DES-SIS-II - A3 Texto de apoio

Site: EAD Mackenzie
Tema: DESENVOLVIMENTO DE SISTEMAS II {TURMA 03B} 2023/1
Livro: DES-SIS-II - A3 Texto de apoio

Impresso por: FELIPE BALDIM GUERRA .
Data: quinta, 27 abr 2023, 02:55

Descrição

Índice

1. PADRÕES: O QUE SÃO E PARA QUE SERVEM?

O que é a atribuição de responsabilidades no Desenvolvimento de Software?

Já aprendemos que padrões não representam novas formas de trabalho. Ao contrário, os padrões representam boas práticas, já testadas em diferentes contextos e situações. Em poucas palavras: dado um problema, busque um padrão conhecido para resolvê-lo!

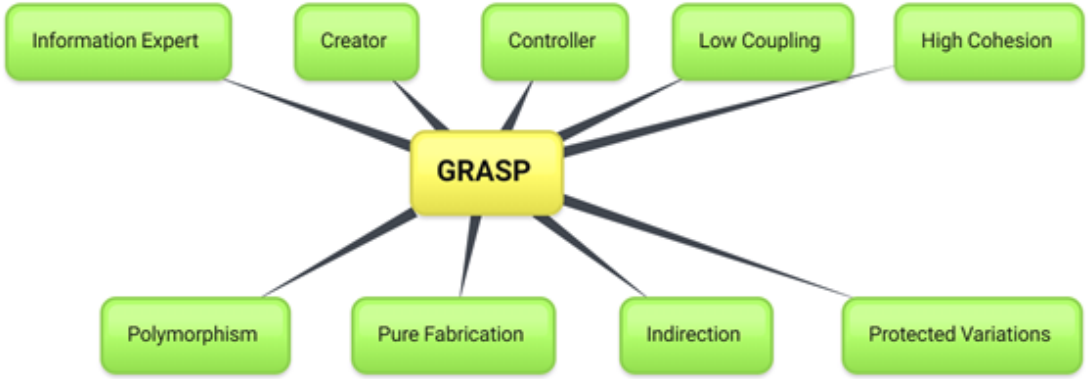
Fonte: Elaborada pelo autor.

Por isso, o uso de um padrão implica na reflexão a respeito das seguintes perguntas:

- O problema que quero resolver, abstraindo os detalhes, tem similaridade com alguma solução já conhecida?
- É possível encontrar algum padrão de desenvolvimento já conhecido que se encaixe como solução para este problema?

- Como avaliar se a aplicação do padrão realmente resolveu o problema?

Há vários tipos e catálogos de padrões. Nesta aula, estudaremos cinco – de um total de nove – padrões GRASP (General Responsibility Assignment Software Patterns – Padrões de Software para Atribuição Geral de Responsabilidades).



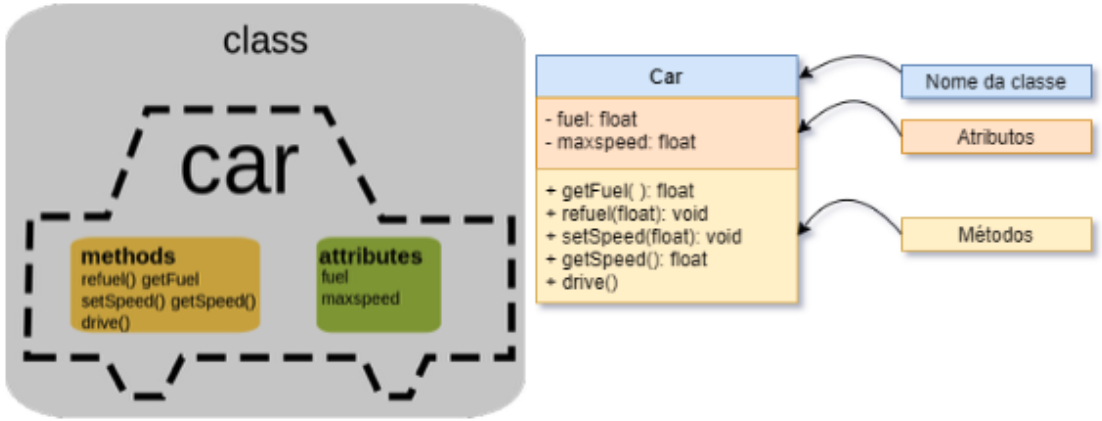
Fonte: Elaborada pelo autor.

Mas você ainda deve estar se perguntando: o que quer dizer esse termo “atribuição de responsabilidades”?

Para responder a essa pergunta, retomaremos um conceito fundamental no Desenvolvimento de Software Orientado a Objetos: a classe.

Sabemos que uma classe representa um conceito, um “molde” para a criação de objetos (suas instâncias). Toda classe pode ser descrita por suas características (geralmente, chamados de atributos) e seu comportamento (geralmente, chamados de métodos). Pois bem, as responsabilidades estão associadas aos comportamentos (métodos).

Por exemplo, pense em uma classe que represente um carro, com dois atributos: combustível (fuel) e velocidade máxima (maxspeed). Para acessar e modificar esses atributos, são definidos quatro métodos – na verdade, são dois pares de métodos do tipo get/set (geralmente, os métodos que expõem o valor de um atributo são nomeados começando com a palavra get; já os métodos que definem um novo valor para o atributo, normalmente são nomeados começando com a palavra set – no exemplo, o método refuel possui a função do método de tipo set para o atributo fuel). Além disso, para o carro não ficar parado na garagem, cria-se o método drive.



Fonte: Wikimedia; Elaborada pelo autor.

A grande pergunta que pode ser feita aqui é: estariam corretas essas atribuições de responsabilidades? Quem deve assumir a responsabilidade de controlar o combustível e fornecer/alterar informações a respeito da velocidade máxima é a classe que representa o carro? Ou outras classes deveriam ser criadas para assumir essas responsabilidades? Com isso em mente, estudaremos alguns padrões de atribuição de responsabilidades que fazem parte de um catálogo conhecido como GRASP, proposto por Craig Larman.

2. PADRÕES GRASP

Information Expert

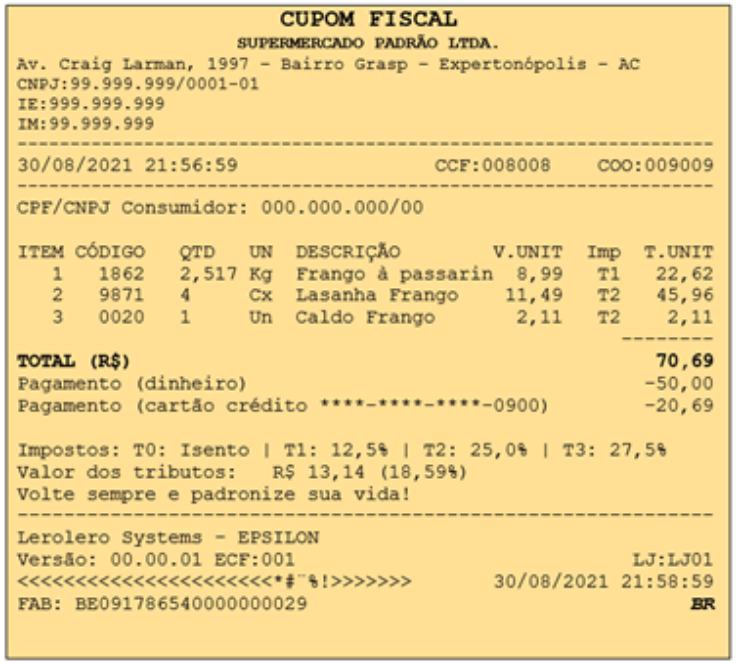
Este padrão tenta responder a uma pergunta essencial no Desenvolvimento Orientado a Objetos: a que classe atribuir responsabilidades?

- Problema: Qual é o princípio básico que deve ser seguido para atribuir responsabilidades a classes?
- Solução: Atribua cada responsabilidade à classe que possui as informações necessárias para cumpri-la.

Em outras palavras: deixe na mão de quem sabe!

Information Expert – Exemplo

Veremos um exemplo em uma compra de supermercado. Observe o cupom fiscal:



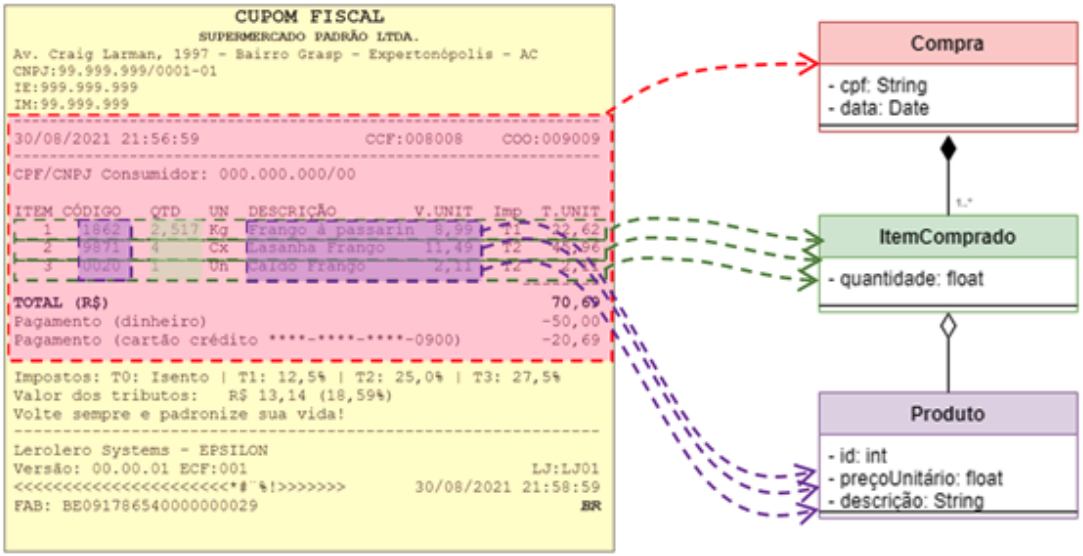
```
                    CUPOM FISCAL
                SUPERMERCADO PADRÃO LTDA.
Av. Craig Larman, 1997 - Bairro Grasp - Expertonópolis - AC
CNPJ:99.999.999/0001-01
IE:999.999.999
IM:99.999.999
------------------------------------------------------------
30/08/2021 21:56:59                CCF:008008    COO:009009
------------------------------------------------------------
CPF/CNPJ Consumidor: 000.000.000/00

ITEM CÓDIGO   QTD   UN  DESCRIÇÃO        V.UNIT  Imp  T.UNIT
   1   1862   2,517 Kg  Frango à passarin  8,99   T1   22,62
   2   9871   4     Cx  Lasanha Frango    11,49   T2   45,96
   3   0020   1     Un  Caldo Frango       2,11   T2    2,11
                                                    --------
TOTAL (R$)                                            70,69
Pagamento (dinheiro)                                 -50,00
Pagamento (cartão crédito ****-****-****-0900)       -20,69

Impostos: T0: Isento | T1: 12,5% | T2: 25,0% | T3: 27,5%
Valor dos tributos:   R$ 13,14 (18,59%)
Volte sempre e padronize sua vida!
------------------------------------------------------------
Lerolero Systems - EPSILON
Versão: 00.00.01 ECF:001                            LJ:LJ01
<<<<<<<<<<<<<<<<<<<<<<*#¨%!>>>>>>>      30/08/2021 21:58:59
FAB: BE091786540000000029                              BR
```

Fonte: Elaborada pelo autor.

Observe, agora, uma primeira modelagem de classes para representar o sistema de vendas do supermercado, contendo, por enquanto, apenas os atributos:



Fonte: Elaborada pelo autor.

> Sim, há muitas outras informações (dados do supermercado, por exemplo) neste cupom fiscal, mas por enquanto não se preocupe com elas.

Note que a modelagem, a princípio, é correta – embora ainda incompleta, pois só tem alguns atributos e nenhum método ainda. Considerando isso, tente responder às seguintes perguntas:

1. Qual classe é responsável por calcular o valor total de cada item da compra (na nota fiscal, a coluna *T.UNIT*)?

2. Qual classe é responsável por calcular o valor total da compra?

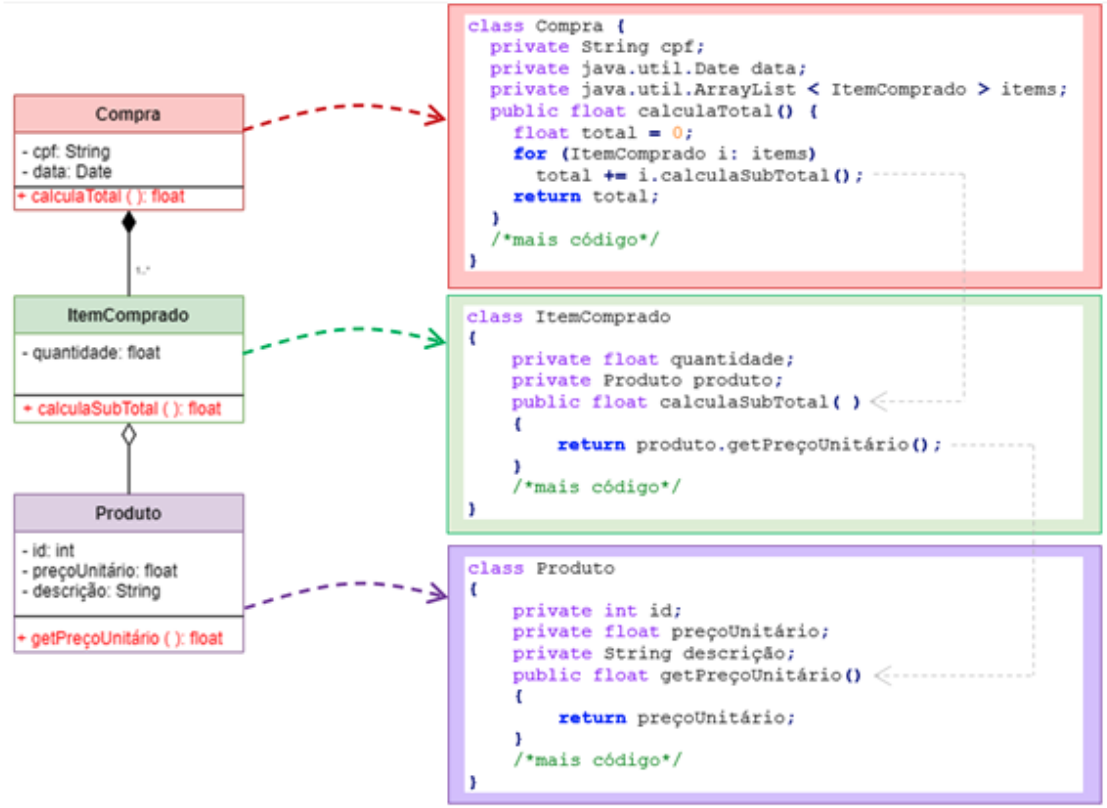
Para responder às perguntas, de acordo com o padrão Information Expert, temos que analisar:

1. Para calcular o subtotal de cada item, precisamos saber a quantidade comprada desse item e o preço unitário.
2. Para calcular o valor total da compra, precisamos somar todos os subtotais.

No item (1): quem tem essas informações são as classes *Produto* (que tem o atributo *preçoUnitário*) e *ItemComprado* (que tem o atributo quantidade). Como *ItemComprado* tem uma agregação à classe *Produto* (ou seja, tem referências aos produtos comprados), é a classe *ItemComprado* que tem acesso a todas as informações necessárias para calcular o subtotal – desde que a classe *Produto* disponibilize o valor do *preçoUnitário* .

No item (2): quem tem a informação de cada subtotal é cada um dos objetos da classe *ItemComprado* . Porém, para somar esses valores, é necessária uma classe que tenha acesso a todas as instâncias de *ItemComprado* – essa classe é a *Compra*.

Veja agora o diagrama remodelado, juntamente com as responsabilidades incluídas, seguindo o padrão Information Expert (e um código em Java correspondente ao lado de cada classe):



Fonte: Elaborada pelo autor.

> Faça, agora, a primeira atividade Praticando, disponível no ambiente virtual.

Creator

Este padrão trata de uma situação muito comum em sistemas orientados a objetos: que classe é responsável pela instanciação (criação de objetos) de outra classe?

- **Problema**: Qual classe deve ser responsável pela criação dos objetos de uma classe (vamos chamá-la de classe A)?

- **Solução**: Em geral, atribua a uma outra classe (vamos chamá-la de classe B) a responsabilidade de criar o objeto A se ao menos uma das seguintes condições acontecer:
- As instâncias de B agregam ou são compostas por instâncias de A.
- As instâncias de B registram instâncias de A em algum serviço de registro.
- As instâncias de B usam diretamente as instâncias de A.
- As instâncias de B têm as informações de inicialização para as instâncias de A e as transmitem para a criação.

Creator – Exemplo

Em nosso exemplo do supermercado, quem seria, por exemplo, responsável por criar um objeto da classe *ItemComprado* ?

Basta olhar o Diagrama de Classes e verificar que a classe Compra é composta de instâncias de *ItemComprado* . Além disso, é a classe *Compra* que possui as informações para a criação de cada objeto *ItemComprado*, já que ela recebe (por meio de alguma UI – User Interface) a identificação do produto (por um código de barras, por exemplo) e a quantidade comprada de cada produto. Assim, um novo método *criaNovoItem*( ) é inserido na classe Compra, com esses dois parâmetros – estes serão usados pelo construtor de *ItemComprado*, como pode ser visto nas classes Java em seguida (novos códigos marcados em amarelo):

Fonte: Elaborada pelo autor.

```
class Compra
{
    private String cpf;
    private java.util.Date data;
    private java.util.ArrayList<ItemComprado> items;
    public float calculaTotal( ) {
        float total=0;
        for (ItemComprado i: items)
         total += i.calculaSubTotal();
        return total;
    }
    public ItemComprado criaNovoItem(int id, float q) {
        return new ItemComprado(id,q);
    }
    /*mais código*/
}
```

```
class ItemComprado
{
    private float quantidade;
    private Produto produto;
    public ItemComprado(int id, float q) {
        this.quantidade = q;
        produto = /*código para recuperar o produto pelo ID*/;
    }
    public float calculaSubTotal( )
    {
        return produto.getPreçoUnitário();
    }
    /*mais código*/
}
```

Note um detalhe importante: as condições para a aplicação do padrão Creator são necessárias, mas não suficientes! Ou seja, não é porque objetos de uma classe A agregam ou são compostos por objetos da classe B, que A será sempre a Creator de B. Por exemplo, veja que não faz sentido a classe *ItemComprado* criar objetos da classe Produto – eles já existem no estoque do supermercado! Considerando isso, propomos outra atividade:

Faça, agora, a segunda atividade Praticando, disponível no ambiente virtual.

Controller

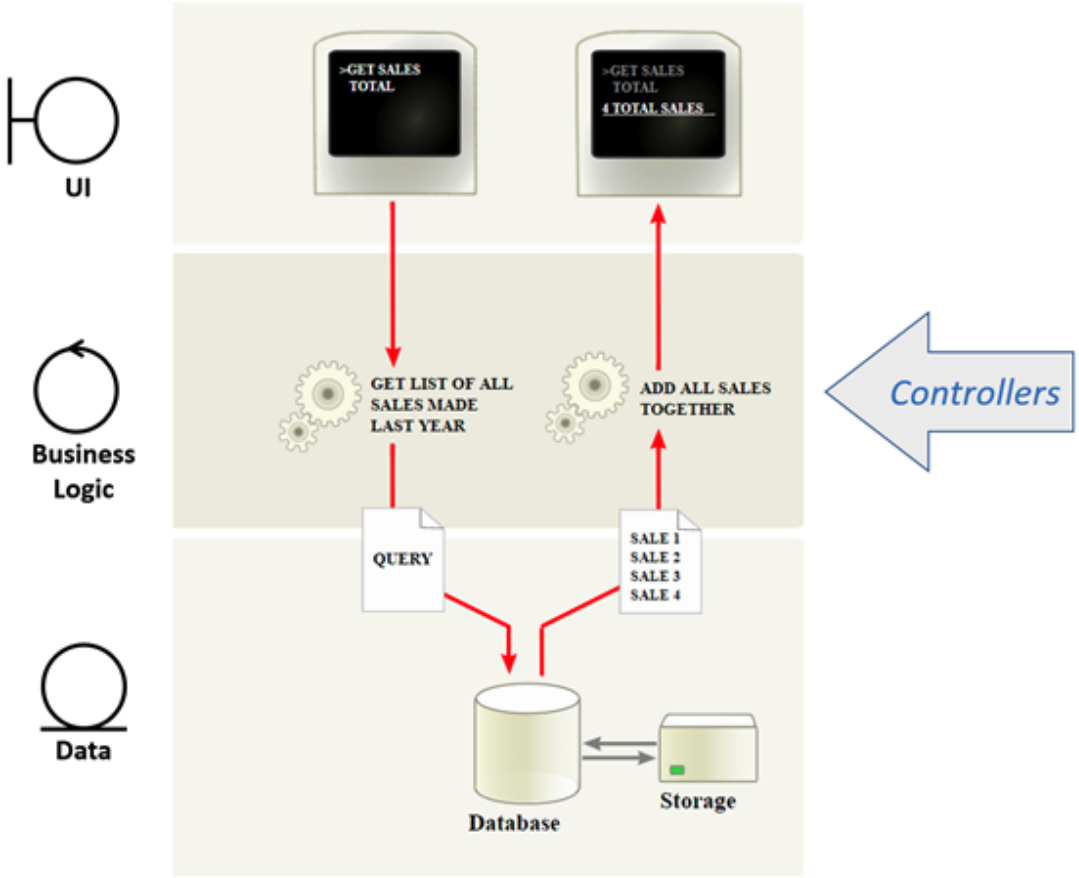
O padrão Controller especifica que deve haver classes específicas para assumir a responsabilidade de lidar com eventos do sistema. Importante: as classes Controller não são parte da interface do usuário (UI)!

Nesse sentido, é importante discutirmos rapidamente alguns pontos relacionados a esse padrão:

- Classes Controller são classes fabricadas artificialmente, ou seja, não representam necessariamente nenhum objeto real do domínio de aplicação.

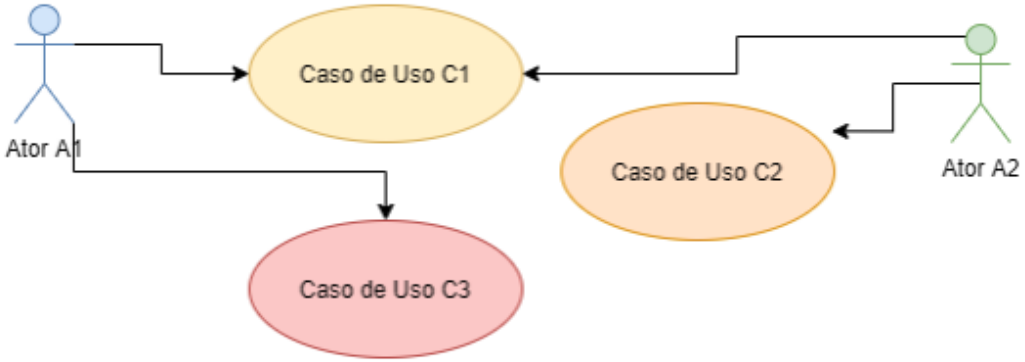
Na próxima aula, veremos o padrão Pure Fabrication, que está relacionado a esse aspecto.

- Em arquiteturas em três camadas (iremos explorá-las mais a fundo em outro momento), as classes Controller pertencem, em geral, à Camada de Lógica de Negócio.

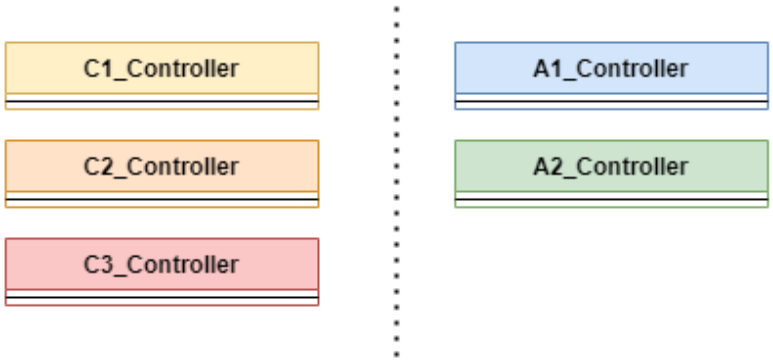


Fonte: Elaborada pelo autor.

- Pensando nos Casos de Uso, pode-se ter uma classe controladora para cada caso de uso, ou uma classe controladora para gerenciar os eventos gerados por cada ator. Veja o exemplo de um Diagrama de Casos de Uso com dois atores e três casos de uso:



- Podem ser criadas classes Controller por ator (cenário à esquerda) ou por caso de uso (cenário à direita).



Fonte: Elaborada pelo autor.

Low Coupling

O padrão Low Coupling é um padrão GRASP avaliativo, ou seja, uma vez definido um diagrama de classes, deve-se verificar o quão acopladas estão essas classes. O acoplamento é uma medida de quão fortemente uma classe depende de outras classes. Quanto maior a dependência entre as classes (ou seja, um acoplamento alto), pior a chance de reutilizar alguma dessas classes em outro contexto. Também faz com que uma mudança em uma classe tenha um impacto grande em outras classes.

Para atingir um modelo com baixo acoplamento, muitas vezes, é necessário refazer o projeto de classes, diminuindo a dependência entre elas.

> Existem maneiras de medir o acoplamento entre classes – uma delas é usando uma métrica de software conhecida como CBO (Coupling Between Objects), que não será tratada aqui. Uma maneira simples e informal de ter essa medida é contando, para cada classe, de quantas outras classes ela depende.

High Cohesion

O padrão High Cohesion é também um padrão avaliativo que tenta identificar se as classes têm as responsabilidades focadas em um único domínio – ou seja, se uma classe tem várias responsabilidades, todas elas devem estar relacionadas entre si – evitando o que popularmente se chama de “classes canivete suíço” (jargão usado para classes que têm muitas funcionalidades não relacionadas). Classes com baixa coesão, em geral, são difíceis de se compreender, dar manutenção, reutilizar ou fazer mudanças. Em geral, este padrão está associado diretamente ao padrão Low Coupling.

> Embora seja mais difícil de medir, existem diferentes métricas para calcular a coesão de uma classe. Possivelmente, a mais famosa é a LCOM (Lack of Cohesion On Methods), que também possui várias formas de ser calculada.