

# Composição e Herança

---

## Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira  
oliveira.edmar@ufjf.edu.br

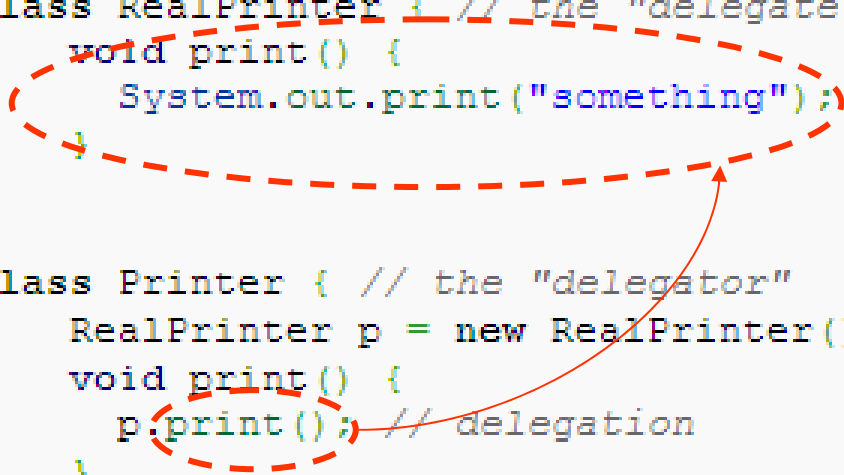
Universidade Federal de Juiz de Fora - UFJF  
Departamento de Ciência da Computação - DCC

# Delegação

```
class RealPrinter { // the "delegate"
    void print() {
        System.out.print("something");
    }
}

class Printer { // the "delegator"
    RealPrinter p = new RealPrinter(); // create the delegate
    void print() {
        p.print(); // delegation
    }
}

public class Main {
    // to the outside world it looks like Printer actually prints.
    public static void main(String[] args) {
        Printer printer = new Printer();
        printer.print();
    }
}
```



# Reuso

## ■ Introdução

- Composição e herança são dois mecanismos para reutilizar funcionalidades. A Herança sempre foi considerada a ferramenta básica de extensão e reuso de funcionalidade. Contudo, outras formas para realização de reuso podem ser consideradas. A composição é uma.
- A composição estende uma classe pela **delegação de trabalho** para outro objeto
- A herança estende atributos e métodos de uma classe

# Composição e Herança

- Uso de Composição ou Herança
  - Existem situações que levam o programador a ter de escolher entre projetar uma classe baseada em composição ou herança. Na maior parte das vezes, a solução mais comum é agrupar classes existentes em novas funcionalidades para criar novas classes, ou seja, utilizar composição.
  - Em outras situações, uma análise levará à percepção de que o uso de herança será necessário.

# Composição e Herança

## ■ Uso de Composição ou Herança

- A utilização de herança ou composição corresponde a se perguntar se a classe a ser criada nunca necessitará de um **upcast** para a suposta superclasse, ou seja, se ela nunca precisará “assumir” o tipo da superclasse em alguma situação.
- O upcast é justamente isso: em qualquer momento, um **objeto da subclasse** pode ser utilizado como se fosse um **objeto da superclasse**.

```
Gato bichano = new Gato();  
Mamifero m = bichano();
```

```
Cd cd1 = new CD();  
Item item = cd1;
```

Objeto de subclasse está sendo usado como um objeto de superclasse.

Gato não deixa de ser gato com upcasting. Ele apenas passa a ser tratado como todo mamífero (não há especialização para gato. Por isso, o objeto m não enxerga os atributos de bichano).

O mesmo vale para CD. Ele apenas passa a ser visto como um item e não mais como algo especializado (CD)

# Composição e Herança

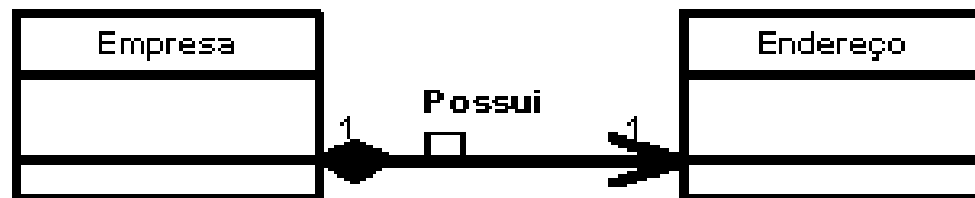
## ■ Concluindo

- O programador, desta forma, deve concluir que se a classe a ser criada nunca precisa assumir o referido tipo da superclasse, provavelmente esta situação será melhor modelada se esta **superclasse for apenas um atributo**, caracterizando assim uma situação de uso de **composição**.
- Exemplo (uso de herança)
  - Superclasse Items e subclasses CD e DVD
  - Vetor de Items - onde são armazenados CDs e DVDs

# Composição

## ■ Quando Usar

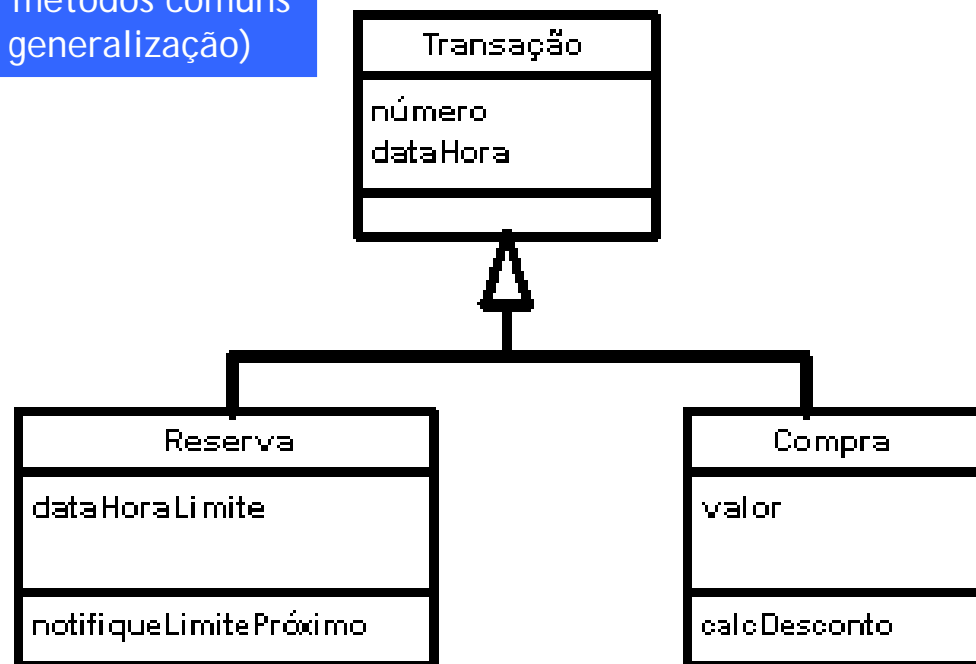
- Use composição para estender as responsabilidades pela **delegação de trabalho** a outros objetos. Ex: domínio de endereços: Uma empresa "tem" um endereço. Podemos deixar o objeto empresa responsável pelo objeto endereço e temos agregação composta (composição)



O endereço como objeto do mundo real não é "destruído" junto com Empresa, ou seja, o endereço físico continua existindo. Contudo, como objeto pertencente à empresa, ele deixa de existir. Ao romper a ligação entre empresa e endereço, endereço deixa de ser referenciado.

# Herança

Atributos, conexões a objetos e métodos comuns vão na superclasse (classe de generalização)



Adicionamos especializações nas subclasses

Vantagem Herança: Captura o que é comum e o isola daquilo que é diferente

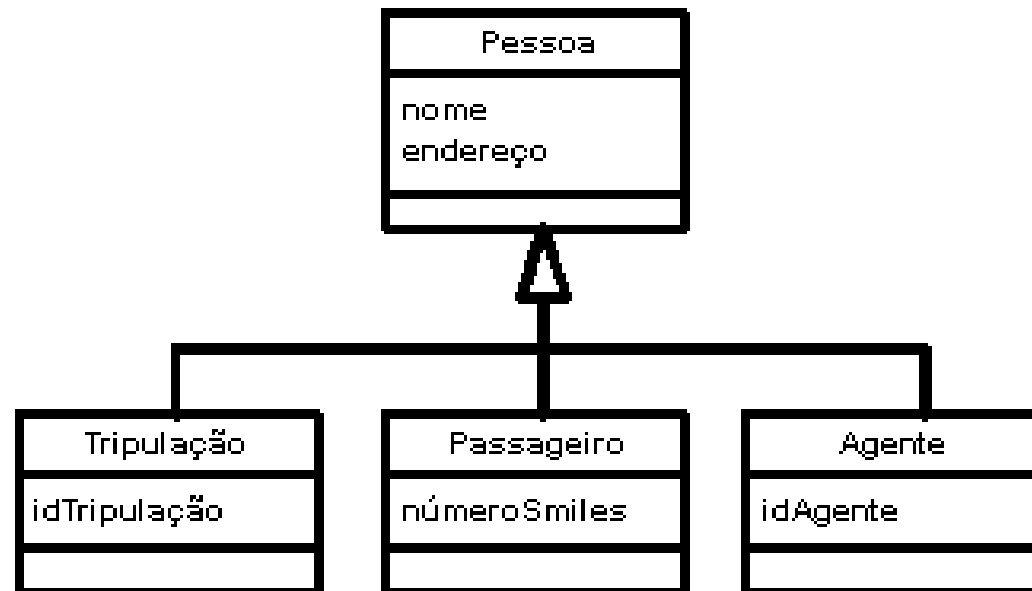


# Herança

## ■ Problemas com Herança

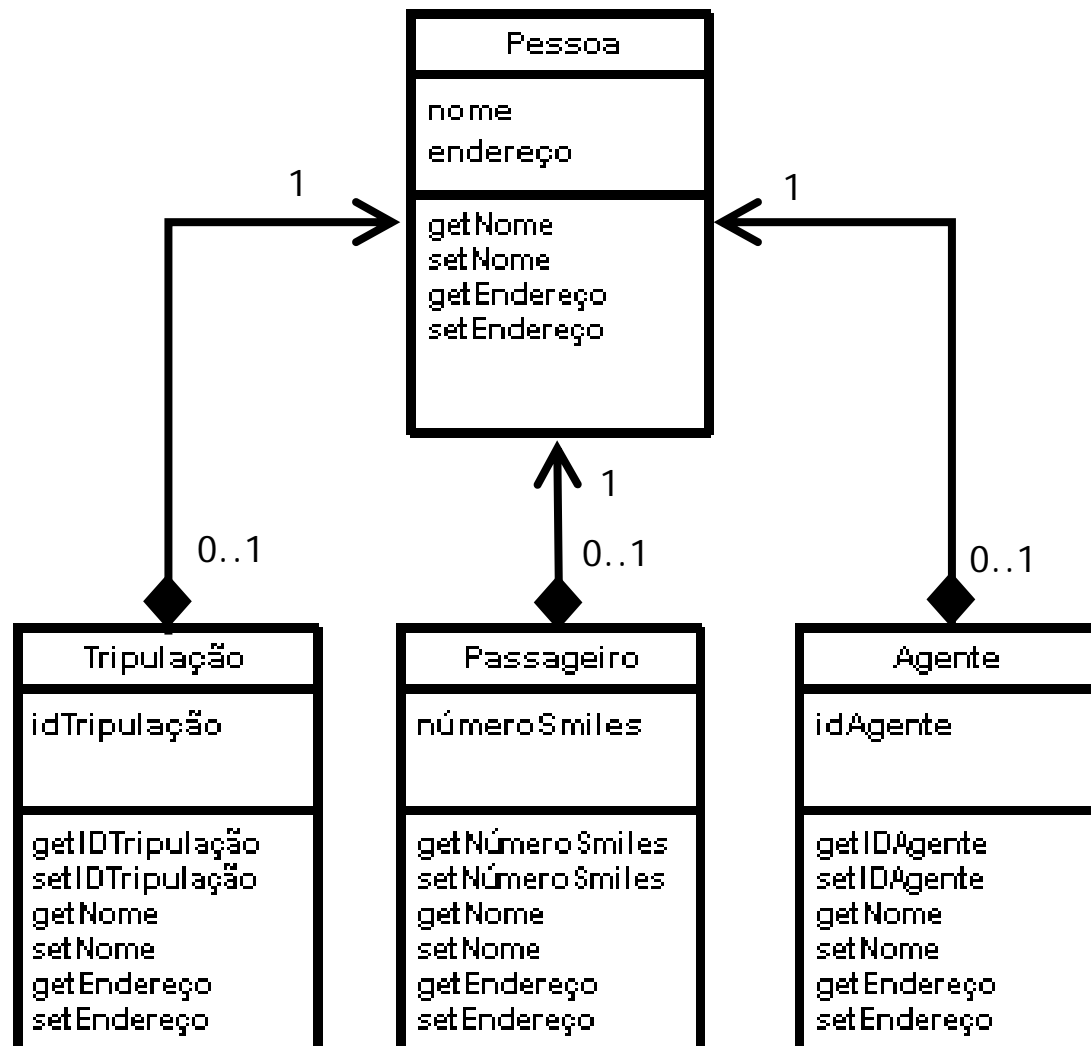
- O encapsulamento entre classes e subclasses é fraco (o acoplamento é forte)
- Mudar uma superclasse pode afetar todas as subclasses
  - Vantagem e desvantagem
- Isso viola um dos princípios básicos de projeto OO
  - Manter fraco acoplamento
- Além disso
  - Às vezes um objeto precisa ser de uma classe diferente em momentos diferentes. Contudo, com herança, a estrutura está fixa no código e não pode sofrer alterações em tempo de execução
  - A herança é um **relacionamento estático** que não muda com tempo

# Herança



Problema: uma pessoa não pode mudar de papel ou assumir combinações de papéis

# Usando Composição



# Usando Composição

- Analisando o Diagrama

- Estendemos a funcionalidade de Pessoa de várias formas, mas sem usar herança. Contudo, observe que a composição pode ser invertida (uma pessoa tem um ou mais papeis)
- No exemplo, está sendo utilizado o conceito de delegação: dois objetos estão envolvidos de forma a atender um pedido. Um exemplo de delegação seria um objeto de Tripulação **delegar uma mudança de nome** para Pessoa.

## Exemplo Delegação

```
3 public class Pessoa {  
4  
5     private String nome;  
6  
7     public Pessoa(String nome) {  
8         this.nome = nome;  
9     }  
10  
11     public String getNome() {  
12         return nome;  
13     }  
14  
15     public void setNome(String nome) {  
16         this.nome = nome;  
17     }  
18 }
```

# Exemplo Delegação

```
3 public class Tripulacao {
4
5     private int idTripulacao;
6     private Pessoa pessoa;
7
8     public Tripulacao(int id, String nome){
9         this.idTripulacao = id;
10        pessoa = new Pessoa(nome);
11    }
12
13    public void alteraNome(String nome){
14        pessoa.setNome(nome);
15    }
16
17    public String obterNome(){
18        return pessoa.getNome();
19    }
19 }
```

Percebe-se que se delega a mudança de nome para o objeto pessoa que tripulação tem por composição

# Exemplo Delegação

```
3 public class Principal {  
4  
5     public static void main(String args[]){  
6  
7         Tripulacao t1 = new Tripulacao(10, "TesteNome");  
8         System.out.println(t1.obterNome());  
9  
10        t1.alteraNome("NovoNome");  
11        System.out.println(t1.obterNome());  
12    }  
13 }
```

Ao se chamar o método "alterarNome(String nome) do objeto "t1", tem-se a impressão de que este método, de fato, contém a implementação para alterar o nome. Em outras palavras, tem-se que a impressão que "t1" é o responsável por alterar o nome

OBS: Contudo, sabemos que ele repassa essa responsabilidade para outro objeto.

# Exemplo de Delegação

- Analisando

- No exemplo: delegar a mudança de nome para Pessoa é semelhante a uma subclasse delegar uma operação para a superclasse (herdando a operação). Logo, **delegação sempre pode ser usada para substituir a herança**
- Em vez de tripulação ser uma pessoa, ele tem uma pessoa



# Código

```
3 public class Pessoa {  
4  
5     private String nome;  
6  
7     public Pessoa(String nome){  
8         this.nome = nome;  
9     }  
10  
11     public String getNome() {  
12         return nome;  
13     }  
14  
15     public void setNome(String nome) {  
16         this.nome = nome;  
17     }  
18 }
```

# Código

```
3 public class Tripulacao {
4
5     private int idTripulacao;
6     private Pessoa pessoa;
7
8     public Tripulacao(int id, Pessoa pessoa){
9         this.idTripulacao = id;
10        this.pessoa = pessoa;
11    }
```

```
3 public class Passageiro {
4
5     private int idPassageiro;
6     private Pessoa pessoa;
7
8     public Passageiro(int idPassageiro, Pessoa pessoa){
9         this.idPassageiro = idPassageiro;
10        this.pessoa = pessoa;
11    }
```

# Código

```
3 public class Principal {  
4  
5     public static void main(String args[]){  
6  
7         Pessoa pessoa1 = new Pessoa("Fulano");  
8         Tripulacao t1 = new Tripulacao(10,pessoa1);  
9         Passageiro p1 = new Passageiro(10,pessoa1);  
10    }  
11 }  
12
```

# Comparação

```
3 public class Principal {  
4  
5     public static void main(String args[]) {  
6  
7         Tripulacao t1 = new Tripulacao(10, "fulano");  
8  
9         System.out.println("Nome Tripulante: " + t1.getNome());  
10        System.out.println("Identificação Tripulante: " + t1.getIdTripulacao());  
11    }  
12 }
```

Problems @ Javadoc Declaration Console Progress

<terminated> Principal (7) [Java Application] C:\Program Files\Java\jdk1.6.0\_21\jre\bin\javaw.exe (21/06/2011 10:46:17)

Nome Tripulante: fulano

Identificação Tripulante: 10

# Comparação

```
3 public class Principal {  
4  
5     public static void main(String args[]) {  
6  
7         Pessoa pessoa1 = new Pessoa("Fulano");  
8         Tripulacao t1 = new Tripulacao(10,pessoa1);  
9         Passageiro p1 = new Passageiro(10,pessoa1);  
10  
11         System.out.println("Nome Tripulante: " + t1.obterNome());  
12         System.out.println("Nome Passageiro "+ p1.obterNome());  
13     }  
14 }
```

Problems Javadoc Declaration Console Progress

<terminated> Principal (4) [Java Application] C:\Program Files\Java\jdk1.6.0\_21\jre\bin\javaw.exe (21/06/2011 10:47:22)

Nome Tripulante: Fulano  
Nome Passageiro Fulano

# Resumo

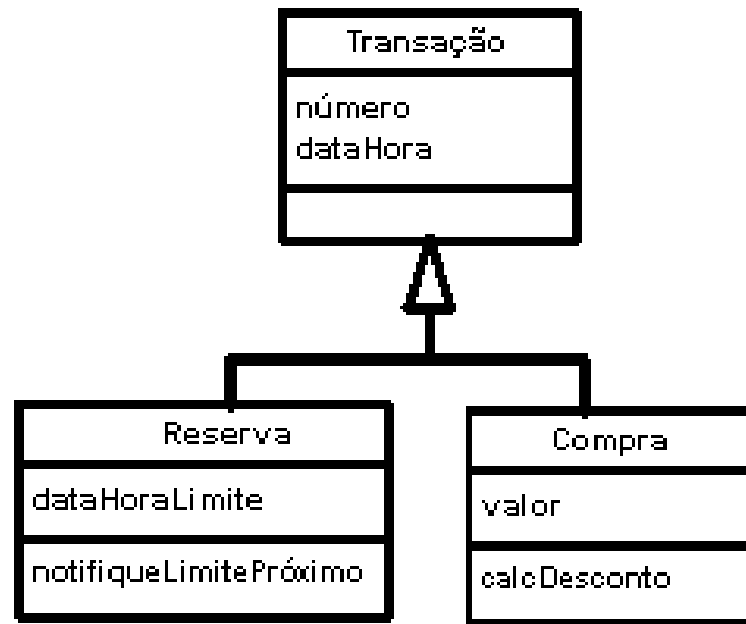
- Composição

- De forma geral, a composição é melhor do que herança normalmente, pois:
  - Permite mudar a associação entre classes em tempo de execução;
  - Permite que um objeto assuma mais de um comportamento (ex. papel);

# Regras para Herança

- Algumas regras para verificar uso de Herança
  - O objeto "é um tipo especial de" e não "um papel assumido por"
  - O objeto nunca tem que mudar para outra classe
  - Subclasse estende a super mas não faz override/anula de variáveis/métodos

# Exemplo

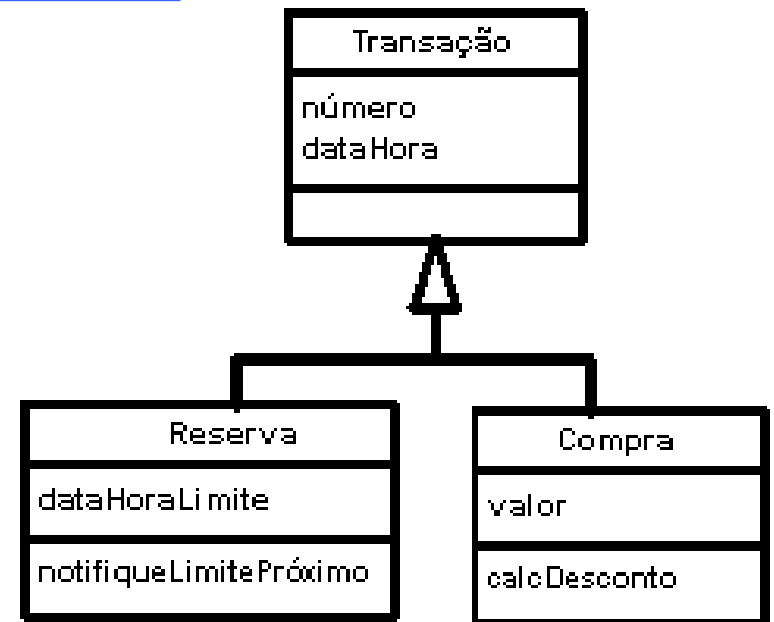


Tipo especial: ok. Uma Reserva é um tipo especial de Transação e não um papel assumido por uma Transação. O mesmo vale para Compra

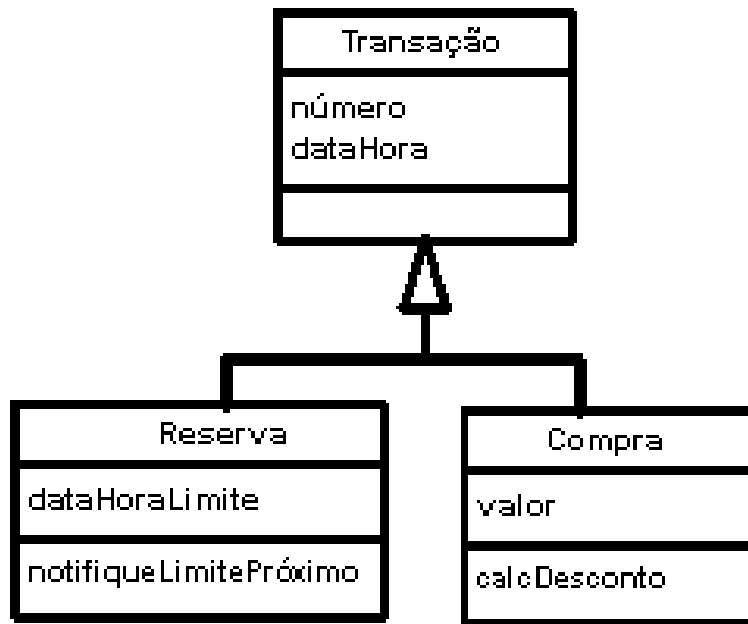


# Exemplo

Mutação: ok. Uma reserva sempre será uma Reserva, e nunca se transforma em Compra (se houver uma compra da passagem, será outra transação). Idem para Compra: sempre será uma Compra



# Exemplo



Só estende: ok. Ambas as subclasses estendem Transação com novas variáveis e métodos e não fazem override ou anulam coisas de Transação

# Composição e Herança

---

## Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira  
oliveira.edmar@ufjf.edu.br

Universidade Federal de Juiz de Fora - UFJF  
Departamento de Ciência da Computação - DCC