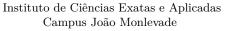


MINISTÉRIO DA EDUCAÇÃO

Universidade Federal de Ouro Preto





 2^{o} semestre de 2024

Data: 05/12/24

Curso: Engenharia de Computação Disciplina: Linguagens de Programação Professor: Luiz Carlos Bambirra Torres

Aluna: Luisa Brito Dias - Matrícula (20.1.8107)

Tarefa 1: Comparação de Paradigmas - Imperativo vs. Orientado a Objetos com Abstração

1. Identificação de Abstrações

- Identifique as abstrações naturais deste problema e explique por que "Música", "Playlist" e "Catálogo" são boas escolhas para abstrações.
 - Essas abstrações representam conceitos claros e intuitivos do domínio
 - Música: Representa a unidade básica, contendo atributos como título, artista, gênero e duração.
 - Playlist: Um agrupamento de músicas com propósitos específicos, como "Favoritas" ou "Treino".
 - Catálogo: Uma coleção geral de todas as músicas organizadas, permitindo busca e navegação.
- Descreva como essas abstrações ajudam a ocultar detalhes específicos (como a reprodução de áudio ou a manipulação de listas) e simplificam o desenvolvimento do sistema.
 - Elas simplificam o desenvolvimento do sistema ao esconder detalhes técnicos, como manipulação de listas e reprodução de áudio, permitindo que os desenvolvedores se concentrem na lógica principal.

2. Implementação Imperativa

- Descreva como resolveria esse problema usando o paradigma imperativo, sem abstrações significativas.
 - Dados são armazenados em estruturas básicas, como arrays ou listas.
 - Funções implementam operações, como adicionar músicas, buscar por título ou criar playlists.
- Explique como a abordagem imperativa lida com abstração e quais limitações você observa se o sistema for ampliado para incluir novos recursos, como busca avançada por artistas ou gêneros.
 - Falta de encapsulamento: Dados e funções são tratados de forma separada, aumentando a complexidade à medida que o sistema cresce.
 - Dificuldade de expansão: Adicionar novos recursos, como busca por artista ou filtros de recomendação, requer a criação de várias funções adicionais, tornando o código mais suscetível a erros.
 - Manutenção: Sem uma estrutura clara, o código se torna difícil de entender e manter.

3. Implementação Orientada a Objetos

- Descreva como resolveria o problema usando o paradigma orientado a objetos, aproveitando a abstração.
 - O sistema é dividido em classes, representando as abstrações principais:
 - Classe Musica: Contém atributos (título, artista, gênero, duração).



MINISTÉRIO DA EDUCAÇÃO

Universidade Federal de Ouro Preto



Instituto de Ciências Exatas e Aplicadas Campus João Monlevade

- Classe Playlist: Gerencia uma lista de músicas, com métodos como adicionar Musica()
 e mostrar Detalhes().
- Classe Catalogo: Organiza músicas e playlists, oferecendo funcionalidades como busca avançada e filtros.
- Explique como a abstração no paradigma orientado a objetos ajuda a modularizar o código e a tornar o sistema mais flexível para adições de novas funcionalidades, como recomendações ou filtros.
 - Modularidade: Cada classe encapsula sua lógica, tornando o código mais organizado e fácil de entender.
 - Facilidade de expansão: Novos recursos, como filtros ou recomendações, podem ser adicionados sem impactar outras partes do sistema.
 - Reutilização de código: Objetos podem ser reutilizados em diferentes contextos, aumentando a eficiência.

4. Comparação e Discussão

• Discuta como o conceito de abstração é tratado em cada paradigma, focando nas vantagens de organizar e ocultar detalhes em um sistema que pode crescer em complexidade.

- Tratamento da abstração em cada paradigma:

- * No paradigma imperativo, a abstração é limitada, com foco em manipular dados diretamente. O sistema se torna difícil de organizar e de esconder a complexidade.
- * No paradigma orientado a objetos, a abstração é central. Dados e comportamentos associados são encapsulados em classes, refletindo conceitos do mundo real e promovendo a modularização.

Vantagens do paradigma OO:

- * Melhor organização do código, com estruturas claras e intuitivas.
- * Facilidade para adicionar novas funcionalidades, como filtros ou recomendações, sem alterar a base do sistema.
- * Maior manutenção e legibilidade, graças à modularização e ao encapsulamento.