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

Ciência da Computação

Programação Orientada a Objetos 2023.2- Professor Mário Hozano

Gabriela Batista Tenório das Neves

Caio Mascarenhas Soares

Relatório WePayU

Fevereiro de 2024

● Projeto

- Nome do Padrão de Projeto = Facade
- Descrição Geral: A classe facade é um design patterns onde todos os métodos são criados mas a implementação dos mesmos não é implementada na facade e sim em outro método que é chamado nele.
- Problema Resolvido: Esse projeto resolve um problema de folha de pagamento de usuários, onde um usuário é criado, atribuído, e pode ser lançado e pegado seu cartão de ponto, suas vendas, sua taxa sindical, suas horas trabalhadas e etc. O programa salva em arquivo XML os dados de cada empregado gerando um ID único e dai resolve o problema específico.
- Identificação da Oportunidade: O uso do padrão facade foi de extrema importância ja que o projeto tem muitos métodos, muitas classes e muitos objetos relacionados, o facade garante a organização e melhor leitura do código, sabendo onde está resolvendo cada problema.
- Aplicação no Projeto: O projeto pode ser facilmente usando em uma empresa que existe trabalhadores horistas, assalariados e comissionados. Ele pode ajudar e contribuir com lançamento de ponto, recebimento de salários e etc

● Descrição Geral do Design Arquitetural

O sistema é projetado como um aplicativo de folha de pagamento que gerencia informações sobre empregados, incluindo seus detalhes pessoais, tipo de emprego, salários, vendas realizadas, horas trabalhadas, taxa de serviço sindical, entre outros. O sistema permite a criação, atualização e remoção de empregados, bem como o lançamento de vendas e taxas de serviço sindical.

O padrão utilizado Facade fornece uma interface simplificada para um conjunto mais complexo de classes, tornando mais fácil para os clientes interagirem com o sistema. No nosso caso, a classe Facade fornece métodos simplificados que encapsulam a lógica de várias classes de controle.

São essas classes:

1. **public Facade()** throws FileNotFoundException {
   SistemaController**.iniciarSistema**();}

O construtor da classe Facade é responsável por iniciar o sistema. Ele chama o método iniciarSistema() da classe SistemaController. Esse método contém a lógica de inicialização, incluindo a inicialização do arquivo .XML para persistência de dados, usando o XMLDecoder.

2. public void **zerarSistema**() {
   SistemaController.zerarSistema();}

Este método chama o método zerarSistema() da classe SistemaController. A função desse método redefine o estado do sistema, limpando dadospara "zerar" o sistema, usando o XMLEncoder e método .clear() na lista que desejar zerar.

3. public void **encerrarSistema**() {
   SistemaController.encerrarSistema();}

Similar ao método anterior, este chama o método encerrarSistema() da classe SistemaController. Esse método contem a lógica para encerrar o sistema de maneira adequada, como salvando os dados com XMLEncoder no arquivo desejado , garantindo assim a persistência e o uso futuro desses.

4. public void **lancaTaxaServico**(String emp, String data, String valor) {
   ServicoController.lancaTaxaServico(emp, data, valor);}

Esse método chama o método lancaTaxaServico() da classe ServicoController. Ele é utilizado para lançar taxas de serviço para um empregado em uma determinada data.

5. public String **getTaxasServico**(String emp, String dataInicial, String dataFinal) {
   return ServicoController.getTaxasServico(emp, dataInicial, dataFinal);}

Este método chama getTaxasServico() da classe ServicoController e retorna as taxas de serviço para um empregado em um intervalo de datas, se houver.

6. public void **lancaVenda**(String emp, String data, String valor) {
   VendasController.lancaVenda(emp, data, valor);}

Este método chama o método lancaVenda() da classe VendasController. Serve para lançar informações(valor) sobre vendas realizadas por um empregado comissionado em uma determinada data.

7. public String **getVendasRealizadas**(String emp, String dataInicial, String dataFinal) {
   return VendasController.getVendasRealizadas(emp, dataInicial, dataFinal);}

Esse método chama getVendasRealizadas() da classe VendasController e retorna as vendas realizadas por um empregado comissionado em um intervalo de datas, se houver.

8. public String **getHorasExtrasTrabalhadas**(String emp, String dataInicial, String dataFinal) {
   return PontoController.getHorasExtrasTrabalhadas(emp, dataInicial, dataFinal);
   }

Este método chama getHorasExtrasTrabalhadas() da classe PontoController e retorna as horas extras trabalhadas por um empregado horista em um intervalo de datas.

9. public String **getHorasNormaisTrabalhadas**(String emp, String dataInicial, String dataFinal) {
   return PontoController.getHorasNormaisTrabalhadas(emp, dataInicial, dataFinal);
   }

   Semelhante ao método anterior, este chama getHorasNormaisTrabalhadas() da classe PontoController, retornando as horas normais trabalhadas por um empregado horista em um intervalo de datas, sendo 8 horas o máximo de horas em um único dia.

10. public void **lancaCartao**(String emp, String data, String horas) {
    PontoController.lancaCartao(emp, data, horas);
    }
    Este método chama lancaCartao() da classe PontoController. É utilizado para lançar informações sobre o cartão de ponto de um empregado horista, especificando a data e as horas trabalhadas.

11. public String **getAtributoEmpregado**(String emp, String atributo) {
    return EmpregadoController.getAtributoEmpregado(emp, atributo);
    }

    Este método chama getAtributoEmpregado() da classe EmpregadoController e retorna um atributo específico de um empregado, com base no Id do empregado e no nome do atributo.

12. public String **getEmpregadoPorNome**(String nome, int indice) {
    return EmpregadoController.getEmpregadoPorNome(nome, indice);
    }
    Semelhante ao método anterior, esse método chama getEmpregadoPorNome() da classe EmpregadoController e retorna o Id de um empregado com base no nome e no índice fornecidos, indice serva caso haja 2 ou mais pessoas com o mesmo nome.

13. public void **removerEmpregado**(String emp) {
    EmpregadoController.removerEmpregado(emp);
    }
    Este método chama removerEmpregado() da classe EmpregadoController e é utilizado para remover um empregado do sistema e da lista com base no Id do empregado.

14. public String **criarEmpregado**(String nome, String endereco, String tipo, String salario) {
    return EmpregadoController.criarEmpregado(nome, endereco, tipo, salario);
    }
    Esse método chama criarEmpregado() da classe EmpregadoController e é

utilizado para criar um novo empregado do tipo horista ou assalariado no sistema com base no nome, endereço, tipo e salário fornecidos. Esse método adiciona o empregado na lista que será salva no XML.

15. public String **criarEmpregado**(String nome, String endereco, String tipo, String salario, String comissao) {
    return EmpregadoController.criarEmpregado(nome, endereco, tipo, salario, comissao);
    }
    Semelhante ao método anterior, este método chama criarEmpregado() da classe EmpregadoController, mas permite a criação de um empregado do tipo comissionado com atributo adicional de comissão.

● Principais Componentes e Interações

- EmpregadoController: Responsável por gerenciar as operações relacionadas aos empregados, como criação, atualização e remoção.
- VendasController: Controla as operações de lançamento de vendas realizadas pelos empregados.
- PontoController: Gerencia o registro de horas trabalhadas pelos empregados, incluindo horas extras e horas normais.
- ServicoController: Responsável pelo lançamento de taxas de serviço sindical para os empregados.
- SistemaController: Controla o fluxo geral do sistema, inicializando-o, encerrando-o e realizando operações de inicialização.

● Estrutura do projeto

O projeto em java na pasta SRC que contem o programa principal foi dividido em demais pastas nomeadas Models, Controllers , Utils e Exception.

- Models: Contém a classe principal de cada entidade do problema: Empregado ,Venda,banco,Cartao de Ponto…
  As subclasses herdam atributos dessa classe, e nela contém os atributos comuns a todos da entidade e lá são setados, recebidos e guardados.
- Controller Contém os controladores dos problemas de cada entidade, os métodos chamados na Facade são implementados em seus respectivos controllers, é onde toda a lógica do problema está implementada.
- Exception: Contém as exceções que o problema retorna a cada erro.

- Utils: Alguns métodos não relacionados ao problema e sim a lógica, que são implementados separadamente para maior organização, por exemplo verificar se um data é válida.