UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO (IC)

YUCK ARTHUR FERREIRA SILVA

Maceió-AL, 12/09/2025

Yuck Arthur Ferreira Silva

Documentação WePayU

Trabalho apresentado ao professor Mario Hozano da disciplina de Programação 2, da Universidade Federal de Alagoas.

Campus A.C. Simões
12/09/2025

I. Introdução

O projeto foi desenvolvido em Java e utiliza o padrão de arquitetura em camadas para separar a lógica de negócios, o modelo de dados e a interface de acesso (neste caso, uma Facade para testes).

O objetivo é fornecer uma visão clara da estrutura do código, facilitando a manutenção e a adição de novas funcionalidades.

II. Arquitetura Geral

O sistema é organizado nos seguintes pacotes principais:

- **br.ufal.ic.p2.wepayu**: pacote raiz que contém as classes de orquestração principal (SistemaFolha) e a fachada (Facade).
- **br.ufal.ic.p2.wepayu.models**: contém as classes que representam as entidades do sistema (POJOs - Plain Old Java Objects), como Empregado, CartaoDePonto, MetodoPagamento, etc.
- **br.ufal.ic.p2.wepayu.Services**: contém as classes de serviço que encapsulam a lógica de negócios do sistema, como gerenciamento de empregados e processamento da folha de pagamento.
- **br.ufal.ic.p2.wepayu.Exception**: contém as exceções personalizadas para tratamento de erros específicos do domínio do sistema.

III. Descrição dos Pacotes e Classes Principais

III.I. Pacote br.ufal.ic.p2.wepayu

Facade.java

- **Responsabilidade:** servir como uma camada de interface simplificada para o sistema. É o único ponto de entrada para a ferramenta de testes EasyAccept.
- **Funcionamento:** cada método nesta classe corresponde a um comando definido nos scripts de teste (ex: criarEmpregado, lancaCartao). Os métodos da Facade recebem os parâmetros como Strings, fazem as validações iniciais e delegam a execução para a classe SistemaFolha.

SistemaFolha.java

- **Responsabilidade:** é o núcleo do sistema, orquestrando todas as operações e mantendo o estado da aplicação (a lista de empregados, o próximo ID, etc.).

- **Principais Funcionalidades:**
    - **Gerenciamento de Estado:** contém o Map de empregados e controla a persistência dos dados.
    - **Lógica de Negócios:** contém os métodos que são chamados pela Facade para executar as principais ações, como criar empregados, lançar vendas, rodar a folha, etc.
    - **Undo/Redo:** implementa o padrão Memento para salvar e restaurar o estado do sistema, permitindo desfazer e refazer operações.

III.II. Pacote br.ufal.ic.p2.wepayu.models

Este pacote define as estruturas de dados do sistema.

- **Empregado.java (Abstrata):** classe base para todos os tipos de empregados. Contém atributos comuns como nome, endereco, metodoPagamento e afiliação sindical (MembroSindicato).
- **EmpregadoHorista.java:** herda de Empregado. Representa um empregado que recebe por hora. Contém uma lista de CartaoDePonto.
- **EmpregadoAssalariado.java:** herda de Empregado. Representa um empregado com salário mensal fixo.
- **EmpregadoComissionado.java:** herda de EmpregadoAssalariado. Representa um empregado assalariado que também recebe comissão sobre as vendas. Contém uma lista de ResultadoDeVenda.
- **MetodoPagamento.java (Abstrata):** classe base para as diferentes formas de pagamento.
    - **Implementações:** Banco.java, EmMaos.java, Correios.java.
- **MembroSindicato.java:** armazena informações de um empregado sindicalizado, como idSindicato, taxaSindical e uma lista de TaxaServico.
- **Outros Modelos:** CartaoDePonto.java, ResultadoDeVenda.java, TaxaServico.java são classes simples que armazenam dados de lançamentos.

III.III. Pacote br.ufal.ic.p2.wepayu.Exception

Contém exceções personalizadas que herdam de WePayUException (que por sua vez herda de Exception). Isso permite um tratamento de erros mais específico e mensagens claras para o usuário.

- Exemplos: EmpregadoNaoExisteException.java, SalarioInvalidoException.java, ComissaoInvalidaException.java, etc.

IV. Funcionalidades Detalhadas

IV.I. Persistência de Dados

- **Mecanismo:** A persistência é realizada através da serialização de objetos para o formato XML.
- **Classes:** java.beans.XMLEncoder e java.beans.XMLDecoder são usadas para salvar e carregar o estado do sistema.
- **Arquivo:** O estado é salvo no arquivo data.xml na raiz do projeto.
- **Gatilhos:**
    - carregarSistema(): Chamado no construtor de SistemaFolha para carregar os dados do data.xml.
    - encerrarSistema(): Chamado para salvar o Map de empregados e o proximoId no data.xml.

IV.II. Padrão de Projeto: Memento (undo/redo)

- **Objetivo:** permitir que o estado do sistema seja salvo antes de uma operação e restaurado posteriormente.
- **Implementação:**
    1. **SistemaFolhaMemento.java:** o "Memento", uma classe que armazena uma cópia profunda do Map de empregados e do proximoId.
    2. **SistemaFolha.java:** o "Originator", que cria (createMemento) e restaura (restoreState) mementos.
    3. **Pilhas undoStack e redoStack:** o "Caretaker", que gerencia o histórico de estados.
- **Fluxo:**
    1. Antes de qualquer operação que modifique o estado (criar, remover, alterar), o método saveState() é chamado.
    2. saveState() cria um memento do estado atual e o empilha na undoStack. A redoStack é limpa.
    3. Ao chamar undo(), o estado atual é salvo na redoStack, e o estado do topo da undoStack é restaurado.
    4. Ao chamar redo(), o processo inverso ocorre.

IV.III. Cálculo da Folha de Pagamento

A lógica de cálculo está centralizada nos métodos rodaFolha e calcularPagamento da classe SistemaFolha.

- **ehDiaDePagamento(empregado, data):** verifica se um determinado empregado deve ser pago na data fornecida, com base no seu tipo e agenda de pagamento.
- **calcularPagamento(empregado, dataFim):**
    1. Calcula o período de pagamento (desde o último pagamento até dataFim).

2. Calcula o salário bruto com base no tipo do empregado:
    - **Horista:** soma as horas normais e extras dos cartões de ponto no período.
    - **Assalariado:** usa o salário mensal fixo.
    - **Comissionado:** calcula o salário base proporcional (quinzenal) e soma as comissões das vendas no período.
3. Calcula os descontos (taxa sindical e taxas de serviço) no período.
4. Retorna um array de `Object` com todos os valores calculados (bruto, descontos, líquido, etc.).

- **`rodaFolha(data, saida)`:**
    1. Filtra os empregados que devem ser pagos na data.
    2. Chama `calcularPagamento` para cada um deles.
    3. Usa os métodos `gerarSecaoHoristas`, `gerarSecaoAssalariados`, e `gerarSecaoComissionados` para formatar e escrever a saída no arquivo de texto especificado.