Relatório de Projeto: Simulação POO I (Química)

Aluno: Robson Orlando Rodrigues Rosa

Turma: ADS 3° Período

Data: 28/07/2025

1. Introdução

Este projeto consiste no desenvolvimento de uma aplicação em Java para simular as operações básicas de um laboratório de química. A ideia central é gerenciar um estoque de substâncias, como ácidos e bases, permitindo a execução de experimentos virtuais como testes de solubilidade e mistura de reagentes.

O sistema foi projetado para ser robusto, utilizando tratamento de exceções para lidar com situações previsíveis, como a tentativa de dissolver uma quantidade excessiva de soluto ou misturar reagentes incompatíveis.

O tema de Química foi escolhido por sua excelente afinidade com os paradigmas da Programação Orientada a Objetos. A forma como as substâncias são classificadas, as propriedades que possuem, as interações que realizam e as regras de segurança envolvidas se traduzem de maneira intuitiva e didática nos conceitos de Classes Abstratas, Interfaces e Tratamento de Exceções, que são o foco deste trabalho.

Essa correspondência natural permitiu não apenas aplicar a teoria, mas também compreender profundamente o propósito por trás de cada decisão de design.

2. Decisões de Design

Sobre a Classe Abstrata

Qual classe abstrata foi criada? Por que foi decidido que ela deveria ser abstrata e não uma classe comum?

Foi criada uma classe abstrata `SubstanciaQuimica`. A decisão de torná-la abstrata foi estratégica e fundamental para o design do sistema de simulação. Uma classe comum não seria apropriada porque o conceito de "Substância Química" é genérico demais para existir por si só neste contexto; ninguém manipula uma "substância genérica", mas sim substâncias específicas como um ácido ou uma base.

Ao usar uma classe abstrata, foi estabelecido um "contrato conjunto":

1. **Reutilização de Código:** Foi definido atributos (`nomeComum`, `formulaQuimica`) e comportamentos (`getDetalhes()`) que são idênticos para todas as substâias, evitando repetição de código.
2. **Obrigação de Implementação:** Através do método abstrato `descreverRisco()`, forcei que toda e qualquer subclasse concreta (como `Acido` e `Base`) implemente sua própria lógica para descrever seus perigos específicos. Isso garante consistência e segurança ao modelo, *pois nenhuma substância poderá ser criada sem que seu risco seja definido.*

Sobre a Interface

Qual interface foi criada? Que "habilidade" ou "contrato" ela representa no sistema? Por que uma interface foi a melhor escolha para essa situação?

Foi criada a interface `Soluvel`. No sistema, ela representa o "contrato" ou a "habilidade" que uma substância tem de poder ser dissolvida em água. Uma classe que implementa `Soluvel` está efetivamente dizendo: "Eu sei como realizar a ação de me dissolver".

Uma interface foi a melhor escolha por um motivo fundamental de design: a solubilidade é uma característica à classificação de uma substância. Em outras palavras, ser um `Acido` ou uma `Base` não determina se você é solúvel, isso caracteriza uma relação de herança. Existem ácidos solúveis e insolúveis. Usar uma interface permitiu adicionar essa funcionalidade de forma flexível apenas às substâncias relevantes, sem "poluir" a classe mãe `SubstanciaQuimica` com um comportamento que não é universal. Isso separa claramente o herança da interface, tornando o sistema mais limpo e modular.

Sobre as Exceções

Quais exceções personalizadas foi criada? Explicação de uma situação no código onde uma delas é lançada. Por que foi melhor criar essa exceção do que usar uma já existente no Java?

Foram criadas duas exceções personalizadas para lidar com erros específicos do nosso domínio: `LimiteDeSolubilidadeException` e `ReacaoIncompativelException`.

1. **`LimiteDeSolubilidadeException`**: Está dentro do método `dissolverEmAgua`, presente nas classes `Acido` e `Base`. A situação que dispara a exceção é uma tentativa de dissolver uma massa de substância em um volume de água que excede a taxa de solubilidade predefinida para aquele composto (por exemplo, tentar dissolver mais de 36g de um ácido em 100ml de água).
2. **`ReacaoIncompativelException`**: É lançada no método `realizarMistura` da classe `Laboratorio` quando se tenta combinar duas substâncias que, segundo as regras da simulação, não deveriam ser misturadas (no nosso caso, duas substâncias do mesmo tipo, como dois ácidos).

A decisão de criar exceções personalizadas em vez de usar as genéricas do Java (como `IllegalArgumentException`) foi uma escolha de design para aumentar a clareza e a manutenibilidade do código. Enquanto `IllegalArgumentException` apenas nos diz que um argumento foi inválido, `LimiteDeSolubilidadeException` nos informa *exatamente qual regra de negócio do nosso domínio foi violada*. Isso torna o código auto-documentado, muito mais fácil de debugar e permite que o sistema crie tratamentos específicos para cada tipo exato de erro.

3. Desafios e Aprendizados

Um dos principais desafios durante o projeto foi definir a fronteira entre os papéis da Classe Abstrata e da Interface. Inicialmente, foi considerado incluir o método `dissolverEmAgua()` diretamente na classe `SubstanciaQuimica`. No entanto, rapidamente nota-se que isso criaria um problema de design: geraria problema em dar uma implementação vazia (ou lançar um

erro) para futuras substâncias que não fossem solúveis, como um `Oleo` ou um `MetalInsoluvel`. A reflexão sobre este problema me levou a adotar a interface `Soluvel`, que se revelou uma solução muito mais elegante e flexível, reforçando o aprendizado de que a herança define "o que algo é", enquanto a interface define "o que algo pode fazer".

O maior aprendizado deste trabalho foi a transição do pensamento de "escrever código que funciona" para "desenhar soluções que duram". Compreender na prática como a abstração ajuda a gerenciar a complexidade, como as interfaces promovem a flexibilidade e como o tratamento de exceções constrói a robustez, medeu uma visão muito mais profunda sobre a arquitetura de software e a importância de tomar decisões de design conscientes.