OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Vladyslav Keidaliuk

IMDB SOFTWARE OF HOLLYWOOD ACTORS AND ACTRESSES. TASK 2

Faculty:	Informatik und Mathematik
Degree:	Computer Science
Submission deadline:	10.12.2024
Supervisor/examiner:	Istvan Lengyel

Table of Contents

Introduction 3
Used data and tools 3
Overview of Current Implementation 4
Web Scraping and Data Saving: 4
GUI Implementation: 4
Pseudocode 5
Pseudocode for Web Scraping Functionality 5
Pseudocode for Tkinter-based GUI Functionality 8
Solution Examples 10
Demonstration of the GUI 13
Next steps 15
Conclusion 16

Introduction

In this report, a partial result of the project will be presented, focusing on the functionality for scraping and displaying actor-related information. The implemented features include extracting details such as actor names, photos, biographies, and a comprehensive list of movies with key attributes such as titles, release years, genres, ratings, and poster URLs. This information is stored locally in a structured format, enabling a user-friendly graphical interface to present the data.

The interface allows users to browse a list of actors, view detailed biographies, and explore movie-related information in an organized manner. Additionally, the project employs a well-structured Page Object Model to ensure scalability, maintainability, and clarity in the codebase.

The following sections describe the completed functionalities and provide the corresponding pseudocode for the implemented features.

Used data and tools

Data was collected from the IMDb website and stored in a JSON file format. The data includes detailed information about actors, such as their names, photos (saved locally), biographies, and a list of movies with attributes like title, release year, genres, ratings, and poster URLs.

The project was developed using **JetBrains PyCharm** as the primary coding environment. Web scraping was performed with the **Selenium** library to interact with the IMDb website dynamically. For data management, Python's **JSON** module was used to process and store the scraped information in structured formats such as dictionaries and lists.

To ensure a clear and maintainable structure, the project follows the **Page Object Model** design pattern. This approach organizes the scraping logic into distinct classes and methods, facilitating scalability and readability in the codebase.

Overview of Current Implementation

At this stage, the project includes both a web scraping functionality to gather data about actors and their movies and the graphical user interface to display the collected information. The implementation is split into two parts: the data scraping process and the GUI creation.

Web Scraping and Data Saving:

The primary functionalities achieved so far in relation to scraping and data storage are as follows:

Actor Information Scraping:

- The actor's name is extracted and stored
- The actor's photograph is downloaded and saved locally
- The biography of the actor is collected

Movie Information Scraping:

- Title of the movie
- Year of release
- A link to the movie poster (downloading all posters was deemed inefficient and is thus not implemented)
- Movie rating, if available
- Movie genres

Structured Design:

The project is implemented using the **Page Object Model** design pattern to ensure maintainability and clarity. Each page's interactions and locators are encapsulated in their respective classes, allowing for a modular and organized approach to the web scraping process.

GUI Implementation:

The GUI part of the project is implemented using **Tkinter** and is designed to display the actors' information, including their names, photos, biographical details, and movies they have appeared in.

The following features are part of the current GUI implementation:

1. Actor List View:

   - The main window presents a list of actors, each represented by a button with the actor's name and a thumbnail photo. Clicking on a button opens a detailed view of the actor.

2. Actor Information Window:

   When an actor is selected, a new window opens showing:

   - Actor's full name and age (calculated from the date of birth).
   - Actor's biography in a scrollable text field.
   - A button to view the actor's movie list.

3. Movie List View:

   A separate window displays the list of movies the actor has been involved in, including:

   - Movie title, release year, rating, and genres.
   - A thumbnail image of the movie poster (if available).

4. Scroll Functionality:

The application supports scrolling for the actor list and the movie list to handle potentially large data sets efficiently.

Pseudocode

Pseudocode for Web Scraping Functionality

This subsection describes the pseudocode for the web scraping functionality that extracts data about Hollywood actors and their movies. The process involves scraping actor details, movie information, and storing it in a structured JSON format, which will be later used in the Tkinter-based GUI application.

Main Scraping Program Flow:

```
START
  SET home_page_url = "https://www.imdb.com/list/ls053501318/"
  INIT driver with browser and options
  NAVIGATE to home_page_url

  CREATE instance of ActorsListPage(driver)
```

```
  CALL actors_page.click_accept_cookies()
  SET actors = actors_page.get_actors()

  IF "actors_data.json" exists:
      LOAD actors_data from JSON file
      SET processed_actors = {actor['name'] for actor in actors_data}
  ELSE:
      INIT actors_data as an empty list
      SET processed_actors = empty set

  FOR actor in actors:
      IF actor['name'] in processed_actors:
          SKIP to next actor

      CALL actors_page.click_by_title(actor['name'])
      CREATE instance of ActorPage(driver)

      SET actor_name = act_page.get_actor_name()
      SET date_of_birth = act_page.get_date_of_birth()
      SET photo_url = act_page.get_photo_url()
      CALL download_img(photo_url, actor_name)

      CALL act_page.open_bio_page()
      CREATE instance of BiographyPage(driver)
      SET biography = bio_page.get_biography()
      CALL driver.back()

      CALL act_page.expand_list_of_movies()

      INIT movies as empty list
      SET number_of_movies = act_page.get_count_of_movies()

      FOR index in range(1, number_of_movies + 1):
          SET movie_data = act_page.get_movie_data_by_index(index)
          APPEND movie_data to movies

      CREATE actor_info dictionary:
          - "name": actor_name
          - "photo_path": f"{actor_name.lower().replace(' ', '_')}.jpg"
          - "date_of_birth": date_of_birth
          - "biography": biography
          - "movies": movies
      APPEND actor_info to actors_data
      SAVE actors_data to "actors_data.json"
      CALL driver.back()
  CALL driver.quit()
END
```

ActorsListPage:

```
CLASS ActorsListPage:
  FUNCTION get_actors():
      CALL self.find_elements(locator)
      RETURN list of actor names and links

  FUNCTION click_accept_cookies():
      CALL self.click_on(locator)

  FUNCTION click_by_title(actor_name):
      CALL self.click_on(title_with_text(actor_name))
```

ActorPage:

```
CLASS ActorPage:
  FUNCTION get_actor_name():
      CALL self.find_element(locator)
      RETURN text of actor name element

  FUNCTION get_date_of_birth():
      CALL self.find_element(locator)
      RETURN text of date of birth element

  FUNCTION get_photo_url():
      CALL self.find_element(locator)
      RETURN "src" attribute of photo element

  FUNCTION expand_list_of_movies():
      CALL self.click_on(locator)

  FUNCTION get_movie_data_by_index(index):
      CALL self.get_movie_image_by_index(index)
      CALL self.click_on_movie_by_index(index)

      SCRAPE:
        - title
        - year
        - genres
        - rating

      CALL self.click_on(CLOSE_MOVIE_BLOCK_BTN)
      RETURN movie_data as dictionary
```

BiographyPage:

```
CLASS BiographyPage:
  FUNCTION get_biography():
      CALL self.find_element(locator)
      RETURN text of biography element
```

BasePage:

```
CLASS BasePage:
  FUNCTION find_element(locator):
      CALL WebDriverWait(driver,
timeout).until(EC.visibility_of_element_located(locator))
      RETURN element

  FUNCTION find_elements(locator):
      CALL WebDriverWait(driver,
timeout).until(EC.presence_of_all_elements_located(locator))
      RETURN list of elements

  FUNCTION click_on(locator):
      CALL self.scroll_to_element(locator)
      CALL element.click()

  FUNCTION try_click(locator, retries, delay):
      ATTEMPT retries times:
          TRY:
              SCROLL to element
              CLICK element
              RETURN success
          CATCH exception:
              WAIT delay seconds
      RAISE exception if all attempts fail
```

Pseudocode for Tkinter-based GUI Functionality

This subsection describes the pseudocode for the Tkinter-based GUI functionality. It explains how the actor data (scraped in the previous subsection) is presented in the GUI, how actor details and movie information are displayed in different windows, and how the user can interact with the interface.

```
START
  LOAD actor data from "actors_data.json"

  CREATE main window
  SET window title to " Vladyslav Keidaliuk - Project 2. Hollywood Actors and
Actresses"

  CREATE scrollable frame for actor buttons
  CALL show_actors()

  FUNCTION show_actors():
      SET columns = 3
      SET row = 0
      SET col = 0

      FOR actor in actor_data:
          CREATE button for actor with name and photo
          SET button action to show_actor_info(actor)
          ADD button to grid in main window

          IF col >= columns:
              RESET col to 0
              INCREMENT row by 1
          ELSE:
              INCREMENT col by 1

  FUNCTION show_actor_info(actor_name):
      SET actor = find_actor_in_data(actor_name)

      IF actor is not found:
          RETURN

      CREATE new window for actor details
      DISPLAY actor name, date of birth, and biography in text box
      DISPLAY button to show actor's movies

      FUNCTION show_actor_movies(actor_name):
          SET actor = find_actor_in_data(actor_name)

          CREATE new window for actor's movies
          DISPLAY movie details (title, year, rating, genres)
          IF movie has an image:
              DOWNLOAD and show movie poster
          END loop

  FUNCTION find_actor_in_data(actor_name):
      FOR actor in actor_data:
```

```
          IF actor name matches actor_name:
              RETURN actor

  RUN GUI main loop to keep window open
END
```

Solution Examples

In this chapter, we present an example of code that demonstrates how actor and movie information is scraped from a web page using Selenium. This example shows the **ActorPage** class, which includes the various methods and elements used to extract data such as actor details, their movie list, movie ratings, and images. The methods in this class are designed to interact with specific parts of the web page and return the required information.

Python code:

```
class ActorPage(BasePage):
    PHOTO = (By.XPATH,
             "//img[contains(@class, 'poster') or contains(@alt, 'Primary')]")
    ABOUT = (By.XPATH,
             "//div[@id='name-bio-text']//div[@class='inline']")
    MOVIES = (By.XPATH,
              "//div[contains(@class,'filmo-row')]")

    BIO_LINK_ELEMENT = (By.XPATH,
                        "//div[starts-with(@class,'ipc-html-content ipc-html-content--
baseAlt')]")
    DATE_OF_BIRTH = (By.XPATH,
                     "(//div[@data-testid='birth-and-death-birthdate']/span)[4]")
    COUNT_OF_ACTOR_MOVIES = (By.XPATH,
                             "//*[@id='actor-previous-
projects']/div[1]/label/span[1]/ul/li[2]")
    COUNT_OF_ACTRESS_MOVIES = (By.XPATH,
                               "//*[@id='actress-previous-
projects']/div[1]/label/span[1]/ul/li[2]")
    SEE_ALL_BTN_ACTOR = (By.XPATH,
                         "//button[@data-testid='nm-flmg-paginated-all-actor']")
    SEE_ALL_BTN_ACTRESS = (By.XPATH,
                           "//button[@data-testid='nm-flmg-paginated-all-actress']")

    MOVIE_TITLE = (By.XPATH,
                   "//h3[@class='ipc-title__text prompt-title-text']")
    CLOSE_MOVIE_BLOCK_BTN = (By.XPATH,
                             "//button[@title='Close Prompt']")
    RELEASE_YEAR = (By.XPATH,
```

```
                    "//ul[@data-testid='btp_ml']/li[1]")
    LIST_OF_GENRES = (By.XPATH,
                      "//ul[@data-testid='btp_gl']/li")
    RATING_OF_MOVIE = (By.XPATH,
                       "//div[@data-testid='btp_rt']//span[@class='ipc-rating-star--
rating']")
    AWARDS_LINK_ELEMENT = (By.XPATH,
                           "//a[text()='Awards' and starts-with(@class,'ipc-link')]")

    def actor_movie_image_by_index(self, index):
        return (By.XPATH,
                f"(//div[@id='accordion-item-actor-previous-
projects']//img[@class='ipc-image'])[{index}]")
    def actress_movie_image_by_index(self, index):
        return (By.XPATH,
                f"(//div[@id='accordion-item-actress-previous-
projects']//img[@class='ipc-image'])[{index}]")

    def get_movie_image_by_index(self, index):
        try:
            element = self.find_element(self.actor_movie_image_by_index(index))
            return element.get_attribute('src')
        except:
            try:
                element = self.find_element(self.actress_movie_image_by_index(index))
                return element.get_attribute('src')
            except:
                return "No image"

    def actor_movie_block_by_index(self, index):
        return (By.XPATH,
                (f"//div[@id='accordion-item-actor-previous-
projects']/div/ul/li[{index}]//button["
                         f"@aria-label='More']"))

    def actress_movie_block_by_index(self, index):
        return (By.XPATH,
                (f"//div[@id='accordion-item-actress-previous-
projects']/div/ul/li[{index}]//button["
                         f"@aria-label='More']"))

    def bio_btn(self):
        return (By.XPATH,
                "//button[text()='Accept']")

    def get_photo_url(self):
        return self.find_element(*self.PHOTO).get_attribute("src")
```

```
    def get_actor_name(self):
        return self.find_element((By.XPATH,
                                  "//span[@class='hero__primary-text']")).text

    def get_about(self):
        return self.find_element(*self.ABOUT).text.strip()

    def get_movies(self):
        movie_elements = self.find_elements(*self.MOVIES)
        movies = []
        for movie in movie_elements:
            title = movie.find_element(By.TAG_NAME, "b").text.strip()
            year = movie.find_element(By.XPATH,
                                      ".//span[contains(@class,
'year_column')]").text.strip()
            movies.append({"title": title, "year": year})
        return movies

    def open_bio_page(self):
        self.click_on(self.BIO_LINK_ELEMENT)

    def get_date_of_birth(self):
        return self.find_element(self.DATE_OF_BIRTH).text

    def get_count_of_movies(self):
        try:
            return int(self.find_element(self.COUNT_OF_ACTOR_MOVIES).text)
        except:
            return int(self.find_element(self.COUNT_OF_ACTRESS_MOVIES).text)

    def expand_list_of_movies(self):
        try:
            self.find_element(self.SEE_ALL_BTN_ACTOR).click()
        except:
            try:
                self.find_element(self.SEE_ALL_BTN_ACTRESS).click()
            except:
                print("No btn 'SEE ALL")

    def click_on_movie_by_index(self, index):
        try:
            self.try_click(self.actor_movie_block_by_index(index))
        except:
            self.try_click(self.actress_movie_block_by_index(index))

    def click_on_close_btn_movie_block(self):
```

```
        self.find_element(self.CLOSE_MOVIE_BLOCK_BTN).click()

    def get_rating_of_movie(self):
        try:
            return float(self.find_element(self.RATING_OF_MOVIE).text)
        except:
            return "Not rated"
    def get_movie_data_by_index(self, index):

        image_link = self.get_movie_image_by_index(index)
        self.click_on_movie_by_index(index)

        title = self.find_element(self.MOVIE_TITLE).text
        year = self.find_element(self.RELEASE_YEAR).text
        genres = self.find_elements(self.LIST_OF_GENRES)
        list_of_genres = [genre.text for genre in genres]
        rating = self.get_rating_of_movie()


        movie_data = {
            "title": title,
            "year": year,
            "image_link": image_link,
            "rating": rating,
            "genres": list_of_genres
        }

        self.click_on(self.CLOSE_MOVIE_BLOCK_BTN)

        return movie_data

    def open_awards_page(self):
        self.try_click(self.AWARDS_LINK_ELEMENT)
```

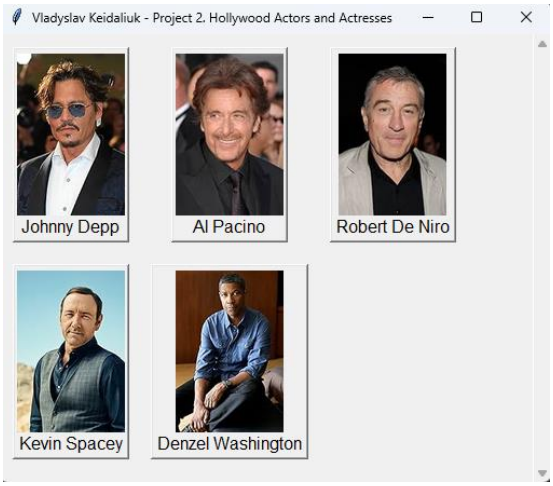
Demonstration of the GUI

At this stage, a demonstration of the graphical user interface (GUI) has been implemented using the Tkinter module. The interface provides a functional structure for interacting with the scraped data related to Hollywood actors and their movies. The key features of the GUI implemented so far include:

- Displaying a list of actors.

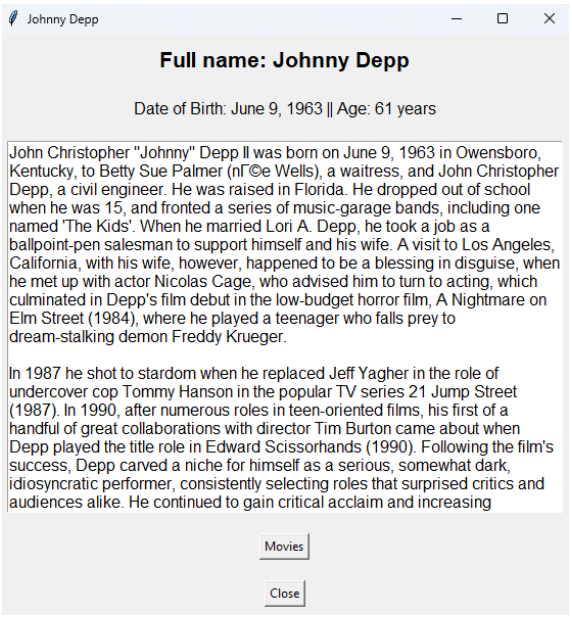
- Allowing users to click on an actor's name to view their details.
- Showing basic actor information such as name, biography, and photo.
- Providing a list of movies associated with the actor, including movie titles, release years, and ratings.

While this implementation is functional, **it is not the final version** of the GUI.



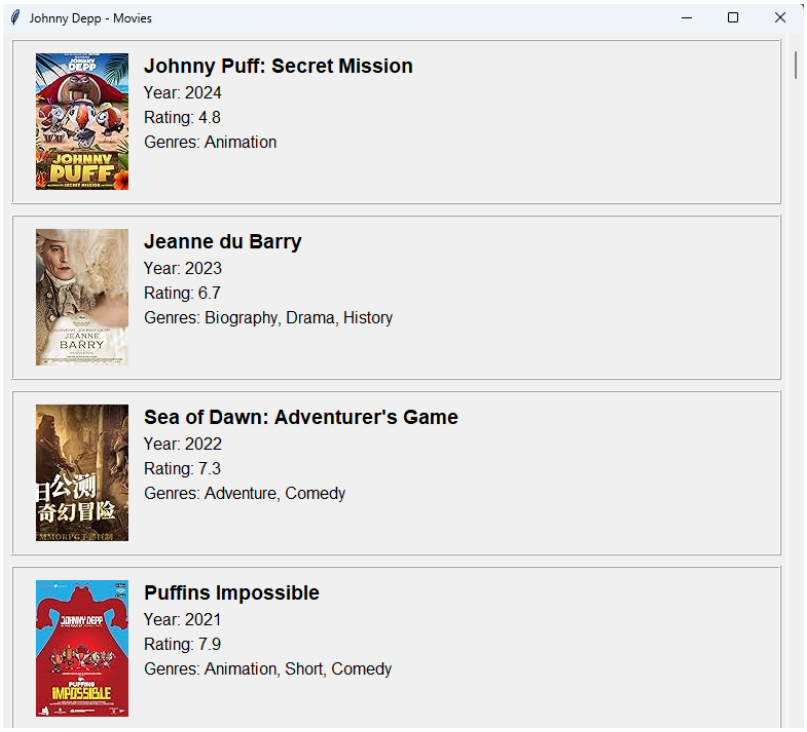
Picture 1. Main Page of application

After clicking on a particular actor, a page will open with their biography and the ability to go to a list of movies.



Picture 2. Information Page about actor/actress

If you click on the “Movies” button, a list of movies will be displayed.



Picture 3. Page with list of movies

Next steps

1. Scrape Actor/Actress Awards:

   - Need to implement functionality to scrape and display the awards associated with each actor/actress, including awards received in different years.

2. Implement Method to Gather All Movie Genres for an Actor/Actress:

   - Need to create a method that will iterate through all the movies in an actor's or actress's filmography, extract the genres of each movie, and then display a list of all unique genres associated with that actor or actress.

3. Display Average Movie Rating:

   - Need to calculate and show the average rating of all movies associated with an actor/actress, both overall and by year.

4. Display Top 5 Movies:

   - Identify and display the top 5 movies for each actor/actress based on their ratings

Conclusion

This report outlines the partial progress of the project, which focuses on scraping and displaying comprehensive data related to Hollywood actors and actresses. The completed steps include the setup and initialization of the web scraping system, which successfully extracts key actor details such as names, biographies, movie lists, genres, and ratings. Additionally, the graphical user interface (GUI) has been developed, allowing users to view actor profiles, including biographies and filmographies, in a user-friendly format.

The next phase of the project will involve implementing several essential features, such as scraping actor and actress awards, calculating average movie ratings, and displaying a list of all unique movie genres associated with each actor and displaying the top 5 movies based on ratings.

In conclusion, the project is progressing well, with the core functionalities for web scraping, data storage, and user interaction already established. The final version will feature a comprehensive IMDb-style recommendation system, highlighting top Hollywood actors and actresses based on dynamically retrieved and detailed information.