# Sistema de Gerenciamento de Itens

Este é um sistema desenvolvido em **Java** para gerenciar um acervo de itens culturais (livros e filmes) com interface gráfica amigável. Permite cadastrar, buscar, listar, exportar e importar informações de maneira organizada e eficiente.

### Objetivo Principal

Criar uma aplicação desktop funcional que demonstra os conceitos fundamentais da **Programação Orientada a Objetos** e **Desenvolvimento de Software**.

## Como Está Dividido o Sistema

### Camadas Lógicas Bem Definidas

**1. Camada de Apresentação (GUI)**

- **Classe Principal**: `ItemManagerGUI`
- **Tecnologia**: Java Swing
- **Função**: Interface com o usuário final

**2. Camada de Serviço (Lógica de Negócio)**

- **Classe Principal**: `GerenciadorItens`
- **Função**: Coordena todas as operações do sistema

**3. Camada de Modelo (Dados)**

- **Classes**: `Item`, `Livro`, `Filme`
- **Função**: Representa os dados e regras do negócio

**4. Camada de Exceções**

- **Classes**: `CampoVazioException`, `DuplicadoException`
- **Função**: Tratamento específico de erros

## Estrutura Detalhada do Projeto

### Pacotes e Suas Responsabilidades

```
src/
├── 📁 exceptions/         # Exceções personalizadas
│   ├── CampoVazioException.java
│   └── DuplicadoException.java
│
├── 📁 model/              # Modelos de dados (CORAÇÃO DO SISTEMA)
│   ├── Item.java          # Classe abstrata base
│   ├── Livro.java         # Especialização para livros
│   └── Filme.java         # Especialização para filmes
│
├── 📁 service/            # Lógica de negócio
│   └── GerenciadorItens.java  # Cérebro do sistema
│
└── 📁 gui/                # Interface gráfica
    └── ItemManagerGUI.java    # Interface com usuário
```

# Comentários sobre o Código e Funções

### Conceitos de POO Aplicados

#### 1. Herança e Polimorfismo

```
// Item.java - CLASSE ABSTRATA BASE
public abstract class Item {
    protected String titulo;
    protected String descricao;
    protected LocalDate dataCadastro;

    // MÉTODO ABSTRATO - obriga as subclasses a implementarem
    public abstract String exibirDetalhes();
}

// Livro.java - HERDA de Item
public class Livro extends Item {
    private String autor;
    private int numeroPaginas;

    // SOBRESCRITA do método abstrato
    @Override
    public String exibirDetalhes() {
```

```
        return String.format("LIVRO: %s\nAutor: %s\nPáginas: %d",
            titulo, autor, numeroPaginas);
    }

}
```

**Explicação**: `Item` é a classe pai abstrata, enquanto `Livro` e `Filme` são especializações. Isso permite tratar todos os itens de forma uniforme.

**2. Encapsulamento**

```
public class Livro extends Item {
    private int numeroPaginas;  // ATRIBUTO PRIVADO

    // GETTER e SETTER com VALIDAÇÃO
    public void setNumeroPaginas(int numeroPaginas) {
        if (numeroPaginas <= 0) {
            throw new IllegalArgumentException("Número de páginas deve ser maior que
zero");
        }
        this.numeroPaginas = numeroPaginas;
    }

}
```

**Explicação**: Os atributos são privados e acessados apenas através de métodos, permitindo controle e validação.

**Tratamento de Exceções Personalizadas**

```
// exceptions/DuplicadoException.java
public class DuplicadoException extends Exception {
    public DuplicadoException(String titulo) {
        super("Já existe um item com o título: " + titulo);
    }
}

// service/GerenciadorItens.java
public void adicionarItem(Item item) throws CampoVazioException,
DuplicadoException {
    // Verifica duplicação
    for (Item i : itens) {
        if (i.getTitulo().equalsIgnoreCase(item.getTitulo())) {
```

```
            throw new DuplicadoException(item.getTitulo());  // LANÇA EXCEÇÃO
PERSONALIZADA
        }
    }
    itens.add(item);

}
```

## Fluxo de uma Operação Típica

### Exemplo: Adicionar um Livro

1. **Usuário** preenche formulário na interface
2. **GUI** valida campos básicos e cria objeto `Livro`
3. **GUI** chama `gerenciador.adicionarItem(livro)`
4. **Gerenciador** valida regras de negócio e adiciona à lista
5. **GUI** exibe confirmação ou trata erros

```
// Fluxo no GerenciadorItens.java
public void adicionarItem(Item item) throws CampoVazioException,
DuplicadoException {
    // 1. VALIDA CAMPOS OBRIGATÓRIOS
    if (item.getTitulo() == null || item.getTitulo().trim().isEmpty()) {
        throw new CampoVazioException("título");
    }

    // 2. VERIFICA DUPLICAÇÃO
    for (Item i : itens) {
        if (i.getTitulo().equalsIgnoreCase(item.getTitulo())) {
            throw new DuplicadoException(item.getTitulo());
        }
    }

    // 3. ADICIONA À LISTA
    itens.add(item);

}
```

## Funcionalidades Destacadas

### 1. Busca Inteligente

```
public List<Item> buscarPorTitulo(String titulo) {
   return itens.stream()
         .filter(item -> item.getTitulo().toLowerCase().contains(titulo.toLowerCase()))
         .collect(Collectors.toList());

}
```

**Característica**: Busca parcial case-insensitive usando Java Streams

**2. Persistência em Arquivo**

```
public void exportarParaArquivo(String caminho) throws IOException {
   try (BufferedWriter writer = new BufferedWriter(new FileWriter(caminho))) {
      for (Item item : itens) {
         writer.write(item.toExportString());  // POLIMORFISMO EM AÇÃO!
         writer.newLine();
      }
   }

}
```

**Característica**: Cada tipo sabe como se serializar graças ao polimorfismo

**3. Estatísticas com Streams**

```
public Map<String, Long> contarPorTipo() {
   return itens.stream()
         .collect(Collectors.groupingBy(
            item -> item.getClass().getSimpleName(),
            Collectors.counting()
         ));

}
```

**Característica**: Uso moderno de Streams API para agregações

# Interface Gráfica - Destaques

**CardLayout para Formulários Dinâmicos**

```
// Mostra campos específicos baseado no tipo selecionado
cardLayout = new CardLayout();
panelEspecifico = new JPanel(cardLayout);
```

```
panelEspecifico.add(panelLivro, "Livro");
panelEspecifico.add(panelFilme, "Filme");

// Quando usuário seleciona tipo:

cardLayout.show(panelEspecifico, tipoSelecionado);
```

### Tratamento de Eventos

```
btnAdicionar.addActionListener(e -> adicionarItem());  // LAMBDA EXPRESSIONS
```