Uso dos princípios de SOLID

Princípio da Responsabilidade Única (SRP):

- **Definição:** Uma classe deve ter apenas uma razão para mudar.
- **Exemplo:**
  - A classe `veiculoController` é responsável por:
    - Manipular requisições HTTP.
    - Acessar o banco de dados.
    - Gerenciar as respostas HTTP.
  - **Sugestão:**
    - Separar essas responsabilidades em classes distintas.

Princípio Aberto/Fechado (OCP):

- **Definição:** Uma classe deve ser aberta para extensão, mas fechada para modificação.
- **Exemplo:**
  - A classe `veiculoController` atualmente para aceitar novos tipos de operações precisa modificar todo o código existente.
  - **Sugestão:**
    - Adicionar novos métodos para diferentes tipos de operadores. Para isso, deve obedecer ao princípio da responsabilidade única.

Princípio da Substituição de Liskov (LSP):

- **Definição:** O método calcular na classe altera o comportamento esperado da classe base (XmlOperation). Além de realizar a remoção de produtos não devolvidos, ele chama super.calcular(xml, itensADevolver), adicionando um comportamento adicional que a classe base não prevê. O LSP pede que a substituição de uma classe base por sua classe derivada não modifique o contrato da classe base.

- **Dica:**
  - A adição de uma interface comum, como Operation é crucial. e durante a execução da cadeia o método calcular da interface Operation é chamadogarantindo que cada operação pode ser invocada de maneira consistente, independentemente da implementação específica.

Violação no LSP Classe xmlService

Antes

```
public class XMLService {

    public ResponseEntity<CalculatedXmlResponse> calculateXml(String xml, List  <ItemXml> items) {
        List operations = new ArrayList<>();
        operations.add(RemoverProdNaoDevolvidos(xml, items));
        operations.add(ZerarTotalXML(xml, items));
        operations.add(IniciarComValorDeTodosProdutosEmTotal(xml, items));
        operations.add(QuantidadeDevolvidaOperation(xml, items));
        operations.add(FreteOperation(xml, items));
        operations.add(DescontoOperation(xml ,items));
        operations.add(ICMSOperation(xml, items));
        operations.add(IPIOperation(xml, items));
        operations.add(BaseCalculoIcmsStOperation(xml, items));
        operations.add(FecopOperation(xml, items));
        operations.add(ICMSStOperation(xml, items));

        for(i=0;i<operations.size();i++){
            xml = operations.get(i)
        }

        CalculatedXmlResponse calculatedXmlResponse = new CalculatedXmlResponse();
        calculatedXmlResponse.setXml(xml)

        return ResponseEntity.ok().body(calculatedXmlResponse);
    }

}
```

Depois

```java
public class XmlOperation implements Operation {
    protected Operation next;
    private String XmlBase;

    public void setXmlBase(String xml) {
        this.XmlBase = xml;
    }

    public String calcular(String xml, List<ItemXml> itensADevolver){
        if (XmlBase==null) {
            setXmlBase(xml);
        }

        if(next != null){
            next.setXmlBase(getXmlBase());
            return next.calcular(xml, itensADevolver);
        }
        return xml;
    }

    public Operation setNext(Operation operation) {
        this.next = operation;
        return this.next;
    }

}
```

```java
public interface Operation {
    public String calcular(String xml, List<ItemXml> itensADevolver);

    public Operation setNext(Operation next);

    public void setXmlBase(String xml);

    public String getXmlBase();
}
```

```java
public class XMLService {

    public ResponseEntity<CalculatedXmlResponse> calculateXml(String xml, List  <ItemXml> items) {
        XmlOperation chain = new RemoverProdNaoDevolvidos(); //ok
        chain.setNext(new ZerarTotalXML()) //ok
            .setNext(new IniciarComValorDeTodosProdutosEmTotal()) //ok
            .setNext(new QuantidadeDevolvidaOperation()) // atualiza quantidade comprada para qtd
            .setNext(new FreteOperation()) //calcula frete
            .setNext(new DescontoOperation()) // calcula desconto
            .setNext(new ICMSOperation()) // calcula icms
            .setNext(new IPIOperation()) //calcula ipi
            .setNext(new BaseCalculoIcmsStOperation()) //calcula base de calculo do icms st
            .setNext(new FecopOperation()) // calcula fecop
            .setNext(new ICMSStOperation()); //calcula icms st

        CalculatedXmlResponse calculatedXmlResponse = new CalculatedXmlResponse();
        String returnedXml = chain.calcular(xml, items);
        calculatedXmlResponse.setXml(returnedXml);

        return ResponseEntity.ok().body(calculatedXmlResponse);
    }

}
```

Princípio da Segregação de Interface (ISP):

- **Definição:** Uma classe não deve ser forçada a implementar interfaces que ela não utiliza.
- **Dica:**
  - Se você utiliza interfaces, certificar-se de que as classes implementem apenas os métodos relevantes.

Violação do princípio ISP

Antes

```
export interface iRepositoryEstoque{
    create(produto: EstoqueDto);
    update(id: number, data: Prisma.EstoqueUpdateInput)
    findAll()
    findInEstoque(id: number);
    delete(id: number);
}
```

Depois

```
export interface iRepositoryEstoqueCreate{
    create(produto: EstoqueDto);
}

export interface iRepositoryEstoqueUpdate{
    update(id: number, data: Prisma.EstoqueUpdateInput)
}

export interface iRepositoryEstoqueFindAll{
    findAll()
}

export interface iRepositoryEstoqueFindInEstouq{
    findInEstoque(id: number);
}

export interface iRepositoryEstoqueDelete{
    delete(id: number);
}
```

Princípio da Inversão de Dependência (DIP):

- **Definição:** Dependa de abstrações, não de implementações concretas.
- **Exemplo:**
  - As dependências foram injetadas na classe `veiculoController` por meio do construtor.
  - **Sugestão:**
    - Definir a interface `veiculoController` de forma a seguir a Inversão de Controle (IoC) para maior flexibilidade e testabilidade.

Classe veiculoController

Antes e Depois

```
@RequestMapping(value = "/veiculo/{id}", method = RequestMethod.PUT)
public ResponseEntity<Veiculo> Putveiculo(@PathVariable(value = "id") Long id, @Validated @RequestBody Veiculo newVeiculo)
{
    Optional<Veiculo> oldVeiculo = veiculoRepository.findById(id);
    if(oldVeiculo.isPresent()){
        Veiculo veiculo = oldVeiculo.get();
        veiculo.setCor(newVeiculo.getCor());
        veiculo.setModelo(newVeiculo.getModelo());
        veiculo.setPlaca(newVeiculo.getPlaca());
        veiculo.setPlaca(newVeiculo.getPlaca());
        veiculoRepository.save(veiculo);
        return new ResponseEntity<Veiculo>(veiculo, HttpStatus.OK);
    }
    throw new ResponseStatusException(
            HttpStatus.NOT_FOUND, "veiculo não encontrado."
    );
}
```

```java
@PutMapping(value = "/veiculo/{id}")
public ResponseEntity<Veiculo> Putveiculo(@PathVariable(value = "id") Long id, @Validated @RequestBody Veiculo newVeiculo)
{
    Veiculo veiculo = veiculoService.Putveiculo(id, newVeiculo);
    if(veiculo == null) {
        throw new ResponseStatusException(
                HttpStatus.NOT_FOUND, "Veiuculo não encontrada!"
        );
    }

    return ResponseEntity.ok().body(veiculo);
}
```

Classe veiculoService

Antes e Depois

```
@RequestMapping(value = "/veiculo/{id}", method = RequestMethod.PUT)
public ResponseEntity<Veiculo> Putveiculo(@PathVariable(value = "id") Long id, @Validated @RequestBody Veiculo newVeiculo)
{
    Optional<Veiculo> oldVeiculo = veiculoRepository.findById(id);
    if(oldVeiculo.isPresent()){
        Veiculo veiculo = oldVeiculo.get();
        veiculo.setCor(newVeiculo.getCor());
        veiculo.setModelo(newVeiculo.getModelo());
        veiculo.setPlaca(newVeiculo.getPlaca());
        veiculo.setPlaca(newVeiculo.getPlaca());
        veiculoRepository.save(veiculo);
        return new ResponseEntity<Veiculo>(veiculo, HttpStatus.OK);
    }
    throw new ResponseStatusException(
            HttpStatus.NOT_FOUND, "veiculo não encontrado."
    );
}
```

```java
@Service
public class VeiculoService {

        @Autowired
        private VeiculoRepository veiculoRepository;

        public List<Veiculo> Getveiculo(){
            return veiculoRepository.findAll();
        }

        public Veiculo GetveiculoById(Long id){
           return veiculoRepository.findById(id).orElse(null);
        }



        public Veiculo Postveiculo(Veiculo veiculo){
            return veiculoRepository.save(veiculo);
        }

        public Veiculo Putveiculo(Long id, Veiculo newVeiculo)
        {
            if(!veiculoRepository.existsById(id)) return null;
            return veiculoRepository.saveAndFlush(newVeiculo);
        }

        public Veiculo Deleteveiculo( Long id){
            Optional<Veiculo> veiculo = veiculoRepository.findById((id));
            if (!veiculo.isPresent()) return null;
            veiculoRepository.delete(veiculo.get());
            return veiculo.get();
        }
}
```

Classe veiculoRepository

```java
@Repository
public interface OperadorRepository extends JpaRepository<Operador, Long> {
    Operador findByEmail(String email);
}
```