**Boundary Objects**

Each boundary object is specifically designed to parse user commands received via Discord, extracting necessary data before interacting with the appropriate control objects to fulfill the user's requests.

**Bot Boundary Objects:**

* **project\_help**: Interprets the user's request for help, parses the command, and communicates with the bot control to retrieve and display a list of available commands along with their descriptions.
* **stop\_bot**: Processes the user's command to terminate all bot operations, parses the message, and interacts with the bot control to initiate the shutdown.
* **receive\_email**: Handles the command to send an email with an attached file, parses the user's message to determine the file to be attached, and coordinates with the control object to manage the email sending process.

**Browser Boundary Objects:**

* **launch\_browser**: Responds to the user's command to open a web browser, parses the command, and directs the browser control to initiate a browser session.
* **close\_browser**: Processes the command to close the web browser, parses the message, and instructs the browser control to end the browser session.
* **login**: Manages the user's command to log into a website, parsing details like the website URL, username, and password before passing them to the browser control for the login operation.
* **navigate\_to\_website**: Captures and parses the user's command to navigate to a specific URL, then communicates with the browser control to perform the navigation.

**Availability Boundary Objects:**

* **check\_availability**: Receives and parses the user's message to extract necessary data such as the URL and date, then contacts the corresponding control object to check availability at the provided URL.
* **start\_monitoring\_availability**: Takes the user's input to begin monitoring availability at a specified URL with certain frequency parameters, parses the message, and forwards the data to the control layer to initiate monitoring.
* **stop\_monitoring\_availability**: Captures the command to cease monitoring availability, parses the user's instructions, and passes the command to the control object to stop the monitoring process.

**Price Boundary Objects:**

* **get\_price**: Receives the command to retrieve a price from a specified URL, parses the command to extract the URL, and contacts the price control to obtain and return the price.
* **start\_monitoring\_price**: Receives the command to start monitoring the price at a specified URL and interval, parses the message for necessary details, and forwards these to the price control to begin the monitoring process.
* **stop\_monitoring\_price**: Processes the command to stop price monitoring, parses the user's instructions, and notifies the price control to end the monitoring and summarize the findings.

**Control Objects**

Each control object acts as a decision-making hub that processes input from its corresponding boundary object, directs operations by interacting with entity objects or utilities (like logging or sending emails), and ultimately returns the outcome to the boundary object for user communication.

1. **BotControl**:
   * **project\_help**: Generates and returns a list of all available commands and their descriptions, assisting the user in navigating the bot's functionalities.
   * **stop\_bot**: Coordinates the shutdown process of the bot, ensuring all operations are cleanly terminated.
   * **receive\_email**: Manages the attachment and sending of an email with specified files, liaising with EmailEntity to perform the email operations.
2. **BrowserControl**:
   * **launch\_browser**: Contacts the BrowserEntity to open a web browser session using settings from the configuration.
   * **navigate\_to\_website**: Checks if the URL is valid or provided. If everything is valid, then contacts the BrowserEntity to perform the actual action.
   * **login**: Checks if the URL, username, and password are valid or provided. If everything is valid, then contacts the BrowserEntity to perform the actual login action.
   * **close\_browser**: Checks if there is an open session, then contacts the BrowserEntity to close the browser.
3. **AvailabilityControl**:
   * **check\_availability**: Checks if the URL is provided or not. If not, takes the default URL from the CSS selectors file. Contacts the AvailabilityEntity to verify availability at a specified URL and date, retrieves the availability status, calls the entity's data export method to save data.
   * **start\_monitoring\_availability**: Initiates a monitoring process at defined intervals by repeatedly calling the check\_availability method, handling the scheduling and continuation of this process, and calls the receive\_email method/control object after obtaining data.
   * **stop\_monitoring\_availability**: Ends the monitoring process, summarizes the collected data, and returns the final status to the boundary object for user notification.
4. **PriceControl**:
   * **get\_price**: Checks if the URL is provided or not. If not, takes the default URL from the CSS selectors file. Contacts the PriceEntity to fetch the price at a specified URL and calls the entity's data export method to save data.
   * **start\_monitoring\_price**: Initiates a monitoring process at defined intervals by repeatedly calling the get\_price method, handling the scheduling and continuation of this process, and calls the receive\_email method/control object after obtaining data.
   * **stop\_monitoring\_price**: Terminates the price monitoring process, summarizes the collected data, and communicates the results back to the boundary for user notification.

**Entity Objects:**

These entities act as the data manipulation layer of your architecture, directly interacting with the data sources and external systems to fetch, process, and store the required information. They provide a clean separation of concerns by encapsulating the logic needed to interact with data sources from the rest of the application, ensuring that the control objects can remain focused on application logic without needing to deal directly with data handling specifics.

1. **AvailabilityEntity**:
   * **Purpose**: Handles all data operations related to checking and monitoring availability. It directly interacts with external systems or databases to retrieve availability information.
   * **Key Methods**:
     + **check\_availability**: Connects to external services to check availability at the given URL on a specified date. It manages direct interactions with web APIs or databases to fetch availability data.
     + **export\_data**: Saves or logs availability data to local storage or a database. It might format the data for export to files such as Excel or HTML formats, which are then used for reporting or email notifications.
2. **BrowserEntity**:
   * **Purpose**: Manages all operations that require direct interaction with a web browser, such as opening, navigating, or closing a browser. It encapsulates all functionalities that involve web automation tools like Selenium.
   * **Key Methods**:
     + **launch\_browser**: Opens a web browser session with predefined configurations.
     + **navigate\_to\_website**: Navigates to a specified URL within an open browser session.
     + **close\_browser**: Closes the currently open web browser session to free up resources.
3. **DataExportEntity**:
   * **Purpose**: Responsible for exporting data into various formats for storage or transmission. This entity ensures data from operations like price checks or availability monitoring is logged appropriately.
   * **Key Methods**:
     + **export\_to\_excel**: Formats and writes data to an Excel file, organizing data into sheets and cells according to specified schemas.
     + **export\_to\_html**: Converts data into HTML format for easy web publication or email attachments.
4. **EmailEntity**:
   * **Purpose**: Handles the configuration and process of sending emails. This entity works with email servers to facilitate the sending of notifications, alerts, or reports generated by the system.
   * **Key Methods**:
     + **send\_email\_with\_attachments**: Prepares and sends an email with specified attachments. It manages attachments, formats the email content, and interacts with email servers to deliver the message.
5. **PriceEntity**:
   * **Purpose**: Specializes in fetching and monitoring price data from various online sources. It uses web scraping techniques to extract pricing information from web pages.
   * **Key Methods**:
     + **get\_price**: Retrieves the current price of a product from a specified URL. It scrapes the web page to find pricing information and returns it to the control layer.
     + **export\_data**: Similar to the AvailabilityEntity, it exports price data to various file formats for reporting or further analysis.

**Associations Among Objects:**

* **Boundary to Control Associations**:
  + AvailabilityBoundary communicates with AvailabilityControl.
  + BotBoundary communicates with BotControl.
  + BrowserBoundary communicates with BrowserControl.
  + PriceBoundary communicates with PriceControl.
* **Control to Entity Associations**:
  + AvailabilityControl interacts with AvailabilityEntity, DataExportEntity, and EmailEntity.
  + BotControl interacts with EmailEntity.
  + BrowserControl interacts with BrowserEntity.
  + PriceControl interacts with PriceEntity and DataExportEntity.

**Aggregates Among Objects:**

* **Availability Aggregate**:
  + Root: AvailabilityEntity
  + Includes: AvailabilityControl (manages AvailabilityEntity and potentially accesses DataExportEntity and EmailEntity for output operations).
* **Price Aggregate**:
  + Root: PriceEntity
  + Includes: PriceControl (manages PriceEntity and handles data through DataExportEntity).
* **Email Aggregate**:
  + Root: EmailEntity
  + Includes: Both BotControl and AvailabilityControl may use this for sending emails, positioning it as a shared resource.

**Attributes for Each Object:**

* **Boundary Objects Attributes**:
  + BotBoundary: commands !stop\_bot, !project\_help, !receive\_email
  + BrowserBoundary: commands !navigate\_to\_website, !login, !close\_browser,
  + AvailabilityBoundary: Commands: !check\_availability, !start\_monitoring\_availability, !stop\_monitoring\_availability
  + PriceBoundary: commands !get\_price, !start\_monitoring\_price, !stop\_monitoring\_price
* **Control Objects Attributes**:
  + AvailabilityControl: monitoring\_active (boolean), scheduled\_tasks (list of tasks).
  + BotControl: active\_sessions (number of active bot sessions).
  + BrowserControl: browser\_instance (current instance of the browser).
  + PriceControl: price\_history (historical prices), monitoring\_active (boolean).
* **Entity Objects Attributes**:
  + AvailabilityEntity: availability\_data, last\_checked.
  + BrowserEntity: cookies, session\_data.
  + DataExportEntity: file\_paths (locations of saved data).
  + EmailEntity: email\_queue (emails waiting to be sent).
  + PriceEntity: price\_data, last\_updated.

A diagram of a computer

Description automatically generated

@startuml

' Define Styles

skinparam classAttributeIconSize 0

skinparam class {

BackgroundColor<<Boundary>> Wheat

BackgroundColor<<Control>> LightBlue

BackgroundColor<<Entity>> LightGreen

}

' Boundary Classes

class AvailabilityBoundary <<Boundary>> {

+commands: List<String>

+check\_availability()

+start\_monitoring\_availability()

+stop\_monitoring\_availability()

}

class BotBoundary <<Boundary>> {

+commands: List<String>

+project\_help()

+stop\_bot()

+receive\_email()

}

class BrowserBoundary <<Boundary>> {

+browser\_status: Boolean

+current\_url: String

+launch\_browser()

+close\_browser()

+login()

+navigate\_to\_website()

}

class PriceBoundary <<Boundary>> {

+current\_price: Double

+monitoring\_status: Boolean

+get\_price()

+start\_monitoring\_price()

+stop\_monitoring\_price()

}

' Control Classes

class AvailabilityControl <<Control>> {

+monitoring\_active: Boolean

+scheduled\_tasks: List<Task>

+receive\_command()

+check\_availability()

+start\_monitoring\_availability()

+stop\_monitoring\_availability()

}

class BotControl <<Control>> {

+active\_sessions: Integer

+receive\_command()

+project\_help()

+stop\_bot()

+receive\_email()

}

class BrowserControl <<Control>> {

+browser\_instance: Object

+receive\_command()

+launch\_browser()

+navigate\_to\_website()

+login()

+close\_browser()

}

class PriceControl <<Control>> {

+price\_history: List<Double>

+monitoring\_active: Boolean

+receive\_command()

+get\_price()

+start\_monitoring\_price()

+stop\_monitoring\_price()

}

' Entity Classes

class AvailabilityEntity <<Entity>> {

+availability\_data: List<Data>

+last\_checked: DateTime

+check\_availability()

+export\_data()

}

class BrowserEntity <<Entity>> {

+cookies: List<Cookie>

+session\_data: Map<String, String>

+launch\_browser()

+navigate\_to\_website()

+close\_browser()

+login()

}

class DataExportEntity <<Entity>> {

+file\_paths: List<String>

+export\_to\_excel()

+export\_to\_html()

}

class EmailEntity <<Entity>> {

+email\_queue: List<Email>

+send\_email\_with\_attachments()

}

class PriceEntity <<Entity>> {

+price\_data: Map<String, Double>

+last\_updated: DateTime

+get\_price()

+export\_data()

}

' Relationships

AvailabilityBoundary --> AvailabilityControl

BotBoundary --> BotControl

BrowserBoundary --> BrowserControl

PriceBoundary --> PriceControl

AvailabilityControl --> AvailabilityEntity

AvailabilityControl ---> DataExportEntity

AvailabilityControl ---> EmailEntity

PriceControl --> PriceEntity

PriceControl ---> DataExportEntity

PriceControl ---> EmailEntity

BrowserControl --> BrowserEntity

@enduml