FUNDAÇÃO ESCOLA DE COMÉRCIO ÁLVARES PENTEADO – FECAP
3°SEMESTRE – CIÊNCIAS DA COMPUTAÇÃO

Programação Orientada a Objetos e Estrutura de Dados – Estrutura de dados utilizadas

Caroliny Rossi Bittencourt,
Duda Lucena Miguel,
Rafael Alves dos Santos Guimarães,
Rafael Moraes Marques.

São Paulo

2025

1. INTRODUÇÃO

Este relatório tem como objetivo apresentar e explicar as principais estruturas utilizadas de Programação Orientada a Objetos (POO) e de Estruturas de Dados aplicadas no projeto desenvolvido na disciplina, destacando exemplos reais extraídos do código-fonte e justificando suas utilizações.

2. Programação Orientada a Objetos (POO)

2.1 Classe - Uso Geral

As classes são a base da Programação Orientada a Objetos. Elas permitem definir estruturas que agrupam atributos (variáveis) e métodos (funções) relacionados a um mesmo conceito ou entidade. Essa organização facilita a manutenção, reutilização de código e modularização.

2.2 Classe e Herança - Exemplo: Motorista.java

A classe `Motorista` estende a classe `Usuario`, utilizando o conceito de herança. Isso permite o reaproveitamento de atributos e métodos comuns entre diferentes tipos de usuários, como Motorista e Passageiro, tornando o código mais organizado e com menos redundância.

Exemplo:

```
public class Motorista extends Usuario {
    private String modeloDoCarro;
    private String placaDoCarro;
    private String disponibilidade;
    private String statusProtocolo;
    private String frase1;
    private String frase2;
    private String frase3;
```

2.3 Classe Abstrata - Exemplo: Usuario.java

A classe `Usuario` é declarada como abstrata, pois ela representa um conceito genérico que não deve ser instanciado diretamente. Ela serve como base para

classes mais específicas como `Motorista` e `Passageiro`, forçando as subclasses a implementarem certos comportamentos, o que garante maior controle sobre a estrutura do sistema.

Exemplo:

```
public abstract class Usuario {
    private int id;
    private String nome;
    private String sobrenome;
    private String telefone;
    private String email;
    private String senha;
```

2.4 Interface - Exemplo: OnServidorDetectado.java

Interfaces são usadas para definir contratos que as classes devem seguir. Elas são especialmente úteis quando diferentes classes precisam compartilhar um mesmo conjunto de comportamentos, mas não têm uma relação direta de herança. No projeto, a interface `OnServidorDetectado` permite que diferentes componentes reajam de forma padronizada à detecção de um servidor, promovendo flexibilidade e baixo acoplamento entre as partes do sistema.

Exemplo:

```
package com.umonitoring.api;

public interface OnServidorDetectado {
    void onDetectado();
}
```

3. Estruturas de Dados

3.1 ArrayList - Exemplo: Viagem.java

A estrutura `ArrayList` é usada para armazenar listas dinâmicas de objetos, como listas de viagens. Ela é importante quando não se sabe previamente a quantidade de elementos ou quando a coleção precisa crescer dinamicamente.

É amplamente utilizada em sistemas que trabalham com registros de dados que podem ser adicionados ou removidos em tempo de execução.

Exemplo:

```
public static List<Viagem> listarTodasAsViagens() {
    String resposta = ViagemAPI.listarTodas();
    List<Viagem> lista = new ArrayList<>();

    try {
        JSONObject obj = new JSONObject(resposta);
        if (obj.has("viagens")) {
            JSONArray array = obj.getJSONArray("viagens");
            for (int i = 0; i < array.length(); i++) {
                JSONObject json = array.getJSONObject(i);
                Viagem v = construirViagem(json);
                lista.add(v);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }

    return lista;
}
```

3.2 Stack - Exemplo: CadastroPage.java

A estrutura `Stack` (pilha) funciona com o princípio LIFO (Last In, First Out), sendo útil para controle de fluxos, histórico de páginas ou chamadas reversas. No exemplo da tela de cadastro, o uso de uma pilha pode ajudar a controlar o retorno de ações ou manter um histórico reversível.

Exemplo:

```
private GeoPoint geocodificarEndereco(String endereco) {
    Geocoder geocoder = new Geocoder(this);
    try {
        List<Address> resultados = geocoder.getFromLocationName(endereco, 1);
        if (resultados != null && !resultados.isEmpty()) {
            Address local = resultados.get(0);
            return new GeoPoint(local.getLatitude(), local.getLongitude());
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

4. CONSIDERAÇÕES FINAIS

O uso correto das estruturas de POO e de dados permite a construção de sistemas mais robustos, escaláveis e organizados. A aplicação prática desses conceitos, conforme demonstrado neste relatório, reflete a importância de se compreender e utilizar boas práticas no desenvolvimento de software.