UNIVERSIDADE FEDERAL DE SÃO CARLOS
DEPARTAMENTO DE COMPUTAÇÃO
Rodovia Washington Luiz, Km. 235 – Caixa Postal 676
Fone: (16) 33518230 – Fax: (16) 33518233
CEP 13565-905 – São Carlos – SP – Brasil

Exercício sobre Coesão

Programação Orientada a Objetos Avançada

1) Considere a classe baixo que foi retirada de um sistema de supermercado disponível no GitHub (https://github.com/catarinaribeir0/Supermarket).
Faça uma análise da coesão dessa classe. Pode-se perceber que ela possui vários problemas menores, como por exemplo o nome pouco significativo de alguns métodos e atributos. Se você considera que esta classe possui coesão baixa, o que você faria para aumentar a coesão dela ?

Exemplo de Resposta:
A classe possui responsabilidades que envolvem:

- A busca de produtos que estão no estoque;
- A inserção de produtos no estoque;
- A inserção e remoção de prateleiras e
- A geração de relatório do estoque.

{*Comentário: Esta primeira parte da resposta consiste em um resumo das funcionalidades da classe... é importante começar a resposta deste jeito*}

É possível agrupar essas responsabilidades em quatro grupos: 1) manipulação de produtos; 2) manipulação de prateleiras, 3) geração de relatório e manipulação de estoque.

{*Comentário: Esta segunda parte da resposta consiste em um agrupamento das funcionalidades da classe, já no sentido de mostrar que ela assume mais de uma responsabilidade (se for o caso).* }

Dessa forma, a classe se mostra com pouca coesão, pois alterações em requisitos que sejam relacionados com um determinado grupo, são feitas na mesma classe que envolve outros requisitos. Por exemplo, alterações nos dados do relatório exige a classe Estoque seja modificada e alterações relacionadas com prateleiras também levam a alterações na classe Estoque. As funcionalidades relacionadas com produto são apenas voltadas à busca e inserção de novos produtos, como existe uma classe Produto, essa responsabilidade não está ruim.

Para aumentar a coesão da classe, as responsabilidades seriam divididas em quatro classes, na classe Estoque, Relatório, Produto e Prateleira. Na classe Estoque ficariam as responsabilidades voltadas à manipulação de prateleiras e inserção e remoção de produtos; na classe Relatório ficaria a responsabilidade de geração de relatórios. Na nova classe Prateleira também seriam colocadas as responsabilidades específicas de prateleiras, mesmo sem saber no momento quais seriam, nota-se que é importante uma classe que representa essa abstração.

{Comentário: Este é o fechamento da resposta, cujo objetivo é deixar claro porque a classe possui problemas de coesão. Um exemplo como foi dado é bem vindo.}

```
public class Estoque {

   private Map<Integer, Produto> identificador = new HashMap();
   private Map<Produto, Float> prateleiras = new HashMap();

   public Produto buscarProduto(Integer codProd) {
      return identificador.get(codProd);
   }
   public void gerarRelatorioEstoque() {
      FileWriter estoqueFinalDia;
      FileReader arquivo;
      DateFormat dat = DateFormat.getDateInstance(DateFormat.LONG, new Locale("pt", "BR"));
      try {
         estoqueFinalDia = new FileWriter(new File("RelatorioEstoque " + dat.format(new Date()) + ".txt"));
         arquivo = new FileReader("Produtos.txt");
         BufferedReader estoqueInicioDia = new BufferedReader(arquivo);
         PrintWriter escreverArquivo = new PrintWriter(estoqueFinalDia);
         escreverArquivo.write("|---------- CDL SUPERMERCADOS ----------|\n");
         escreverArquivo.write("|-------------------------------------------------|\n");
         escreverArquivo.write("|---------- RELATORIO DE ESTOQUE ----------|\n");
         escreverArquivo.write("| DATA: " + dat.format(new Date()) + "                    |\n");
         escreverArquivo.write("|       ESTOQUE DISPONIVEL - INICIO DO DIA       |\n");
         escreverArquivo.write(" PRODUTO                          QUANTIDADE\n");
         String linha = estoqueInicioDia.readLine();
         DecimalFormat df = new DecimalFormat("0.###");
         do {
            escreverArquivo.write(" " + linha);
            for (int i = 0; i < 2; i++) {
               linha = estoqueInicioDia.readLine();
```

```
            }
            escreverArquivo.write("      "+ df.format(new Float(linha))+"\n");
            for (int i = 0; i < 2; i++) {
               linha = estoqueInicioDia.readLine();
            }
         } while (linha != null);
         escreverArquivo.write("|                                                  | \n");
         escreverArquivo.write("|          ESTOQUE DISPONIVEL - FIM DO DIA          | \n");
         escreverArquivo.write(" PRODUTO                              QUANTIDADE\n");
         Set<Map.Entry<Produto, Float>> entrySet = prateleiras.entrySet();
         for(Map.Entry<Produto,Float> entrada : entrySet){
            escreverArquivo.write(" " + entrada.getKey() + "       " + df.format(entrada.getValue().floatValue()) +
"\n");
         }
         escreverArquivo.write("|                                                  | \n");
         escreverArquivo.write("|--------------------------------------------------| \n");

         estoqueFinalDia.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
   //Calculos com unidades
   public float quantidadeEstoque(Produto produto) {
      return prateleiras.get(produto);
   }
   public void inserirProduto(float valor, String nome, Float quantidade, int tipo) {
      Produto produto = new Produto(valor, nome, tipo);
      this.identificador.put(produto.getCodigo(), produto);
      this.prateleiras.put(produto, quantidade);
  }
   public void inserirPrateleira(Produto produto, Float quantidade) {
      Float quant = prateleiras.get(produto);
      quant += quantidade;
      prateleiras.put(produto, quant);
   }
   public boolean retirarPrateleira(Produto produto, Float quantidade) {
      Float quant = prateleiras.get(produto);
```

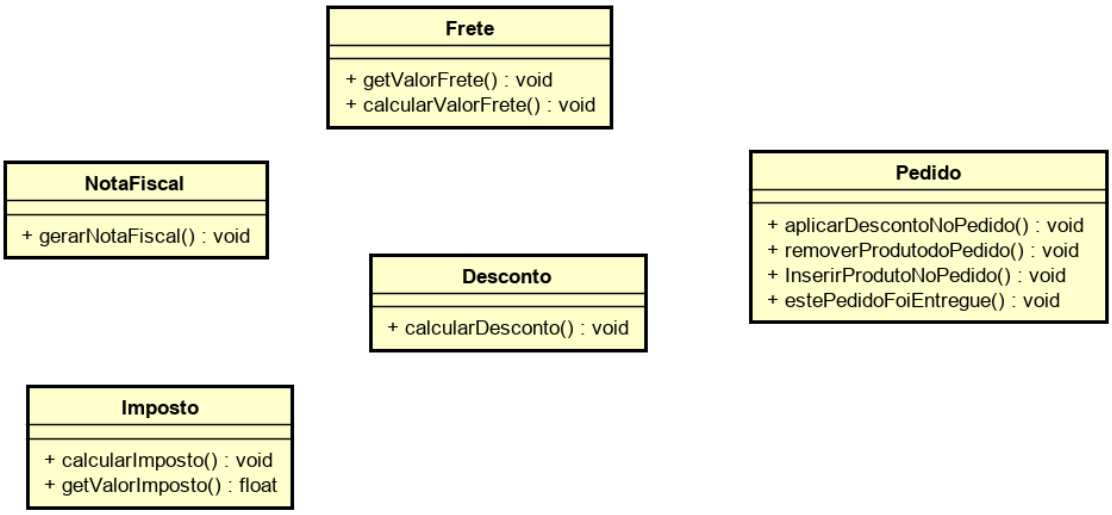
```
        quant -= quantidade;
        if (quant < 0) {
            return false;
        }
        prateleiras.put(produto, quant);
        return true;
    }
}
```

2) Todos os métodos abaixo encontram-se em uma única classe, mas nitidamente a classe possui baixa coesão. Refatore essa classe para que as classes resultantes tenham alta coesão.

   - gerarNotaFiscal()
   - void calcularImposto()
   - float getValorDoImposto()
   - aplicarDescontoNoPedido()
   - calcularDesconto();
   - removerProdutoPedido()
   - inserirProdutoPedido()
   - boolean estePedidoFoiEntregue()
   - getValorFrete()
   - calcularValorFrete()

Exemplo de Resposta:

Considerando que o nome dos métodos revela claramente a funcionalidade que o mesmo realiza, pode-se agrupá-los em cinco grupos funcionalmente relacionados. As classes abaixo mostram a refatoração realizada no sentido de fazer com que cada classe tenha alta coesão.



3) Aplique a métrica LCOO na classe Point.

Passo 1: Inicialmente devemos construir os conjuntos de atributos usados por cada operação.

Point() = {x, y, color, observers}

getX() = {x}

getY() = {y}

setX() = {x}

setY = {y}

getColor() = {color}

setColor() = {color}

attach() = {observers}

detach() = {observers}

notifyObservers() = {observers}

***Passo 2**: Calcular o nro de interseções vazias |V| e não vazias |nV|. Para isso devemos comparar cada método com todos os outros.*

Point() x getX() = {x}

Point() x getY() = {y}

Point() x setX() = {x}

Pont() x setY() = {y}

Point() x getColor() = {color}

Point() x setColor() = {color}

Point() x attach() = {observers}

Point() x detach() = {observers }

Point() x notifyObservers() = {observers }

---

getX() x setX() = { x }

getX() x getY() = { }

getX() x setY() = { }

getX() x getColor() = { }

getX() x setColor() = { }

getX() x attach() = { }

getX() x detach() = { }

getX() x notifyObservers() = { }

---

getY() x setX() = { }

getY() x setY() = { y }

getY() x getColor() = { }

getY() x setColor() = { }

getY() x attach() = { }

getY() x detach() = { }

getY() x notifyObservers = { }

---

setX() x setY() = { }

setX() x getColor() = { }

setX() x setColor() = { }

setX() x attach() = { }

setX() x detach() = { }

setX() x notifyObservers = { }

---

setY() x getColor() = { }

setY() x setColor() = { }

setY() x attach() = { }

setY() x detach() = { }

setY() x notifyObservers = { }

---

getColor() x setColor() = {color}

getColor() x attach() = { }

getColor() x detach() = { }

getColor() x notifyObservers = { }

---

setColor() x attach() = { }

setColor() x detach() = { }

setColor() x notifyObservers = { }

---
attach() x detach() = { observers }

attach() x notifyObservers() = { observers }

Passo 3: Contar o número de interseções vazias |V| e não vazias |nV|

|V| = 30

|nV| = 15

Nota: É importante atentar-se ao número total de interseções existentes, pois isso dá uma panorama do tamanho da classe. Neste caso, o número total de interseções é 45.

Passo 4: Calcular o LCOO

$$\begin{cases} |V| - |nV|, & \text{se } |V| >= |nV| \\ 0, & \text{se } |V| < |nV| \end{cases}$$

Assim:

O LCOO desta classe é 15

Lembrem-se que isso é uma medida de falta de coesão. Assim, se uma outra classe tiver LCOO maior do que 15 signfica que ela é menos coesa do que a classe Point, isto é, quanto maior o número, menos coesa ela é.

Como foi mostrado nos slides, é melhor considerar a porcentagem comparando com o número total de interseções. Neste caso, há 45 interseções, então 15/45 = 30%. Isto é, essa classe possui 30% de interseções vazias a mais do que o número de não vazias. Então, pode-se dizer que essa classe sofre de falta de coesão.

4) Aplique a métrica LCOO na classe abaixo.

```
public class C {

        private int att1;
        private String att2;

        public m1(){

                usa os atributos att1 e att2;

        }

        public m2() {

                usa os atributos att1 e att2;


        }

        public m3() {

                usa os atributos att1 e att2;

        }
```

Cálculo da Métrica:

Passo 1:

m1() = {att1, att2}
m2() = {att1, att2}
m3() = {att1, att2}

Passo 2:

m1x m2 = {att1, att2}
m1x m3 = {att1, att2}
m2x m3 = {att1, att2}

Passo 3:

|V| = { }
|nV| = 3

Passo 4:

|V| - |nV|, se |V| >= |nV|
0 , se |V| < |nV|

Assim:

O LCOO desta classe é 0, demonstrando que ela possui alta coesão.

5) Aplique a métrica LCOO na classe abaixo.

```
public class C {

        private int att1;
        private String att2;
        private int att3;
        private String att4;
```

```
public m1(){
        usa os atributos att1 e att2;
}

public m2() {
        usa os atributos att1 e att2;
}

public m3() {
        usa os atributos att1 e att2;
}

public m4(){
        usa os atributos att3 e att4;
}

public m5() {
        usa os atributos att3 e att4;
}

public m6() {
        usa os atributos att3 e att4;

}
```

Cálculo da Métrica:

Passo 1:

m1() = {att1, att2}
m2() = {att1, att2}
m3() = {att1, att2}
m4() = {att3, att4}
m5() = {att3, att4}
m6() = {att3, att4}

Passo 2 (Interseções):

m1x m2 = {att1, att2}
m1x m3 = {att1, att2}
m1 x m4 = { }
m1 x m5 = { }
m1 x m6 = { }

m2x m3 = {att1, att2}
m2 x m4 = { }
m2 x m5 = { }
m2 x m6 = { }

m3 x m4 = { }
m3 x m5 = { }
m3 x m6 = { }

m4 x m5 = { att3, att4 }
m4 x m6 = { att3, att4 }

m5 x m6 = { att3, att4 }

Passo 3:

|V| = 9
|nV| = 6

Passo 4:

|V| - |nV|, se |V| >= |nV|
0 , se |V| < |nV|

Assim:

O LCOO desta classe é 3.

Analisando a porcentagem: = 3/15 = 0,2 = 20%

Assim, a classe possui 20% de interseções vazias a mais do que interseções não vazias. Então, a classe também sofre de falta de coesão.