Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

09.03.01 "Информатика и вычислительная техника"

профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

**ОТЧЕТ**

по учебной практике

на кафедре Прикладной Математики и Кибернетики

Выполнил:

студент гр. ИП-311

Николаев М.Ю.

«5» апреля 2025г.

Руководитель практики

доцент каф. ПМиК Приставка П.А.

«5» апреля 2025г. Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Новосибирск 2025г.

Оглавление

[1.Условие задачи 3](#_Toc194750778)

[2. Описание используемых алгоритмов 4](#_Toc194750779)

[3. Листинг программы 6](#_Toc194750780)

[4. Результаты тестирования 22](#_Toc194750781)

[5. Список используемых источников 24](#_Toc194750782)

# **1.Условие задачи**

Разработать программу реализующую ввод, хранение и обработку данных о 250 лучших фильмах на основе данных сайта [https:/www.kinopoisk.ru/lists/movies/top250/](https://www.kinopoisk.ru/lists/movies/top250/)

Общие требования к программе:

1. Язык разработки: **Python версии не ниже 3.x**

2. Операционная система: определяются студентом

3. Набор свойств фильмов:

• Name – наименование • alternativeName • Year – год выпуска • shortDescription – короткое описание • movieLength- длительность • top250 – позиция в рейтинге

4. Ввод данных

Непосредственно с главной страницы сайта [https:/www.kinopoisk.ru/lists/movies/top250/](https://www.kinopoisk.ru/lists/movies/top250/) в момент запуска программы. Загрузка и парсинг веб-страницы производится с помощью библиотек Requests и Beautifulsoup или их аналогов. Примечание: допускается считывание строчек в количестве менее 250 (Например, 10 строчек с данными о фильмах).

5. Хранение:

Типы и структуры для хранения данных: определяются студентом

6. Обработка:

Реализовать функцию поиска информации о свойствах фильма по его названию.

# **2. Описание используемых алгоритмов**

Я занимался следующими алгоритмами:

**В слое data:**

**class MovieRepository(IMovieRepository)** – наследник абстрактного класса IMovieRepository, управляет данными о фильмах. В нём я реализовал следующие методы:

**get\_top250(self) -> List[Movie]** – получает топ-250 фильмов из API Кинопоиска или из кэша, если данные уже сохранены.

**Возвращает**: список объектов `Movie`, содержащих информацию о фильмах.

**QuickSort(self, movies: List[Movie], L: int, R: int)** – алгоритм быстрой сортировки для фильмов по их названию (без учета регистра). Предназначен для реализации двоичного поиска, который сделал мой напарник.

**class CacheManager** – управляет кэшированием данных в JSON-файл. Создает директорию для кэша при инициализации и предоставляет методы для загрузки и сохранения данных. Содержит в себе методы:

**load(self)** – Загружает данные из кэш-файла (JSON и возвращает загруженные данные (массив или словарь) или None при ошибке.

**save(self, data)** – Сохраняет переданные данные в кэш-файл (JSON).

**В слое domain:**

**class IMovieRepository(ABC)** - абстрактный класс, определяющий интерфейс для работы с данными о фильмах. Содержит в себе абстрактные методы:

**get\_top250(self) -> List[Movie]** – метод, предназначенный для получения списка топ-250 фильмов. Возвращает список объектов типа Movie.

**search\_movies(self, movies: List[Movie], query: str) -> List[Movie]** – принимает список фильмов для поиска и ключ поиска. Возвращает список найденных фильмов.

**sort\_movies(self, movies: List[Movie]) -> List[Movie]** - сортирует список фильмов

Введение абстракции гарантирует, что все методы будут реализованы в классах-наследников. Таким образом, наш код выполняет один из принципов SOLID, а именно принцип подстановки Лисков, позволяя легко заменять реализацию работы с данными.

**В слое presentation:**

В этом слое приложения содержатся функции, поддерживающие графическое отображение.

**setup\_navigation(self)** - создает панель навигации: кнопки переключения страниц, поле ввода номера страницы, элементы поиска (поле ввода и кнопки).

**setup\_scrollable\_area(self)** - настраивает прокручиваемую область для отображения фильмов, создает canvas с вертикальным скроллбаром.

**create\_movie\_cards(self, movies: List[Movie])** - создает карточки фильмов в графическом интерфейсе. При необходимости выводит информацию о фильме в консоль.

**create\_movie\_card(self, movie: Movie)** - создает отдельную карточку фильма с постером, рейтингом и основной информации о фильме.

**print\_movie\_to\_console(self, movie: Movie)** - выводит информацию о фильме в консоль разными цветами в зависимости от жанра.

**update\_movies\_display(self, movies: List[Movie]** - обновляет отображение фильмов на текущей странице (по 10 записей на странице).

**prev\_page(self) и next\_page(self)** - реализуют навигацию по страницам (вперёд, назад)

**go\_to\_page(self)** - переходит на указанную страницу и проверяет корректность введенного номера страницы.

**start\_search(self) и end\_search(self)** - реализуют функционал поиска по названию фильма и останавливают поиск.

# **3. Листинг программы**

//models.py

**from dataclasses import dataclass from typing import List, Optional, Dict, Any**

**@dataclass**

**class Movie: id: str name: str year: int rating: float genres: List[str] countries: List[str] poster\_url: Optional[str]**

**@classmethod**

**def from\_dict(cls, data: Dict[str, Any]) -> 'Movie':**

**return cls(**

**id=str(data.get('id', '')),**

**name=data.get('name', ''),**

**year=data.get('year', 0),**

**rating=data.get('rating', {}).get('kp', 0.0),**

**genres=[g.get('name', '') for g in data.get('genres', [])],**

**countries=[c.get('name', '') for c in data.get('countries', [])],**

**poster\_url=data.get('poster', {}).get('url', None)**

**)**

//movie\_Irepository.py

**from abc import ABC, abstractmethod**

**from typing import List**

**from domain.models import Movie**

**class IMovieRepository(ABC):**

**@abstractmethod**

**def get\_top250(self) -> List[Movie]:**

**pass**

**@abstractmethod**

**def search\_movies(self, movies: List[Movie], query: str) -> List[Movie]:**

**pass**

**@abstractmethod**

**def sort\_movies(self, movies: List[Movie]) -> List[Movie]:**

**pass**

//kinopoisk\_api

**import requests**

**from config import API\_KEY, API\_URL**

**class KinopoiskAPI:**

**def \_\_init\_\_(self):**

**self.headers = {'X-API-KEY': API\_KEY}**

**self.base\_url = API\_URL**

**def get\_top250(self):**

**params = {**

**'selectFields': ['name', 'rating.kp', 'year', 'genres.name', 'countries.name', 'poster.url'],**

**'lists': 'top250',**

**'limit': 250**

**}**

**try:**

**response = requests.get(self.base\_url, headers=self.headers, params=params, timeout=10)**

**if response.status\_code == 200:**

**return response.json().get('docs', [])**

**except requests.RequestException:**

**return []**

**return []**

//movie\_repository.py

**from typing import List**

**from domain.models import Movie**

**from infrastructure.api.kinopoisk\_api import KinopoiskAPI**

**from infrastructure.cache.cache\_manager import CacheManager**

**from functools import lru\_cache**

**from domain.movie\_Irepository import IMovieRepository**

**class MovieRepository(IMovieRepository):**

**def \_\_init\_\_(self):**

**self.api = KinopoiskAPI()**

**self.cache = CacheManager('data/cache/movies\_cache.json')**

**@lru\_cache(maxsize=1)**

**def get\_top250(self) -> List[Movie]:**

**cached = self.cache.load()**

**if cached:**

**return [Movie.from\_dict(m) for m in cached]**

**data = self.api.get\_top250()**

**if data:**

**self.cache.save(data)**

**return [Movie.from\_dict(m) for m in data]**

**return []**

**def QuickSort(self, movies: List[Movie], L: int, R: int) -> None:**

**while L < R:**

**i, j = L, R**

**pivot = movies[L].name.lower()**

**while i <= j:**

**while movies[i].name.lower() < pivot:**

**i += 1**

**while movies[j].name.lower() > pivot:**

**j -= 1**

**if i <= j:**

**movies[i], movies[j] = movies[j], movies[i]**

**i += 1**

**j -= 1**

**if (j - L) < (R - i):**

**self.QuickSort(movies, L, j)**

**L = i**

**else:**

**self.QuickSort(movies, i, R)**

**R = j**

**def binary\_search(self, sorted\_movies: List[Movie], query: str) -> List[Movie]:**

**if not query:**

**return []**

**query = query.lower()**

**left, right = 0, len(sorted\_movies) - 1**

**while left < right:**

**mid = (left + right) // 2**

**if sorted\_movies[mid].name.lower() >= query:**

**right = mid**

**else:**

**left = mid + 1**

**result = []**

**while left < len(sorted\_movies) and query in sorted\_movies[left].name.lower():**

**result.append(sorted\_movies[left])**

**left += 1**

**return result**

**def search\_movies(self, movies: List[Movie], query: str) -> List[Movie]:**

**if not query:**

**return []**

**sorted\_movies = movies.copy()**

**self.QuickSort(sorted\_movies, 0, len(sorted\_movies) - 1)**

**return self.binary\_search(sorted\_movies, query)**

**def sort\_movies(self, movies: List[Movie]) -> List[Movie]:**

**sorted\_movies = movies.copy()**

**self.QuickSort(sorted\_movies, 0, len(sorted\_movies) - 1)**

**return sorted\_movies**

//image\_storage.py

**import os**

**from hashlib import md5**

**from PIL import Image, ImageTk**

**from io import BytesIO**

**import requests**

**class ImageStorage:**

**def \_\_init\_\_(self, cache\_dir: str):**

**self.cache\_dir = cache\_dir**

**os.makedirs(cache\_dir, exist\_ok=True)**

**def get\_image(self, url: str, size=(136, 204)):**

**if not url:**

**return None**

**cache\_name = f"{md5(url.encode()).hexdigest()}\_{size[0]}x{size[1]}.png"**

**cache\_path = os.path.join(self.cache\_dir, cache\_name)**

**if os.path.exists(cache\_path):**

**try:**

**return ImageTk.PhotoImage(Image.open(cache\_path))**

**except Exception:**

**os.remove(cache\_path)**

**try:**

**response = requests.get(url, timeout=10)**

**if response.status\_code == 200:**

**img = Image.open(BytesIO(response.content))**

**img = img.resize(size, Image.LANCZOS)**

**img.save(cache\_path, "PNG")**

**return ImageTk.PhotoImage(img)**

**except Exception:**

**return None**

//movie\_service.py

**from domain.movie\_Irepository import IMovieRepository**

**from domain.models import Movie**

**from typing import List**

**class MovieService:**

**def \_\_init\_\_(self, repository: IMovieRepository):**

**self.repository = repository**

**def fetch\_movies(self) -> List[Movie]:**

**return self.repository.get\_top250()**

**def search\_movies(self, query: str) -> List[Movie]:**

**movies = self.repository.get\_top250()**

**return self.repository.search\_movies(movies, query)**

**def get\_sorted\_movies(self) -> List[Movie]:**

**movies = self.repository.get\_top250()**

**return self.repository.sort\_movies(movies)**

//movie\_ui.py

**from tkinter import messagebox, Frame, Label, Button, Entry, Canvas, Scrollbar**

**from typing import List**

**from domain.models import Movie**

**from infrastructure.storage.image\_storage import ImageStorage**

**from colorama import Fore, Style, init**

**from infrastructure.repository.movie\_repository import MovieRepository**

**class MovieUI:**

**def \_\_init\_\_(self, root):**

**self.root = root**

**self.movies: List[Movie] = []**

**self.movies\_for\_search: List[Movie] = []**

**self.page = 1**

**self.print\_in\_terminal = True**

**self.image\_storage = ImageStorage('data/images')**

**self.setup\_ui()**

**init(autoreset=True)**

**def setup\_ui(self):**

**self.root.title("Кинопоиск: Топ 250 фильмов")**

**self.root.geometry("650x700")**

**self.setup\_navigation()**

**self.setup\_scrollable\_area()**

**def setup\_navigation(self):**

**nav\_frame = Frame(self.root, bg="lightgray", pady=10)**

**nav\_frame.pack(fill="x")**

**Button(nav\_frame, text="← Предыдущая", command=self.prev\_page, font=("Arial", 8)).pack(side="left", padx=5)**

**Button(nav\_frame, text="Следующая →", command=self.next\_page, font=("Arial", 8)).pack(side="left", padx=0)**

**self.page\_entry = Entry(nav\_frame, width=10)**

**self.page\_entry.pack(side="left", padx=10)**

**Button(nav\_frame, text="Перейти", command=self.go\_to\_page).pack(side="left", padx=10)**

**self.page\_label = Label(nav\_frame, text=f"Номер страницы: {self.page}", font=("Arial", 8))**

**self.page\_label.pack(side="left")**

**Button(nav\_frame, text="Поиск", command=self.start\_search, font=("Arial", 8)).pack(side="left", padx=5)**

**Button(nav\_frame, text="Окончить поиск", command=self.end\_search, font=("Arial", 6)).pack(side="left")**

**self.search\_entry = Entry(nav\_frame, font=("Arial", 12), width=10)**

**self.search\_entry.pack(side="left", padx=5)**

**def setup\_scrollable\_area(self):**

**self.canvas = Canvas(self.root, bg="white")**

**self.scrollbar = Scrollbar(self.root, orient="vertical", command=self.canvas.yview)**

**self.canvas.configure(yscrollcommand=self.scrollbar.set)**

**self.canvas.bind('<Configure>', lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))**

**self.scrollbar.pack(side="right", fill="y")**

**self.canvas.pack(side="left", fill="both", expand=True)**

**self.movie\_frame = Frame(self.canvas, bg="white")**

**self.canvas.create\_window((0, 0), window=self.movie\_frame, anchor="nw")**

**def create\_movie\_cards(self, movies: List[Movie]):**

**for widget in self.movie\_frame.winfo\_children():**

**widget.destroy()**

**for movie in movies:**

**self.create\_movie\_card(movie)**

**if self.print\_in\_terminal:**

**self.print\_movie\_to\_console(movie)**

**def create\_movie\_card(self, movie: Movie):**

**movie\_item\_frame = Frame(self.movie\_frame, bg="white", bd=2, relief="groove")**

**movie\_item\_frame.pack(fill="x", pady=5, padx=10, expand=True)**

**poster\_image = self.image\_storage.get\_image(movie.poster\_url)**

**poster\_frame = Frame(movie\_item\_frame, bg="white")**

**poster\_frame.pack(side="left", padx=10, pady=10)**

**if poster\_image:**

**poster\_label = Label(poster\_frame, image=poster\_image, bg="white")**

**poster\_label.image = poster\_image**

**poster\_label.pack()**

**info\_frame = Frame(movie\_item\_frame, bg="white")**

**info\_frame.pack(side="left", fill="both", expand=True, padx=10, pady=10)**

**name\_label = Label(info\_frame, text=movie.name, bg="white", font=("Arial", 12, "bold"))**

**name\_label.pack(anchor="w")**

**rating\_frame = Frame(info\_frame, bg="white")**

**rating\_frame.pack(anchor="w")**

**Label(rating\_frame, text=f"Рейтинг: {movie.rating}", bg="white", font=("Arial", 12)).pack(side="left")**

**Label(rating\_frame, text="⭐", fg="gold", bg="white", font=("Arial", 12)).pack(side="left", padx=5)**

**info\_text = (**

**f"Год: {movie.year}\n"**

**f"Жанры: {', '.join(movie.genres)}\n"**

**f"Страны: {', '.join(movie.countries)}"**

**)**

**Label(info\_frame, text=info\_text, bg="white", justify="left", font=("Arial", 12)).pack(anchor="w")**

**def print\_movie\_to\_console(self, movie: Movie):**

**color = self.get\_color\_by\_genre(movie.genres)**

**print(f"{color}{movie.name}")**

**print(f"Рейтинг: {movie.rating}")**

**print(f"Год: {movie.year}")**

**print(f"Жанры: {', '.join(movie.genres)}")**

**print(f"Страны: {', '.join(movie.countries)}")**

**print("-" \* 40)**

**def get\_color\_by\_genre(self, genres: List[str]):**

**genre\_colors = {**

**'драма': Fore.RED,**

**'комедия': Fore.GREEN,**

**'боевик': Fore.YELLOW,**

**'фантастика': Fore.BLUE,**

**'триллер': Fore.MAGENTA,**

**'мелодрама': Fore.CYAN,**

**'детектив': Fore.WHITE,**

**}**

**for genre in genres:**

**if genre.lower() in genre\_colors:**

**return genre\_colors[genre.lower()]**

**return Fore.WHITE**

**def update\_movies\_display(self, movies: List[Movie]):**

**start\_idx = (self.page - 1) \* 10**

**end\_idx = self.page \* 10**

**movies\_to\_show = movies[start\_idx:end\_idx]**

**if movies\_to\_show:**

**self.page\_label.config(text=f"Номер страницы: {self.page}")**

**self.create\_movie\_cards(movies\_to\_show)**

**return True**

**else:**

**messagebox.showinfo("Информация", "Фильмы закончились или произошла ошибка.")**

**return False**

**def prev\_page(self):**

**if self.page > 1:**

**self.page -= 1**

**self.update\_movies\_display(self.movies)**

**def next\_page(self):**

**self.page += 1**

**if not self.update\_movies\_display(self.movies):**

**self.page -= 1**

**def go\_to\_page(self):**

**try:**

**new\_page = int(self.page\_entry.get()) if self.page\_entry.get().isdigit() else 0**

**if 0 < new\_page <= (len(self.movies) // 10 + 1):**

**self.page = new\_page**

**self.update\_movies\_display(self.movies)**

**return True**

**else:**

**messagebox.showwarning("Ошибка", "Некорректный номер страницы.")**

**return False**

**except ValueError:**

**messagebox.showwarning("Ошибка", "Введите число.")**

**return False**

**def start\_search(self):**

**key = self.search\_entry.get()**

**if key:**

**if not self.movies\_for\_search:**

**self.movies\_for\_search = self.movies.copy()**

**repo = MovieRepository()**

**repo.QuickSort(self.movies\_for\_search, 0, len(self.movies\_for\_search) - 1)**

**repo = MovieRepository()**

**found\_movies = repo.binary\_search(self.movies\_for\_search, key)**

**self.create\_movie\_cards(found\_movies)**

**def end\_search(self):**

**self.search\_entry.delete(0, 'end')**

**self.update\_movies\_display(self.movies)**

//test\_ui.py

**import pytest**

**from unittest.mock import Mock, patch, MagicMock**

**from ui.movie\_ui import MovieUI**

**import tkinter as tk**

**from tkinter import Tk**

**from domain.models import Movie**

**@pytest.fixture**

**def root\_window():**

**root = Tk()**

**yield root**

**root.destroy()**

**def test\_navigation\_buttons(root\_window):**

**ui = MovieUI(root\_window)**

**nav\_frame = [w for w in root\_window.winfo\_children() if isinstance(w, tk.Frame)][0]**

**buttons = [w for w in nav\_frame.winfo\_children() if isinstance(w, tk.Button)]**

**assert len(buttons) >= 4, "Navigation buttons are missing"**

**def test\_scrollable\_area(root\_window):**

**ui = MovieUI(root\_window)**

**canvas = [w for w in root\_window.winfo\_children() if isinstance(w, tk.Canvas)]**

**assert canvas, "Canvas for scrollable area is missing"**

**def test\_scrollbar(root\_window):**

**ui = MovieUI(root\_window)**

**assert any(isinstance(widget, tk.Scrollbar) for widget in root\_window.winfo\_children()), "Scrollbar is missing"**

**def test\_movie\_frame(root\_window):**

**ui = MovieUI(root\_window)**

**canvas = [w for w in root\_window.winfo\_children() if isinstance(w, tk.Canvas)]**

**assert canvas, "Movie frame is missing"**

**@patch('ui.movie\_ui.ImageStorage')**

**@patch('ui.movie\_ui.Label')**

**def test\_load\_image\_from\_url(mock\_label, mock\_imagestorage, root\_window):**

**mock\_image = MagicMock()**

**mock\_imagestorage.return\_value.get\_image.return\_value = mock\_image**

**ui = MovieUI(root\_window)**

**test\_movie = Movie(**

**id="1",**

**name="Test Movie",**

**year=2020,**

**rating=8.0,**

**genres=["Drama"],**

**countries=["USA"],**

**poster\_url="test\_url"**

**)**

**ui.create\_movie\_card(test\_movie)**

**mock\_imagestorage.return\_value.get\_image.assert\_called\_once\_with("test\_url")**

**mock\_label.assert\_called()**

//main.py

**import tkinter as tk**

**from tkinter import messagebox**

**from ui.movie\_ui import MovieUI**

**from infrastructure.repository.movie\_repository import MovieRepository**

**def main():**

**root = tk.Tk()**

**repo = MovieRepository()**

**movies = repo.get\_top250()**

**if not movies:**

**messagebox.showwarning("Ошибка", "Не удалось загрузить данные.")**

**return**

**app = MovieUI(root)**

**app.movies = movies**

**app.movies\_for\_search = None**

**app.update\_movies\_display(movies)**

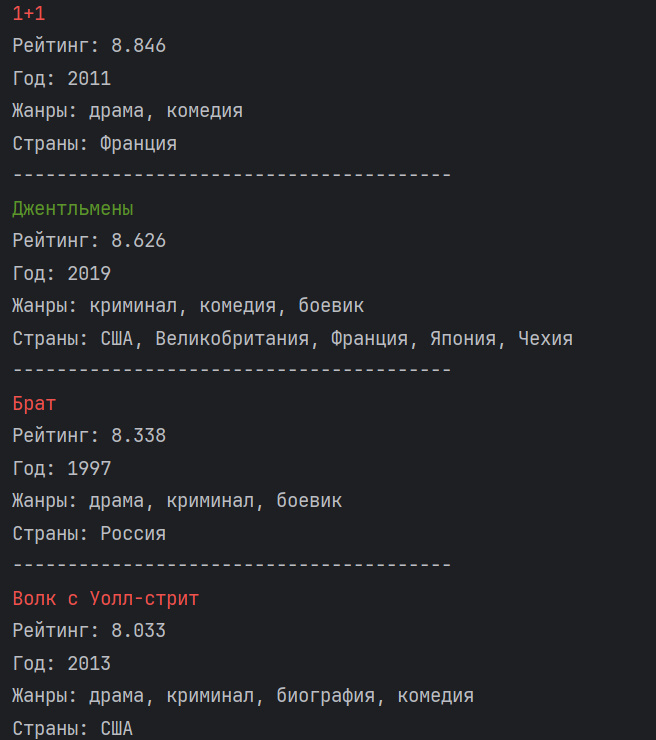
**root.mainloop()**

**if \_\_name\_\_ == "\_\_main\_\_":**

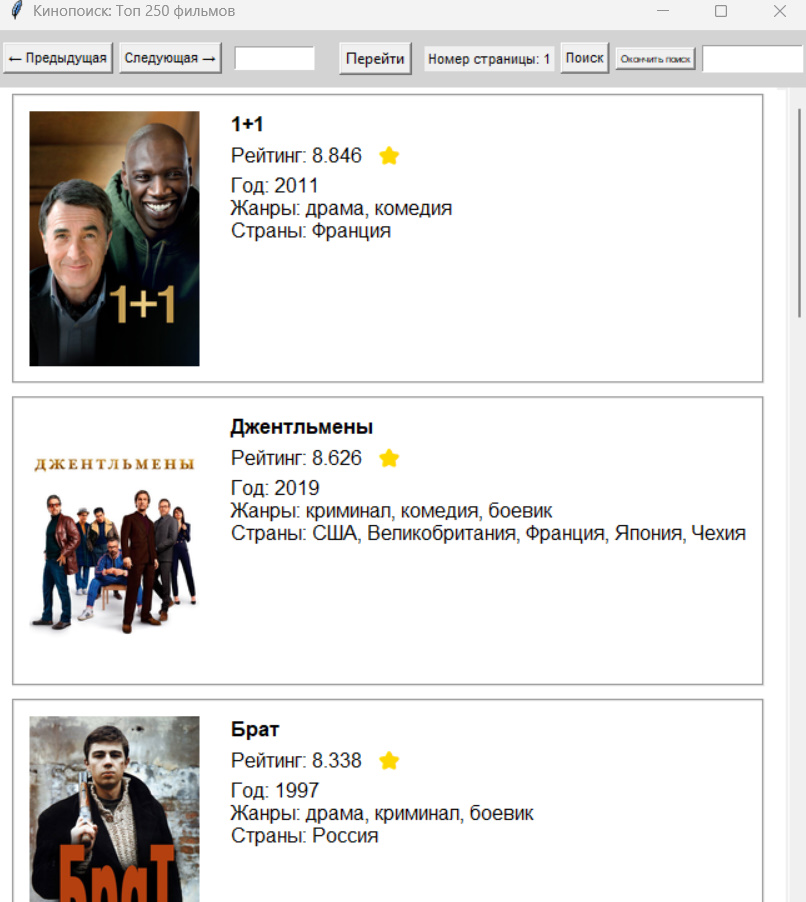
**main()**

# **4. Результаты тестирования**

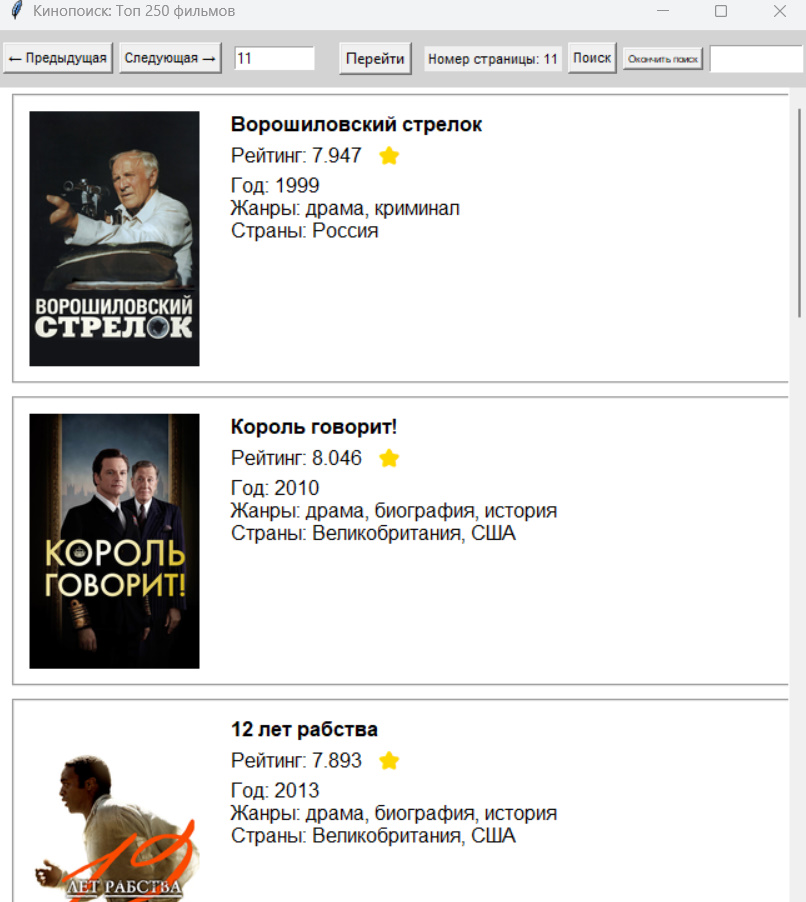
1. Консольный вывод данных



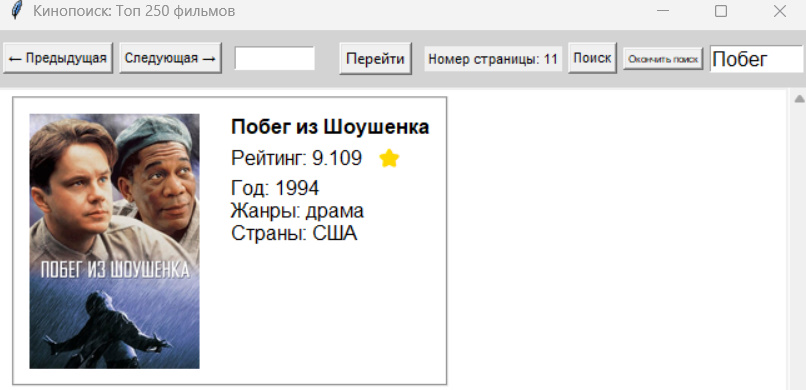
1. Графический вывод (первая страница)



1. Переход на конкретную страницу



1. Поиск фильма по названию



# **5. Список используемых источников**

1. Руководство по языку программирования Python: сайт – URL: <https://metanit.com/python/tutorial/> (дата обращения: 15.02.2025)
2. Объектно-ориентированное программирование на Python: сайт – URL:  [https://metanit.com/python/tutorial/7.1.php/](https://metanit.com/python/tutorial/) (дата обращения: 18.02.2025)
3. Словари в языке Python: сайт – URL:  [https://metanit.com/python/tutorial/3.3.php/](https://metanit.com/python/tutorial/) (дата обращения: 25.02.2025)
4. Python Testing с pytest. Начало работы с pytest: сайт – URL:  [https://habr.com/ru/articles/448782/](%20https://habr.com/ru/articles/448782/) (дата обращения: 1.03.2025)
5. functools — Higher-order functions and operations on callable objects: сайт – URL: <https://docs.python.org/3/library/functools.html> (дата обращения: 9.03.2025)
6. Руководство по Tkinter: сайт – URL: <https://metanit.com/python/tkinter/l> (дата обращения: 15.03.2025)
7. Библиотека requests сайт – URL: <https://education.yandex.ru/handbook/python/article/modul-requests> (дата обращения: 28.03.2025)