Instituto de Computação

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrões Adapter & Facade

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Padrões de Projeto

Introdução

Como fazer a interface de um objeto parecer o que ela de fato não é?

Como adaptar um projeto que espera uma interface para uma classe que implementa uma interface distinta?

Como encapsular objetos com o intuito de simplificar suas interfaces?

Padrões de Projeto

Exemplo de Problema

Como fazer a interface de um objeto parecer o que ela de fato não é?

Como adaptar um projeto que espera uma interface para uma classe que implementa uma interface distinta?

Como encapsular objetos com o intuito de simplificar suas interfaces?

Padrões de Projeto

Um primeiro exemplo

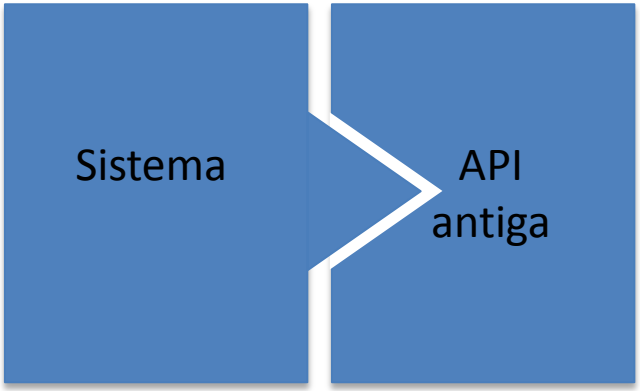
Considere duas classes que representam respectivamente uma peça quadrada SquarePeg e uma peça redonda RoundPeg com diferentes interfaces

```
public class SquarePeg {
   public void insert(String str) {
   System.out.println("SquarePeg insert(): " + str);
  }
}
```

```
public class RoundPeg {
   public void insertIntoHole(String msg) {
   System.out.println("RoundPeg insertIntoHole(): " + msg);
  }
}
```

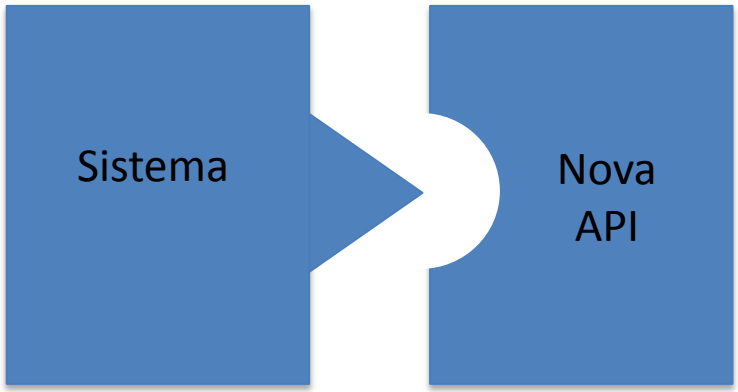
O cliente somente entende a interface de SquarePeg, como os método insert(String str) então como podemos inserir peças redondas via a interface insertIntoHole(String msg)?

Considere um sistema de software que funciona bem utilizando uma API fornecida por um vendedor



O Problema de adaptação de interfaces

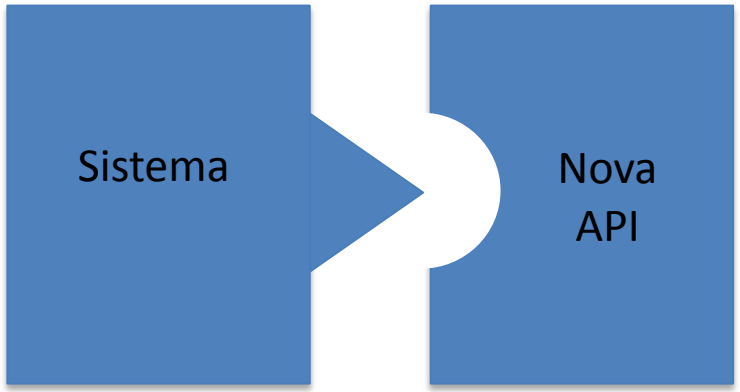
Em um determinado momento, o sistema precisa trabalhar com uma API semelhante, fornecida por um novo vendedor, que não implementa as interfaces esperadas pelo sistema



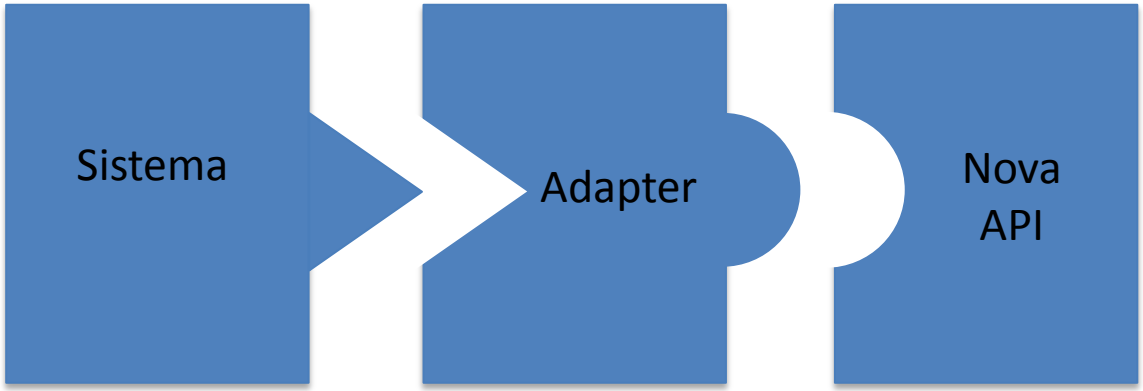
Sistema não deve ser modificado

As classes da nova API também não podem ser modificadas

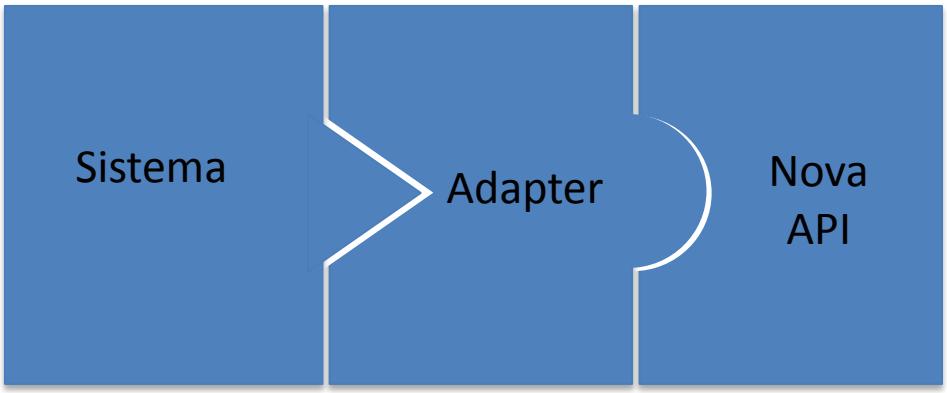
Como resolver tal problema?



Iremos criar uma classe adaptador que adapte a interface da nova API para a interface esperada pelo sistema



O Adaptador funciona como um meio de campo, obtendo as requisições do cliente e traduzindo tais requisições para um formato esperado pela classe do vendedor



Padrões de Projeto

Metáfora do Adaptador

Cliente

Adapter

Vendor
API

O Padrão Adapter converte a interface de uma classe em uma outra interface esperada pelo cliente. O adaptador permite que classes com interfaces originalmente incompatíveis possam trabalhar em conjunto.

Padrões de Projeto

Solução usando o Padrão Adapter

```
public class PegAdapter extends SquarePeg {
  private RoundPeg roundPeg;
  public PegAdapter(RoundPeg peg) {this.roundPeg = peg;}
  public void insert(String str) {roundPeg.insertIntoHole(str);}
}
```

```
public class TestPegs {
  public static void main(String args[]) {
  // Create some pegs.
  RoundPeg roundPeg = new RoundPeg();
  SquarePeg  squarePeg = new PegAdapter(roundPeg);
  // Do an insert using the square peg.
  squarePeg.insert("Inserting square peg...");
}
```

Padrões de Projeto

Um segundo exemplo

E se quiséssemos um adaptador que agisse como um SquarePeg ou um RoundPeg

Um solução é usar múltipla herança o que é impossível em Java

Entretanto, o adaptador pode implementar duas interfaces

Padrões de Projeto

Um segundo exemplo

E se quisessemos um adaptador que agisse como um SquarePeg ou um RoundPeg

Um solução é usar múltipla herança o que é impossível em Java

Entretanto, o adaptador pode implementar duas interfaces

Padrões de Projeto

Um segundo exemplo

```
public interface IRoundPeg {
   public void insertIntoHole(String msg);
}
```

```
public interface ISquarePeg {
   public void insert(String str);
}
```

Padrões de Projeto

Um segundo exemplo

```
public class SquarePeg implements ISquarePeg{
   public void insert(String str) {
   System.out.println("SquarePeg insert(): " + str);
  }
}
```

```
public class RoundPeg implements IRoundPeg{
   public void insertIntoHole(String msg) {
   System.out.println("RoundPeg insertIntoHole(): " + msg);
  }
}
```

Padrões de Projeto

Um segundo exemplo

```
public class PegAdapter implements ISquarePeg, IRoundPeg {
  private RoundPeg roundPeg;
  private SquarePeg squarePeg;
  public PegAdapter(RoundPeg peg) {this.roundPeg = peg;}
  public PegAdapter(SquarePeg peg) {this.squarePeg = peg;}
  public void insert(String str) {roundPeg.insertIntoHole(str);}
  public void insertIntoHole(String msg){squarePeg.insert(msg);}
}
```

Padrões de Projeto

Um segundo exemplo

```
public class PegAdapter implements ISquarePeg, IRoundPeg {

  private RoundPeg roundPeg;
  private SquarePeg squarePeg;

  public PegAdapter(RoundPeg peg) {this.roundPeg = peg;}
  public PegAdapter(SquarePeg peg) {this.squarePeg = peg;}
  public void insert(String str) {roundPeg.insertIntoHole(str);}
  public void insertIntoHole(String msg){squarePeg.insert(msg);}

}
```

Padrões de Projeto

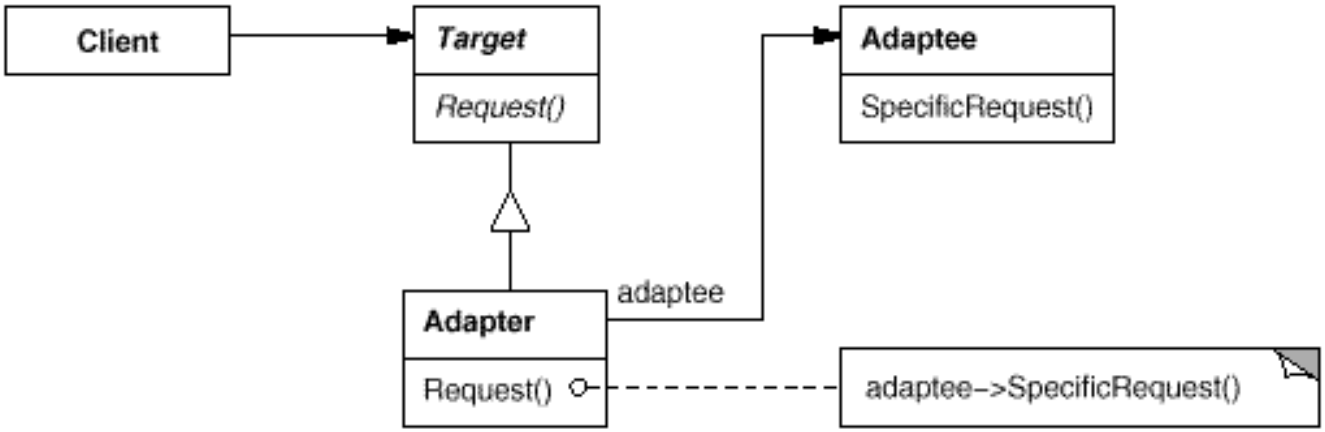
Um segundo exemplo

```
public class TestPegs {

  public static void main(String args[]) {
  // Create some pegs.
  RoundPeg roundPeg = new RoundPeg();
  SquarePeg squarePeg = new SquarePeg();
 // Do an insert using the square peg.
 squarePeg.insert("Inserting square peg...");
 // Create a two-way adapter and do an insert with it.
 ISquarePeg roundToSquare = new PegAdapter(roundPeg);
 roundToSquare.insert("Inserting round peg...");
 // Do an insert using the round peg.
 roundPeg.insertIntoHole("Inserting round peg...");
 // Create a two-way adapter and do an insert with it.
 IRoundPeg squareToRound = new PegAdapter(squarePeg);
 squareToRound.insertIntoHole("Inserting square peg...");
  }
}
```

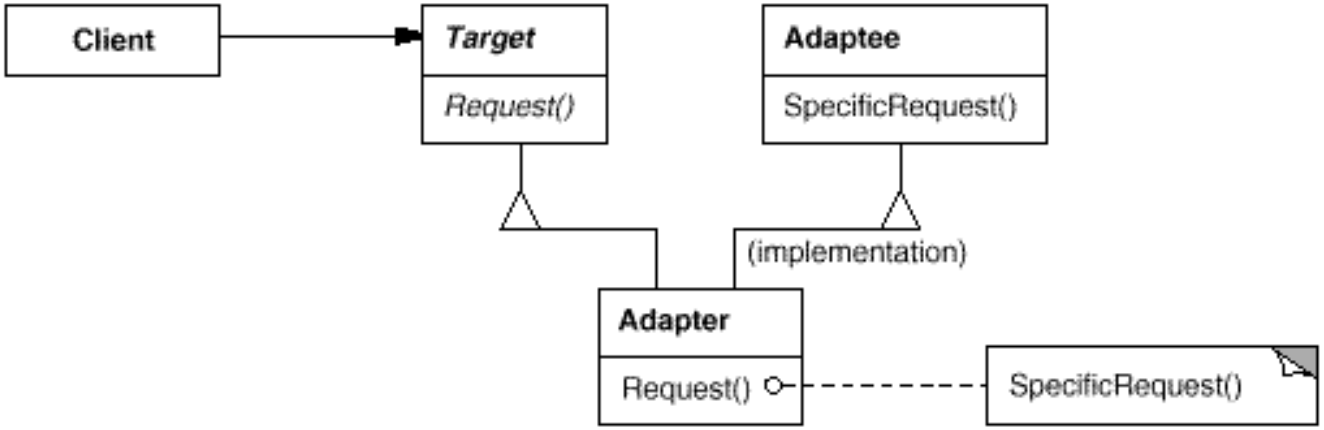
Padrões de Projeto

O Padrão Adapter – Diagrama de Classes



Padrões de Projeto

O Padrão Adapter (versão Class Adapter)– Diagrama de Classes



Utilização

Quando se quiser utilizar uma classe existente, mas sua interface não é adequada.

Quando se deseja criar e reutilizar uma classe para interoperar com classes ainda não existentes

Quando se deseja utilizar classes e é impraticável criar inúmeras subclasses.

Participantes

Alvo – define a interface utilizada pelo cliente

Cliente – utiliza os objetos com a interface alvo.

Adaptado – define a interface que se quer “modificar”

Adaptador – define a interface que será mapeada para a interface do adaptado.

Utilização

Quando se quiser utilizar uma classe existente, mas sua interface não é adequada.

Quando se deseja criar e reutilizar uma classe para interoperar com classes ainda não existentes

Quando se deseja utilizar classes e é impraticável criar inúmeras subclasses.

Padrões de Projeto

Padrão Facade

Fachada de la iglesia de la Compañía de Jesús en Segovia

http://commons.wikimedia.org/wiki/File:IGLESIA_COMPA%C3%91IA_FACHADA_ALZADO.jpg

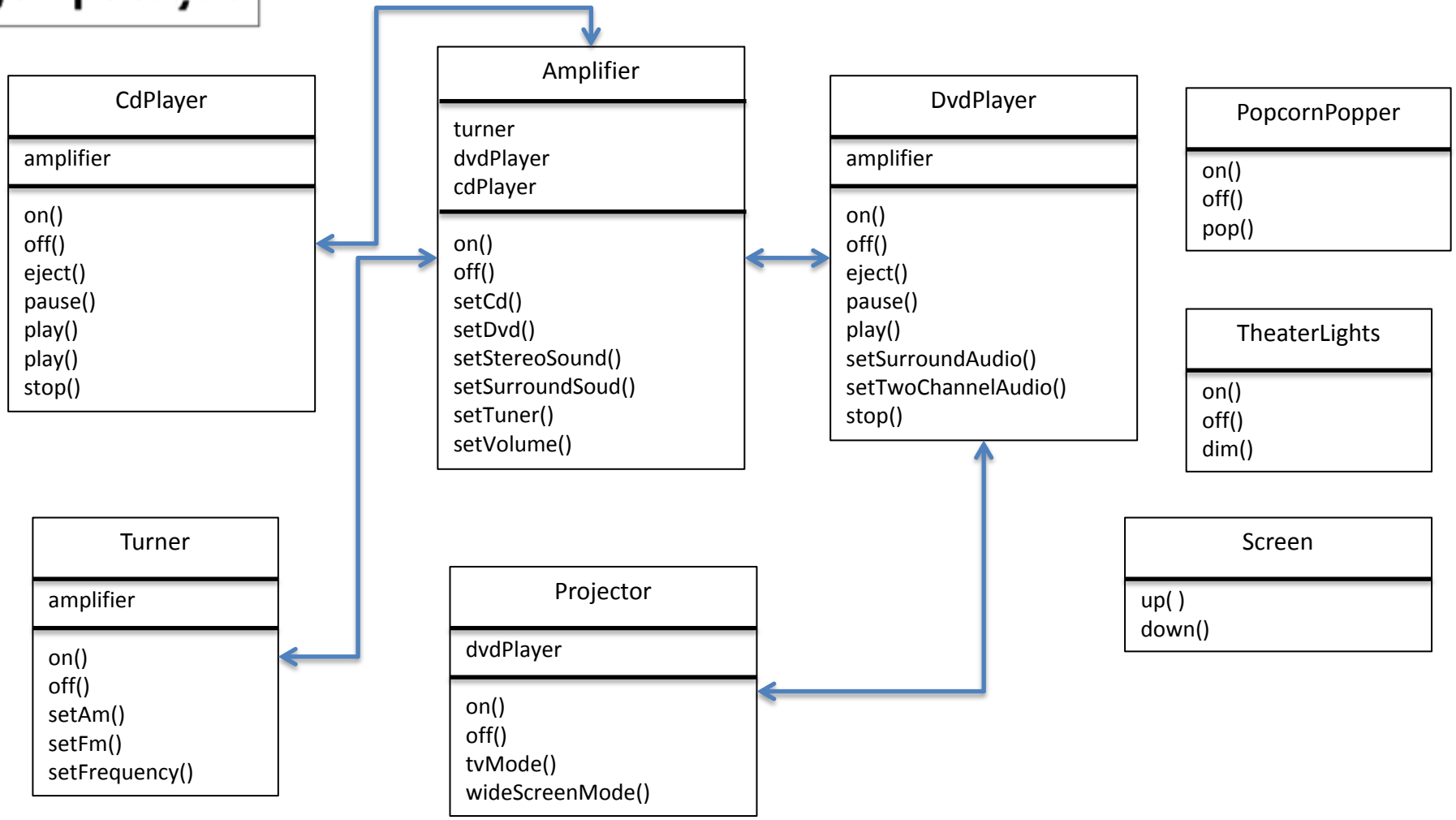
Como lidar com um sistema que depende de um subsistema muito complexo?

Como limitar a dependência do cliente principal as classes do subsistema?

Como combinar as interfaces das classes no subsistema para prover um interface simplificada, com uma funcionalidade específica e delimitada?

Padrões de Projeto

Exemplo de problema



Passos para assistir um filme usando o Home Theater

1. Ligar a pipoqueira (popcornPopper.on( ))
2. Iniciar a preparação das pipocas (popcornPopper.pop( ))
3. Diminuir as luzes (theaterLights.dim( ))
4. Baixar a tela (screen.down( ))
5. Ligar o projetor (projector.on( ))
6. Vincular a entrada do projetor ao Dvd (projector.tvMode( ))
7. Colocar o projetor no modo wide-screen (project.wideScreenMode( ))
8. Ligar o amplificador de som (amplifier.on( ))
9. Configurar a entrada do amplificador para o DVD (amplifier.setDvd( ))
10. Configurar o amplificador para surround sound (amplifier.setSurroundSound()))
11. Configurar o volume do amplificador para médio (amplifier.setSound( ))
12. Ligar o DVD Player (dvd.on( ))
13. Iniciar o filme (dvd.play( ))

Passos para assistir um filme usando o Home Theater

1. Ligar a pipoqueira (popcornPopper.on( ))
2. Iniciar a preparação das pipocas (popcornPopper.pop( ))
3. Diminuir as luzes (theaterLights.dim( ))
4. Baixar a tela (screen.down( ))
5. Ligar o projetor (projector.on( ))
6. Vincular a entrada do projetor ao Dvd (projector.tvMode( ))
7. Colocar o projetor no modo wide-screen (project.wideScreenMode( ))
8. Ligar o amplificador de som (amplifier.on( ))
9. Configurar a entrada do amplificador para o DVD (amplifier.setDvd( ))
10. Configurar o amplificador para surround sound (amplifier.setSurroundSound()))
11. Configurar o volume do amplificador para médio (amplifier.setSound( ))
12. Ligar o DVD Player (dvd.on( ))
13. Iniciar o filme (dvd.play( ))

Padrões de Projeto

Utilização do Padrão Adapter

```
public class HomeTheaterTestDrive {
  public static void main(String[] args) {
   // instantiate components here
   popcornPopper.on( );
   popcornPopper.pop( ) ;
   theaterLights.dim( ) ;
   screen.down( ) ;
   projector.on( ) ;
   projector.tvMode( );
   project.wideScreenMode( ) ;
   amplifier.on( ) ;
   amplifier.setDvd( ) ;
   amplifier.setSurroundSound() ;
   amplifier.setSound( ) ;
   dvd.on( );
   dvd.play("Raiders of the Lost Ark") ;
   }
}
```

Padrões de Projeto

O Princípio do Conhecimento Mínimo

Princípio do conhecimento mínimo – interaja apenas com seus amigos mais próximos.

Qual o significado do princípio do conhecimento mínimo?

Significa que, em um projeto, deve-se ter cuidado com o número de classes que interagem com um dado objeto e como ele interage com tais classes

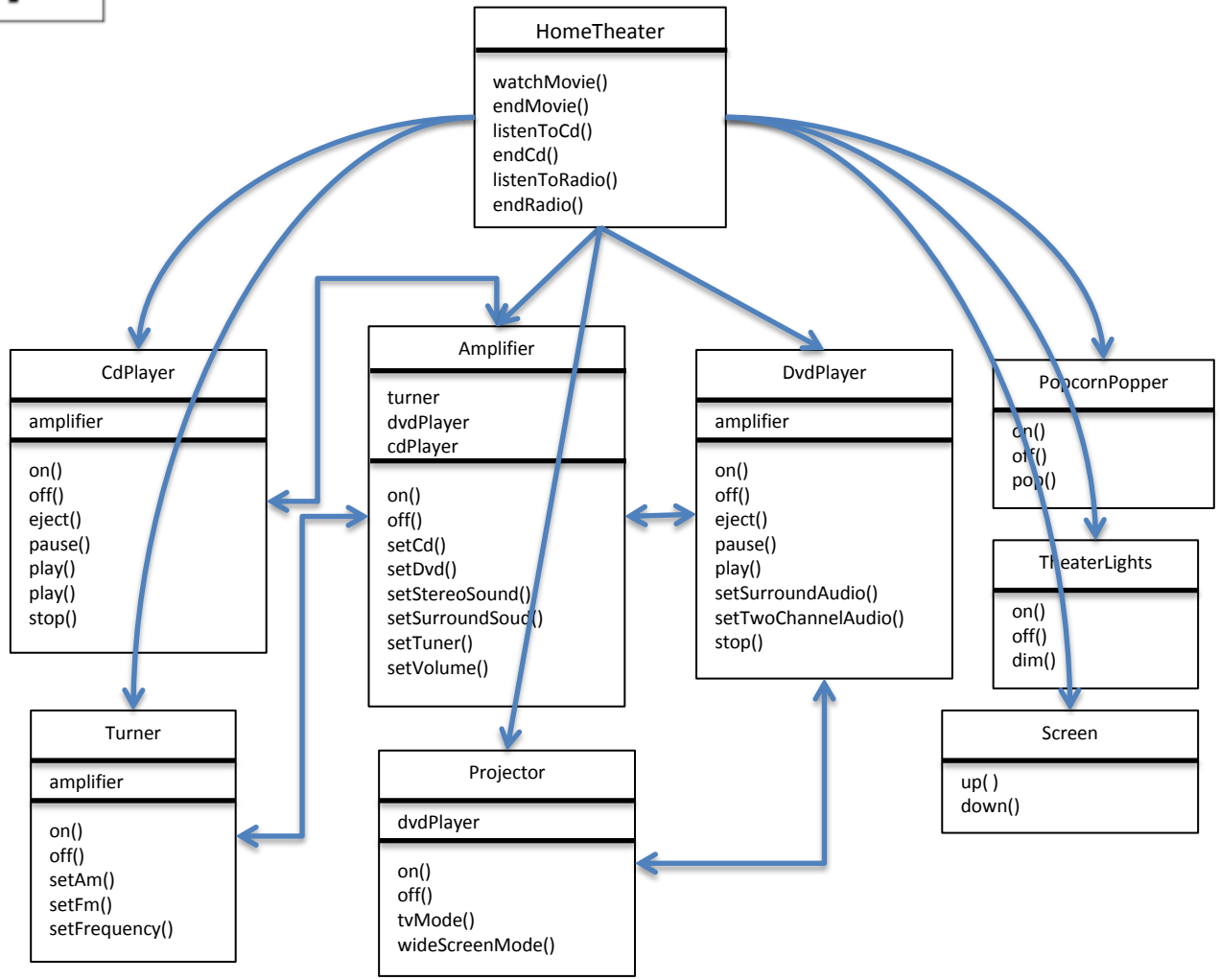
O princípio evita o projeto de classes acopladas a um grande número de outras classes

Qual a dificuldade em controlar todos esses dispositivos para realizar a tarefa desejada?

- Aplicação ou objeto cliente depende de muitas classes diferentes
- As classes utilizadas possuem interfaces mais complexas do que o necessário para a tarefa a ser realizada
- Se o subsistema for modificado, os passos anteriores terão que ser revisitados e possivelmente alterados

Padrões de Projeto

Solução usando o Padrão Facade



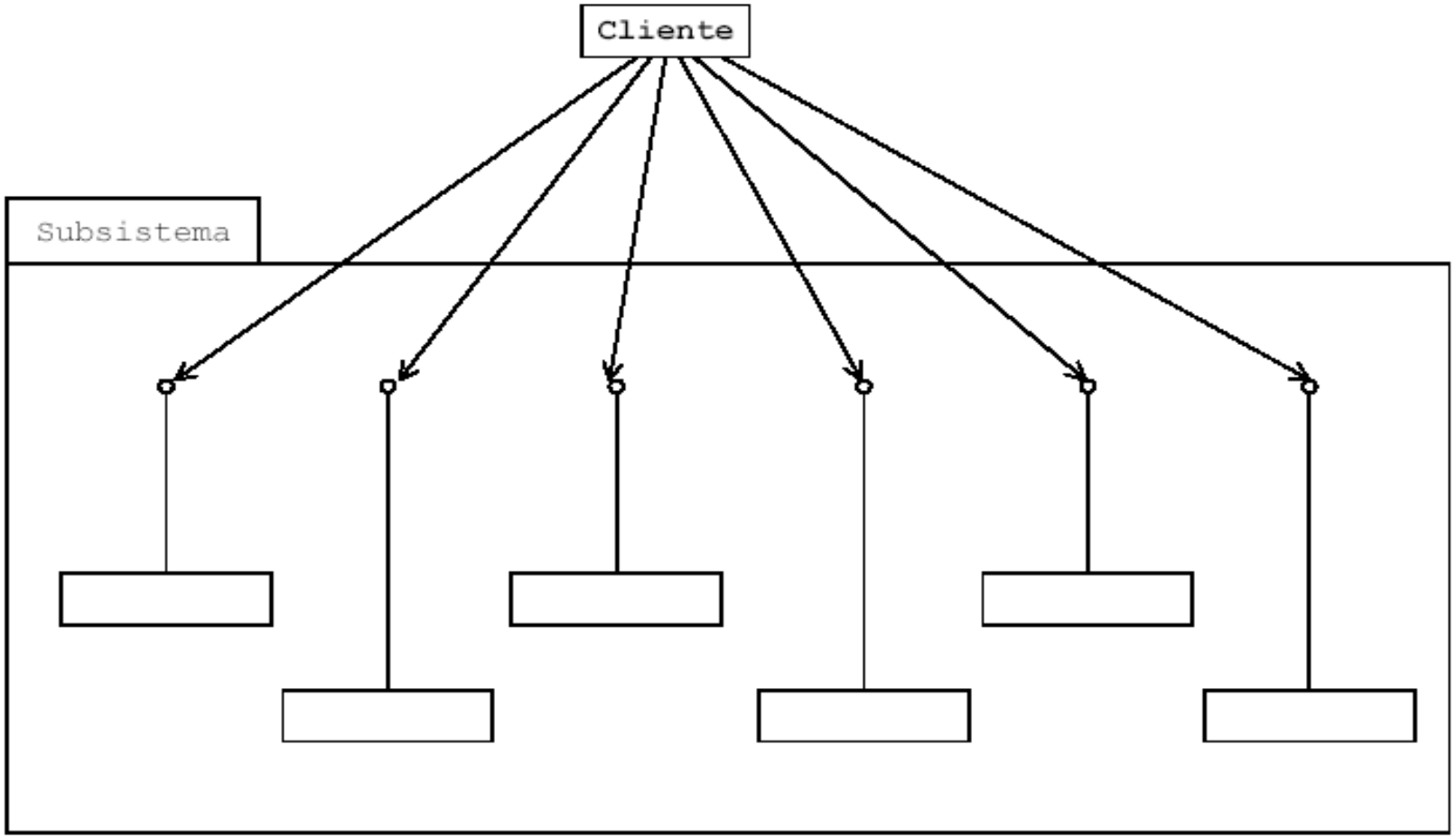
Padrões de Projeto

O Padrão Facade

O **Padrão Facade** oferece uma interface única para um conjunto de interfaces de um subsistema e define uma interface de nível mais elevado que torna o subsistema mais fácil de usar.

Padrões de Projeto

Utilização do Padrão Facade



Padrões de Projeto

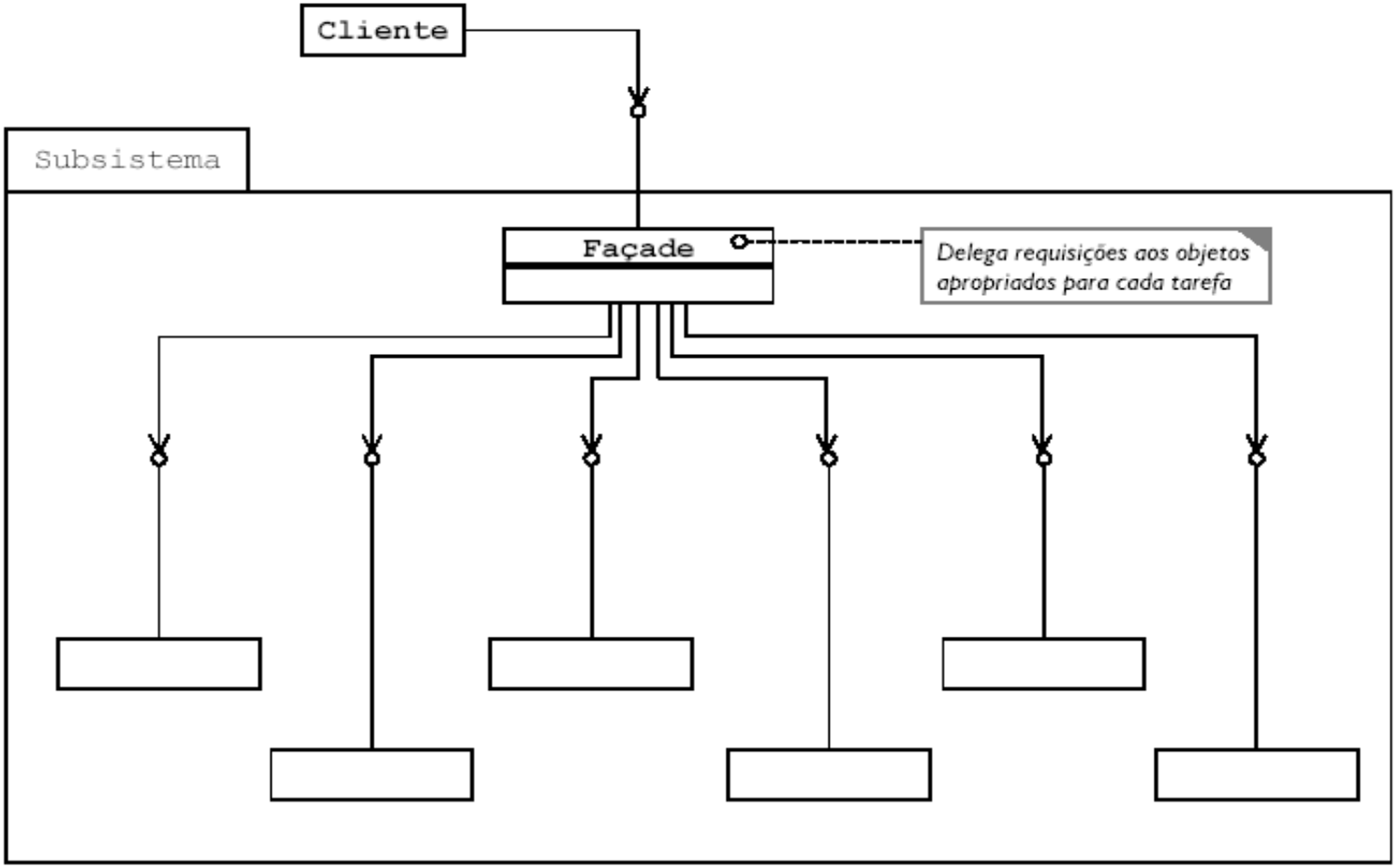
Utilização do Padrão Adapter

```
public class HomeTheaterFacade {
  Amplifier amp; Tuner tuner; DvdPlayer dvd; CdPlayer cd;
  Projector projector; TheaterLights lights; Screen screen; PopcornPopper popper;

  public HomeTheaterFacade(Amplifier amp, Tuner tuner,
  DvdPlayer dvd, CdPlayer cd, Projector projector, Screen screen,
  TheaterLights    lights, PopcornPopper popper) {
    this.amp this.tuner = tuner;
    this.dvd = dvd; this.cd = cd; this.projector = projector;
    this.screen = screen; this.lights = lights; this.popper = popper;
    // other methods here
  }
}
```

Padrões de Projeto

Utilização do Padrão Facade - Estrutura



Padrões de Projeto

Utilização do Padrão Adapter

```
public void watchMovie(String movie) {
  System.out.println("Get ready to watch a movie...");
  popper.on(); popper.pop();
  lights.dim(10); screen.down();
  projector.on(); projector.wideScreenMode();
  amp.on(); amp.setDvd(dvd); amp.setSurroundSound(); amp.setVolume(5);
  dvd.on(); dvd.play(movie);
}

public void endMovie() {
  System.out.println("Shutting movie theater down...");
  popper.off(); lights.on(); screen.up();
  projector.off(); amp.off(); dvd.stop(); dvd.eject(); dvd.off();
}
```

```
public class HomeTheaterTestDrive {
  public static void main(String[] args) {
  // instantiate components here
  HomeTheaterFacade homeTheater =
  new HomeTheaterFacade(amp, tuner, dvd, cd, projector, screen, lights, popper);
  homeTheater.watchMovie("Raiders of the Lost Ark");
  homeTheater.endMovie();
   }
}
```

Facade

- Conhece quais classes do subsistema seriam responsáveis pelo atendimento de uma solicitação.
- Delega solicitações de clientes a objetos apropriados do subsistemas.

Classes de subsistema

- Implementam as funcionalidades do subsistema
- Respondem a solicitações de serviços da Facade
- Não têm conhecimento da Facade

Facade

- Conhece quais classes do subsistema seriam responsáveis pelo atendimento de uma solicitação.
- Delega solicitações de clientes a objetos apropriados do subsistemas.

Classes de subsistema

- Implementam as funcionalidades do subsistema
- Respondem a solicitações de serviços da Facade
- Não têm conhecimento da Facade

Padrões de Projeto

Exercício

Considere um sistema de vendas típico com os seguintes requisitos:

RF1 – O sistema deve permitir o registro de um cliente baseado em seu nome, cpf e endereço em um banco de dados

RF2 - O sistema deve criar e vincular um carrinho de compras ao cliente no ato de registro

RF3 – O sistema deve permitir a consulta de dados do cliente em um BD através do identificador do cliente

RF4 – O sistema deve permitir a consulta de produto em uma banco de dados via o identificador do produto

RF5 – O sistema deve permitir compras de produtos por um cliente

RF6 – O sistema deve processar uma compra feita por um cliente no ato de seu fechamento e armazenar o resultado em um banco de dados

Padrões de Projeto

Exercício

```
class Aplicação {
    ...
    Facade f;
    // Obtem instancia f
    f.registrar("Zé", 123);
    f.comprar(223, 123);
    f.comprar(342, 123);
    f.fecharCompra(123);
    ...
}
```

```
public class Facade {
   BancoDeDados banco = Sistema.obterBanco();
   public void registrar(String nome, int id) {
      Cliente c = Cliente.create(nome, id);
      Carrinho c = Carrinho.create();
      c.adicionarCarrinho();
   }
   public void comprar(int prodID, int clienteID) {
      Cliente c = banco.selectCliente(cliente ID);
      Produto p = banco.selectProduto(prodID) {
      c.getCarrinho().adicionar(p);
    }
    public void fecharCompra(int clienteID) {
       Cliente c = banco.selectCliente(clienteID);
       double valor = c.getCarrinho.getTotal();
       banco.processarPagamento(c, valor);
    }
}
```

```
public class Carrinho {
   static Carrinho create() {...}
   void adicionar(Produto p) {...}
   double getTotal() {...}
}
```

```
public class Produto {
   static Produto create(String nome,
                         int id,double preco) {...}
   double getPreco() {...}
}
```

```
public class Cliente {
 static Cliente create(String nome,
                       int id) {...}
 void adicionarCarrinho(Carrinho c) {...}
 Carrinho getCarrinho() {...}
}
```

```
public class BancoDeDados {
 Cliente selectCliente(int id) {...}
 Produto selectProduto(int id) {...}
 void processarPagamento() {...}
}
```

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books
- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman
- Bob Barr. The Adapter Pattern. http://userpages.umbc.edu/~tarr/dp/lectures/Adapter-2pp.pdf. Acessado em 02/03/2011 - 17:12 h