# Adapter

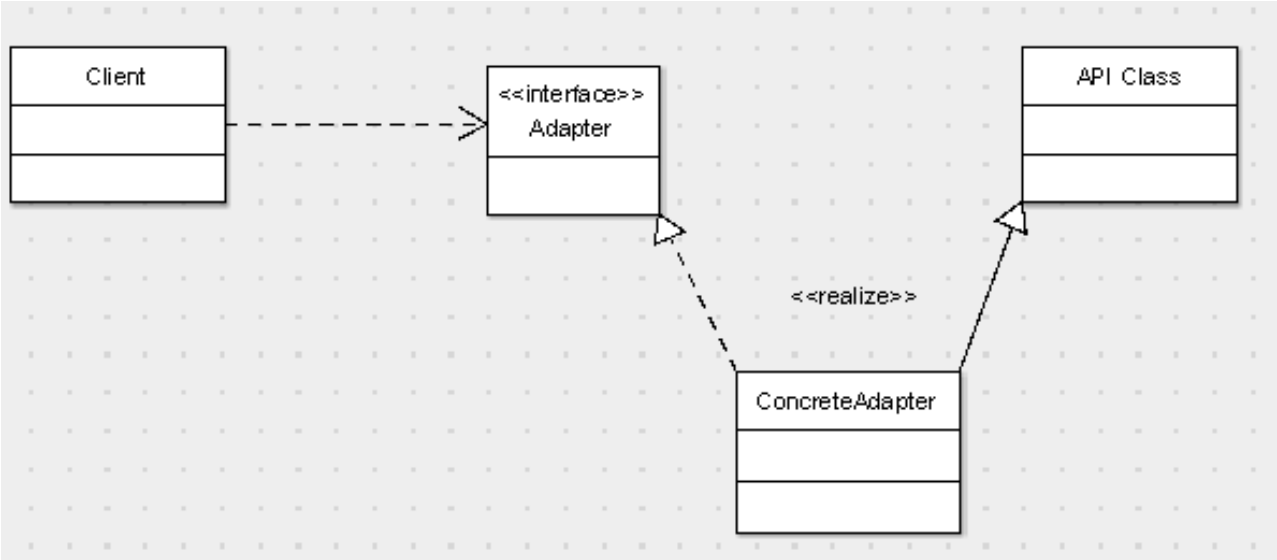
Instituto Federal do Espirito Santo – Campus Colatina
Disciplina: Projeto de Sistemas
Professor: Giovany Frossard Teixeira

# Elementos Essenciais

- **Nome:** *Adapter (Wrapper)*
- **Problema:** O uso de frameworks tem se intensificado nos últimos anos, cada um com suas próprias interfaces de acesso. Programar de forma a ficar dependente de um framework pode ser perigoso (ele pode ser descontinuado, se tornar obsoleto ou podem ser criadas opções melhores).
- **Solução:** Converter a interface de uma classe em outra interface, esperada pelo cliente. O Adapter permite que interfaces incompatíveis trabalhem em conjunto – o que, de outra forma, seria impossível.
- **Consequências:** Independência de uma API (Application Programming Interface) específica. Utilizando este padrão as regras de negócio de um sistema ficam independentes de detalhes de implementação.

# Adapter (escopo classe)

- Ao invés da classe Client acessar diretamente a API Class ela o faz através de uma interface padrão (Adapter).



# Exemplo

**Como utilizar essas APIs sem ficar dependentes de suas assinaturas ?**

- Nesse exemplo vamos utilizar APIs para tratar a questão de carregamento e desenho de imagens:

**API 1**

```
public class OpenGLImage {
        public void glCarregarImagem(String arquivo) {
                System.out.println("Imagem " + arquivo + " carregada.");
        }
        public void glDesenharImagem(int posicaoX, int posicaoY) {
                System.out.println("OpenGL Image desenhada");
        }
}
```

**API 2**

```
public class SDL_Surface {
        public void SDL_CarregarSurface(String arquivo) {
                System.out.println("Imagem " + arquivo + " carregada.");
        }
        public void SDL_DesenharSurface(int largura, int altura, int posicaoX,
                        int posicaoY) {
                System.out.println("SDL_Surface desenhada");
        }
}
```

Fonte: https://brizeno.wordpress.com/category/padroes-de-projeto/adapter/
Acessado em 29/01/2020

# Diagrama de classes

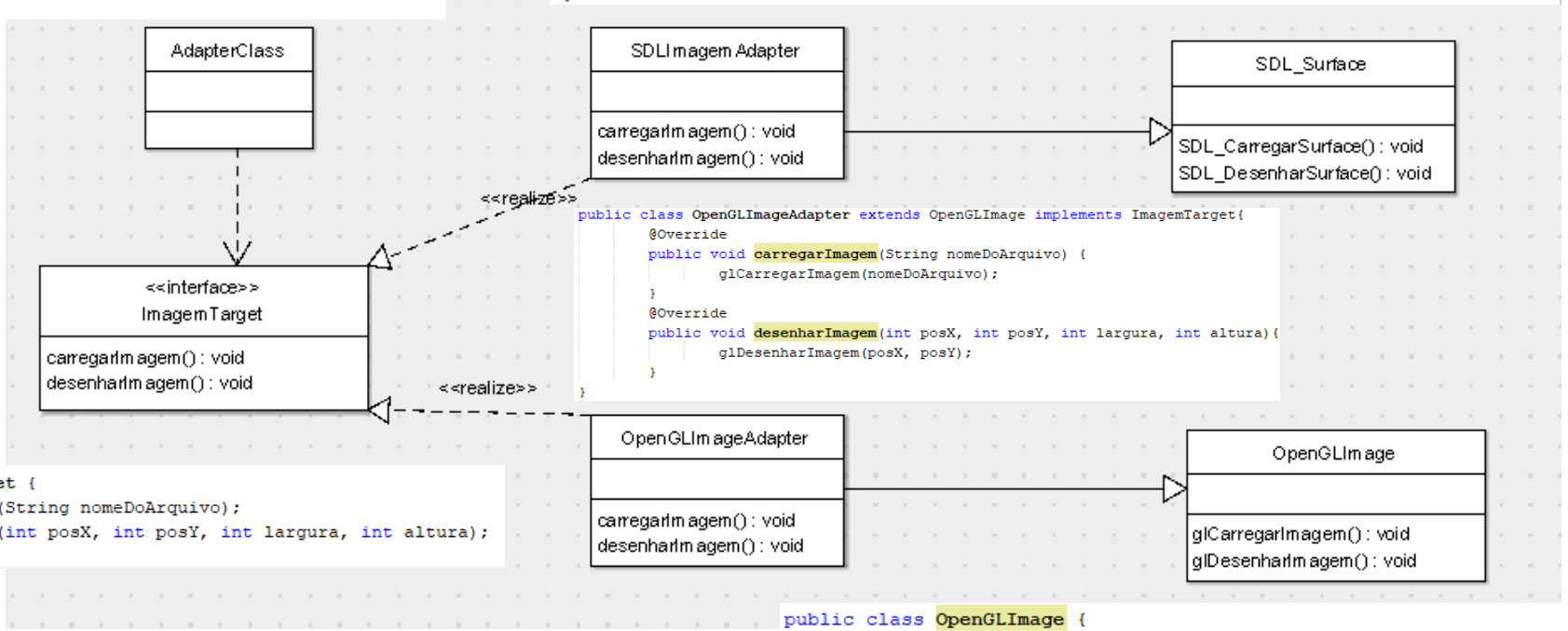
```java
public class AdapterClass {
    public static void main(String[] args) {
        ImagemTarget imagem = new SDLImagemAdapter();
        imagem.carregarImagem("teste.png");
        imagem.desenharImagem(0, 0, 10, 10);

        imagem = new OpenGLImageAdapter();
        imagem.carregarImagem("teste.png");
        imagem.desenharImagem(0, 0, 10, 10);

    }
}
```

```java
public class SDL_Surface {
        public void SDL_CarregarSurface(String arquivo) {
                System.out.println("Imagem " + arquivo + " carregada.");
        }
        public void SDL_DesenharSurface(int largura, int altura, int posicaoX,
                        int posicaoY) {
                System.out.println("SDL_Surface desenhada");
        }
}
```

```java
public class SDLImagemAdapter extends SDL_Surface implements ImagemTarget {
        @Override
        public void carregarImagem(String nomeDoArquivo) {
                SDL_CarregarSurface(nomeDoArquivo);
        }
        @Override
        public void desenharImagem(int posX, int posY, int largura, int altura){
                SDL_DesenharSurface(largura, altura, posX, posY);
        }
}
```

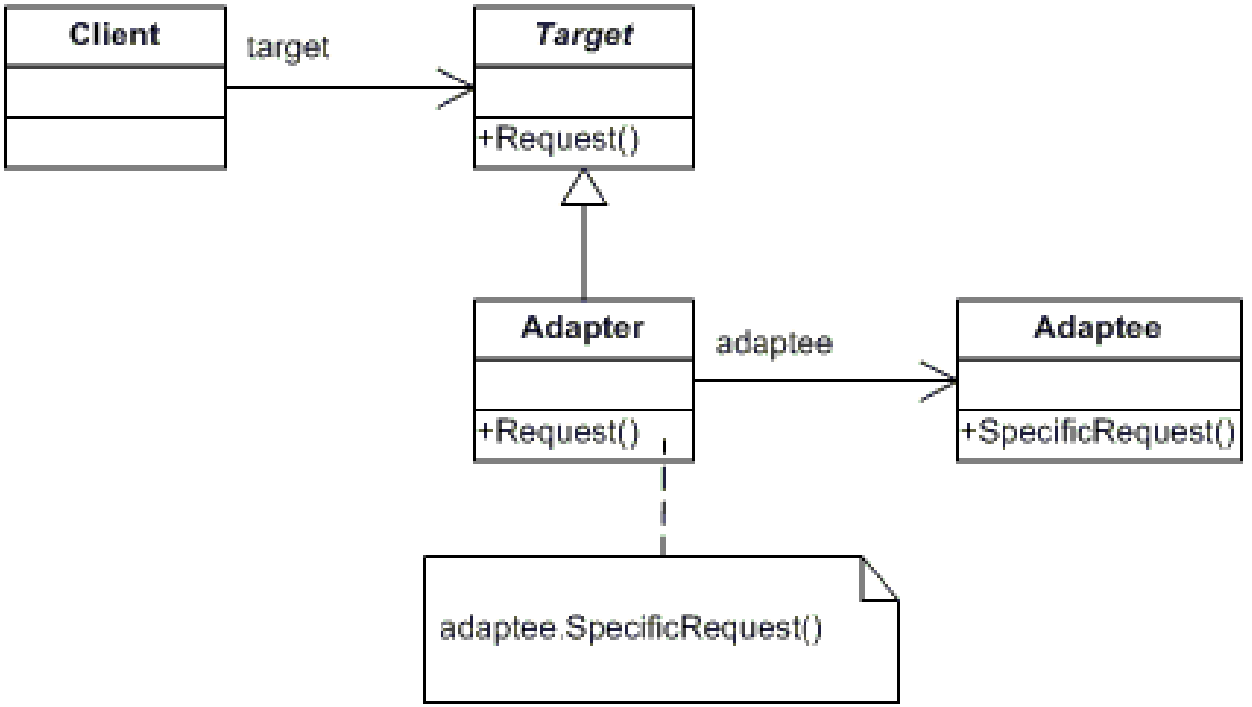



```java
public class OpenGLImageAdapter extends OpenGLImage implements ImagemTarget{
        @Override
        public void carregarImagem(String nomeDoArquivo) {
                glCarregarImagem(nomeDoArquivo);
        }
        @Override
        public void desenharImagem(int posX, int posY, int largura, int altura){
                glDesenharImagem(posX, posY);
        }
}
```

```java
public interface ImagemTarget {
        void carregarImagem(String nomeDoArquivo);
        void desenharImagem(int posX, int posY, int largura, int altura);
}
```

```java
public class OpenGLImage {
        public void glCarregarImagem(String arquivo) {
                System.out.println("Imagem " + arquivo + " carregada.");
        }
        public void glDesenharImagem(int posicaoX, int posicaoY) {
                System.out.println("OpenGL Image desenhada");
        }
}
```

# Adapter (escopo objeto)



# Exemplo

- O Hi-Top Game foi um console lançado pela Milmar em 1991. O console era compatível com o sistema NES (Americano - 72 pinos).
- Havia também o sistema Famicon (Japonês) de 60 pinos.
- Na época foram criados Adaptadores para que videogames de 72 pinos pudessem rodar cartuchos de 60 pinos.
- Esse exemplo será nosso problema para uso de Adapter no escopo de objetos.

# Diagrama de classes

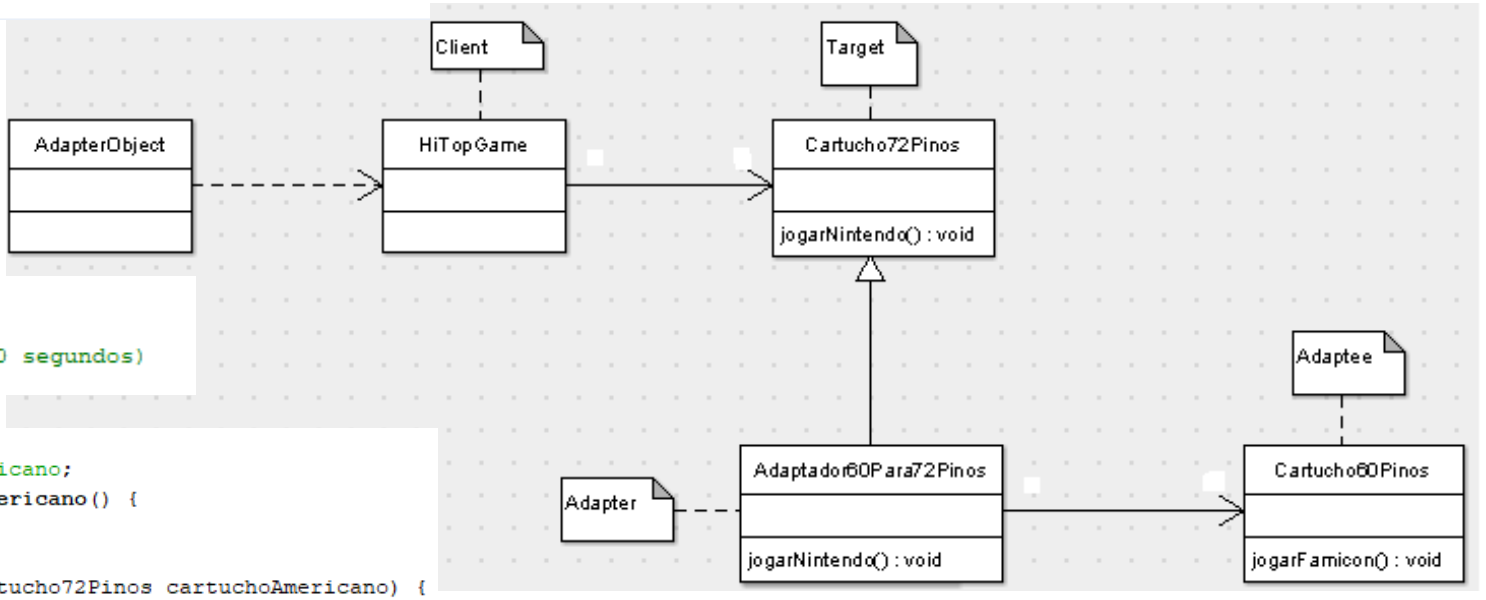
```java
public class AdapterObject {
    public static void main(String[] args) {
        HiTopGame videogame = new HiTopGame();

        Cartucho60Pinos jogo = new Cartucho60Pinos();
        Adaptador60Para72Pinos adaptador = new Adaptador60Para72Pinos(jogo);

        videogame.setCartuchoAmericano(adaptador);
        videogame.jogar();
    }
}
```

```java
public class Cartucho60Pinos {
    public void jogarFamicon(){
        System.out.println("Estou jogando com cartucho japonês");
    }
}
```

```java
public class Cartucho72Pinos {
    public void jogarNintendo(){
        System.out.println("Estou jogando com cartucho americano");
    }
}
```




```
run:
Estou jogando com cartucho japonês
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

```java
public class HiTopGame {
    private Cartucho72Pinos cartuchoAmericano;
    public Cartucho72Pinos getCartuchoAmericano() {
        return cartuchoAmericano;
    }
    public void setCartuchoAmericano(Cartucho72Pinos cartuchoAmericano) {
        this.cartuchoAmericano = cartuchoAmericano;
    }
    public void jogar(){
        this.cartuchoAmericano.jogarNintendo();
    }
}
```

```java
public class Adaptador60Para72Pinos extends Cartucho72Pinos {
    private Cartucho60Pinos cartuchoJapones;
    public Adaptador60Para72Pinos(Cartucho60Pinos cartuchoJapones) {
        this.cartuchoJapones = cartuchoJapones;
    }
    @Override
    public void jogarNintendo() {
        cartuchoJapones.jogarFamicon();
    }
}
```

---


Instituto Federal do Espirito Santo – Campus Colatina
Disciplina: Projeto de Sistemas
Professor: Giovany Frossard Teixeira

# Considerações

- Note que, no Adapter de classe, precisamos definir uma interface (ImagemTarget) com métodos comuns às chamadas concretas, dos Adapters Concretos, necessárias às APIs. Nesse contexto, a dificuldade desse tipo de Adapter é definir o que ficará presente nessa Interface comum aos Adapters Contretos.
- Um outro problema, no Adapter de classe, é que Adapters Concretos não vão conseguir trabalhar com as subclasses das classes adaptadas (note que os Adapters Concretos já são subclasses das classes adaptadas ). Esse problema não ocorre no Adapter de Objeto (qualquer classe que herdasse de Cartucho60Pinos poderia ser utilizada no exemplo).
- O padrão Adapter é um padrão estrutural. Note como ele procura melhor organizar a estrutura das classes e os relacionamentos entre classes e objetos.

# Dúvidas?