I’ve tried best to address all your feedback and made updates Here's what I understood and changed based on your comments:

Boundary Objects

In your feedback you mentioned that boundary objects should only collect information from the actor (me, user) and pass it to the control objects, without performing business logic or calling control objects directly. I’ve implemented this across my boundary objects.

For example, with the !fetch_all_accounts command, the AccountBoundary collects the user input, then passes it to AccountControl, which handles the logic of fetching accounts. The boundary object no longer performs any business operations—it only manages user interaction as per your feedback.

Similarly, for commands like ‘!launch_browser’, the LaunchBrowserBoundary sends the command to LaunchBrowserControl to handle the browser actions, but the boundary itself doesn’t decide what to do next. Each boundary object is now directly tied to its respective use case and only handles communication between the user and the system.

Control Objects

In your feedback you pointed out that my control objects were previously doing things they shouldn't, like handling presentation logic or performing actions that should be managed by entity objects. I’ve now updated the control objects to act purely as “decision-makers”.

For instance, in the ‘!login bestbuy’ command, LoginControl gathers the input from LoginBoundary, checks for the correct account information via AccountControl, and then calls BrowserEntity to perform the actual login. The control object decides what should be done based on user input but delegates the task to the entity objects, which handle the execution.

The same applies for !navigate_to_website, where the NavigationControl determines which URL to navigate to and passes that information to BrowserEntity for execution. Control objects no longer touch the presentation layer or perform actions outside their scope.

Entity Objects

In your feedback you said that my entity objects were previously acting more like utilities or DAOs. I’ve refocused them so that they now handle all core business operations and rules.

For example, the BrowserEntity now contains all methods related to interacting with the browser, such as launching it, navigating to a URL, and logging in. It doesn’t make decisions about when to do these things—that’s the job of the control objects. It only knows how to perform these actions (like clicking, logging in, fetching prices).

For ‘!login bestbuy’, BrowserEntity handles the browser interactions, entering credentials, and logging into the site, but it doesn’t decide when to do this—that’s handled by the control object (entity_project_text).

DAO (Data Access Object)

You pointed out that I was mixing up my DAO and entity logic. Now, the DAO objects (like AccountDAO) are used strictly for database interactions, as you suggested. They perform CRUD operations (Create, Read, Update, Delete) and nothing else.

For instance, in !fetch_all_accounts, the AccountDAO is responsible for fetching the data from the database, but it does not handle any logic beyond that. It simply returns the account data to AccountControl, which then processes it and sends it back to the boundary to return to the user. I also created a "exportUtils.py" file which includes excel and html outputs. So, they are not in control or entity object anymore since they are presentation layer. That's what I understood from your comments.

You also mentioned every use case should have boundary and control objects. That's the reason I have created multiple files/objects for each use case. I don't remember a similar thing for entity objects. That's why I didn't create that much.

I rewatched the class multiple times where you explained the roles of Boundary, Control, and Entity objects (between minute 48 and 52 of the "Online Class-20240827_000119-Meeting Recording"), and I've implemented the changes based on that discussion. Here's what I understood from your explanation:

You emphasized that boundary objects are responsible for collecting user input and passing that information to the control object. The control object then executes the steps/methods of the use case and calls the entity object to perform the actual business logic. The entity object handles the business operations and interacts with the data layer (such as a repository or DAO) to access the database and return the results to the control object. Finally, the control object processes the result and may display the outcome to the user through the boundary object.

You highlighted the importance of this pipeline between boundary, control, and entity objects, and stressed that each object should have clear, distinct responsibilities without repeating the same operations across different layers. You also mentioned that a control object should typically have one main method that coordinates the steps, while the entity objects handle core business operations like get, set, add, update, and delete.

Example of How it Works in my project:

Here's a breakdown of how the system handles the !login bestbuy command:

1. User Input (Boundary Object):
   - The user(me) issues the command !login bestbuy to discord channel.
   - LoginBoundary collects the input and passes it to LoginControl.
2. Decision-Making (Control Object):
   - LoginControl fetches the account details for Best Buy using AccountControl, which gets the data from AccountDAO.
   - LoginControl then decides to call the login method in BrowserEntity.
3. Action (Entity Object):

    - BrowserEntity launches the browser, navigates to the Best Buy site, enters the username and password, and logs in.
    - The result is returned to LoginControl.

4. Result (Boundary Object):
    - LoginBoundary sends the result (success/failure) back to the user in Discord.

Same Process for Other Commands:

All the other commands; **!stop_bot, !project_help, !fetch_all_accounts, !add_account, !fetch_account_by_website, !delete_account, !launch_browser, !close_browser, !navigate_to_website, !login bestbuy, !get_price, !start_monitoring_price, !stop_monitoring_price**

!check_availability, !check_availability, !monitor_availability, !stop_monitoring_availability

follows the same flow:

- Boundary objects gather user input and pass it to the control objects.
- Control objects make the decisions and call the necessary methods in the entity objects.
- Entity objects handle the core actions, such as interacting with the browser or fetching data.
- DAO objects interact with the database for data storage or retrieval, if needed.

I’ve implemented these changes according to your guidance and have made every effort to fully address your concerns. However, I’m unclear about what aspects of the latest version still don't meet your requirements. If you could provide further clarification, I’d be happy to make the necessary updates.

NEXT PAGE: **Mapping of Use Cases to Objects**

Mapping of Use Cases to Objects

Command	Boundary Object	Control Object	Entity Object	DAO
!stop_bot	StopBoundary.py	StopControl.py	N/A	N/A
!project_help	HelpBoundary.py	HelpControl.py	N/A	N/A
!fetch_all_accounts	AccountBoundary.py	AccountControl.py	N/A	AccountDAO.py
!add_account <username> <password> <website>	AccountBoundary.py	AccountControl.py	N/A	AccountDAO.py
!fetch_account_by_website <website>	AccountBoundary.py	AccountControl.py	N/A	AccountDAO.py
!delete_account <id>	AccountBoundary.py	AccountControl.py	N/A	AccountDAO.py
!launch_browser	LaunchBrowserBoundary.py	LaunchBrowserControl.py	BrowserEntity.py	N/A
!close_browser	CloseBrowserBoundary.py	CloseBrowserControl.py	BrowserEntity.py	N/A
!navigate_to_website <url>	NavigationBoundary.py	NavigationControl.py	BrowserEntity.py	N/A
!login <website>	LoginBoundary.py	LoginControl.py	BrowserEntity.py, AccountControl.py	AccountDAO.py
!get_price	GetPriceBoundary.py	GetPriceControl.py	PriceEntity.py	N/A
!start_monitoring_price <url>	MonitorPriceBoundary.py	MonitorPriceControl.py	PriceEntity.py	N/A
!stop_monitoring_price	StopMonitoringPriceBoundary.py	MonitorPriceControl.py	PriceEntity.py	N/A
!check_availability <url> <date?>	CheckAvailabilityBoundary.py	CheckAvailabilityControl.py	AvailabilityEntity.py	N/A
!monitor_availability <url> <date?>	MonitorAvailabilityBoundary.py	MonitorAvailabilityControl.py	AvailabilityEntity.py	N/A
!stop_monitoring_availability	MonitorAvailabilityBoundary.py	MonitorAvailabilityControl.py	AvailabilityEntity.py	N/A