

# Delegação

---

## Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira  
[edmar.oliveira@ufjf.edu.br](mailto:edmar.oliveira@ufjf.edu.br)

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

# Delegação

## ■ Definição

- É outra forma de se pensar/realizar reuso
- Delegação: sempre que um objeto não puder realizar uma operação por si próprio, ele **delega** uma parte dela para outro(s) objeto(s). Ou seja, **um objeto reusa as operações** dos objetos para os quais ele delega responsabilidades.
- OBS: delegar é repassar a mensagem (do objeto delegador para o delegado)
- A delegação é mais genérica que a herança
  - Um objeto pode reutilizar o comportamento de outro sem que o primeiro precise ser uma subclasse do segundo. Em outras palavras, um objeto A pode reutilizar o comportamento de um objeto B, sem que A precise ser subclasse de B

# Delegação

- Definição

- Se o objeto A não implementa uma determinada mensagem ele delega (repassa) a mensagem para o objeto B. Se o objeto B implementa aquela mensagem então ele a executa com os dados de A, senão ele a delega para seus outros objetos

# Exemplo

```
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();
    }
}
```

# Delegação - Exemplo

| Lista  |
|--|
|  |
| addInicio(Item)<br>addFinal(Item)<br>removeInicio()<br>removeFinal() |

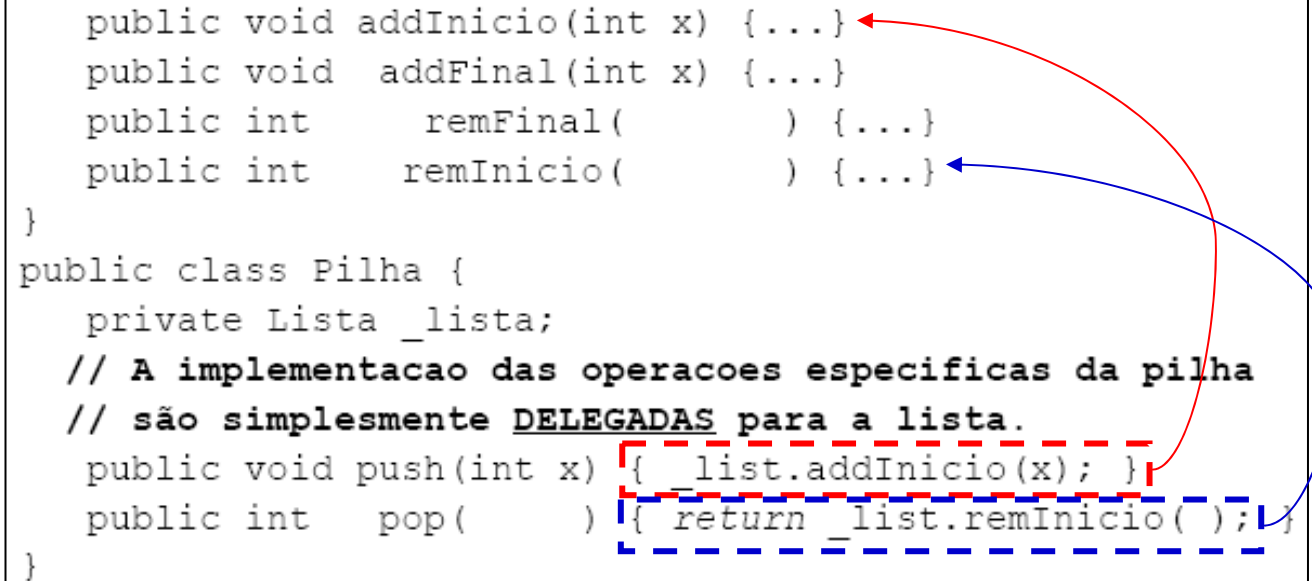


| Pilha  |
|--|
|  |
| empilhar() → addInicio(Item)<br>Desempilhar() → removeInicio() |

# Delegação - Exemplo

```
public class Lista {
    private Vector _dados = new Vector();
    .....
    public void addInicio(int x) {...}
    public void addFinal(int x) {...}
    public int remFinal(      ) {...}
    public int remInicio(     ) {...}
}

public class Pilha {
    private Lista _lista;
    // A implementacao das operacoes especificas da pilha
    // são simplesmente DELEGADAS para a lista.
    public void push(int x) { _list.addInicio(x); }
    public int pop(      ) { return _list.remInicio( ); }
}
```



```
public void push(int x) { _list.addInicio(x); }
```

O método push() - ou empilhar() - simplesmente chama o método addInicio() da classe Lista. Ou seja, ele delega para Lista a tarefa de adicionar um item (x) no início da pilha (lembre-se que toda pilha é uma lista)

# Delegação - Quando Usar

## ■ Cenário

- Quando você tem um código a ser reutilizado entre duas ou mais classes e não existe herança entre elas

```
public class ClienteJuridico extends Cliente {  
    private Endereco _endereco;  
    public void setEndereco(numero, rua, cidade, estado, cep) {  
        _endereco.setData(numero, rua, cidade, estado, cep) ;}  
}
```

```
public class Fornecedor {  
    private Endereco _end;  
    public void setEndereco(numero, rua, cidade, estado, cep) {  
        _end.setData(numero, rua, cidade, estado, cep) ;}  
}
```

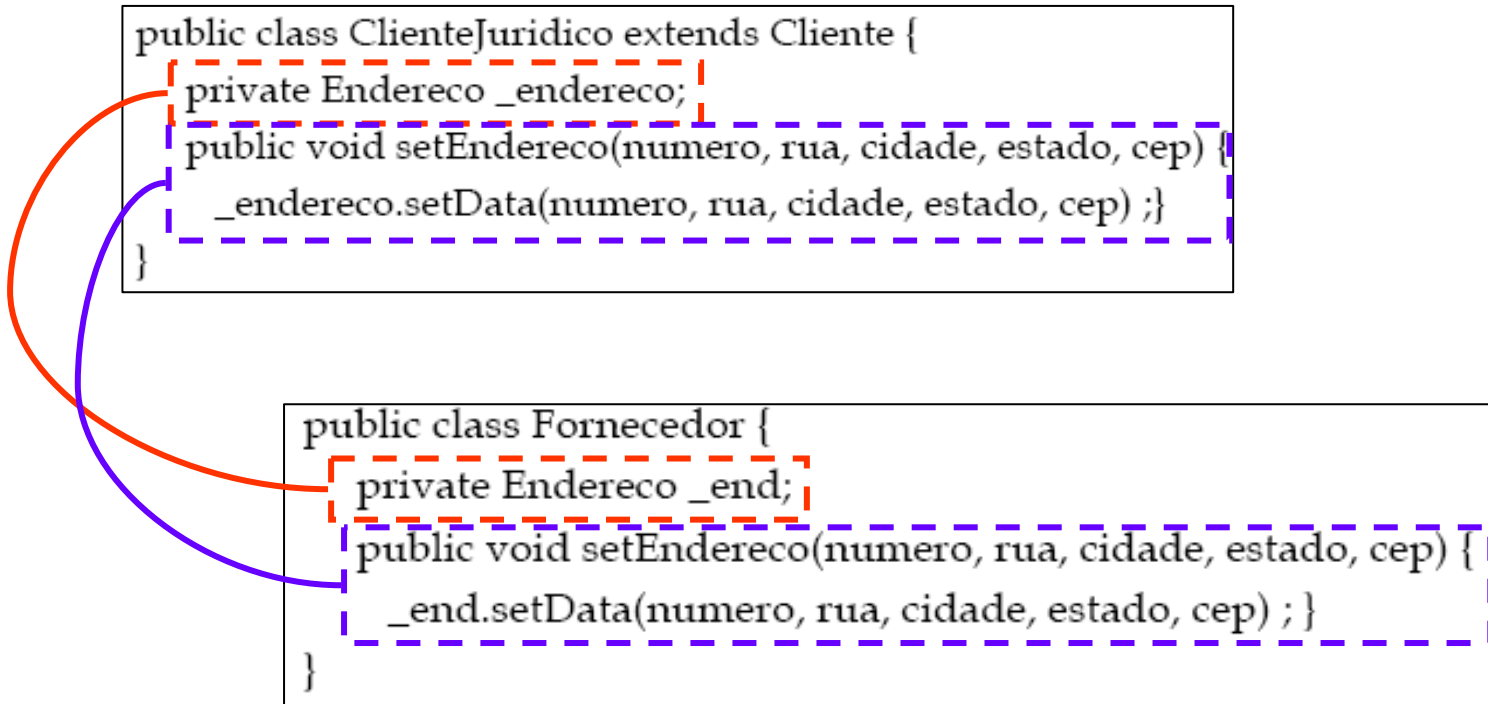
# Delegação - Quando Usar

## ■ Cenário

- Quando você tem um código a ser reutilizado entre duas ou mais classes e não existe herança entre elas

```
public class ClienteJuridico extends Cliente {  
    private Endereco _endereco;  
    public void setEndereco(numero, rua, cidade, estado, cep) {  
        _endereco.setData(numero, rua, cidade, estado, cep) ;}  
}
```

```
public class Fornecedor {  
    private Endereco _end;  
    public void setEndereco(numero, rua, cidade, estado, cep) {  
        _end.setData(numero, rua, cidade, estado, cep) ;}  
}
```





# Delegação - Quando Usar

## ■ Cenário

- Quando você tem um código a ser reutilizado entre duas ou mais classes e não existe herança entre elas

```
public class ClienteJuridico extends Cliente {  
    private Endereco _endereco;  
    public void setEndereco(numero, rua, cidade, estado, cep) {  
        _endereco.setData(numero, rua, cidade, estado, cep) ;}  
}
```

```
public class Fornecedor {  
    private Endereco _end;  
    public void setEndereco(numero, rua, cidade, estado, cep) {  
        _end.setData(numero, rua, cidade, estado, cep) ;}  
}
```

```
public class Endereco {  
    private numero, rua, cidade, estado, cep;  
}
```

Esta classe possui o método setData

# Delegação e Herança

## ■ Pensando em Reuso

- No reuso por generalização/especialização, subclasses herdam comportamento da superclasse. Exemplo: um objeto ContaCorrente não tem como atender à mensagem para executar a operação debitar só com os recursos de sua classe. Ele, então, utiliza a operação herdada da superclasse.
- Vantagem
  - Fácil de implementar.
- Desvantagem:
  - Exposição dos detalhes da superclasse às subclasses (Violação do princípio do encapsulamento).

# Delegação e Herança

- OBS

- A herança permite o compartilhamento de comportamento baseado em classes, enquanto que a delegação permite o compartilhamento baseado em objetos;
- Além disso
  - Qualquer extensão de uma classe que pode ser feita utilizando-se a herança, também pode ser feita com a delegação. Entretanto, a recíproca não é verdadeira;

# Delegação e Herança

- Vantagem de Delegação sobre Herança
  - Uma outra vantagem da delegação sobre a herança de classes é que o compartilhamento de comportamento e o reuso podem ser realizados em tempo de execução. Um exemplo, considerando o caso de Printer e RealPrinter, seria a classe Printer conter um método onde se criaria um objeto de RealPrinter e utilizaria o objeto criado para chamar um método desta classe que seria chamado se, e somente se, fosse necessário.
  - Na herança de classes, o reuso é especificado estaticamente e não pode ser modificado.

# Delegação e Herança

- Desvantagem de Delegação sobre Herança
  - Não pode ser utilizada quando uma classe parcialmente abstrata está envolvida. A explicação fica para quando estudarmos Classes Abstratas.

# Generalização e Delegação

## ■ Generalização vs Delegação

- Há vantagens e desvantagens tanto na generalização quanto na delegação.
- De forma geral, não é recomendado utilizar generalização nas seguintes situações:
  - Para representar papéis de uma superclasse.
  - Quando a subclasse herda propriedades que não se aplicam a ela.
  - Quando um objeto de uma subclasse pode se transformar em um objeto de outra subclasse. Por exemplo, um objeto Cliente se transforma em um objeto Funcionário.

# Delegação

---

## Orientação a Objetos - DCC025

Prof. Edmar Welington Oliveira  
[edmar.oliveira@ufjf.edu.br](mailto:edmar.oliveira@ufjf.edu.br)

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