Cinema booking system

Ignas Kirkilionis EKF-23

**Introduction**

The goal of this coursework is to develop a cinema management system that enables users to manage movies and showtimes, book seats for a selected show, and print tickets. The application, named CinemaBookingSystem, is designed to streamline the process of managing movie schedules and seat reservations for both theater administrators and customers.

**How to run the program:**

1. Ensure you have Python installed on your system.
2. Save the provided code into a file named cinema\_booking\_system.py.
3. Place the movie and showtime data in movies.txt and showtimes.txt files respectively.
4. Run the program using the command python cinema\_booking\_system.py.

**How to use the program:**

1. The program will automatically load movies and showtimes from the provided text files.
2. Follow the prompts to choose a movie and a showtime.
3. Specify the number of seats to book and select the seats.
4. The program will confirm the booking and print tickets to text files.

**Polymorphism**

Polymorphism allows methods to do different things based on the object it is acting upon. In this case, the create\_show method of the ShowFactory class creates different types of shows based on the show\_type parameter.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas

Automatiškai sugeneruotas aprašymas

Here, create\_show method can return different types of shows (currently only MovieShow) based on the show\_type.

**Abstraction**

Abstraction is demonstrated by the Show class, which provides abstract methods that must be implemented by any subclass.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, Multimedijos programinė įranga

Automatiškai sugeneruotas aprašymas

The Show class cannot be instantiated directly and mandates that subclasses implement the display\_seats, book\_seat, and choose\_seats methods.

**Inheritance**

Inheritance allows a class to inherit attributes and methods from another class. The MovieShow class inherits from the Show class and implements its abstract methods.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga

Automatiškai sugeneruotas aprašymasThe MovieShow class inherits the constructor from the Show class and provides concrete implementations for the abstract methods.

### **Encapsulation**

Encapsulation is the bundling of data with the methods that operate on that data. The Theater class encapsulates the details of managing movies and showtimes.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, programinė įranga, Multimedijos programinė įranga

Automatiškai sugeneruotas aprašymasPaveikslėlis, kuriame yra tekstas, ekrano kopija

Automatiškai sugeneruotas aprašymas

**Encapsulation** is evident in how the movies and showtimes lists are managed within the class, with methods provided to add and list these entities, while keeping the underlying data structures hidden from outside access.

**Singleton Pattern**

The Singleton pattern ensures that a class has only one instance and provides a global point of access to it.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas

Automatiškai sugeneruotas aprašymas

* The class variable \_instance is used to store the single instance of the Theater class.
* The \_\_new\_\_ method checks if \_instance is None. If it is, it creates a new instance of the class. If not, it returns the existing instance.
* This ensures that only one instance of Theater exists throughout the program.

**Factory Method Pattern**

The **Factory Method Pattern** defines an interface for creating an object, but lets subclasses alter the type of objects that will be created. It is used to delegate the responsibility of object instantiation to subclasses or helper methods.

Paveikslėlis, kuriame yra tekstas, ekrano kopija, Šriftas, linija

Automatiškai sugeneruotas aprašymas

*  The create\_show method is a static method that does not require an instance of the ShowFactory class to be called.
* The method takes show\_type as a parameter to determine which type of show object to create.
* If show\_type is "MovieShow", it creates and returns an instance of MovieShow.

### **Results**

* Implementing the singleton pattern for the Theater class ensured that only one instance of the theater exists, simplifying state management.
* The seat booking functionality had to handle concurrent bookings, ensuring no double bookings occurred.
* Loading data from text files and handling errors gracefully was a challenge but necessary for robust file input/output operations.

**Conclusion**

The **CinemaBookingSystem** successfully implements a theater management system that allows users to manage movies, showtimes, and seat bookings. This work has achieved a functional and user-interactive application.

**Key findings and outcomes:**

* The use of OOP principles improved the modularity and maintainability of the code.
* Error handling and input validation ensured a user-friendly experience.
* The program successfully generates printable tickets for booked seats.

**Future prospects:**

* Implement a database for persistent storage of movies, showtimes, and bookings.
* Add more advanced booking features, such as group bookings and seat selection algorithms for optimal placement.