UNIVERSIDADE FEDERAL DE ALAGOAS –
UFAL CAMPUS A. C. SIMÕES

Ciência da Computação

Programação Orientada a Objetos 2023.2- Professor Mário Hozano

José Victor Dias da Silva Oliveira

Relatório WePayU

Fevereiro de 2024

1. Descrição Geral do Design Arquitetural

1.1. Facade

A arquitetura de Facade é um padrão de design de software que fornece uma interface simplificada para um conjunto complexo de classes, bibliotecas ou subsistemas. Ele envolve encapsular um grupo de interfaces de um sistema em uma única interface de nível mais alto. O objetivo principal é fornecer uma interface unificada e simplificada para um conjunto de funcionalidades mais complexas, facilitando assim o uso e reduzindo a complexidade para os clientes que interagem com o sistema.

2. Principais Componentes e Interações

2.1. ControladorEmpregados.java

A classe ControladorEmpregados unifica toda a lógica do programa e é a classe mais chamada na Facade. É nela também que o armazenamento interno do programa é feito. No LinkedHashmap<String idEmpregado, Empregado>, no ArrayList<Cartao> cartoes, no ArrayList<Venda> vendas, no ArrayList<TaxaServico> taxas e no LinkedHashMap<String, Sindicato> sindicatos estão as informações que são usadas para o funcionamento do sistema. Além disso, o ArrayList<Empregado> empregadosPersistencia foi usado para cobrir algumas dificuldades para armazenar o HashMap no .xml.

2.2. Empregado.java

```
public class Empregado {
    private String nome;
    private String endereco;
    private String tipo;
    private String salario;
    private boolean sindicalizado;
    private String id;
    private Sindicato sindicato;
    protected String metodoPagamento;
    protected Banco banco;
```

A classe principal possui em sua maioria atributos com tipos primitivos, além dos atributos banco, que guarda o objeto do tipo banco(nome, agencia, contaCorrente) de cada empregado e o atributo sindicato(idSindicato, taxaSindical).

O construtor de Empregado exige somente os atributos mais básicos e conforme os testes vão se complexificando e exigindo outros atributos, eles são ajustados com seus respectivos setters.

```
    public Empregado(String nome, String endereco, String tipo,
String salario){
        //public Empregado(String nome, String endereco, String
tipo, float salario) throws EmpregadoNaoExisteException {
            this.nome = nome;
            this.endereco = endereco;
            this.tipo = tipo;
            this.salario = salario;
            this.sindicalizado = false;
            this.id = null;
            this.sindicato = null;
            this.metodoPagamento = null;
            this.banco = null;
        }
```

2.3. Comissionado.java

```
    public class Comissionado extends Empregado{
        private String comissao;

        public Comissionado(String nome, String endereco,
String tipo, String salario, String comissao){
        super(nome, endereco, tipo, salario);
        this.comissao = comissao;
        }
```

A classe Comissionado apenas herda Empregado acrescentando o atributo comissao, na reta final da implementação houve alguns erros envolvendo casting para a classe Empregado que serão resolvidos futuramente.

2.4. Cartao.java

```
public class Cartao {
    private String id;
    private LocalDate data;
    private double horas;
```

Cartao implementa o cartão de ponto, a partir dessa classe comecei a guardar os atributos com tipos mais coerentes e versáteis, algo que provavelmente colocarei em prática na classe Empregado futuramente, mas que ainda não fiz por conta do deadline.

2.5. Venda.java

```
public class Venda {
    public String id;
    public LocalDate data;
    public double valor;
```

2.6. TaxaServico.java

```
public class TaxaServico {
    public String idSindicato;
    public LocalDate data;
    public double valor;
```

2.7. Sindicato.java

```
public class Sindicato {
    private String idSindicato;
   private double taxaSindical;
```

2.8. Banco.java

```
public class Banco {
    private String nome;
    private String agencia;
    private String contaCorrente;
```

3. Considerações Finais

Consegui desenvolver até a us5_1 sem erros, e as us6 e us6_1 com alguns erros que não pude corrigir a tempo. Acabei não usando as classes Assalariado e Horista nem as classes do package persistencia, porque não senti necessidade real de modularizar esses componentes pelas exigências dos testes. Contudo, pretendo implementá-las e tornar Empregado abstrata futuramente, bem como, armazenar os atributos de Empregado em tipos mais apropriados para cada atributo.