

# Resolução de Exercícios

## Lista 1

Profa. Fabíola S. F. Pereira

[fabiola.pereira@ufu.br](mailto:fabiola.pereira@ufu.br)

# LISTA 1

## QUESTÃO 2

Suponha que um programador adote a seguinte estratégia: ao implementar qualquer nova funcionalidade ou corrigir um bug que implique na modificação de duas classes A e B localizadas em arquivos diferentes, ele conclui a tarefa movendo as classes para o mesmo arquivo. Explicando melhor: após terminar a tarefa de programação que ficou sob sua responsabilidade, ele escolhe uma das classes, digamos a classe B, e a move para o mesmo arquivo da classe A. Agindo dessa maneira, ele estará melhorando qual propriedade de projeto? Por outro lado, qual propriedade de projeto estará sendo afetada de modo negativo? Justifique.

## QUESTÃO 2

```
import Produto;

class ItemDeCompra {

    Produto produto;
    float preco;
    int quantidade;

    float calculaDescontoPromocaoAtacado(){

        if quantidade > 5
            return "10% de desconto"

    }

}
```

```
import ItemDeCompra;

class Compra {

    Date dataDeCompra;
    float valorTotal;
    float desconto;
    String formaPagamento;
    List<ItemDeCompra> itens;

    float calculaDescontoFormaPagamento(){
        return "20% se dinheiro"
    }

}
```

# QUESTÃO 2

```
import Produto;

class Compra {

    Date dataDeCompra;
    float valorTotal;
    float desconto;
    String formaPagamento;
    List<Produto> itens;
    List<Float> precos;
    List<Integer> quantidades;
    List<Float> precosFinaisComDescontoAtacado;

    float calculaDescontoFormaPagamento(){
        return "20% se dinheiro"
    }

    float calculaDescontoPromocaoAtacado(){

        if quantidade > 5
            return "10% de desconto"

    }

}
```

# QUESTÃO 2

Basicamente, o desenvolvedor estará diminuindo o número de dependências entre os arquivos do sistema. Isso pode ser considerado como sendo uma diminuição do acoplamento (estrutural) entre tais arquivos. A princípio, isso poderia ser considerado como uma boa política. Porém, por outro lado, o desenvolvedor está também diminuindo a coesão dos arquivos, pois após mover a classe B para o arquivo da classe A, ele expande a lista de tarefas que A realiza. Por consequência, aumentam as chances de A realizar tarefas distintas e não relacionadas. Ou ainda, tarefas não-coesas. Logo, diminui-se a coesão de A, o que é um resultado ruim.

Imagina então, que o acoplamento é o "número de imports" que vão aparecer no seu código. Se você juntou duas classes, você diminuiu os "imports". Logo, diminuiu o acoplamento.

Porém, você trouxe mais responsabilidades pra uma classe apenas. Logo, essa classe estará fazendo mais do que deveria. Ou seja, não está coesa.

# QUESTÃO 6

Costuma-se afirmar que existem três conceitos chaves em orientação a objetos: encapsulamento, polimorfismo e herança. Suponha que você tenha sido encarregado de projetar uma nova linguagem de programação. Suponha ainda que você poderá escolher apenas dois dos três conceitos que mencionamos. Qual dos conceitos eliminaria então da sua nova linguagem? Justifique sua resposta.

# QUESTÃO 6

Costuma-se afirmar que existem três conceitos chaves em orientação a objetos: encapsulamento, polimorfismo e herança. Suponha que você tenha sido encarregado de projetar uma nova linguagem de programação. Suponha ainda que você poderá escolher apenas dois dos três conceitos que mencionamos. Qual dos conceitos eliminaria então da sua nova linguagem? Justifique sua resposta.


**Herança, pois existe um princípio de projeto que recomenda que devemos "preferir composição a herança". Inclusive, linguagens de programação mais recentes — como Go — não incluem suporte a herança.**



# QUESTÃO 9

( ) Getters e setters são essenciais para implementação de classes com ocultamento de informação.

# QUESTÃO 9

(  ) Getters e setters são essenciais para implementação de classes com ocultamento de informação.

```
import java.util.HashtableEspecial;  
  
public class Estacionamento {  
  
    private HashtableEspecial<String, String> veiculos;  
  
    public Estacionamento() {  
        veiculos = new Hashtable<String, String>();  
    }  
  
    public void estaciona(String placa, String veiculo) {  
        this.veiculos.put(placa, veiculo)  
    }  
  
    public static void main(String[] args) {  
  
        Estacionamento estacionamento = new Estacionamento();  
  
        estacionamento.estaciona("TCP-7030", "Uno");  
        estacionamento.estaciona("BNF-4501", "Gol");  
        estacionamento.estaciona("JKL-3481", "Corsa");  
    }  
}
```

# QUESTÃO 9

( **F** ) Getters e setters são essenciais para implementação de classes com ocultamento de informação.

```
import java.util.HashtableEspecial;  
  
public class Estacionamento {  
    private HashtableEspecial<String, String> veiculos;  
  
    public Estacionamento() {  
        veiculos = new Hashtable<String, String>();  
    }  
  
    public void estaciona(String placa, String veiculo) {  
        this.veiculos.put(placa, veiculo)  
    }  
  
    public static void main(String[] args) {  
        Estacionamento estacionamento = new Estacionamento();  
  
        estacionamento.estaciona("TCP-7030", "Uno");  
        estacionamento.estaciona("BNF-4501", "Gol");  
        estacionamento.estaciona("JKL-3481", "Corsa");  
    }  
}
```

# QUESTÃO 9

( ) Classes grandes tendem a ter problemas de coesão (baixa) e acoplamento (alto).

# QUESTÃO 9

( **V ou F** ) Classes grandes tendem a ter problemas de coesão (baixa) e acoplamento (alto).

Classes “faz tudo” tendem a ter baixo acoplamento (uma única classe).

No entanto, assertiva muito genérica. É possível ter alto acoplamento em classes grandes também. Por isso, aceita-se as duas respostas.

# QUESTÃO 11

( ) Suponha uma classe de domínio X com grupos de métodos M1, M2 e M3, os quais são usados, respectivamente, pelos sistemas contábil, financeiro e de compras de uma empresa. Esse fato é um indicativo de que X viola o Princípio da Responsabilidade Única.

# QUESTÃO 11

( **V** ) Suponha uma classe de domínio X com grupos de métodos M1, M2 e M3, os quais são usados, **respectivamente**, pelos sistemas contábil, financeiro e de compras de uma empresa. Esse fato é um indicativo de que X viola o Princípio da Responsabilidade Única.

FIM