

David H.
Daniella
Marcelo
Tharlys
Vitor A.



Gestão de
Pessoas | HCM

FECHAMENTO DA FOLHA



ProlWay

Timeline

Brainstorming
Definição de Estrutura de Dados

23

Implementação do MVC
Interfaces

26

DAO
Testes unitários
interfaces

27

Merge fix
Sprint Review
Apresentação

28





Implementação da Interface

```
package br.com.proway.senior.model;

import java.util.ArrayList;

public interface InterfaceFolhaDAO {
    public ArrayList<Folha> getAll();
    public Folha getFolhasPorId(int id);
    public Folha getFolhaPorDataEId(String data, int id);
    public void saveFolha(Folha folha);
    public boolean updateFolha(Folha folha, int id);
    public boolean removeFolha(int id);
    public Folha getFolhaIdColaborador(Integer idColaborador);
}
```



```
package br.com.proway.senior.model;

public interface InterfacePontoFolha {
    public double getHorasTrabalhadas();
    public double getHorasExtra();
    public double getHorasFaltas();
}
```



DAO

```
public final class FolhaDAO implements InterfaceFolhaDAO {  
  
    private static FolhaDAO instance;  
    private ArrayList<Folha> listaFolhas = new ArrayList<Folha>();  
  
    private FolhaDAO() {  
    }  
  
    public static FolhaDAO getInstance() {... // Singleton  
  
    public ArrayList<Folha> getAll() {... // Busca todas as folhas cadastradas  
  
    public Folha getFolhasPorId(int id) {...  
  
    public ArrayList<Folha> getFolhasPorColaborador(int idColaborador) {... // TODO  
  
    // Busca todas a folha de um determinado usuário e determinada data  
    public Folha getFolhaPorDataEId(String data, int idColaborador) {  
        for (Folha folha : listaFolhas) {  
            String dataEmissao = folha.getDataEmissao();  
            if (dataEmissao != null && dataEmissao.equals(data) && folha.getIdColaborador() == idColaborador) {  
                return folha;  
            }  
        }  
        return null;  
    }  
  
    public void saveFolha(Folha folha) {...  
  
    public boolean updateFolha(Folha folha, int id) {  
        for (Folha up : listaFolhas) {  
            if (up.getId() == id) {  
                int i = listaFolhas.indexOf(up);  
                listaFolhas.remove(i);  
                listaFolhas.add(i, folha);  
                return true;  
            }  
        }  
        return false;  
    }  
  
    public boolean removeFolha(int id) {...  
  
    public Folha getFolhaIdColaborador(Integer idColaborador) {...  
}
```

Folha

```
public Folha(InterfaceColaboradorFolha colaborador, InterfacePontoFolha ponto,  
            InterfaceFeriasFolha ferias, InterfaceCargoFolha cargo) {  
  
    this.horasTrabalhadas = ponto.getHorasTrabalhadas();  
    this.horasExtra = ponto.getHorasExtra();  
    this.horasFalta = ponto.getHorasFaltas();  
    this.valorBonificacao = cargo.getValorBonificacao();  
    this.percentualInsalubridade = cargo.getPercentualInsalubridade();  
    this.mensalidadePlanoSaude = colaborador.getPlanoSaudeMensalidade();  
    this.valorCooparticipacaoPlanoSaude = colaborador.getPlanoSaudeCooparticipacao();  
    this.salarioBase = cargo.getSalario();  
    this.valeTransporte = colaborador.getValeTransporte();  
    this.numeroDependentes = colaborador.getDependentes().size();  
    this.dias = ferias.getDias();  
    this.abono = ferias.getAbono();  
}
```



Classe CalculoFolha

```
calculoData.setDataEmissao(folha);
salarioLiquido = calculoHoras.calcularValorDasHorasTrabalhadas(folha);
folha.setSalarioBruto(folha.getSalarioBruto() + salarioLiquido);
salarioLiquido = calculoHoras.calcularValorHorasFaltas(folha);
folha.setSalarioBruto(folha.getSalarioBruto() - salarioLiquido);
salarioLiquido = calculoHoras.calcularValorHorasExtras(folha);
folha.setSalarioBruto(folha.getSalarioBruto() + salarioLiquido);
salarioLiquido = calculosDeExtras.calcularDSR(folha);
folha.setSalarioBruto(folha.getSalarioBruto() + salarioLiquido);
salarioLiquido = calculosDeExtras.calcularBonificacao(folha);
folha.setSalarioBruto(folha.getSalarioBruto() + salarioLiquido);
salarioLiquido = calculosDesconto.calcularDescontoInss(folha);
folha.setSalarioBruto(folha.getSalarioBruto() - salarioLiquido);
salarioLiquido = calculosDesconto.calcularDescontoImpostoRenda(folha);
folha.setSalarioBruto(folha.getSalarioBruto() - salarioLiquido);
salarioLiquido = calculosDesconto.calcularDescontoPlanoSaude(folha);
folha.setSalarioBruto(folha.getSalarioBruto() - salarioLiquido);
salarioLiquido = calculosDesconto.calcularDescontoValeTransporte(folha);
folha.setSalarioBruto(folha.getSalarioBruto() - salarioLiquido);
```



Teste Integrado

```
ArrayList<String> dependentes = new ArrayList<String>();
FeriasFolha feriasVazias = new FeriasFolha();
CalculoFolha calculo = new CalculoFolha();

@Test
public void testeCalculoFolha() {
    ColaboradorFolha colab = new ColaboradorFolha(0, true, 100, 25, dependentes);
    PontoFolha ponto = new PontoFolha(220, 0, 0);
    CargoFolha cargo = new CargoFolha(1500, 250, 0);
    Folha folha = new Folha(colab, ponto, feriasVazias, cargo);
    calculo.calculoFolha(folha);

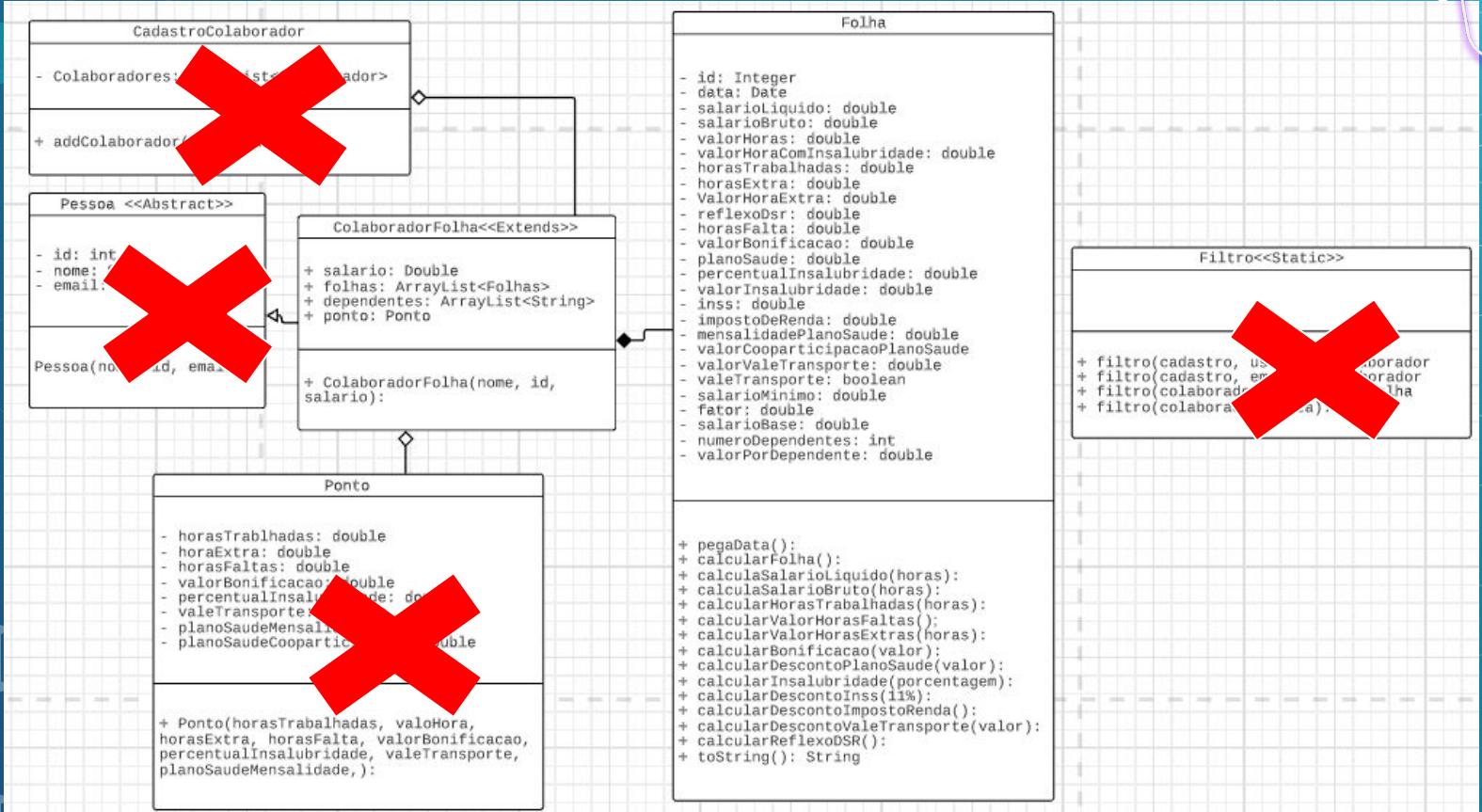
    assertEquals(1342.50, folha.getSalarioLiquido(), 0.01);
}
```



Inversão de dependência

```
public double calcularFeriasBruto(int dias, int abono, double valorHoras) {  
    double valorDia = valorHoras * VALORHORASDIAS;  
    double valorFerias = dias * valorDia;  
    double valorFeriasUmTerco = valorFerias / 3;  
    double valorTotalFerias = 0;  
    if (abono <= 0) {  
        valorTotalFerias = valorFerias + valorFeriasUmTerco;  
    } else {  
        double valorAbono = abono * valorDia;  
        double valorAbonoUmTerco = valorAbono / 3;  
        valorTotalFerias = (valorFerias + valorFeriasUmTerco) + (valorAbono + valorAbonoUmTerco);  
    }  
    return valorTotalFerias;  
}
```

Diagrama de classes



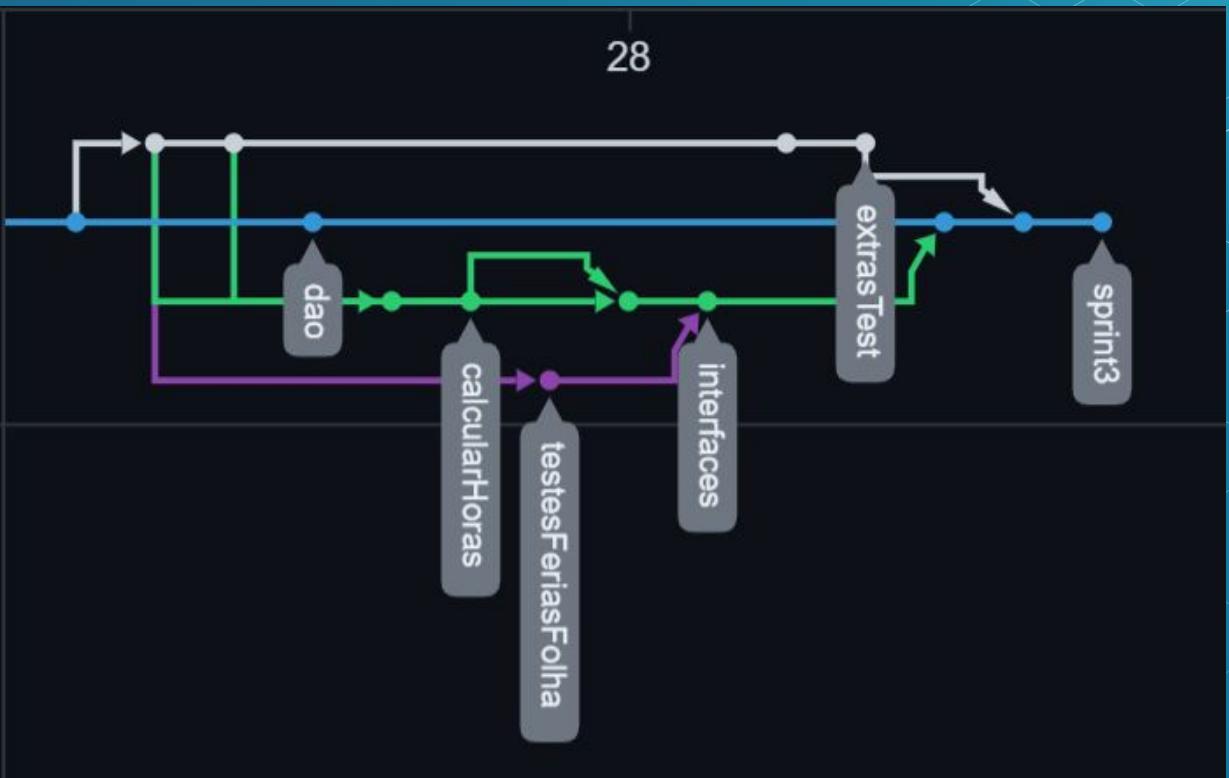
M(V?)C

```
br.com.proway.senior.model  
  > CargoFolha.java  
  > ColaboradorFolha.java  
  > FeriasFolha.java  
  > Folha.java  
  > FolhaDAO.java  
  > InterfaceCargoFolha.java  
  > InterfaceColaboradorFolha.java  
  > InterfaceFeriasFolha.java  
  > InterfaceFolhaDAO.java  
  > InterfacePontoFolha.java  
  > PontoFolha.java
```

```
br.com.proway.senior.controller  
  > CalcularFerias.java  
  > CalcularHoras.java  
  > CalculoData.java  
  > CalculoFolha.java  
  > CalculosDeExtras.java  
  > CalculosDesconto.java  
  > InterfaceBonificacaoExtra.java  
  > InterfaceCalculoData.java  
  > InterfaceCalculoFerias.java  
  > InterfaceHorasExtras.java  
  > InterfaceHorasFaltando.java  
  > InterfaceHorasTrabalhadas.java  
  > InterfaceImpostoDeRendaDesconto.java  
  > InterfaceInsalubridadeExtra.java  
  > InterfaceINSSDesconto.java  
  > InterfacePlanoDeSaudeDesconto.java  
  > InterfaceReflexoDsrExtra.java  
  > InterfaceValeTransporteDesconto.java
```



Commits





Sprint Review

O que foi feito?

- Organizamos o diagrama
- Interfaces / novas Classes
- Desacoplamento da classe Folha
- Refatoração das classes inutilizadas
- Classe DAO
- Testes das classes refatoradas





Sprint Review

O que não foi feito?

- Classe individual para Folha Ferias
- Classe Controller
- Revisão da documentação





Sprint Review

O que foi acrescentado?

- Aplicação Singleton
- Backlog no trello
- Pacotes - MVC
- Encapsulamento





Sprint Review

O que foi retirado/bloqueado?

- Método de calcular valorFérias (imposto de renda)
- Classes dependentes do objeto
- Não identificamos aplicabilidade para outro tipo de Design Patterns





Sprint Retrospective

O que deu certo?

- Rodízio Pair programming
- Group programming
- Não se preocupar com BD
- Testes unitários
- Planejamento
- Organização do Trello
- Troca de conhecimento





Sprint Retrospective

O que deu errado?

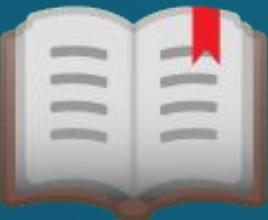
- Teste integrado
- Organização do diagrama



Sprint Retrospective

O que nós aprendemos ?

- Interfaces
- SOLID
- DAO
- MVC
- Design Patterns
- GIT
- Colaboração visando o aprendizado





Sprint Retrospective

O que devemos fazer diferente?

- Consistência na daily
- Reservar um tempo para Commits
- Testar antes de programar
- Testes diferenciados para um mesmo método
- Revisar a documentação
- Gestão do tempo



Sprint Retrospective

★ Fazer mais

- Planejamento
- Uma daily no início e no fim do dia
- Commits

★ Fazer menos

- Trabalhar fora do horário

★ Parar de fazer

- Mudar de foco durante a tarefa
- Testar apenas no final

★ Continuar fazendo

- Pair programming
- Testes unitários
- Diagrama de classes
- Ensinar de forma altruísta

★ Começar a fazer

- Nos preparamos para as condições que estão por vir
- Daily do grupo diária
- Padronização

Obrigadx!

Perguntas?



Senior ProlWay

