UNIVERSIDADE FEDERAL DE GOIÁS
CAMPUS SAMAMBAIA
INSTITUTO DE INFORMÁTICA
PROGRAMAÇÃO ORIENTADA A OBJETOS
PROF. DIRSON SANTOS DE CAMPOS
PROFA. RENATA DUTRA BRAGA

FERNANDO MALTEZ BEZERRA E PEDRO CÉZAR SILVA FERREIRA

ATIVIDADE SUPERVISIONADA 1: SISTEMA DE GESTÃO DE BIBLIOTECA

GOIÂNIA
2023

1. DESENVOLVIMENTO TEÓRICO

No contexto do sistema de gestão da biblioteca, devemos levar em conta de que é um ambiente com um elevado número de objetos físicos, nesse caso livros, os quais possuem características próprias, dentre elas nome, autor, edição, editora, ano, além de poderem estar ou não disponíveis para empréstimo. Além disso, ele deve contar com a possibilidade de alteração dos registros, com o intuito de atualizar a situação atual de um livro, permitir o cadastro de novos livros no acervo, além de armazenar as informações cadastrais daqueles que em algum momento fizerem uso da biblioteca.

Sendo assim, temos como uma solução simples e viável, no quesito de programação, implementar junto ao banco de dados da biblioteca um sistema (software) que permita a manipulação de todos esses dados, usando para isso, dentre outras coisas, as Streams do Java.

Streams consistem em fluxos de entrada e saída de dados, que permitem a comunicação entre arquivos e dispositivos de input e output, como por exemplo teclados e monitores. Dentro do universo das Streams, temos como principais classes de entrada o FileInputStream e o DataInputStream, e, para saída, FileOutputStream e o DataOutputStream. Destaco também as classes FileReader, BufferedReader, FileWriter e PrinterWriter, que serão abordadas com mais detalhes a seguir.

- **Aplicações Input:**

A primeira parte é a da entrada de dados (input), que seria composta por classes do Java como FileInputStream e DataInputStream, que por sí só já conseguiriam cumprir o papel de abrir arquivos preexistentes no banco de dados da biblioteca, e fazer a leitura dos atributos contidos no arquivo desejado (atrelado a algum livro ou perfil de cadastro).

Em caso de querermos uma leitura mais especializada desse arquivo, podemos utilizar as classes de "Reading", dentre elas FileReader e BufferedReader, que nos darão uma visão mais ampla do que está contido naquele arquivo, uma vez que permitem a leitura de atributos isolados e linhas inteiras de caracteres.

- *Exemplo 1: Sistema de Busca*

Em um sistema de busca hipotético, onde cada livro físico possui um arquivo equivalente no banco de dados, e esse arquivo possui atributos, dentre eles um booleano, poderíamos relacioná-lo à disponibilidade do livro, recebendo o valor "true" quando disponível e "false" quando indisponível. A partir disso, o sistema de busca, utilizando as funções de leitura e input, teria acesso a esse atributo do arquivo, devolvendo como resposta à solicitação de busca a possibilidade ou não de pegar emprestado aquele livro.

- *Exemplo 2: Validação de empréstimo*

Com o intuito de evitar perdas e garantir a manutenção do acervo, podemos criar um sistema que verificará a credibilidade da pessoa que está solicitando o empréstimo de um livro. Isso pode ser feito por meio das funções de leitura e input, que revelariam para um dos funcionários da biblioteca o histórico do solicitante, em seguida julgando se ele poderá ou não realizar o empréstimo do livro.

- **Aplicações Output:**

A segunda parte consiste na saída de dados (output), que possui as classes FileOutputStream e DataOutputStream como duas das mais importantes, realizando funções de criação de arquivos e adição de atributos. Em comunhão a essas, destacamos as funções de "Writtinng" FileWriter e PrinterWriter, que, de maneira geral, nos permitem adicionar linhas inteiras de caracteres a um arquivo preexistente.

- Exemplo 1: Digitalização do acervo

Dentro do ambiente da biblioteca contamos com a presença de milhares de livros. Com o intuito de facilitar a preservação desse enorme acervo, devemos criar, do banco de dados, um arquivo específico para cada livro, contendo atributos como quantidade, nome, editora, etc. Para tal tarefa podemos desenvolver um software que faz uso de funções de output e escrita, criando os arquivos e adicionando atributos a cada um deles.

- Exemplo 2: Documentação de infrações

Assim como qualquer outra instituição, a Biblioteca da UFG também possui seu conjunto de regras, que visam o bom funcionamento de toda a estrutura. Com o intuito de documentar infrações e prevenir problemas futuros, podemos utilizar de funções de output e escrita para adicionar ao perfil de um usuário observações acerca de transgressões que ele tenha cometido. Essas observações ficarão salvas em um histórico pessoal, sendo, posteriormente, usado para julgar o usuário como apto ou não a se beneficiar de recursos da biblioteca.

2. ANÁLISE DAS IDES

O Eclipse e o IntelliJ IDEA são ambientes altamente populares para desenvolvimento em Java, cada um com suas próprias características distintivas. O Eclipse é renomado por sua extensibilidade por meio de plugins, proporcionando uma ampla variedade de extensões para diversas tarefas de desenvolvimento. Apesar de não apresentar recursos específicos integrados para a manipulação direta de arquivos e *streams*, sua flexibilidade permite a inclusão de plugins especializados nessas operações. Adicionalmente, o Eclipse oferece suporte completo às APIs

padrão do Java para manipulação de arquivos e *streams*, permitindo que os desenvolvedores utilizem todas as funcionalidades disponíveis nessas bibliotecas.

Por outro lado, o IntelliJ IDEA se destaca por sua ampla gama de recursos e ferramentas avançadas que simplificam o desenvolvimento. Possui funcionalidades robustas de refatoração que podem ser particularmente úteis ao trabalhar com arquivos e *streams*. A capacidade de renomear, extrair métodos e outras operações de refatoração facilitam a organização e a manipulação de código relacionado a essas operações. Além disso, o IntelliJ IDEA oferece ferramentas de análise estática que podem auxiliar na identificação de problemas potenciais em operações de manipulação de arquivos e *streams*.

Ambas as IDEs oferecem um ambiente rico para o desenvolvimento Java, mas a escolha entre o Eclipse e o IntelliJ IDEA para manipulação de arquivos e streams muitas vezes se resume à preferência pessoal do desenvolvedor e ao fluxo de trabalho específico de cada projeto.

3. CONCORRÊNCIA, SINCRONIZAÇÃO E COMUNICAÇÃO ENTRE THREADS

Para um sistema de gestão de biblioteca que permite operações simultâneas, como busca, empréstimos e devoluções de livros por diferentes usuários, é fundamental implementar concorrência com cuidado e garantir a sincronização para acessar os recursos compartilhados, como os arquivos de dados do acervo. Dessa forma, a aplicação de *locks* se torna essencial para assegurar a exclusão mútua durante o acesso aos recursos compartilhados, como os arquivos de dados. Essa medida evita que múltiplas *threads* modifiquem os dados ao mesmo tempo, prevenindo condições de corrida, onde diferentes operações interferem entre si.

Além disso, a sincronização das operações é de extrema importância para manter a consistência dos dados. Por exemplo, ao realizar um empréstimo, é necessário bloquear o acesso ao livro em questão e atualizar o status de empréstimo para evitar que dois usuários peguem o mesmo livro simultaneamente. No que tange à comunicação entre as *threads*, o uso de métodos como **wait(), notify()** e **notifyAll()** é recomendado. O wait() permite que uma *thread* aguarde até que uma condição específica seja atendida antes de prosseguir, enquanto o notify() e o notifyAll() notificam outras *threads* quando uma condição desejada é alcançada, permitindo que elas prossigam ou executem ações determinadas.

Nesse sentido, gerenciar operações concorrentes também demanda um controle de acesso adequado aos recursos compartilhados. Por exemplo, durante uma busca no acervo,

outras operações como empréstimos ou devoluções podem ser permitidas, mas a modificação dos dados do livro pode ser bloqueada até que a busca seja concluída. Garantir transações atômicas é igualmente relevante para o funcionamento consistente do sistema. Isso significa que operações críticas, como empréstimos, devem ser executadas de maneira atômica, garantindo que, se uma parte da operação falhar, o sistema possa retornar a um estado consistente.

Por fim, testes rigorosos são essenciais para identificar possíveis problemas de concorrência, como condições de corrida ou *deadlocks*. Um design de software robusto aliado a uma estratégia clara de gerenciamento de *threads* são elementos cruciais para o sucesso de um sistema de gestão de biblioteca que suporta operações concorrentes. Implementar um controle de acesso seguro, sincronização adequada e realizar testes abrangentes são passos fundamentais para assegurar a estabilidade e a integridade do sistema em ambientes de operações simultâneas. A garantia de transações atômicas e a gestão cuidadosa de exclusão mútua são aspectos-chave para evitar conflitos e manter a consistência dos dados em situações concorrentes.

4. RECOMENDAÇÕES E BOAS PRÁTICAS PARA O TRATAMENTO DE EXCEÇÕES E ATUALIZAÇÃO/CONSULTA DE DADOS

No desenvolvimento de programas, independentemente da sua complexidade, é comum depararmos com funções de entrada de dados. A fim de antecipar possíveis falhas humanas nesse processo, a prática de tratamento de exceções é essencial. Essa abordagem é implementada por meio da estrutura "Try-catch", que identifica e aborda problemas no início da execução, impedindo a interrupção do código.

Para garantir a eficácia da estrutura "Try-catch", é crucial que as classes de exceção estejam devidamente definidas. Elas devem ter nomes distintos para evitar conflitos no código, abranger todas as possíveis situações de erro e apresentar mensagens claras que explicitam o tipo de exceção. Essas medidas facilitam a depuração durante a execução do código em ambiente de produção.

No contexto de atualização ou consulta de dados, manter a consistência dos dados é fundamental em sistemas de biblioteca. Modificações diretas nos streams originais podem resultar em comportamentos inesperados, prejudicando a experiência do usuário com as operações fornecidas pela biblioteca.

A utilização de operações imutáveis não apenas aumenta a previsibilidade do sistema, mas também previne a ocorrência de efeitos colaterais indesejados. Estes últimos poderiam comprometer a robustez do código e, consequentemente, a experiência do usuário. Vale ressaltar que modificações no código tornam mais complexo o rastreamento e a depuração de problemas potenciais.