
Métodos Avançados de Programação

PADRÕES DE PROJETO

DR^a ALANA MORAIS

Padrões GoF – Padrões de Estrutura

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Factory Method	Class Adapter	Interpreter Template Method
Objeto	Classe	Abstract Factory Builder Prototype Singleton	Object Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Padrões de Estrutura

Lidam com as estruturas do projeto, facilitando a comunicação entre suas entidades.

- Padrões de classe: **herança**
- Padrões de objeto: **composição**

“Padrões de estrutura com escopo de classe usam herança para compor interfaces ou implementações. Os com escopo de objeto descrevem formas de compor objetos para realizar novas funcionalidades.”

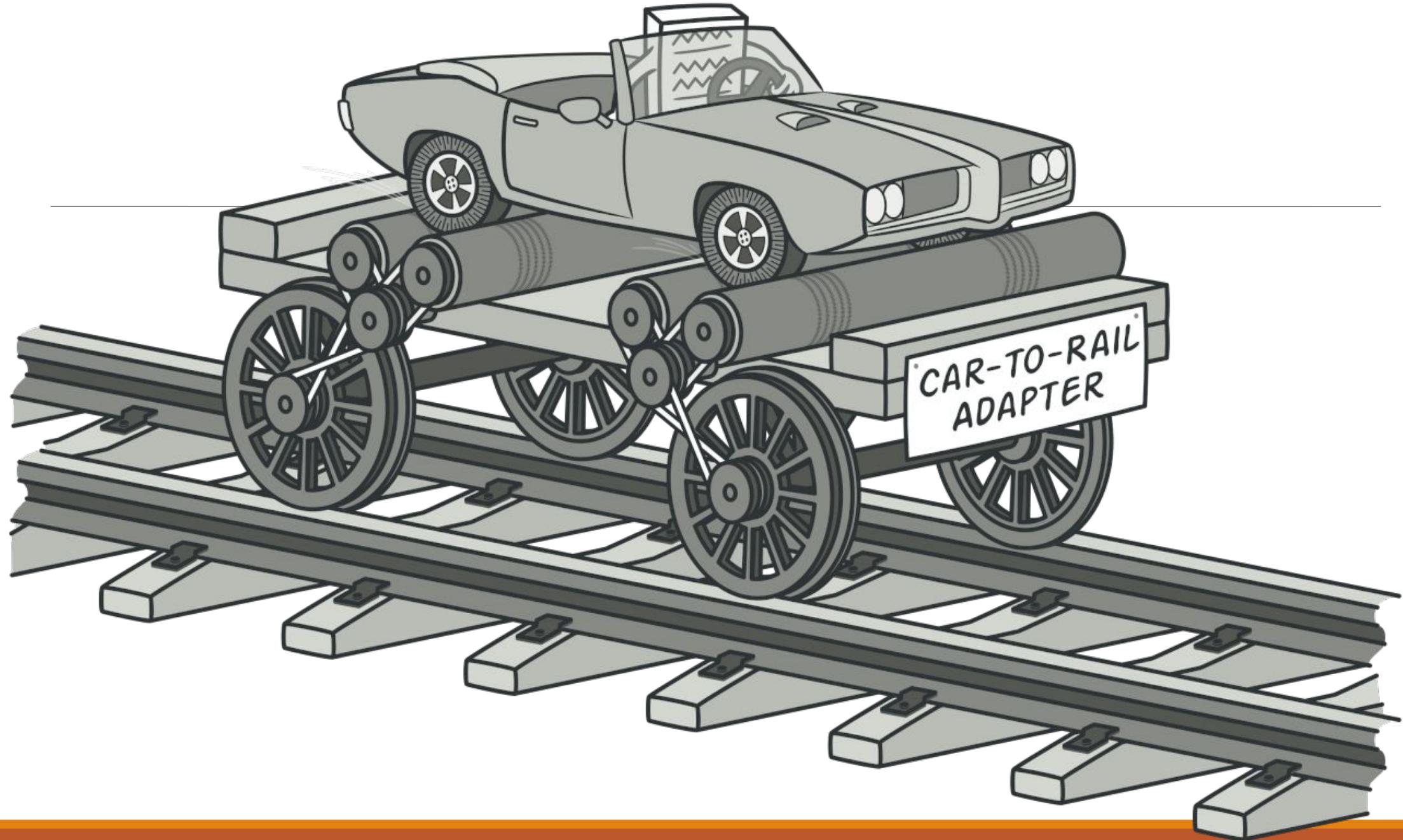
Adapter

"Converter a interface de uma classe em outra interface esperada pelos clientes. Adapter permite a comunicação entre classes que não poderiam trabalhar juntas devido à incompatibilidade de suas interfaces."

Padrão Adapter

Soluções simples para problemas reais!





Adapter

Intenção:

- Converter a interface de uma classe em outra interface esperada pelo cliente.
- Permite que classes com interfaces incompatíveis possam colaborar.

Também conhecido como:

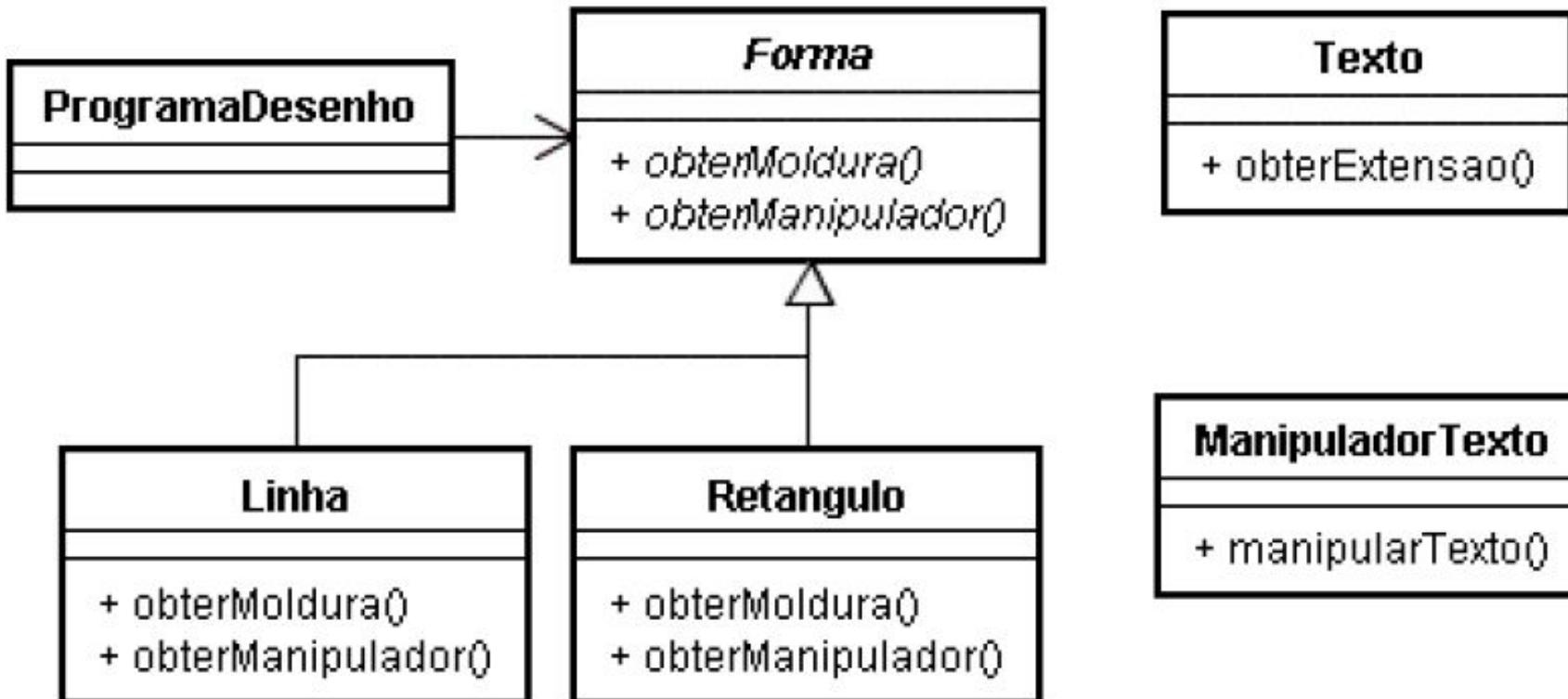
- Wrapper.

JSDK API: Wrappers de tipos em Java

- Double, Integer, Character, etc.
 - "Adaptam" tipos primitivos à interface de java.lang.Object.

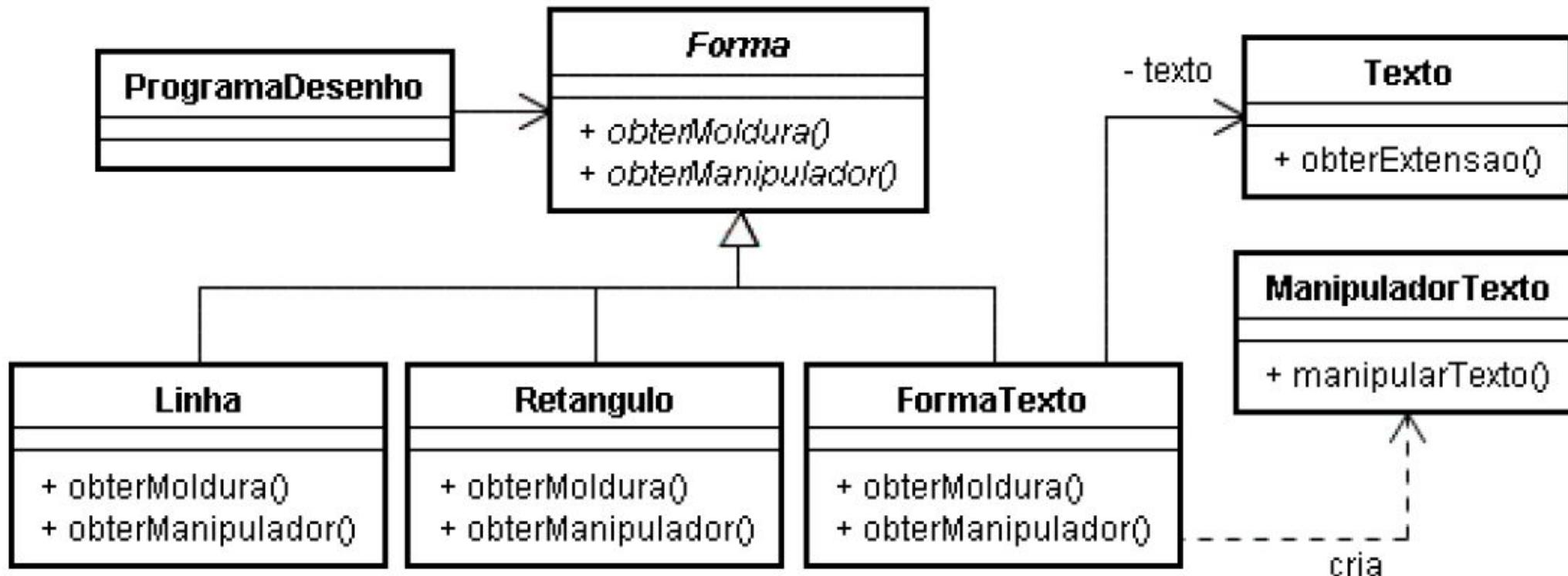
Adapter Problema

Existe uma ferramenta gráfica de texto pronta, mas o programa de desenho só pode trabalhar com formas.



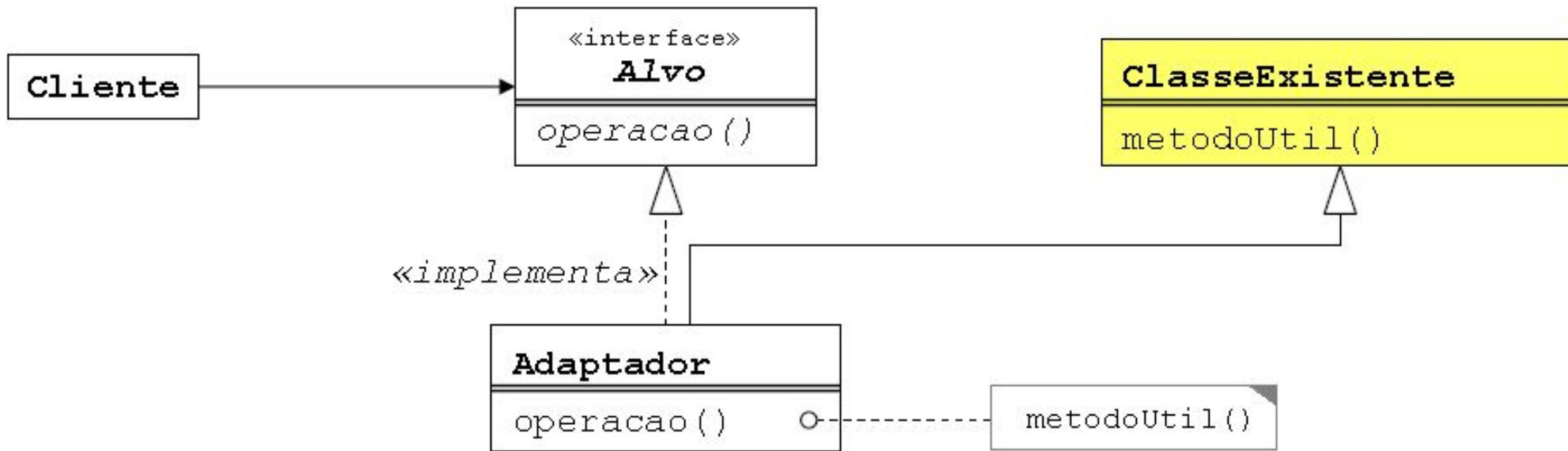
Adapter Solução

FormaTexto adapta a classe pronta à interface esperada pelo programa de desenho.



Adapter

(i) ClassAdapter: usa herança múltipla



1. Cliente: aplicação que colabora com objetos aderentes à interface Alvo
2. Alvo: define a interface requerida pelo Cliente
3. ClasseExistente: interface que requer adaptação
4. Adaptador(Adapter): adapta a interface do Recurso à interface Alvo

Adapter Class

```
public class ClienteExemplo {
    Alvo[] alvos = new Alvo[10];
    public void inicializaAlvos() {
        alvos[0] = new AlvoExistente();
        alvos[1] = new Adaptador();
        // ...
    }
    public void executaAlvos() {
        for (int i = 0; i < alvos.length; i++) {
            alvo.operacao();
        }
    }
}
```

```
public interface Alvo {
    void operacao();
}
```

```
public class Adaptador extends ClasseExistente implements Alvo {
    public void operacao() {
        String texto = metodoUtilDois("Operação Realizada.");
        metodoUtilUm(texto);
    }
}
```

```
public class ClasseExistente {
    public void metodoUtilUm(String texto) {
        System.out.println(texto);
    }
    public String metodoUtilDois(String texto) {
        return texto.toUpperCase();
    }
}
```

Adapter Class

```
public class ClienteExemplo {
    Alvo[] alvos = new Alvo[10];
    public void inicializaAlvos() {
        alvos[0] = new AlvoExistente();
        alvos[1] = new Adaptador();
        // ...
    }
    public void executaAlvos() {
        for (int i = 0; i < alvos.length; i++) {
            alvo.operacao();
        }
    }
}
```

```
public interface Alvo {
    void operacao();
}
```

```
public class Adaptador extends ClasseExistente implements Alvo {
    public void operacao() {
        String texto = metodoUtilDois("Operação Realizada.");
        metodoUtilUm(texto);
    }
}
```

```
public class ClasseExistente {
    public void metodoUtilUm(String texto) {
        System.out.println(texto);
    }
    public String metodoUtilDois(String texto) {
        return texto.toUpperCase();
    }
}
```

Adapter

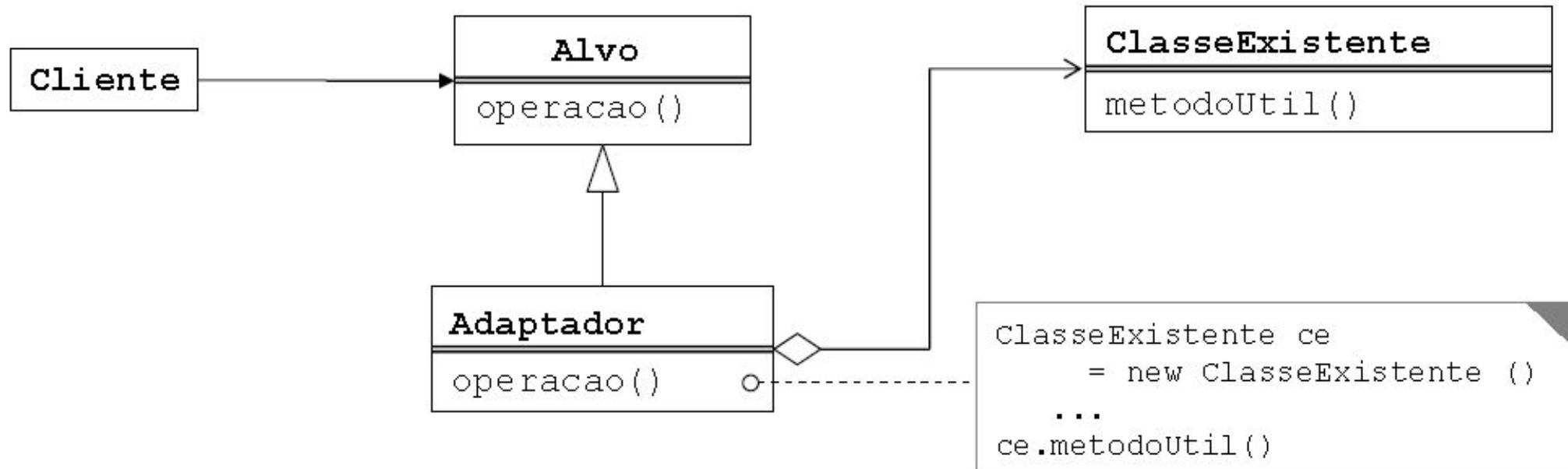
Vantagens Adapter de Classe

Adapter de Classe:

Não funciona bem quando se quer adaptar uma hierarquia de classes.

Permite que o adaptador sobrescreva algumas funções do adaptado.

Adapter (ii) ObjectAdapter: usa composição



1. Única solução se Alvo não for uma interface Java
2. Adaptador possui referência para objeto que terá sua interface adaptada (instância de **ClasseExistente**).
3. Cada método de **Alvo** chama o(s) método(s) correspondente(s) na interface adaptada.

Adapter Object

```
public class ClienteExemplo {  
    Alvo[] alvos = new Alvo[10];  
    public void inicializaAlvos() {  
        alvos[0] = new AlvoExistente();  
        alvos[1] = new Adaptador();  
        // ...  
    }  
    public void executaAlvos() {  
        for (int i = 0; i < alvos.length; i++) {  
            alvos[i].operacao();  
        }  
    }  
}
```

```
public abstract class Alvo {  
    public abstract void operacao();  
    // ... resto da classe  
}
```

```
public class Adaptador extends Alvo {  
    ClasseExistente existente = new ClasseExistente();  
    public void operacao() {  
        String texto = existente.metodoUtilDois("Operação Realizada.");  
        existente.metodoUtilUm(texto);  
    }  
}
```

```
public class ClasseExistente {  
    public void metodoUtilUm(String texto) {  
        System.out.println(texto);  
    }  
    public String metodoUtilDois(String texto) {  
        return texto.toUpperCase();  
    }  
}
```

Adapter

Vantagens Adapter de Objeto

Adapter de Objeto:

Permite o uso de um único adaptador para uma hierarquia de classes adaptadas.

É mais difícil sobrescrever funções do adaptado.

Adapter

Quando Usar

Há uma classe já pronta que possui uma interface diferente da qual você precisa;

Necessita criar uma classe reutilizável já prevendo que a situação acima ocorrerá no future;

Sempre que for necessário adaptar uma interface para um cliente.

ClassAdapter

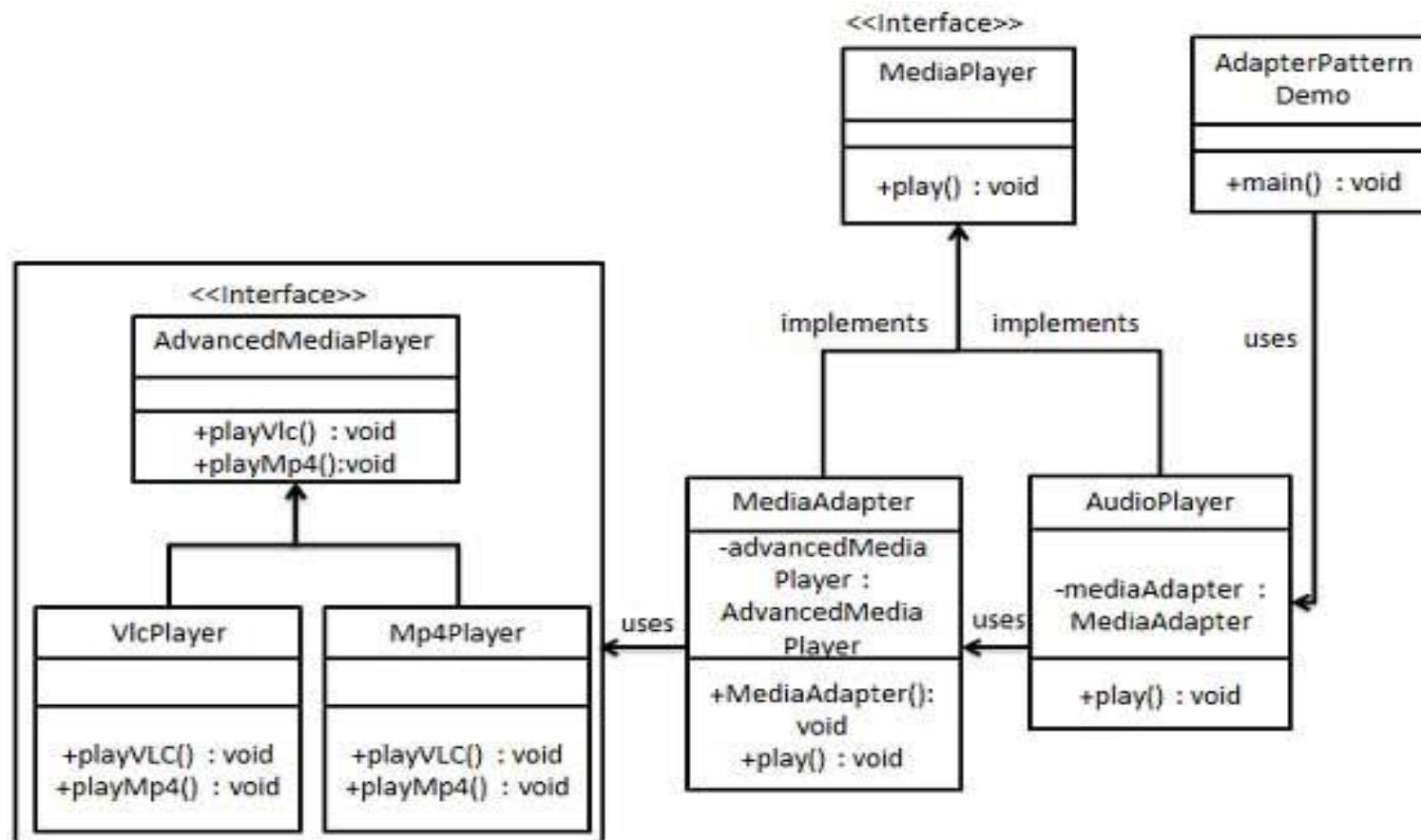
- Quando houver uma interface que permita a implementação estática

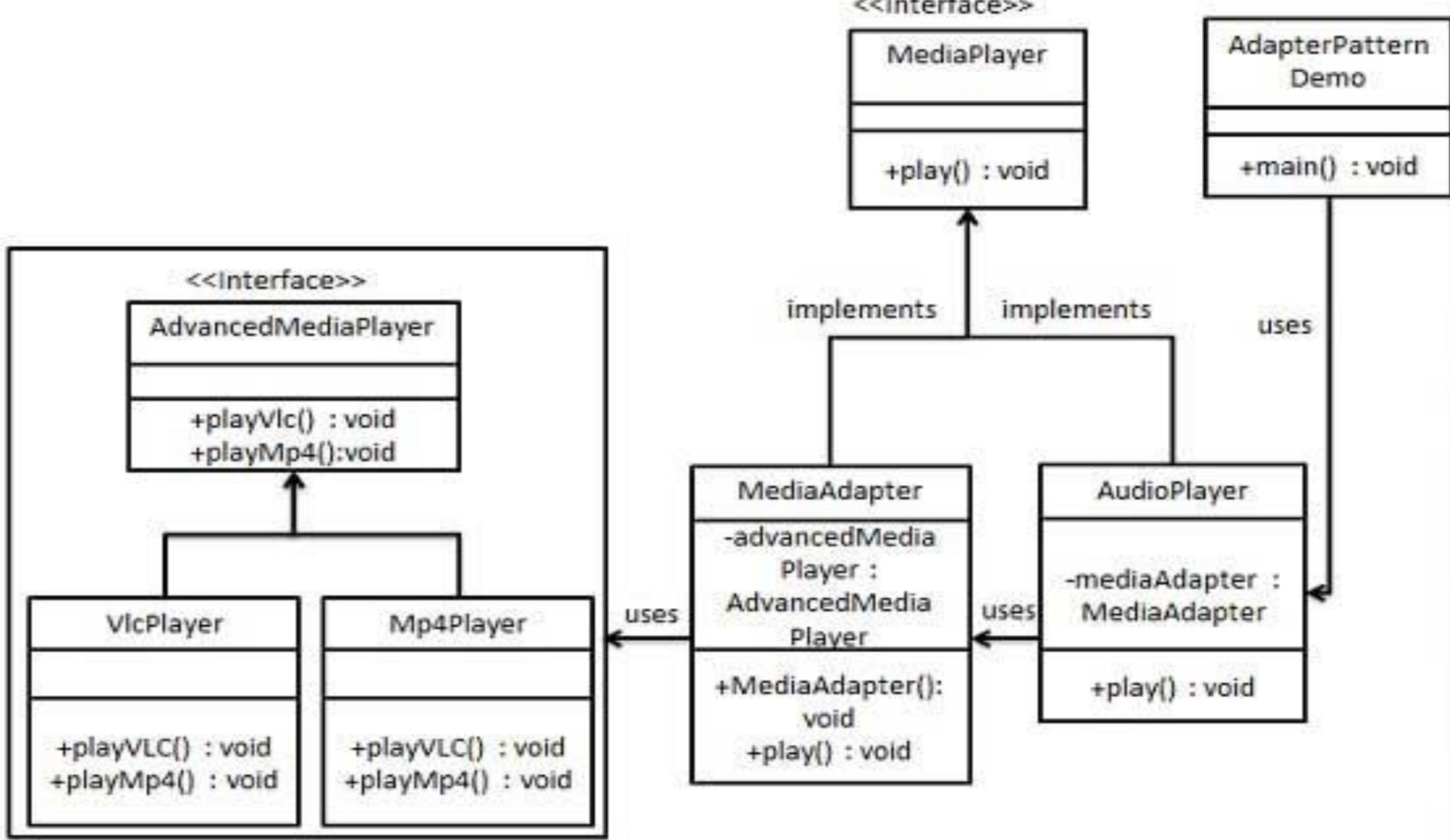
ObjectAdapter

- Quando menor acoplamento for desejado
- Quando o cliente não usa uma interface Java ou classe abstrata que possa ser estendida

Adapter

Exemplo 2 – Qual o tipo de Adapter?





Adapter Exercício

Considere os códigos fonte de um cliente, uma interface para um somador que ele espera utilizar e uma classe concreta que implementa uma soma, mas não da maneira esperada pelo cliente.

Como você pode ver abaixo, o cliente espera usar uma classe que soma inteiros em um vetor, mas a classe pronta soma inteiros em uma lista.

Crie um adaptador para resolver esta situação.

```
public class Cliente {
    private SomadorEsperado somador;
    private Cliente(SomadorEsperado somador) {
        this.somador = somador;
    }
    public void executar() {
        int[] vetor = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        int soma = somador.somaVetor(vetor);
        System.out.println("Resultado: " + soma);
    }
}

public interface SomadorEsperado {
    int somaVetor(int[] vetor);
}

import java.util.List;
public class SomadorExistente {
    public int somaLista(List<Integer> lista) {
        int resultado = 0;
        for (int i : lista) resultado += i;
        return resultado;
    }
}
```

Façade

“OFERECER UMA INTERFACE ÚNICA PARA UM CONJUNTO DE INTERFACES DE UM SUBSISTEMA. FAÇADE DEFINE UMA INTERFACE DE NÍVEL MAIS ELEVADO QUE TORMA O SUBSISTEMA MAIS FÁCIL DE USAR.”

Façade

Provê uma interface unificada para um conjunto de interfaces de um subsistema

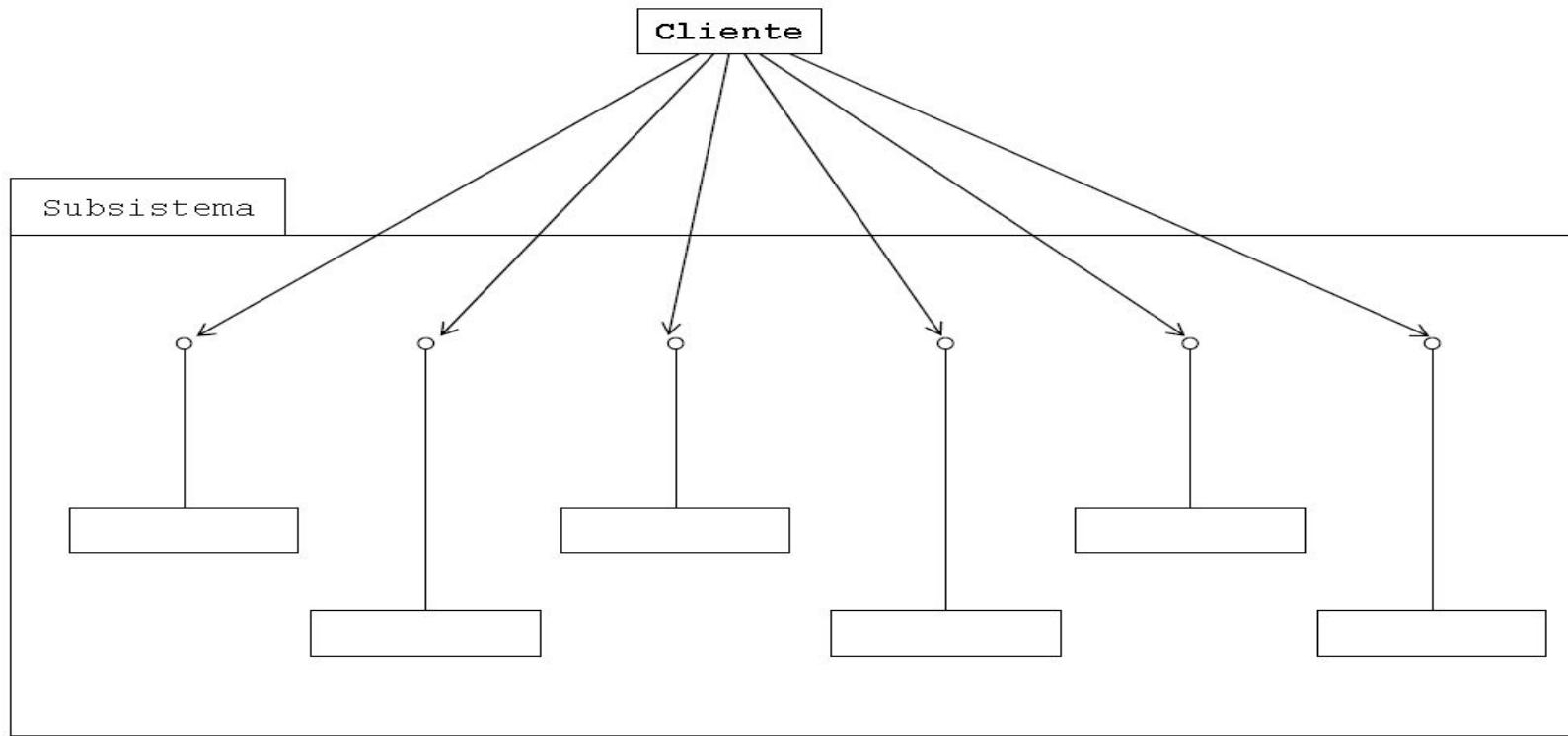
Define uma interface de mais alto nível que torna o subsistema mais fácil de manipular

Use o Façade quando:

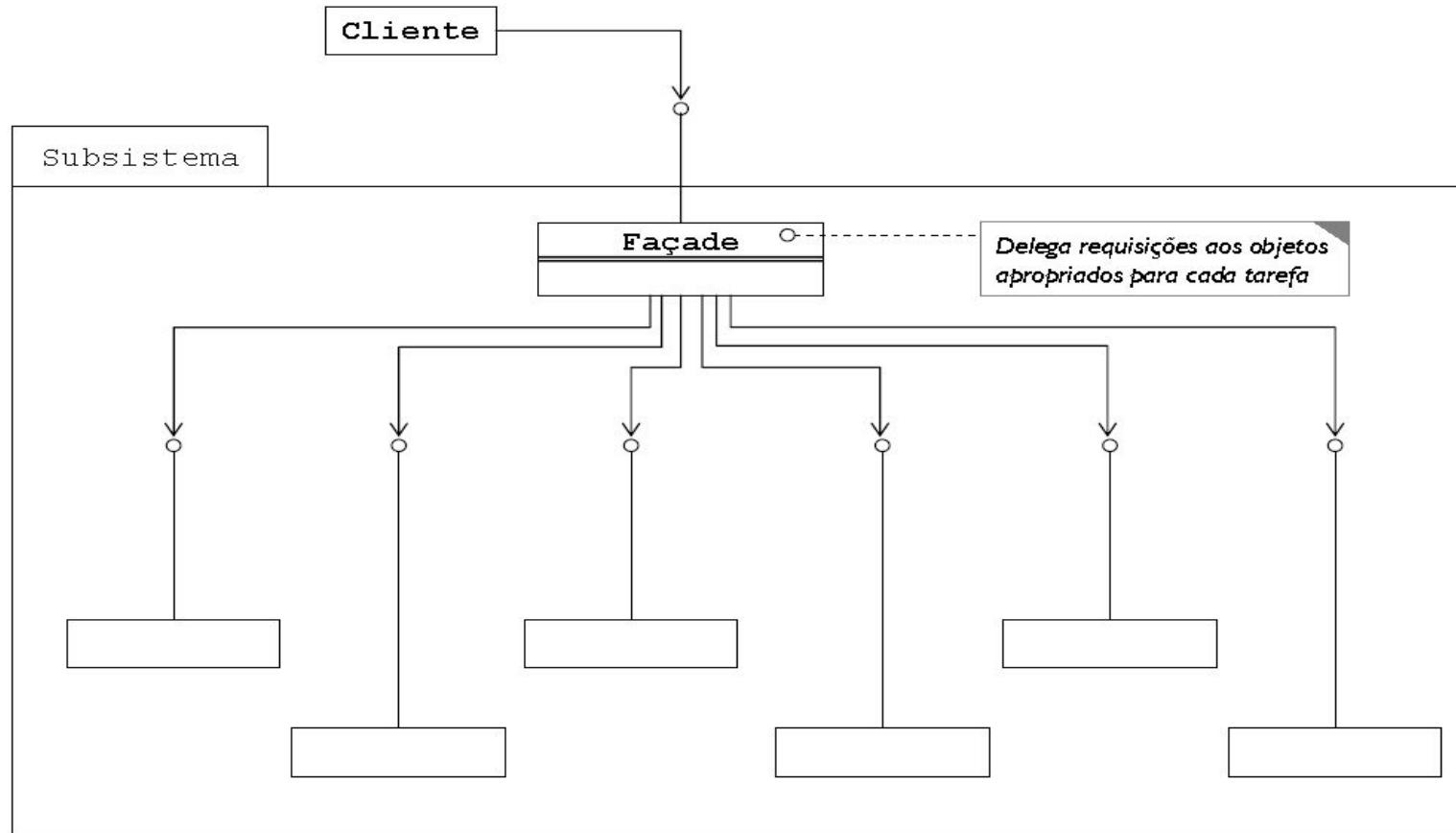
- Você quiser prover uma interface simples para um subsistema complexo

Façade Problema

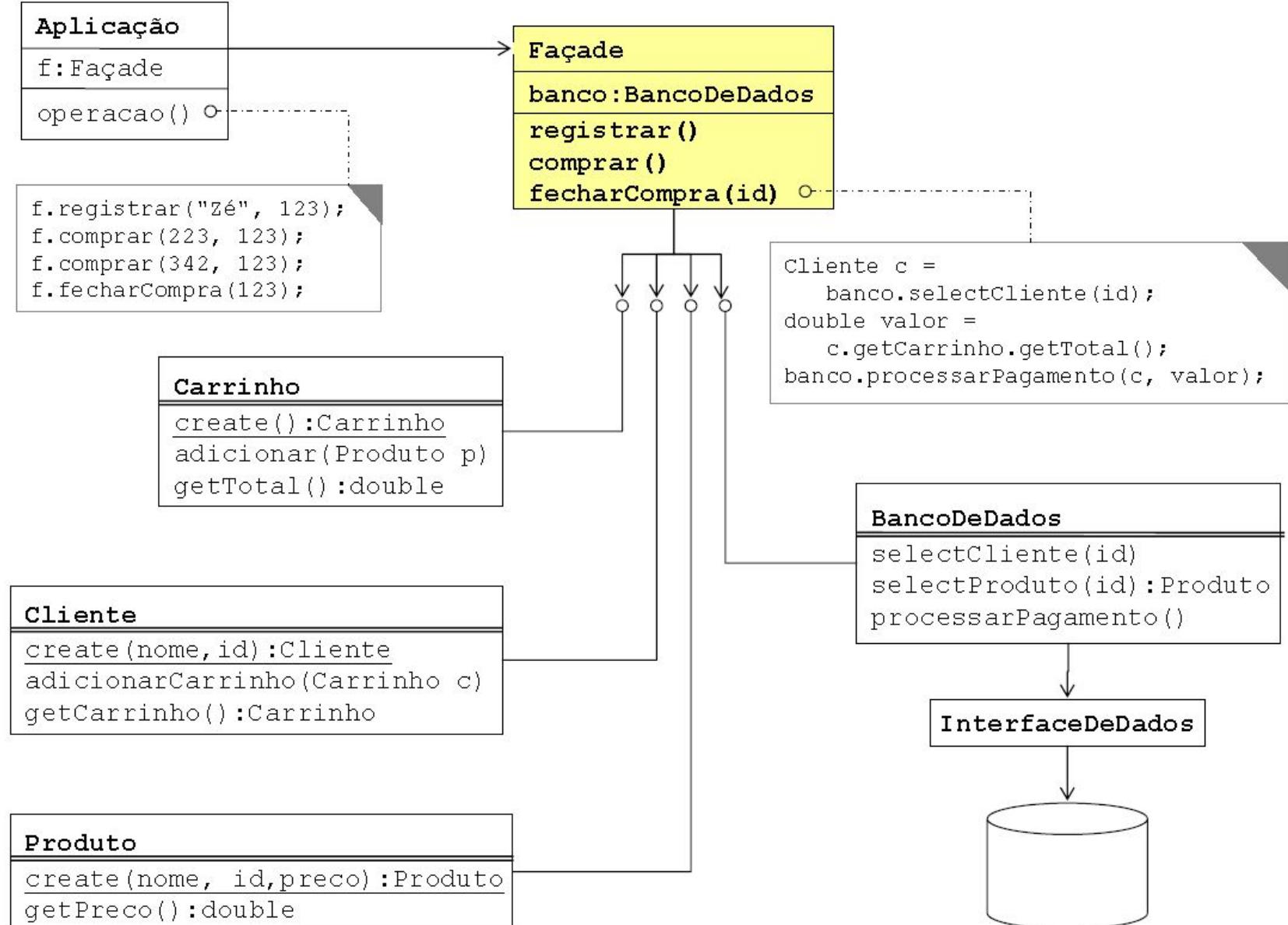
Cliente precisa saber muitos detalhes do subsistema para utilizá-lo!



Façade Solução – Crie uma fachada!!



Façade Exemplo



Façade Vantagens e Desvantagens

Facilita a utilização do sistema:

- Cliente só precisa conhecer a fachada.

Promove acoplamento fraco:

- Pequenas mudanças no subsistema não mais afetam o cliente.

Versatilidade:

- Quando necessário, clientes ainda podem acessar o subsistema diretamente (se quiser permitir isto).

Façade e Singleton

Façade geralmente é implementado como Singleton;

Pode não ser o caso se o sistema tiver múltiplos usuários e cada um usar uma fachada separada;

Fachada só com métodos estáticos é chamada de Utilitário.

Façade Exercício

Controle as funções principais de um computador por meio de uma fachada. A classe de fachada deve ser responsável por controlar CPU, Memória e o HardDrive durante o processo de ligar o computador.

Os métodos que devem ser implementados nas classes estão descritos na tabela a seguir:

Cpu	Memória	HardDrive
start()	load()	read()
execute()	free()	write()
load()		
free()		

Bridge

"DESACOPLAR UMA ABSTRAÇÃO DE SUA IMPLEMENTAÇÃO PARA QUE OS DOIS POSSAM VARIAR INDEPENDENTEMENTE."

Bridge

Intenção:

- Desacoplar uma abstração de sua implementação para que ambos possam variar independentemente.

Também conhecido como:

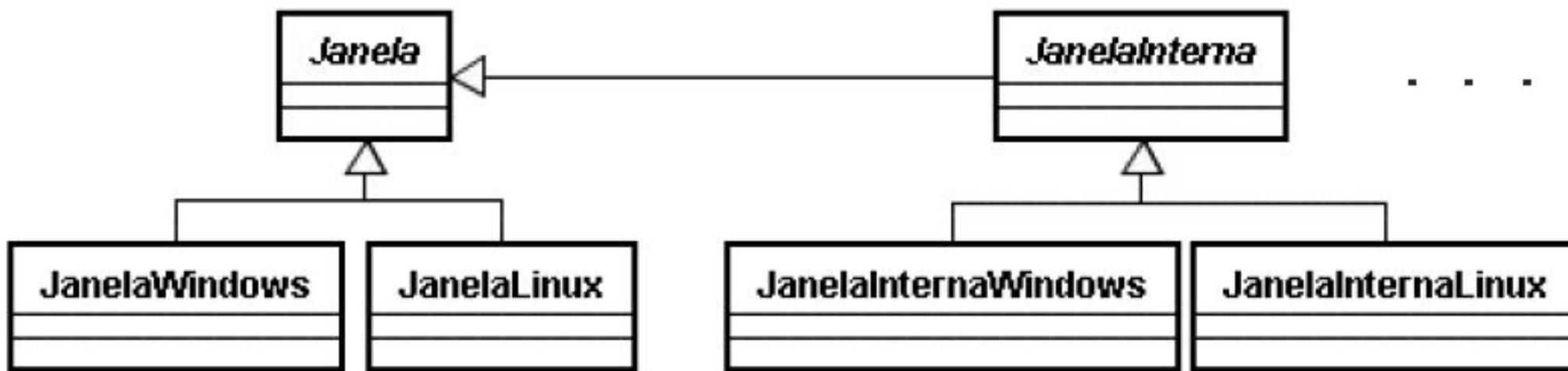
- Handle, Body
- Exemplo: clientes (programadores) somente precisam interagir com JFrame, JButton, etc. ao manipular interfaces gráficas.

Bridge Problema

Componentes gráficos devem ser implementados para várias arquiteturas;

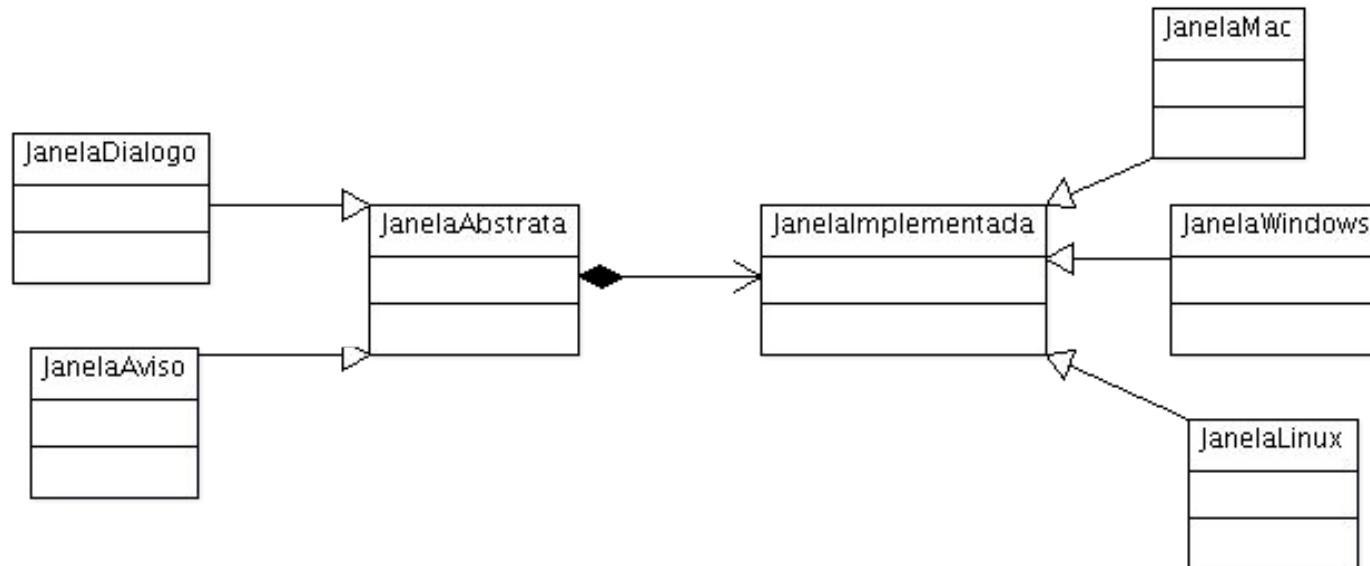
Cada novo componente exige várias implementações;

Cada nova arquitetura também.

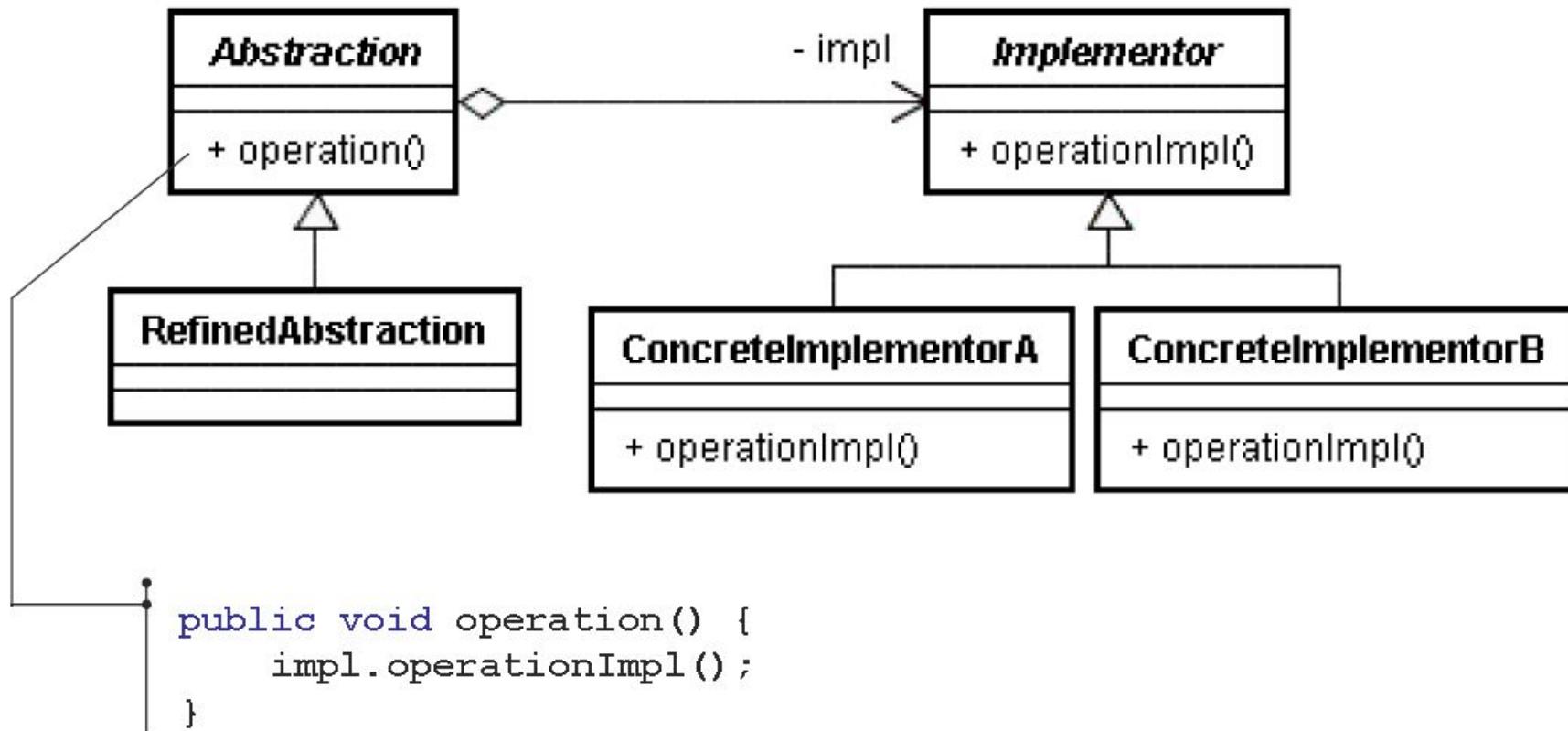


Bridge Solução

Janela concentra métodos que utilizam recursos específicos de plataforma;
Subclasses utilizam os métodos de Janela para implementar itens específicos.



Bridge Estrutura



Bridge Estrutura

Abstraction

- Define a interface de abstração. Mantém uma referência a um objeto do tipo Implementador.

RefinedAbstraction

- Estende a interface definida por Abstração.

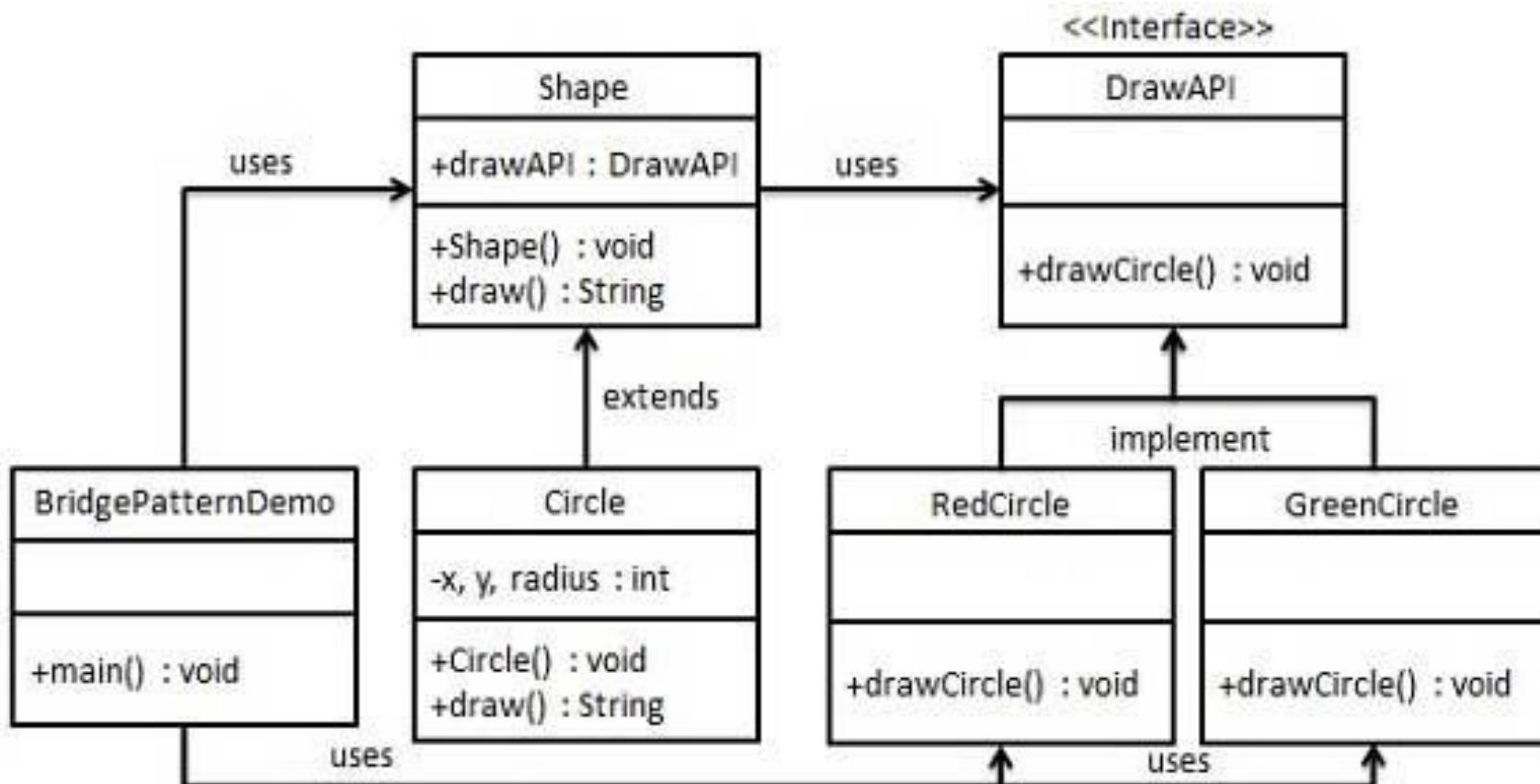
Implementor

- Define a interface para classes de implementação. Esta não tem a obrigação de corresponder exatamente à interface de abstração. De fato, as duas interfaces podem ser bastante diferentes. Tipicamente, a interface de implementação fornece apenas operações primitivas, cabendo à abstração a responsabilidade de definir operações de alto nível baseadas nestas primitivas.

ConcreteImplementorA e ConcreteImplementorB

- Implementação concreta da interface definida por Implementador.

Bridge Exemplo 2



Bridge

Quando Usar?

Quando for necessário evitar uma ligação permanente entre a interface e implementação

Quando alterações na implementação não puderem afetar clientes

Quando tanto abstrações como implementações precisarem ser capazes de suportar extensão por meio da herança

Quando implementações são compartilhadas entre objetos desconhecidos do cliente

Bridge Vantagens e desvantagens

Desacoplar a implementação:

- Podendo até mudá-la em tempo de execução.

Melhorar a extensibilidade:

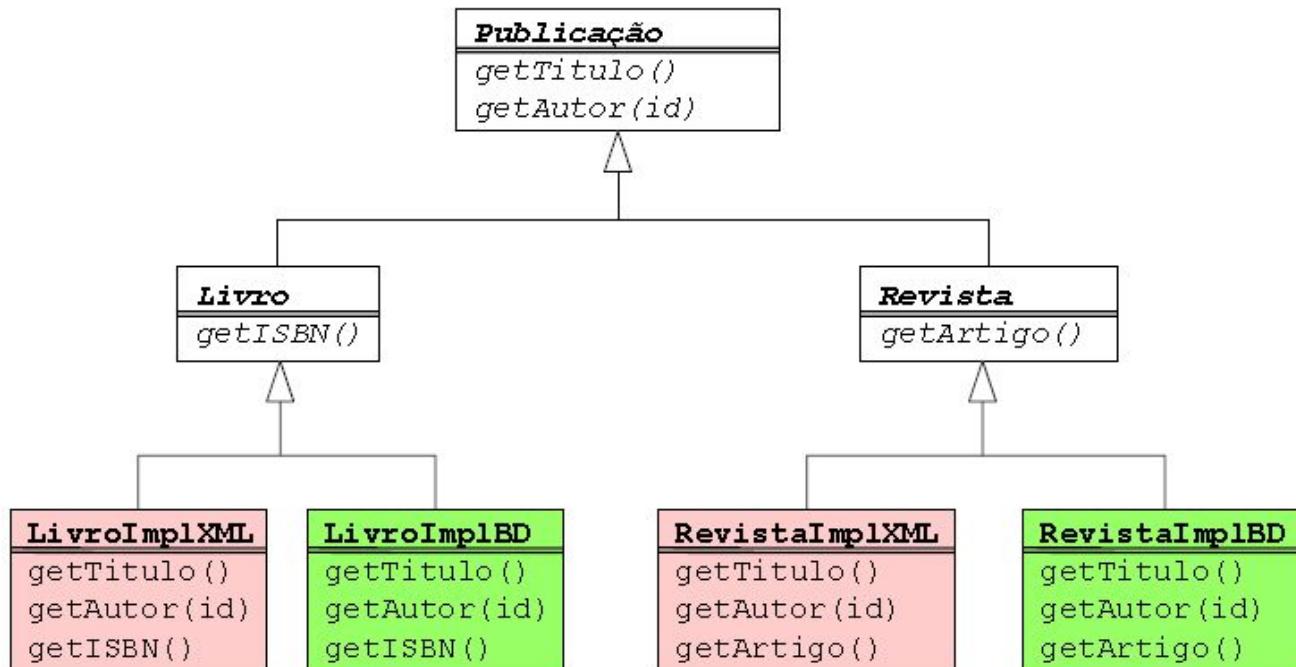
- É possível estender a abstração e a implementação separadamente.

Esconder detalhes de implementação:

- Clientes não precisam saber como é implementado.

Bridge Exercício

Utilize a arquitetura apresentada a seguir utilizando o padrão Bridge. A estrutura abaixo serve para lidar com implementações específicas para tratar um objeto em diferentes meios persistentes.



Dúvidas?

ALANAMM.PROF@GMAIL.COM