CASE STUDY

Notes Keeping app

OBJECT-ORIENTED PROGRAMMING WITH JAVA

GROUP MEMBERS

1. **Rukan Mubarak- 2023001166 (GROUP LEADER)**
2. **Abid Shaik - 2023000048**
3. **Aniket Anil - 2023002366**
4. **A. Reshwanth - 2023002682**
5. **S.V.N.R Sumedh - 2023001339**

**Notes Keeping App**

**Introduction**

* This Java program is a simple **Notes Keeping App** that allows users to:
* Add notes
* View all saved notes
* Edit existing notes
* Delete notes
* Exit the application
* The program uses Java classes, ArrayList, and Scanner to manage user input and note storage. Below is a detailed breakdown of how the program works.

1. **Understanding the Note Class**

* The Note class represents a single note. It contains:
* A **StringBuffer** content to store the note text.
* **A constructor** to initialise a note with given content.
* A **getContent()** method to return the note’s text.
* An **editNote()** method to modify the note.

**CODE**

import java.util.ArrayList;

import java.util.Scanner;

class Note

{

private StringBuffer content;

public Note(String content)

{

this.content = new StringBuffer(content);

}

public String getContent()

{

return content.toString();

}

public void editNote(String newContent)

{

content.setLength(0);

content.append(newContent);

}

}

class NotesManager

{

private ArrayList<Note> notes;

public NotesManager()

{

notes = new ArrayList<>();

}

public void addNote(String content)

{

notes.add(new Note(content));

System.out.println("Note added successfully.");

}

public void viewNotes()

{

if (notes.isEmpty())

{

System.out.println("No notes available.");

return;

}

System.out.println("Your Notes:");

for (int i=0;i<notes.size();i++)

{

System.out.println((i+1)+". "+notes.get(i).getContent());

}

}

public void editNote(int index, String newContent)

{

if (index>=1 && index<=notes.size())

{

notes.get(index-1).editNote(newContent);

System.out.println("Note updated successfully.");

} else

{

System.out.println("Invalid note index.");

}

}

public void deleteNote(int index)

{

if (index>=1 && index<=notes.size())

{

notes.remove(index-1);

System.out.println("Note deleted successfully.");

} else

{

System.out.println("Invalid note index.");

}

}

}

public class NotesApp {

public static void main(String[] args)

{

Scanner scanner = new Scanner(System.in);

NotesManager manager = new NotesManager();

while (true)

{

System.out.println("\nNotes Keeping App");

System.out.println("1. Add Note");

System.out.println("2. View Notes");

System.out.println("3. Edit Note");

System.out.println("4. Delete Note");

System.out.println("5. Exit");

System.out.print("Choose an option: ");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice)

{

case 1:

System.out.print("Enter note content: ");

String content = scanner.nextLine();

manager.addNote(content);

break;

case 2:

manager.viewNotes();

break;

case 3:

System.out.print("Enter note index to edit: ");

int editIndex = scanner.nextInt();

scanner.nextLine();

System.out.print("Enter new content: ");

String newContent = scanner.nextLine();

manager.editNote(editIndex, newContent);

break;

case 4:

System.out.print("Enter note index to delete: ");

int deleteIndex = scanner.nextInt();

manager.deleteNote(deleteIndex);

break;

case 5:

System.out.println("Exiting... Thank you!");

scanner.close();

return;

default:

System.out.println("Invalid option. Try again.");

}

}

}

}

**How It Works:**

* When a new note is created, its text is stored in a StringBuffer. The editNote() method replaces the old text with the new text, ensuring updates are efficient.

1. **Understanding the NotesManager Class**

* The NotesManager class manages multiple notes using an **ArrayList of Note objects**.

**Functions:**

* **addNote(String content)** → Adds a new note.
* **viewNotes()** → Displays all saved notes.
* **editNote(int index, String newContent)** → Updates a specific note.
* **deleteNote(int index)** → Removes a note from the list.

**How It Works:**

* Notes are stored in an ArrayList<Note>.
* The methods perform basic **CRUD operations** (Create, Read, Update, Delete).
* If a user tries to edit or delete a note that doesn’t exist, an error message is shown.

1. **Understanding the NotesApp Class (Main Program)**

* This is the **program's entry point**, where user interaction happens.

**How It Works:**

1. **Creates a Scanner for input**
2. **Creates a NotesManager object** to manage notes
3. **Runs an infinite loop (while (true))** to keep the program running
4. **Displays a menu** with five options
5. **Takes user input** and calls the appropriate method in NotesManager

**Breakdown of Menu Options:**

|  |  |
| --- | --- |
| **1** | Adds a new note |
| **2** | Displays all notes |
| **3** | Edits an existing note |
| **4** | Deletes a note |
| **5** | Exits the program |

1. **Error Handling & Edge Cases**

* The program includes basic error handling:
* **Empty Note List:** If there are no notes, viewNotes() shows:
* No notes available.
* **Invalid Index:** If a user enters an out-of-range index for editing or deleting, it shows:
* Invalid note index.

SAMPLE OUTPUTS

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Conclusion**

A Notes Keeping App is a Java project that displays the fundamental features of object-oriented programming concepts like classes, objects, encapsulation, and array lists. It ensures the simplicity of note management and, at the same time, consolidates the core Java programming principles, such as the control of user input and working with collections.

The program structure is menu-driven and has a convenient layout that can be easily navigated. It can work as a personal note organiser, and once improvements such as file storage, a search function or a graphical interface are added, it can turn into a full-featured note-taking app.