Relatório Explicativo — Gerenciamento de Patch Panel

Feito por: Giordan Ebertz

Descrição do Cenário Escolhido

O cenário escolhido foi o **Sistema de mapeamento de cabos de rede** em um ambiente corporativo. Neste tipo de ambiente usa-se muitas vezes os chamados **Patch Panels** que são equipamentos usados em redes de comunicação para facilitar a **organização e o gerenciamento de cabos**.

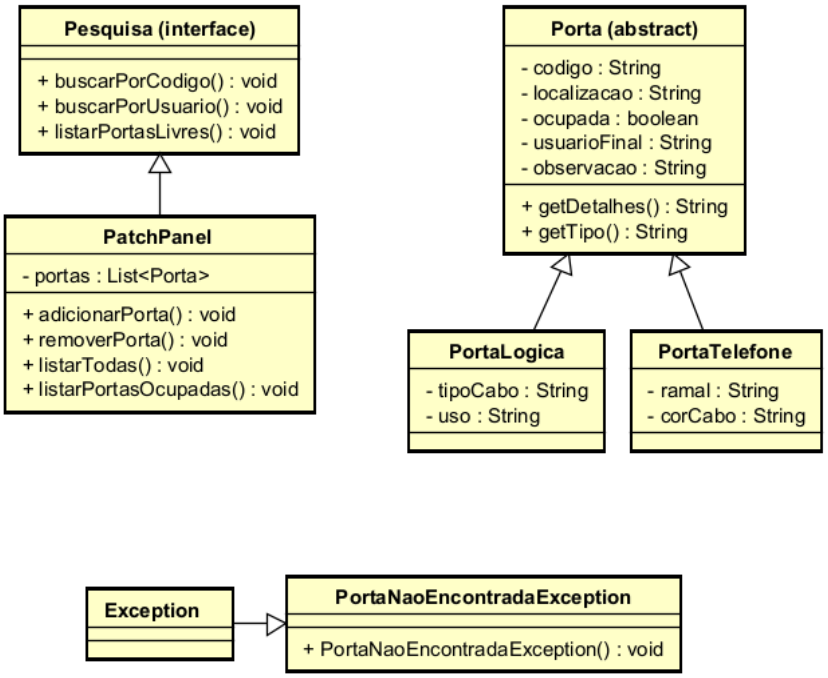
Porém aí começa um **problema**: Com o crescimento das redes, fica cada vez mais difícil controlar quais portas estão ocupadas, livres, onde estão localizadas e quem está utilizando.

Diante deste problema pensou-se em uma **solução**: Um sistema de linha de comando que permita:

- Cadastrar novas portas (rede lógica ou ramal telefônico)
- Listar as portas (todas, livres ou ocupadas)
- Buscar portas específicas por código ou usuário
- Remover portas específicas cadastradas
- Gerenciar observações e informações técnicas (tipo de cabo, cor, uso, etc.)

O sistema foi projetado com base em princípios de orientação a objeto, visando escalabilidade, modularidade e reuso de código.

Diagrama(s) de classes



Justificativa para as Escolhas de Modelagem

Herança (Classe Abstrata `Porta`):

- `Porta` foi criada como classe abstrata porque representa o conceito geral de uma porta (atributos comuns como código, localização, ocupação, usuário e observação).

- `PortaLogica` e `PortaTelefone` **herdam** `Porta` e implementam comportamentos específicos para portas de rede lógica e ramais telefônicos.

Interface (`Pesquisa`):

- `Pesquisa` foi criada para definir **padrões de busca** no sistema. Dessa forma, se amanhã for necessário implementar outro tipo de pesquisa (por exemplo, por localização), será possível sem alterar o PatchPanel diretamente.

Polimorfismo (Sobrescrita de Métodos):

- `getDetalhes()` e `getTipo()` são métodos sobrescritos em `PortaLogica` e `PortaTelefone`, permitindo que, mesmo trabalhando com uma lista genérica `List<Porta>`, o comportamento se ajuste dinamicamente dependendo do tipo da porta.

Uso de Collections (`ArrayList`):

- A coleção escolhida foi um `ArrayList` por ser eficiente para listas que sofrem adições/removeres frequentes e que precisam ser percorridas sequencialmente (listagens, buscas).

Tratamento de Exceções:

- Criada a exceção personalizada `PortaNaoEncontradaException` para tratar erros de busca de forma controlada.

- Uso de `try-catch` no `Main` para evitar que entradas inválidas quebrem o sistema.