INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA PIAUÍ	**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ** **Curso: ADS** **Disciplina: Engenharia de Software III** **Professor: Ely**

Exercícios 04 – Parte 2

Responda as questões abaixo usando o princípio da substituição de Liskov.

1. Em um sistema bancário temos as seguintes classes:

```
public class ContaBancaria {
    private double saldo;

    public ContaBancaria(double saldoInicial) {
        this.saldo = saldoInicial;
    }

    public void depositar(double valor) {  saldo += valor;  }

    public void sacar(double valor) { saldo -= valor; }
}

public class ContaPoupanca extends ContaBancaria {
    public ContaPoupanca(double saldoInicial) {
        super(saldoInicial);
    }

    @Override
    public void sacar(double valor) {
       if (valor > 1000) {
         throw new
           RuntimeException("Não pode sacar mais de 1000 em uma poupança");
        }
        super.sacar(valor);
    }
}
```

Explique por que motivo a herança entre as classes não é justificada e proponha uma solução.

2. Proponha uma solução que evite o uso da herança no código abaixo:

```
public class Conta {
    private double saldo;
    private String numeroConta;
```

```
    public Conta(String numeroConta, double saldoInicial) {
        this.numeroConta = numeroConta;
        this.saldo = saldoInicial;
    }

    public void depositar(double valor) {
        saldo += valor;
    }

    public void sacar(double valor) {
        if (valor > saldo) {
            throw new IllegalArgumentException("Saldo insuficiente.");
        }
        saldo -= valor;
    }

    // Outros métodos relevantes...
}

public class ContaCliente extends Conta {
    private String nome;
    private String cpf;
    private String endereco;

     public ContaCliente(String numeroConta, double saldoInicial,
                         String nome, String cpf, String endereco) {
        super(numeroConta, saldoInicial);
        this.nome = nome;
        this.cpf = cpf;
        this.endereco = endereco;
    }

    // Métodos específicos do cliente...
}
```

3. Aplique o princípio LSP à implementação abaixo de forma que persistência seja um atributo da segunda classe. Crie um exemplo real com o resultado da refatoração.

```
import java.io.*;

public class Persistencia {
    public void salvar(String dados, String arquivo) throws IOException {
        try (FileWriter writer = new FileWriter(arquivo)) {
            writer.write(dados);
        }
    }
}

public class PersistenciaJSON extends Persistencia {
    @Override
    public void salvar(String dados, String arquivo) throws IOException {
        if (!dados.startsWith("{")) {
              throw new
                IllegalArgumentException("Dados não estão em formato JSON.");
        }
        super.salvar(dados, arquivo);
    }
}
```

4. Entenda o problema da herança entre patos e suas capacidades presentes no exemplo do link: https://www.quora.com/What-are-some-Java-examples-for-the-OOP-principle-of-favoring-object-composition-over-inheritance

   Proponha um exemplo que também viola o LISP e exiba sua solução.

5. As classes Postagem, Reacao e Comentario possuem uma herança apenas para aproveitar alguns atributos e reescrever o método exibir().

```
public class Perfil {
    private int id;
    private String nomeUsuario;

    public Perfil(int id, String nomeUsuario) {
        this.nomeUsuario = nomeUsuario;
        this.id = id;
    }

    // Outros métodos
}

public class Postagem {
    private String id;
    private Perfil autor;
    private String conteudo;

    public Postagem(String id, Perfil autor, String conteudo) {
        this.id = id;
        this.autor = autor;
        this.conteudo = conteudo;
    }

    public void exibir() {
        System.out.println("Postagem [" + id + "] de " + autor.getNomeUsuario() +
": " + conteudo);
    }

     // outros métodos
}

public class Reacao extends Postagem {
    private String tipoReacao;

    public Reacao(String id, Perfil autor, String tipoReacao) {
        super(id, autor, null);
        this.tipoReacao = tipoReacao;
    }

    @Override
    public void exibir() {
        System.out.println("Reação [" + tipoReacao + "] de " +
getAutor().getNomeUsuario() + " na postagem " + getId();
    }
```

```
     // outros métodos
}

public class Comentario extends Postagem {
    private Postagem postagemOriginal;

    public Comentario(String id, Perfil autor, String conteudo, Postagem
postagemOriginal) {
        super(id, autor, conteudo);
        this.postagemOriginal = postagemOriginal;
    }

    @Override
    public void exibir() {
        System.out.println("Comentário de " + getAutor().getNomeUsuario() + " em
resposta a postagem [" + postagemOriginal.getId() + "]: " + conteudo);
    }

    // outros métodos
}
```

Refatore as classes de forma a:

a. Reacao tenha uma composição com postagem;
b. Comentário também tenha uma composição com postagem;
c. Postagem tenha uma coleção de reações e comentários;
d. Todos implementem a interface abaixo:

```
public interface Publicavel {
    void exibir();
    Perfil getAutor();
}
```