



Análise e Projeto de Sistemas II

Aula 9: Padrões de Projeto

Prof. Fernando Xavier

fernando.xavier@udf.edu.br



Análise e Projeto de Sistemas II

- Pendente da aula anterior

Análise e Projeto de Sistemas II

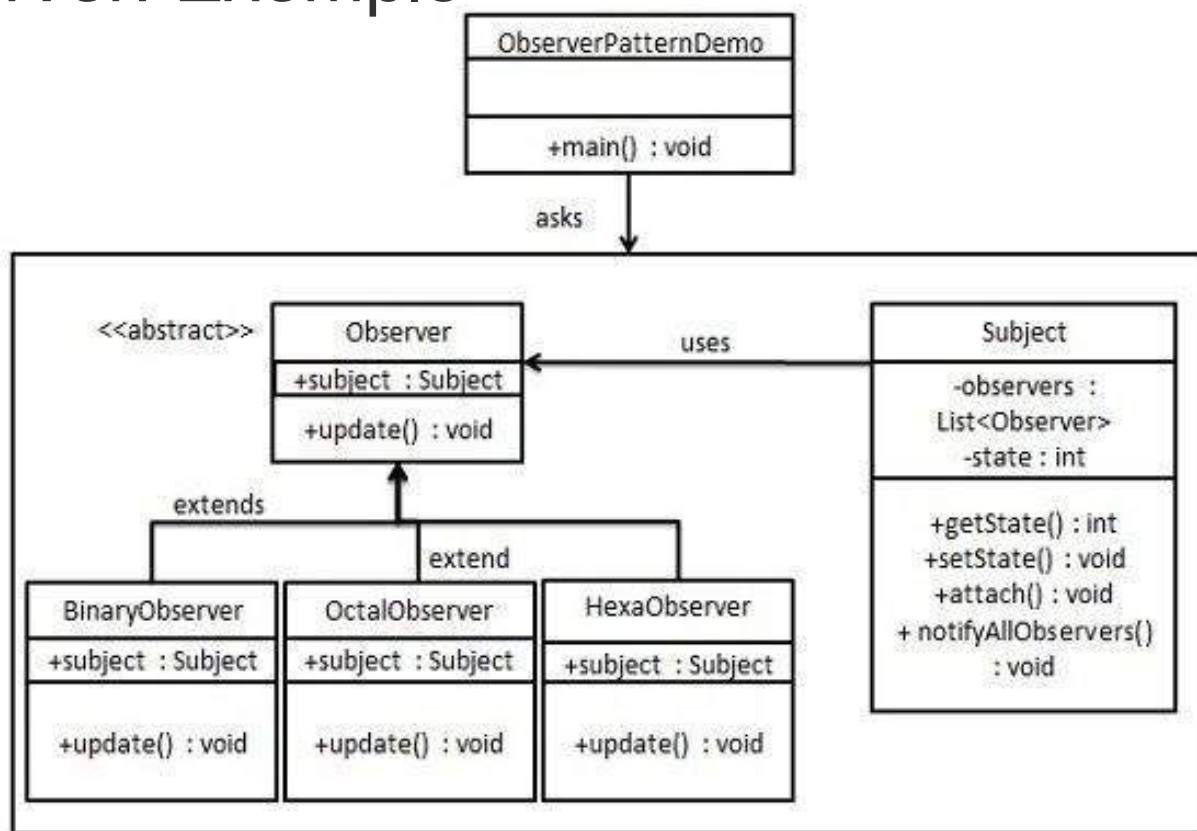
- Observer
 - Define uma dependência um-para-muitos entre objetos para que, quando um objeto muda de estado, todos os seus dependentes são notificados e atualizados automaticamente
 - Composto da classe observada (Subject) e das classes observadoras (Observers)

Análise e Projeto de Sistemas II

- Observer - Quando usar:
 - Quando uma mudança em um objeto requer a mudança de outros e você não sabe quantos objetos precisam ser alterados
 - Quando um objeto deve ser capaz de notificar outros objetos sem fazer suposições sobre quem são esses objetos. Em outras palavras, você não quer que estes objetos firmemente acoplados

Análise e Projeto de Sistemas II

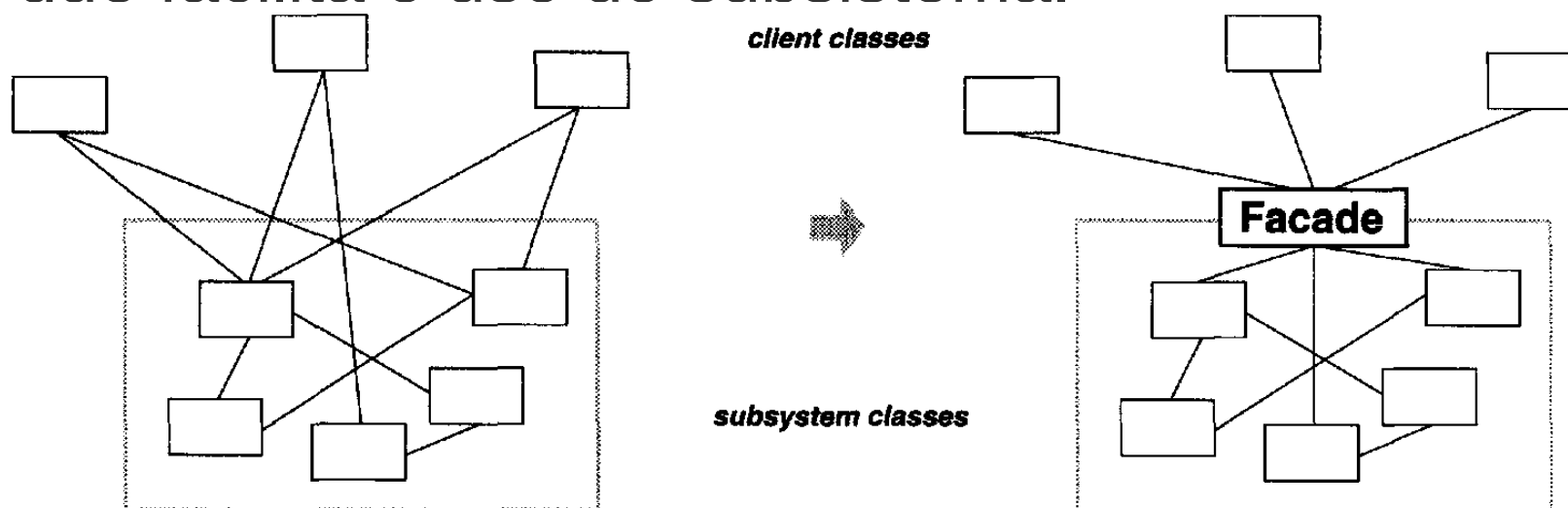
- Observer: Exemplo



Fonte: https://www.tutorialspoint.com/design_pattern/observer_pattern.htm

Análise e Projeto de Sistemas II

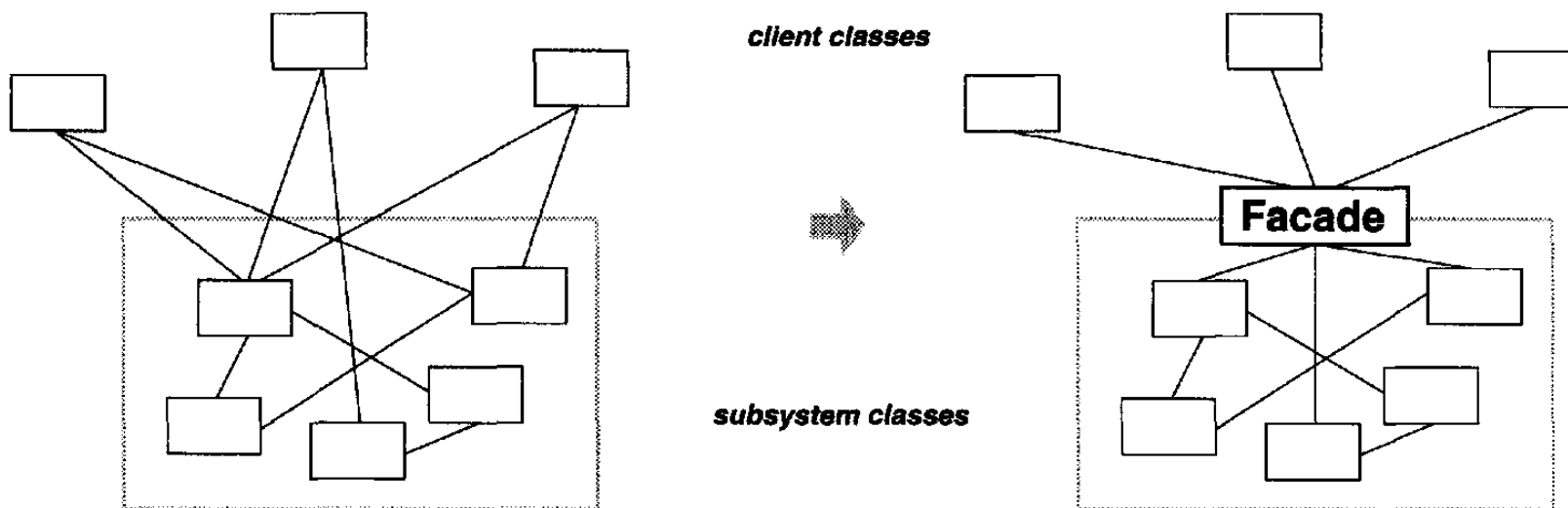
- Facade
 - Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. O Facade define uma interface de nível superior que facilita o uso do subsistema.



Fonte: Livro Design Patterns - GoF
Prof. Fernando Xavier

Análise e Projeto de Sistemas II

- Facade
 - Usando esse padrão, o que pode ser dito sobre acoplamento?



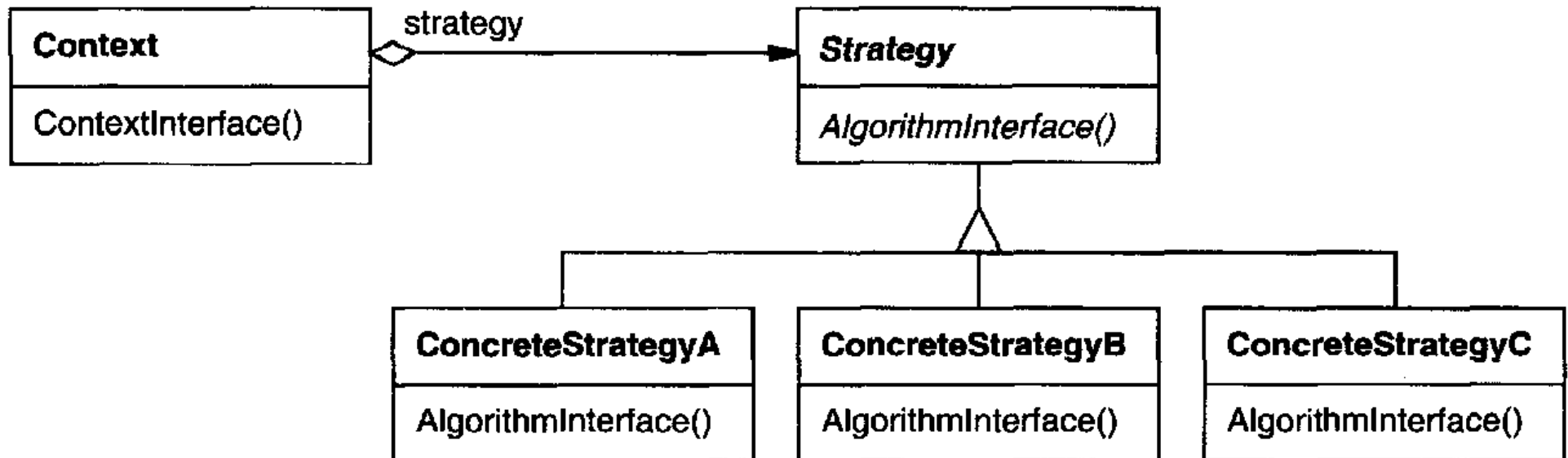
Fonte: Livro Design Patterns - GoF

Análise e Projeto de Sistemas II

- Strategy
 - Defina uma família de algoritmos, encapsule cada um deles e torne-os intercambiáveis. A estratégia permite que o algoritmo varie independentemente dos clientes que o use
 - Suponha que uma determinada funcionalidade tem variações. Tratar com diversos if-else? E quando precisar acrescentar outras?
 - Com Strategy: cada algoritmo é encapsulado em uma classe diferente

Análise e Projeto de Sistemas II

- Strategy



Análise e Projeto de Sistemas II

- Strategy

```
1 package gof.strategy;
2
3 public interface Financiamento {
4
5     double calcularParcela(int prazo, double valor);
6
7 }
8
```

Análise e Projeto de Sistemas II

- Strategy

```
1 package gof.strategy;
2
3 public class TabelaPrice implements Financiamento {
4
5     @Override
6     public double calcularParcela(int prazo, double valor) {
7
8         double valorParcela = (valor/prazo)*1.2;
9         return valorParcela;
10
11     }
12
13 }
```

Análise e Projeto de Sistemas II

- Strategy

```
1 package gof.strategy;
2
3 public class TabelaFipe implements Financiamento {
4
5     @Override
6     public double calcularParcela(int prazo, double valor) {
7
8         double valorParcela = (valor/prazo)*1.3;
9         return valorParcela;
10    }
11 }
12
13
```

Análise e Projeto de Sistemas II

- Strategy

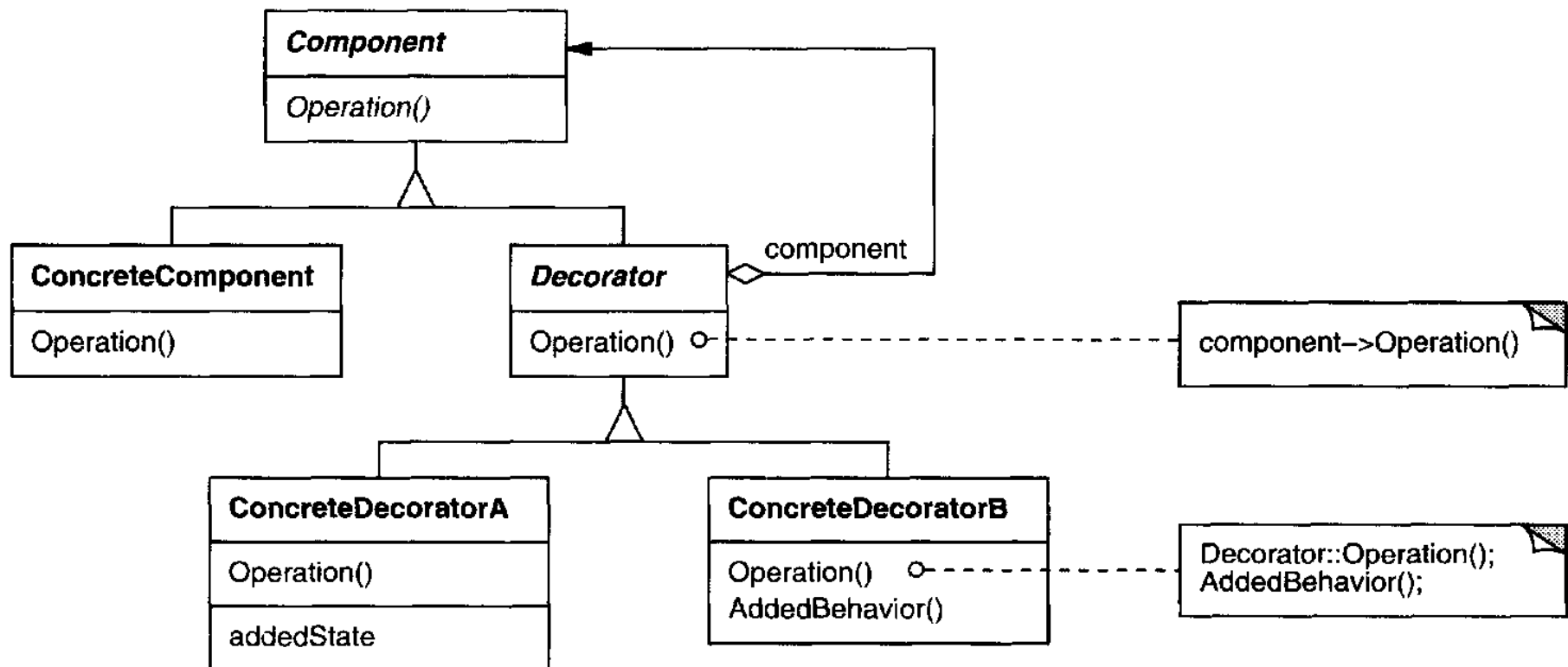
```
1 package gof.strategy;
2
3 public class Calculo {
4
5     Financiamento tabela;
6
7     public Calculo(Financiamento fin) {
8         this.tabela = fin;
9     }
10
11     public static void main(String[] args) {
12
13         Calculo calc = new Calculo(new TabelaPrice());
14         double parcela = calc.tabela.calcularParcela(12, 1200);
15         System.out.println(parcela);
16     }
17 }
18
```

Análise e Projeto de Sistemas II

- Decorator
 - Anexar responsabilidades adicionais a um objeto dinamicamente. Os decoradores fornecem uma alternativa flexível à subclasse para estender a funcionalidade.
 - Adicionar responsabilidades apenas a um objeto e não o conjunto de inteiro de objetos do mesmo tipo

Análise e Projeto de Sistemas II

- Decorator

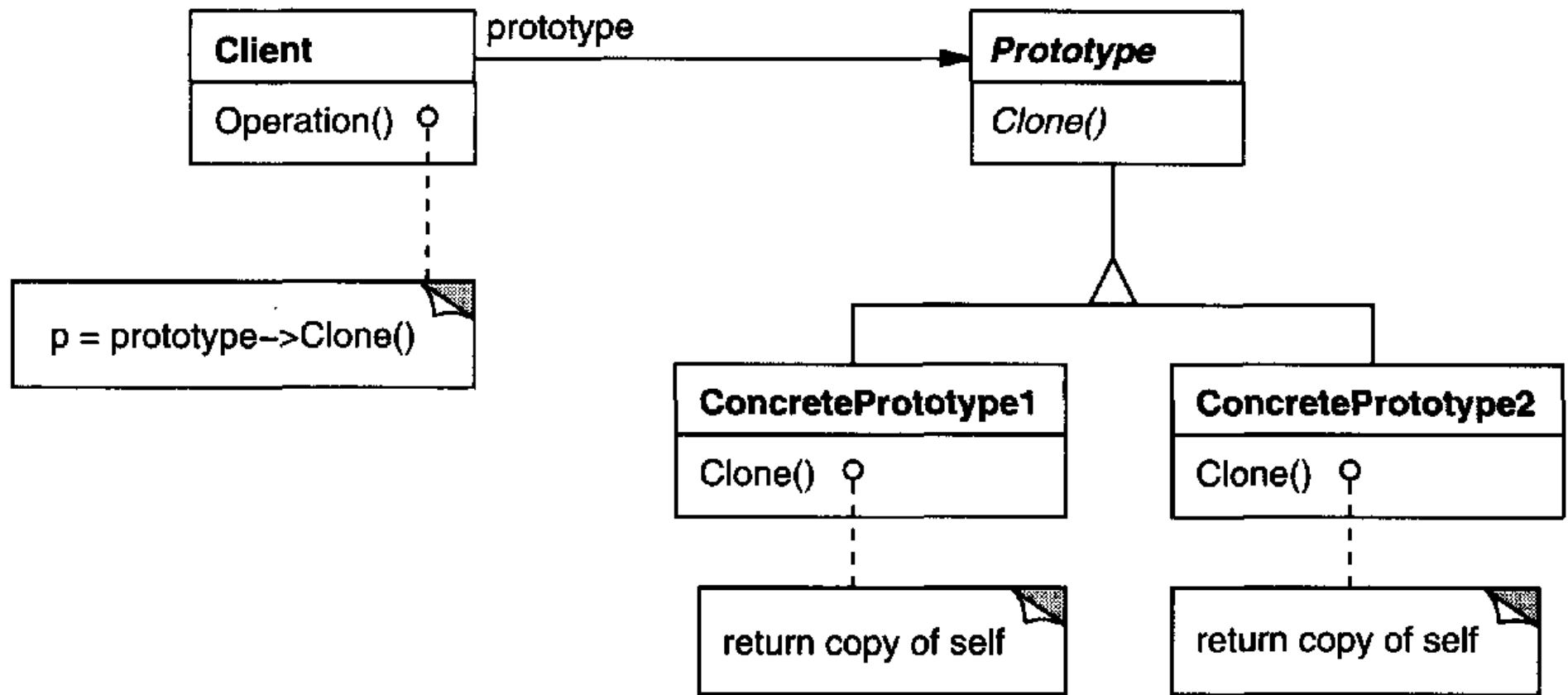


Análise e Projeto de Sistemas II

- Prototype
 - Especifique os tipos de objetos para criar usando uma instância prototípica e crie novos objetos copiando esse protótipo
 - Usado quando as classes que deverão ser instanciadas são definidas em tempo de execução

Análise e Projeto de Sistemas II

- Prototype



Análise e Projeto de Sistemas II

- Prototype

```
1 package gof.prototype;
2
3 public abstract class Figura implements Cloneable {
4
5     String name;
6
7     protected Figura clone() {
8         Object clone = null;
9         try {
10             clone = super.clone();
11         } catch (CloneNotSupportedException e) {
12             e.printStackTrace();
13         }
14         return (Figura) clone;
15     }
16 }
17
```

Análise e Projeto de Sistemas II

- Prototype

```
1 package gof.prototype;
2
3 public class Circulo extends Figura {
4
5     public Circulo() {
6         this.name = "Circulo";
7     }
8 }
```

```
1 package gof.prototype;
2
3 public class Quadrado extends Figura{
4
5     public Quadrado() {
6         this.name = "Quadrado";
7     }
8 }
```

Análise e Projeto de Sistemas II

- Prototype

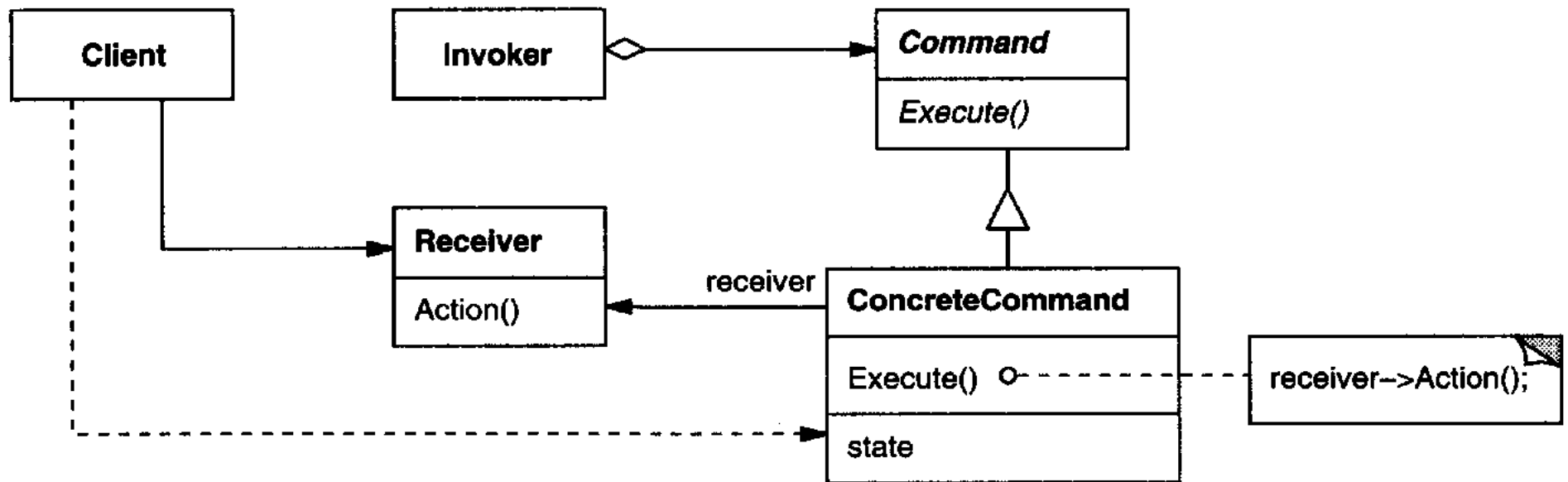
```
1 package gof.prototype;
2
3 import java.util.HashMap;
4 import java.util.Map;
5 import java.util.Scanner;
6
7 public class Cliente {
8
9     Map<String, Figura> figuras = new HashMap<String, Figura>();
10
11     public Cliente() {
12         this.createMap();
13     }
14
15     public void createMap() {
16         this.figuras.put("C", new Circulo());
17         this.figuras.put("Q", new Quadrado());
18     }
19
20     public static void main(String[] args) {
21
22         Scanner sc = new Scanner(System.in);
23         System.out.println("Digite o código: ");
24         String cod = sc.next();
25         Cliente cl = new Cliente();
26         System.out.println(cl.figuras.get(cod).clone().name);
27
28     }
29 }
30
```

Análise e Projeto de Sistemas II

- Command
 - Encapsule uma solicitação como um objeto, permitindo assim que você faça a parametrização de clientes com diferentes solicitações, enfileire ou registre solicitações e dê suporte a operações que podem ser desfeitas

Análise e Projeto de Sistemas II

- Command



Análise e Projeto de Sistemas II

- Command

```
1 package gof.command;  
2  
3 public interface Command {  
4     public void execute();  
5 }  
6  
7 }
```

```
1 package gof.command;  
2  
3 public class Lampada {  
4  
5     public void ligar() {  
6         System.out.println("liga");  
7     }  
8  
9     public void desligar() {  
10        System.out.println("desliga");  
11    }  
12  
13 }
```

Análise e Projeto de Sistemas II

- Command

```
1 package gof.command;
2
3 public class LigarCommand implements Command {
4
5     Lampada lamp;
6
7     public LigarCommand(Lampada l) {
8         this.lamp = l;
9     }
10
11     @Override
12     public void execute() {
13         this.lamp.ligar();
14     }
15 }
16 }
```

```
1 package gof.command;
2
3 public class DesligarCommand implements Command {
4
5     Lampada lamp;
6
7     public DesligarCommand(Lampada l) {
8         this.lamp = l;
9     }
10
11     @Override
12     public void execute() {
13         this.lamp.desligar();
14     }
15 }
16 }
```


Análise e Projeto de Sistemas II

- Command

```
1 package gof.command;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class Invoker {
7
8     List<Command> comandos = new ArrayList<Command>();
9
10    public void enfileirarComando(Command comando) {
11        comandos.add(comando);
12    }
13
14    public void executar(){
15
16        for (Command comando : comandos) {
17            comando.execute();
18        }
19    }
20
21 }
```

Análise e Projeto de Sistemas II

- Command

```
1 package gof.command;
2
3 public class Cliente {
4
5     public static void main(String[] args) {
6         Lampada lamp = new Lampada();
7
8         LigarCommand ligar = new LigarCommand(lamp);
9         DesligarCommand desligar = new DesligarCommand(lamp);
10
11         Invoker invoker = new Invoker();
12         for (int i=1; i<10; i++) {
13             invoker.enfileirarComando(ligar);
14             invoker.enfileirarComando(desligar);
15         }
16         invoker.executar();
17     }
18 }
19
```

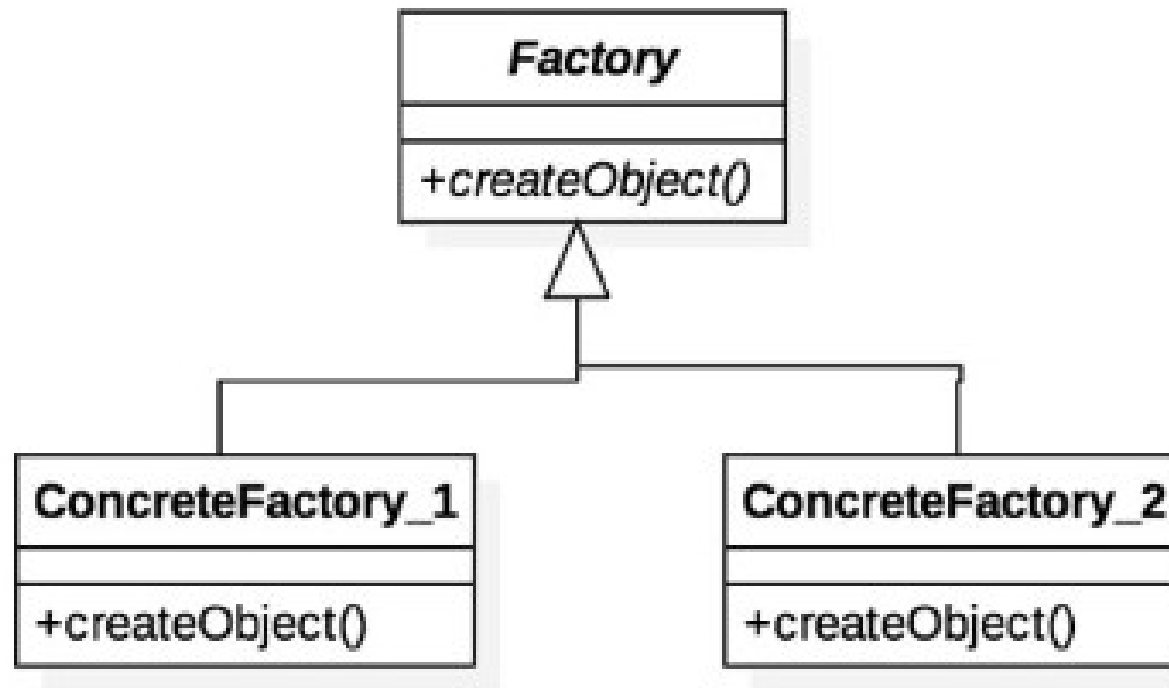
```
<terminated> C\
liga
desliga
liga
desliga
liga
desliga
liga
desliga
liga
desliga
liga
desliga
liga
```

Análise e Projeto de Sistemas II

- Factory Method
 - Defina uma interface para criar um objeto, mas deixe as subclasses decidirem qual classe instanciar. O Factory Method permite que uma classe adie a instanciación para subclasses.
 - Relacionado ao Simple Factory e ao Abstract Factory

Análise e Projeto de Sistemas II

- Factory Method





Análise e Projeto de Sistemas II

Nos vemos por aí!

Muito obrigado!

fernando.xavier@gmail.com