PUC-MG: Pontifícia Universidade Católica de Minas Gerais	
Curso: Engenharia de Software	
Disciplina: Programação Modular	
Professor(a): Pedro Pongelupe Lopes	
Semestre: 2024.1	
Aluno:	**Matrícula:**

Exercício de Revisão Prova 2

Questão 1 Analise os seguintes fragmentos de código e responda: Houve violação de algum princípio SOLID? Se sim, qual e por quê? Reescreva o fragmento para ajustar o problema, caso necessário.

Para todos os problemas, todas as classes têm os getters, setters e os construtores que forem necessários.

(a)

```
public abstract class Estabelecimento  {
        private String razaoSocial;
}
public class Cafeteria extends Estabelecimento {
        public int getAreaConstruidaCafeteria() { return 70; }
        public double getValorMetroQuadradoCafeteria() { return 40d; }
        public double getAliquotaCafeteria() { return 0.5; }
}
public class Livraria extends Estabelecimento {
        public int getAreaConstruidaLivraria() { return 150; }
        public double getValorMetroQuadradoLivraria() { return 10d; }
        public double getAliquotaLivraria() { return 0.8; }
}

public class CalculadoraIPTU {

   public double calculaIPTU(Estabelecimento e) {
        double iptu = 0;

        if (e instanceOf Cafeteria) {
                Cafeteria c  = (Cafeteria) e;
                iptu = c.getAreaConstruidaCafeteria() * c.getValorMetroQuadradoCafeteria() * c.getAliquotaCafeteria();
        }
        if (e instanceOf Livraria) {
                Livraria l  = (Livraria) e;
                iptu = l.getAreaConstruidaLivraria() * l.getValorMetroQuadradoLivraria() * l.getAliquotaLivraria();
        }

        return iptu;
   }

}
```

Resposta O princípio representado pela letra 'O', *Open-Close Principle*, foi violado, pois as classes *Cafeteria* e *Livraria* estabelecem métodos que deveriam estar na superclasse como métodos abstratos. Consequentemente, a classe *CalculadoraIPTU* deverá ser alterada para cada nova especialização de *Estabelecimento*.

```
public abstract class Estabelecimento  {
        private String razaoSocial;
        public abstract int getAreaConstruida();
        public abstract double getValorMetroQuadrado();
        public abstract double getAliquota();
}
public class Cafeteria extends Estabelecimento {
        @Override
        public int getAreaConstruida() { return 70; }
        @Override
        public double getValorMetroQuadrado() { return 40d; }
        @Override
        public double getAliquota() { return 0.5; }
}
public class Livraria extends Estabelecimento {
        @Override
        public int getAreaConstruida() { return 150; }
        @Override
        public double getValorMetroQuadrado() { return 10d; }
        @Override
        public double getAliquota() { return 0.8; }
}

public class CalculadoraIPTU {

   public double calculaIPTU(Estabelecimento e) {
        return e.getAreaConstruida() * e.getValorMetroQuadrado() * e.getAliquota();
   }

}
```

(b)

```
public abstract class InstrumentoMusical {
        private String modelo;
        private String ano;

        public abstract void tocar();

        public abstract void afinar();
}
public class Piano implements InstrumentoMusical {
        private Corda[] cordas;

        @Override
        public void tocar() { /*...*/ }

        @Override
        public void afinar() { /*...*/ }

        public void trocarCordas(Corda[] cordas) { this.cordas = cordas; }
}
public class Trompete implements InstrumentoMusical {
        private Surdina surdina;

        @Override
        public void tocar() { /*...*/ }

        @Override
        public void afinar() { /*...*/ }

        public void adicionarSurdina(Surdina s) { this.surdina = s; }
}
```

Resposta Não há violação dos princípios SOLID neste fragmento.

(c)

```
public class GerenciadorDeSistema {

        public void adicionaUsuario(Usuario u) { /*..*/ }

        public void removeUsuario(Usuario u) { /*..*/ }

        public void enviaNotificacao(String notificacao) { /*..*/ }

        public void enviaEmail(String email, Usuario destinatario, Usuario remetente) { /*..*/ }

}
```

Resposta O princípio representado pela letra 'S', *Single Responsibility Principle*, foi violado, pois a classe *GerenciadorDeSistema* possui inumeras responsabilidades, sendo uma classe deus. Portanto, o melhor procedimento a ser feito é refatorar a classe *GerenciadorDeSistema* em outras.

```
public class GerenciadorDeUsuarios {

        public void adicionaUsuario(Usuario u) { /*..*/ }

        public void removeUsuario(Usuario u) { /*..*/ }
}

public class GerenciadorDeComunicacoes {

        public void enviaNotificacao(String notificacao) { /*..*/ }

        public void enviaEmail(String email, Usuario destinatario, Usuario remetente) { /*..*/ }
}
```

Questão 2 Para cada cenário proposto, discuta uma possível implementação utilizando uma *collection* para resolver o problema. Justifique a escolha da *collection*.

(a) Em um sistema de gestão acadêmica, precisamos de uma classe Aluno é responsável por armazenar os dados e comportamentos dessa entidade do sistema. Essa classe é responsável por gerir as notas, permitindo adicionar, editar, consultar e deletar dessas notas.

Resposta Para o cenário descrito, a utilização de uma lista, *List*, é ideal, pois, provê todas as funcionalidades necessárias. Além disso, não há requisitos quanto a ordem de inserção ou de remoção.

(b) Em um sistema de restaurante, precisamos de uma classe para gerenciar todas as reservas de clientes caso o restaurante não tenha mesas vagas. A ordem de chegada deve ser a mesma ordem da saída, ou seja, o primeiro a chegar deve ser o primeiro a sair.

Resposta Para o cenário descrito, a utilização de uma Fila, *Queue*, é ideal, pois, provê a inserção e a remoção utilizando a lógica de uma fila, logo, o primeiro que chega, é o primeiro que sai.

(c) Em um sistema de stream de música, precisamos de uma classe para a playlist de músicas. Uma playlist pode conter entre 0 e 150 músicas únicas, ela pode ordernar as músicas de várias maneiras, como: ordem alfabética, duração das músicas e pela ordem de inserção. Nesse sistema, a playlist só pode ter uma ordernação e a ordenação padrão é feita pela ordem de inserção.

Resposta Para o cenário descrito, a utilização de um conjuto, *Set*, é ideal, pois, permite apenas itens únicos. A implementação que poderia ser utilizada é o TreeSet que pode receber um compartor para ordernar as músicas.

(d) Em um sistema para a agência de transito, precisamos de uma classe para calcular o valor do IPVA de um carro. O IPVA é calculado pela multiplicação do valor venal do veículo pela alíquota referente ao Estado qual o veículo foi registrado, portanto, existem 27 alíquotas no Brasil.

Resposta Para o cenário descrito, a utilização de um mapa, *Map*, é ideal, pois, podemos guardar todas as alíquotas por estado. Ou seja, a chave seria o Estado e o valor seria a alíquota associada.