**Assignment 3**

```
class Account:
    """
    Represents a user account with a username and password.
    """

    def __init__(self, username, password):
        # Initialize account with username and password
        self.username = username
        self.__password = password

    def set_username(self, username):
        # Set a new username
        self.username = username

    def set_password(self, password):
        # Set a new password
        self.__password = password

    def get_username(self):
        # Return the username
        return self.username

    def info_account(self):
        # Print detailed information about the account
        print(f"Username: {self.username}")
        # Password is protected and should not be printed out directly
        print("Password: [PROTECTED]")

    def validate_account(self, username, password):
        # Validate the account credentials
```

```
if self.username == username and self.__password == password:
    print("Login successful.")
    return True
else:
    print("Login failed. Invalid credentials.")
    return False
```

```
import os

class Command:
    """
    Represents a command given to the bot.

    The Command class processes user inputs and executes the corresponding actions based on the command type.
    Commands are stored in a specific folder, and the class will check the input against the available commands,
    executing the appropriate logic when a match is found.

    Examples of commands:
    - "get price, 'url.com'": Fetches the price from the provided URL.
    - "track availability, 'date'": Continuously checks the availability of a specific date.
    - "send notification, 'message'": Sends a custom notification to the user.
    """

    def __init__(self, description, command_input, user_id, timestamp):
        # Initialize command with description, input, user ID, and timestamp
        self.description = description
        self.input = command_input  # Command input provided by the user
        self.user_id = user_id  # Who issued the command
        self.timestamp = timestamp  # When the command was issued
        self.status = "pending"  # Initial status of the command
        self.folder_path = '/path/to/commands/'  # Placeholder path where commands are stored

    def get_description(self):
        # Return the command's description
        return self.description

    def get_input(self):
```

```
    # Return the input for the command
    return self.input

  def execute(self):
    """
    Execute the command by matching the input with predefined commands and performing the corresponding actions.

    The command input is checked against a list of available commands stored in a specific folder.
    If a match is found, the appropriate logic is executed.

    For example:
    - "get price, 'url.com'" -> Fetches the price from the provided URL.
    - "track availability, 'date'" -> Checks if a specific date is available and notifies the user.
    - "send notification, 'message'" -> Sends a notification to the user.
    """
    print(f"Executing command: {self.description}")
    print(f"This is the command I got: '{self.input}' from user {self.user_id} at {self.timestamp}.")
    print(f"I'll do this: Searching the folder {self.folder_path} for matching commands.")

    # Simulate checking the input against available commands
    if os.path.exists(self.folder_path):
      print("Folder found. Searching for matching commands...")
      command_action = self.match_command_with_input()
      if command_action:
        print(f"Command found: {command_action}")
        self.perform_action(command_action)
      else:
        print("No matching command found. Please check your input.")
    else:
      print("Folder not found. Cannot execute the command.")
```

```
    self.status = "completed"
    print(f"Command execution completed. Status: {self.status}")

def match_command_with_input(self):
    """
    Match the user input with predefined commands and return the corresponding action.

    This method simulates checking the user input against a set of available commands.
    If a match is found, the corresponding action is returned.

    For example:
    - Input: "get price, 'url.com'" -> Action: "Fetch price from URL".
    - Input: "track availability, 'date'" -> Action: "Check date availability".

    Returns:
        str: The action corresponding to the matched command.
    """
    # Placeholder for actual command matching logic
    if "get price" in self.input:
        return "Fetch price from URL"
    elif "track availability" in self.input:
        return "Check date availability"
    elif "send notification" in self.input:
        return "Send custom notification"
    else:
        return None

def perform_action(self, action):
    """
    Perform the action associated with the matched command.
```

This method executes the logic corresponding to the matched command.

Depending on the action, it may involve fetching data from a URL, checking availability, or sending notifications.

```
    Args:
        action (str): The action to be performed based on the matched command.
    """
    print(f"Performing action: {action}")
    if action == "Fetch price from URL":
        # Placeholder logic to fetch price from a given URL
        url = self.extract_url_from_input()
        if url:
            print(f"Fetching price from: {url}")
            # Simulate fetching price
            print("Price fetched: $123.45")
        else:
            print("No URL found in the input.")
    elif action == "Check date availability":
        # Placeholder logic to check date availability
        date = self.extract_date_from_input()
        if date:
            print(f"Checking availability for: {date}")
            # Simulate availability check
            print("Date is available.")
        else:
            print("No date found in the input.")
    elif action == "Send custom notification":
        # Placeholder logic to send notification
        message = self.extract_message_from_input()
        if message:
```

```
            print(f"Sending notification: {message}")
            # Simulate sending notification
            print("Notification sent.")
        else:
            print("No message found in the input.")

    def extract_url_from_input(self):
        """
        Extract the URL from the user input.

        This is a placeholder method to simulate extracting a URL from the input string.

        Returns:
            str: The extracted URL.
        """
        # Example extraction logic
        if "url.com" in self.input:
            return "http://url.com"
        return None

    def extract_date_from_input(self):
        """
        Extract the date from the user input.

        This is a placeholder method to simulate extracting a date from the input string.

        Returns:
            str: The extracted date.
        """
        # Example extraction logic
        if "2024-08-15" in self.input:
```

```
        return "2024-08-15"
    return None

def extract_message_from_input(self):
    """
    Extract the message from the user input.

    This is a placeholder method to simulate extracting a message from the input string.

    Returns:
        str: The extracted message.
    """
    # Example extraction logic
    if "notify" in self.input:
        return "This is a custom notification."
    return None

def is_valid(self):
    # Check if the command is valid (example business rule)
    return True if self.input else False

def info_command(self):
    # Print detailed information about the command
    print(f"Command Description: {self.description}")
    print(f"Input: {self.input}")
    print(f"User ID: {self.user_id}")
    print(f"Timestamp: {self.timestamp}")
    print(f"Status: {self.status}")
```

```
class Date:
    """
    Represents the date being checked for availability.
    """

    def __init__(self, date, available=True):
        # Initialize with the date and availability status
        self.date = date
        self.available = available

    def get_date(self):
        # Return the date
        return self.date

    def fetch_date_details(self):
        # Simulate fetching date details (Placeholder logic)
        if self.available:
            details = {
                'date': self.date,
                'status': 'Available'
            }
            self.print_date_details(details)
        else:
            self.no_date_found()

    def print_date_details(self, details):
        # Print out the date details
        print(f"Date: {details.get('date')}")
        print(f"Status: {details.get('status')}")

    def no_date_found(self):
```

```
    # Handle the case where no date is available
    print("The date you requested is not available.")

def is_available(self):
    # Check if the date is available
    return self.available

def info_date(self):
    # Print the date information
    if self.available:
        print(f"Date: {self.date} is available.")
    else:
        print(f"Date: {self.date} is not available.")
```

```
class Notification:
    """
    Represents a notification sent to the user.
    """

    def __init__(self, notif_type, content, timestamp):
        # Initialize notification with type, content, and timestamp
        self.type = notif_type
        self.content = content
        self.timestamp = timestamp

    def get_type(self):
        # Return the type of the notification
        return self.type

    def get_content(self):
        # Return the notification content
        return self.content

    def get_timestamp(self):
        # Return when the notification was sent
        return self.timestamp

    def info_notification(self):
        # Print detailed information about the notification
        print(f"Notification Type: {self.type}")
        print(f"Content: {self.content}")
        print(f"Timestamp: {self.timestamp}")
```

```
class Product:
    """
    Represents a product to track.
    """

    def __init__(self, name, url, options=None):
        # Initialize the product with a name, URL, and options (like size, color)
        self.name = name
        self.url = url
        self.options = options if options is not None else {}

    def set_url(self, url):
        # Update the product's URL
        self.url = url

    def get_name(self):
        # Return the product's name
        return self.name

    def get_options(self):
        # Return the options (like size, color)
        return self.options

    def fetch_product_details(self):
        # Fetch product details from the web (Placeholder logic)
        details = {
            'price': 'To be fetched',  # Placeholder
            'availability': 'To be checked'
        }
        if details:
            self.print_product_details(details)
```

```
    else:
        self.no_details_found()

def print_product_details(self, details):
    # Print out the product details
    print(f"Product: {self.name}")
    print(f"Price: {details.get('price')}")
    print(f"Availability: {details.get('availability')}")
    if self.options:
        print(f"Options: {self.options}")

def no_details_found(self):
    # Handle the case where no details are found
    print("No product details found for the given URL.")
```

```
class User:
    """
    Represents a user of the system.
    """

    def __init__(self, user_id, email):
        # Initialize user with ID and email
        self.__user_id = user_id
        self.email = email

    def get_user_id(self):
        # Return the user's ID
        return self.__user_id

    def get_email(self):
        # Return the user's email
        return self.email

    def info_user(self):
        # Print detailed information about the user
        print(f"User ID: {self.__user_id}")
        print(f"Email: {self.email}")
```

Oguz Kaan Yildirim