**Relatório de Projeto: Simulador de Agência Bancária**

**Curso:** Analise desenvolvimento de sistemas  
**Disciplina:** Programação Orientada a Objetos (POO)  
**Aluno(s):** Kelvin jose arruda, Eduardo andrade dos santos, Vitor silva de Santana, Eweerton Gomes de Paula, Diogenes jose da silva  
**Professor:** Paulo Oliveira Epfanio  
**Data:** 13/11/2024

**Sumário**

1. [Introdução](#1-introdu%C3%A7%C3%A3o)
2. [Objetivos](#2-objetivos)
3. [Requisitos do Sistema](#3-requisitos-do-sistema)
4. [Análise e Projeto Orientado a Objetos](#4-an%C3%A1lise-e-projeto-orientado-a-ob)
5. [Implementação](#5-implementa%C3%A7%C3%A3o)
6. [Testes](#6-testes)
7. [Conclusão](#7-conclus%C3%A3o)

**1. Introdução**

O projeto de um simulador de agência bancária em Java foi desenvolvido para aplicar os conceitos de Programação Orientada a Objetos (POO), com foco na criação de classes, herança, encapsulamento, polimorfismo e outros princípios de design de software. Este sistema simula operações bancárias básicas como abertura de conta, depósito, saque, transferência e consulta de saldo.

**2. Objetivos**

O objetivo deste projeto é desenvolver um sistema simples que simule uma agência bancária. O sistema permite que usuários criem contas bancárias e realizem operações financeiras. Além de fornecer experiência prática com POO, o projeto visa a desenvolver habilidades na construção de soluções modulares e reutilizáveis em Java.

**3. Requisitos do Sistema**

**3.1 Funcionalidades Principais**

1. **Criar Conta**: Permitir a criação de contas bancárias (conta corrente e conta poupança).
2. **Depositar**: Permitir ao usuário realizar depósitos em suas contas.
3. **Sacar**: Permitir ao usuário realizar saques, com verificação de saldo.
4. **Transferir**: Permitir transferências entre contas.
5. **Consultar Saldo**: Exibir o saldo atual da conta.

**3.2 Regras de Negócio**

* **Limite de Saque**: Saques não podem exceder o saldo disponível.
* **Tipos de Conta**: Implementação de contas com características distintas (ex.: conta corrente com limite de cheque especial).
* **Taxa de Juros**: Contas poupança podem ter uma atualização mensal de juros.

**4. Análise e Projeto Orientado a Objetos**

**4.1 Diagrama de Classes**

**Principais Classes**

1. **Conta**: Classe abstrata que representa uma conta bancária. Possui métodos abstratos para depósito e saque.
2. **ContaCorrente**: Subclasse de Conta que representa uma conta corrente, com limite de cheque especial.
3. **ContaPoupanca**: Subclasse de Conta que representa uma conta poupança, com rendimento de juros.
4. **Cliente**: Representa o cliente do banco, com atributos como nome e CPF.
5. **Banco**: Classe que gerencia as contas e clientes.

**4.2 Diagrama (Exemplo)**

*(Aqui, inclua um diagrama UML das classes principais e suas relações)*

**4.3 Descrição das Classes**

* **Conta**: Classe abstrata com atributos comuns a todas as contas, como número da conta e saldo.
* **ContaCorrente** e **ContaPoupanca**: Classes que herdam de Conta e implementam características específicas.
* **Cliente**: Classe que armazena informações do cliente, incluindo os métodos para associar contas.
* **Banco**: Classe principal que gerencia as operações e os registros de contas e clientes.

**5. Implementação**

**5.1 Estrutura do Código**

O código foi dividido em pacotes de acordo com as classes principais, incluindo um pacote para exceções personalizadas, como SaldoInsuficienteException.

**5.2 Detalhes de Implementação**

* **ContaCorrente** e **ContaPoupanca** implementam o método sacar() de maneiras distintas.
* **Banco** contém métodos para adicionar novos clientes, abrir contas e realizar operações de depósito, saque e transferência.

**5.3 Exemplos de Código**

**AgenciaBancaria.Java**

package main;

import java.util.ArrayList;

import java.util.Scanner;

import model.Pessoa;

import model.Conta;

public class AgenciaBancaria {

static Scanner input = new Scanner(System.in);

static ArrayList<Conta> contasBancarias;

public static void main(String[] args) {

contasBancarias = new ArrayList<>();

operacoes();

}

@SuppressWarnings("empty-statement")

public static void operacoes() {

System.out.println("------------------------------------------------------");

System.out.println("-------------Bem vindos a nossa Agência---------------");

System.out.println("------------------------------------------------------");

System.out.println("\*\*\*\*\* Selecione uma operação que deseja realizar \*\*\*\*\*");

System.out.println("------------------------------------------------------");

System.out.println("| Opção 1 - Criar conta |");

System.out.println("| Opção 2 - Depositar |");

System.out.println("| Opção 3 - Sacar |");

System.out.println("| Opção 4 - Transferir |");

System.out.println("| Opção 5 - Listar |");

System.out.println("| Opção 6 - Sair |");

int operacao = input.nextInt();;

switch (operacao) {

case 1:

criarConta();

break;

case 2:

depositar();

break;

case 3:

sacar();

break;

case 4:

transferir();

break;

case 5:

listarContas();

break;

case 6:

System.out.println("Flw é nóis!");

System.exit(0); // para o sistema

default:

System.out.println("Opção inválida!");

operacoes();

break;

}

}

public static void criarConta() {

//System.out.println("Você está criando uma conta\n");

System.out.println("\nNome: ");

String nome = input.next();

System.out.println("\nCPF: ");

String cpf = input.next();

System.out.println("Email: ");

String email = input.next();

Pessoa cliente = new Pessoa(nome, cpf, email);

Conta conta = new Conta(cliente);

contasBancarias.add(conta);

System.out.println("--- Sua conta foi criada com sucesso! ---");

operacoes();

}

private static Conta encontrarConta(int numeroConta) {

Conta conta = null;

if(!contasBancarias.isEmpty()) {

for(Conta contaa : contasBancarias) {

if(contaa.getNumeroConta() == numeroConta) {

conta = contaa;

}

}

}

return conta;

}

public static void depositar() {

System.out.println("Número da conta: ");

int numeroConta = input.nextInt();

Conta conta = encontrarConta(numeroConta);

if(conta != null) {

System.out.println("Qual valor deseja depositar? ");

Double valorDeposito = input.nextDouble();

conta.depositar(valorDeposito);

}else {

System.out.println("--- Conta não encontrada ---");

}

operacoes();

}

public static void sacar() {

System.out.println("Número da conta: ");

int numeroConta = input.nextInt();

Conta conta = encontrarConta(numeroConta);

if(conta != null) {

System.out.println("Qual valor deseja sacar? ");

Double valorSaque = input.nextDouble();

conta.sacar(valorSaque);

System.out.println("--- Saque realizado com sucesso! ---");

}else {

System.out.println("--- Conta não encontrada ---");

}

operacoes();

}

public static void transferir() {

System.out.println("Número da conta que vai enviar a transferência: ");

int numeroContaRemetente = input.nextInt();

Conta contaRemetente = encontrarConta(numeroContaRemetente);

if(contaRemetente != null) {

System.out.println("Número da conta do destinatário: ");

int numeroContaDestinatario = input.nextInt();

Conta contaDestinatario = encontrarConta(numeroContaDestinatario);

if(contaDestinatario != null) {

System.out.println("Valor da transferência: ");

Double valor = input.nextDouble();

contaRemetente.transferencia(contaDestinatario, valor);

}else {

System.out.println("--- A conta para depósito não foi encontrada ---");

}

}else {

System.out.println("--- Conta para transferência não encontrada ---");

}

operacoes();

}

public static void listarContas() {

if(!contasBancarias.isEmpty()) {

for(Conta conta: contasBancarias) {

System.out.println(conta);

}

}else {

System.out.println("--- Não há contas cadastradas ---");

}

operacoes();

}

}

**Conta.Java**

package model;

import utils.Utils;

public class Conta {

private static int accountCounter = 1;

private final int numeroConta;

private Pessoa pessoa;

private Double saldo = 0.0;

public Conta(Pessoa pessoa) {

this.numeroConta = Conta.accountCounter;

this.pessoa = pessoa;

this.updateSaldo();

Conta.accountCounter += 1;

}

public int getNumeroConta() {

return numeroConta;

}

public Pessoa getClient() {

return pessoa;

}

public void setClient(Pessoa pessoa) {

this.pessoa = pessoa;

}

public Double getSaldo() {

return saldo;

}

public void setSaldo(Double saldo) {

this.saldo = saldo;

}

private void updateSaldo() {

this.saldo = this.getSaldo();

}

/\*\*

\*

\* @return

\*/

@Override

public String toString() {

return "\nBank account: " + this.getNumeroConta() +

"\nCliente: " + this.pessoa.getName() +

"\nCPF: " + this.pessoa.getCpf() +

"\nEmail: " + this.pessoa.getEmail() +

"\nSaldo: " + Utils.doubleToString(this.getSaldo()) +

"\n" ;

}

public void depositar(Double valor) {

if(valor > 0) {

setSaldo(getSaldo() + valor);

//this.saldo = this.getSaldo() + valor;

System.out.println("Seu depósito foi realizado com sucesso!");

}else {

System.out.println("Não foi possível realizar o depósito!");

}

}

public void sacar(Double valor) {

if(valor > 0 && this.getSaldo() >= valor) {

setSaldo(getSaldo() - valor);

System.out.println("Saque realizado com sucesso!");

}else {

System.out.println("Não foi possível realizar o saque!");

}

}

public void transferencia(Conta contaParaDeposito, Double valor) {

if(valor > 0 && this.getSaldo() >= valor) {

setSaldo(getSaldo() - valor);

//this.saldo = this.getSaldo() - valor;

contaParaDeposito.saldo = contaParaDeposito.getSaldo() + valor;

System.out.println("Transferência realizada com sucesso!");

}else {

System.out.println("Não foi possível realizar a tranferência");

}

}

}

**Pessoa.Java**

package model;

import java.util.Date;

import utils.Utils;

public class Pessoa {

private static int counter = 1;

private int numeroPessoa ;

private String name;

private String cpf;

private String email;

private Date accountCreationDate;

public Pessoa() { }

public Pessoa(String name, String cpf, String email) {

this.numeroPessoa = Pessoa.counter;

this.name = name;

this.cpf = cpf;

this.email = email;

this.accountCreationDate = new Date();

Pessoa.counter += 1;

}

public int getNumeroPessoa() {

return this.numeroPessoa;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCpf() {

return cpf;

}

public void setCpf(String cpf) {

this.cpf = cpf;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Date getAccountCreationDate() {

return this.accountCreationDate;

}

/\*\*

\*

\* @return

\*/

@Override

public String toString() {

return "\nName: " + this.getName() +

"\nCPF: " + this.getCpf() +

"\nEmail: " + this.getEmail() +

"\nAccount Creation Date: " + Utils.dateToString(this.getAccountCreationDate());

}

}

**Utils.Java**

package utils;

import java.text.DecimalFormat;

import java.text.NumberFormat;

import java.text.SimpleDateFormat;

import java.util.Date;

public class Utils {

static NumberFormat formatandoNumeros = new DecimalFormat("R$ #,##0.00");

static SimpleDateFormat formatandoData = new SimpleDateFormat("dd/MM/yyyy");

public static String dateToString(Date data) {

return Utils.formatandoData.format(data);

}

public static String doubleToString(Double valor) {

return Utils.formatandoNumeros.format(valor);

}

}

**6. Testes**

**6.1 Testes Unitários**

Foram realizados testes unitários com as principais funcionalidades do sistema usando JUnit. Exemplos de testes:

* **Teste de Depósito**: Verificar se o valor depositado é adicionado ao saldo.
* **Teste de Saque**: Testar se o saque reduz o saldo e lança exceção para saldo insuficiente.
* **Teste de Transferência**: Testar a transferência entre contas e a atualização de saldos.

**6.2 Resultados dos Testes**

* Todos os testes unitários foram aprovados.
* Testes de exceção para saldo insuficiente e limites de cheque especial funcionaram conforme esperado.

**7. Conclusão**

O projeto de simulador de agência bancária foi uma experiência prática valiosa para aplicar conceitos de POO em Java. Ele proporcionou uma compreensão mais profunda sobre herança, encapsulamento, polimorfismo e exceções. Todas as funcionalidades planejadas foram implementadas e testadas com sucesso. Como melhorias futuras, poderiam ser adicionados recursos como interface gráfica, persistência de dados em banco de dados, e integração de funcionalidades de segurança para simular autenticação.