

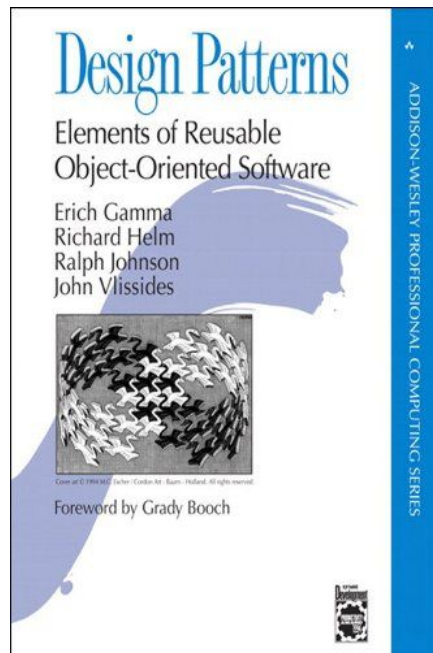
Padrões de Projeto

Profa. Fabíola S. F. Pereira

fabiola.pereira@ufu.br

Padrões de Projeto

- Soluções recorrentes para problemas de projeto enfrentados por engenheiros de software



1994, conhecido como livro da
"Gangue dos Quatro" ou GoF

Criacionais	Estruturais	Comportamentais
Abstract Factory Factory Method Singleton Builder Prototype	Proxy Adapter Facade Decorator Bridge Composite Flyweight	Strategy Observer Template Method Visitor Chain of Responsibility Command Interpreter Iterator Mediator Memento State

23 padrões

(6) Decorador

Contexto: Sistema que usa canais de comunicação

- Já foi usado para explicar o padrão Fábrica

```
interface Channel {  
    void send(String msg);  
    String receive();  
}  
  
class TCPChannel implements Channel {  
    ...  
}  
  
class UDPChannel implements Channel {  
    ...  
}
```

Problema: Precisamos adicionar funcionalidades extras em canais

- Ou seja, os canais default (TCP, UDP) não são suficientes
- Precisamos também de canais com:
 - Compactação e descompactação de dados
 - Buffers (ou caches) de dados
 - Logging dos dados enviados e recebidos
 - etc

(Primeira Solução)

- Uma primeira solução consiste no uso de herança para criar subclasses com cada possível seleção de funcionalidades

(Primeira Solução)

`TCPZipChannel extends TCPChannel`

`TCPBufferedChannel extends TCPChannel`

`TCPBufferedZipChannel extends TCPZipChannel extends TCPChannel`

`TCPLogChannel extends TCPChannel`

`TCPLogBufferedZipChannel extends TCPBufferedZipChannel extends`

`TCPZipChannel extends TCPChannel`

`UDPZipChannel extends UDPChannel`

`UDPBufferedChannel extends UDPChannel`

`UDPBufferedZipChannel extends UDPZipChannel extends UDPChannel`

`UDPLogChannel extends UDPChannel`

`UDPLogBufferedZipChannel extends UDPBufferedZipChannel extends`

`UDPZipChannel extends UDPChannel`

(Primeira Solução)

- Uma primeira solução consiste no uso de herança para criar subclasses com cada possível seleção de funcionalidades
- Uma solução via herança é quase que inviável, pois ela gera uma explosão combinatória do número de classes relacionadas com canais de comunicação.

Solução: **Padrão Decorador**

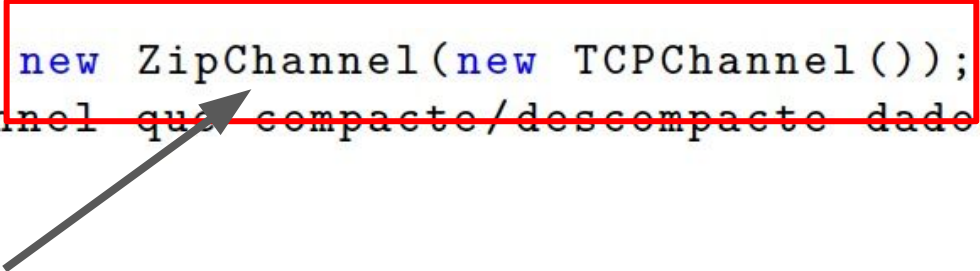
- Resolve esse problema -- adicionar funcionalidades em uma classe -- por meio de **composição** e sem gerar um número excessivo de classes

Solução: **Padrão Decorador**

- Resolve esse problema -- adicionar funcionalidades em uma classe -- por meio de composição e sem gerar um número excessivo de classes
- Decorador é um exemplo de aplicação do princípio de projeto **Prefira Composição a Herança**

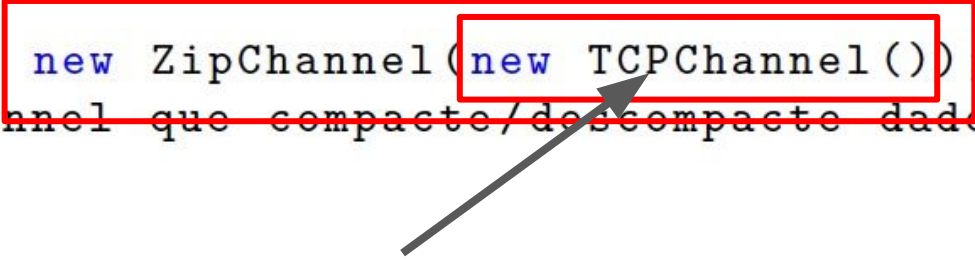
Exemplo

```
channel = new ZipChannel(new TCPChannel());  
// TCPChannel que compacta/descompacta dados enviados/recebidos
```



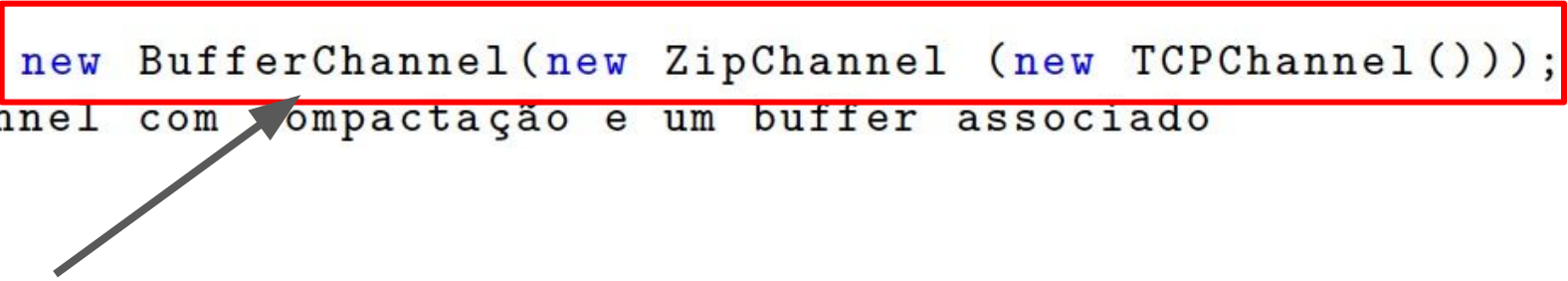
Exemplo

```
channel = new ZipChannel(new TCPChannel());  
// TCPChannel que compacta/descompacta dados enviados/recebidos
```



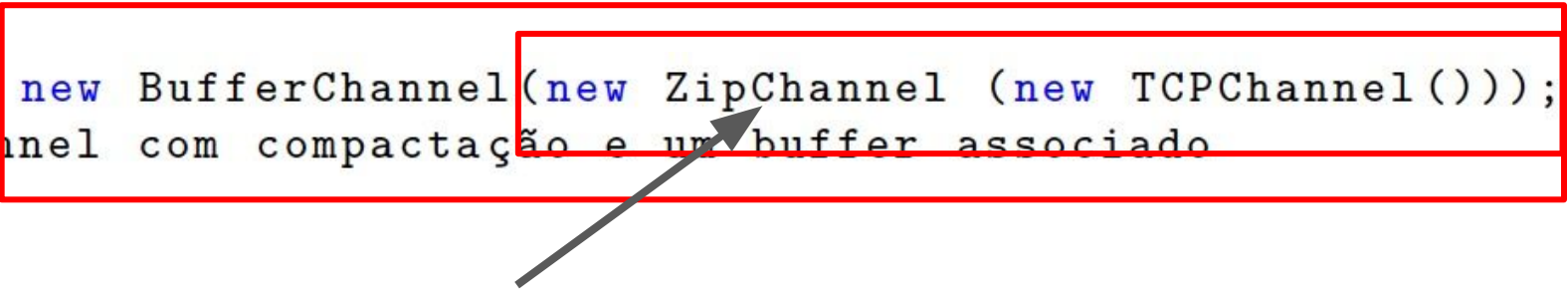
Mais um exemplo

```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```



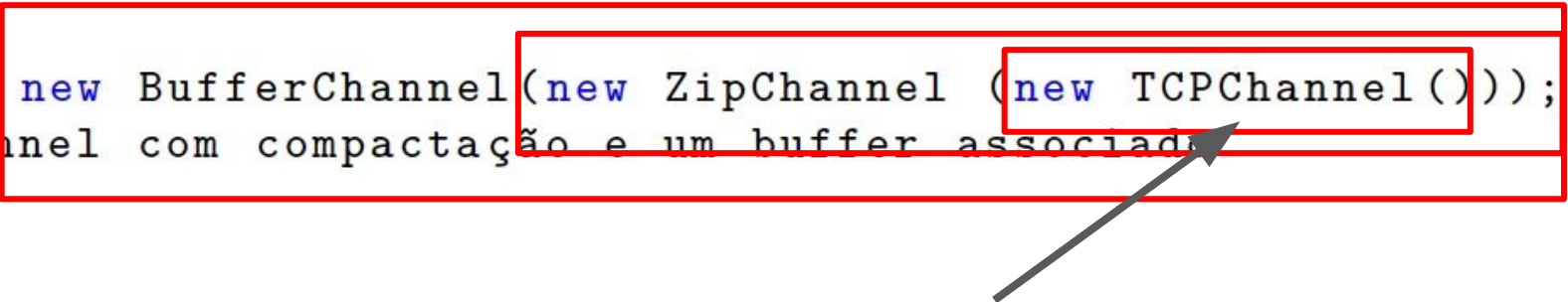
Mais um exemplo

```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```



Mais um exemplo

```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```



The diagram consists of three nested red rectangular boxes. The outermost box encloses the entire code snippet. The middle box encloses the expression `new ZipChannel (new TCPChannel())`. The innermost box encloses the expression `new TCPChannel()`. A grey arrow points from the bottom right towards the `new TCPChannel()` expression, which is highlighted by the innermost box.

Comparação

```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```



Dentro de uma caixa, tem outra caixa, que tem outra caixa ... até chegar no presente. Isto é, até chegar em TCPChannel ou UDPChannel

Implementação do Padrão Decorador

ChannelDecorator

```
class ChannelDecorator implements Channel {  
  
    protected Channel channel;  
  
    public ChannelDecorator(Channel channel) {  
        this.channel = channel;  
    }  
  
    public void send(String msg) {  
        channel.send(msg);  
    }  
  
    public String receive() {  
        return channel.receive();  
    }  
}
```

Todos os decoradores são subclasses dessa classe única

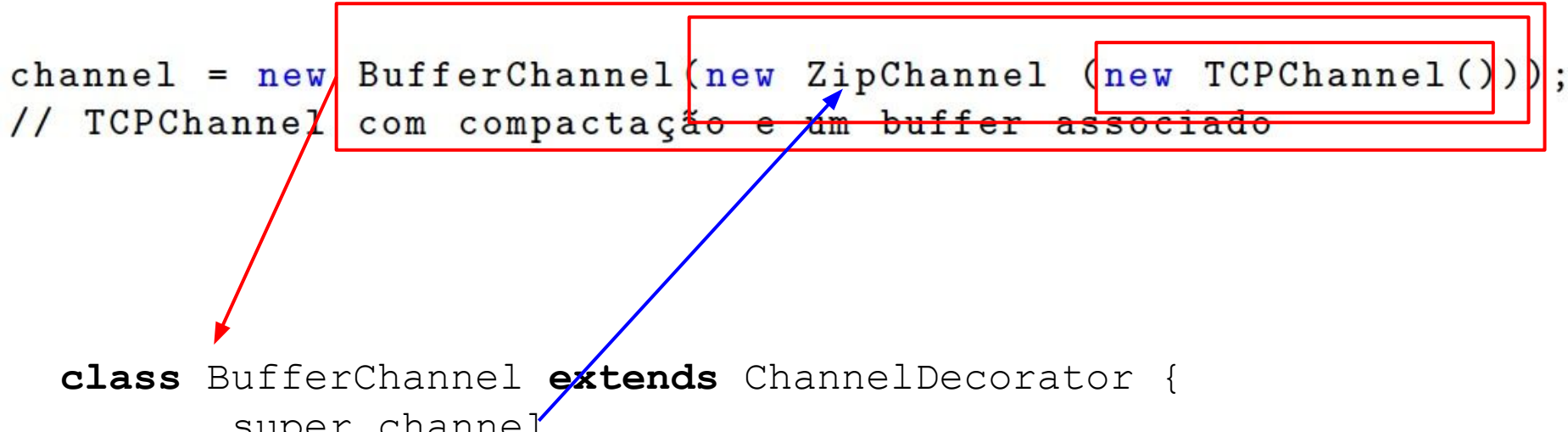
Implementação: ZipChannel

ZipChannel

```
class ZipChannel extends ChannelDecorator {  
  
    public ZipChannel(Channel c) {  
        super(c);  
    }  
  
    public void send(String msg) {  
        "compacta mensagem msg"  
        super.channel.send(msg);  
    }  
  
    public String receive() {  
        String msg = super.channel.receive();  
        "descompacta mensagem msg"  
        return msg;  
    }  
  
}
```

Exemplo

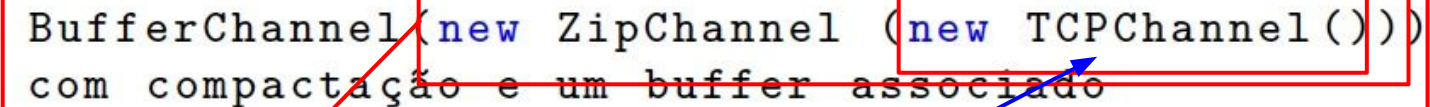
```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```



```
class BufferChannel extends ChannelDecorator {  
    ... super.channel  
}
```

Exemplo

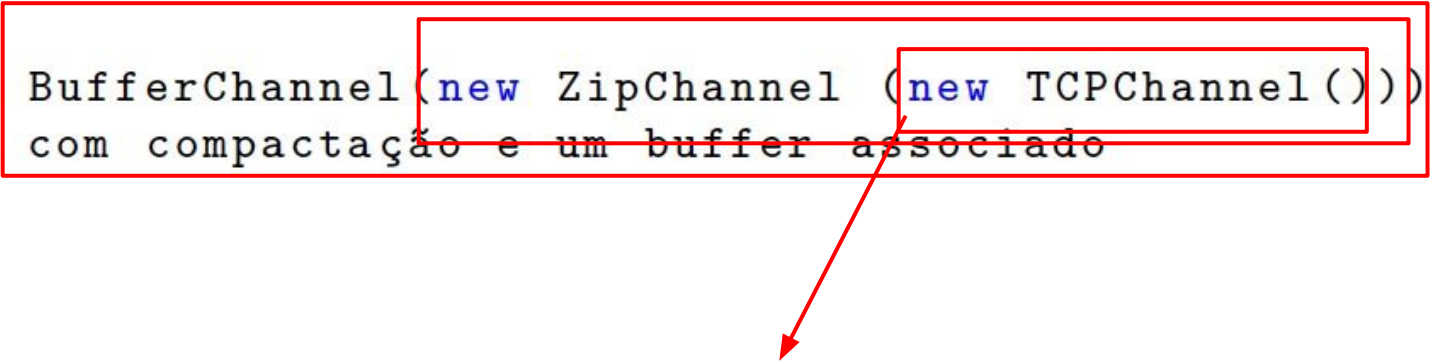
```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```



```
class ZipChannel extends ChannelDecorator {  
    ... super.channel  
}
```

Exemplo

```
channel = new BufferChannel(new ZipChannel (new TCPChannel()));  
// TCPChannel com compactação e um buffer associado
```

The code is annotated with three nested red rectangular boxes. The innermost box encloses 'new TCPChannel()', the middle box encloses 'new ZipChannel (new TCPChannel())', and the outermost box encloses the entire line 'channel = new BufferChannel(new ZipChannel (new TCPChannel()));'. A red arrow points from the 'new TCPChannel()' box down to the 'TCPChannel' class definition below.

```
class TCPChannel implements Channel {  
    // canal "final"  
}
```


(7) Strategy

Contexto: Biblioteca de Estruturas de Dados

```
class MyList {  
  
    ... // dados de uma lista  
    ... // métodos de uma lista: add, delete, search  
  
    public void sort() {  
        ... // ordena a lista usando Quicksort  
    }  
  
}
```

Problema

- Classe MyList **não** está aberta a extensões
- Possível extensão: mudar o algoritmo de ordenação
- Usar ShellSort, HeapSort, etc

Solução: **Padrão Strategy**

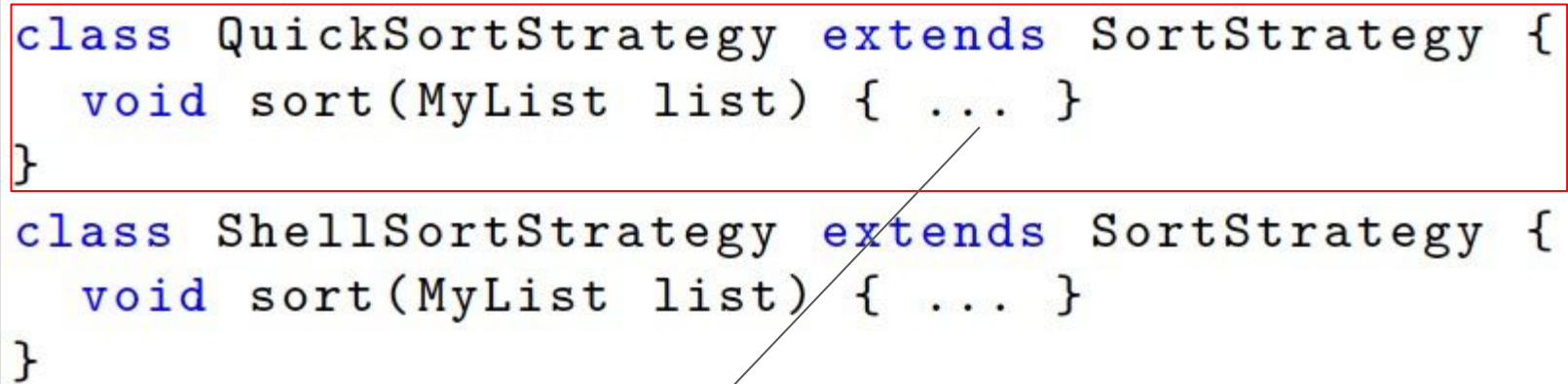
- Objetivo:
 - Parametrizar os algoritmos usados por uma classe
 - Tornar uma classe "aberta" a novos algoritmos
- No exemplo: novos algoritmos de ordenação

Passo #1: Criar uma hierarquia de "estratégias"
(estratégia = algoritmo)

```
abstract class SortStrategy {  
    abstract void sort(MyList list);  
}  
class QuickSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}  
class ShellSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}
```

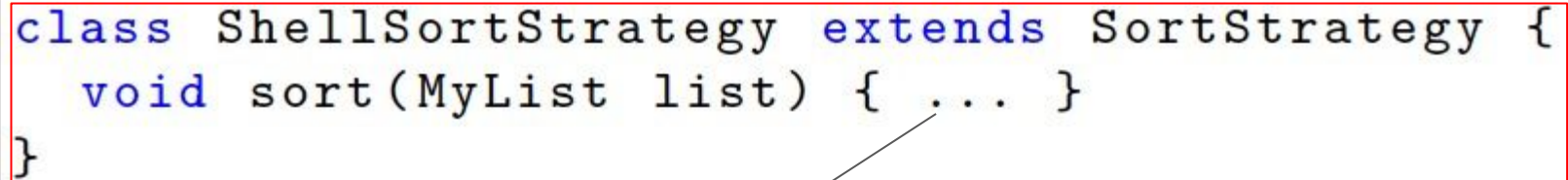
```
abstract class SortStrategy {  
    abstract void sort(MyList list);  
}  
  
class QuickSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}  
  
class ShellSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}
```

```
abstract class SortStrategy {  
    abstract void sort(MyList list);  
}  
class QuickSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}  
class ShellSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}
```



Código do QuickSort


```
abstract class SortStrategy {  
    abstract void sort(MyList list);  
}  
class QuickSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}  
class ShellSortStrategy extends SortStrategy {  
    void sort(MyList list) { ... }  
}
```



Código do ShellSort

Passo #2: Modificar MyList para usar a hierarquia de estratégias

```
class MyList {  
  
    ... // dados de uma lista  
    ... // métodos de uma lista: add, delete, search  
  
    private SortStrategy strategy;  
  
    public MyList() {  
        strategy = new QuickSortStrategy();  
    }  
  
    public void setSortStrategy(SortStrategy strategy) {  
        this.strategy = strategy;  
    }  
  
    public void sort() {  
        strategy.sort(this);  
    }  
  
}
```

```
class MyList {  
  
    ... // dados de uma lista  
    ... // métodos de uma lista: add, delete, search  
  
    private SortStrategy strategy;  
  
    public MyList() {  
        strategy = new QuickSortStrategy();  
    }  
  
    public void setSortStrategy(SortStrategy strategy) {  
        this.strategy = strategy;  
    }  
  
    public void sort() {  
        strategy.sort(this);  
    }  
  
}
```

```
class MyList {  
  
    ... // dados de uma lista  
    ... // métodos de uma lista: add, delete, search  
  
    private SortStrategy strategy;  
  
    public MyList() {  
        strategy = new QuickSortStrategy();  
    }  
  
    public void setSortStrategy(SortStrategy strategy) {  
        this.strategy = strategy;  
    }  
  
    public void sort() {  
        strategy.sort(this);  
    }  
  
}
```

```
class MyList {  
  
    ... // dados de uma lista  
    ... // métodos de uma lista: add, delete, search  
  
    private SortStrategy strategy;  
  
    public MyList() {  
        strategy = new QuickSortStrategy();  
    }  
  
    public void setSortStrategy(SortStrategy strategy) {  
        this.strategy = strategy;  
    }  
  
    public void sort() {  
        strategy.sort(this);  
    }  
}
```

Exercícios de Fixação

Assinale V ou F

() Decorador é um padrão que permite converter a interface de uma classe para outra interface esperada pelos seus clientes. Ele viabiliza então que classes trabalhem juntas, o que não seria possível devido à incompatibilidade de suas interfaces.

() Strategy é um padrão que permite parametrizar os algoritmos usados por uma classe.

Créditos

- CC-BY: Slides adaptados de Marco Tulio Valente, ESM

Fim

Assinale V ou F

(F) Decorador é um padrão que permite converter a interface de uma classe para outra interface esperada pelos seus clientes. Ele viabiliza então que classes trabalhem juntas, o que não seria possível devido à incompatibilidade de suas interfaces.

(V) Strategy é um padrão que permite parametrizar os algoritmos usados por uma classe.