UNIVERSIDADE FEDERAL
DE ALAGOAS

Universidade Federal de Alagoas
Instituto de Computação

Rafael Luciano Lima da Silva
Vinícius da Costa Neitzke

Relatório do Projeto WePayU

Maceió
2024

Design arquitetural

O design arquitetural do sistema é formado por quatro camadas:

Camada de Lógica de Negócios

Esta camada contém a lógica central do sistema, sendo responsável por definir a manipulação de todas as operações e dados relacionados aos empregados, tais como seu tipo ou métodos de pagamento.

Dentro do diretório “src”, a classe Facade – que utiliza um padrão de projeto de mesmo nome – faz parte desta camada, fornecendo uma interface menos complexa para interagir com o sistema, permitindo a execução de todos os métodos presentes nele.

Também faz parte da camada a classe Sistema, que inclui a lógica de adição de empregados no sistema, utilizando a estrutura de dados HashMap.

Camada de Acesso a Dados

Como recomendado no arquivo de descrição do projeto, o sistema está utilizando a linguagem XML para persistir os dados manipulados. Eles serão armazenados de forma compartilhada, persistindo tanto nas etapas de execução dos códigos durante os casos testes quanto após a execução deles. A adaptação ao XML se deu por salvar o conteúdo do HashMap dos empregados no arquivo XML logo após o encerramento do sistema, com isso, ao ser construída uma nova Facade, os dados presentes neste arquivo serão carregados.

Camada de Modelagem de Dados

Dentro da pasta “br.ufal.ic.p2.wepayu/models” estão localizadas as classes que modelam os objetos de domínio do sistema, como Empregado e suas subclasses (EmpregadoAssalariado, EmpregadoComissionado, EmpregadoHorista), além de

outras classes como MetodoPagamento, PointCard, TaxaServico, etc. Essas classes representam os dados manipulados pelo sistema e integram a camada de modelo.

Camada de Exceções

No diretório “br.ufal.ic.p2.wepayu/models/Exception”, estão localizadas as classes de exceções personalizadas que lidam com erros específicos do sistema. Essas classes tratam as situações excepcionais e garantem a robustez do projeto.

No geral, o trabalho segue uma arquitetura de quatro camadas, com a lógica de negócios sendo o núcleo do sistema e as outras camadas interagindo com ela para oferecer as funcionalidades necessárias.

Principais componentes e suas interações

Os principais componentes do sistema são:

Classes do empregado

As classes do empregado são feitas para representar os diferentes tipos de empregados no sistema, onde cada um possui seus próprios atributos e métodos específicos. Na classe pai “Empregado” são definidos os atributos básicos “nome”, “endereço” e “metodoPagamento”, bem como os métodos getters e setters desses atributos e getters e setters abstratos para atributos que são definidos somente de acordo com o tipo de “Empregado”, tais como “salario” e “tipo”. Também é definido um get para a checagem se o membro faz parte do sindicato.

Nas classes filhas “EmpregadoAssalariado”, “EmpregadoComissionado”, “EmpregadoHorista” os métodos abstratos da classe pai são definidos de acordo com os requisitos de cada classe, sendo também definidos outros atributos, junto com seus getters e setters, que estão presentes em apenas um tipo de empregado, como o “salarioPorHora” e o “cartao” no caso do horista, “salarioMensal” para comissionados e assalariados, e “comissao” e “vendas” para os comissionados.

Classes do pagamento

As classes de pagamento são utilizadas para armazenar os dados relacionados à remuneração dos empregados. Ao ser cadastrado no sistema, o empregado inicia o método de pagamento no estado “em mãos” e pode alterá-lo de acordo com seu desejo. Os métodos de pagamento “em mãos” e “correios” utilizam apenas as informações já pré estabelecidas na criação de um empregado, como seu endereço, assim, não são necessárias desenvolver as classes “EmMaos” e “Correios”. Outro método de pagamento é pelo banco, para isso, existe a classe Banco, que contém atributos que guardam as informações acerca da conta bancária de um empregado (banco, agência, conta corrente), bem como seus métodos getters e setters.

Classes de serviço

Classes de serviço são aquelas que estão vinculadas com a realização de tarefas e atribuição de horas e taxas por elas. Uma dessas classes é a “CartaoDePonto”, que terá atributos da quantidade de horas que um empregado trabalhou atrelada ao dia em que ele o fez, bem como seus métodos getters. Outras duas classes são o “ResultadoDeVenda”, que irá guardar o valor da venda e sua data e a “TaxaServico” também guardando os mesmos atributos, onde o valor é relacionado à taxa que deve ser paga ao sindicato. E por último, há a classe “MembroSindicado”, que terá como atributos o id do membro, sua taxa sindical e uma lista de taxas de serviço.

Classe Sistema

Como falado na seção de design arquitetural, a classe sistema é responsável por manter uma coleção de empregados utilizando um HashMap estático. Possui métodos getters e setters para manipular os dados presentes diretamente no HashMap, onde utilizam-se dos métodos definidos nas classes das seções acima. Exemplos de métodos presentes: “getEmployeeAttribute”, que retorna o atributo do empregado pelo seu id**;** “getWorkedHours”, que retorna a quantidade de horas trabalhadas por um empregado**;** e métodos que mudam o tipo de um empregado, como “changeEmployeeTypeToHorista”.

Padrões de projeto adotados

Facade

Descrição geral

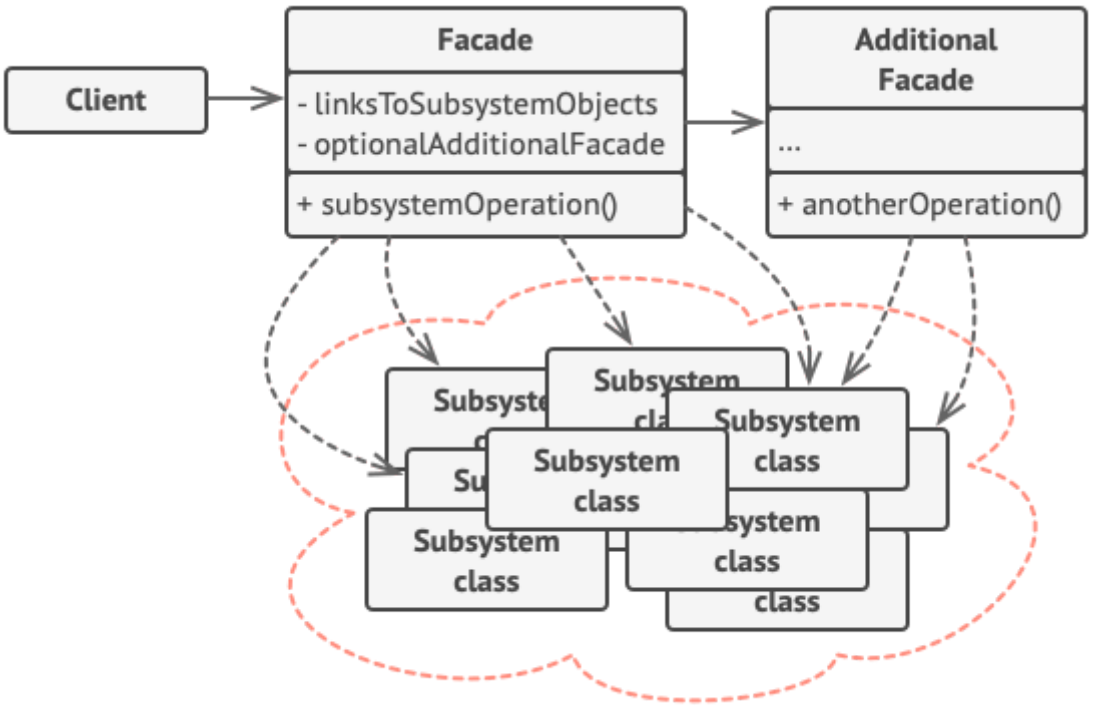
Uma Facade é um padrão de projeto que dispõe uma área/interface mais simplificada para interagir com o sistema, ocultando a complexidade dos componentes do sistema e melhorando a usabilidade do trabalho para os clientes.

De acordo com o site Refactoring Guru em 2014, a Facade estabelece a seguinte estrutura:

“A Facade fornece um acesso à uma parte da funcionalidade do subsistema e direciona o pedido do cliente, operando em todas as partes móveis.

O Subsistema consiste em dúzias de objetos variados. As classes do subsistema não estão cientes da existência da fachada. Elas operam dentro do sistema e trabalham entre si diretamente.

O Cliente usa a Facade ao invés de chamar os objetos do subsistema diretamente.”



Fonte: Refactoring Guru

Disponível em: <https://refactoring.guru/pt-br/design-patterns/facade>. Acesso em: 8 fev. 2024.

Problema resolvido

Em sistemas que possuem uma grande quantidade de componentes e interações, tornaria-se muito complexo criar, agrupar e manipular todos os objetos juntamente com seus métodos de maneira lógica e usual, respeitando as dependências entre os mesmos. Então, faz-se necessário a criação de um espaço que consiga solucionar tais desafios.

Identificação da Oportunidade

No sistema, foi notado que havia muitas classes, resultando em diversos métodos que ao decorrer do desenvolvimento deveriam se relacionar, o que aumentaria em grande escala a dificuldade de construir o projeto de uma maneira logicamente correta e eficiente. Logo, para amenizar tal situação, descobriu-se a existência do padrão de projeto que possibilitou esclarecer a visão acerca do trabalho e solucionar o desafio.

Aplicação no projeto

No projeto foi implementada uma classe Facade, que oferece uma área mais simplificada para interagir com o sistema. Nela todos os métodos presentes nas classes descritas na seção anterior foram utilizados para calcular e desenvolver métodos de CRUD (Create, Read, Update, Delete) no sistema.

Como exemplo, temos os métodos de criação, remoção e update de um empregado: “criarEmpregado”, “removerEmpregado”, “alteraEmpregado”**;**

Métodos de leitura de informações acerca do empregado, onde também irão utilizar métodos de classes fora das classes de empregados para calcular e retornar o dado desejado: “getAtributoEmpregado”, “getHorasNormaisTrabalhadas”, “getHorasExtrasTrabalhadas”**;**

Método que verifica se um empregado está associado ao sindicato: “idSindicadoExiste”**;**

Método que checa se um empregado comparece ao serviço, servindo para calcular a quantidade de horas trabalhadas: “lancaCartao”**;**

Métodos relacionados à realização de serviços: “lancaVenda”, “getVendasRealizadas”, “lancaTaxaServico”, “getTaxasServico”**;**

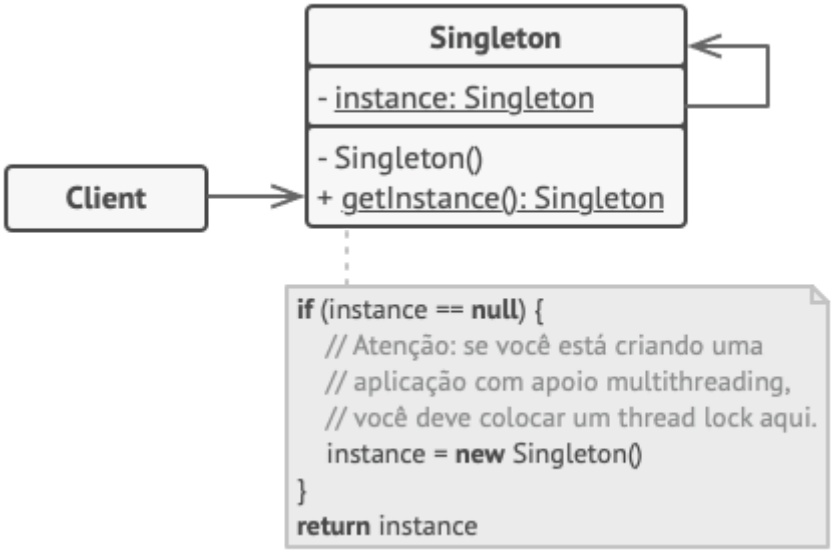
E por fim, operações que apagam ou encerram o sistema: “zerarSistema”, “encerrarSistema”.

Singleton

Descrição geral

Singleton é um padrão de projeto que consiste em criar apenas uma instância de uma classe, não sendo permitido criar objetos adicionais dessa classe, e provendo um ponto de acesso global à ela.

A estrutura do padrão consiste em uma classe permitir ser instanciado apenas um objeto, possuindo um método “getInstance()”, que retornará a instância da própria classe, sendo esse método a única forma de obter o objeto singleton (Refactoring Guru, 2014).



Fonte: Refactoring Guru
Disponível em: <https://refactoring.guru/pt-br/design-patterns/singleton>. Acesso em: 9 fev. 2024.

Problema resolvido

Garante que uma classe tenha somente um objeto, controlando o acesso a algum recurso compartilhado, sendo usada apenas quando necessário. Além disso, protege a instância de ser sobrescrita por outro código.

Identificação da Oportunidade

Ao ter que implementar a persistência de dados utilizando XML, como haveria um acesso compartilhado dos dados, identificou-se a necessidade de evitar a sobrescrição do arquivo e também evitar gerar novas instâncias para o armazenamento das informações, tornando o processo complexo e ineficiente.

Aplicação no projeto

Para garantir a persistência dos dados, foi desenvolvido o método “getInstance”, que segue os conceitos de Singleton, garantindo que caso não exista uma instância da Classe Database, uma nova instância será criada e retornada pelo método. Tal instância é responsável em conectar o sistema com os dados salvos no formato XML.