Assigned in: Lesson 6

Due before: Lesson 7

Upload your finished lab to the Learning Hub (Activities / Lab 5) before the start of lesson 7.

Create a **DiaryEntry** class with two private fields: a String `date` (formatted as YYYY-MM-DD) and a String `content`.

- Provide constructors, getters, and setters for these fields, as appropriate.

- Implement validation in the setters that checks if the date is properly formatted and if the content is non-null, non-blank, and does not contain the word "bcit".

- If validation fails, throw a custom exception: **DiaryEntryException**

Define a custom exception class, **DiaryEntryException**, which extends the Exception class.

- This exception must be thrown if the date format is incorrect, or if the content is null, blank, or contains the restricted word "bcit".

Create a **Diary** class, with the following data and properties.

Implement a method **addEntry()** that prompts the user for the date and content, creates a DiaryEntry object, and writes this information to **diary.txt** formatted as "date|content", each on a new line. Catch any DiaryEntryExceptions during entry creation and handle them appropriately, e.g., by informing the user of the input error.

Implement a method **viewAllEntries()** to display all diary entries. Use `FileReader` and `Scanner` to read from `diary.txt`, parsing each line to separate the date and content. Print each entry neatly formatted.

Implement a method **searchEntriesByDate()** that prompts the user for a date and searches `diary.txt` for matching entries, displaying any found content.

Create a **Main** class with a **main()** method that displays a text-based menu allowing users to choose from adding an entry, viewing all entries, searching by date, or exiting. Maintain a loop to keep the program running until the user decides to exit.

Notes:

- File I/O Operations: Use only `FileReader`, `FileWriter`, and `Scanner`.

- Resource Management: Manually manage resources, opening and closing them in `finally` blocks.

- Error Handling: Handle both the custom `DiaryEntryException` and standard IO exceptions like `FileNotFoundException` and `IOException`.

Sample Successful Run:

Develop a simple Personal Diary Management System in Java that allows users to add new diary entries, view all entries, and search for entries by date. This lab focuses on practicing file I/O operations using `Scanner`, `FileReader`, and `FileWriter` classes (no other IO classes), as well as basic exception handling and resource management without the use of try-with-resources syntax.

\*\*Part 1: Diary Entry Model\*\*

1. \*\*No need for a separate class model\*\* in this exercise. You will directly work with strings and file operations to manage diary entries.

\*\*Part 2: Implementing the Diary Management System\*\*

1. \*\*Add New Entry:\*\*

- Implement a method to add a new diary entry. The user should input the date in the format "YYYY-MM-DD" and the content of the diary entry.

- Use `FileWriter` to append this information to a file named `diary.txt`, with the date and content separated by a "|" symbol. Ensure each entry is on a new line.

2. \*\*View All Entries:\*\*

- Implement a method to display all diary entries.

- Use `FileReader` and `Scanner` to read from `diary.txt`, parsing each line to separate the date and content. Print each entry with the date and content on separate lines.

3. \*\*Search Entries by Date:\*\*

- Implement a method that prompts the user for a date and searches `diary.txt` for entries with that date, displaying the content of any entries found.

\*\*Part 3: Main Method and User Interface\*\*

- Implement a text-based menu in the main method allowing users to choose an action (add an entry, view all entries, search by date, or exit).

- Use a loop to keep the program running until the user chooses to exit.

#### Implementation Notes:

- \*\*File I/O Operations:\*\* Strictly use `FileReader` for reading files, `FileWriter` for writing to files, and `Scanner` for both reading user input and file content.

- \*\*Resource Management:\*\* Manually manage resources (opening and closing files) without using the try-with-resources statement. Ensure resources are closed properly in a `finally` block to avoid resource leaks.

- \*\*Error Handling:\*\* Implement basic error handling to manage exceptions that may occur during file operations, providing user-friendly error messages.

#### Deliverables:

- Source code for the `DiaryManager` class, implementing the specified functionalities.

- A text file named `diary.txt` used by the application to store diary entries.

- A brief report documenting your approach to implementing the lab, any challenges you faced, and how you resolved them.

#### Provisos:

- Do not use any classes for file I/O operations other than `Scanner`, `FileReader`, and `FileWriter`.

- Avoid using the try-with-resources syntax for resource management; instead, manually open and close your resources.

- Ensure your application can handle exceptions gracefully, such as file not found errors or I/O errors, without crashing.

- Use all best practices including no magic numbers.

Sample Run:

A screenshot of a computer

Description automatically generated