Padrão de Projeto GoF – Padrão Facade

O Padrão de Projeto Facade oculta toda a complexidade de uma ou mais classes através de uma Facade (Fachada). A intenção desse Padrão de Projeto é simplificar uma interface.

O Padrão Facade é simples de ser aplicado e que traz grandes benefícios aos projetos é dito como sendo um padrão estrutural e está entre os 23 padrões de projeto do GoF (Gang of Four).

Entende-se por padrão estrutural todo padrão de projeto que trata da associação entre classes e objetos.

Quando usar o padrão Facade?

- É preciso efetuar uma sequência de operações complexas;
- Simplificar a interface com o usuário;
- É preciso fornecer uma interface única e uniforme para as diversas funcionalidades de um subsistema;
- É preciso criar sistemas em camadas. Um Facade provê o ponto de entrada para cada camada (nível) do subsistema;

Benefícios usar o padrão Facade

- Reduz a complexidade de uma API, liberando acesso a métodos de alto nível encapsulando os demais.
- Produz uma interface comum e simplificada.
- Pode encapsular uma ou mais interfaces mal projetadas em uma mais concisa.
- Reduz drasticamente o acoplamento entre as camadas do projeto.

Modelo de implementação

```
public class CarEngineFacade {
    private static int DEFAULT_COOLING_TEMP = 90;
    private static int MAX_ALLOWED_TEMP = 50;
    private FuelInjector fuelInjector = new FuelInjector();
    private AirFlowController airFlowController = new AirFlowController()
;
    private Starter starter = new Starter();
    private CoolingController coolingController = new CoolingController()
;
```

```
    private CatalyticConverter catalyticConverter = new CatalyticConverte
r();

    public void startEngine() {
        fuelInjector.on();
        airFlowController.takeAir();
        fuelInjector.on();
        fuelInjector.inject();
        starter.start();
        coolingController.setTemperatureUpperLimit(DEFAULT_COOLING_TEMP);
        coolingController.run();
        catalyticConverter.on();
    }

    public void stopEngine() {
        fuelInjector.off();
        catalyticConverter.off();
        coolingController.cool(MAX_ALLOWED_TEMP);
        coolingController.stop();
        airFlowController.off();
    }
```

No exemplo acima podemos notar a grande quantidade de classes e métodos envolvidos quando precisamos inicializar o computador.

Toda essa complexidade é exposta ao cliente que poderia chamar todas as classes e cada um dos métodos das classes para realizar a tarefa de ligar o motor. No entanto, ao usar uma Facade encapsulamos essa complexidade oferecendo uma interface simples e unificada ao cliente evitando acoplamento e complexidade. Apenas chamando o método startEngine() ou stopEngine() da classe `CarEngineFacade`, cria-se uma interface simplificada que diz o que ela faz exatamente, sem expor a complexidade envolvida na operação.

Todas essas chamadas que estão no exemplo de implementação do Facade poderiam ser efetuadas uma a uma no cliente, porém isso gera muito acoplamento e complexidade para o cliente, por isso a Facade simplifica e unifica esse conjunto de classes.

Exemplo de uso, se possível ligado ao projeto do semestre

```
public class PublicarVagaAprovada {
    private EnviarNotificacaoSMS enviarNotificacaoSMS = new EnviarNotific
acaoSMS();
        //Envia SMS notificando migrante sobre nova vaga disponivel para
seu perfil
```

```
    private EnviarNotificacaoEmail enviarNotificacaoEmail = new EnviarNot
ificacaoEmail();
        //Envia e-
mai notificando migrante sobre nova vaga disponivel para seu perfil

    private RetornarEmpresaVagaPublicada retornarEmpresaVagaPublicada = n
ew RetornarEmpresaVagaPublicada();
        //Envia e-
mail para responvável da empresa, sobre a sua publicação de vaga aprovada

    public void startEngine() {
        enviarNotificacaoEmail.enviar();
        enviarNotificacaoSMS.enviar();
        retornarEmpresaVagaPublicada.enviar();
    }
```

Correlações com outros padrões

Adapter: o Adapter visa adaptar um conjunto de classes que já existem, para uma outra interface, que é a requerida para outro sistema.

O Padrão Adapter é utilizado quando temos uma classe existente cuja interface não é adequada para as suas necessidades. Além disso, o adaptador consegue mudar a interface de um fornecedor para uma interface que o cliente espera encontrar. O Adapter é um padrão que utiliza boas praticas de orientação a objetos e a sua implementação fica mais complexa de acordo com a complexidade da interface do fornecedor.

Referências:

https://www.devmedia.com.br/padrao-de-projeto-facade-em-java/26476

https://www.devmedia.com.br/padrao-de-projeto-adapter-em-java/26467

https://refactoring.guru/pt-br/design-patterns/facade

http://www.facom.ufu.br/~bacala/ESOF/05b-Padr%C3%B5es%20Gof.pdf