Fundação Escola de Comércio Álvares Penteado

Programação Orientada a Objetos e Estrutura de Dados

Graduação em Ciência da Computação

Projeto Interdisciplinar: Aplicativo Móvel

Guilherme Barioni

Iury Xavier

Lillian Conde

Marcus Duque

Murilo Vieira

São Paulo

2025

1. Introdução

A Programação Orientada a Objetos (POO) foi essencial no nosso projeto, permitindo um código mais limpo, otimizado e reutilizável. Conceitos de estrutura de dados também foram amplamente utilizados para melhor organização dos dados e facilitação do armazenamento e manipulação das informações.

2. Classe

Através de classes, conseguimos modelar entidades do nosso sistema com variáveis e métodos, tornando o código modular e reutilizável. Um exemplo claro disso é a classe Usuario.java, que armazena as informações do usuário, como nome, senha, e-mail e celular. Ela também contém métodos para acessar e modificar esses dados de maneira estruturada e segura.

```java
public class usuario {

    3 usages
    private String nome;
    4 usages
    private String senha;
    4 usages
    private String email;
    3 usages
    private String celular;

    6 usages
    public usuario(String nome, String senha, String email, String celular) {
        this.email = email;
        this.nome = nome;
        this.senha = senha;
        this.celular =celular;
    }

    1 usage
    public usuario(String senha, String email){
        this.email = email;
        this.senha = senha;
    }
```

3. Interface

A interface ApiService foi criada para padronizar a comunicação com a API do projeto. Ela define os endpoints disponíveis e as operações que podem ser realizadas, como obter, adicionar, alterar ou excluir dados de usuários e outros recursos.

```
public interface ApiService {

    1 usage
    @POST("logErro")
    Call<LogErros> addErro(@Body LogErros logErros);

    // Endpoint para pegar todos os usuarios
    no usages
    @GET("tudo")
    Call<List<usuario>> getUsers();

    // Endpoint para adicionar usuarios
    1 usage
    @POST("usuario")
    Call<Void> addUser(@Body usuario user);

    //Endpoint para pegar um usuario
    1 usage
    @POST("login")
    Call<usuario> validarUser(@Body usuario user);

    //Endpoint para deletar um usuario
    1 usage
    @POST("deletar")
    Call<usuario> deletarUser(@Body usuario user);
```

4.1 Estrutura de Dados - Arrays

O uso de arrays no código é um exemplo clássico de como podemos organizar dados de forma eficiente. Na Criptografia.java, temos dois arrays que armazenam caracteres extraídos de uma String para serem usados no processo de criptografia, e um array que armazenara o código ASCII dos caracteres. Um dos maiores benefícios dos arrays é que eles permitem o acesso direto e rápido aos seus elementos, usando um índice. Como os elementos do array são armazenados de forma contígua na memória, é possível acessar qualquer posição do array em tempo constante O(1).

```
public class Criptografia {

    14 usages
    public static String Criptografar(String dado, String chave) {
        char[] dadoSplit = dado.toCharArray();
        char[] chaveSplit = chave.toCharArray();
        int[] keyCodes = new int[chaveSplit.length];
        int keyCodeDado;
        char dadoCriptChar;
        String dadoCriptografado ="";
```

4.2 Estrutura de Dados - Listas

O uso de listas no código é um exemplo claro de como podemos organizar dados de forma eficiente e dinâmica. No código abaixo, três listas são usadas para armazenar itens de localização (representados pela classe OverlayItem), com base na distância calculada. Cada lista é destinada a uma categoria específica de proximidade: muito longe, longe e perto. Ao utilizar listas, o código consegue armazenar uma quantidade variável de itens de localização de maneira eficiente, sem a necessidade de pré-definir o número de elementos. O uso de listas também facilita o acesso dinâmico aos itens, já que você pode adicionar ou remover elementos conforme necessário.

```
private List<OverlayItem> muitoLonge , longe, perto;
//Verifica a distancia
if(item.getTitle() !="Minha localização"){
    if (distancia < 100000 && distancia > 10000) {
        muitoLonge.add(item);
    } else if (distancia < 10000 && distancia > 1000) {
        muitoLonge.add(item);
        longe.add(item);
    } else if (distancia < 1000 && distancia > 250) {
        muitoLonge.add(item);
        perto.add(item);
    } else if (distancia <= 250) {
        muitoLonge.add(item);
        if(distancia>5) {
            alertaPerto(item.getTitle(), distancia, item.getSnippet());
        }
    }
```

5. Conclusão

A implementação de conceitos de Programação Orientada a Objetos (POO) e estruturas de dados foi fundamental para o sucesso do nosso projeto, garantindo um código bem estruturado, modular e eficiente. Em conjunto, esses elementos trouxeram não apenas uma maior organização ao código, mas também contribuíram para a performance e escalabilidade do sistema, tornando-o mais robusto e fácil de manter.