4 – Conceito de Herança

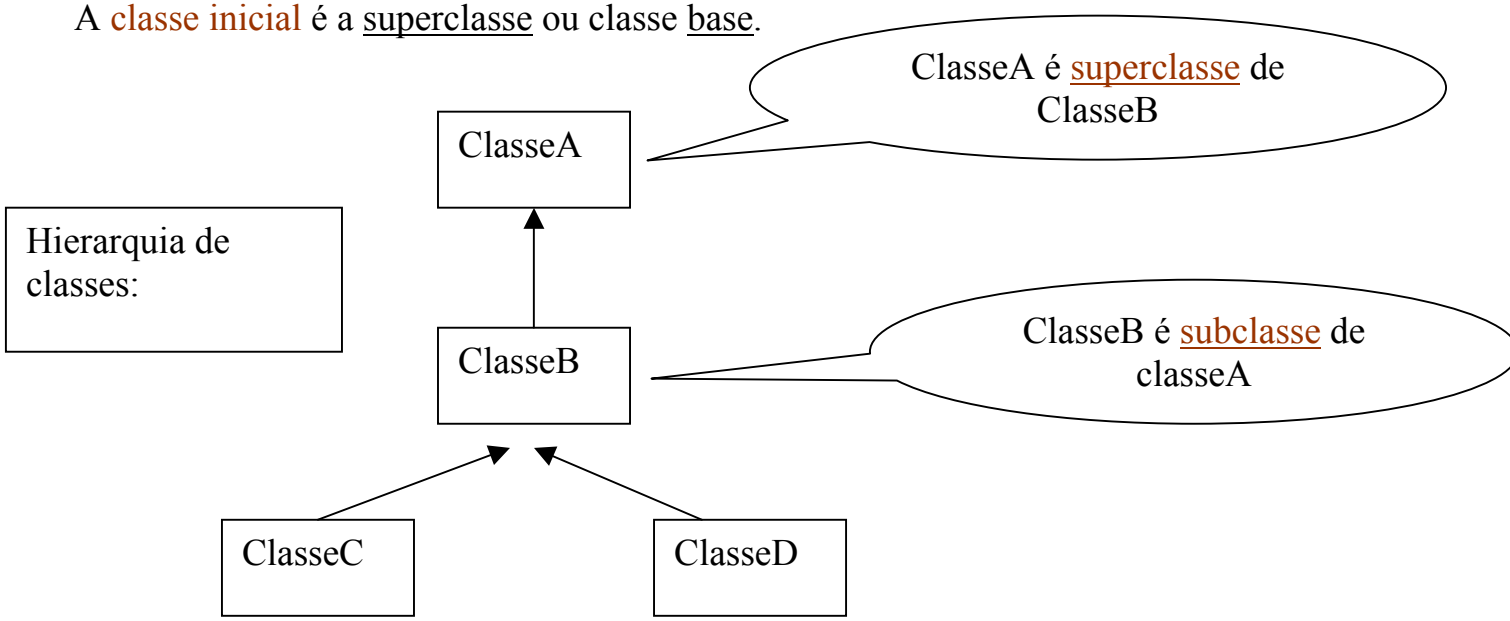
Hierarquia de classes e mecanismo de ligação

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.

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 ...

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:

- 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

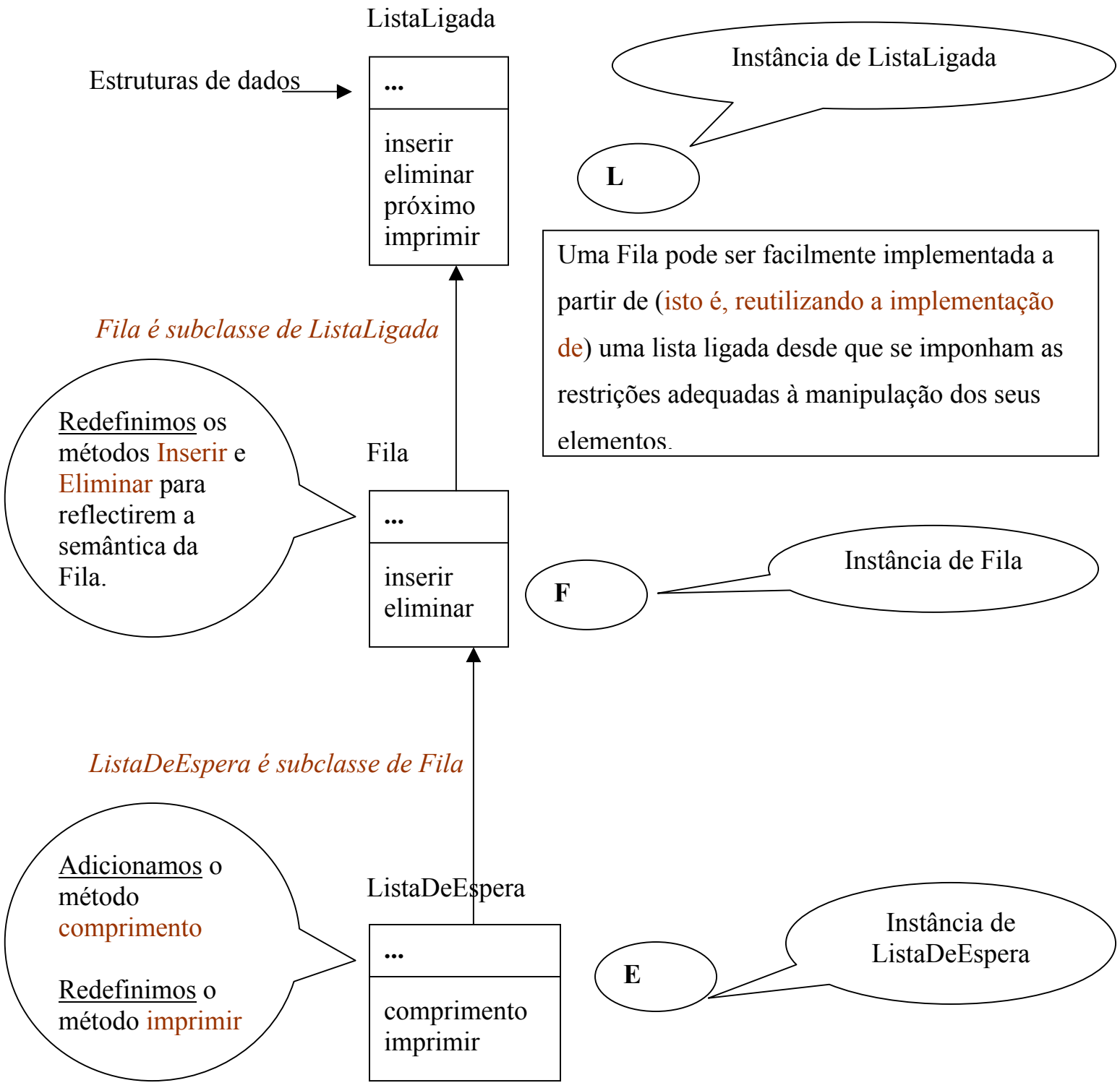
e

- no conjunto das suas superclasses.

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

Mecanismo de Ligação ➔

Suponhamos a hierarquia de classes:



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.

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.

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.

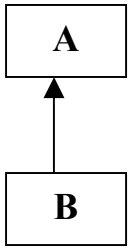
Desvantagens:

- 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 { ...
```

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

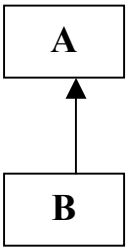
A classe Object define o comportamento comum a todas as classes.

```
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

    ...

}
```

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.

Dada uma classe A e uma subclasse B,



- B tem acesso directo a todas as variáveis e métodos da instância que não sejam declarados como private.

- B pode definir novas variáveis e novos métodos.

- B pode redefinir variáveis e métodos herdados.

- 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.

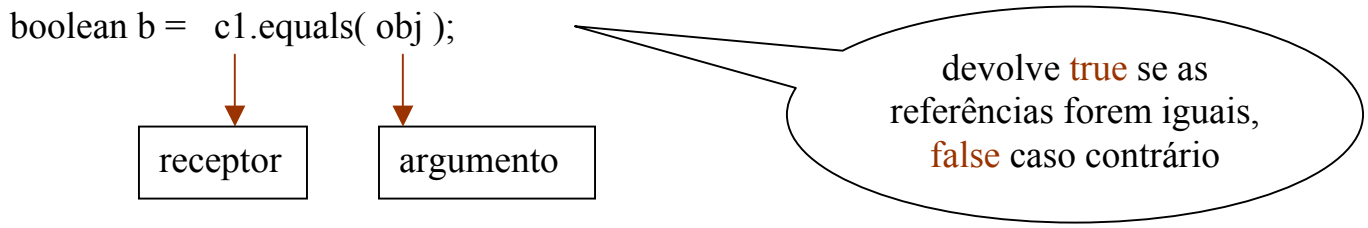
Principio da substitutividade:

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

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.



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.

```
public boolean equals (Object  obj ) {
// se o argumento for diferente de null e (*) for uma instância da classe Contador
   if (obj != null && obj instanceof Contador ){
      // compara as variáveis de instância dos dois objectos
         return ( this.conta == ( (Contador)obj ).conta
   } else {
         return false;
   }
```

conta é atributo da classe Contador

(*) E condicional

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 uma valor da sua classe ou de qualquer das suas subclasses

Para uma qualquer classe Exemplo:

```
public boolean equals (Object obj) {
    if ( obj != null && obj instanceof Exemplo ){
                        comparação
                    específica da classe
        return ...
     }
     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;

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() {
 return (new Contador(this.conta) );
}
```

Utilização:

Contador c1,c2;

c1=new Contador();

...

c2 = (Contador) c1.clone();

operador de coerção

Propostas de trabalho:

- Construa uma classe que represente um aluno da UBI. O número de cada aluno deverá ser atribuído automaticamente, de forma sequencial, cada vez que é criado um novo objecto do tipo aluno.

- Construa uma classe que represente um jogador de futebol. Defina um construtor, os métodos de consulta (getters) e de modificação (setters), o método toString, o método equals e o método clone.