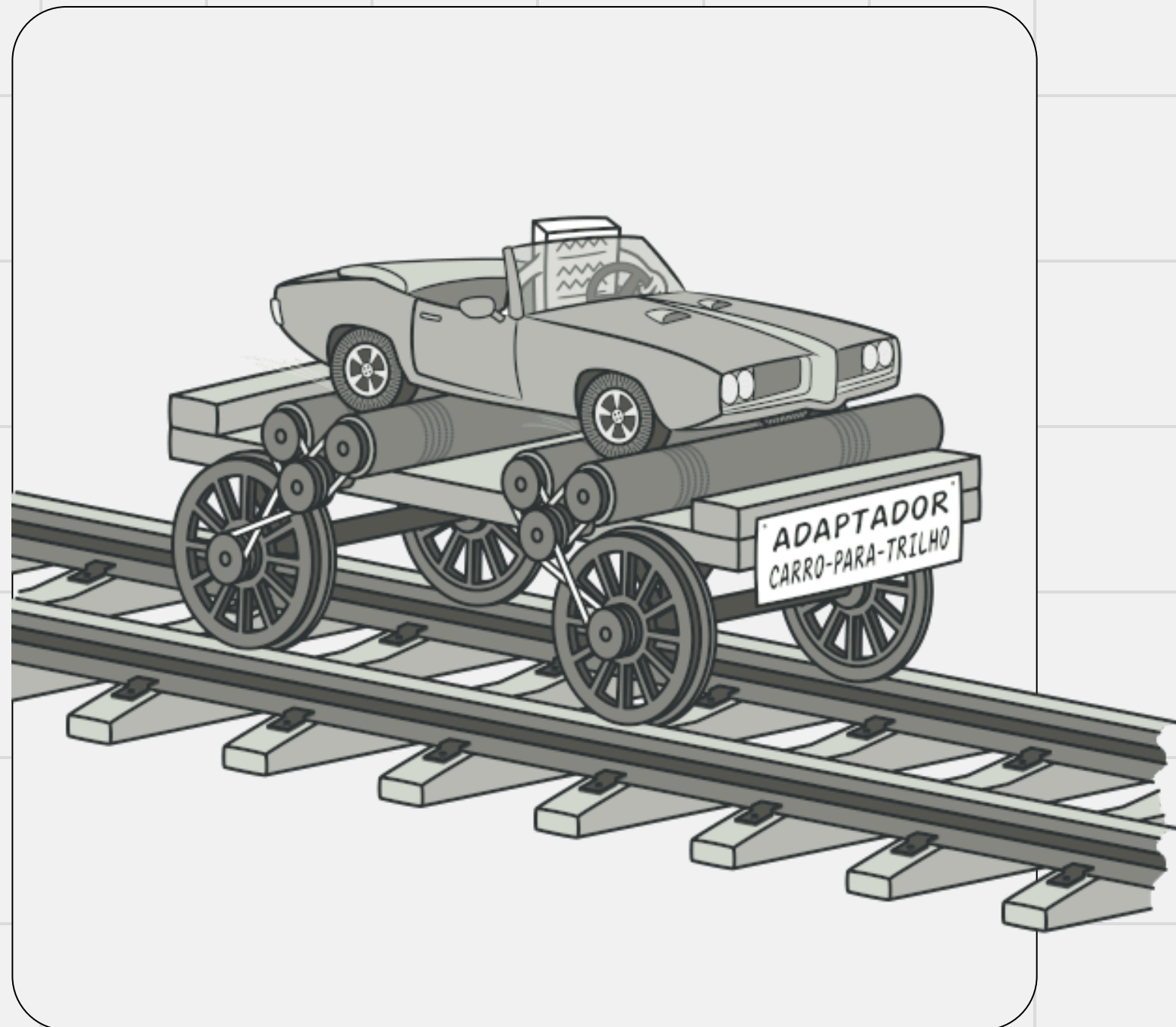
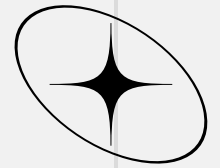


PADRÕES DE PROJETO

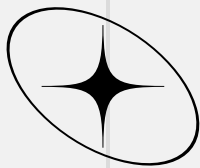
ADAPTER & PROTOTYPE

GABRIEL P.
LUCAS B.



O QUE É O ADAPTER? (GOF)

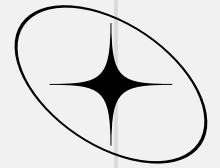
- Padrão de Projeto Estrutural
- Faz interfaces incompatíveis trabalharem juntas.
- Exemplo: como um adaptador de tomada (plugue X tomada incompatível).



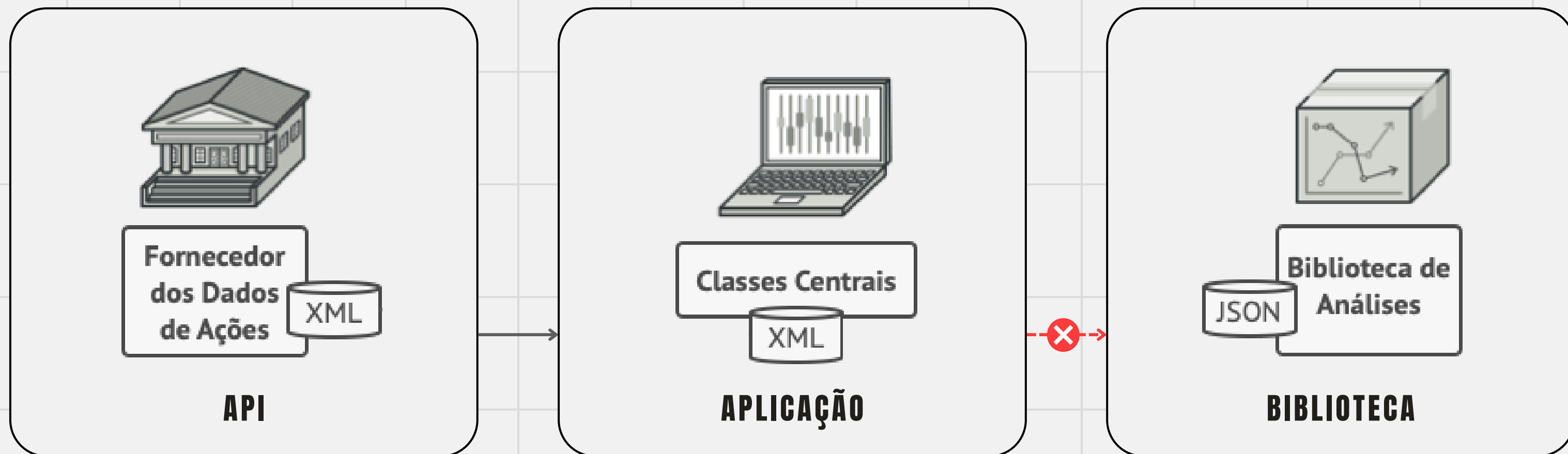
PROBLEMA

Imagine uma aplicação de monitoramento do mercado de ações. Esta, baixa os dados de múltiplas fontes em **formato XML** e então mostra gráficos e diagramas para o usuário.



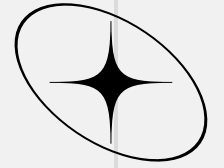


PROBLEMA



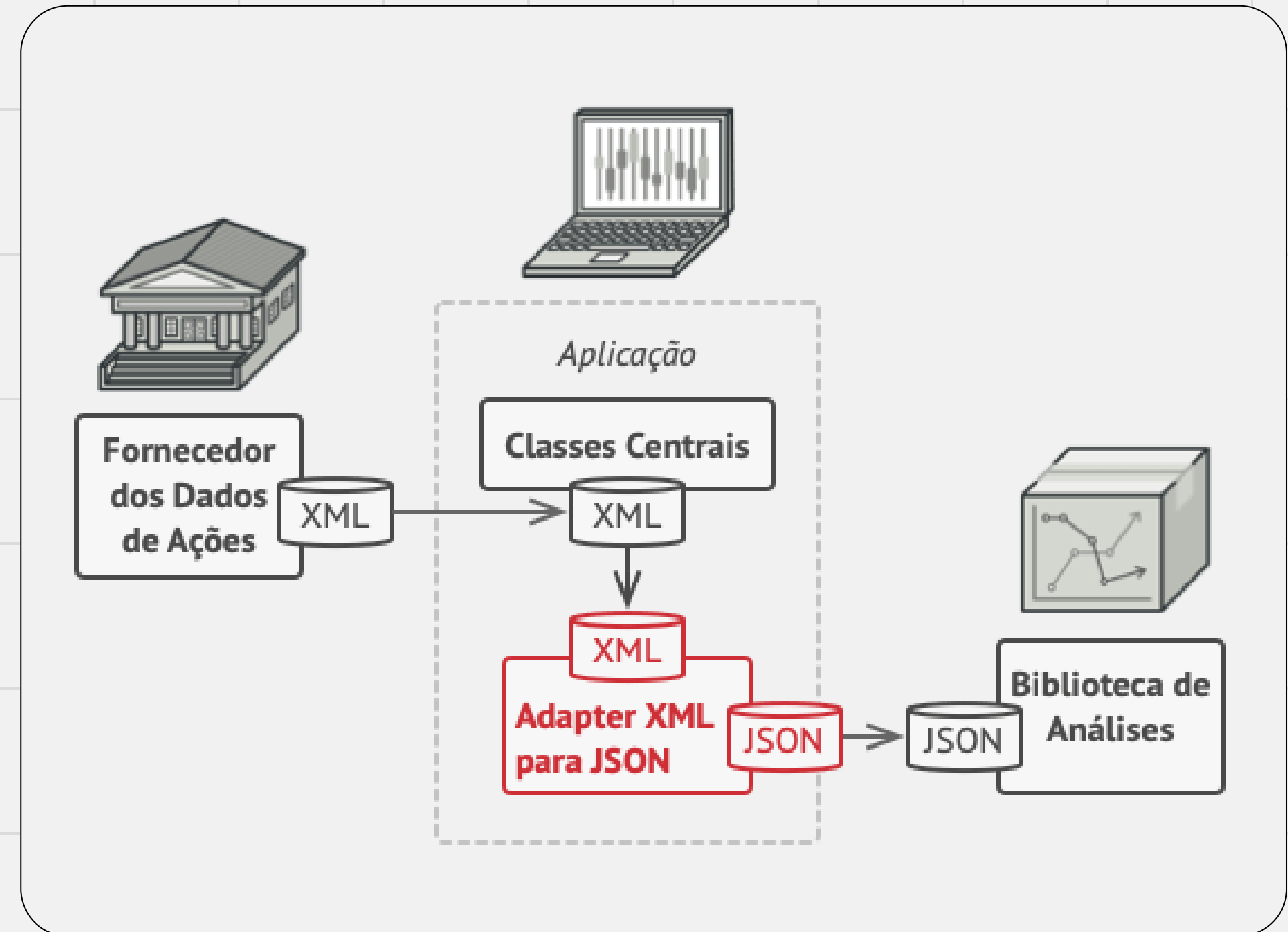
Em algum ponto, você decide **melhorar** a aplicação ao integrar uma **biblioteca** de análise de terceiros. Porém, a biblioteca só trabalha com dados em **formato JSON**.





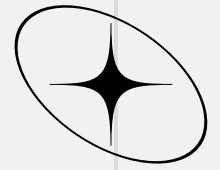
SOLUÇÃO

Um **adaptador** encobre um dos objetos para **esconder** a **complexidade** da conversão acontecendo nos bastidores.



FONTE: DMITRY ZHART



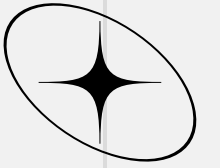


CÓDIGO DE EXEMPLO

CONVERSOR DE KM PARA MILHAS

```
1 ① public interface MedidorMiles { 1 usage 1 implementation
2 ①     ObjMiles getMiles(); no usages
3     }
```

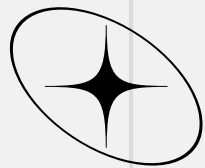
```
1  ▶ public class Main {
2  ▶     public static void main(String[] args) {
3         ObjKm valorKm = new ObjKm(10.0); // exemplo com 10,0 Km
4         MedidorMiles adaptador = new AdapterKmMiles(valorKm);
5
6         ObjMiles valorMiles = adaptador.getMiles();
7
8         System.out.println(valorKm + " km equivale " + valorMiles + " mi");
9         // output: 10.0 km equivale 6.21371 mi
10    }
11 }
```



CÓDIGO DE EXEMPLO

CONVERSOR DE KM PARA MILHAS

```
1 public class AdapterKmMiles implements MedidorMiles { 2 usages
2     private ObjKm km; 2 usages
3
4     public AdapterKmMiles(ObjKm km) { 1 usage
5         this.km = km;
6     }
7
8     @Override 1 usage
9     public ObjMiles getMiles() {
10         double milhas = km.getKm() * 0.621371;
11
12         return new ObjMiles(milhas);
13     }
14 }
15
```



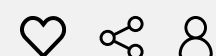
PROS & CONS

Dada a explicação e aplicabilidade do padrão de projeto **Adapter**, é possível identificarmos seus **Prós e Contras**, ao lado:

PRINCÍPIO DA RESPONSABILIDADE ÚNICA



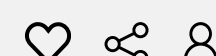
Mantém o código organizado, pois separa a lógica de conversão de interface/dados da lógica principal do negócio.



PRINCÍPIO ABERTO / FECHADO



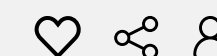
Facilita a manutenção e extensão. É possível adicionar novos adapters sem mexer no código existente, contanto que sigam a interface padrão.

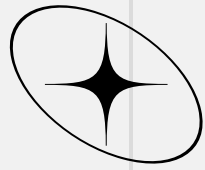


AUMENTA A COMPLEXIDADE



Para adaptar uma nova interface, você precisa de novas classes e interfaces, o que pode deixar o código mais complexo.



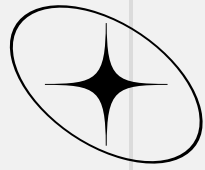


PADRÃO PROTOTYPE (GOF)

A ideia principal desse padrão de projeto é poder realizar a **clonagem de objetos**, copiando instâncias existentes, **reduzindo a sobrecarga** da criação de objetos complexos do zero.



ILUSTRADO POR: DMITRY ZHART



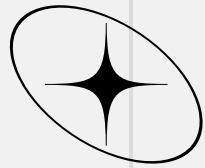
PADRÃO PROTOTYPE (GOF)

A ideia principal desse padrão de projeto é poder realizar a **clonagem de objetos**, copiando instâncias existentes, **reduzindo a sobrecarga** da criação de objetos complexos do zero.

Para isso, é implementada uma **interface** com o método “**clonar**” em todas as classes que suportam clonagem. A partir disso, são criados objetos da classe atual e estes carregam todos os valores de campo do antigo objeto para o novo.



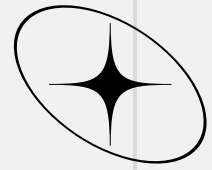
ILUSTRADO POR: DMITRY ZHART



EXEMPLO EM CÓDIGO


```
public interface UsuarioPrototype {  
    UsuarioPrototype clone();  
}
```

```
public class Usuario implements UsuarioPrototype {  
    private String nome;  
    private String email;  
    private String perfil;  
  
    public Usuario(String nome, String email, String perfil) {  
        this.nome = nome;  
        this.email = email;  
        this.perfil = perfil;  
    }  
  
    // Método clone que cria uma cópia do objeto atual  
    @Override  
    public UsuarioPrototype clone() {  
        return new Usuario(this.nome, this.email, this.perfil);  
    }  
  
    // Getters e Setters  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getEmail() {
```

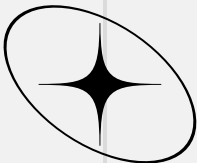


EXEMPLO EM CÓDIGO

```
public class Main {  
    public static void main(String[] args) {  
        // Criação de um usuário protótipo  
        Usuario usuarioAdmin = new Usuario(nome: "Admin Base", email: "admin@empresa.com", perfil: "Administrador");  
  
        // Clonagem do usuário para criar novos perfis similares  
        Usuario usuarioClonado1 = (Usuario) usuarioAdmin.clone();  
        usuarioClonado1.setNome("João Silva");  
        usuarioClonado1.setEmail("joao.silva@empresa.com");  
  
        Usuario usuarioClonado2 = (Usuario) usuarioAdmin.clone();  
        usuarioClonado2.setNome("Maria Oliveira");  
  
        // Impressão dos objetos clonados  
        System.out.println(usuarioAdmin);  
        System.out.println(usuarioClonado1);  
        System.out.println(usuarioClonado2);  
    }  
}
```



```
Usuario{nome='Admin Base', email='admin@empresa.com', perfil='Administrador'}  
Usuario{nome='João Silva', email='joao.silva@empresa.com', perfil='Administrador'}  
Usuario{nome='Maria Oliveira', email='admin@empresa.com', perfil='Administrador'}
```



DESENVOLVIMENTO DE JOGOS

jogos que possuem muitos personagens ou elementos semelhantes podem fazer uso do prototype para clonar instâncias de um modelo base, economizando tempo e recursos.

SISTEMAS DE DESIGN GRÁFICO

clonagem de elementos gráficos com base em configurações já definidas. Por exemplo no Photoshop ou em programas semelhantes, quando se cria uma nova camada ou objeto a partir de uma existente, o sistema está usando a lógica de clonagem que o padrão Prototype oferece.

EXEMPLOS DE USO



REFERÊNCIAS:

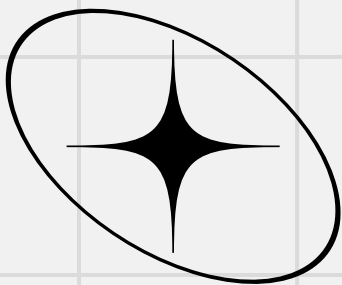
<https://refactoring.guru/pt-br/design-patterns/adapter>

<https://refactoring.guru/pt-br/design-patterns/prototype>

<https://medium.com/@jonesroberto/desing-patterns-parte-8-adapter-21ed67ceb9ed>

<https://www.geeksforgeeks.org/adapter-pattern/>





OBRIGADO

Questões ou comentários?

