Universidade Federal do Cariri

Curso: Bacharelado em Ciência da Computação.

Disciplina: Programação Orientada à Objetos.

Professora: Paola Accioly.

Equipe: João Isaac Alves Farias e Vinícius Bezerra Leite.

SISTEMA DE GERENCIAMENTO DE FUNCIONÁRIOS

Link do projeto no GitHub

1. DESCRIÇÃO DO SISTEMA

Este sistema é um projeto CRUD que tem o intuito de gerir os funcionários de uma empresa, de forma que seja possível manipular as informações dos mesmos, tais como código de cadastro, nome, cargo, remuneração, telefone e endereço. As manipulações possíveis serão cadastro, edição, exclusão e listagem de informações dos funcionários. Tal sistema tem o intuito de facilitar o gerenciamento dos recursos humanos (R.H.) da empresa, centralizando todas as informações necessárias de forma que tanto o responsável pelo R.H. possa acessar, quanto o gerente da empresa.

Desta forma, o sistema terá um módulo, que é um app web acessível pelo R.H. da empresa em questão e vai permitir que o funcionário encarregado possa criar um funcionário novo na empresa, editar informações pré-existentes ou acessar as informações necessárias, e acessível também pelo gerente da empresa, de forma que possa acessar as informações dos funcionários, alterar alguns dados e removê-lo, caso necessário. Nossa escolha foi baseada na necessidade de um sistema unificado que possa gerir estas informações nas empresas atualmente, para assim manter um controle mais rígido sobre os dados dos funcionários, além de remuneração por cargo e um gasto total da empresa.

2. BACKLOG

Funcionalidade	Responsável	Status
Adicionar Funcionário	João	Feito
Adicionar Cargo	Vinicius	Feito
Adicionar Departamento	João	Feito
Listar Funcionários	João	Feito
Listar Cargos	Vinicius	Feito
Listar Departamentos	João	Feito
Buscar Funcionário	João	Feito
Buscar Cargo	Vinicius	Feito
Buscar Departamento	João	Feito
Remover Funcionário	João	Feito
Remover Cargo	João	Feito

Remover Departamento	João	Feito
Editar Funcionário	João	Feito
Editar Cargo	Vinicius	Feito
Editar Departamento	Vinicius	Feito

3. ARQUITETURA DO SISTEMA

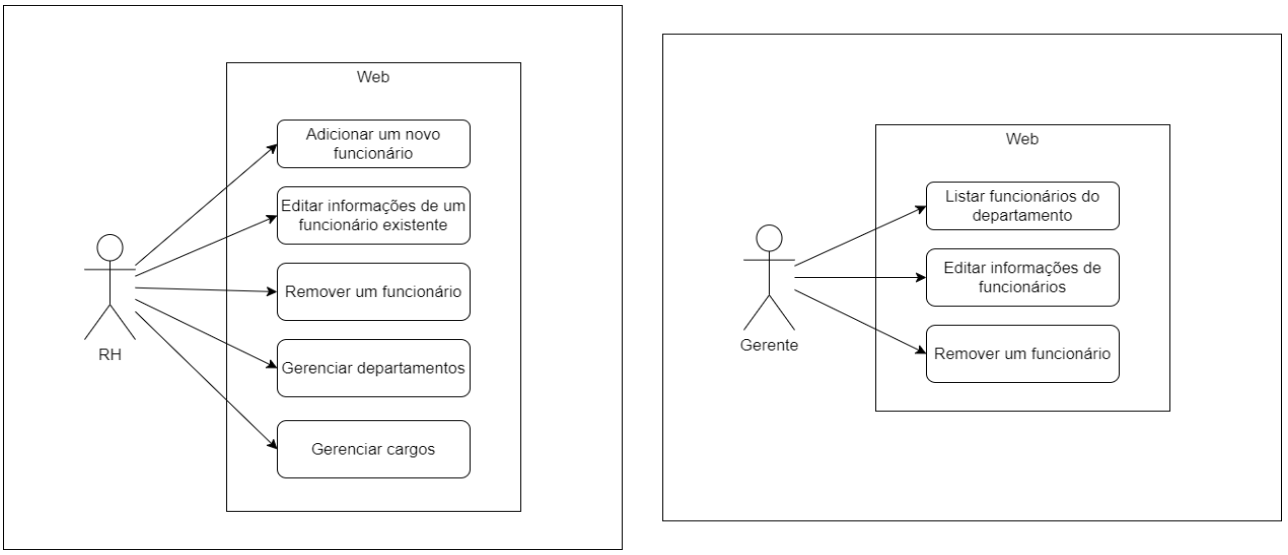

Figura 1: Diagrama de casos de uso.

Link para diagramas

O primeiro caso de uso descreve a interação do RH da empresa com o sistema de gerenciamento de funcionários. O RH pode cadastrar novos funcionários preenchendo seus dados, bem como gerenciar os cargos e departamentos da empresa. O RH também pode visualizar e editar informações de cargos e departamentos existentes no sistema.

Já o segundo caso de uso descreve a interação dos gerentes da empresa com o sistema de gerenciamento de funcionários. Os gerentes acessam o sistema usando suas credenciais de acesso e podem visualizar os dados dos funcionários sob sua responsabilidade, bem como gerenciar as informações dos funcionários, como cargos, departamentos e salários. Os gerentes também podem realizar buscas e filtrar os funcionários por diferentes critérios.

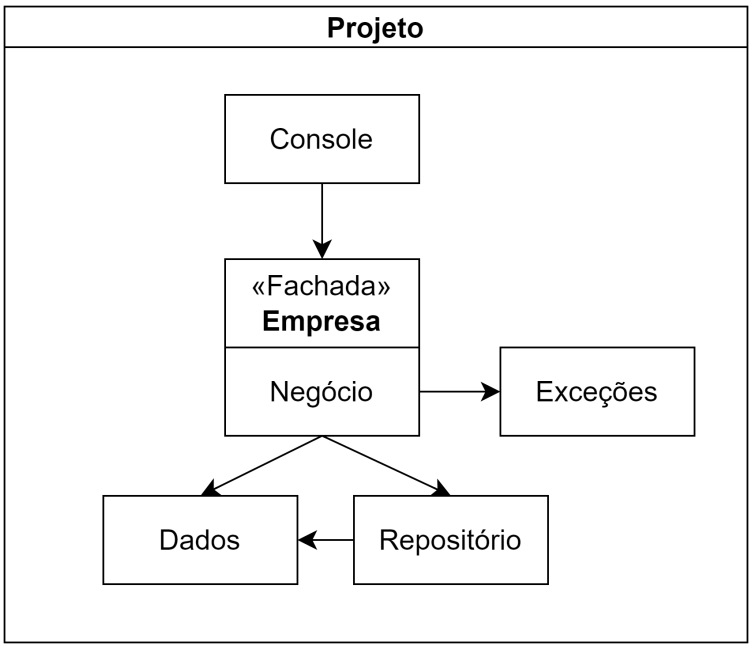

Figura 2: Diagrama de pacotes.

No nosso projeto, utilizamos a organização de código em camadas para estruturar o software de forma modular e escalável. Dividimos a funcionalidade do sistema em diferentes camadas, cada uma responsável por um conjunto específico de tarefas.

Começando pela camada de Dados, nela incluímos as classes de domínio do nosso sistema. Essas classes representam as entidades do mundo real relacionadas a gestão de recursos humanos de uma empresa. Elas encapsulam a estrutura dessas entidades, definindo propriedades, relacionamentos e métodos apropriados. Essas classes estão focadas exclusivamente na representação dos dados e, embora métodos setters e construtores tenham regras de validação, não contêm lógica de negócio.

A próxima camada é a de Repositório, que se encarrega da persistência dos dados do sistema. Nessa camada, fornecemos interfaces e implementações para armazenar e recuperar objetos da camada de Dados em algum meio de armazenamento, como um banco de dados, arquivos ou memória. Essa camada abstrai os detalhes de como os dados são armazenados e oferece métodos simples para a criação, leitura, atualização e exclusão (CRUD) dos objetos do domínio.

Na camada de Negócio, concentramos a lógica de negócio do nosso sistema de R.H. É nessa camada que definimos os serviços e regras de negócio que operam nos objetos da camada de Dados. Ela aplica validações e toma decisões sobre os objetos de domínio da aplicação. Embora dependa da camada de Repositório para acessar os dados e realizar operações persistentes, a camada de Negócio não conhece nem precisa conhecer os detalhes de implementação do Repositório

Por fim, temos a camada de Console, que oferece uma interface de usuário no terminal. Nessa camada, incluímos classes que interagem com o usuário, exibindo informações, coletando entrada e apresentando resultados.

A organização em camadas do nosso projeto nos permite ter uma clara separação de responsabilidades e um baixo acoplamento entre as partes do sistema. Cada camada tem uma função bem definida e pode ser substituída ou estendida facilmente sem afetar as outras camadas. Isso nos dá flexibilidade para realizar alterações e melhorias em diferentes partes do sistema sem causar impactos indesejados.

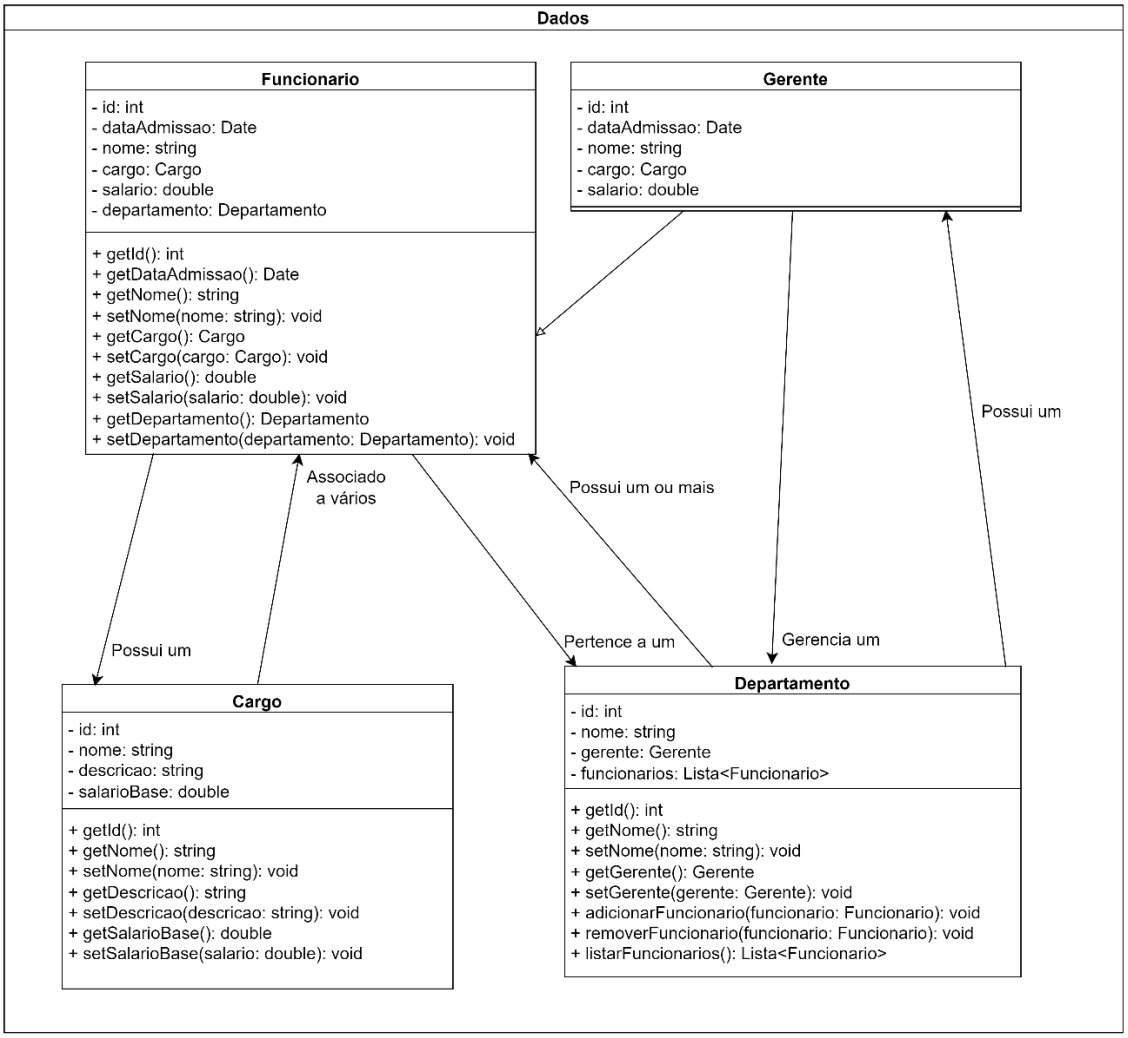

Figura 3: Diagrama de classes do pacote dados.

As classes de dados são responsáveis por modelar as entidades do mundo real e seus relacionamentos. Por exemplo, a classe Funcionário contém informações pessoais do funcionário, como nome e data de admissão, bem como relacionamentos com outras entidades do sistema, como seu cargo e gerente.

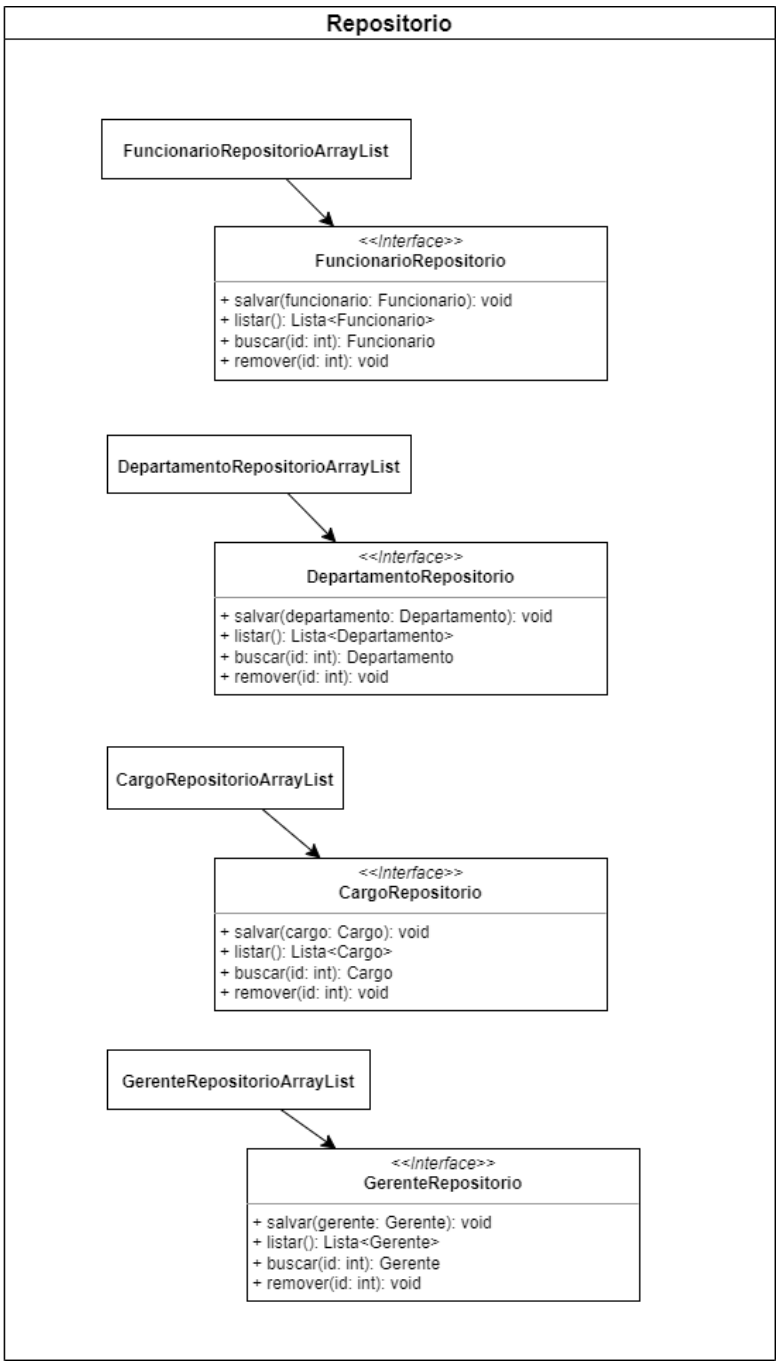

Figura 4: Diagrama de repositório.

As interfaces de repositório foram criadas para desacoplar a implementação do repositório em si do resto do sistema. Isso permite que diferentes implementações de repositório possam ser usadas, como armazenamento em memória ou em banco de dados, sem afetar o restante do sistema.

Em resumo, o uso de interfaces permite desacoplar a implementação do repositório do resto do sistema, permitindo diferentes implementações de

armazenamento de dados. A implementação em ArrayList fornece uma maneira simples de armazenar dados em memória para prototipação e testes, embora não seja útil em produção já que os dados não são persistidos.

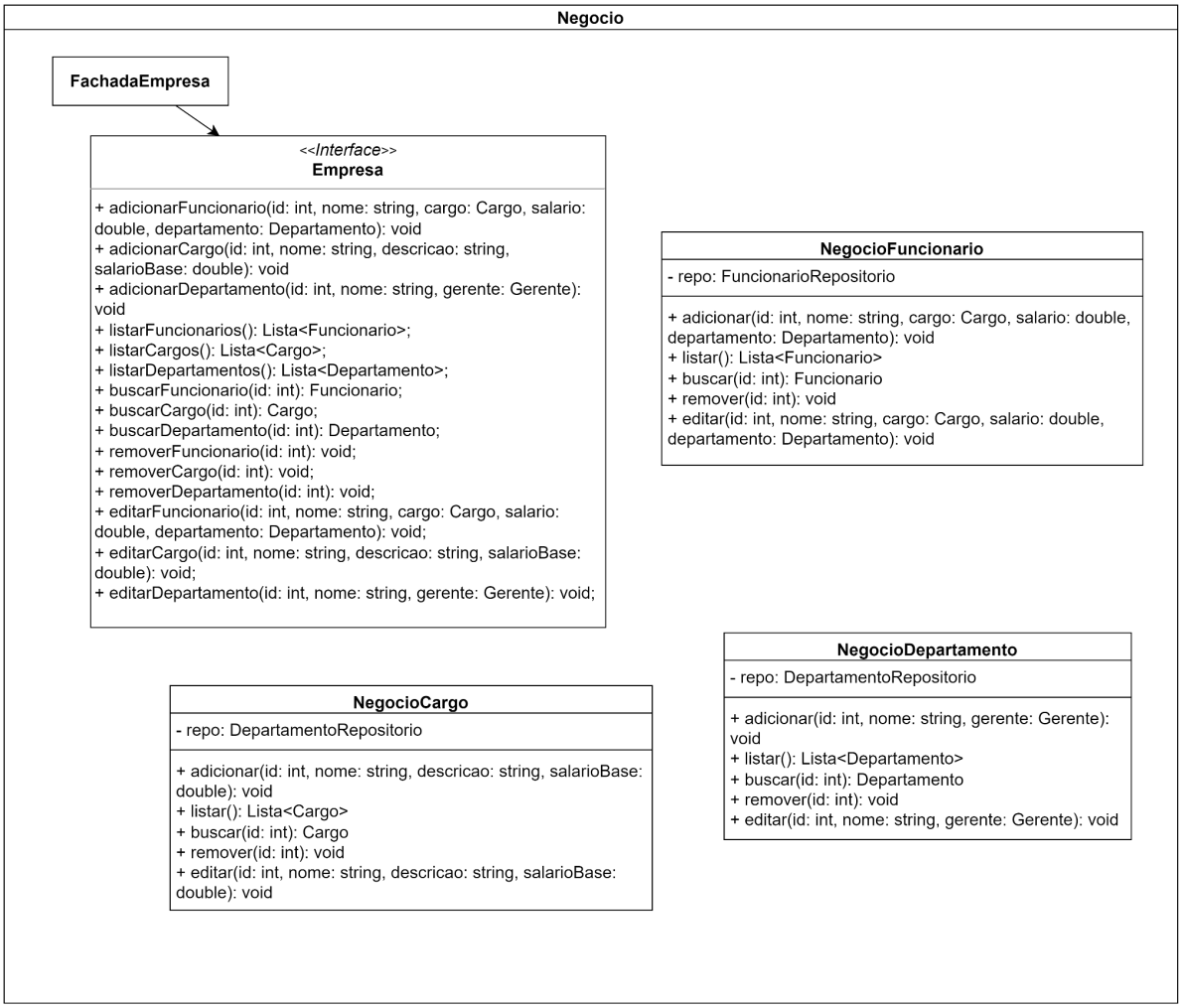

Figura 5: Diagrama de negócio.

As classes de negócio, como NegocioFuncionario, NegocioDepartamento e NegocioCargo, são responsáveis por implementar a lógica de negócio específica para funcionários, departamentos e cargos em nosso sistema de gestão de funcionários. Elas lidam com operações como adicionar, atualizar e excluir registros, aplicar regras de negócio e realizar consultas. Essas classes atuam como intermediárias entre a camada de dados e a interface do usuário, garantindo a consistência dos dados e a aplicação das regras de negócio no sistema.

A classe de fachada, Empresa, desempenha um papel importante ao fornecer uma interface simplificada para os usuários do sistema de gestão de funcionários. Ela

encapsula as funcionalidades das classes de negócio, facilitando o uso e a interação com o sistema. Através da fachada, os usuários podem acessar as operações comuns que envolvem funcionários, departamentos e cargos de forma mais conveniente. Além disso, a fachada oculta a complexidade do sistema subjacente, permitindo que as mudanças internas sejam feitas sem afetar os usuários externos. Isso proporciona uma maior modularidade, facilita a manutenção do código e melhora a organização geral do projeto.

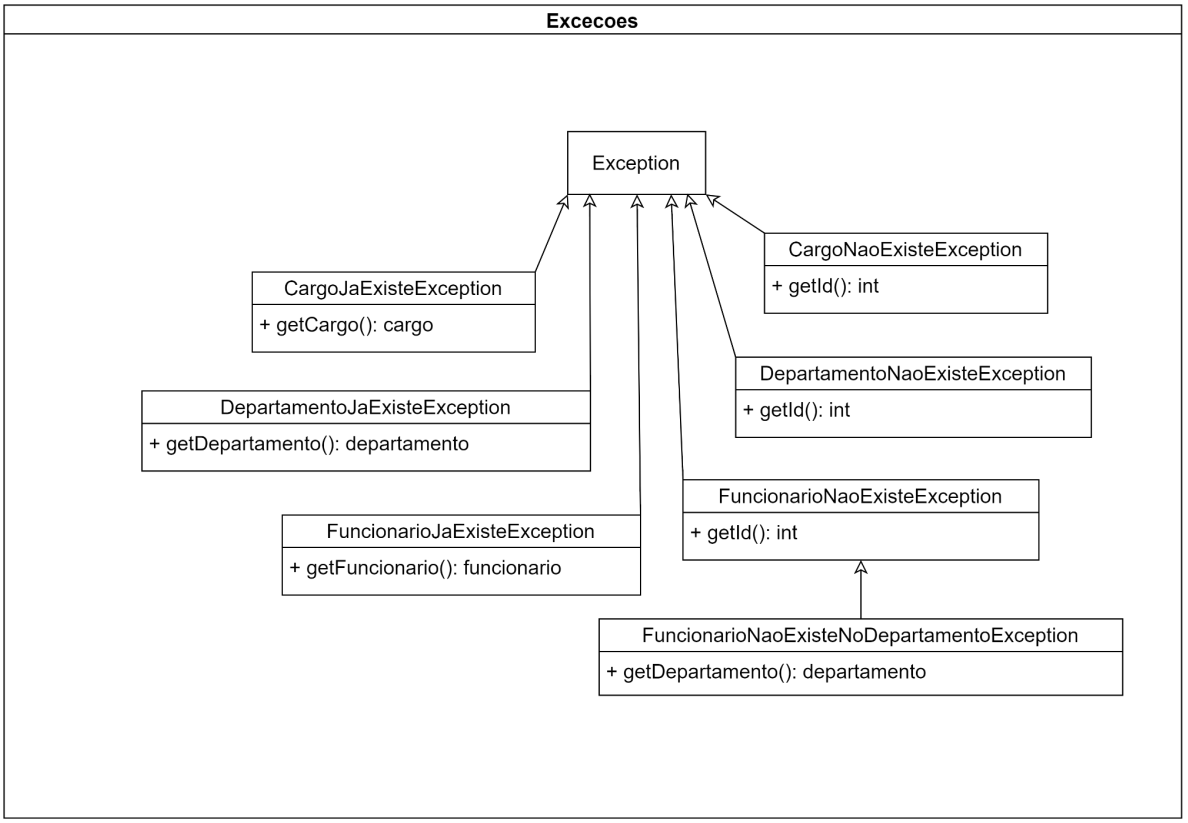

Figura 6: Diagrama de exceções.

O pacote de exceções contém exceções personalizadas para tratar situações específicas relacionadas aos funcionários e outras classes do domínio. Essas exceções são projetadas para lidar com erros comuns que podem ocorrer durante a execução do sistema, como o "funcionário já existe" (para cadastro de novos funcionários) ou o "funcionário não existe" (no caso de busca, edição e exclusão). Ao lançar essas exceções, podemos capturá-las em pontos relevantes do código e

tomar medidas apropriadas, como exibir mensagens de erro adequadas ao usuário ou realizar ações específicas para lidar com essas situações. O uso desse pacote de exceções nos permite tratar exceções de forma mais precisa e fornecer um feedback mais claro e preciso aos usuários do sistema.

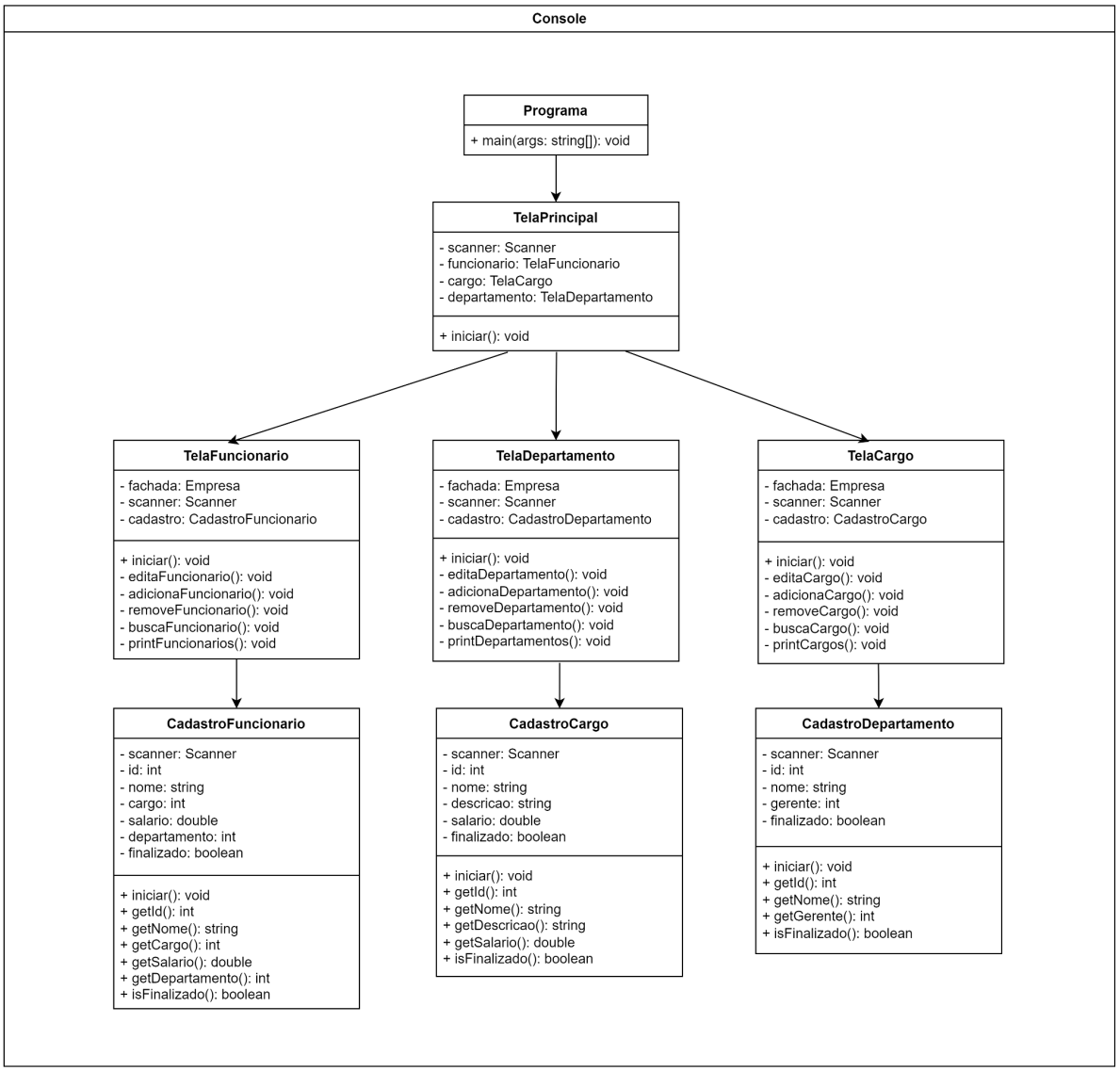

Figura 7: Diagrama de console.

No pacote "console" do nosso projeto, encontramos a classe Programa, que contém o método main e serve como ponto de entrada do sistema. Esse método main é responsável por iniciar a execução do programa e chama a classe TelaPrincipal. A TelaPrincipal é a primeira tela exibida para o usuário no console e apresenta opções relacionadas às funcionalidades disponíveis, como CRUD de funcionários, cargos e departamentos. Essa tela pode redirecionar o usuário para outras três telas:

TelaFuncionario, TelaCargo e TelaDepartamento. Essas telas exibem opções específicas de CRUD para cada entidade e são capazes de ler a escolha do usuário a partir da entrada do console.

Além das telas, o pacote "console" também contém as classes CadastroFuncionario, CadastroCargo e CadastroDepartamento. Essas classes atuam como DTOs (Data Transfer Objects) e são responsáveis por ler as informações inseridas pelo usuário, seja para criar novos objetos dos tipos Funcionario, Cargo e Departamento, ou para editar objetos existentes. Elas funcionam como intermediárias entre as telas e as classes de negócio, recebendo e armazenando as informações fornecidas pelo usuário, que serão posteriormente utilizadas para construir ou atualizar os objetos pertinentes. Essas classes de cadastro são responsáveis por garantir a integridade e validade dos dados inseridos pelo usuário antes de serem utilizados nas operações de criação ou edição dos objetos correspondentes.