PUC-SP

Engenharia de Software
Tecnologia

Princípios de Projeto Orientado a Objetos (OO)

Prof. Carlos Paes

PRINCÍPIOS SOLID

1) Considere o seguinte trecho de código em Java que representa uma Conta Corrente de um sistema de bancário.

```
class Conta {

    public boolean autenticar(String agencia, String conta, String senha){
        //Lógica de autenticacao
    }

    public void trocarSenha(){
        //Lógica de trocar senha
    }

    public void depositar(float valor){
        //Lógica para incrementar saldo
    }

    public boolean retirar(float valor){
        //Lógica para retirada
    }

    public boolean retirar(float valor, Conta destino){
        //Lógica para transferencia
    }

    public void imprimirExtratoNaTela(){
        //Lógica para imprimir extrato
    }

    public void extratoPorEmail(){
        //Lógica para imprimir extrato
    }
}
```

A classe Conta está com diversas responsabilidades, fazer a autenticação do usuário, cuidar do saldo e imprimir extrato. A primeira vista pode parecer correto, afinal a responsabilidade da classe é a conta de usuário. Mas se por algum motivo você precisar mudar a lógica de autenticação, você pode sem querer afetar a funcionalidade de transferência entre contas por exemplo. Proponha uma refatoração desse código usando os princípios SOLID apresentado em sala de aula. Faça a implementação necessária no código e

crie novas classes para um protótipo funcional de um sistema de conta corrente. Sugestão: usar com ponto de partida o modelo de objetos apresentado no Anexo A.

2) Considere o seguinte problema: A classe Funcionário tem um método que calcula o salário do funcionário baseado no seu cargo e sua comissão.

```
class Funcionario{

   //Atributos

   private float salarioBase;
   private String cargo;

   //Outros Methodos

   public float getSalario(float totalVendas){
      if(cargo == "Secretaria"){
         return salarioBase;
      }

      if(cargo == "Vendedor"){
         return salarioBase + (totalVendas * 0.3);
      }
   }
}
```

Essa classe parece correta, está bem coesa e parece não ter responsabilidades demais. Mas imagine agora que você precisa adicionar a essa classe o cálculo do salário do supervisor regional. A regra é a seguinte: 10% (dez por cento) da soma das comissões de todos os vendedores sob sua supervisão adicionado de seu salário base.

```
class Funcionario{

   private float salarioBase;
   private String cargo;
   private float totalVendas;
   private float vendas;

   public float getComissao(List funcionarios){
      if(cargo == "Vendedor"){
         return this.vendas * 0.3;
      }

      if(cargo == "Supervisor Regional"){
         float comissao = 0.0f;
         for(Funcionario funcionario : funcionarios){
            comissao += funcionario.getComissao();
         }

         return (comissao * 0.1);
      }
   }
   public float getSalario(List funcionarios){
      if(cargo == "Secretaria"){
         return salarioBase;
      }
      if(cargo == "Vendedor"){
         return salarioBase + (vendas * 0.3);
      }
      if(cargo == "Supervisor Regional"){
         float comissao = 0.0f;
         for(Funcionario funcionario : funcionarios){
            comissao += funcionario.getComissao();
         }

         return salarioBase + (comissao * 0.1);
      }
      if(cargo == "Supervisor Regional"){
         return salarioBase * getComissao(funcionarios);
      }
   }
}
```

As seguintes alterações foram realizadas na classe:

- Um campo ***totalVendas*** foi adicionado para saber quanto cada Funcionário vendeu;
- Um novo método ***getComissao*** foi criado para calcular a comissão de cada Vendedor
- O método ***getSalario*** foi modificado para adicionar o novo tipo de Funcionário Supervisor Regional. Além disso a assinatura do método foi alterada.

Agora aparece um novo requisito. O sistema precisa calcular o salário do Gerente de Vendas usando a seguinte regra: salário base adicionado de 20% (vinte por cento) do valor total da comissão de cada um dos supervisores sob seu comando, mais um valor fixo que vai de 0 a 20.000 de acordo com metas de vendas. Segue abaixo o código Java com a implementação da nova funcionalidade:

```
Class Funcionario {

//Atributos

   private float salarioBase;
   private String cargo;
   private float totalVendas;
   private float vendas;

   //Outros Methodos

   public float getComissao(List funcionarios){
      if(cargo == "Vendedor"){
         return this.vendas * 0.3;
      }

      if(cargo == "Supervisor Regional"){
         float comissao = 0.0f;
         for(Funcionario funcionario : funcionarios){
            comissao += funcionario.getComissao();
         }

         return (comissao * 0.1);
      }

      if(cargo == "Gerente"){
         float comissao = 0.0f;
         float vendas = 0.0f;

         for(Funcionario funcionario : funcionarios){
            if(funcionario.getCargo() == "Supervisor Regional"){
               comissao += funcionario.getComissao();
               vendas += funcionario.getVendas();
            }
         }

         float comissaoPorMeta = 0.0f;

         if(vendas == 50.000){
            comissaoPorMeta = 5.000;
         }

         if(vendas == 100.000){
            comissaoPorMeta = 8.000;
         }
```

```
         if(vendas &gt; 100.000){
            comissaoPorMeta = 8.000;
         }

         if(vendas &gt; 200.000){
            comissaoPorMeta = 15.000;
         }

         if(vendas &gt; 500.000){
            comissaoPorMeta = 20.000;
         }

         return (comissao * 0.2) + comissaoPorMeta;
      }
   }

   public float getSalario(List funcionarios){
      if(cargo == "Secretaria"){
         return salarioBase;
      }

      if(cargo == "Vendedor"){
         return salarioBase + (vendas * 0.3);
      }

      if(cargo == "Supervisor Regional"){
         float comissao = 0.0f;
       for(Funcionario funcionario : funcionarios){

            comissao += funcionario.getComissao();
         }

         return salarioBase + (comissao * 0.1);
      }

      if(cargo == "Supervisor Regional"){
         return salarioBase * getComissao(funcionarios);
      }
   }
}
```

Para cada novo comportamento que adicionamos, a nossa classe fica mais complexa. Mais isso só iria tornar a classe mais complexa do ponto de vista de quem a utiliza. Proponha uma refatoração desse código usando os princípios SOLID apresentado em sala de aula. Faça a implementação necessária no código e crie novas classes para um protótipo funcional do exemplo apresentado.

3) Considere a classe Funcionário e os relacionamentos de herança com as classes Vendedor e Gerente.

```
class Funcionario{
    private float salario;
    private String cargo;
    private String nome;
}

class Vendedor extends Funcionario{
    private float comissao;

    public float getSalarioVendedor(){
        return salario + comissao;
    }
}

class Gerente extends Funcionario {
    private float bonus;

    public float getSalarioGerente(){
        return salario + bonus;
    }
}
```

Analisando essas classes parece não haver qualquer problema. Cada classe possui a sua responsabilidade e não há necessidade de alterar a classe cada vez que criamos um novo cargo. Porém, imagine que você possua o seguinte módulo que imprima a folha salarial.

```
class FolhaSalarial{
   private Date data;

   public void imprimirFolhaSalarial(List funcionarios){
      for(Funcionario f : funcionarios){

         if(funcionario.getCargo() == "Vendedor"){
            System.io.println(funcionario.getNome() + " ----- " + funcionario.getSalarioVendedor());
         }

         if(funcionario.getCargo() == "Gerente"){
            System.io.println(funcionario.getNome() + " ----- " + funcionario.getSalarioGerente());
         }
      }
   }
}
```

Qual princípio SOLID está sendo violado neste exemplo? Explique sua resposta!

Caso algum princípio esteja sendo violado, proponha uma solução (refatore) para o problema. Apresente o código fonte com as alterações e um protótipo funcional.

4) O código Java abaixo implementa as seguintes classes:

- A classe abstrata Funcionário que possui a lógica básica de um funcionário de uma empresa
- A classe Vendedor que é um funcionário, mas possui uma própria lógica de calcular o seu salário e a sua comissão.
- A classe Representante que não é exatamente um funcionário, pois não recebe um salário base, porém recebe uma comissão maior sobre suas vendas.
- A Atendente de Caixa que é um funcionário mas não recebe comissão.

```
abstract class Funcionario{
   private float salarioBase;
   private String cargo;
   private String nome;

   public float getSalarioBase(){
      return this.salarioBase;
   }

   abstract float getSalario();
   abstract float getComissao();
}

class Vendedor extends Funcionario {
   private float totalVendas;

   public float getSalario(){
      return this.salarioBase + this.getComissao();
   }

   public float getComissao(){
      return this.totalVendas * 0.1;
   }
}

class Representante extends Funcionario {
   private float totalVendas;

   public float getSalario(){
      return 0.0f;
   }

   public float getComissao(){
      return this.totalVendas * 0.3;
   }
}

class AtendenteDeCaixa extends Funcionario {

   public float getSalario(){
      return this.getSalarioBase();
   }

   public float getComissao(){
      return 0.0f;
   }
}
```

Qual o princípio SOLID que está sendo violado neste caso? Explique a sua resposta!

Faça a refatoração necessária no código para resolver o problema.

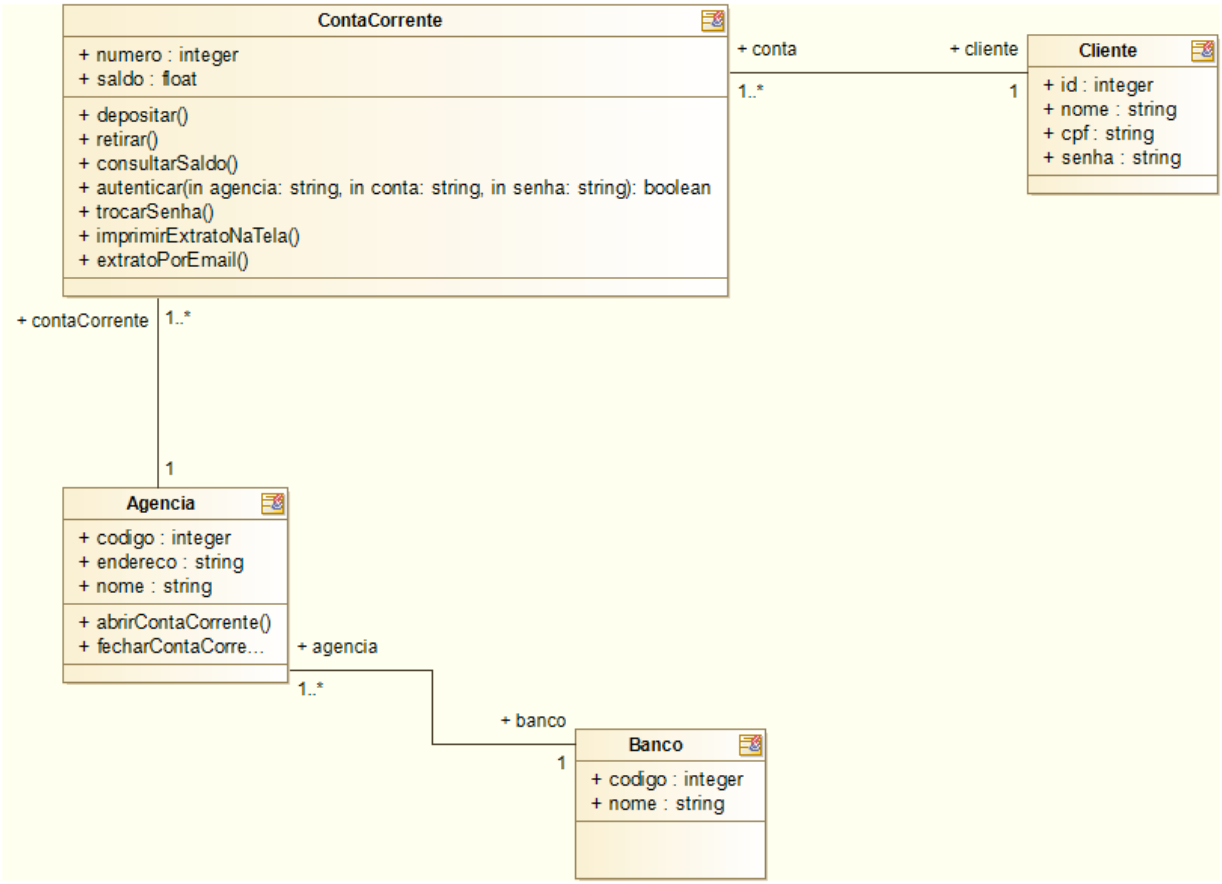
5) Considere o código Java abaixo:

```
class Funcionario {
    private String cargo;
    private float salario;
    private float comissao;

    public String getSalario(){
        return this.salario + this.getComissao();
    }

    public float getComissao(){
        return this.comissao;
    }

    public String getCargo(){
        return this.cargo;
    }

}

class Pagamento {

    private Funcionario funcionario;

    public float getSalario(){
        if(funcionario.getCargo() == "Representante"){
            return funcionario.getComissao();
        }

        return funcionario.getSalario();
    }
}
```

Qual princípio SOLID está sendo violado? Explique sua resposta?

Faça a refatoração necessária para resolver o problema.

Anexo A


ContaCorrente
+ numero : integer
+ saldo : float
+ depositar()
+ retirar()
+ consultarSaldo()
+ autenticar(in agencia: string, in conta: string, in senha: string): boolean
+ trocarSenha()
+ imprimirExtratoNaTela()
+ extratoPorEmail()
+ conta
1..*
+ cliente
1
Cliente
+ id : integer
+ nome : string
+ cpf : string
+ senha : string
+ contaCorrente
1..*
1
Agencia
+ codigo : integer
+ endereco : string
+ nome : string
+ abrirContaCorrente()
+ fecharContaCorre...
+ agencia
1..*
+ banco
1
Banco
+ codigo : integer
+ nome : string