PADRÕES DE PROJETO DE SOFTWARE

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

1º de junho de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

PADRÕES ESTRUTURAIS

PADRÕES ESTRUTURAIS

São padrões relacionados à composição de classes (com herança) e objetos.

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight
- Private Class Data
- Proxy

ADAPTER

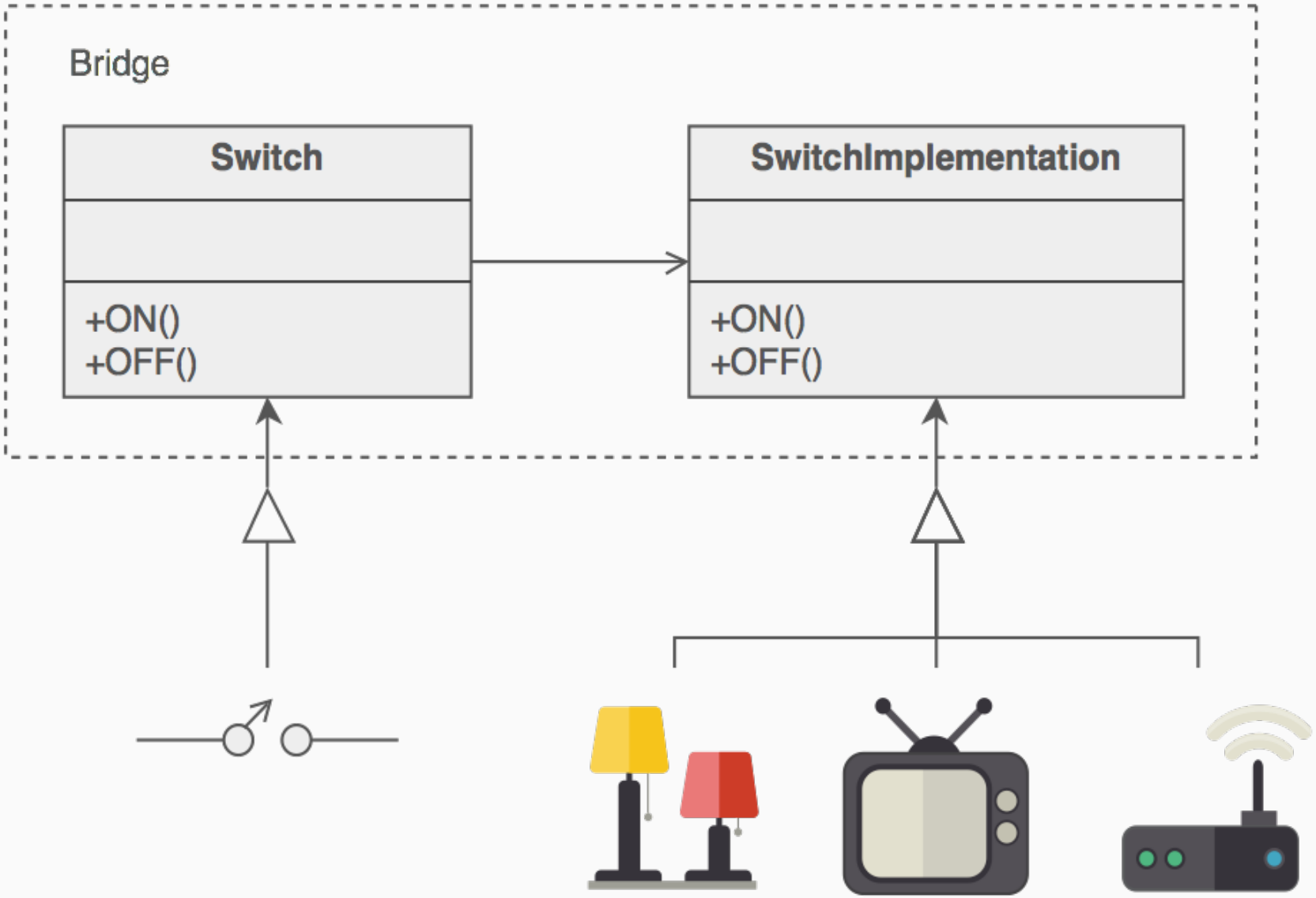
- Reutilização de código antigo é sempre um problema no desenvolvimento de um sistema novo
- Passamos por esse tipo de problema recentemente:

ADAPTER

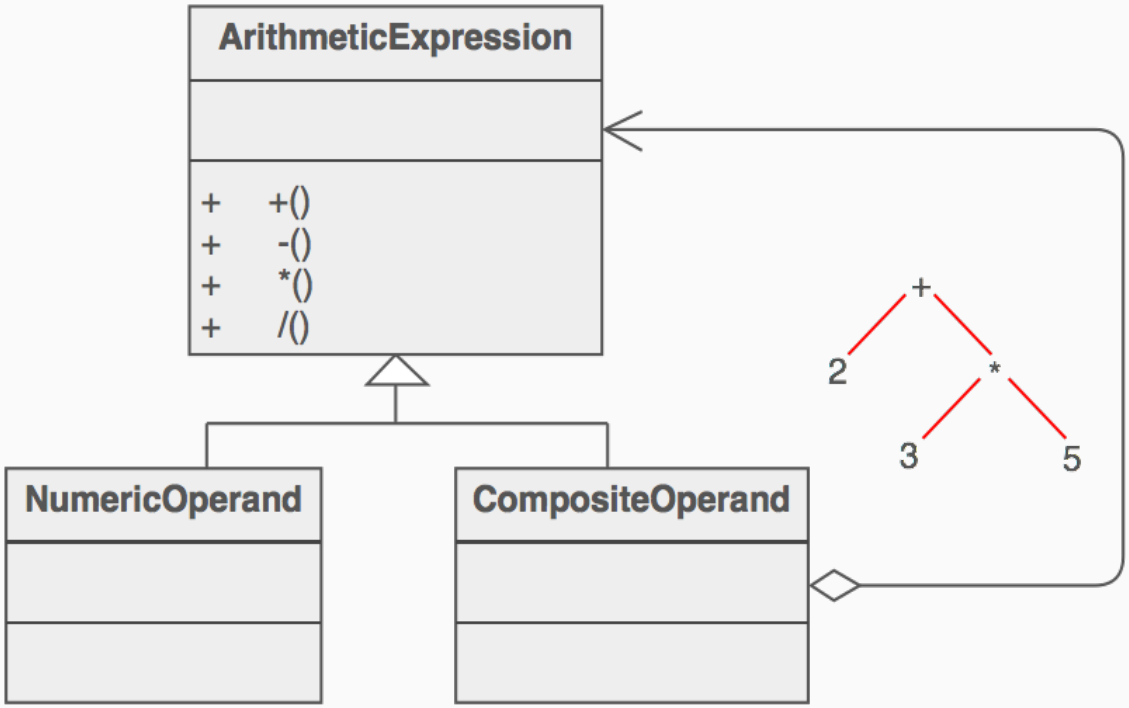
- Reutilização de código antigo é sempre um problema no desenvolvimento de um sistema novo
- Passamos por esse tipo de problema recentemente:

- O padrão Adapter trata do problema de criar uma abstração intermediária que traduza (ou mapeie) um componente antigo para um sistema novo
- O cliente chama os métodos do objeto adaptador, que os redirecionam para o componente legado

BRIDGE

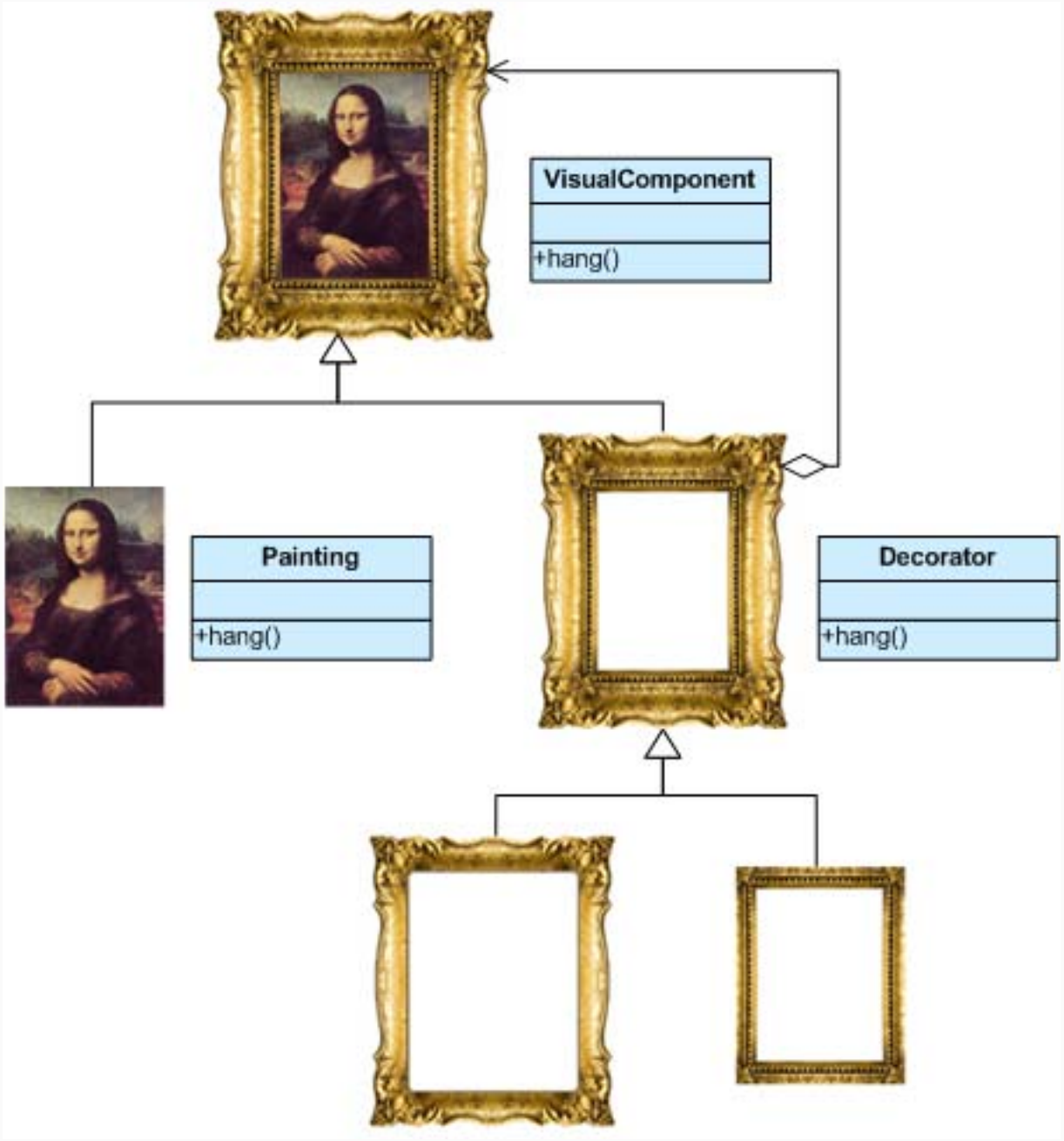


COMPOSITE



- Uma expressão aritmética é um exemplo de objeto composto
- Uma expressão binária tem um operando, um operador (`+` `-` `*` `/`) e um outro operando
- Cada operando pode ser um número ou uma nova expressão

DECORATOR



FAÇADE

- Prover uma interface unificada para um conjunto de interfaces de um subsistema.
- Embrulhar um subsistema complexo em uma interface simples

Problema

Uma parte dos seus clientes precisa de uma interface simplificada para acessar as funcionalidades de um subsistema complicado.

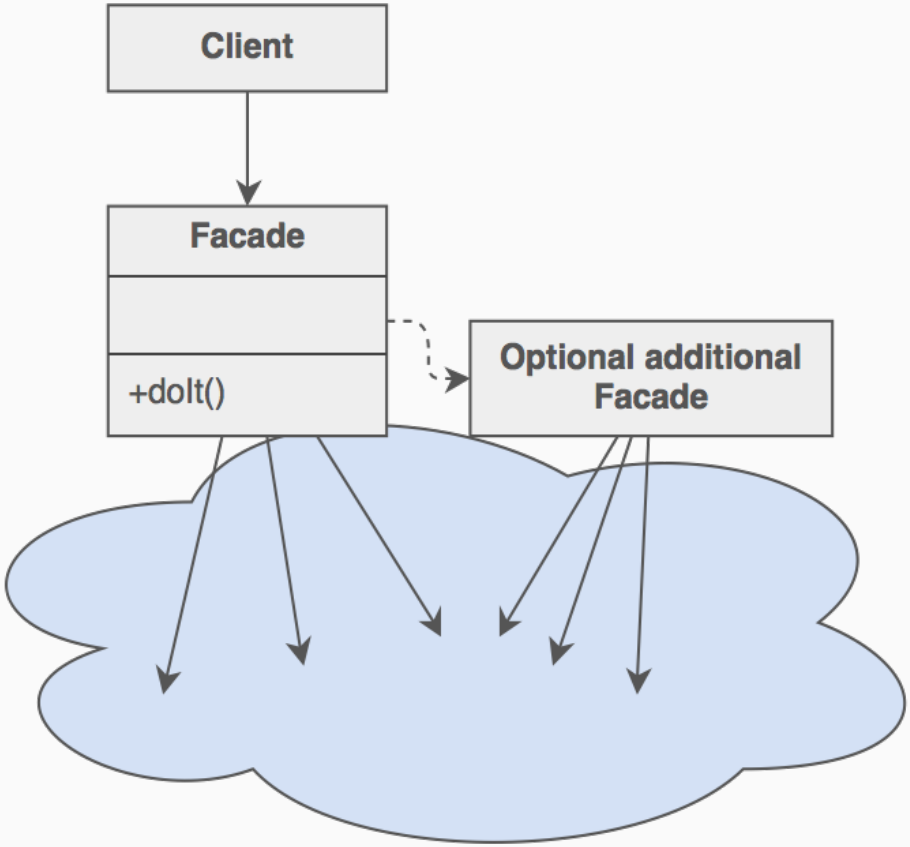
DISCUSSÃO

- Façade trata do problema de encapsular um subsistema complexo dentro de um objeto com interface mais simples
- Reduz a curva de aprendizado para o uso do subsistema
- Promove uma implementação dos clientes que é desacoplada do subsistema
- ... por outro lado, limita o acesso de *power users* a toda funcionalidade e flexibilidade do subsistema

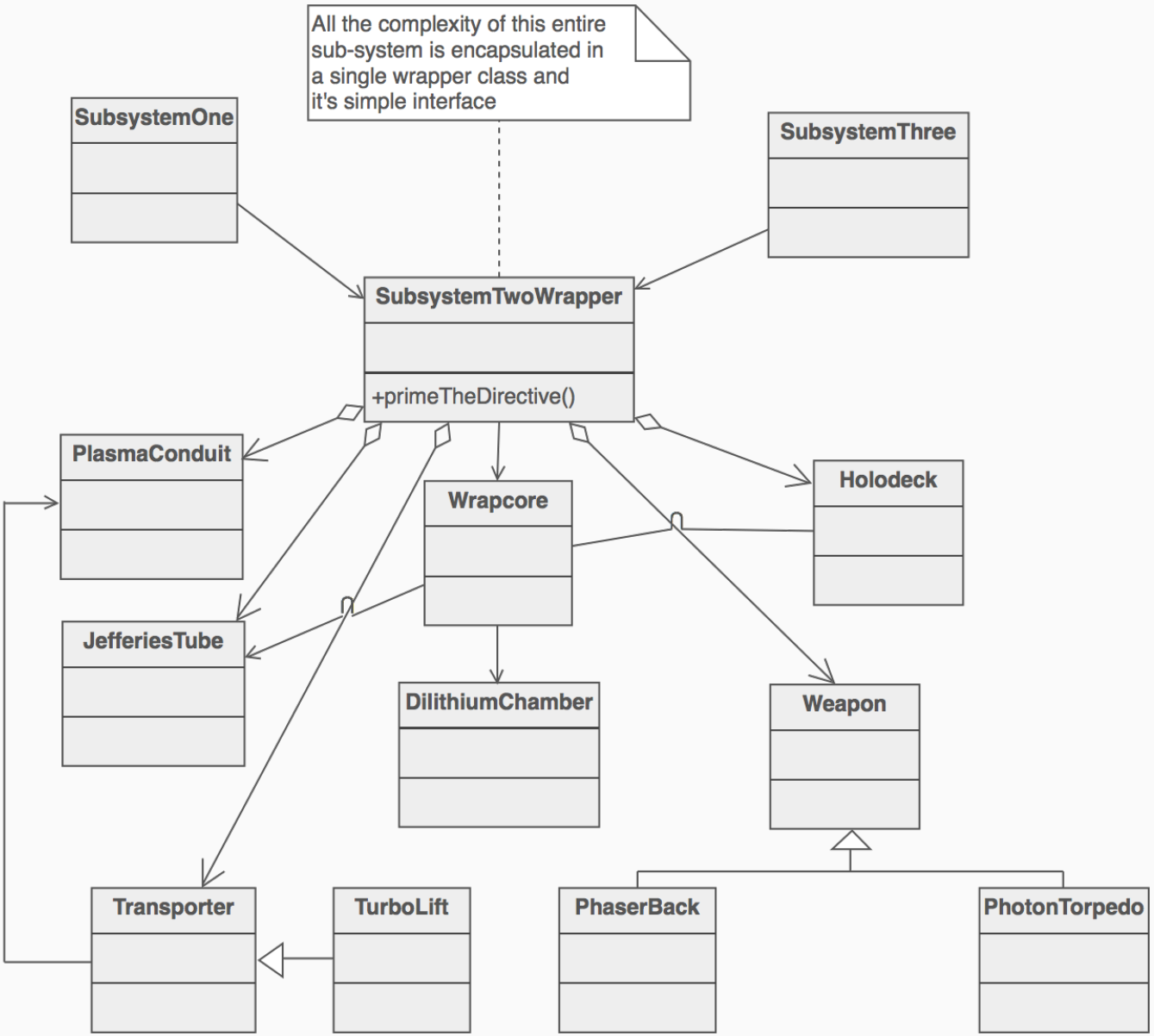
Cuidado

O objeto Façade deve ser visto simplesmente como um facilitador e não um objeto “deus” que sabe e controla tudo

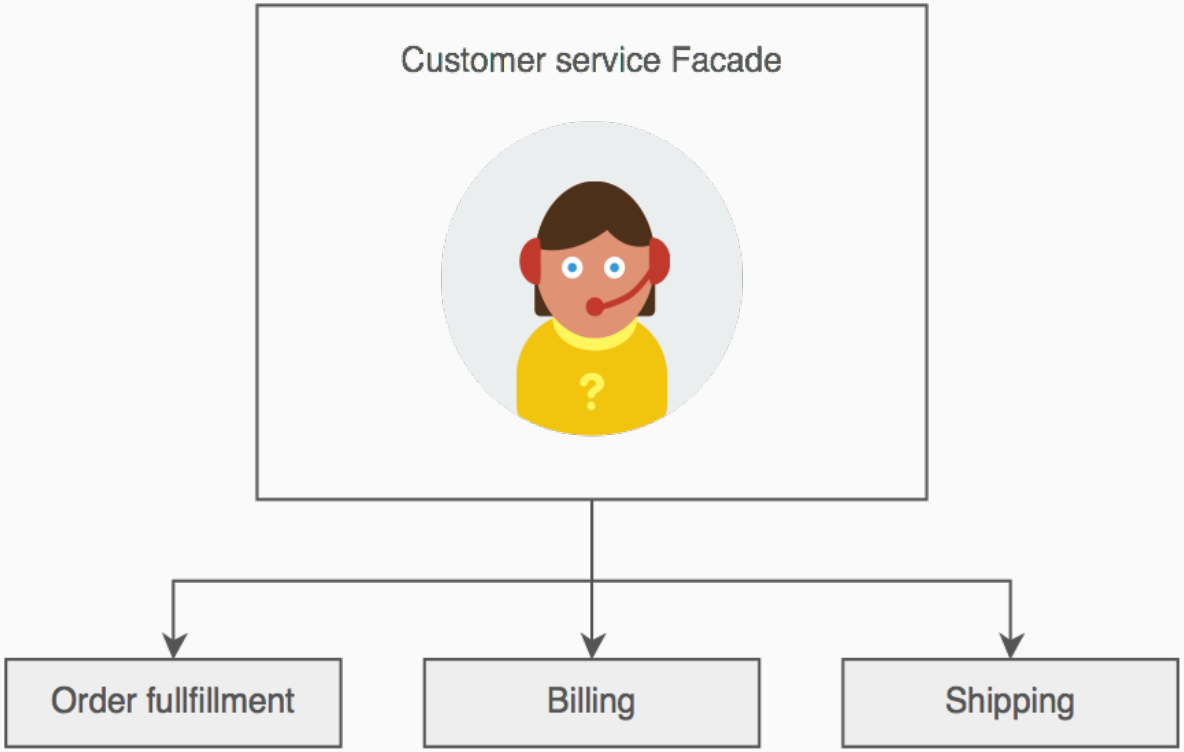
ESTRUTURA



ESTRUTURA


All the complexity of this entire sub-system is encapsulated in a single wrapper class and it's simple interface
SubsystemOne
SubsystemThree
SubsystemTwoWrapper
+primeTheDirective()
PlasmaConduit
Wrapcore
Holodeck
JefferiesTube
DilithiumChamber
Weapon
Transporter
TurboLift
PhaserBack
PhotonTorpedo

EXEMPLO


Customer service Facade
Order fullfillment
Billing
Shipping

LISTA DE VERIFICAÇÃO

1. Identifique uma interface unificada, mais simples, para o seu subsistema ou componente
2. Defina uma classe que irá encapsular o subsistema
3. Capture a complexidade e colaborações do componente na fachada e delegue para os métodos apropriados
4. Faça cliente usar somente (acople-o) a fachada

FLYWEIGHT

- Compartilhar um grande número de pequenos objetos
- Aplicar a estratégia da interface Motif: substituir elementos gráficos pesados por outros mais leves

Problema

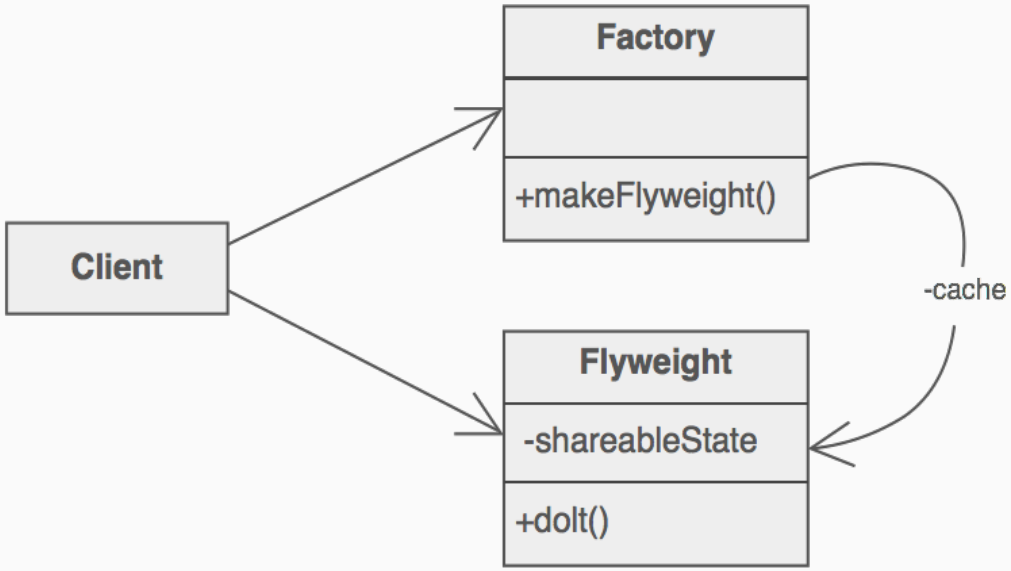
Refinar um projeto orientado a objeto até que a granularidade da representação seja a mais fina possível permite o máximo de flexibilidade, mas pode ter um desempenho e uso de memória inaceitáveis.

DISCUSSÃO

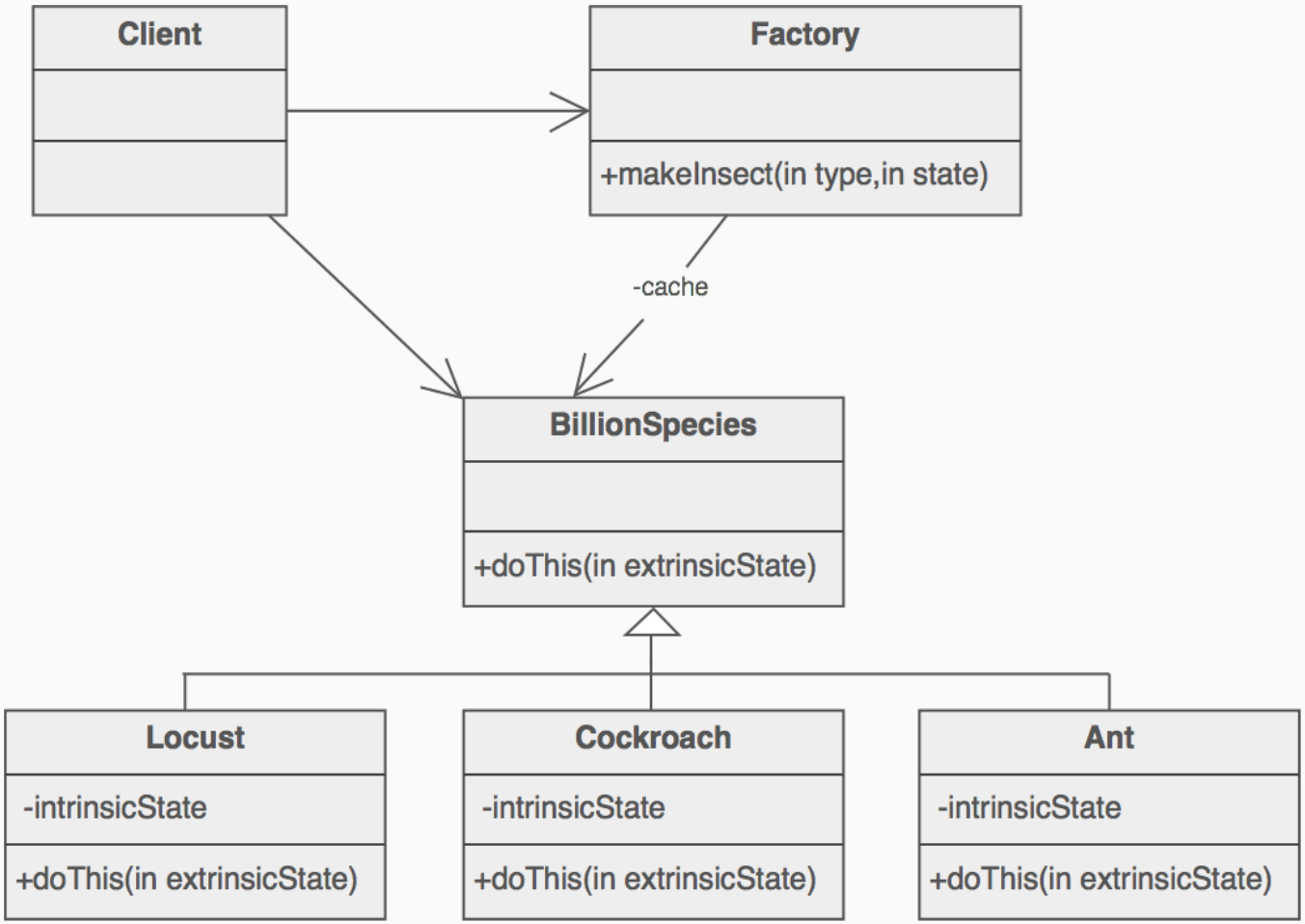
- O padrão Flyweight descreve como compartilhar objetos e permitir a modelagem com uma granularidade mais fina sem um custo proibitivo
- Cada objeto “flyweight” (um objeto “peso-mosca”) é dividido em duas partes: a parte que depende do estado (extrínseca) e a parte independente de estado (intrínseca)
  - A parte intrínseca é armazenada (compartilhada) no objeto Flyweight
  - A parte extrínseca é armazenada no cliente e é passada para o objeto Flyweight como parâmetro
- Nas interfaces gráficas Motif, *gadgets* dependem do gerenciador de leiaute do componente gráfico para funcionar: cada gerenciador realiza o tratamento de eventos dependente de contexto, gerenciamento da área visível e outros recursos para os *gadgets flyweights* e cada *gadget* é responsável apenas pelo seu estado independente de contexto e por seu comportamento

ESTRUTURA

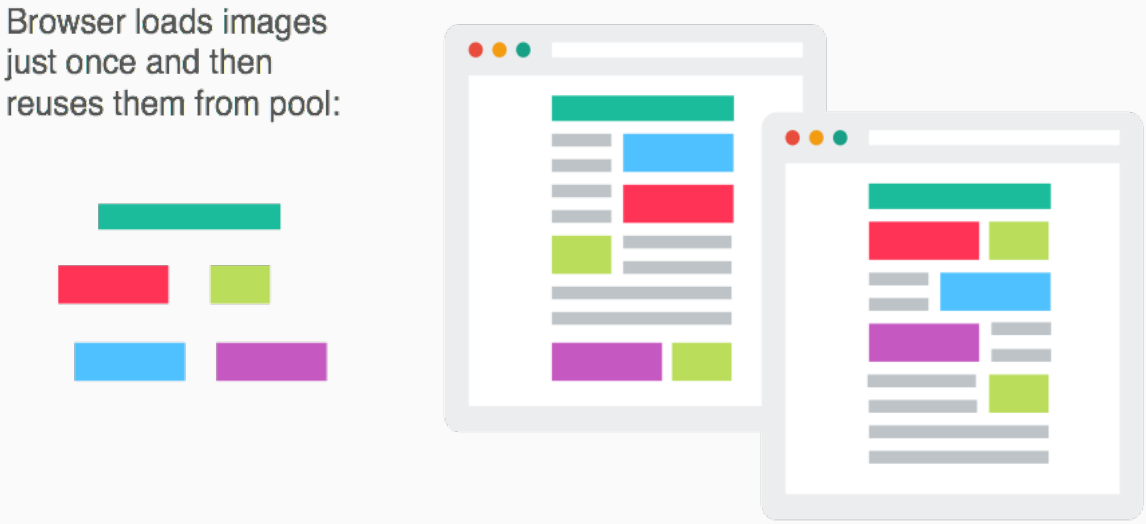
- Flyweights são armazenados em um repositório de uma fábrica
- Os clientes requisitam os objetos da fábrica ao invés de criá-los
- Todos os atributos que tornaria o compartilhamento impossível são fornecidos pelo cliente em toda chamada feita ao objeto flyweight



ESTRUTURA


Client
Factory
+makeInsect(in type,in state)
-cache
BillionSpecies
+doThis(in extrinsicState)
Locust
-intrinsicState
+doThis(in extrinsicState)
Cockroach
-intrinsicState
+doThis(in extrinsicState)
Ant
-intrinsicState
+doThis(in extrinsicState)

EXEMPLO



- navegadores web carregam as imagens só uma vez e as colocam em um cache
- se uma mesma imagem for usada em mais de um lugar, um objeto flyweight (que possui informações como a posição dessa imagem na página) é criado, mas todo o resto é utilizado do cache

LISTA DE VERIFICAÇÃO

1. Tenha certeza de que o sobrecusto (*overhead*) do objeto é realmente um problema e que o cliente pode absorver parte das responsabilidades do objeto
2. Divida o estado da classe em: compartilhável (intrínseco) e não compartilhável (extrínseco)
3. Remova o estado não compartilhável da classe e faça com que sejam parâmetros dos métodos afetados
4. Crie uma Factory que faça o cache e reutilize instâncias existentes
5. Faça o cliente usar a fábrica (e não o `new`)
6. O cliente deve recuperar (ou calcular) o estado não compartilhado e passado para os métodos da classe

PRIVATE CLASS DATA

- Controlar o acesso de escrita aos atributos da classe
- Separar os dados dos métodos que os usam
- Encapsular a inicialização dos dados da classe
- Prover um novo tipo de `final` — “final depois de contruído”

Problema

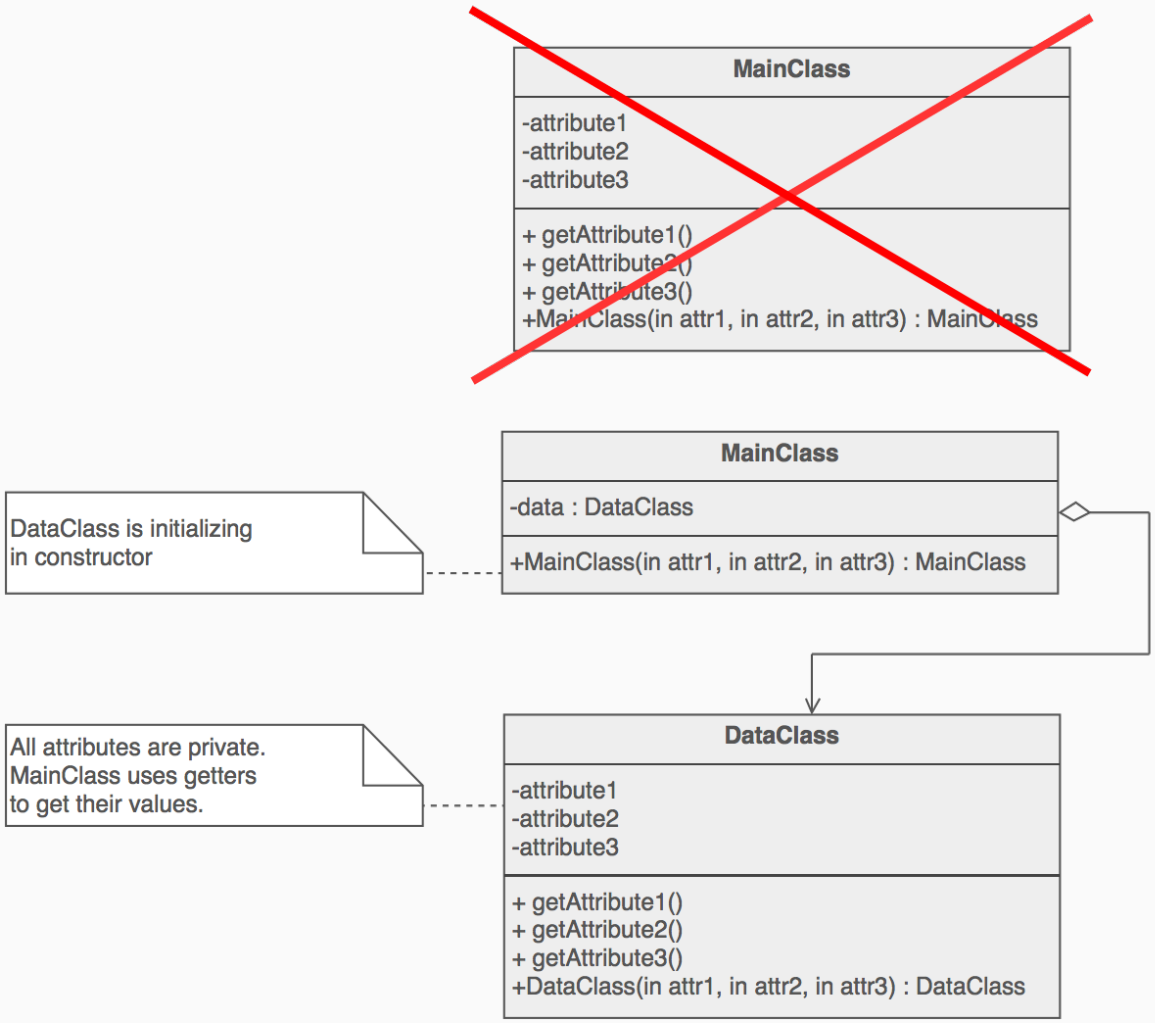
Uma classe pode expor seus atributos (variáveis de classe) quando sua manipulação não é mais desejável (ex: depois da chamada ao construtor).

Uma classe pode ter atributos que só podem ser mudados uma única vez e, portanto, que não podem ser declarados como `final`.

DISCUSSÃO

- O padrão Private Class Data tenta reduzir a exposição dos atributos ao limitar sua visibilidade
- Reduz o número de atributos de classe ao encapsulá-los em um único objeto de dados.
- Permite que a permissão de escrita dos atributos seja revogada

ESTRUTURA


MainClass
-attribute1
-attribute2
-attribute3
+ getAttribute1()
+ getAttribute2()
+ getAttribute3()
+MainClass(in attr1, in attr2, in attr3) : MainClass
MainClass
-data : DataClass
+MainClass(in attr1, in attr2, in attr3) : MainClass
DataClass is initializing
in constructor
All attributes are private.
MainClass uses getters
to get their values.
DataClass
-attribute1
-attribute2
-attribute3
+ getAttribute1()
+ getAttribute2()
+ getAttribute3()
+DataClass(in attr1, in attr2, in attr3) : DataClass

LISTA DE VERIFICAÇÃO

1. Crie a classe de dados. Mova para a nova classe os dados que precisam ser protegidos
2. Na classe principal, crie uma instância da classe de dados
3. Na classe principal, inicialize os dados usando o construtor da classe de dados
4. Exponha cada atributo da classe de dados usando um método *getter*
5. Exponha cada atributo que pode ser modificado usando um método *setter*

PROXY

- Fornecer um substituto para um objeto para que seja possível controlar o seu acesso
- Usar um nível adicional de indireção para permitir acesso distribuído, controlado ou mais inteligente
- Adicione um *wrapper* e delegue o acesso para o componente real para protegê-lo de uma complexidade indevida

Problema

Você precisa de objetos que usam muitos recursos e você não quer que esses objetos sejam instanciados até que eles realmente sejam requisitados por um cliente.

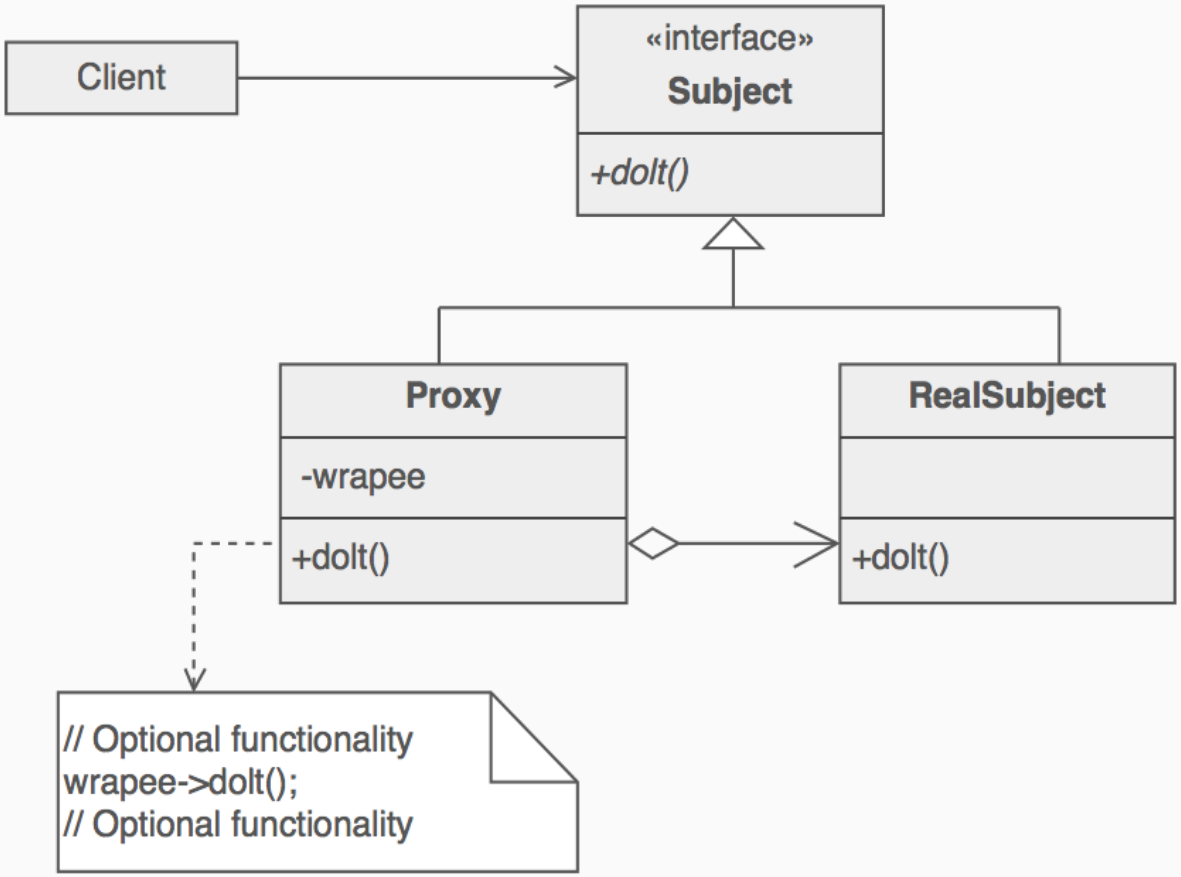
DISCUSSÃO

- Crie um objeto substituto (*proxy*) que instancie o objeto real na primeira vez que o cliente fizer uma requisição ao proxy
- Lembre-se da identidade do objeto real e encaminhe para ele a requisição do cliente
- Todas as requisições subsequentes serão simplesmente encaminhadas para o objeto real

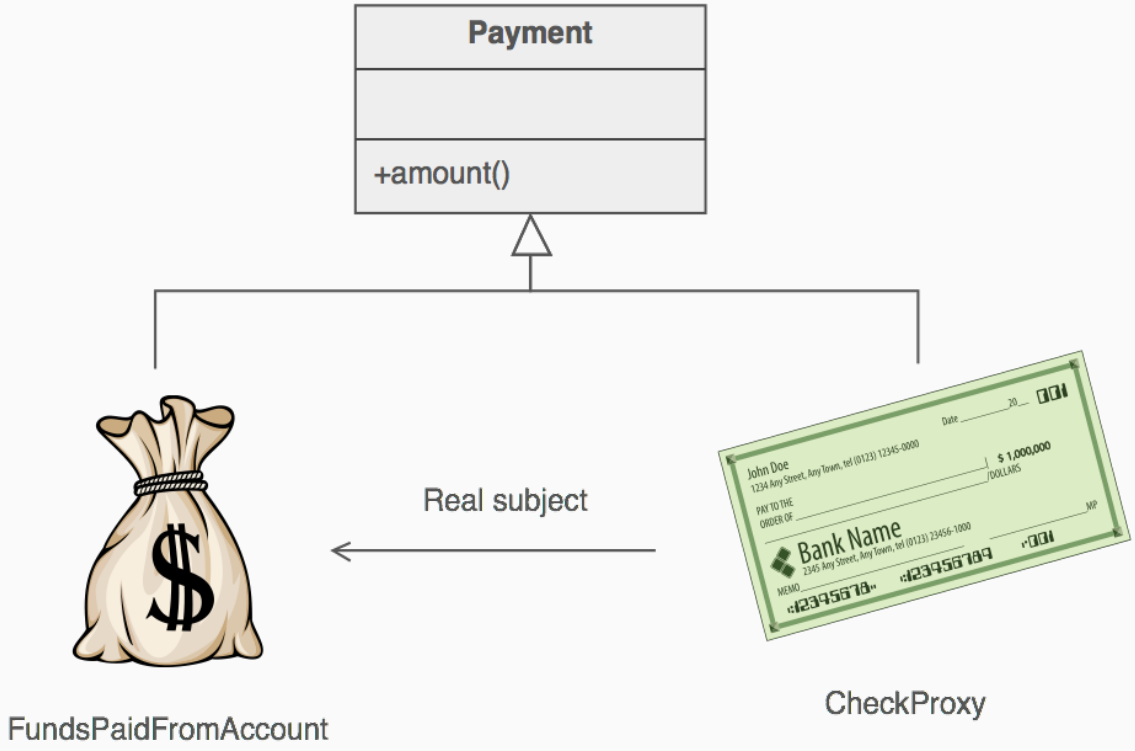
USOS COMUNS DO PADRÃO

1. Proxy como um espaço reservado a objetos “pesados”
2. Proxy remoto como um representante local de um objeto instanciado em um espaço de memória diferente (ex: Java RMI)
3. Proxy protetor para controlar o acesso a um objeto real “sensível” (ex: verificação se o cliente possui as permissões necessárias)
4. Proxy inteligente que adiciona funcionalidades ao acesso ao objeto real:
    - contar o número de referências a um objeto real para que sua memória possa ser liberada automaticamente (*smart pointers*)
    - ler um objeto persistido do disco para a memória quando ele for referenciado
    - verificar se um objeto real está protegido para evitar acesso concorrente ao objeto

ESTRUTURA


Client
«interface»
Subject
+dolt()
Proxy
-wrapee
+dolt()
RealSubject
+dolt()
// Optional functionality
wrapee->dolt();
// Optional functionality

EXEMPLO


Payment
+amount()
Real subject
FundsPaidFromAccount
CheckProxy

EXEMPLO DE IMPLEMENTAÇÃO I

```
import java.io.*;  import java.net.*;

// 5. To support plug-compatibility between
// the wrapper and the target, create an interface
interface SocketInterface {
  String readLine();
  void  writeLine( String str );
  void  dispose();
}

public class ProxyDemo {
  public static void main( String[] args ) {

    // 3. The client deals with the wrapper
    SocketInterface socket = new SocketProxy( "127.0.0.1", 8189,
      args[0].equals("first") ? true : false );

    String  str = null;
    boolean skip = true;
    while (true) {
      if (args[0].equals("second")  &&  skip) {
```

EXEMPLO DE IMPLEMENTAÇÃO II

```
        skip = ! skip;
      }
      else {
        str = socket.readLine();
        System.out.println( "Receive - " + str );  // java ProxyDemo first
        if (str.equals("quit")) break;             // Receive - 123 456
      }                                            // Send ---- 234 567
      System.out.print( "Send ---- " );            // Receive - 345 678
      str = Read.aString();                        //
      socket.writeLine( str );                     // java ProxyDemo second
      if (str.equals("quit")) break;               // Send ---- 123 456
    }                                              // Receive - 234 567
    socket.dispose();                              // Send ---- 345 678
  }
}

class SocketProxy implements SocketInterface {
  // 1. Create a "wrapper" for a remote,
  // or expensive, or sensitive target
  private Socket      socket;
  private BufferedReader in;
  private PrintWriter   out;
```

EXEMPLO DE IMPLEMENTAÇÃO III

```
public SocketProxy( String host, int port, boolean wait ) {
  try {
    if (wait) {
      // 2. Encapsulate the complexity/overhead of the target in the wrapper
      ServerSocket server = new ServerSocket( port );
      socket = server.accept();
    } else
      socket = new Socket( host, port );
      in  = new BufferedReader( new InputStreamReader(
                                        socket.getInputStream()));
      out = new PrintWriter( socket.getOutputStream(), true );
    } catch( IOException e ) {
      e.printStackTrace();
    }
}
public String readLine() {
  String str = null;
  try {
    str = in.readLine();
  } catch( IOException e ) {
    e.printStackTrace();
```

EXEMPLO DE IMPLEMENTAÇÃO IV

```
    }
    return str;
  }
  public void writeLine( String str ) {
    // 4. The wrapper delegates to the target
    out.println( str );
  }
  public void dispose() {
    try {
      socket.close();
    } catch( IOException e ) {
      e.printStackTrace();
    }
  }
}
```

LISTA DE VERIFICAÇÃO

1. Identifique qual aspecto do código é melhor implementado em um *wrapper*
2. Defina uma interface que irá fazer com que o proxy e o objeto original sejam intercambiáveis
3. Considere criar uma Factory que possa encapsular a decisão de usar ou não um proxy
4. A classe *wrapper* mantém um ponteiro para o objeto real e implementa a interface
5. O ponteiro deve ser inicializado pelo construtor ou no primeiro uso
6. Cada método do *wrapper* executa sua ação e então delega para o objeto real

BIBLIOGRAFIA

- The Gang of Four Book, ou GoF: E. Gamma and R. Helm and R. Johnson and J. Vlissides. Design Patterns — Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
- Alexander Shvets. Design patterns explained simply. `https://sourcemaking.com/design_patterns/`