UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMAÇÃO 2
THIAGO RIBEIRO DA SILVA

RELATÓRIO: WEPAYU - MILESTONE 1

MACEIÓ/AL
2024

SUMÁRIO

1. INTRODUÇÃO

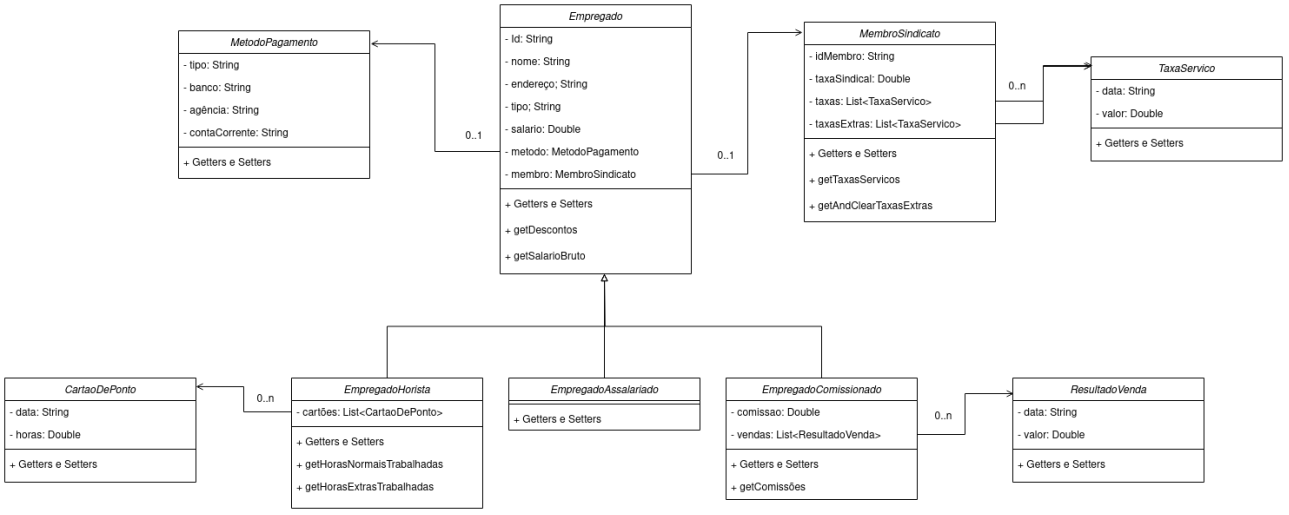
O *WePayU* é um sistema simples para administração de pagamentos a empregados de uma empresa, que possibilita a adição, leitura, alteração e remoção de usuários, bem como o lançamento de taxas, impostos e serviços relacionados para a geração de folhas de pagamento. Foi desenvolvido de forma incremental através da abordagem de Desenvolvimento Orientado a Testes (TDD). A persistência dos dados é realizada através de uma simulação de banco de dados com um arquivo XML. Deste modo, este relatório aborda o design arquitetural do sistema; os principais componentes que o definem e as escolhas de padrões de projeto utilizadas, com presença de diagramas para a melhor compreensão da proposta.

2. DESIGN ARQUITETURAL DO SISTEMA

A arquitetura do sistema é centralizada por meio de um *Facade,* que possibilita a unificação e simplificação dos subsistemas que compõem o projeto. Foi organizada em camadas, levando em conta as diferentes responsabilidades das classes para o correto funcionamento do *software* e possibilitando a manutenção, reutilização e escalabilidade. Deste modo, foram definidas as camadas *models*, *DAOs*, *services* e *exceptions*, que se comunicam direta e indiretamente com o *Facade*, e que serão descritas nos subtópicos.

2.1. *Models*

Os Modelos consistem na representação das classes principais do projetos, responsáveis pela estruturação dos empregados e demais objetos associados. Foram estruturados a partir de um diagrama de classes construido com base no modelo conceitual do escopo do projeto.



2.2. *DAOs*

Os *Data Access Objects* (DAOs) representam as estruturas de gerenciamento dos modelos, que possibilitam a criação, leitura, alteração e remoção de instâncias da base de dados. Desta forma, se comunicam com o repositório local e possibilitam a manipulação e manutenção segura dos objetos.

2.3. *Services*

Os serviços lidam com o gerenciamento geral do sistema, incluindo o controle da persistência de dados por meio de um arquivo XML, a determinação de constantes que representam as configurações do sistema, um construtor para geração das folhas de pagamento e métodos com utilidades relacionadas a verficação e tratamento de dados recebidos e enviados.

2.4. *Exceptions*

No contexto das exceções, criou-se um espaço dedicado para lidar com todas as possíveis situações excepcionais que o sistema pode encontrar durante sua execução. Isso proporciona uma gestão mais eficaz e centralizada dos erros, facilitando a manutenção e o debugging.

3. PRINCIPAIS COMPONENTES

Através das camadas, certos componentes se mostram essenciais para a compreensão do funcionamento do sistema, de modo que possibilitam a manutenção, gerenciamento e construção de elementos fundamentais para o objetivo da aplicação. Estes componentes serão descritos nos subtópicos seguintes.

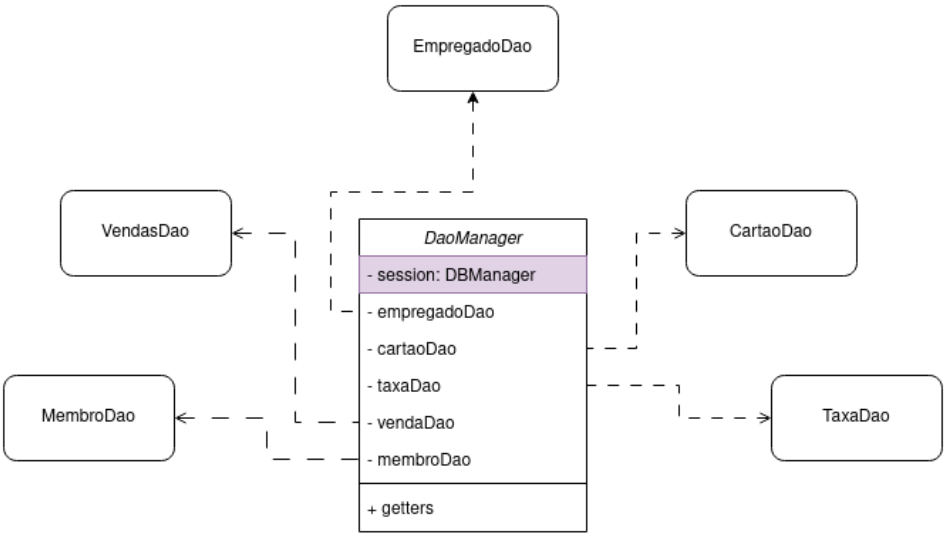
3.1. Empregado

Consiste no componente central que relaciona todos os demais modelos do projeto, representando a instância de um empregado da empresa. O mesmo se subdivide em Horista, Assalariado e Comissionado, a depender da maneira com a qual recebe pagamentos pelos seus serviços. Empregados também seguem relacionados a um método de pagamento, padronizado como “em Mãos”, mas podendo ser alterado. Bem como pode estar associado ao Sindicato (único para

esse protótipo), tendo assim um imposto sindical e a possibilidade de solicitar serviços associados a taxas.

3.2. DaoManager

Representa o gerenciamento de todos os DAOs do projeto, fornecendo instâncias para uma melhor centralização do uso desses objetos durante a manipulação dos dados. Segue um diagrama que representa a configuração desse componente.



3.3. DBManager

Simula um banco de dados, persistindo e gerenciando os dados salvos em XML através de uma instância única (Singleton). Os seguintes métodos compõem esse objeto:

- **getSession():** Retorna uma instância única do *Manager*
- **clearAll():** Limpa os dados da estrutura de armazenamento da classe e apaga o XML
- **getAllData():** Lê os dados do XML e escreve na estrutura de armazenamento da classe
- **query():** Retorna a estrutura de armazenamento da classe
- **commit():** Sobrescreve os dados armazenados dentro do XML
- **add(Empregado):** Adiciona um novo empregado ao armazenamento da classe
- **update(id, Empregado):** Atualiza um empregado por meio do id

3.4. FolhaBuilder

Constrói a folha de pagamento de forma sequencial, através de métodos para geração de partes da folha e um método que une todos os geradores, montando o arquivo com os dados dos pagamentos. Os métodos geradores acessam templates (txt) com os cabeçalhos referentes aos diferentes tipos de empregados e preenchem as informações de maneira ordenada. Seguem as descrições do métodos do componente:

- **getTotalFolha(data):** Soma os sálarios brutos de todos os empregados referentes a data;
- **geraDadosHoristas(data):** Lê template de horistas, escreve dados dos horistas referentes a data e o total de pagamento;
- **geraDadosAssalariados(data):** Lê template de assalariados, escreve dados dos assalariados referentes a data e o total de pagamento;
- **geraDadosComissionados(data):** Lê template de comissionados, escreve dados dos comissionados referentes a data, o total de pagamento e o total da folha;
- **geraFolha(data):** Escreve o título da folha de pagamento referente a data e chama os geradores parciais.

3.5. Utils

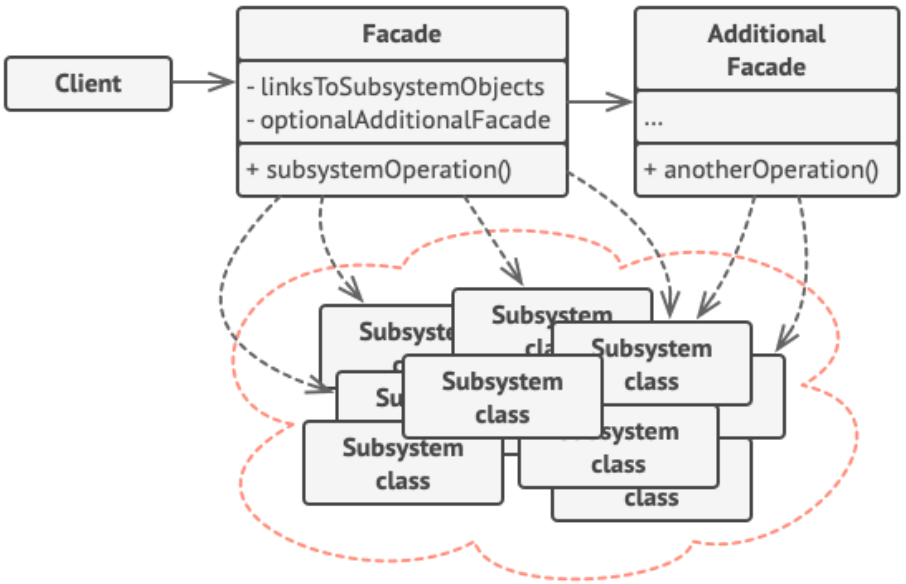
Mantém métodos voltados para a verificação, formatação e recebimento de dados, que possibilitam o bom funcionamento do código e o tratamento de exceções para entradas inválidas ou inconsistentes. Possui métodos para conversão de valores númericos em textuais, verificação da validade de atributos, tipos, checagem de datas e dias referentes ao pagamento dos empregados.

4. PADRÕES DE PROJETO

4.1. Facade

Descrição geral: O *Facade* fornece uma interface unificada simplificada para um conjunto de interfaces em um subsistema. Ele encapsula um conjunto complexo de classes e fornece uma interface mais amigável para o cliente, ocultando a complexidade interna do sistema.

Problema resolvido: Resolve problemas relacionados a existência de subsistemas complexos, com muitas partes interdependentes, onde o cliente precisa interagir com ele de forma simples. Desde modo, fornece uma fachada que abstrai as complexidades e permite um acesso facilitado aos processos dos subsistemas.

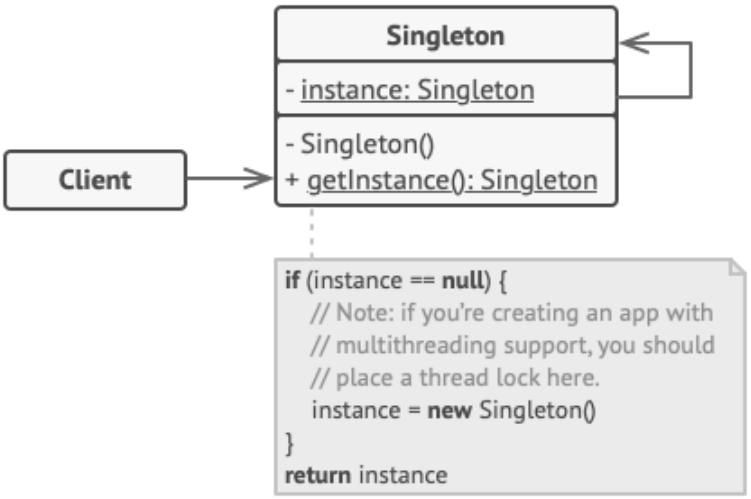


Identificação da oportunidade e aplicação no projeto: Devido a quantidade de modelos a serem adminstrados e aos processos de verficação e formatação de dados, torna-se necessário, neste projeto, a criação de uma estrutura que abstraia essas complexidades e funcione como uma entrada para o acesso aos processos do sistema. Sendo assim, o *Facade* foi responsável por possuir os métodos que representam os requisitos funcionais do sistemas, envolvendo a criação, alteração, remoção, leitura de empregados e seus dados relacionados, além da geração da folha de pagamento.

4.2. Singleton

Descrição geral: O padrão *Singleton* garante que uma classe tenha apenas uma instância e fornece um ponto de acesso global a essa instância. Ele geralmente envolve a criação de uma classe com um método estático que retorna a mesma instância sempre que é chamado.

Problema resolvido: Pela própria descrição do padrão, fica claro que o mesmo é útil para a manutenção de um único estado de acesso a uma classe. Deste modo, ele possibilta um controle estrito de instâncias globais e evita a múltiplicação de uma informação na memória, podendo resultar em overload e erros de conflito de estados.

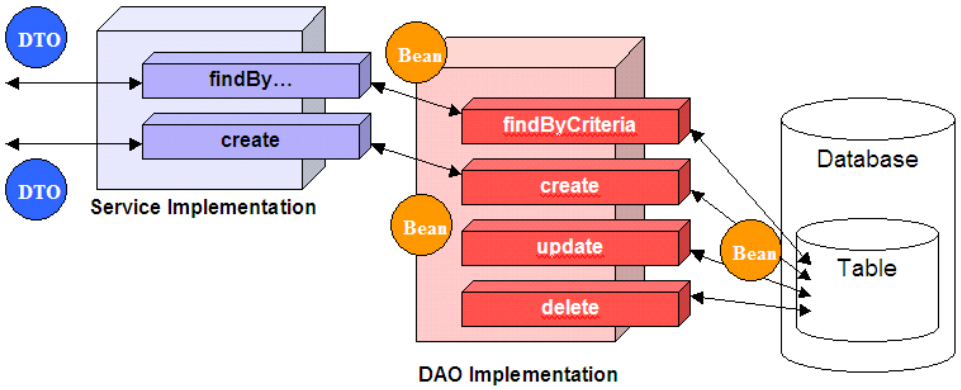


Identificação da oportunidade e aplicação no projeto: Durante o projeto, os DAOs que gerenciam os diferentes modelos necessitam, individualmente, do acesso ao banco de dados (DBManager) para realizar as atualizações e leituras do XML. Sendo assim, é criada uma instância única de acesso ao banco e transmitida para todos os gerenciadores.

4.3. DAO (Data Access Object)

Descrição geral: O DAO (Data Access Object) encapsula o acesso a um banco de dados, fornecendo uma interface abstrata para operações de CRUD (Create, Read, Update, Delete). Isso torna o código da aplicação mais independente da implementação específica do banco de dados.

Problema resolvido: De modo geral, o uso desse padrão permite separar a lógica de negócios da lógica de acesso aos dados, abstraindo o acesso a eles e centralizando os processos de manipulação dos objetos.

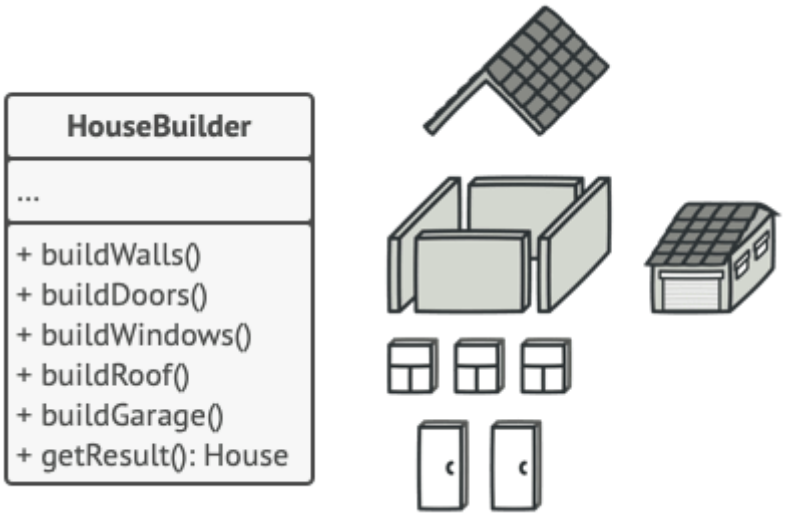


Identificação da oportunidade e aplicação no projeto: Devido a necessidade de CRUDs para os modelos do WePayU, fez-se necessário e essencial criar DAOs de manipulação dos dados, que juntos a lógica de simulação de banco de dados, facilitaram muito o gerenciamento dos objetos nos diferentes processos requeridos no projeto.

4.4. Builder

Definição geral: Builder fornece uma maneira de construir objetos complexos passo a passo. Isso torna o código da aplicação mais fácil de ler e entender, pois a construção do objeto é dividida em etapas menores. Essas etapas são mescladas e podem resultar em diferentes estruturas a depender da ordem e dos atributos utilizados.

Problema resolvido: De modo geral, resolve problemas relacionados a criação de objetos complexos, que demanda uma separação em passos menores que podem ser unidos posteriormente.



Identificação da oportunidade e aplicação no projeto: A geração de folha de pagamento se tornou um processo que demandou complexidade na geração dos textos. Sendo assim, foi necessária a separação em blocos geradores das partes da folha , que foram unidos em uma método de geração final.