BEATRIZ CERQUEIRA DA COSTA
EDUARDO SQUETINI DE CAMPOS
GABRIEL EVANGELISTA DOS SANTOS
GABRIEL GUILHERME VIEIRA DOS SANTOS
NICOLY DHAYANE. ALVES

A3- Sistema de cadastro e listagem de clientes

SÃO PAULO- SP
2022

BEATRIZ CERQUEIRA DA COSTA
EDUARDO SQUETINI DE CAMPOS
GABRIEL EVANGELISTA DOS SANTOS
GABRIEL GUILHERME VIEIRA DOS SANTOS
NICOLY DHAYANE ALVES

A3- Sistema de cadastro e listagem de clientes

Projeto apresentado aos cursos de análise e desenvolvimento de sistemas e gestão da tecnologia da informação da universidade Anhembi Morumbi, como um dos requisitos para avaliação e conclusão da UC de Programação de soluções computacionais.

Professores:
Arthur Diego Dias Rocha
Bruno Henrique Oliveira Mulina

SÃO PAULO- SP
2022

1. ESTUDO DO PROBLEMA

Sistema de cadastramento de clientes pessoas físicas e pessoas jurídicas.

Toda empresa necessita de uma forma de armazenar os dados relevantes de seus clientes. Dependendo da quantidade de clientes do estabelecimento, o ideal seria um programa para manter organizado os dados e informações de cada cliente, separando pessoas físicas de pessoas jurídicas, além de permitir a listagem de clientes cadastrados daquele tipo.

Dependendo do tamanho do local, sem o sistema fica tudo na responsabilidade e memória dos funcionários, o que pode fazer com que uma informação deixe de ser incluída no cadastro ou que essa informação seja registrada de forma equivocada, gerando uma demanda de retrabalho, com isso, todo o setor perde em produtividade. Assim, um sistema que consiga incluir os dados do cliente, tipo de pessoa (PJ ou PF), seus dados relevantes e que permita listar tais pessoas, faz com que o processo de cadastramento e armazenamento seja mais organizado e otimize tempo.

2. SOLUÇÃO

Como solução para os problemas que foram descritos, nós propomos um sistema de cadastro de clientes. Onde a empresa pode manter de forma simples e detalhada os dados relevantes do cliente e a listagem total de clientes filtradas como PF ou PJ.

Nossa solução resolve o problema da lentidão e complicação para se cadastrar clientes em um estabelecimento, ainda mais tendo que especificar eles dentro do sistema.

Esse programa vai incluir os dados principais do cliente, o tipo de pessoa, nome, CPF ou CNPJ, etc.

As informações ficarão armazenadas em um banco de dados de Arrays, sendo uma Array para clientes no geral, uma Array de clientes PJ e uma Array de clientes PF.

3. DESCRIÇÃO DOS CONCEITOS DA APLICAÇÃO

3.1 CLASSE E OBJETO

O conceito de Classe e Objeto se aplica nas classes criadas: Cliente.java, BancoDeDados.java, Cadastrar.java, ClientePessoaJuridica.java, ClientePessoaFisica.java, Excecoes. Java e Listar.java. O objeto instanciado principal é o cadastro do cliente, que é adicionado ao banco de dados de Arrays de clientes.

3.2 HERANÇA

A herança aplica-se da classe Cliente para as classes ClientePessoaJuridica e ClientePessoaFisica, visto que para definir PF e PJ, elas herdam da superclasse (Cliente) características já existentes. Por exemplo, sabemos que Pessoa Física e Pessoa Jurídica possuem o atributo nome como uma informação em comum. Dentre as diversas informações, o que as diferencia é o uso de CPF para pessoa física e CNPJ para pessoa jurídica. Ao invés de definir o atributo nome para as duas classes, cria-se uma classe abstrata (a classe Cliente) e insere um atributo nome dentro dela, onde acontecerá a herança das propriedades para as subclasses (PF e PJ), o atributo nome vem automaticamente pela superclasse e ficam definidas dentro de ClientePessoaFisica o atributo CPF e para ClientePessoaJuridica o CNPJ. Além disso, a Classe Cadastrar é uma subclasse de Listar, pois herda seus métodos e atributos, a classe Cadastrar também implementa a interface BancoDeDados. A subclasse Excecoes também é herança da superclasse do java Exception.

3.3 CLASSES ABSTRATAS E INTERFACES

Classificamos a classe Cliente como abstrata pois ela é a classe mãe e não precisa ser instanciada, sendo apenas um modelo para a geração de outras classes. E utilizamos a interface na classe BancoDeDados, pois não é necessário métodos e outras implementações nela já que ela é a base de armazenamento do sistema e “obriga” que as demais classes que a implementar tenham seus métodos, no caso o vetor, onde através do ArrayList vai ser possível que os dados coletados sejam listados e separados em pessoas físicas e pessoas jurídicas.

3.4 TRATAMENTO DE EXCEÇÕES

As exceções vêm para tratamento de possíveis erros que ocorram durante a execução do programa. Onde utilizamos os comandos throw e throws para invocá-las e gerá-las, try e catch para tratá-las e informar ao usuário de forma clara qual é o erro. Por exemplo, no caso de fornecimento de dados inválidos, de acordo com os requisitos do atributo, o programa emite uma mensagem e pede para que o usuário revise as informações fornecidas para que ele continue rodando.

4. DESCRIÇÃO DO PROGRAMA

Na classe Interface Banco de dados armazenamos as Arrays de clientes no geral, clientes PJ e clientes PF. Por ser a base de armazenamento do programa a criamos como Interface, pois tais Arrays serão essenciais e implementadas por todo o programa.

Para a implementação do banco de dados desenvolvemos uma classe abstrata para popular clientes em geral, a classe Clientes. Lá se encontram os atributos nome, e-mail e data de nascimento que todos os clientes terão em modo private para encapsulamento e proteção de dados:

Em seguida criamos os setters e getters com a devida validação de cada um utilizando o método matches para verificar a quantidade de dígitos suficientes e se contém letras ou números necessários ou desnecessários, se inválido (utilizamos if e else para guia) ele retorna a exceção da classe Excecoes criada por nós com a mensagem de erro escolhida.

Extensão da superclasse Cliente, a classe ClientePessoaFisica adiciona o atributo privado CPF com encapsulamento set e get para especificação de Clientes PFs.

O método setCpf tem como parâmetro o cpf do cliente em tipo String, recebendo o dado apenas se o cpf estiver dentro das eespecificações impostas pelo matches. Caso não esteja, ele invoca uma exceção criada na classe Exceções que retorna a mensagem de CPF inválido.

O método matches verifica se o que foi inserido é compatível com a regex utilizada nele, a regex aprova uma composição de 11 dígitos numéricos apenas.

O método getCpf retorna o cpf validado.

Por meio de polimorfismo, o Override que criamos do método toString nessa classe retorna as informações dadas do cliente PF.

Extensão da superclasse Cliente, classe herança ClientePessoaJuridica. Adiciona os atributos privados CNPJ e quantidade de filiais com encapsulamento set e get para especificação de Clientes PJs.

O método setCnpj tem como parâmetro o CNPJ do cliente em tipo String, recebendo o dado apenas se o CNPJ estiver dentro das eespecificações impostas pelo matches. Caso não esteja, ele invoca uma exceção criada na classe Exceções que retorna a mensagem de CNPJ inválido.

O método matches verifica se o que foi inserido é compatível com a regex utilizada nele, a regex aprova uma composição de 14 dígitos numéricos apenas.
O método getCnpj retorna o CNPJ validado.
O método setQtdFiliais tem como parâmetro a quantidade de filiais do cliente em tipo int, recebendo o dado apenas se o número for maior ou igual a 1. Caso não seja, ele invoca uma exceção criada na classe Exceções que vai retornar a mensagem de número inválido para quantidade de filiais.
O método getQtdFiliais retorna a quantidade de filiais já validada.
Por meio de polimorfismo, o Override que criamos do método toString nessa classe retorna as informações dadas do cliente PJ.
A subclasse Excecoes extende a super classe Exceptions do Java. O atributo que essa subclasse possui é uma mensagem de erro imposta pelo programador que usamos como parâmetro na função construtura Excecoes() onde irá "setar" a mensagem em tipo String e retornar na função getMessage(), mas essa mensagem só será mostrada no console com o método printMessage() que utiliza o System.out.println
A classe Listar implementa a classe BancoDeDados, permitindo listar os cadastros de clientes por meio do método listarCadastros(). Este inclui a checagem de existencia de cadastros nas Arrays e a organização de cadastros em tipo PJ e PF, mostrando-os no console. Uma crucial função utilizada foi a size() para verificar o tamanho das arrays de clientes e o loop for para a listagem. Além disso, as funções de desvio if e else também foram utilizadas para guiar os casos.
Em nossa classe Main Teste invocamos as Excecoes utilizando throws e possui a instância de um novo objeto que sera manipulado através do método Cadastrar().
A classe Cadastrar herda os métodos da classe Listar e implementa o BancoDeDados. De início importamos uma Exceção que pode ser detectada e o Scanner para leitura de entrada de dados. Ao instanciar o Scanner criamos também um atributo booleano chamado execute. Na criação da função construtora da classe invocamos a classe Excecoes através do throws.
Enquanto o atributo execute for true, o método irá rodar o loop while onde criamos uma variável String chamada opcao que receberá o valor de retorno do método menu(), se o retorno for n será iniciado um novo cadastro de cliente, se

for será listado os clientes existentes e se for x a variável execute receberá false e encerrará o loop e o programa. Essas opções poderão ser lidas seja em forma minúscula ou maiúscula devido o uso do método equalsIgnoreCase(). Caso não seja digitado nenhuma dessas opções, o programa retornará como opção inválida.

No método menu() são apresentadas as opções que o usuário pode escolher ao iniciar o programa e é retornado a String de entrada devido o Scanner.

No método menuPfOuPj() o programa pergunta se o usuário deseja cadastrar uma pessoa física (PF) ou pessoa jurídica (PJ), retornando a String digitada utilizando o Scanner in.

No método cadastrar() invocamos a classe de Excecoes através do throws, criamos uma variável booleana chamada cadastrando que recebe o valor true e iniciamos um loop enquanto tal variável for verdadeira. Dentro deste criamos a variável String pfOuPj que recebe o retorno do método menuPfouPj()

Com o uso de desvios if e else checamos se o cliente que estamos cadastrando será um objeto adicionado ao vetor de clientes PF ou um objeto adicionado ao vetor de clientes PJ. Instanciamos esse objeto e fazemos um tratamento de exceções onde criamos uma variável booleana que representa o valor inválido e inicia como true, dentro do try colocamos o setNome para o objeto cliente e o método input para ler essa String, se validado pelo setNome a variável que representa invalidez de dados recebe o valor false e encerra o loop, se não o catch do tipo de Excecoes e irá mostrar a mensagem de erro.

O mesmo conceito de leitura e validação ocorre com os outros dados de entrada do cliente.

Uma variável String chamada cadastrar é criada para ler se o usuário deseja confirmar o cadastro, recebendo apenas as letras S para sim ou N para não, sendo maiúsculas ou minúsculas, devido o uso do método equalsIgnoreCase(). Dependendo da escolha do cliente, que também será guiada pelo if e else, o objeto cadastro de cliente será adicionado à array de clientes e à array de tipo de pessoa correspondente (PF ou PJ) por meio do método add. Ou o objeto cadastro de cliente será cancelado. Se não for entrado nem S nem N a opção será dada como inválida e pedirá novamente pela escolha.

Uma variável String é criada recebendo o retorno do método input com a pergunta de adicionar mais cadastros ou não, por meio dos ifs e elses e do

método equalsIgnoreCase() checamos se foi digitado S para voltar ao início do loop de cadastrar ou N para tornar a variável cadastrando false e encerrar o loop de cadastro. Além disso, se a opção digitada for inválida o programa também encerrará o loop de cadastro tornando a variável cadastrando como false.

Para o primeiro método input utilizamos o parâmetro de String rótulo e retornamos o que foi digitado por meio do Scanner criado por nós.

Para o segundo método input criamos devido o polimorfismo de sobrecarga, nele recebemos o parâmetro int de algum número e retornamos esse número.