

4 – Conceito de Herança

Hierarquia de classes e mecanismo de ligação

Herança – Uma classe pode herdar operações de uma superclasse e as suas operações podem ser herdadas por subclasses.

O mecanismo de herança permite definir uma nova classe em termos de uma classe existente, com modificações e/ou extensões de comportamento.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

A nova classe é a subclasse da anterior ou classe derivada.

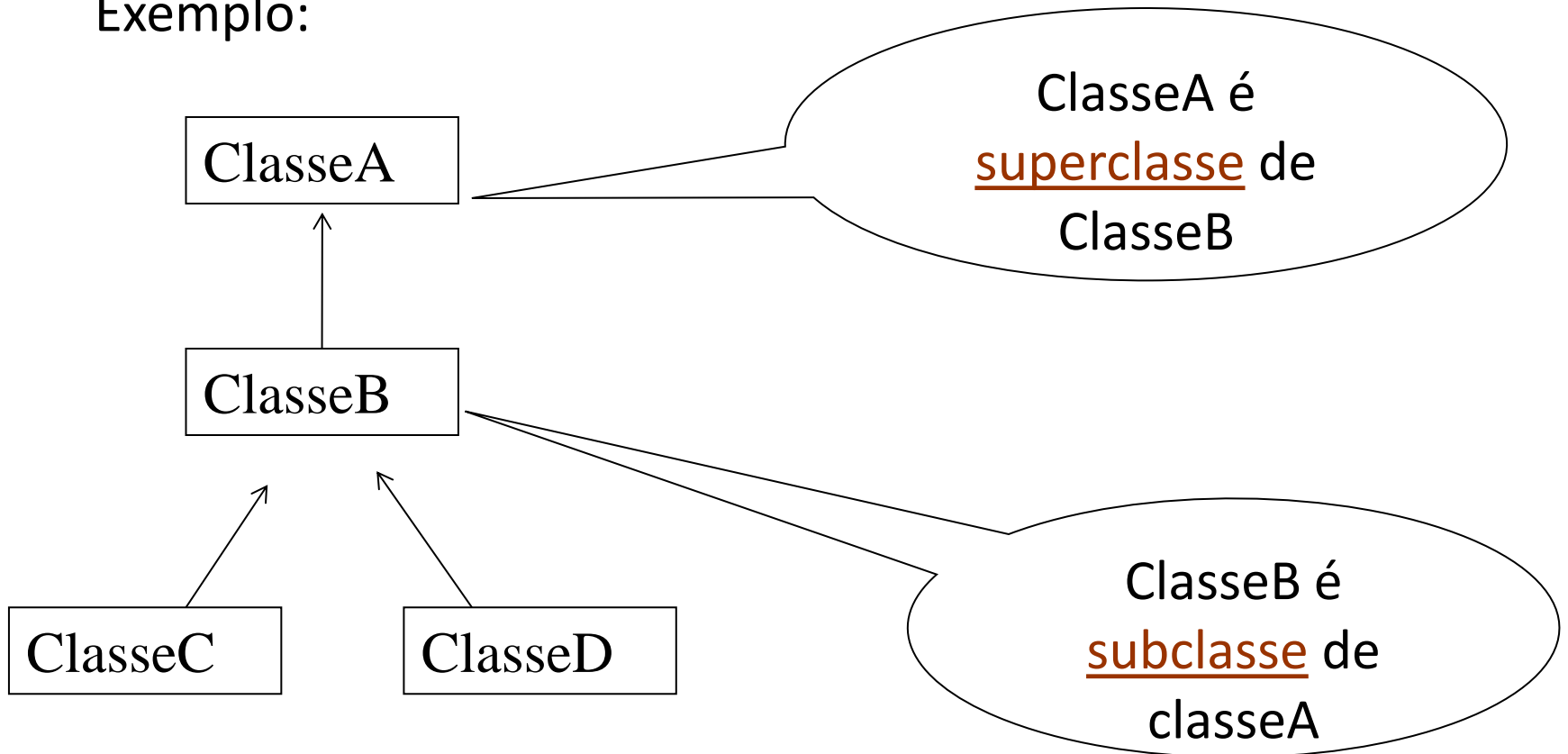
A classe inicial é a superclasse ou classe base.

- Pode repetir-se o processo, definindo uma nova classe a partir da classe derivada anterior ...

Construindo uma hierarquia de classes →

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Exemplo:



Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Todos os métodos e atributos da superclasse vão ser herdados pela subclasse.

À subclasse, podem ser adicionados novos métodos e novos atributos num processo de especialização sucessiva.

Dada uma hierarquia de classes,

. uma instância de uma subclasse vai conter →

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

- as variáveis de instância da superclasse (ou superclasses)

mais

- as variáveis de instância declaradas na classe derivada (subclasse).

. O comportamento dessa instância está definido

na sua classe

e

no conjunto das suas superclasses.

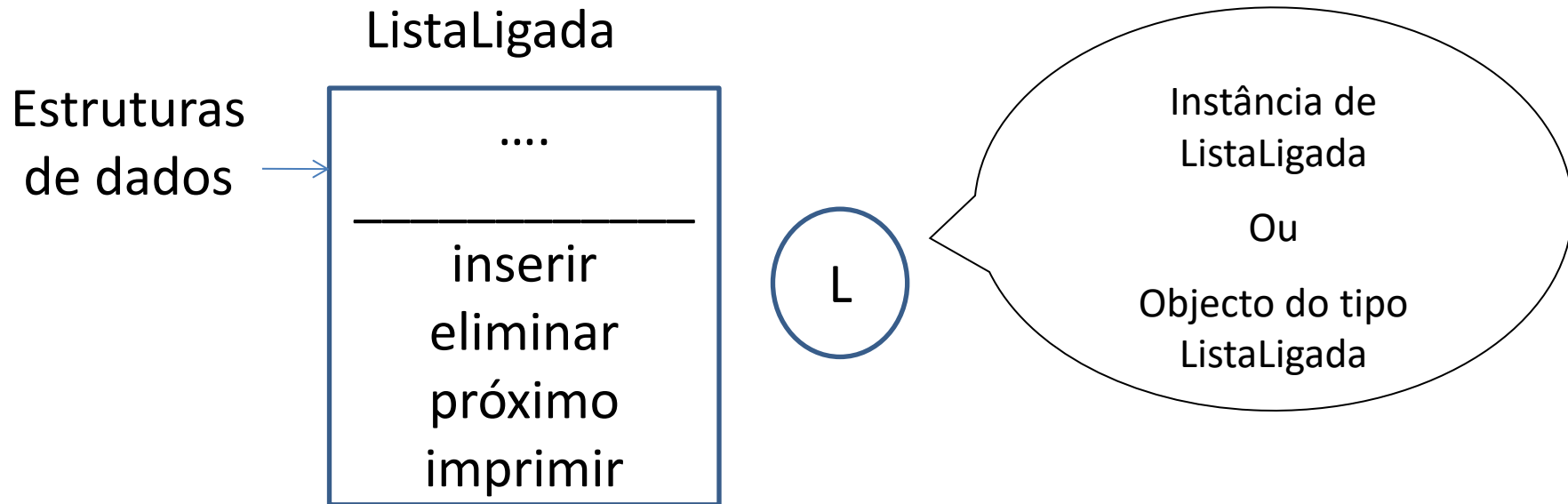
Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Quando um método é invocado, isto é, quando é enviada uma mensagem a um objecto, torna-se necessário ligar a mensagem à correspondente implementação:

(Por outras palavras, associar (ligar) a assinatura do método ao código que o implementa)

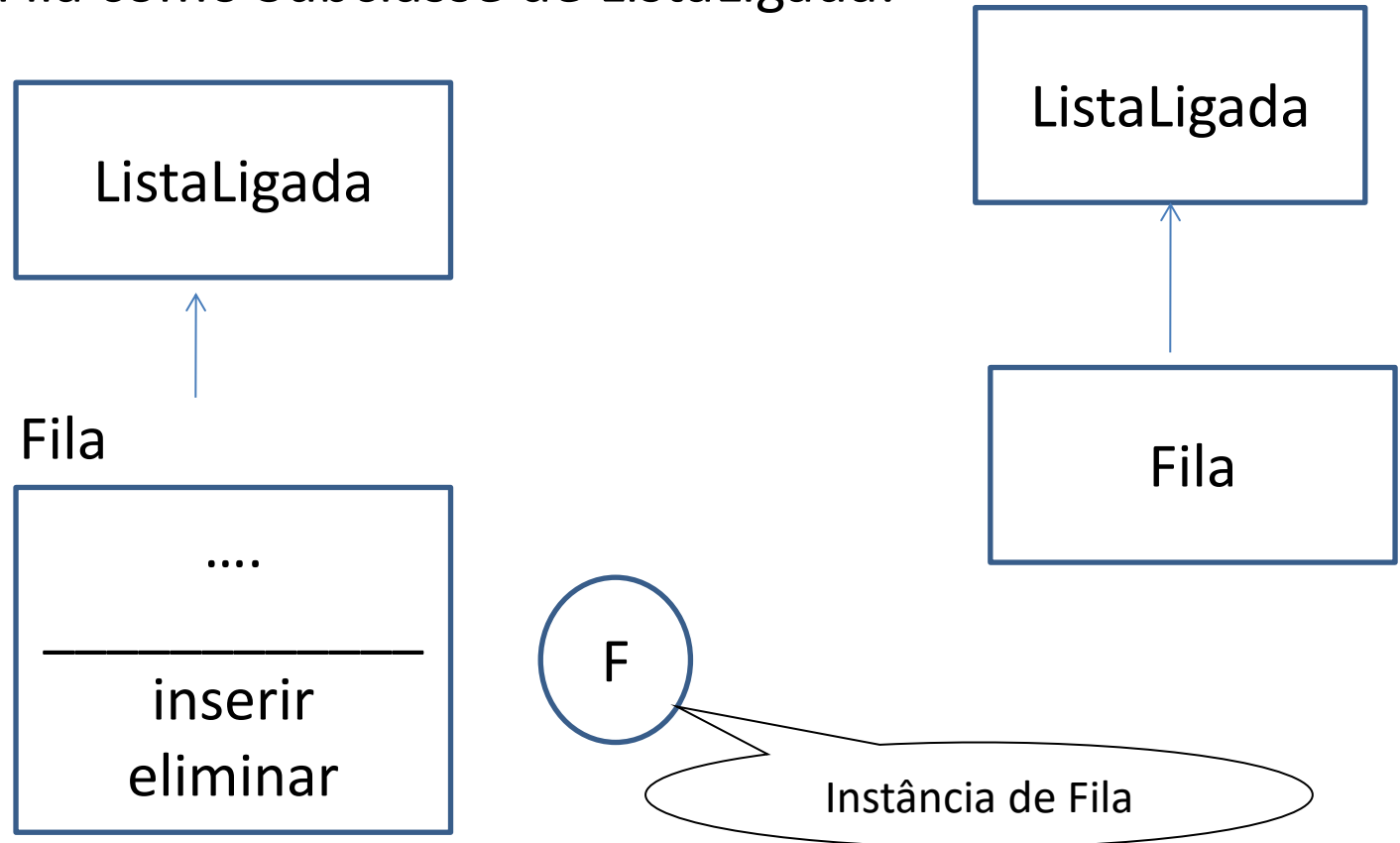
Mecanismo de Ligação →

Suponhamos uma classe ListaLigada:



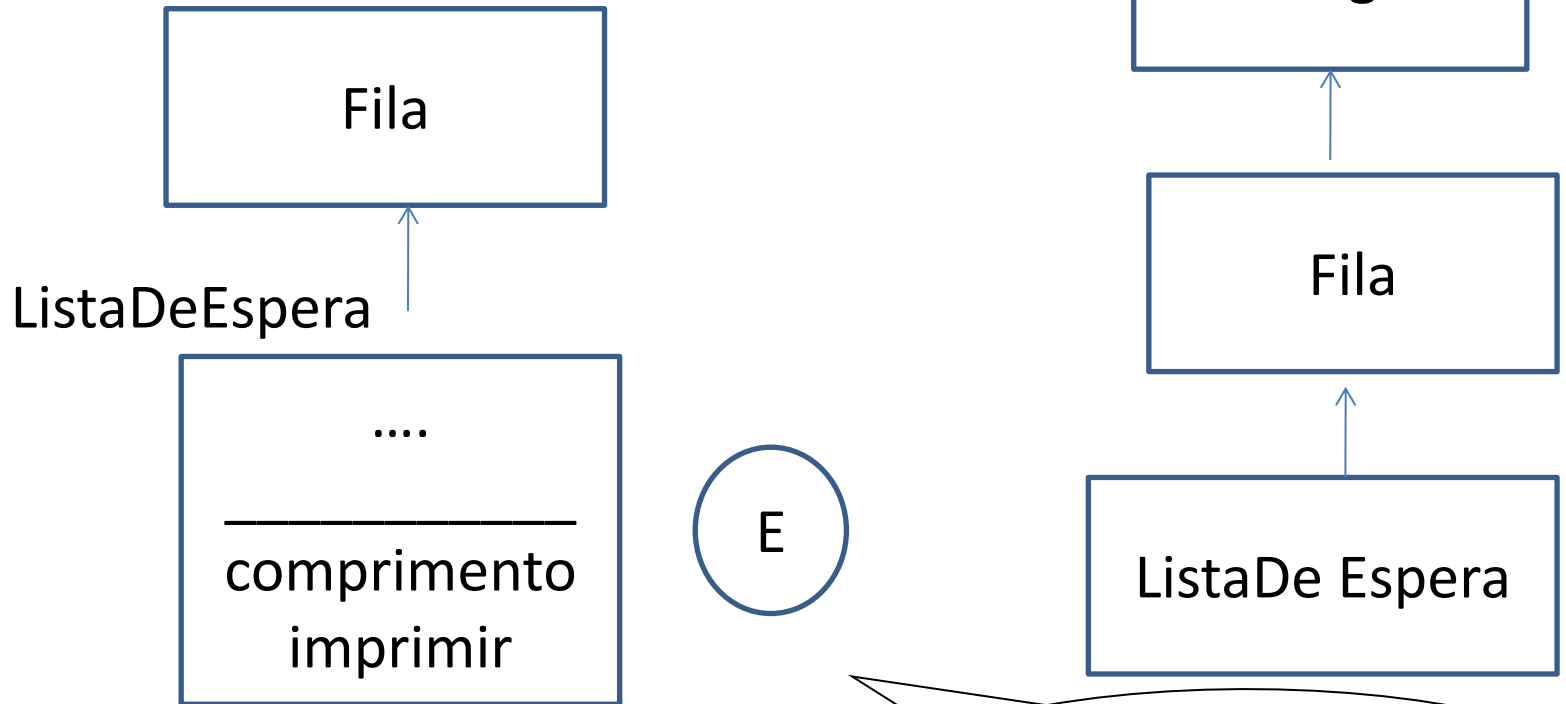
- Uma Fila pode ser facilmente implementada a partir de (isto é, reutilizando a implementação de) uma lista ligada desde que se imponham as restrições adequadas à manipulação dos seus elementos.

Definimos Fila como Subclasse de ListaLigada:



- Redefinimos os métodos Inserir e Eliminar para reflectirem a semântica da Fila.

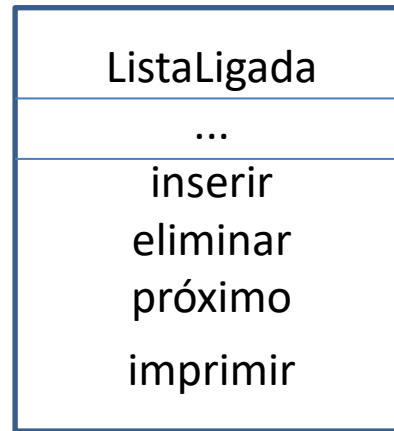
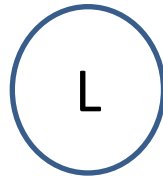
Podemos definir uma classe ListaDeEspera
Como subclasse de Fila:



- Adicionamos o método comprimento
- Redefinimos o método imprimir

Instância de
ListaDeEspera

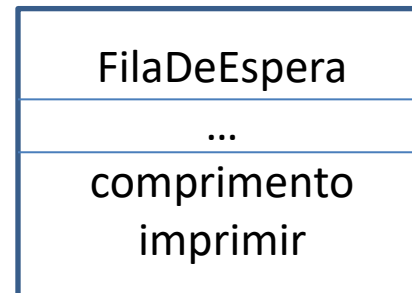
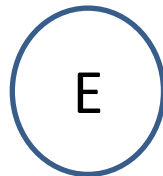
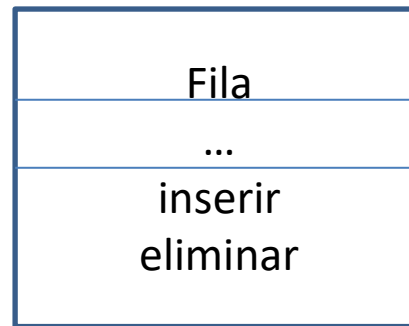
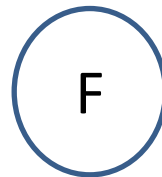
Construímos a hierarquia:



← Classe

← Atributos

← Métodos



Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Suponhamos agora as situações:

1º - A mensagem “imprimir” é enviada ao objecto F

F.imprimir()

- Primeiro é pesquisada a classe Fila e só depois a classe ListaLigada onde o método é encontrado.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

2º - A mensagem “imprimir” é enviada ao objecto E

E.imprimir()

- O método é imediatamente encontrado na classe ListaDeEspera.

3º - A mensagem “inserir” é enviada ao objecto E

E.inserir()

- A classe ListaDeEspera é pesquisada em primeiro lugar, segue-se a classe Fila onde o método é encontrado.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Resumindo:

A hierarquia é pesquisada, em direcção à superclasse, com início na classe do objecto que recebe a mensagem.

O método mais próximo é o executado.

Observação: Algumas linguagens permitem explicitar o ponto de início da pesquisa através da especificação da superclasse juntamente com o nome do método (ex.lo: C++), Java?

Tipos de ligação

A ligação do nome de um método a uma implementação pode ser feita

- em tempo de compilação (ligação estática)

ou

- em tempo de execução (ligação dinâmica)

Ligação estática

Abordagem mais simples.

- O compilador constrói uma tabela de classes e métodos associados. O código produzido contém as ligações entre os nomes dos métodos e as correspondentes implementações .

Vantagem:

- ligações erradas (isto é, chamadas de métodos não existentes) são detectadas em tempo de compilação.

Desvantagem:

- para introduzir alterações na ligação é necessário recompilar todo o código.

Ligação dinâmica

A correspondência entre o nome do método e a sua implementação é feita em cada invocação.

- A hierarquia é pesquisada, se o método não existir é devolvida uma mensagem do tipo “método desconhecido”.

Vantagem:

- alterações na hierarquia reflectem-se na ligação sem necessidade de recompilar todas as classes.

Ligação dinâmica (cont ...)

Desvantagens:

-a pesquisa na hierarquia provoca alguma degradação no desempenho do sistema;

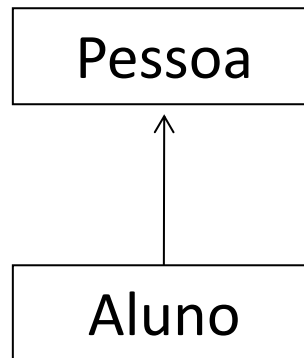
- necessidade de manipular mensagens
“MétodoDesconhecido” em tempo de execução.

Herança de classes em Java

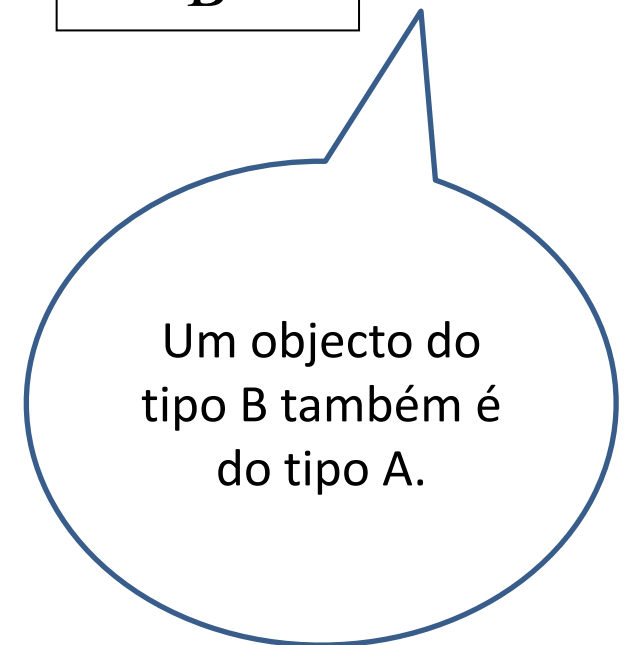
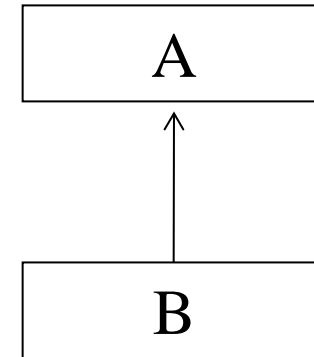
Declarar B como subclasse de A:

```
public class B extends A { ...
```

Ex.lo:



Um Aluno também é Pessoa.



Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Notas:

- Cada classe possui uma e uma só superclasse directa.
- Apenas a superclasse directa é identificada na cláusula *extends*
- A classe de topo da hierarquia é a classe **Object**.
- Quando a cláusula extends não é usada significa que a classe é subclasse directa da classe Object.

A classe Object

define o comportamento comum a todas as classes.

```
public class Object {  
  
    public final Class getClass()  
        // devolve a classe do objecto  
  
    public String toString()  
        // representação textual do objecto  
  
    public boolean equals( Object obj) ...  
        // igualdade de referências  
  
    protected Object clone()  
        // clonagem, cria uma cópia do objecto  
  
    ...  
  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

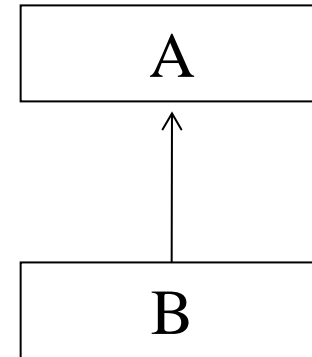
A classe Object define métodos genéricos que normalmente necessitam de ser redefinidos.

- Qualquer instância de qualquer classe pode responder às mensagens correspondentes aos métodos públicos da classe Object.

- Se algum método não foi redefinido na classe do utilizador será executado o código definido na classe Object.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Dada uma classe A e uma subclasse B,



- B tem acesso directo a todas as variáveis e métodos da instância de A que não sejam declarados como private.
- B pode definir novas variáveis e novos métodos.
- B pode redefinir variáveis e métodos herdados.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

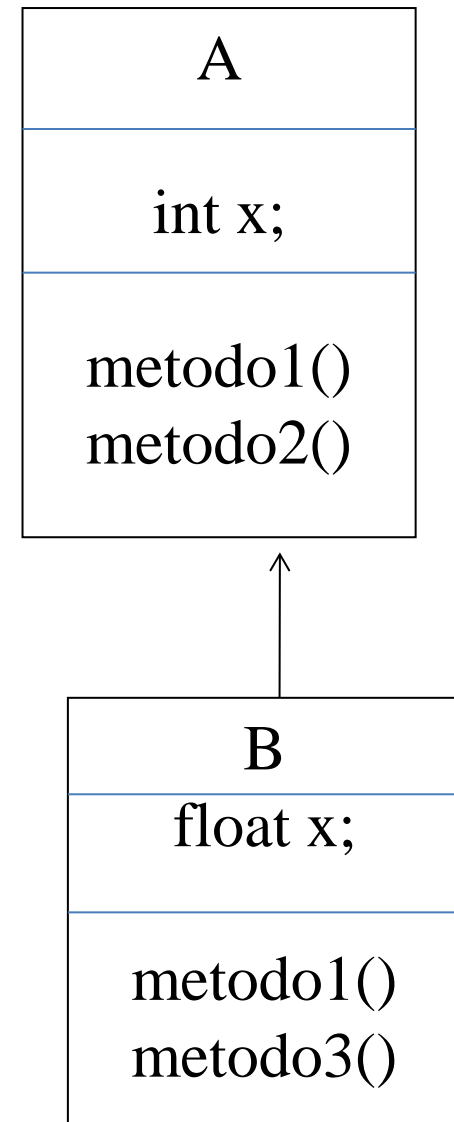
- Uma instância de B pode responder a mensagens que correspondam a todos os métodos públicos de B e de A.

- Os atributos de uma instância de B são os atributos definidos nas classes A e B.

Supondo,

```
B b = new B();
```

Quais são as variáveis e os métodos de b ?



Princípio da substitutividade:

“Declarada uma variável como sendo de uma dada classe é permitido atribuir-lhe um valor de sua classe ou de qualquer sua subclasse”.

A a1, a2;

a1= new A();

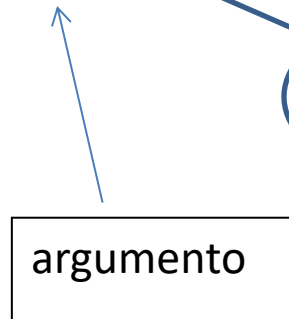
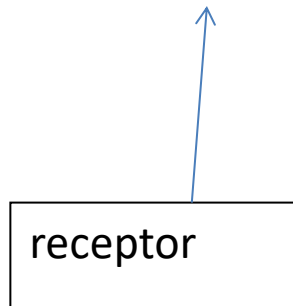
a2 = new B(); // atribuição válida

Métodos equals e clone

Comparar objectos: método equals

-o método “public boolean equals (Object)” da classe Object compara a referência do objecto, que recebe como argumento, com a referência do objecto receptor.

boolean b = c1.equals(obj);



Devolve true se as referências forem iguais, false caso contrário

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Vamos redefinir o método “equals” para a classe Contador de tal forma que dois objectos do tipo Contador são iguais se o seu estado for igual.

Isto é, dois objectos do tipo Contador serão iguais, se as suas variáveis **conta** tiverem o mesmo valor.

Antes de testarmos o valor das variáveis, vamos testar se o argumento é diferente de null (isto é, se o objecto foi instanciado) e se o argumento e o recetor são objetos da mesma classe.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public boolean equals (Object obj ) {  
    if (obj != null && this.getClass() == obj.getClass() ){  
        // compara as variáveis de instância dos dois objectos  
        return ( this.conta == ( (Contador) obj).conta ) ; ←  
    } else {  
        return false;  
    }  
}
```

(&&) E condicional (Porquê ?)

Mais fácil:

```
public boolean equals (Object obj ) {  
    if (obj != null && this.getClass() == obj.getClass() ){  
        // compara as variáveis de instância dos dois objectos  
        Contador x = (Contador) obj;  
        return ( this.conta == x.conta ) ;  
    } else {  
        return false;  
    }  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Utilização do método:

...

Contador c1,c2;

c1= new Contador();

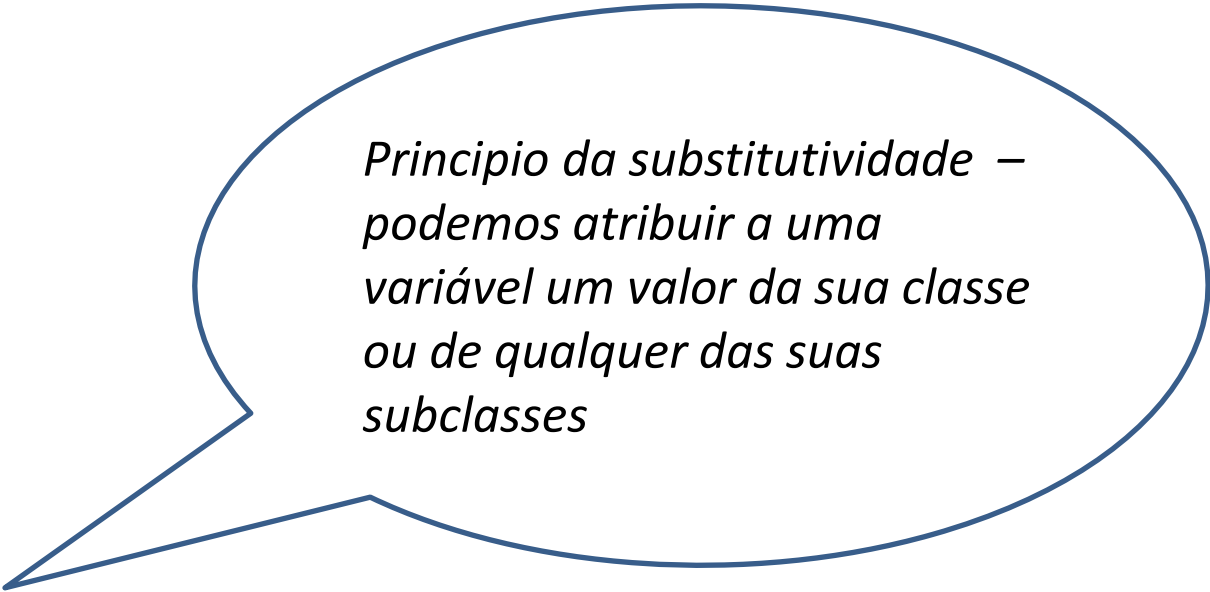
c2= new Contador ();

...

boolean iguais;

iguais = c1.equals(c2);

...

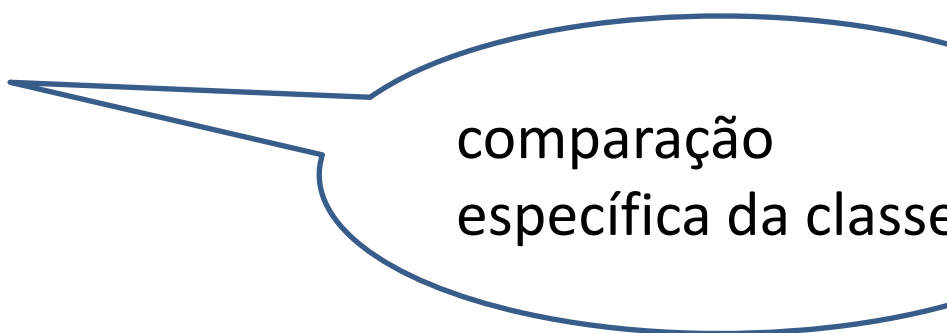


*Principio da substitutividade –
podemos atribuir a uma
variável um valor da sua classe
ou de qualquer das suas
subclasses*

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Para uma qualquer classe Exemplo:

```
public boolean equals (Object obj) {  
    if ( obj != null && this.getClass() == obj.getClass() ) {  
        return ...  
    }  
    return false;  
}
```



comparação
específica da classe

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Método **equals** para a classe **Telefone** definida em TP04:

```
public boolean equals (Object obj) {  
  
    if ( obj != null && this.getClass() == obj.getClass() )  
        Telefone tel = (Telefone) obj;  
        return ( this.numero == tel.numero &&  
                this.tipo.equals ( tel.tipo) );  
  
    else  
        return false;  
}
```

O método Clone

Queremos definir um método que crie e devolva uma cópia do objecto receptor.

-Essa cópia deve ser tal que o objecto criado e o objecto que recebe a mensagem:

1 – não são o mesmo objecto

```
y = x.clone(); // y != x;
```


Programação Orientada a Objectos - P. Prata, P. Fazendeiro

2 – são instâncias da mesma classe

```
x.clone().getClass() == x.getClass()
```

3 – têm o mesmo valor nas variáveis de instância

```
x.clone().equals( x ) == true
```

```
public Object clone() {
```

```
    Contador c = new Contador ( this.conta);
```

```
    return c;
```

```
}
```

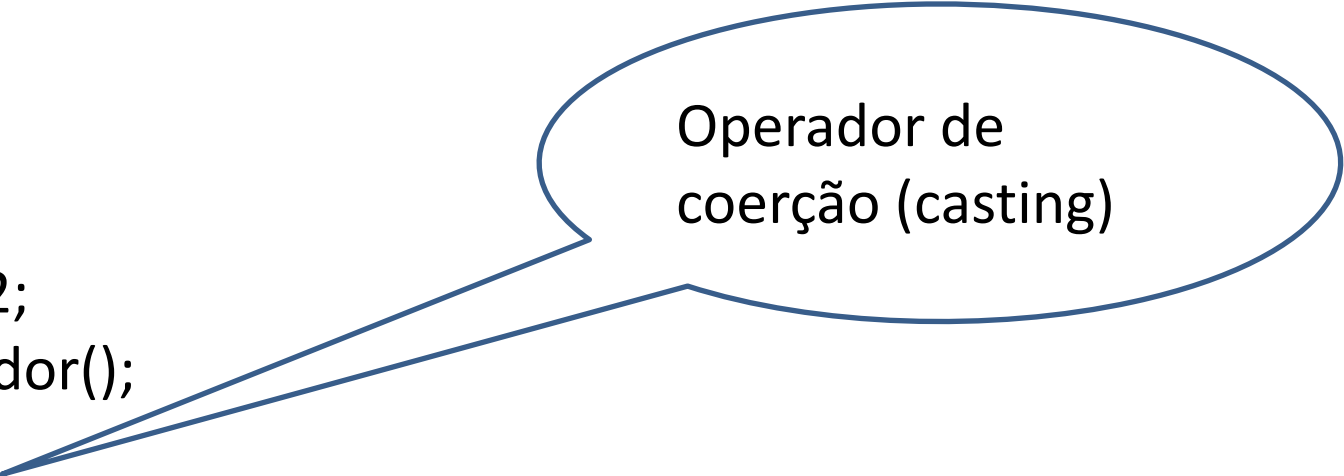
Utilização:

```
Contador c1,c2;
```

```
c1=new Contador();
```

```
...
```

```
c2 = (Contador) c1.clone();
```



Operador de
coerção (casting)

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Método **clone** para a classe **Telefone** definida em TP04:

```
public Object clone () {
```

```
    Telefone copia = new Telefone ();  
    copia .numero = this.numero;  
    copia.tipo = this.tipo;
```

```
    return copia;
```

```
}
```

ou

```
Telefone copia = new telefone ( this.tipo, this.numero)
```

Comparação de Strings

1 - Valores constantes do tipo String têm a mesma referência.

`"XPTO" == "XPTO"` → expressão com valor true

2 – Strings construídas em tempo de compilação são tratadas como valores constantes do tipo String.

```
String s1 = "XPTO";
```

```
String s2 = "XPTO"
```

`s1 == s2` → expressão com valor true

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

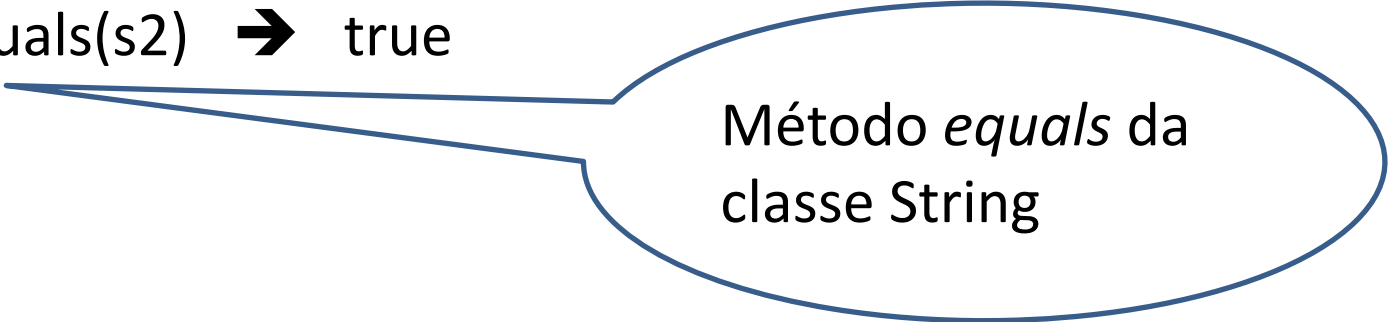
3 – Strings construídas em tempo de execução têm referências distintas:

```
String s1 = new String( "XPTO");  
String s2 = new String ("XPTO");
```

`s1 == s2` → `false`

mas

`s1.equals(s2)` → `true`



Método *equals* da classe String

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Ex.lo 1

```
public class Exemplo {  
  
    static String s0;  
  
    public static void setS0 (String s){  
  
        s0 = s;  
  
    }  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

// ainda na classe exemplo.

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

```
    String s = “XPTO”;
```

```
    setS0 (s);
```

```
    System.out.println ( s + “    “ + s0);
```

```
    System.out.println ( s == s0);
```

```
    s = “XX”;           //é criada uma nova instância da String s;
```

```
    System.out.println ( s + “    “ + s0)
```

```
    }
```

```
} // fim da classe exemplo
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Output:

XPTO XPTO

true

XX XPTO

E se no método **setS0**

substituírmos

s0 = s

por

s0= new String (s);

O que acontece?

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

`s == s0` `???`

`s.equals(s0)` `????`

Exercício: testar exemplo anterior

Qual o output do seguinte programa:

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

```
public static void main(String[] args) {  
    String s1 = new String ("XPTO");  
    String s2 = "XPTO";  
    System.out.println ( s1 == s2);  
    System.out.println ( s1.equals(s2) );  
  
    String s3 = "XPTO";  
    System.out.println ( s2 == s3);  
    System.out.println ( s2.equals(s3) );  
}
```

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

false

true

true

true

Regra:

- comparar Strings sempre com o método equals definido na classe String.

Programação Orientada a Objectos - P. Prata, P. Fazendeiro

Exercício – Considere a classe Exemplo,

```
public class Exemplo{  
    private int n  
    private String s;  
    private Telefone [] listaE;    // dimensão 10  
    private ArrayList<Telefone> listaD;
```

Construa:

- a) Construtor de omissão;
- b) getters e setters;
- c) método equals;
- d) método clone;
- e) método toString