Diário Cultural:
Sistema de Acompanhamento de Leitura e Audiovisual

Enzo Cauã da S. Barbosa

[1]Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana – BA, Brasil – 44036-900

caua7uefs@gmail.com

Abstract. *The second phase of the Diário Cultural project focused on implementing data persistence and restructuring the system using the MVC design pattern. Data persistence was achieved through binary serialization in Java, ensuring automatic saving and loading of user information. The adoption of MVC improved code organization and separation of concerns. Tests confirmed data integrity after serialization and deserialization. These enhancements make the system more robust and ready for future developments, such as a graphical user interface and improved exception handling.*

Resumo. *A segunda fase do projeto Diário Cultural teve como objetivo implementar a persistência de dados e reorganizar a estrutura do sistema com base no padrão de projeto MVC. A persistência foi realizada por meio da serialização binária em Java, garantindo o salvamento e a recuperação automática das informações. A adoção do padrão MVC trouxe maior organização e separação de responsabilidades no código. Os testes comprovaram a integridade dos dados após serialização e desserialização. Essas melhorias tornam o sistema mais robusto e preparado para evoluções futuras, como a inclusão de uma interface gráfica e o aprimoramento do tratamento de exceções.*

1. Introdução

Na continuidade do desenvolvimento da aplicação *Diário Cultural*, esta segunda fase tem como foco a implementação da persistência de dados, aspecto essencial para qualquer sistema que visa preservar informações entre diferentes sessões de uso. Na versão anterior, todos os dados inseridos eram perdidos ao encerrar a execução do programa, limitando a utilidade prática da ferramenta para o usuário.

A persistência de dados permite que registros inseridos — como livros lidos, filmes assistidos, avaliações e comentários — sejam salvos em disco e recuperados posteriormente. Essa funcionalidade torna o sistema mais confiável, duradouro e coerente com a proposta de construir um acervo pessoal de experiências culturais.

Além disso, nesta etapa também foi realizada a refatoração da estrutura interna do sistema, com a adoção do padrão de projeto MVC (Model-View-Controller). Essa mudança visa promover uma separação mais clara entre as responsabilidades de cada componente, facilitando a manutenção, a extensibilidade do sistema e a futura integração com uma interface gráfica.

A implementação da persistência foi realizada por meio da serialização binária dos objetos Java [Oracle 2025b], garantindo um armazenamento simples e eficaz sem a necessidade de bibliotecas externas. Com isso, os dados do acervo passam a ser automaticamente carregados ao iniciar o programa e salvos a cada modificação ou encerramento, proporcionando uma experiência mais fluida e confiável ao usuário.

As próximas seções deste relatório apresentam os fundamentos teóricos que sustentam essa etapa, a metodologia aplicada, os resultados obtidos e as conclusões relativas à evolução do sistema na Fase 2.

2. Fundamentação Teórica

A segunda fase do projeto *Diário Cultural* introduz conceitos fundamentais relacionados à engenharia de software e à programação orientada a objetos, com destaque para o padrão arquitetural MVC, a persistência de dados, o tratamento de exceções e o uso da serialização binária na linguagem Java.

2.1. Padrão MVC

O padrão Model-View-Controller (MVC) é uma abordagem arquitetural que separa a lógica do sistema em três camadas principais:

- **Modelo (Model)**: responsável pela representação e manipulação dos dados do sistema. Contém as regras de negócio e o estado da aplicação.
- **Visão (View)**: responsável pela apresentação das informações ao usuário. No estágio atual, essa camada é implementada por menus e interações em terminal.
- **Controlador (Controller)**: faz a mediação entre o modelo e a visão, processando as ações do usuário e atualizando o estado do sistema.

A adoção desse padrão melhora a organização do código, facilita a manutenção e permite maior escalabilidade do sistema, especialmente em futuras expansões como a adição de uma interface gráfica [Gamma et al. 1995].

2.2. Persistência de Dados

A persistência de dados consiste na capacidade de manter as informações do sistema disponíveis mesmo após o encerramento da aplicação [Sommerville 2011]. No contexto deste projeto, ela foi implementada por meio da serialização binária dos objetos em arquivos locais.

Ao utilizar esse mecanismo, o estado dos objetos do sistema — como livros, filmes, séries e avaliações — é convertido em uma sequência de bytes e armazenado em disco. Posteriormente, esses dados podem ser restaurados por meio da desserialização, recriando os objetos em memória com os mesmos atributos e valores.

2.3. Exceções

Durante a manipulação de arquivos e a leitura de dados do usuário, podem ocorrer diversas situações de erro, como arquivos inexistentes ou dados mal formatados. O Java oferece uma estrutura robusta para o tratamento de exceções [Oracle 2025a], permitindo capturar e lidar com falhas em tempo de execução, evitando o encerramento inesperado da aplicação e promovendo uma experiência mais estável para o usuário.

3. Metodologia

3.1. Definição de Requisitos e Funcionalidades

A aplicação *Diário Cultural* foi concebida para permitir ao usuário o registro e acompanhamento de obras culturais consumidas, com foco em livros, filmes e séries. As funcionalidades já implementadas incluem:

- Cadastro de mídias: inclusão de informações detalhadas sobre livros, filmes e séries.
- Avaliação de obras: pontuação de 1 a 5 estrelas, marcação como visto/lido, data e registro de impressões.
- Busca por conteúdo: filtragem por critérios como título, autor, ano, elenco, entre outros.
- Listagem com ordenação e filtros: visualização de conteúdos com base em avaliações, gêneros e datas.

A essa lista de funcionalidades, foi adicionada a persistência de dados, que tem como requisito garantir que todas as informações registradas pelo usuário sejam salvas em disco e recuperadas automaticamente nas próximas execuções da aplicação. Essa funcionalidade precisa garantir:

- Carregamento dos dados existentes ao iniciar o programa;
- Salvamento automático sempre que o acervo for alterado;
- Armazenamento em arquivos locais de forma segura e transparente ao usuário.

3.1.1. Serialização Binária e Interface `Serializable`

A persistência de dados foi implementada por meio da serialização binária, uma técnica que permite converter objetos Java em uma sequência de bytes para armazenamento em disco. Para isso, a classe mídia e todas as suas subsclasses implementam a interface marcadora `Serializable`.

Essa interface não requer a implementação de métodos, mas indica que os objetos daquela classe podem ser serializados e desserializados. Com isso, o estado completo do acervo pode ser salvo em um arquivo binário e restaurado automaticamente em execuções futuras, sem necessidade de conversões intermediárias para formatos como JSON ou XML.

A escolha pela serialização binária foi motivada por sua simplicidade, compatibilidade direta com o Java e adequação ao escopo do projeto. Essa abordagem garante desempenho eficiente e facilidade de implementação, além de manter a integridade dos dados.

3.2. Descrição de Alto Nível do Sistema

O sistema foi estruturado com base no padrão de projeto Model-View-Controller (MVC), o que resultou em uma reorganização das classes e pacotes para separar claramente as responsabilidades entre modelo (dados e lógica), visão (interface textual) e controlador (fluxo da aplicação).

O modelo contém classes como `Midia`, `Livro`, `Filme`, `Serie` e `Acervo`, responsáveis por representar as entidades principais da aplicação e suas regras de negócio. A visão, neste estágio, é composta por menus exibidos no terminal e métodos que recebem entradas do usuário. O controlador coordena a lógica entre os menus e as operações no acervo.

Com a adição da persistência de dados, o sistema agora realiza automaticamente a serialização a lista de cada tipo de mídia do objeto `Acervo` para um arquivo binário. Esse processo é realizado ao final de cada modificação no acervo, bem como na finalização do programa. Ao iniciar a aplicação, o sistema verifica a existência do arquivo e realiza a desserialização, restaurando os dados previamente salvos.

Essa implementação foi facilitada pela utilização da interface `Serializable`, o que permitiu manter o sistema coeso e consistente com os princípios da orientação a objetos.

3.3. Processo de Codificação

A codificação seguiu princípios de boas práticas da orientação a objetos e foi adaptada para atender à arquitetura MVC. Houve uma refatoração das classes e pacotes originalmente criados na Fase 1, separando as responsabilidades e preparando a estrutura para futura integração com uma interface gráfica.

A serialização foi integrada ao fluxo de forma automática: sempre que uma operação altera o conteúdo do acervo (como adicionar, editar ou avaliar uma mídia), o sistema dispara o salvamento em disco. Do mesmo modo, ao iniciar a aplicação, o sistema realiza a leitura do arquivo salvo anteriormente e restaura o estado completo do acervo.

Essa abordagem reduziu a complexidade do código no que diz respeito à manipulação de arquivos, uma vez que a serialização lida com toda a estrutura dos objetos de forma transparente ao programador.

Diferente da fase anterior, os testes realizados nesta etapa focaram na validação da serialização e desserialização, garantindo que os dados sejam corretamente salvos e recuperados entre execuções. Os testes verificam, por exemplo, se os objetos estão sendo preservados nos arquivos e se eles estão sendo lidos corretamente.

4. Resultados e Discussões

4.1. Manual de Uso

O manual de uso permanece inalterado em relação à primeira fase do projeto. O sistema continua operando por meio de menus e entradas no terminal, utilizando classes e métodos acessíveis que permitem o cadastro, avaliação, busca e listagem de mídias.

A principal mudança é que agora a persistência dos dados ocorre de forma automática e transparente para o usuário. Ao iniciar a aplicação, o sistema carrega os dados previamente salvos por meio de desserialização. Durante o uso, qualquer modificação feita no acervo é imediatamente refletida no arquivo de armazenamento. Além disso, ao encerrar a aplicação, os dados são novamente salvos, garantindo que nenhuma informação seja perdida entre sessões.

4.2. Dados de Entrada e Saída

As entradas continuam sendo fornecidas pelo usuário em tempo de execução, por meio do terminal, com o uso da classe `Scanner`. As informações solicitadas incluem título, autor, gênero, ano de lançamento, nota (de 1 a 5), e comentários.

As saídas permanecem no formato textual, apresentadas no terminal por meio de `System.out.println()`, exibindo os dados cadastrados, resultados de buscas e resumos formatados.

Exemplo de entrada:

```
Livro livro = new Livro("Acotar", "Corte de Espinhos e Rosas",
"Romance", "2017", "Sarah", "Galera", "978-3-16-148410-0", false);
```

Exemplo de saída:

```
Título: Acotar
Título Original: Corte de Espinhos e Rosas
Gênero: Romance
Ano de Lançamento: 2017
Autor: Sarah
Editora: Galera
ISBN: 978-3-16-148410-0
Possui exemplar: Não
Nota: 0
Review: null
Foi lido: Não
```

4.3. Testes Realizados

Nesta fase do projeto, os testes concentraram-se na validação da persistência de dados por meio da serialização e desserialização de objetos representando livros, filmes e séries.

Dois testes principais foram desenvolvidos:

4.3.1. Teste de Serialização (`TesteSerial`)

Este teste gera dados fictícios e os armazena em arquivos binários. Ele realiza os seguintes passos:

1. Criação de listas de mídias fictícias (`Livro`, `Filme`, `Serie`) com dados pre-definidos;
2. Impressão da quantidade de mídias geradas;
3. Serialização das listas usando métodos da classe `Serializacao`.

Saída esperada:

```
Livros gerados: 5
Filmes gerados: 5
Séries geradas: 5
Objetos salvos com sucesso nos arquivos!
```

Este teste comprova que o sistema é capaz de armazenar corretamente os dados em arquivos locais, utilizando serialização binária.

4.3.2. Teste de Desserialização (`TesteDeserial`)

Após a escrita dos arquivos, o segundo teste verifica se a desserialização restaura corretamente os objetos e seus atributos. As etapas incluem:

1. Leitura dos arquivos binários gerados;
2. Armazenamento dos objetos recuperados em listas;
3. Impressão da quantidade e dos títulos das mídias restauradas.

Saída esperada (exemplo):

```
Livros desserializados: 5
- Dom Quixote
- 1984
- O Hobbit
- A Revolução dos Bichos
- O Senhor dos Anéis

Filmes desserializados: 5
- Matrix
- Interestelar
- O Poderoso Chefão
- Forrest Gump
- Vingadores

Séries desserializadas: 5
- Breaking Bad
- Stranger Things
- Game of Thrones
- The Office
- Dark
```

Esse teste garante que os dados mantêm sua integridade após o processo de escrita e leitura. Títulos, atributos e listas internas (como diretores e atores) são preservados fielmente, demonstrando a eficácia da solução adotada para persistência.

Os testes foram realizados com sucesso e servem como evidência da robustez do mecanismo de serialização binária implementado no sistema.

4.4. Erros e Limitações

A introdução da persistência de dados e da arquitetura MVC solucionou limitações identificadas na fase anterior. No entanto, algumas restrições ainda permanecem:

- **Validação de entradas do usuário**: continua sendo necessário implementar verificações mais robustas para tipos e formatos incorretos;
- **Tratamento de exceções**: embora melhorado, ainda existem situações específicas que podem causar falhas não tratadas;
- **Interface textual limitada**: a interação continua sendo feita por menus no terminal, o que exige maior familiaridade do usuário com comandos textuais. Este aspecto será abordado na fase seguinte, com a introdução da interface gráfica.

5. Conclusão

A segunda fase do desenvolvimento da aplicação *Diário Cultural* representou um avanço significativo em relação à versão inicial. As principais melhorias introduzidas foram a implementação da persistência de dados por meio da serialização binária e a adoção do padrão de projeto MVC, que juntos elevaram o nível de organização, confiabilidade e escalabilidade do sistema.

A persistência de dados garantiu que as informações cadastradas fossem mantidas entre sessões, promovendo uma experiência mais completa e contínua para o usuário. A escolha pela serialização binária mostrou-se adequada ao escopo do projeto, oferecendo uma solução eficiente, prática e totalmente integrada à linguagem Java.

A refatoração do código em conformidade com o padrão Model-View-Controller possibilitou uma melhor separação de responsabilidades e prepara o sistema para futuras evoluções, como a incorporação de uma interface gráfica.

Os testes realizados comprovaram a integridade e o correto funcionamento do mecanismo de persistência, validando tanto o processo de salvamento quanto de recuperação dos dados. Os resultados obtidos demonstram que o sistema está preparado para evoluir com segurança nas próximas fases.

Como próximos passos, destaca-se a necessidade de aprimorar a interface com o usuário, substituindo os menus textuais por uma interface gráfica mais intuitiva, além de aperfeiçoar o tratamento de exceções e a validação das entradas. Tais melhorias, previstas para a terceira fase, devem proporcionar uma experiência ainda mais rica e acessível aos usuários finais.

References

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Oracle (2025a). Exceptions in java. `https://docs.oracle.com/javase/tutorial/essential/exceptions/`. Acesso em: maio 2025.

Oracle (2025b). Java object serialization specification. `https://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html`. Acesso em: maio 2025.

Sommerville, I. (2011). *Software Engineering*. Pearson, 9 edition.