Notes App - Classes & Methods

Complete Reference Guide

Class: Note

Purpose: Represents a single note with its data and behaviors

`__init__(self, title, content, tags=None, author=None, status=None, priority=None)`

What: Constructor - creates a new Note object

Does: Sets up the note with data and creates timestamps

Example: `note = Note('My Title', 'My content', ['tag1'])`

`save(self, filename)`

What: Saves the note to a file

Does: Updates modified timestamp, converts to YAML format, writes to .note file

Example: `note.save('my-note')`

`load_note(cls, filepath) - @classmethod`

What: Reads a note file and creates a Note object from it

Why @classmethod: Creates a NEW Note (factory method)

Example: `note = Note.load_note('/path/to/file.note')`

`update(self, title=None, content=None, tags=None, author=None, status=None, priority=None)`

What: Updates note fields

Why None checks: So you can update SOME fields without changing ALL

Example: `note.update(title='New Title')` - only changes title

`to_dict(self)`

What: Converts Note to dictionary format

Used for: Compatibility with old code, testing, future features

Class: Notebook

Purpose: Manages the collection of ALL notes (like a filing cabinet)

`__init__(self, notes_folder)`

What: Constructor - sets up the Notebook

Does: Stores the folder path where notes are kept

`list_notes(self)`

What: Gets all note filenames

Returns: List of filenames like `['note1.note', 'note2.note']`

`get_note(self, filename)`

What: Retrieves a specific note as a Note object

Does: Calls Note.load_note() to load the file

`search_notes(self, query)`

What: Searches all notes for keywords

How it works: Splits query into words, finds notes containing ANY of the keywords

Example: `results = notebook.search_notes('python programming')`

`delete_note(self, filename)`

What: Deletes a note file

Parameters: Filename WITHOUT .note extension

`get_stats(self)`

What: Gets statistics about all notes

Returns: Dictionary with total_notes, total_tags, all_tags

Class: Application

Purpose: Handles the user interface and menu system

`__init__(self, notebook)`
What: Constructor - sets up the Application with a Notebook

`display_menu(self)`
What: Shows the main menu with options 1-9

`collect_note_input(self)`
What: Gets all input from user for creating a note

Returns: Tuple of (filename, title, content, tags, author, status, priority)

`run(self)`
What: Main loop - runs the entire application

Does: Shows menu, gets choice, calls appropriate handler, repeats

`handle_list(self)`
What: Menu option 1 - List all notes

`handle_create(self)`
What: Menu option 2 - Create a new note

Flow: Collect input → Create Note → Save → Success message

`handle_read(self)`
What: Menu option 3 - Read a note

Does: Shows list, gets user choice, loads note, displays content

`handle_edit(self)`
What: Menu option 4 - Edit a note

Smart: Only updates fields user changes

`handle_search(self)`
What: Menu option 5 - Search notes

Features: Keyword search, numbered results, Read/Edit/Exit options

`handle_delete(self)`
What: Menu option 6 - Delete a note

Safety: Requires 'y' confirmation before deleting

`handle_stats(self)`
What: Menu option 7 - Show statistics

`handle_help(self)`
What: Menu option 8 - Display help information

Key Concepts

Concept	Explanation	Example
self	Refers to 'this specific instance'	self.title = THIS note's title
@classmethod + cls	Works on the class itself, creates NEW objects	Note.load_note(filepath) Creates a new Note
Parameters =None	Optional parameters, can be skipped	tags=None means tags is optional
if value is not None:	Only update if provided, lets you update some fields without changing all	update(title='New') Only changes title

Class Relationships

```
Application → uses Notebook → uses Note
↓
uses read_note()
```

Example Flow: Create Note

1. User picks 'Create' from menu
2. Application.handle_create() runs
3. Calls collect_note_input() to get data
4. Creates new Note object with that data
5. Calls note.save() to write file
6. Returns to menu

Phase 1: Class-Based Refactoring Complete