To-Do List Manager: SOLID Principles Implementation Report

Introduction

Ugh, finally finished this To-Do List Manager project! Went with Python since it's my go-to. Tried to actually use those SOLID principles Prof. Chen keeps going on about, and not gonna lie, it was a pain at first but totally worth it in the end.

How SOLID Principles Were Applied

1. Single Responsibility Principle (SRP)

Broke everything down so each class just does its own thing:

- Task: Holds the actual task info (title, description, priority, etc.)
- TaskManager: Keeps track of all tasks - adding new ones, marking stuff complete, deleting
- TaskPrinter: Handles displaying tasks (saved me from that awful formatting mess I had before)
- PrioritySorter: Just sorts tasks by priority level
- FileStorageManager: Deals with saving/loading from files

Took longer to set up but way easier to fix when stuff broke later!

2. Open/Closed Principle (OCP)

This one was kinda confusing at first tbh. Basically made it so I can add new features without touching existing code:

- Need different sorting? Just make a new sorter class
- Want cloud storage instead? Swap in a different manager

Saved me from that nightmare refactoring situation I had with my last project!

3. Liskov Substitution Principle (LSP)

Had to google this one again lol:

- Made PersistentTaskManager extend TaskManager so they work the same way
- Set up FileStorageManager with that IStorageManager interface thing

Sounds fancy but basically just means I can swap parts without breaking everything else.

4. Interface Segregation Principle (ISP)

Split interfaces instead of making one giant one:

- ITaskManager: Just task stuff
- IStorageManager: Just storage stuff

Was gonna skip this but remembered how annoying it was dealing with bloated interfaces in that group project last semester.

5. Dependency Inversion Principle (DIP)

This one's actually pretty cool:

- PersistentTaskManager doesn't care HOW storage works, just that it works
- Made switching from file storage to database super simple

Why I Chose to Use Interfaces

Almost skipped interfaces cuz they seemed like extra work, but:

- ITaskManager keeps everything consistent
- IStorageManager lets me swap storage methods easily

Def worth the extra time even tho it felt unnecessary at first.

Challenges I Faced and How I Solved Them

1. Keeping Data Between App Runs

Kept losing all my tasks whenever I closed the app (so annoying). Fixed it with FileStorageManager to save everything as JSON.

2. Avoiding a Cluttered TaskManager

My TaskManager was getting ridiculous - doing like 10 different things. Split off the printing and storage which made everything way cleaner.

3. Making the App Extensible

Needed to add sorting without messing up everything else. Made PrioritySorter separate and it worked perfectly!

Final Thoughts

Actually pretty happy with how this turned out:

- Can add new features without everything breaking
- Code actually makes sense when I look at it a week later
- Could actually scale this up if needed

What's Next?

- Might add a GUI cuz command line is kinda lame
- Probably need database storage eventually

Check Out More of My Work

If you're bored and wanna see my other stuff:

- **Portfolio:** medzyamara.dev
- **GitHub:** github.com/Shaku-Med

Yo prof...

Would love any feedback on my portfolio/GitHub when you get a chance. Still figuring out if I'm doing this right! Thanks for checking this out, seriously.