**Assignment 4**

import requests

from bs4 import BeautifulSoup

class BrowserInterface:

    """

    Handles interactions between the bot and the browser for scraping web data.

    """

    def \_\_init\_\_(self, browser\_type, url):

        # Initialize the browser interface with browser type and URL

        self.\_\_browser\_type = browser\_type

        self.\_\_url = url

    def launch\_browser(self):

        # Launch the browser with the specified URL

        if self.\_\_url:

            print(f"Launching {self.\_\_browser\_type} browser with URL: {self.\_\_url}")

        else:

            raise ValueError("URL must not be null.")

    def close\_browser(self):

        # Close the browser

        print(f"Closing {self.\_\_browser\_type} browser.")

        self.\_\_browser\_type = None

        self.\_\_url = None

    def login(self, account):

        # Use the Account class to log in to a website

        if account.get\_username() and account.get\_password():

            print(f"Logging in with username: {account.get\_username()}")

            # Placeholder for actual login logic using account credentials

            # This would involve interacting with the web page elements to enter the username and password

        else:

            raise ValueError("Account credentials must not be null.")

    def display\_data\_in\_html(self, data):

        # Create an HTML page to display the data read from Excel

        html\_content = "<html><head><title>Product Data</title></head><body>"

        html\_content += "<h1>Product Data</h1><table border='1'>"

        html\_content += "<tr><th>Timestamp</th><th>URL</th><th>Price</th><th>Product</th></tr>"

        for row in data:

            html\_content += f"<tr><td>{row['Timestamp']}</td><td>{row['URL']}</td><td>{row['Price']}</td><td>{row['Product']}</td></tr>"

        html\_content += "</table></body></html>"

        # Save the HTML content to a file

        with open("product\_data.html", "w") as file:

            file.write(html\_content)

        print("Data displayed in HTML page (product\_data.html).")

class DateInfoInterface:

    """

    Manages the input and output for date availability requests.

    """

    def \_\_init\_\_(self):

        # Initialize with empty date info

        self.\_\_date\_info = ""

    def fetch\_date\_info(self, date):

        # Fetch date information (Placeholder for actual date info retrieval logic)

        if date:

            self.\_\_date\_info = f"Availability checked for {date}"

            print(self.\_\_date\_info)

        else:

            raise ValueError("Date must not be null.")

    def get\_date\_info(self):

        # Return the fetched date information

        if self.\_\_date\_info:

            return self.\_\_date\_info

        else:

            raise ValueError("Date information has not been fetched yet.")

import discord

class DiscordInterface:

    """

    Manages the interactions between the bot and the user on Discord.

    """

    def \_\_init\_\_(self, interface\_name, discord\_bot):

        # Initialize with the interface name and Discord bot instance

        self.\_\_interface\_name = interface\_name

        self.\_\_discord\_bot = discord\_bot

    def connect(self):

        # Connect the bot to Discord

        if self.\_\_discord\_bot:

            print(f"Connecting {self.\_\_interface\_name} to Discord...")

            self.\_\_discord\_bot.run()

        else:

            raise ValueError("Discord bot instance must not be null.")

    def disconnect(self):

        # Disconnect the bot from Discord

        if self.\_\_discord\_bot:

            self.\_\_discord\_bot.close()

            print(f"Disconnecting {self.\_\_interface\_name} from Discord...")

        else:

            raise ValueError("Discord bot instance must not be null.")

import pandas as pd

from datetime import datetime

class ExcelInterface:

    """

    Handles data extraction to and from Excel files.

    """

    def \_\_init\_\_(self, file\_path):

        # Initialize with the file path where the Excel file will be saved

        self.\_\_file\_path = file\_path

    def save\_data\_to\_excel(self, data):

        # Save the data to an Excel file with additional details

        if data:

            df = pd.DataFrame(data)

            df['Timestamp'] = datetime.now()  # Add a timestamp column

            df.to\_excel(self.\_\_file\_path, index=False)

            print(f"Data saved to {self.\_\_file\_path}")

        else:

            raise ValueError("Data must not be null.")

    def load\_data\_from\_excel(self):

        # Load data from an Excel file

        try:

            data = pd.read\_excel(self.\_\_file\_path).to\_dict(orient="records")

            print(f"Data loaded from {self.\_\_file\_path}")

            return data

        except Exception as e:

            print(f"Failed to load data from Excel: {e}")

            return None

class ProductInfoInterface:

    """

    Manages the input and output for product information requests.

    """

    def \_\_init\_\_(self):

        # Initialize with empty product details

        self.\_\_product\_details = ""

    def fetch\_product\_info(self, product\_url):

        # Fetch product details from the URL (Placeholder for actual scraping logic)

        if product\_url:

            self.\_\_product\_details = f"Details fetched from {product\_url}"

            print(self.\_\_product\_details)

        else:

            raise ValueError("Product URL must not be null.")

    def get\_product\_details(self):

        # Return the fetched product details

        if self.\_\_product\_details:

            return self.\_\_product\_details

        else:

            raise ValueError("Product details have not been fetched yet.")

Oguz Kaan Yildirim