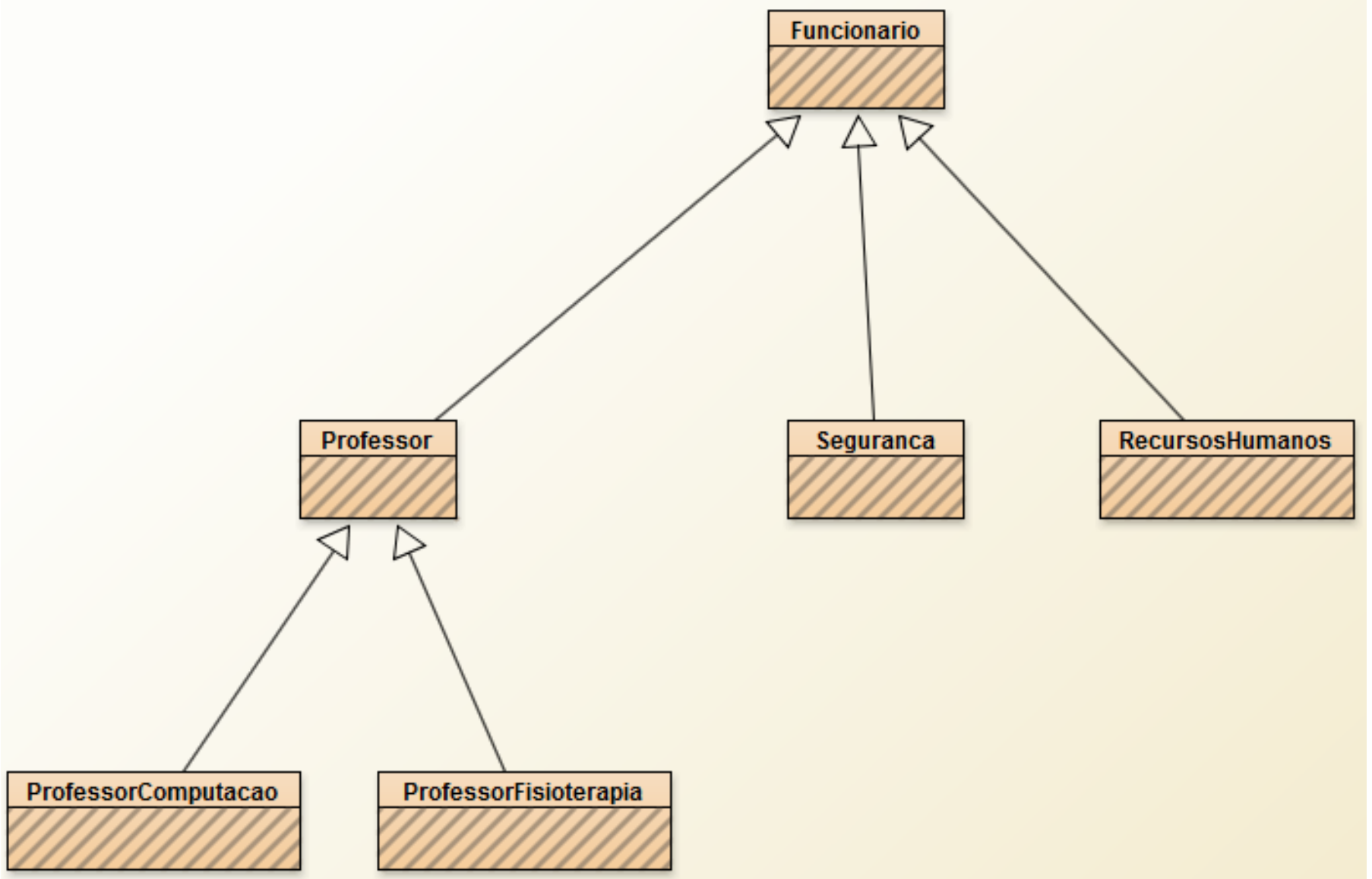
- Representação Gráfica



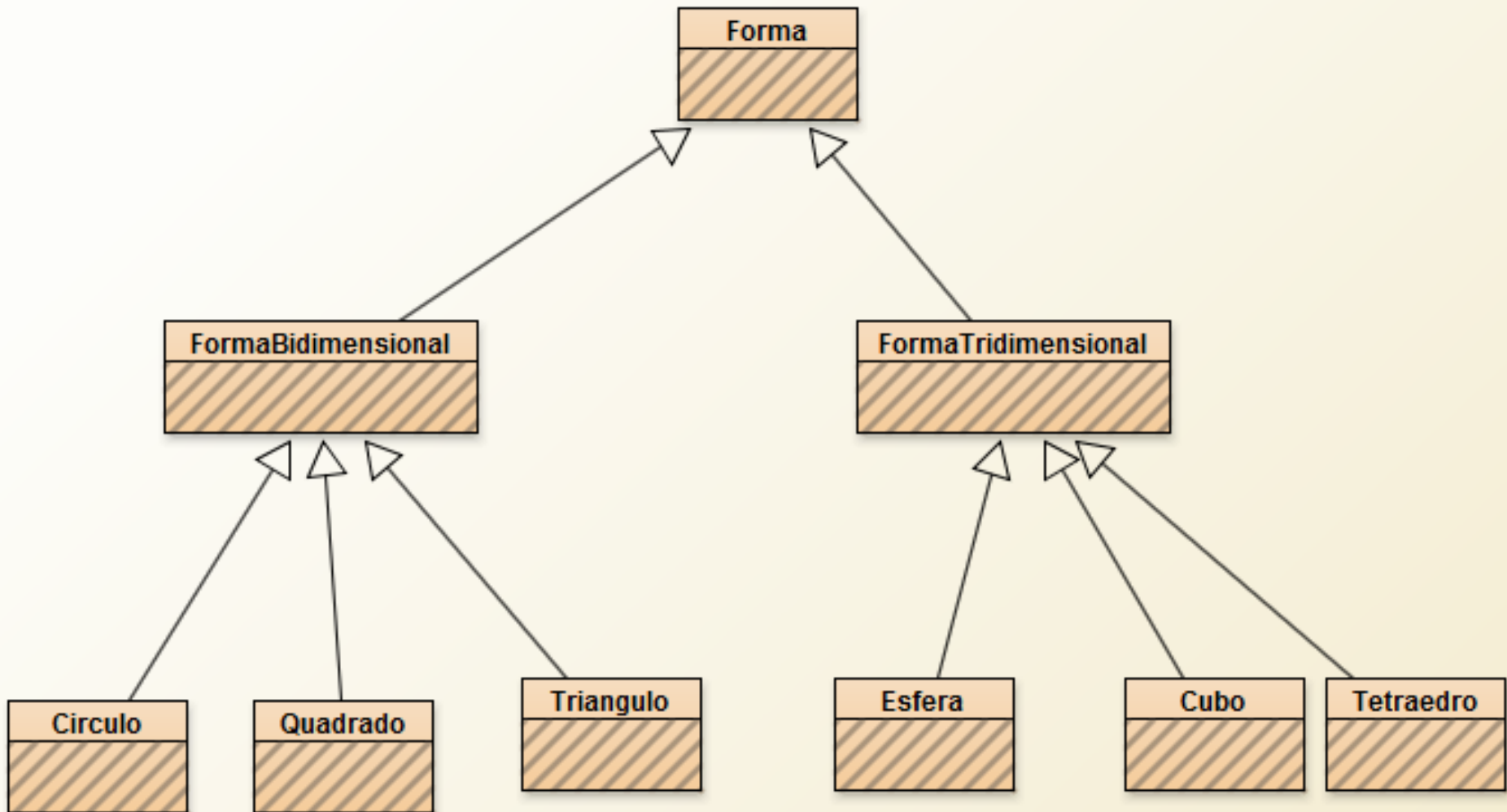
Herança



Herança



Herança: outro exemplo



Herança: sintaxe java

```
class Funcionario{  
.....  
}
```

```
class Professor extends Funcionario{
```

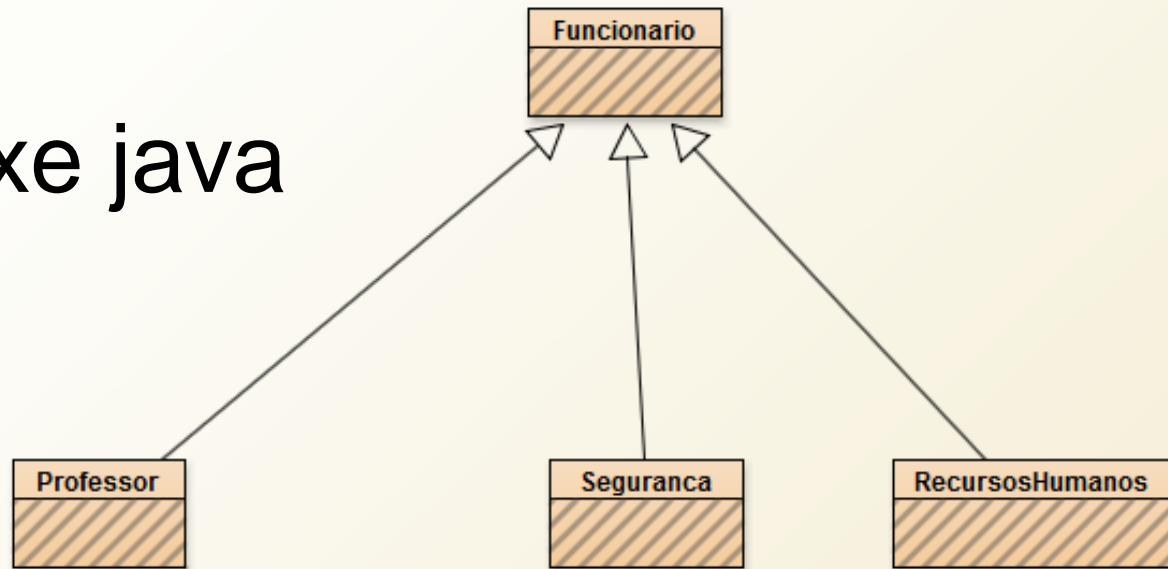
```
.....  
}
```

```
class Seguranca extends Funcionario{
```

```
.....  
}
```

```
class RecursosHumanos extends Funcionario{
```

```
.....  
}
```



Exemplo – Sem Polimorfismo

- Imagine que tenhamos que catalogar CDs e Videos em um estabelecimento
- Poderíamos representar essas entidades pelas classes

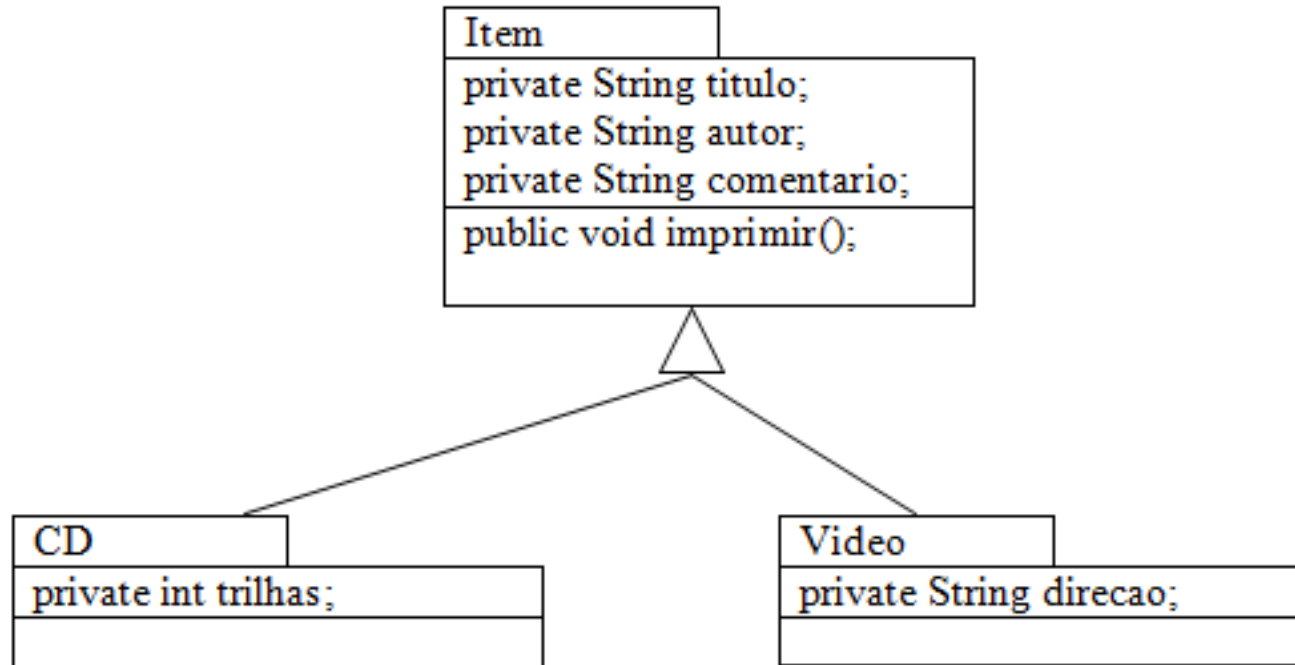
CD
<code>private String titulo;</code> <code>private String autor;</code> <code>private String comentario;</code> <code>private int trilhas;</code>
<code>public void imprimir();</code>

Video
<code>private String titulo;</code> <code>private String autor;</code> <code>private String comentario;</code> <code>private String direcao;</code>
<code>public void imprimir();</code>

Exemplo – Sem Polimorfismo

- Problemas dessa solução
 - Armazenamento em diferentes vetores
 - Alterações comuns devem ser realizadas nas duas classes
 - A adição de novos tipos de itens demandaria grande esforço para a evolução do código
 - Entre muitos outros
- Uma solução para resolver muitos desses problemas é utilizar a herança com polimorfismo

Exemplo – Polimorfismo



Exemplo – Com Polimorfismo

- Método imprimir da classe item

```
public void imprimir(){  
    System.out.println(titulo);  
    System.out.println(autor);  
    System.out.println(comentario);  
}
```

Exemplo – Com Polimorfismo

```
Item itens[] = new Item[100];  
itens[0] = new CD("CD1", "Autor1", "Bom", 15);  
itens[1] = new Video("Video1", "Autor1", "Otimo", "DiretorX" );
```

```
Item it = itens[0];  
it.imprimir();  
it = itens[1];  
it.imprimir();
```

Exemplo – Com Polimorfismo

```
Item itens[] = new Item[100];
```

```
itens[0] = new CD("CD1", "Autor1", "Bom", 15);
```

```
itens[1] = new Video("Video1", "Autor1", "Otimo", "DiretorX" );
```

```
Item it = itens[0];
```

```
it.imprimir();
```

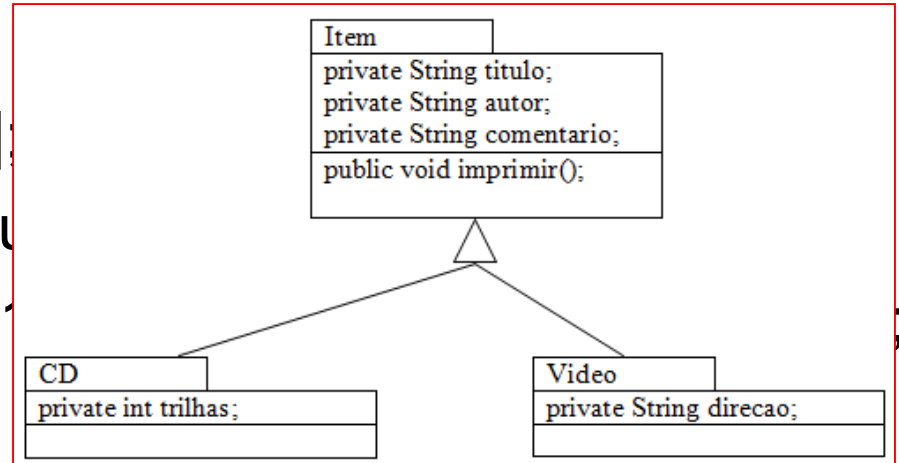
```
it = itens[1];
```

```
it.imprimir();
```



Exemplo – Com Polimorfismo

```
Item itens[] = new Item[100];  
itens[0] = new CD("CD1", "Autor1", "Bom");  
itens[1] = new Video("Video1", "Autor1", "Otimo");
```



```
Item it = itens[0];
```

```
it.imprimir();
```

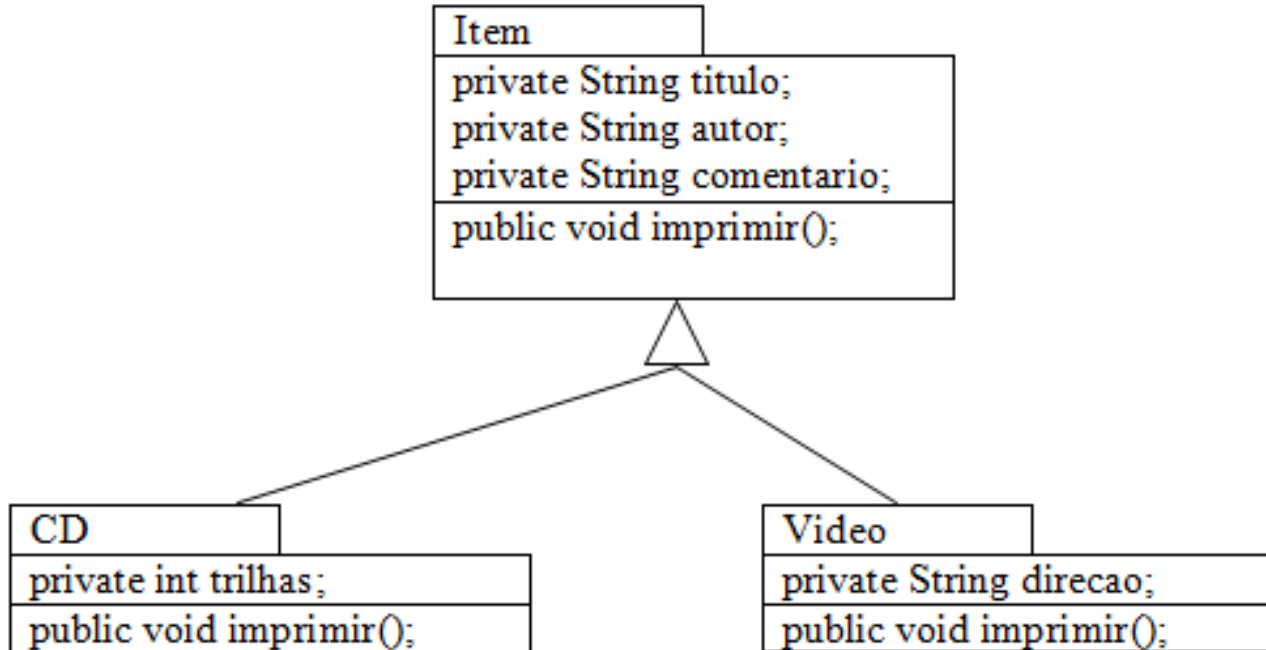
```
it = itens[1];
```

```
it.imprimir();
```

```
CD1  
Autor1  
Bom  
  
Video1  
Autor1  
Otimo
```

Exemplo – Com Polimorfismo

- A classe Item não tem acesso aos atributos das subclasses CD e Video
- A solução é sobrescrever o método imprimir em cada subclasse



Exemplo – Com Polimorfismo

```
Item itens[] = new Item[100];  
itens[0] = new CD("CD1", "Autor1", "Bom", 15);  
itens[1] = new Video("Video1", "Autor1", "Otimo", "DiretorX" );
```

```
Item it = itens[0];  
it.imprimir();  
it = itens[1];  
it.imprimir();
```

Exemplo – Com Polimorfismo


```
Item itens[] = new Item[100];
```

```
itens[0] = new CD("CD1", "Autor1", "Bom", 15);
```

```
itens[1] = new Video("Video1", "Autor1", "Otimo", "DiretorX" );
```


```
Item it = itens[0];
```

```
it.imprimir();
```



```
it = itens[1];
```

```
it.imprimir();
```



CD1
Autor1
Bom
15
Video1
Autor1
Otimo
DiretorX

Exemplo – Com Polimorfismo

```
Item itens[] = new Item[100];
```

```
itens[0] = new CD("CD1", "Autor1", "Bom", 15);
```

```
itens[1] = new Video("Video1", "Autor1", "Otimo", "DiretorX" );
```

```
Item it = itens[0];
```

```
it.imprimir();
```

```
it = itens[1];
```

```
it.imprimir();
```

CD1
Autor1
Bom
15

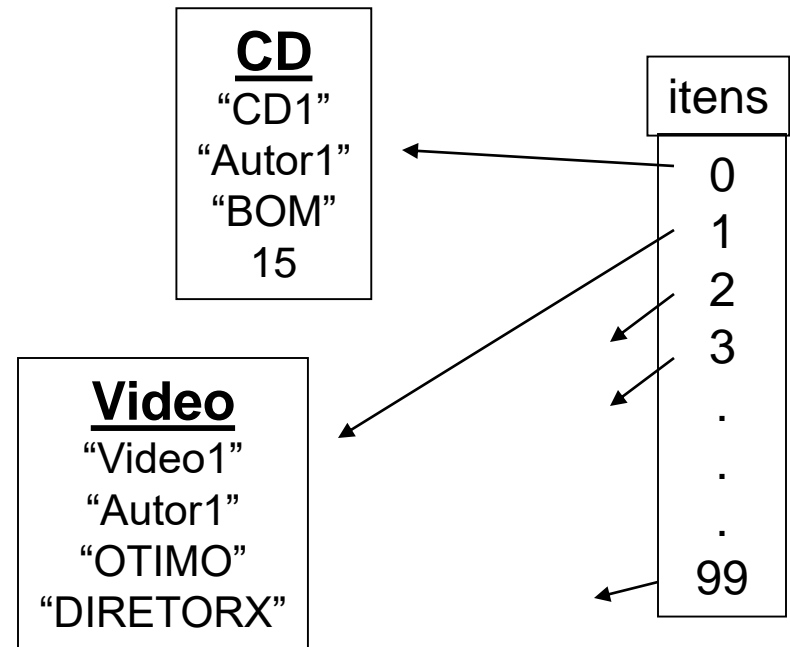
Video1
Autor1
Otimo
DiretorX

Porque isso funciona se a variável utilizada para referenciar os objetos é do tipo Item?

Exemplo – Com Polimorfismo

- A variável `it` é do tipo `Item` (estático), mas o tipo dos objetos referenciados é dinâmico

```
Item it = itens[0];  
it.imprimir();  
it = itens[1];  
it.imprimir();
```



Exemplo – Com Polimorfismo

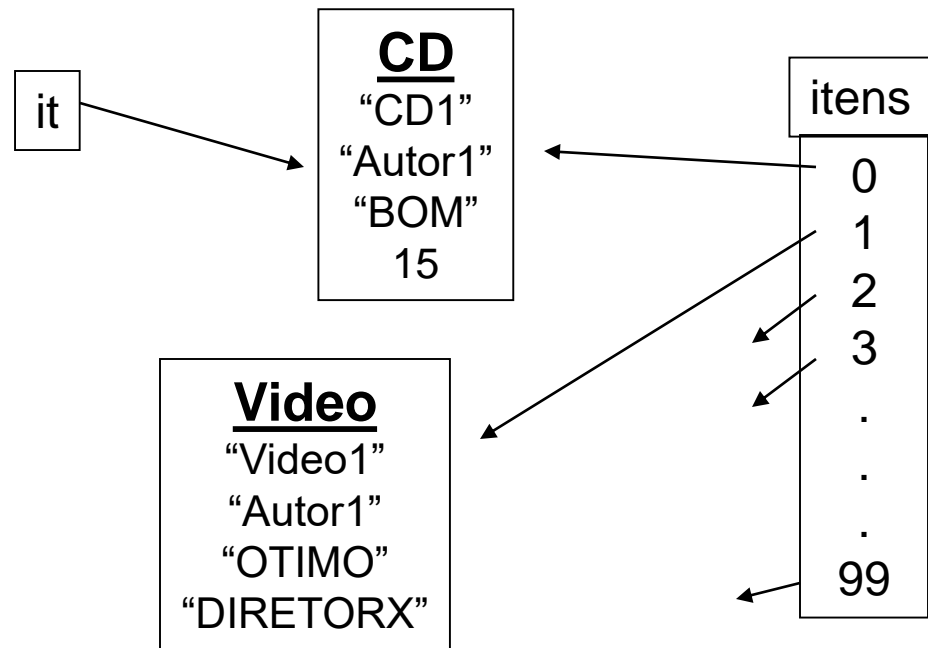
- A variável `it` é do tipo `Item` (estático), mas o tipo dos objetos referenciados é dinâmico

```
Item it = itens[0];
```

```
it.imprimir();
```

```
it = itens[1];
```

```
it.imprimir();
```



Exemplo – Com Polimorfismo

- A variável `it` é do tipo `Item` (estático), mas o tipo dos objetos referenciados é dinâmico

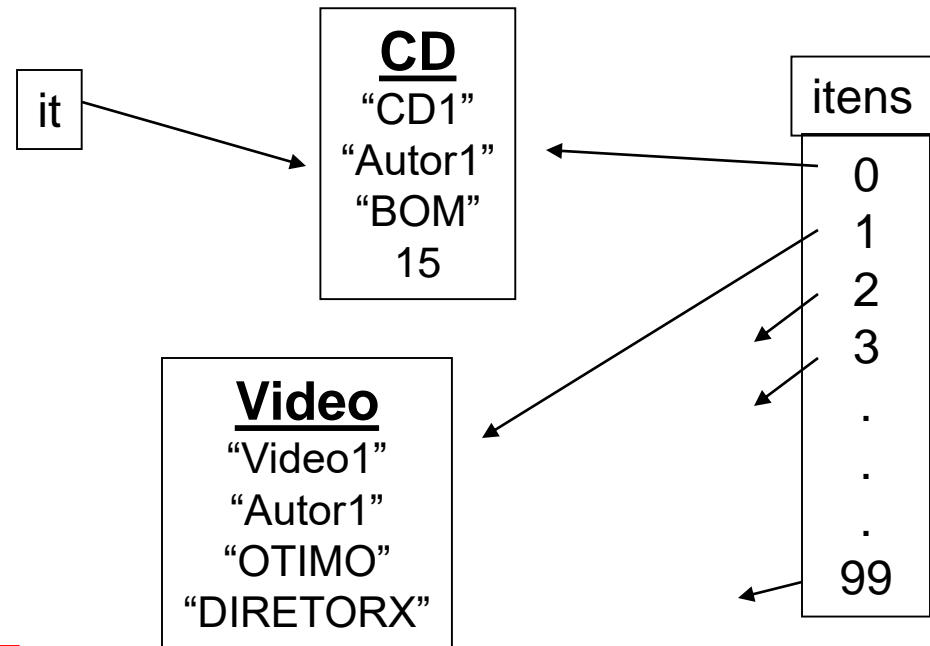
```
Item it = itens[0];
```

```
it.imprimir();
```

```
it = itens[1];
```

```
it.imprimir();
```

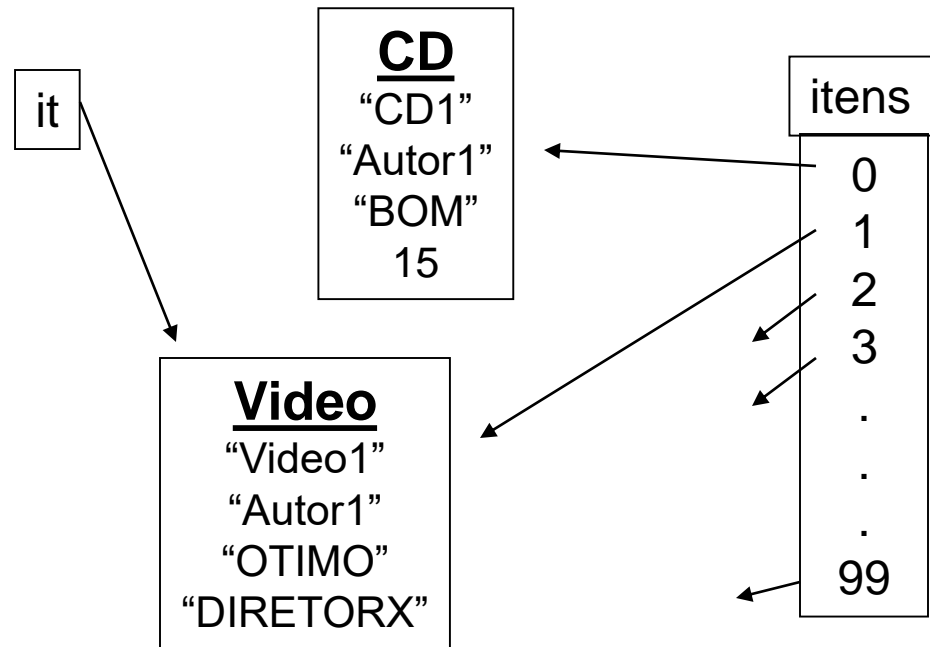
Invoca o método `imprimir` da classe `CD`, que é o tipo do objeto apontado por `it`



Exemplo – Com Polimorfismo

- A variável `it` é do tipo `Item` (estático), mas o tipo dos objetos referenciados é dinâmico

```
Item it = itens[0];  
it.imprimir();  
it = itens[1];  
it.imprimir();
```

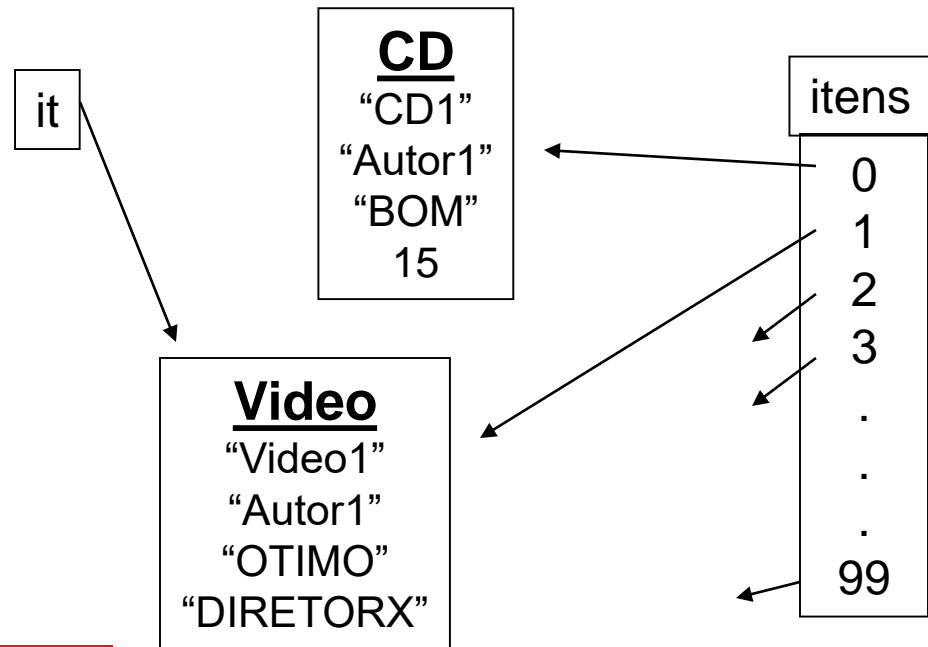


Exemplo – Com Polimorfismo

- A variável `it` é do tipo `Item` (estático), mas o tipo dos objetos referenciados é dinâmico

```
Item it = itens[0];  
it.imprimir();  
it = itens[1];  
it.imprimir();
```

Invoca o método `imprimir` da classe `Video`, que é o tipo do objeto apontado por `it`



Exemplo – Com Polimorfismo

- A variável `it` é uma variável polimórfica, ou seja, ela pode assumir o comportamento de diferentes objetos
- Além disso, o método `imprimir` é um método polimórfico, pois ele pode assumir diferentes comportamentos

Polimorfismo

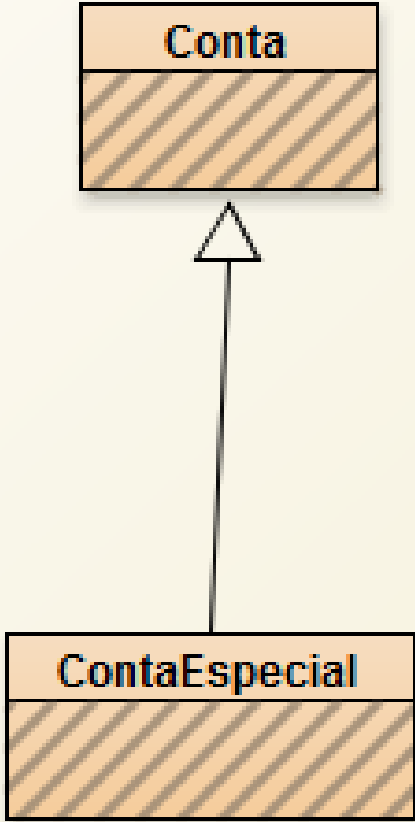
- Polimorfismo significa muitas formas
- Ele permite que um único nome de classe ou nome de método represente um código diferente
- Assim, um nome pode assumir muitas formas e, como pode representar códigos diferentes, pode assumir diferentes comportamentos

Polimorfismo

```
public static void main(String args[]) {  
    Conta cel = new Conta("CE 1111", 200);  
    cel.imprimir();  
    cel.depositar(100);  
    cel.imprimir();  
    cel.sacar(200);  
    cel.imprimir();  
}
```

Variável Polimórfica

```
public static void main(String args[]) {  
    Conta cel = new ContaEspecial("CE 1111", 200, 400);  
    cel.imprimir();  
    cel.depositar(100);  
    cel.imprimir();  
    cel.sacar(200);  
    cel.imprimir();  
}
```



Polimorfismo

- Como a variável polimórfica aponta para diferentes tipos de objetos, os métodos invocados por ela pode ter diferentes tipos de comportamento
- Polimorfismo libera o programador de ter que saber a classe específica do objeto que recebe uma mensagem
 - Exemplo: método sacar ou relatorio da Conta e da Conta Especial

Polimorfismo

- Permite agrupar objetos de classes distintas em uma mesma estrutura

– Exemplo:

```
Conta vetor[] = new Conta[100];
vetor[0] = new Conta(....);
vetor[1] = new ContaEspecial(....);
vetor[2] = new Conta(....);
vetor[3] = new Conta(....);
vetor[4] = new ContaEspecial(...);
for (int i=0; i<5; i++){
    vetor[i].relatorio();
}
```

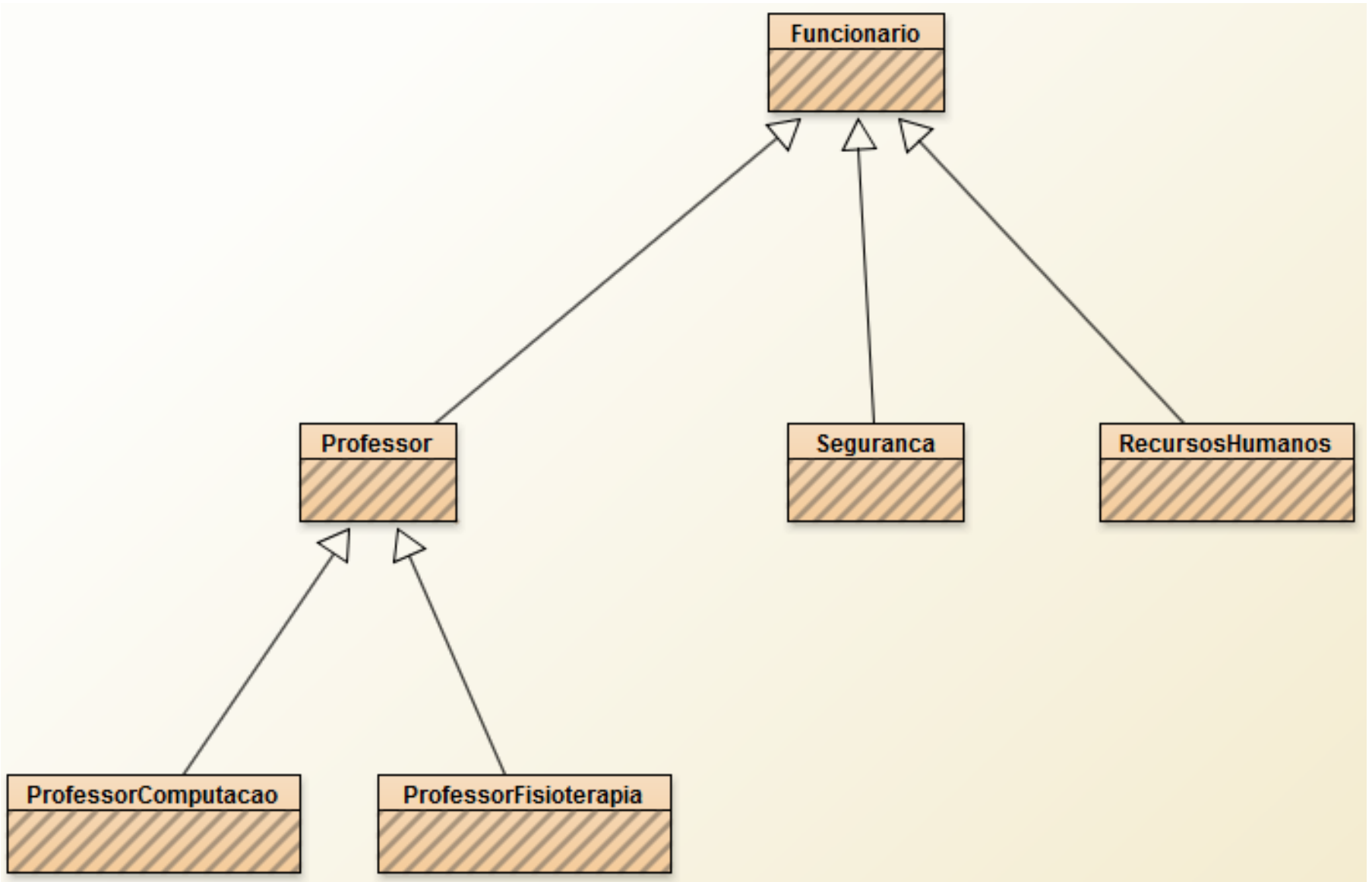
Polimorfismo

- Facilita a evolução do software

– Exemplo:

```
Conta vetor[] = new Conta[100];
vetor[0] = new Conta(...);
vetor[1] = new ContaEspecial(...);
vetor[2] = new Conta(...);
vetor[3] = new Conta(...);
vetor[4] = new ContaEspecial(...);
vetor[5] = new ContaEspecial2(...);
for (int i=0; i<6; i++){
    vetor[i].relatorio();
}
```

Polimorfismo



Polimorfismo

Funcionario f = new ProfessorComputacao();

f = new ProfessorFisioterapia();

Professor p = new Professor();

p = new ProfessorComputacao();

p = new ProfessorFisioterapia();

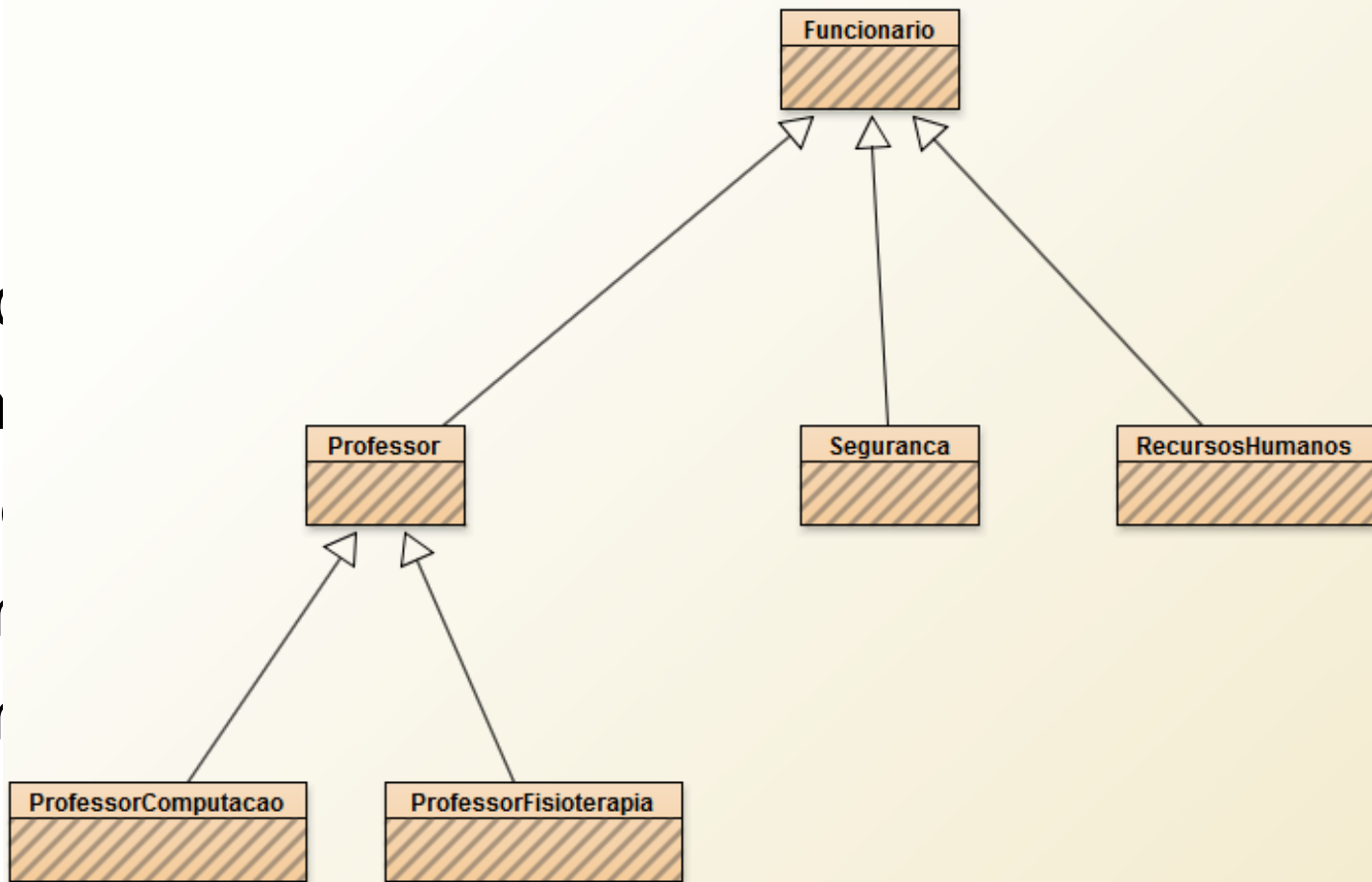
Posso fazer as seguintes referências?

ProfessorComputacao pc = new Funcionario();

Professor p = new Funcionario();

Professor p = new Seguranca();

Func
f = n
Prof
p = r
p = r



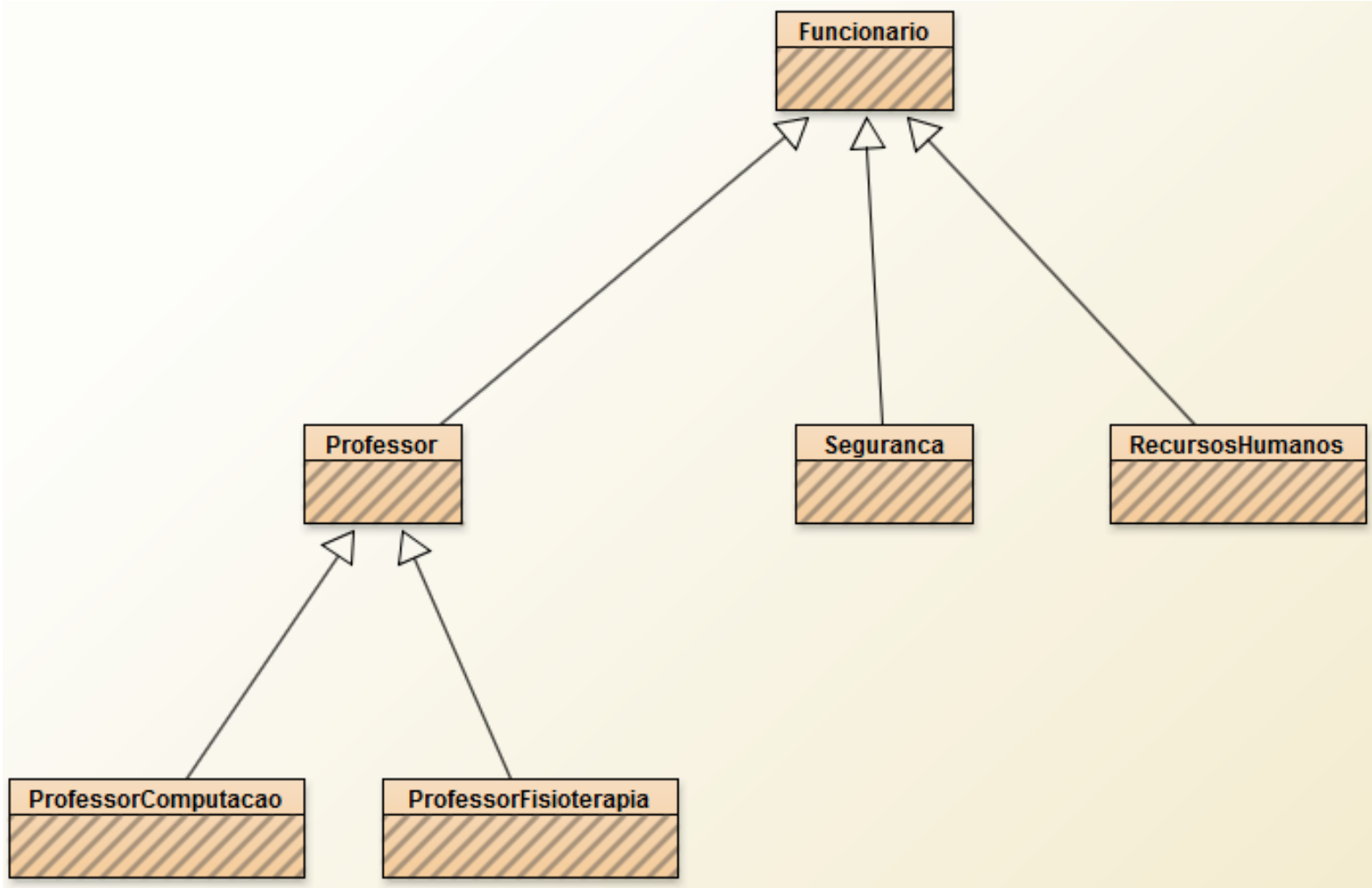
~~ProfessorComputacao pc = new Funcionario();~~

~~Profes = new Funcionario();~~

~~Professor p = new Seguranca();~~

TIPOS INCOMPATÍVEIS

Como saber o tipo do objeto?



Como saber o tipo do objeto?

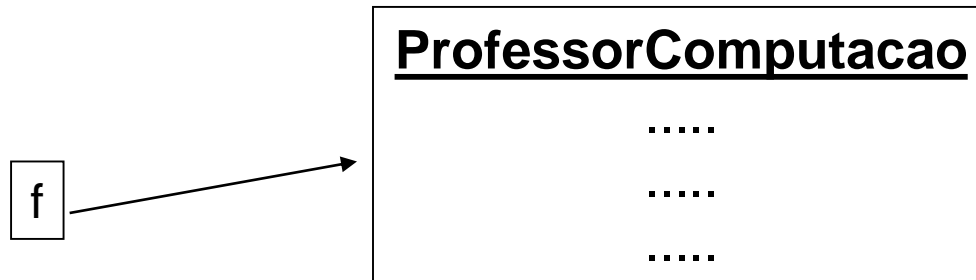
- Para identificar o tipo de um objeto é possível utilizar *instanceof*

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof ProfessorComputacao){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

- Para identificar o tipo de um objeto é possível utilizar *instanceof*

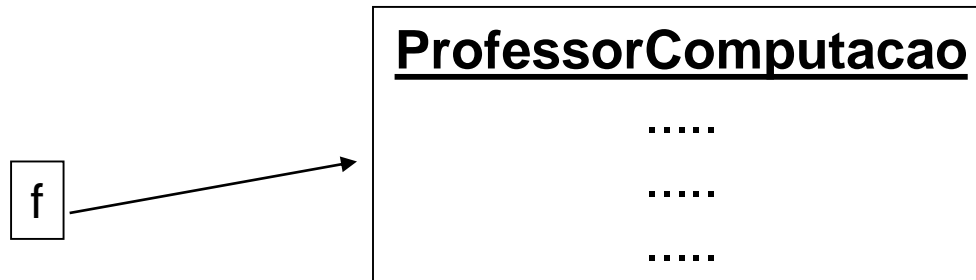
TRUE

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof ProfessorComputacao){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

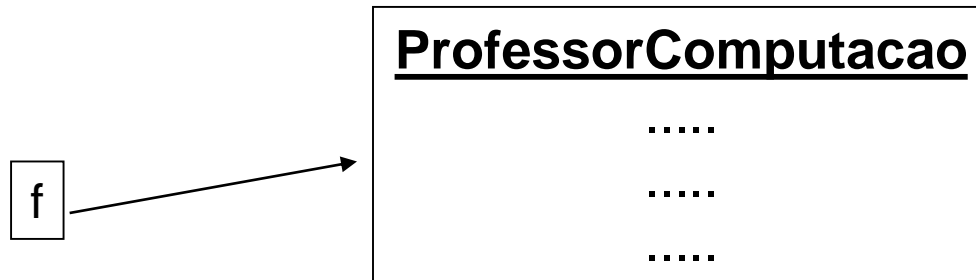
- Para identificar o tipo de um objeto é possível utilizar *instanceof*

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof ProfessorFisioterapia){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

- Para identificar o tipo de um objeto é possível utilizar *instanceof*

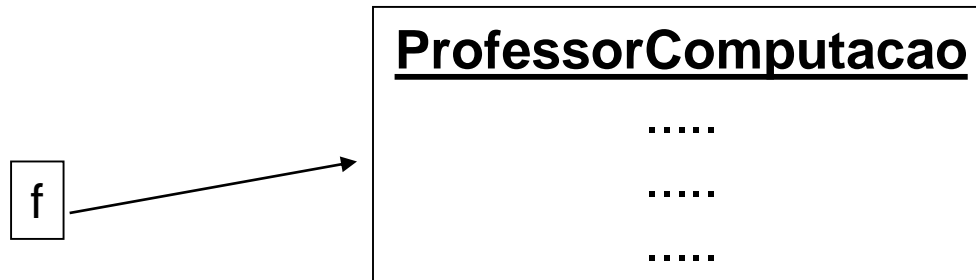
FALSE

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof ProfessorFisioterapia){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

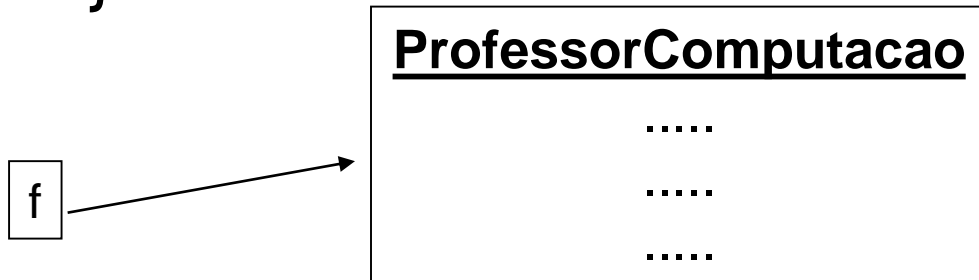
- Para identificar o tipo de um objeto é possível utilizar *instanceof*

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof Professor){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

- Para identificar o tipo de um objeto é possível utilizar *instanceof*

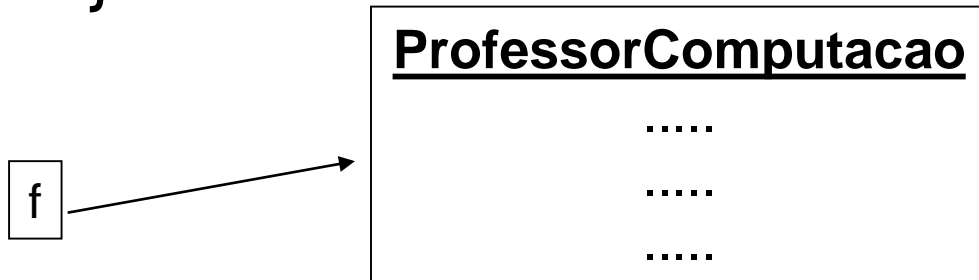
TRUE

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof Professor){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

- Para identificar o tipo de um objeto é possível utilizar *instanceof*

TRUE

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof Professor){
```

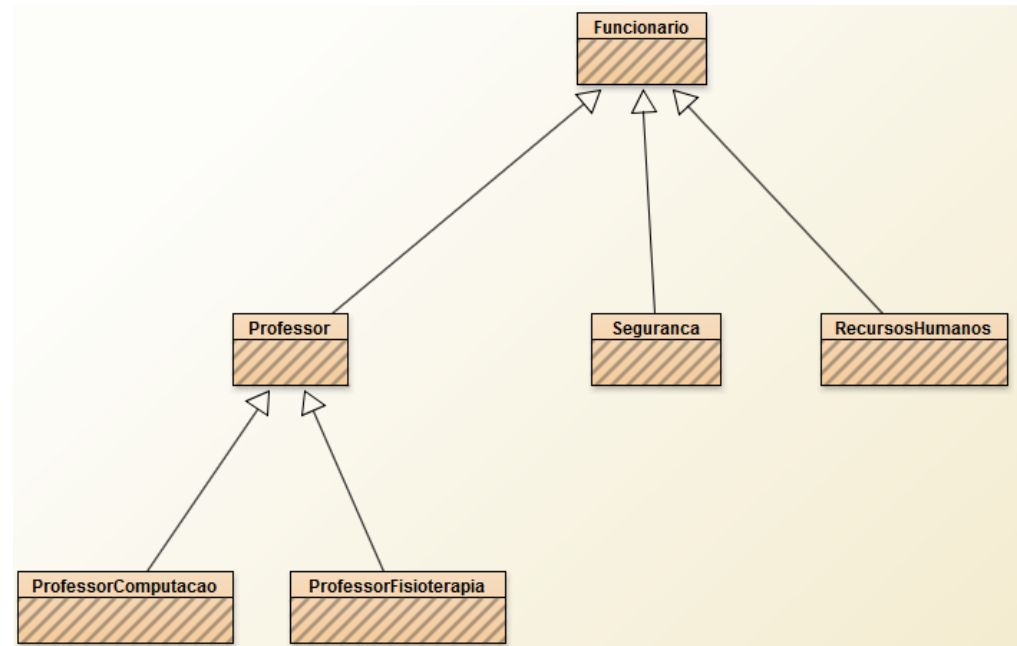
```
.....
```

```
}
```

ProfessorComputacao

.....
.....
.....

f



Como saber o tipo do objeto?

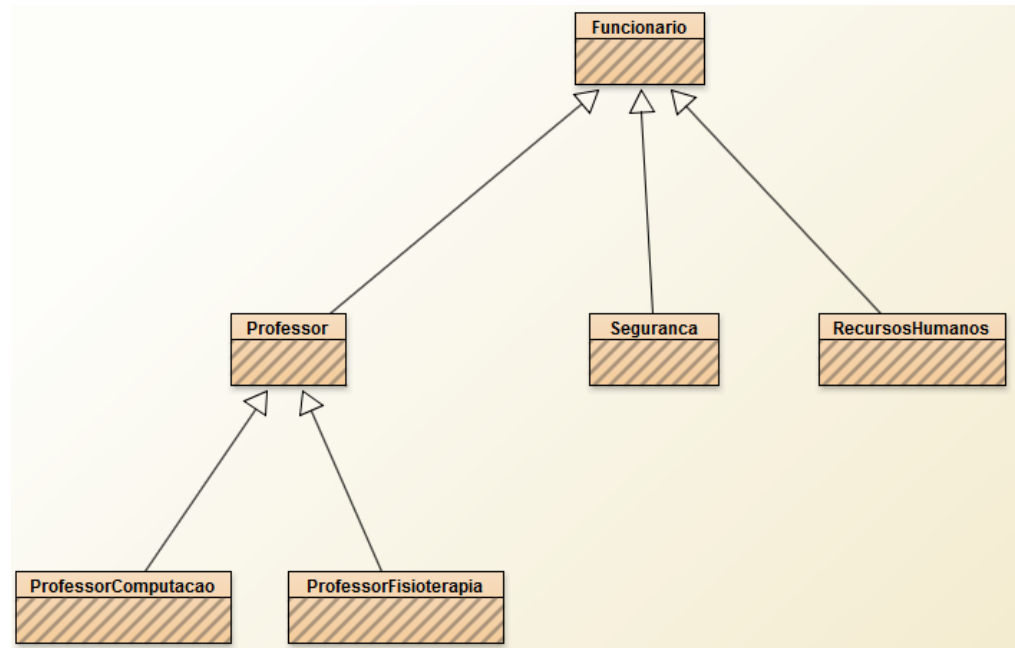
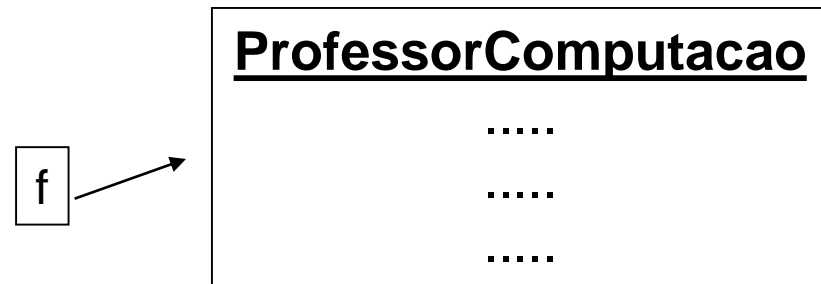
- Para identificar o tipo de um objeto é possível utilizar *instanceof*

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof Funcionario){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

- Para identificar o tipo de um objeto é possível utilizar *instanceof*

TRUE

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof Funcionario){
```

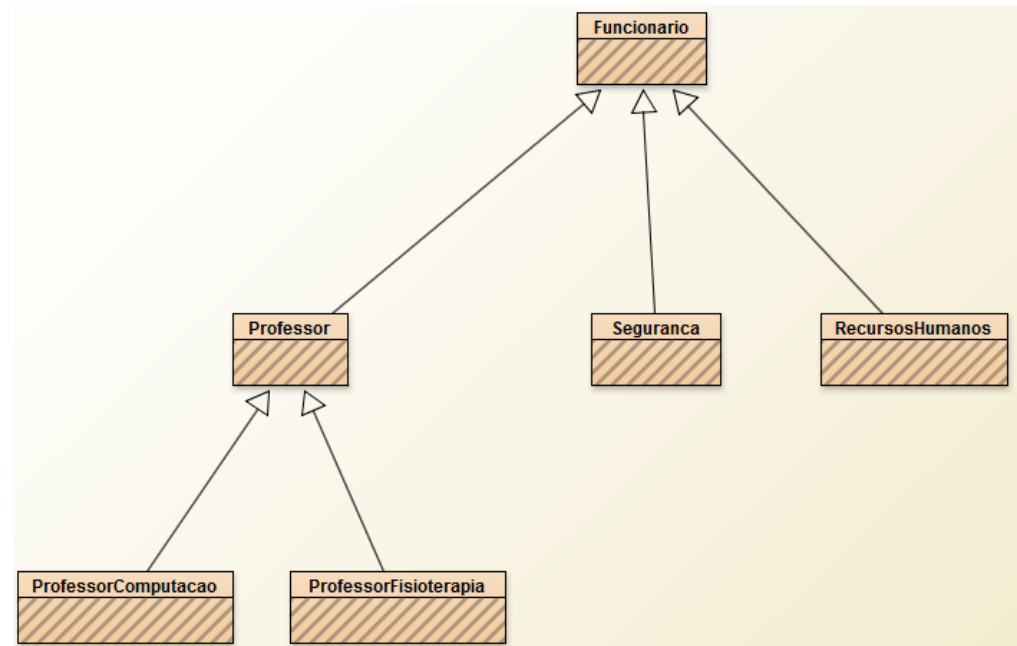
```
.....
```

```
}
```

ProfessorComputacao

.....
.....
.....

f



Como saber o tipo do objeto?

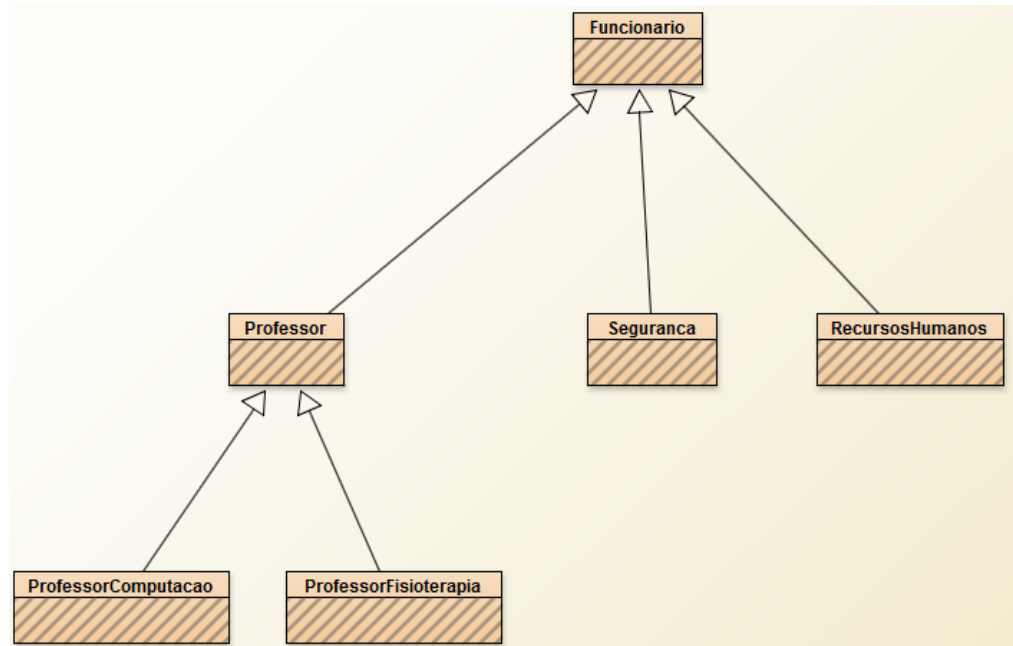
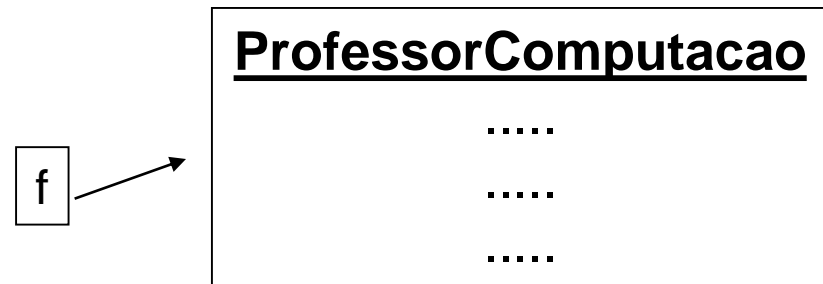
- Para identificar o tipo de um objeto é possível utilizar *instanceof*

```
Funcionario f = new ProfessorComputacao();
```

```
If (f instanceof Seguranca){
```

```
.....
```

```
}
```



Como saber o tipo do objeto?

- Para identificar o tipo de um objeto é possível utilizar *instanceof*

FALSE

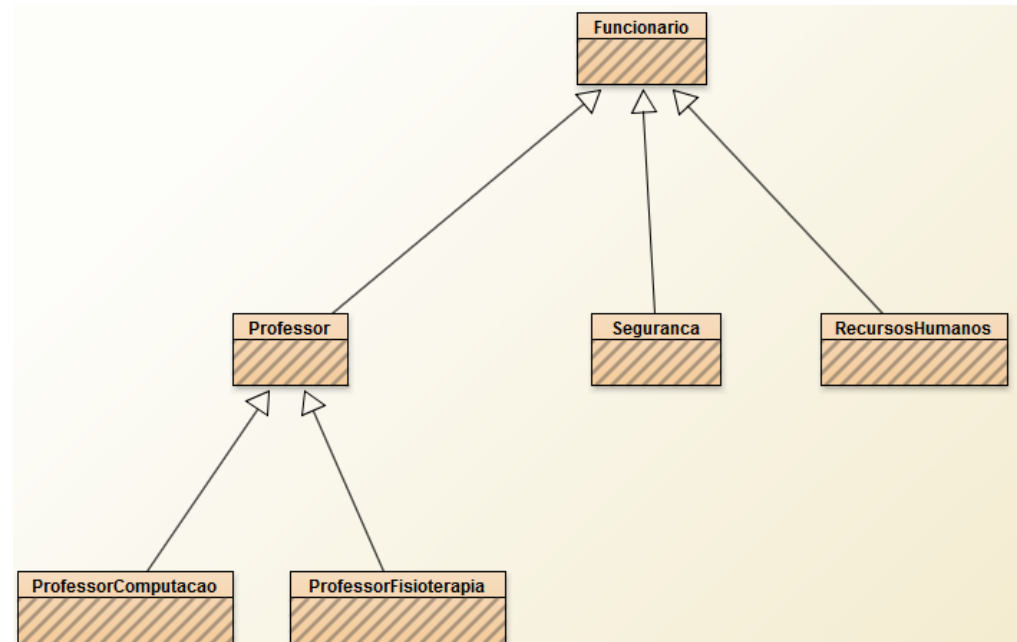
```
Funcionario f = new ProfessorComputacao();  
If (f instanceof Seguranca){
```

```
.....  
}
```

ProfessorComputacao

.....
.....
.....

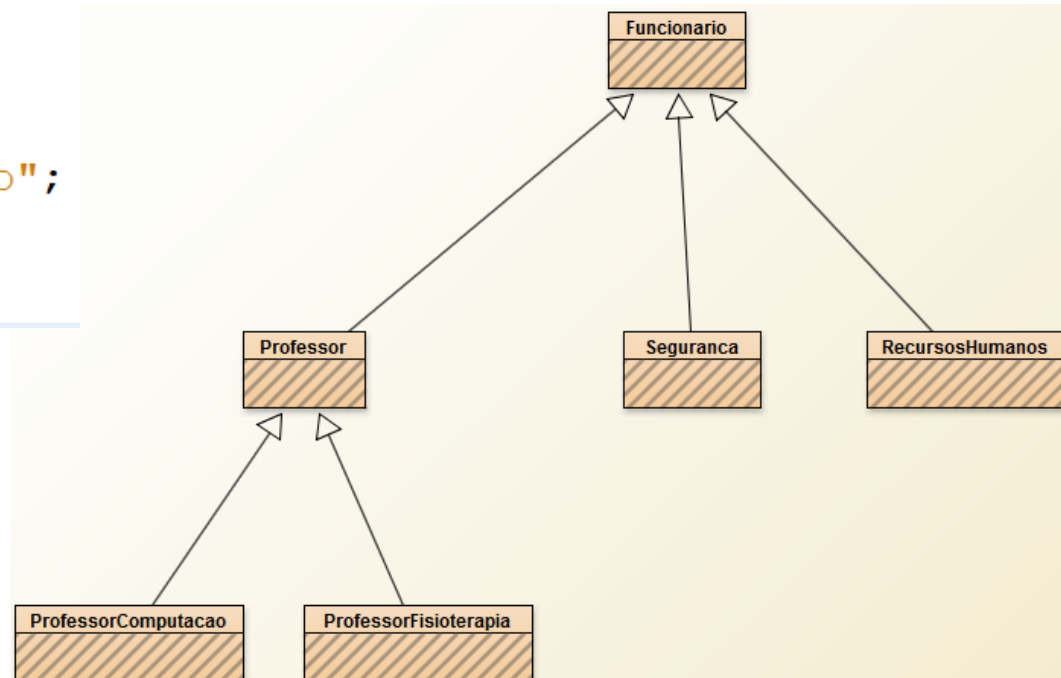
f



Como saber o tipo do objeto?

- Uma outra maneira é definir um atributo tipo, que armazena o tipo de dado da classe

```
public class Funcionario {  
    private String tipo;  
    private String codigo;  
    private String nome;  
    public Funcionario() {  
        this.tipo = "Funcionario";  
    }  
}
```



Como saber o tipo do objeto?

```
public class Funcionario {  
    private String tipo;  
    private String codigo;  
    private String nome;  
    public Funcionario() {  
        this.tipo = "Funcionario";  
    }  
}
```

```
public class Professor extends Funcionario {  
    protected String titulacao;  
    public Professor() {  
        this.tipo = "Professor";  
    }  
  
    public Professor(String codigo, String nome, String titulacao) {  
        this.codigo = codigo;  
        this.nome = nome;  
        this.titulacao = titulacao;  
        this.tipo = "Professor";  
    }  
}
```

Como saber o tipo do objeto?

```
public class Funcionario {
    private String tipo;
}

public class Professor extends Funcionario{
    protected String titulacao;
    public Professor() {
        this.tipo = "Professor";
    }
}

public class ProfessorComputacao extends Professor{
    protected String codigoSBC;
    public ProfessorComputacao() {
        this.tipo = "ProfessorComputacao";
    }

    public ProfessorComputacao(String codigo, String nome, String titulacao,
        String codigoSBC) {
        super(codigo, nome, titulacao);
        this.codigoSBC = codigoSBC;
        this.tipo = "ProfessorComputacao";
    }
}
```

Como saber o tipo do objeto?

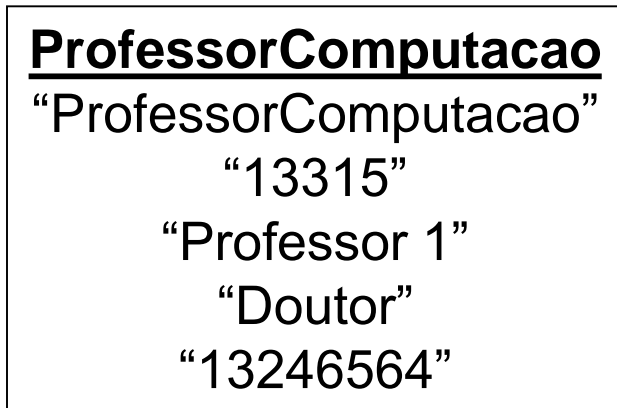
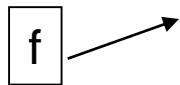
```
public class Funcionario {
    protected String tipo;
    protected String codigo;
    protected String nome;
    public Funcionario() {
        this.tipo = "Funcionario";
    }

    public String getTipo() {
        return tipo;
    }
}
```

Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("ProfessorComputacao")){  
    ...  
}
```

f

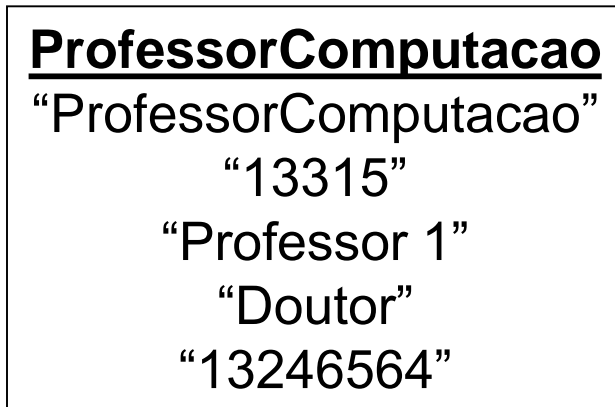


Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("ProfessorComputacao")){  
    ...  
}
```

TRUE

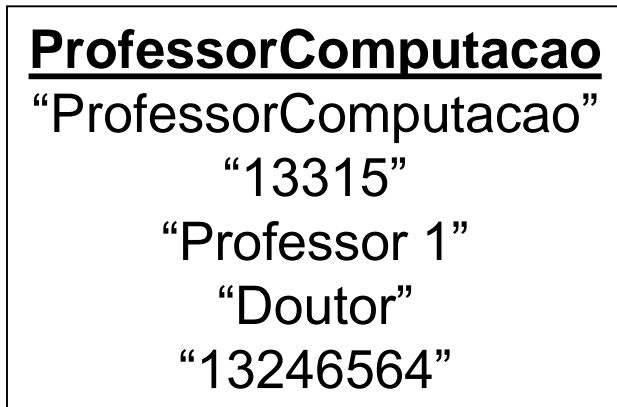
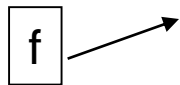
f



Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("ProfessorFisioterapia")){  
    ...  
}
```

f

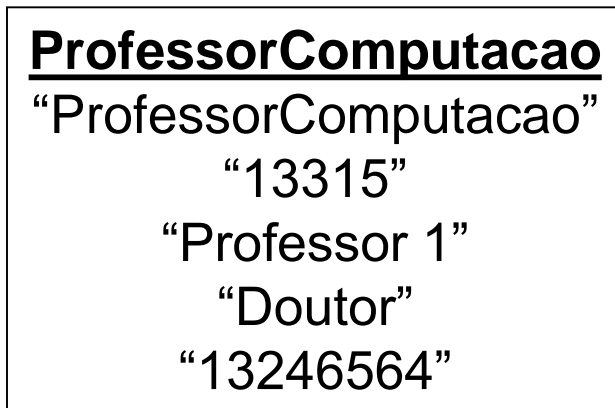


Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("ProfessorFisioterapia")){  
    ...  
}
```

FALSE

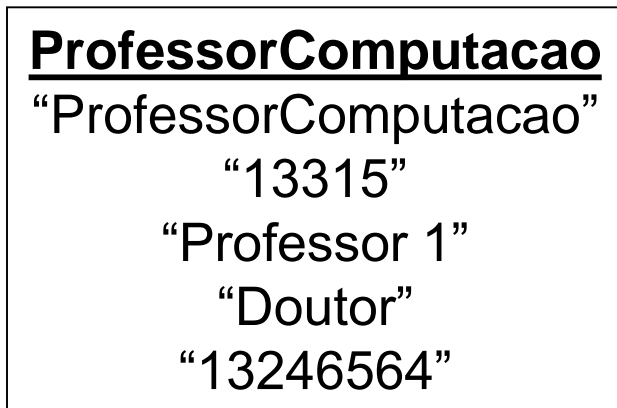
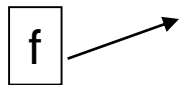
f



Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("Professor")){  
    ...  
}
```

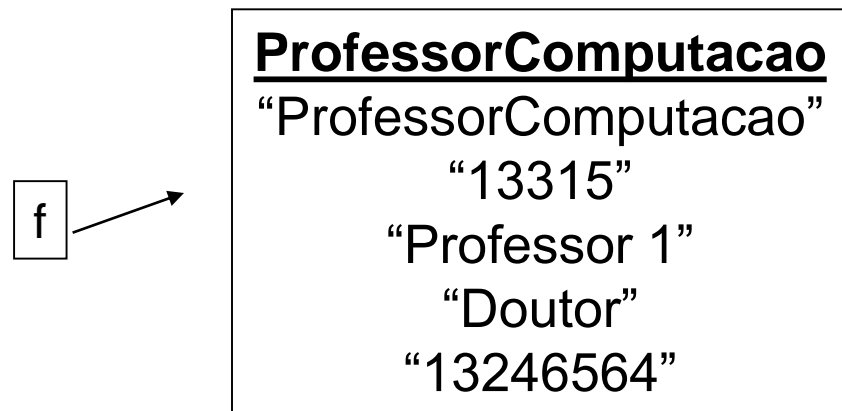
f



Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("Professor")){  
    ...  
}
```

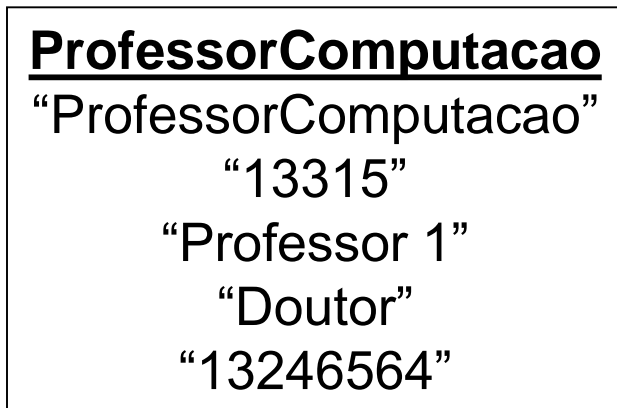
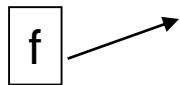
FALSE



Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("Funcionario")){  
    ...  
}
```

f

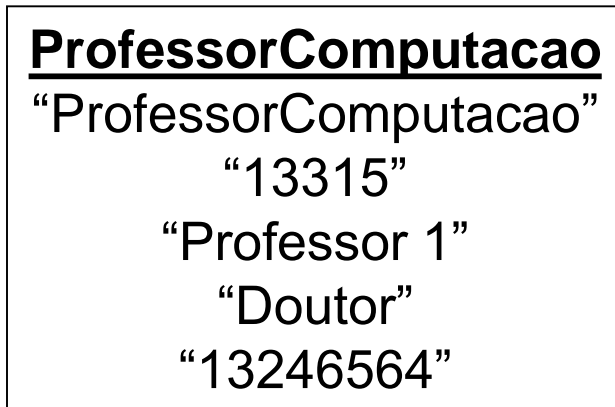


Como saber o tipo do objeto?

```
Funcionario f = new ProfessorComputacao();  
if (f.getTipo().equals("Funcionario")){  
    ...  
}
```

FALSE

f



Casting

- Para acessar um método de uma classe especializada a partir de uma variável cujo tipo de dado é mais alto nível, teremos que forçar o tipo de dado
- Por exemplo, deseja-se pegar um conteúdo específico de ProfessorComputacao
 - As duas maneiras abaixo estão corretas

```
System.out.println("PROFESSORES COMPUTACAO");
for (int i = 0; i < cont; i++) {
    if (vetor[i].getTipo().equals("ProfessorComputacao")) {
        System.out.println("Nome: " + vetor[i].getNome());
        System.out.println("Codigo SBC: " + ((ProfessorComputacao) vetor[i]).getCodigoSBC());
        ProfessorComputacao p = (ProfessorComputacao) vetor[i];
        System.out.println("Codigo SBC: " + p.getCodigoSBC());
    }
}
```


Casting

- Antes de fazer o casting é necessário verificar o tipo de dado do objeto, pois se o tipo estiver errado ou não for possível fazer o casting, será lançada uma exceção

```
for (int i = 0; i < cont; i++) {  
    System.out.println("Nome: " + vetor[i].getNome());  
    System.out.println("Codigo SBC: " + ((ProfessorComputacao) vetor[i]).getCodigoSBC());  
}
```

Casting

- Antes de fazer o casting é necessário verificar o tipo de dado do objeto, pois se o tipo estiver errado ou não for possível fazer o casting, será lançada uma exceção

```
Exception in thread "main" java.lang.ClassCastException: polimorfismo.Funcionario cannot be cast to polimorfismo.ProfessorComputacao
    at polimorfismo.Principal.main(Principal.java:21)
C:\Users\danil\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:111: The following error occurred while executing this line:
C:\Users\danil\AppData\Local\NetBeans\Cache\11.1\executor-snippets\run.xml:94: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

```
for (int i = 0; i < cont; i++) {
    System.out.println("Nome: " + vetor[i].getNome());
    System.out.println("Codigo SBC: " + ((ProfessorComputacao) vetor[i]).getCodigoSBC());
}
```

Polimorfismo

- Parte práctica

Referências

BIBLIOGRAFIA BÁSICA

1. SINTES, A., Aprenda programação orientada a objetos em 21 dias, Pearson Education do Brasil, 2002.
2. VAREJÃO, F., Linguagens de programação : Java, C e C++ e outras : conceitos e técnicas, Campus, 2004.
3. DEITEL, H. M., DEITEL, P. J., **Java:** como programar, São Paulo: Pearson Education do Brasil, 2010. 1144p.
4. DEITEL, H. M., DEITEL, P. J., **Java:** como programar, Porto Alegre: Bookman, 2003. 1386p.
5. SAVITCH, W. J., C++ absoluto, Pearson Education : Addison Wesley, 2004.

Capítulo 9

BIBLIOGRAFIA COMPLEMENTAR

1. BERMAN, A. M. *Data Structures via C++: Objects by Evolution*, Oxford University Press Inc., 1997.
2. BARNES, D.J. & KÖLLING, M., Programação orientada a objetos com Java, Pearson Education : Prentice Hall, 2004.
3. DEITEL, H. M. e DEITEL, P. J. *C++: Como Programar*, Bookman, 2001.
4. GILBERT, R. F. e FOROUZAN, B. A. *Data Structures: A Pseudo Approach with C++*, Brooks/Cole Thomson Learning, 2001.
5. MUSSER, D. R. e SAINI, A. *STL Tutorial and Reference Guide: Programming with the Standard Template Library*, Addison-Wesley, 1996.
6. SEBESTA, R. W. *Conceitos de Linguagem de Programação*, 4ª Ed., Bookman, 2003.
7. SEDGEWICK, R. *Algorithms in C++*, Addison-Wesley, 2002.
8. STROUSTRUP, B. *A Linguagem de Programação C++*, 3ª Ed., Bookman, 2000.

FCT/Unesp – Presidente Prudente
Departamento de Matemática e Computação

Programação Orientada a Objetos Polimorfismo

Prof. Danilo Medeiros Eler
danilo.eler@unesp.br