Design Patterns:
Adapter

William Gabriel Pereira

Instituto Federal Catarinense
Campus Rio do Sul

May 25, 2019

agenda

1 Design Patterns Estruturais
- O que resolvem

2 Adapter
- Como funciona

3 Definição

4 Programando
- Interfaces
- Classes concretas
- Factory
- Main
- Adapter em si

5 Água em vinho
- ICopoAgua
- Vinho
- Jesus
- Factory

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Design Patterns Estruturais

Design Patterns Estruturais

O que é um padrão estrutural

Assim como os outros Design Patterns estudados anteriormente, estes padrões são usados para resolver problemas comuns entre os sistemas. A diferença é "o que" resolvem.

O que resolvem então?

Tratam a maneira de como as classes são armazenadas em um projeto, ou seja, esses padrões são envelopadores. É utilizada classe dentro de uma outra classe, modificando algo em tal objeto/função, ainda em tempo de execução. Em suma, são classes que modificam a disposição e alteram funções de outras classes, tornando elas compativeis com o projeto.

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Padrões estruturais:

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Adapter

Adapter

O que é

Como diz o deu próprio nome, o padrão Adapter é um padrão usado para adaptar uma classe, ou seja, serve para flexibilidade e compatibilidade.

Como funciona

Imagine o seguinte: você fez um sistema que utiliza uma *API* com *WEBSERVICE*, porém o *WEBSERVICE* foi fechado, então o que fazer? modificar todo o sistema para funcionar com uma nova *API*, certo? **ERRADO**!

Para não gastar muito tempo modificando todo o sistema, fazemos um adaptador, uma classe que acessa outra *WEBSERVICE*, porém sem mexer muito no código principal.

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Outro exemplo

Vamos para um exemplo mais cotidiano. Quando temos uma tomada de pinos planos, porém a entrada da tomada é de pinos redondos, o que fazemos?
Em casos extremos, trocamos a ponta da tomada, mas somente em casos realmente extremos. Geralmente corremos atrás de um *T*, ou um adaptador.

O Adapter é exatamente isso, o intermediário, o adaptador de tomada.

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Definição

Definição Oficial

O Padrão Adapter converte uma interface de uma classe para outra interface que o cliente espera encontrar. O Adaptador permite que classes com interfaces incompatíveis trabalhem juntas

"Se anda como um pato, grasna como um pato, então talvez seja peru envelopado num adaptador de pato"

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Programando

Programando

Interface

Para que funcione, precisamos de uma interface com as funções que são implementadas na classe que desejamos modificar, neste caso a *TomadaPlana*. O cliente vai estar solicitando a função pela interface, porém a implementação da interface vai estar no adaptador, que vai chamar a função equivalente da *TomadaCircular*.

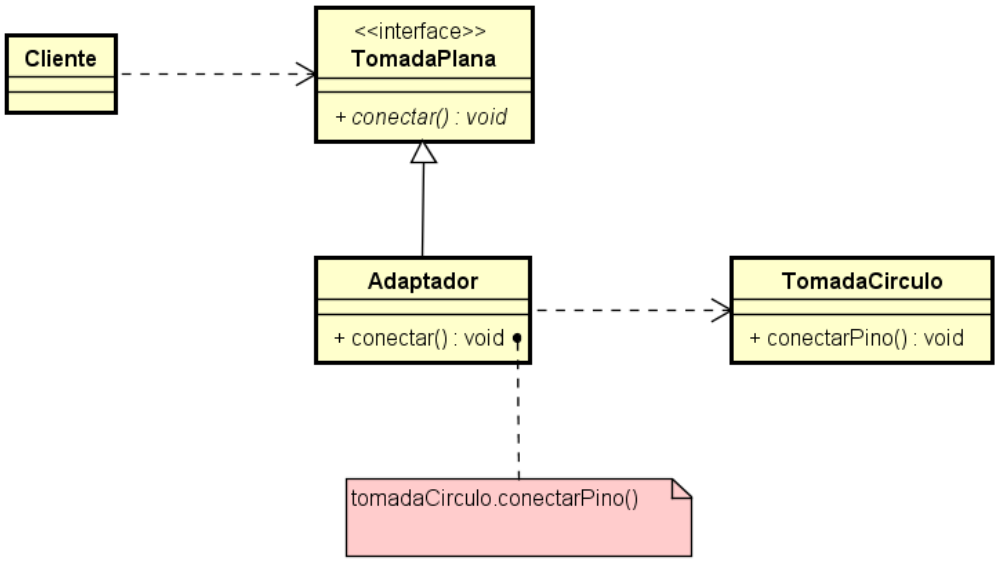

Figure: Diagrama do exemplo tomada

INSTITUTO FEDERAL Catarinense Campus Rio do Sul

Implementar primeiro as interfaces:

- ITomadaPlana
- ITomadaCircular

```
1 public interface ITomadaPlana {
2   public void conectar();
3 }
```

```
1 public interface ITomadaCircular {
2   public void conectarCirculo();
3 }
```

INSTITUTO FEDERAL Catarinense Campus Rio do Sul

Implementar então as classes concretas:

- TomadaPlana
- TomadaCircular

```java
public class TomadaPlana implements ITomadaPlana{
  @Override
  public void conectar() {
    System.out.println("Conectado Plano");
  }
}
```

```java
public class TomadaCircular implements ITomadaCircular{
  @Override
  public void conectarCirculo() {
    System.out.println("Conectado circulo");
  }
}
```

INSTITUTO FEDERAL Catarinense Campus Rio do Sul

Usei uma Factory para encapsular melhor as classes não Main

```
1 public class Factory {
2   public static ITomadaPlana criar() {
3     return new TomadaPlana();
4   }
```

Façamos então uma classe Main para testar as classes que vamos adaptar antes de adaptar

```
1 public class Main {
2
3   public static void main(String[] args) {
4     ITomadaPlana plana = Factory.criar();
5
6     plana.conectar();
7   }
8
9 }
```

Conectado Plano

```
1 public class Main2 {
2   public static void main(String[] args) {
3     ITomadaCircular circular = new TomadaCircular();
4
5     circular.conectarCirculo();
6   }
7 }
```

Conectado circulo

INSTITUTO FEDERAL Catarinense Campus Rio do Sul

Agora sim vamos implementar a classe adaptadora

```
1  public class Adaptador implements ITomadaPlana{
2
3    TomadaCircular t;
4
5    public Adaptador() {
6      t = new TomadaCircular();
7    }
8
9    @Override
10   public void conectar() {
11     t.conectarCirculo();
12   }
13
14 }
```

Agora façamos essa pequena alteração na classe Factory e rodemos o primeiro Main

```
1  public static ITomadaPlana criar() {
2    return new Adaptador();
3  }
```

Em teoria, a tomada plana deveria conectar em tomada plana, porém com o adaptador temos como print o seguinte:

Conectado circulo

Água em vinho

Água em vinho

Podemos com isso, brincar de Jesus, vamos transformar a água em vinho agora!

ICopoAgua

```
1 public interface ICopoAgua {
2   public void encher();
3 }
```

Vinho

```
1 public class Vinho {
2   public void encherVinho() {
3     System.out.println("Cheio de vinho!");
4   }
5 }
```

INSTITUTO FEDERAL Catarinense Campus Rio do Sul

Jesus

```
1  public class Jesus implements ICopoAgua{
2    Vinho v;
3
4    public Jesus() {
5      v = new Vinho();
6    }
7
8    @Override
9    public void encher() {
10     v.encherVinho();
11   }
12 }
```

Factory

```
1  public class Factory {
2    public static ICopoAgua criar() {
3      return new Jesus();
4    }
5  }
```

INSTITUTO FEDERAL
Catarinense
Campus
Rio do Sul

Main

```
1  public class Main {
2
3    public static void main(String[] args) {
4      ICopoAgua agua = Factory.criar();
5
6      agua.encher();
7
8    }
9
10 }
```

Temos como resultado disso tudo a transformação da água em vinho ao rodar o nosso Main

Cheio de vinho!

INSTITUTO FEDERAL Catarinense Campus Rio do Sul