Relatório do Projeto em Java

Descrição Geral do Design Arquitetural do Sistema

O sistema, construído em Java, tem o objetivo de possibilitar o gerenciamento de empregados (funcionários) que abrange diferentes aspectos relacionados a pagamento, vendas e sindicato. O design arquitetural do sistema segue uma abordagem modular, com componentes bem definidos para as funcionalidades específicas.

Principais Componentes e Suas Interações

O sistema é composto por diversos componentes, sendo os principais:

1. **Main Class (main.java):**
    - Responsável por iniciar o programa.
    - Interage com a classe Facade para chamar os métodos principais do sistema.
2. **Facade Class (br.ufal.ic.p2.wepayu.Facade):**
    - Funciona como uma fachada para o sistema, encapsulando a complexidade interna.
    - Coordenador das operações, interagindo com os controllers e gerenciadores de entidades.
3. **EmployeeController Class (br.ufal.ic.p2.wepayu.controllers.EmployeeController):**
    - Gerencia as operações relacionadas aos empregados (funcionários).
    - Lida com a criação, atualização, remoção e consulta de informações sobre os empregados.
4. **SaleController Class (br.ufal.ic.p2.wepayu.controllers.SaleController):**
    - Controla as operações relacionadas às vendas realizadas pelos empregados.
5. **SyndicateController Class (br.ufal.ic.p2.wepayu.controllers.SyndicateController):**
    - Gerencia as operações relacionadas ao sindicato, como lançamento de taxas de serviço.
6. **PayrollController Class (br.ufal.ic.p2.wepayu.controllers.PayrollController):**
    - Controla o processamento da folha de pagamento, incluindo cálculos e geração de relatórios.
7. **Exceptions Package (br.ufal.ic.p2.wepayu.exceptions):**
    - Contém exceções personalizadas lançadas durante a execução do sistema para lidar com situações específicas, como validações e operações relacionadas aos empregados.
8. **Utils Package (br.ufal.ic.p2.wepayu.utils):**
    - Inclui classes utilitárias que oferecem funcionalidades auxiliares para diferentes partes do sistema, como manipulação de datas e gerenciamento de arquivos XML.

Padrões de Projeto Adotados

Padrão de Projeto: Singleton

- **Descrição Geral:**
    - O Singleton é um padrão de projeto que garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a ela.
- **Problema Resolvido:**
    - Garante que uma única instância do controlador (como EmployeeController, SaleController, SyndicateController, PayrollController) seja utilizada em todo o sistema.
- **Identificação da Oportunidade:**
    - Necessidade de manter uma única instância dos controladores para garantir consistência nos dados e evitar duplicação de recursos.
- **Aplicação no Projeto:**
    - Os controladores (EmployeeController, SaleController, SyndicateController, PayrollController) são implementados como Singletons, sendo acessados globalmente.

Padrão de Projeto: Factory Method

- **Descrição Geral:**
    - O Factory Method é um padrão de projeto que define uma interface para criar um objeto, mas deixa as subclasses alterarem o tipo de objetos que serão criados.
- **Problema Resolvido:**
    - Proporciona uma abstração para a criação de objetos, permitindo que subclasses decidam qual classe instanciar.
- **Identificação da Oportunidade:**

    - Necessidade de criar diferentes tipos de empregados (horista, assalariado, comissionado) de forma flexível e extensível.
- **Aplicação no Projeto:**
    - A classe EmployeeController utiliza o Factory Method para criar instâncias específicas de empregados com base no tipo.

Padrão de Projeto: Facade

- **Descrição Geral:**
    - O padrão Facade é utilizado para fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Ele define uma interface de nível mais alto que facilita o uso do subsistema.
- **Problema Resolvido:**
    - Simplifica a interação e utilização de um sistema, fornecendo uma interface mais amigável e coesa.
- **Identificação da Oportunidade:**
    - A complexidade do sistema pode ser reduzida através da criação de uma fachada que coordena as interações entre diferentes partes do sistema.
- **Aplicação no Projeto:**
    - A classe `Facade` (por exemplo, `br.ufal.ic.p2.wepayu.Facade`) atua como uma fachada para o sistema de gerenciamento de empregados. Ela encapsula a complexidade interna e fornece métodos simples e unificados para interagir com as principais funcionalidades do sistema.

Exemplos de Uso dos Padrões de Projeto

Singleton

```
public class EmployeeController {
    private static final EmployeeController instance = new EmployeeController();

    private EmployeeController() {}

    public static EmployeeController getInstance() {
        return instance;
    }
    // ...
}
```

Facade

```
public class Facade {
    private EmployeeController employeeController;
    private SaleController saleController;
    private SyndicateController syndicateController;
    private PayrollController payrollController;

    public Facade() {
        this.employeeController = EmployeeController.getInstance();
        this.saleController = new SaleController();
        this.syndicateController = new SyndicateController();
        this.payrollController = new PayrollController();
    }

    // Métodos unificados para interação com o sistema

    public void addEmployee(String name, String address, String type, String remuneration, String commission) {
        employeeController.addEmployee(name, address, type, remuneration, commission);
    }

    public void recordSale(String employeeId, double saleAmount) {
        saleController.recordSale(employeeId, saleAmount);
    }

    public void addServiceCharge(String employeeId, double serviceCharge) {
        syndicateController.addServiceCharge(employeeId, serviceCharge);
    }

    public void runPayroll() {
        payrollController.calculatePayroll();
        payrollController.printPayrollReport();
    }

    // Outros métodos conforme necessário

    // ...
}
```

Factory Method

```
public class EmployeeController {
    // ...
    public Employee createEmployee(String name, String address, String type, String remuneration, String commission) {
        // ...
        if (type.equals("horista")) {
            return new HourlyEmployee(name, address, remuneration);
        } else if (type.equals("assalariado")) {
            return new SalariedEmployee(name, address, remuneration);
        } else if (type.equals("comissionado")) {
            return new CommissionedEmployee(name, address, remuneration, commission);
        }
        // ...
    }
    // ...
}
```

Como Executar o Projeto

1. Clone o repositório para a sua máquina.
2. Abra o projeto em um ambiente de desenvolvimento Java.
3. Execute a classe `Main.java` para iniciar o sistema.

Licença

Este projeto está licenciado sob a Licença MIT.