SISTEMA DE GERENCIAMENTO DE COLEÇÃO DE DISCOS UTILIZANDO PARADIGMA ORIENTADO A OBJETOS COM JAVA

Giovanni Braga Soares Vasconcelos[1], Antônio Heitor Gomes Azevedo[1], Cauã Maia de Souza Nara[1]

[1] Escola de Negócios, Tecnologia e Inovação - ARGO, Centro Universitário do Estado do Pará, Brasil, 5º Período de Ciência da Computação.

{giovanni23070008, antonio23070017, caua23070005}@aluno.cesupa.br

Paradigmas de Linguagens de Programação.

Profª. Suzana Lustosa de Souza

Belém, Pará. Maio de 2025.

RESUMO

Este artigo apresenta o desenvolvimento de um sistema de gerenciamento para coleção de discos utilizando o paradigma orientado a objetos, especificamente com a linguagem Java. O objetivo é demonstrar como o paradigma orientado a objetos pode ser aplicado para solucionar problemas reais de organização e gestão de informações, onde o foco está na modelagem de entidades do mundo real através de classes, objetos, herança e polimorfismo. O sistema desenvolvido permite gerenciar diferentes tipos de mídia musical (vinil, CD, digital), artistas e coleções, possibilitando operações especializadas para cada formato através da implementação de polimorfismo, encapsulamento e herança.

Palavras-chave: Paradigma Orientado a Objetos; Java; Sistema de Informação; Gerenciamento de Coleção de Discos.

ABSTRACT

This paper presents the development of a record collection management system using the object-oriented programming paradigm, specifically with the Java language. The goal is to demonstrate how the object-oriented paradigm can be applied to solve real-world information organization and management problems, where the focus is on modeling real-world entities through classes, objects, inheritance, and polymorphism. The developed system allows managing different types of musical media (vinyl, CD, digital), artists, and collections, enabling specialized operations for each format through the implementation of polymorphism, encapsulation, and inheritance.

Keywords: Object-Oriented Paradigm; Java; Information System; Record Collection Management.

1. INTRODUÇÃO

Na era digital atual, a organização e gestão de coleções pessoais tornaram-se uma necessidade crescente para muitos entusiastas da música. Colecionadores modernos frequentemente possuem bibliotecas musicais híbridas, incluindo discos de vinil, CDs e arquivos digitais, cada formato com suas características, vantagens e necessidades específicas de manuseio. Neste contexto, o presente artigo tem como objetivo desenvolver um sistema de gerenciamento para coleção de discos utilizando o paradigma orientado a objetos, especificamente com a linguagem Java, demonstrando como conceitos avançados como polimorfismo e herança podem ser aplicados para criar soluções elegantes e extensíveis.

Como destacado por Deitel e Deitel (2017), o paradigma orientado a objetos revolucionou a forma como pensamos sobre programação, permitindo que modelemos sistemas computacionais de maneira mais próxima ao mundo real. Este paradigma se baseia na criação de objetos que encapsulam dados e comportamentos, facilitando a manutenção, reutilização e extensibilidade do código através de conceitos fundamentais como herança e polimorfismo.

Um sistema de gerenciamento para coleção de discos com múltiplos formatos representa um caso ideal para aplicação completa do paradigma orientado a objetos, pois envolve entidades relacionadas mas distintas (discos de vinil, CDs, arquivos digitais) que compartilham características comuns mas possuem comportamentos específicos. Ao utilizar Java e implementar hierarquias de classes com polimorfismo, podemos criar um sistema que trata diferentes tipos de mídia de forma uniforme enquanto preserva suas particularidades, como observado por Horstmann (2019).

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Paradigmas de Programação

Os paradigmas de programação representam diferentes filosofias e abordagens para estruturar e resolver problemas computacionais. Cada paradigma oferece uma perspectiva única sobre como organizar código e dados. Segundo Sebesta (2018), os principais paradigmas incluem:

Paradigma Imperativo: Baseia-se na execução sequencial de comandos que modificam o estado do programa. O programador especifica explicitamente os passos que o computador deve seguir. Linguagens como C e Pascal são exemplos clássicos deste paradigma.

Paradigma Funcional: Enfatiza o uso de funções matemáticas puras, evitando mudanças de estado e dados mutáveis. Linguagens como Haskell e Lisp seguem este paradigma.

Paradigma Orientado a Objetos: Organiza o código em torno de objetos que encapsulam dados e comportamentos. Este paradigma busca modelar sistemas computacionais de forma similar ao mundo real, utilizando conceitos como classes, objetos, herança e polimorfismo.

2.2 Paradigma Orientado a Objetos

O paradigma orientado a objetos surgiu da necessidade de criar sistemas mais organizados, reutilizáveis e de fácil manutenção. Booch et al. (2007) definem os pilares fundamentais deste paradigma:

Encapsulamento: Permite ocultar detalhes internos de implementação, expondo apenas interfaces necessárias para interação com o objeto. Isso promove segurança e modularidade no código, garantindo que mudanças internas não afetem outros componentes do sistema.

Herança: Possibilita que classes filhas herdem características e comportamentos de classes pais, promovendo reutilização de código e criação de hierarquias lógicas. Este mecanismo permite especializar comportamentos mantendo uma base comum entre classes relacionadas.

Polimorfismo: Permite que objetos de diferentes classes sejam tratados de forma uniforme através de interfaces comuns, aumentando a flexibilidade do sistema. Como destacado por Meyer (1997), o polimorfismo é essencial para criar sistemas extensíveis, pois permite que o mesmo código funcione com diferentes tipos de objetos sem modificações.

Abstração: Facilita a modelagem de conceitos complexos do mundo real através de representações simplificadas em classes e objetos. A abstração permite focar nos aspectos essenciais de uma entidade, ignorando detalhes irrelevantes para o contexto específico.

O polimorfismo, em particular, manifesta-se de várias formas na programação orientada a objetos. Cardelli e Wegner (1985) classificam os tipos de polimorfismo, sendo os mais relevantes para este trabalho o polimorfismo de inclusão (subtipos) e o polimorfismo paramétrico (generics), que permitem que uma única interface suporte múltiplas implementações específicas.

2.3 A Linguagem Java

Java, desenvolvida pela Sun Microsystems (atualmente Oracle), é uma linguagem que implementa completamente o paradigma orientado a objetos. Horstmann (2019) destaca as principais características que tornam Java adequada para desenvolvimento de sistemas orientados a objetos:

1. **Sintaxe clara e legível:** Java possui uma sintaxe próxima à linguagem natural, facilitando a leitura e compreensão do código.

2. **Suporte completo a polimorfismo:** Java oferece suporte nativo a herança, interfaces e classes abstratas, facilitando a implementação de hierarquias polimórficas complexas.
3. **Garbage Collection automático:** O gerenciamento de memória é automatizado, reduzindo a possibilidade de vazamentos de memória.
4. **Sistema de tipos robusto:** Java possui verificação rigorosa de tipos em tempo de compilação, ajudando a detectar erros antes da execução.
5. **Portabilidade:** O conceito "write once, run anywhere" permite que programas Java executem em diferentes plataformas sem modificações.
6. **Bibliotecas extensas:** A linguagem oferece um vasto conjunto de bibliotecas padrão que aceleram o desenvolvimento.

2.4 Sistemas de Gerenciamento de Coleções

Um sistema de gerenciamento de coleções é uma aplicação projetada para organizar, catalogar e facilitar o acesso a conjuntos de itens relacionados. No contexto de coleções de discos modernos, que frequentemente incluem múltiplos formatos, estes sistemas tipicamente precisam lidar com:

1. **Catalogação heterogênea:** Registro detalhado de diferentes tipos de mídia com informações específicas para cada formato.
2. **Operações especializadas:** Diferentes tipos de mídia requerem tratamentos específicos para cálculos de envio, armazenamento e manutenção.
3. **Análises comparativas:** Capacidade de comparar características entre diferentes formatos de forma uniforme.
4. **Extensibilidade:** Facilidade para adicionar novos tipos de mídia sem modificar código existente.

3. METODOLOGIA

O desenvolvimento do sistema de gerenciamento para coleção de discos seguiu uma abordagem sistemática baseada nos princípios do paradigma orientado a objetos, com ênfase especial na implementação de polimorfismo através de herança e interfaces. As etapas da metodologia incluíram:

1. **Análise de requisitos:** Identificação das funcionalidades essenciais para um sistema de coleção de discos multi-formato, incluindo gestão de diferentes tipos de mídia, artistas e operações especializadas para cada formato.
2. **Modelagem orientada a objetos:** Seguindo as recomendações de Booch et al. (2007), foi realizada a identificação das classes principais e suas hierarquias, definindo uma classe abstrata base e subclasses especializadas para cada tipo de mídia.
3. **Design de interfaces:** Criação de interfaces que definem contratos comuns para diferentes tipos de mídia, permitindo implementação polimórfica de comportamentos específicos.

4. **Implementação da hierarquia de classes:** Desenvolvimento de classes abstratas e concretas em Java, aplicando herança para compartilhar código comum e polimorfismo para especializar comportamentos.
5. **Desenvolvimento de funcionalidades polimórficas:** Implementação de métodos que demonstram polimorfismo em ação, onde o mesmo código trata diferentes tipos de objetos de forma uniforme.
6. **Testes e validação:** Verificação da corretude das funcionalidades implementadas através de casos de teste que demonstram o comportamento polimórfico do sistema.

Para implementação, foi escolhida a linguagem Java em sua versão mais recente, aproveitando suas características avançadas de orientação a objetos, incluindo suporte nativo a interfaces, classes abstratas e polimorfismo, conforme recomendado por Deitel e Deitel (2017).

4. PARADIGMA ESCOLHIDO

4.1 Justificativa da Escolha do Paradigma Orientado a Objetos

O paradigma orientado a objetos foi escolhido para este trabalho devido às seguintes razões, alinhadas com as observações de Horstmann (2019):

1. **Modelagem natural do domínio:** O sistema de coleção de discos envolve entidades claras (diferentes tipos de mídia, artistas, coleções) que podem ser facilmente representadas como classes e objetos, tornando o código mais intuitivo e próximo ao problema real.
2. **Hierarquias naturais:** Diferentes tipos de mídia (vinil, CD, digital) compartilham características comuns mas possuem comportamentos específicos, criando uma hierarquia natural que pode ser elegantemente modelada através de herança e polimorfismo.
3. **Reutilização de código:** A orientação a objetos promove a criação de componentes reutilizáveis, permitindo que classes base sejam utilizadas como fundação para especializações sem duplicação de código.
4. **Extensibilidade através de polimorfismo:** A implementação de polimorfismo permite que novos tipos de mídia sejam adicionados ao sistema sem modificar código existente, seguindo o princípio aberto-fechado de Bertrand Meyer (1997).
5. **Manutenibilidade:** O encapsulamento permite modificar implementações internas de classes específicas sem afetar outras partes do sistema, facilitando evoluções e correções.
6. **Tratamento uniforme de objetos heterogêneos:** O polimorfismo permite que o sistema trate diferentes tipos de mídia de forma uniforme através de interfaces comuns, simplificando operações complexas.

4.2 Critérios de Linguagem

A escolha de Java como linguagem para implementação do sistema foi baseada nos seguintes critérios, conforme discutido por Sebesta (2018):

1. **Legibilidade:** Java possui sintaxe clara e expressiva, com convenções de nomenclatura que facilitam a compreensão do código, especialmente importante em hierarquias complexas de classes.
2. **Suporte avançado à orientação a objetos:** Java oferece suporte completo a todos os conceitos de orientação a objetos, incluindo classes abstratas, interfaces, herança múltipla através de interfaces e polimorfismo dinâmico.
3. **Facilidade de implementação de polimorfismo:** A linguagem torna simples a implementação de comportamentos polimórficos através de métodos virtuais automáticos e resolução dinâmica de métodos.
4. **Confiabilidade:** O sistema de tipos forte e o tratamento obrigatório de exceções contribuem para a criação de software mais robusto, especialmente importante em hierarquias complexas.
5. **Eficiência:** A máquina virtual Java (JVM) oferece otimizações avançadas para chamadas de métodos polimórficos, garantindo desempenho adequado mesmo com uso intensivo de polimorfismo.
6. **Portabilidade:** Programas Java podem executar em diferentes sistemas operacionais sem modificações, aumentando a versatilidade da solução.
7. **Comunidade e recursos:** Java possui vasta comunidade de desenvolvedores e abundante documentação sobre padrões de design orientados a objetos, facilitando o desenvolvimento de arquiteturas sofisticadas.

5. IMPLEMENTAÇÃO DO SISTEMA

O sistema de gerenciamento para coleção de discos foi implementado utilizando Java para criar uma hierarquia robusta de classes que demonstra todos os pilares da orientação a objetos. A implementação seguiu as melhores práticas descritas por Deitel e Deitel (2017) e consistiu em uma abordagem estruturada que priorizou o uso efetivo de polimorfismo, herança e encapsulamento para criar um sistema extensível e de fácil manutenção.

5.1 Arquitetura do Sistema

O sistema foi estruturado utilizando uma arquitetura hierárquica que demonstra polimorfismo através de herança e interfaces:

Interface Media: Define o contrato fundamental que todos os tipos de mídia devem seguir, estabelecendo métodos como calculateShippingCost(), getFormatDetails() e getStorageRecommendation(). Esta interface permite que diferentes implementações sejam tratadas de forma polimórfica.

Classe abstrata Record: Serve como classe base para todos os tipos de mídia, implementando a interface Media e definindo comportamentos comuns. Contém métodos abstratos como getFormat() e getDurabilityRating() que devem ser implementados pelas subclasses, demonstrando o padrão Template Method.

Subclasses especializadas:

- **VinylRecord:** Especializa Record para discos de vinil, incluindo atributos específicos como RPM e tamanho, implementando cálculos de envio baseados no peso e fragilidade.
- **CDRecord:** Especializa Record para CDs, incluindo informações sobre remasterização e edição, com custos de envio reduzidos devido ao menor peso.
- **DigitalRecord:** Especializa Record para mídia digital, incluindo qualidade de áudio e tamanho do arquivo, com custo de envio zero por ser download instantâneo.

Classe Artist: Representa artistas musicais com encapsulamento adequado e métodos de comparação.

Classe RecordCollection: Gerencia coleções heterogêneas de diferentes tipos de mídia, demonstrando polimorfismo ao tratar todos os tipos através da interface comum.

5.2 Implementação de Polimorfismo

As principais funcionalidades que demonstram polimorfismo incluem:

1. **Cálculo polimórfico de custos de envio:** O método printShippingCosts() na classe RecordCollection demonstra polimorfismo ao chamar calculateShippingCost() em diferentes tipos de mídia, onde cada implementação calcula custos específicos.
2. **Recomendações especializadas de armazenamento:** Cada tipo de mídia implementa getStorageRecommendation() de forma específica, mas o código cliente trata todos uniformemente através da interface comum.
3. **Análise polimórfica de durabilidade:** O método printDurabilityAnalysis() demonstra como diferentes tipos de mídia podem ser avaliados uniformemente, mesmo tendo implementações específicas de getDurabilityRating().
4. **Sistema de busca por formato:** Utiliza polimorfismo para identificar tipos de mídia através do método getFormat(), implementado diferentemente em cada subclasse.

5.3 Demonstração dos Pilares da Orientação a Objetos

Encapsulamento: Todos os atributos são privados com acesso controlado através de métodos getter e setter. Cada classe mantém suas responsabilidades bem definidas.

Herança: As subclasses VinylRecord, CDRecord e DigitalRecord herdam funcionalidades comuns da classe Record, especializando apenas comportamentos específicos.

Polimorfismo: Múltiplas formas de polimorfismo são demonstradas:

- Polimorfismo de método (overriding)
- Polimorfismo paramétrico através de interfaces
- Polimorfismo de inclusão (subtipos tratados como supertipo)

Abstração: Classes abstratas e interfaces fornecem abstrações apropriadas que simplificam a interação com objetos complexos.

5.4 Código Fonte e Disponibilidade

O código completo do sistema demonstra implementação avançada de conceitos orientados a objetos:

Interfaces bem definidas: A interface Media estabelece contratos claros que garantem comportamento consistente entre diferentes implementações.

Métodos template: A classe abstrata Record define algoritmos gerais que são customizados pelas subclasses através de métodos abstratos.

Polimorfismo dinâmico: O sistema utiliza ligação dinâmica (dynamic binding) para resolver chamadas de métodos em tempo de execução, permitindo comportamento especializado sem comprometer a uniformidade da interface.

Extensibilidade: Novos tipos de mídia podem ser adicionados implementando a interface Media e especializando a classe Record, sem modificar código existente.

6. RESULTADOS E DISCUSSÃO

A implementação do sistema de gerenciamento para coleção de discos utilizando o paradigma orientado a objetos com Java, enfatizando polimorfismo e herança, demonstrou resultados significativos que confirmam as vantagens deste paradigma destacadas por Booch et al. (2007) e Meyer (1997).

6.1 Análise da Implementação

A aplicação sistemática de polimorfismo resultou em um código elegante e altamente extensível. A separação entre interface comum (através da classe abstrata Record) e implementações específicas (VinylRecord, CDRecord, DigitalRecord) demonstra como polimorfismo pode ser usado para gerenciar complexidade.

A implementação de diferentes comportamentos através de polimorfismo ficou evidente nos métodos especializados:

- **VinylRecord** calcula 15% do preço como custo de envio (peso e fragilidade)
- **CDRecord** calcula 8% do preço (mais leve)
- **DigitalRecord** tem custo zero (download instantâneo)

6.2 Vantagens do Polimorfismo Implementado

Comparando com uma possível implementação sem polimorfismo, a solução orientada a objetos apresentou vantagens significativas:

1. **Eliminação de código condicional:** Sem polimorfismo, seria necessário usar estruturas if-else ou switch para determinar o tipo de mídia e executar código específico. O polimorfismo elimina essa necessidade através de dispatch dinâmico.
2. **Extensibilidade sem modificação:** Novos tipos de mídia podem ser adicionados criando novas subclasses, seguindo o Princípio Aberto-Fechado de Meyer (1997) - o sistema está aberto para extensão mas fechado para modificação.
3. **Reutilização maximizada:** A classe base Record encapsula comportamentos comuns, enquanto subclasses especializam apenas o que é necessário, evitando duplicação de código.
4. **Manutenibilidade aprimorada:** Mudanças em comportamentos específicos de um tipo de mídia afetam apenas sua classe correspondente, não impactando outras partes do sistema.
5. **Testabilidade:** Cada implementação pode ser testada independentemente, e o comportamento polimórfico pode ser verificado através de interfaces comuns.

6.3 Análise de Desempenho

O uso de polimorfismo em Java beneficia-se das otimizações da JVM. A máquina virtual Java utiliza técnicas como inline caching e profile-guided optimization para otimizar chamadas de métodos virtuais, minimizando o overhead do polimorfismo. Em testes informais, o sistema demonstrou desempenho adequado mesmo com uso intensivo de chamadas polimórficas.

6.4 Desafios Encontrados

Apesar das vantagens significativas, alguns desafios foram encontrados durante a implementação:

1. **Complexidade de design inicial:** O planejamento de hierarquias polimórficas requer análise cuidadosa do domínio para identificar abstrações apropriadas e evitar hierarquias excessivamente complexas.
2. **Curva de aprendizado:** Para desenvolvedores menos experientes, conceitos como classes abstratas, interfaces e polimorfismo podem representar uma barreira inicial de compreensão.
3. **Debugging mais complexo:** Rastrear o fluxo de execução em código polimórfico pode ser mais desafiador, pois a resolução de métodos ocorre em tempo de execução.

7. CONCLUSÕES

A implementação de um sistema de gerenciamento para coleção de discos utilizando o paradigma orientado a objetos com Java, com ênfase especial em polimorfismo e herança, demonstrou de forma conclusiva a eficácia desta abordagem para o desenvolvimento de sistemas extensíveis e

manuteníveis, confirmando as observações teóricas de Booch et al. (2007), Meyer (1997) e Horstmann (2019). As principais conclusões obtidas são:

1. **Eficácia do polimorfismo:** O polimorfismo mostrou-se fundamental para criar um sistema que trata diferentes tipos de mídia de forma uniforme enquanto preserva comportamentos específicos. A capacidade de adicionar novos tipos sem modificar código existente demonstra o poder da extensibilidade polimórfica.
2. **Elegância da solução:** A combinação de classes abstratas, interfaces e herança resultou em uma arquitetura elegante que reflete naturalmente a estrutura do domínio do problema, onde diferentes tipos de mídia compartilham características mas possuem particularidades.
3. **Reutilização maximizada:** A implementação de uma classe base comum (Record) com especializações específicas (VinylRecord, CDRecord, DigitalRecord) demonstrou como herança pode ser usada efetivamente para maximizar reutilização de código sem comprometer flexibilidade.
4. **Manutenibilidade superior:** O encapsulamento de comportamentos específicos em classes especializadas, combinado com interfaces bem definidas, resultou em um sistema onde mudanças são localizadas e não propagam efeitos colaterais indesejados.
5. **Extensibilidade comprovada:** A facilidade com que novos tipos de mídia podem ser adicionados ao sistema (implementando a interface Media e especializando Record) demonstra como polimorfismo facilita evolução de software sem comprometer estabilidade.
6. **Adequação do Java:** A linguagem Java provou ser uma escolha excepcional para implementação de sistemas orientados a objetos complexos, oferecendo suporte robusto a todos os conceitos necessários e otimizações de desempenho que tornam o polimorfismo viável em aplicações práticas.
7. **Padrões de design emergentes:** A implementação naturalmente incorporou padrões de design reconhecidos, como Template Method (na classe abstrata Record) e Strategy (através das diferentes implementações de cálculo de custos), demonstrando como bom design orientado a objetos converge para soluções estabelecidas.

O sistema desenvolvido serve como uma demonstração prática de como os pilares da orientação a objetos - encapsulamento, herança, polimorfismo e abstração - trabalham em conjunto para criar soluções robustas e elegantes. A implementação de polimorfismo, em particular, transformou o que poderia ser código procedural complexo com múltiplas estruturas condicionais em uma solução limpa e extensível.

Em trabalhos futuros, o sistema poderia ser expandido para demonstrar conceitos adicionais como padrões de design mais avançados (Factory, Observer, Decorator), programação genérica (Java Generics) para maior type safety, e integração com frameworks modernos que aproveitam polimorfismo para injeção de dependências. A base polimórfica estabelecida facilita todas essas extensões sem comprometer a arquitetura existente.

A experiência de desenvolvimento confirmou que a compreensão profunda de polimorfismo e herança é essencial para criar software orientado a objetos de qualidade, e que investir tempo no design adequado de hierarquias de classes resulta em benefícios significativos a longo prazo em termos de manutenibilidade, extensibilidade e elegância da solução.

REFERÊNCIAS

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML: Guia do Usuário**. 2ª ed. Rio de Janeiro: Elsevier, 2007.

CARDELLI, L.; WEGNER, P. On understanding types, data abstraction, and polymorphism. **ACM Computing Surveys**, v. 17, n. 4, p. 471-523, 1985.

DEITEL, P.; DEITEL, H. **Java: Como Programar**. 10ª ed. São Paulo: Pearson, 2017.

HORSTMANN, C. S. **Core Java Volume I - Fundamentals**. 11ª ed. Upper Saddle River: Pearson, 2019.

MEYER, B. **Object-Oriented Software Construction**. 2ª ed. Upper Saddle River: Prentice Hall, 1997.

SEBESTA, R. W. **Conceitos de Linguagens de Programação**. 11ª ed. Porto Alegre: Bookman, 2018.