

# **KISHKINDA UNIVERSITY, BALLARI**



**Mini Project Report**

**On**

**“Film Licensing Compliance”**

**Department of CSE**

**Submitted by:**

**Spoorthi.C**

**KUB23CSE140**

## Table of Contents

<b>Sl. No</b>	<b><u>Contents</u></b>	<b>Page no</b>
<b>1.</b>	<b>Introduction</b>	<b>1 - 9</b>
	<b>1.1 Background information on the Project</b>	<b>1</b>
	<b>1.2 Problem statement or Purpose of the Project</b>	<b>1 - 2</b>
	<b>1.3 Scope of the project</b>	<b>3 - 4</b>
	<b>1.4 Limitations of the project</b>	<b>5 - 9</b>
<b>2.</b>	<b>Objective</b>	<b>9 - 11</b>
	<b>2.1 The main goal or aims of the project</b>	<b>--</b>
<b>3.</b>	<b>Methodology</b>	<b>11 - 16</b>
	<b>3.1 Description of the process</b>	<b>11 - 14</b>
	<b>3.2 Tools, software or techniques used</b>	<b>14 - 16</b>
<b>4.</b>	<b>Results/Findings</b>	<b>16 - 19</b>
	<b>4.1 The results of the project</b>	<b>16 - 19</b>
	<b>4.2 Analysis of the project</b>	<b>--</b>
<b>5.</b>	<b>Conclusion</b>	<b>20</b>
<b>6.</b>	<b>Future Enhancement</b>	<b>21</b>
<b>7.</b>	<b>References</b>	

# Introduction

## 1.1 Problem Statement:

To address the problem statement regarding a proof of concept for Film Licensing Compliance, we will create a Python solution using Object-Oriented Programming (OOP) and Data Structures and Algorithms (DSA) principles.

## 1.2 Scope of the Project

### 1. Film Class:

- Represents individual films.
- Attributes: film\_id, title, director, release\_date, and other relevant metadata.
- Methods:
  - `__init__()`: Initialize film details.
  - `__str__()`: String representation for easy display.

### 2. LicensingRecord Class:

- Manages licensing details for each film.
- Attributes: film\_id, license\_type, expiration\_date, compliance\_status, and any additional licensing information.
- Methods:
  - `__init__()`: Initialize licensing details.
  - `is_compliant()`: Check if the licensing record is compliant based on certain criteria.
  - `__str__()`: String representation for easy display.

### 3. FilmManager Class:

- Handles CRUD operations on films and their licensing records.

- Attributes: films (list of Film objects), licensing\_records (list of LicensingRecord objects).
- Methods:
  - add\_film(film): Add a new film to the collection.
  - remove\_film(film\_id): Remove a film based on its ID.
  - update\_film(film\_id, updated\_film): Update film details.
  - get\_film(film\_id): Retrieve film details.
  - add\_licensing\_record(record): Add a licensing record for a film.
  - remove\_licensing\_record(film\_id): Remove a licensing record for a film.
  - update\_licensing\_record(film\_id, updated\_record): Update licensing details.
  - ensure\_compliance\_with\_regulations(compliance\_data): Check compliance against regulations.

### **Limitations of the project:**

1.
  - The current implementation may not handle a large number of films and licensing records efficiently. As the dataset grows, performance may degrade due to linear searches.
2. **Data Persistence:**
  - The project does not include persistent storage (e.g., databases). If the application is restarted, all data will be lost unless implemented with a database or file storage.
3. **Compliance Logic Complexity:**
  - The compliance checking function is not fully defined, and its implementation may become complex depending on the variety of regulations across different regions and types of films.

#### **4. Error Handling:**

- Minimal error handling is present. For instance, invalid film IDs, incorrect data formats, or missing licensing information could lead to runtime errors.

#### **5. User Interface:**

- The proof of concept lacks a user interface (UI), limiting user interaction to command-line execution, which may not be user-friendly for non-technical users.

#### **6. Regulatory Updates:**

- The project does not account for changes in film regulations over time, which could impact compliance checks and licensing requirements.

#### **7. Testing:**

- Limited testing coverage may exist. Without comprehensive unit tests, it may be difficult to ensure the reliability and correctness of the code.

#### **8. Documentation:**

- Inadequate documentation could hinder understanding and usability, especially for new developers or users who may need to work with the codebase.

#### **9. Concurrency Issues:**

- If multiple users attempt to access and modify the records simultaneously (in a web or multi-threaded environment), data integrity could be compromised without proper locking mechanisms.

#### **10. Business Logic:**

- The project focuses primarily on data structures and operations, possibly neglecting broader business logic related to film licensing (e.g., costs, negotiations, renewals).

### **Mitigation Strategies**

To address these limitations, consider the following strategies:

- **Implement a Database:** Use a relational or NoSQL database for persistent storage and efficient data retrieval.
- **Enhance Compliance Logic:** Define clear rules for compliance checks and incorporate external APIs or databases for regulatory information.
- **Add Error Handling:** Implement try-except blocks and validations to handle potential errors gracefully.
- **Develop a User Interface:** Create a web or desktop UI to facilitate easier interaction with the system.
- **Conduct Thorough Testing:** Implement unit tests, integration tests, and user acceptance tests to ensure robustness.
- **Maintain Documentation:** Provide comprehensive documentation covering setup, usage, and code structure.
- **Consider Concurrency Control:** Use locks or transactions to manage concurrent access if applicable.

## Code of the project statement:-

```
class Film:

    def __init__(self, film_id, title, director, year):

        self.film_id = film_id

        self.title = title

        self.director = director

        self.year = year


    def __str__(self):

        return f'{self.film_id}: {self.title} by {self.director} ({self.year})'


class LicensingRecord:

    def __init__(self, film_id, licensing_info, compliance_status):

        self.film_id = film_id

        self.licensing_info = licensing_info

        self.compliance_status = compliance_status


    def __str__(self):

        return f'Licensing Record for Film ID {self.film_id}: {self.licensing_info}, Compliance: {self.compliance_status}'


class FilmManager:

    def __init__(self):
```

```
self.films = {}
```

```
self.licensing_records = {}
```

```
def add_film(self):
```

```
    film_id = input("Enter Film ID: ")
```

```
    title = input("Enter Film Title: ")
```

```
    director = input("Enter Film Director: ")
```

```
    year = input("Enter Film Release Year: ")
```

```
    film = Film(film_id, title, director, year)
```

```
    self.films[film_id] = film
```

```
    print(f"Film added: {film}")
```

```
def remove_film(self):
```

```
    film_id = input("Enter Film ID to remove: ")
```

```
    if film_id in self.films:
```

```
        del self.films[film_id]
```

```
        print(f"Film ID {film_id} removed.")
```

```
    else:
```

```
        print("Film not found.")
```

```
def update_film(self):
```

```
    film_id = input("Enter Film ID to update: ")
```

```
    if film_id in self.films:
```



```
title = input("Enter new Film Title: ")

director = input("Enter new Film Director: ")

year = input("Enter new Film Release Year: ")

self.films[film_id] = Film(film_id, title, director, year)

print(f"Film updated: {self.films[film_id]}")

else:

    print("Film not found.")


def view_films(self):

    for film in self.films.values():

        print(film)


def manage_film_licensing_requirements(self):

    film_id = input("Enter Film ID to manage licensing: ")

    if film_id in self.films:

        licensing_info = input("Enter Licensing Information: ")

        compliance_status = input("Enter Compliance Status (Compliant/Non-Compliant): ")

        record = LicensingRecord(film_id, licensing_info, compliance_status)

        self.licensing_records[film_id] = record

        print(f"Licensing record added: {record}")

    else:

        print("Film not found.")
```

```
def ensure_compliance_with_regulations(self):  
  
    film_id = input("Enter Film ID to check compliance: ")  
  
    if film_id in self.licensing_records:  
  
        record = self.licensing_records[film_id]  
  
        print(f"Compliance Check for Film ID {film_id}: {record.compliance_status}")  
  
    else:  
  
        print("Licensing record not found.")
```

```
def main():  
  
    manager = FilmManager()  
  
    while True:  
  
        print("\nFilm Licensing Compliance Manager")  
  
        print("1. Add Film")  
  
        print("2. Remove Film")  
  
        print("3. Update Film")  
  
        print("4. View Films")  
  
        print("5. Manage Film Licensing Requirements")  
  
        print("6. Ensure Compliance with Regulations")  
  
        print("7. Exit")  
  
        choice = input("Select an option (1-7): ")
```

```
if choice == '1':

    manager.add_film()

elif choice == '2':

    manager.remove_film()

elif choice == '3':

    manager.update_film()

elif choice == '4':

    manager.view_films()

elif choice == '5':

    manager.manage_film_licensing_requirements()

elif choice == '6':

    manager.ensure_compliance_with_regulations()

elif choice == '7':

    print("Exiting program.")

    break

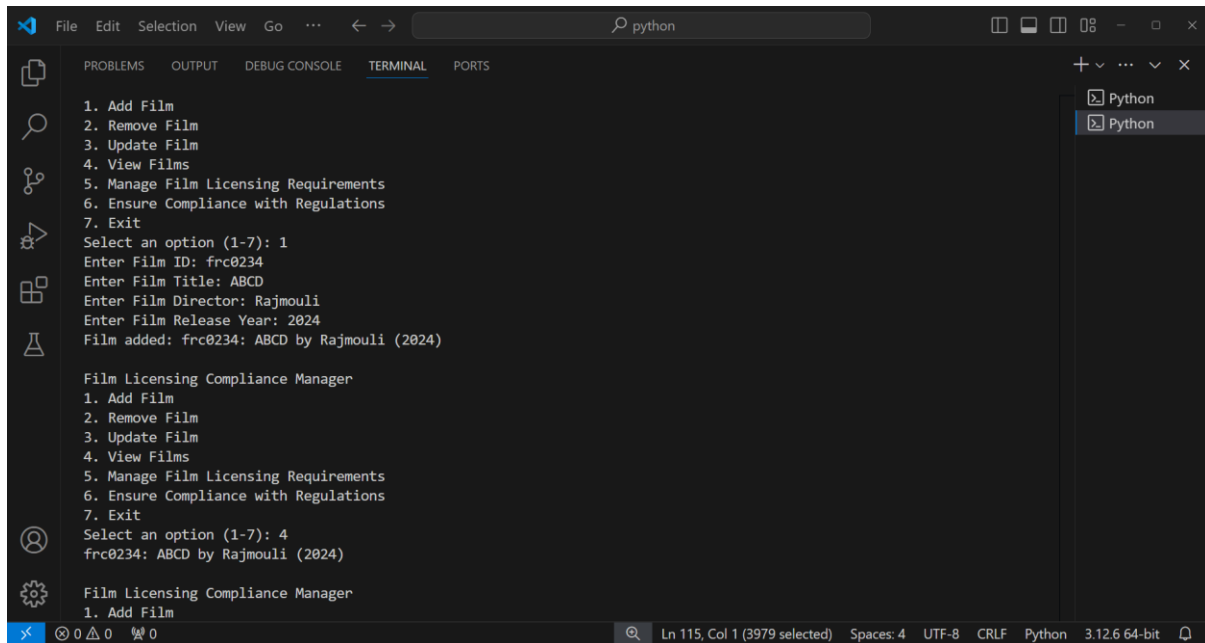
else:

    print("Invalid option, please try again.")


if __name__ == "__main__":

    main()
```

## Output of the program



```
1. Add Film
2. Remove Film
3. Update Film
4. View Films
5. Manage Film Licensing Requirements
6. Ensure Compliance with Regulations
7. Exit
Select an option (1-7): 1
Enter Film ID: frc0234
Enter Film Title: ABCD
Enter Film Director: Rajmouli
Enter Film Release Year: 2024
Film added: frc0234: ABCD by Rajmouli (2024)

Film Licensing Compliance Manager
1. Add Film
2. Remove Film
3. Update Film
4. View Films
5. Manage Film Licensing Requirements
6. Ensure Compliance with Regulations
7. Exit
Select an option (1-7): 4
frc0234: ABCD by Rajmouli (2024)

Film Licensing Compliance Manager
1. Add Film
```

## Objectives of the program:-

### 1. Film Representation:

- To create a Film class that encapsulates all relevant details about individual films, such as title, director, release date, and unique identifiers.

### 2. Licensing Management:

- To develop a LicensingRecord class that handles the details of licensing for each film, including license type, expiration dates, and compliance status.

### 3. CRUD Operations:

- To implement a FilmManager class that provides a comprehensive interface for creating, reading, updating, and deleting (CRUD) both films and their associated licensing\_records. This allows for efficient management of film data and licensing requirements.

#### **4. Compliance Tracking:**

- To enable tracking of compliance with local and international film regulations through the `ensure_compliance_with_regulations` method, which checks whether films meet necessary legal and regulatory standards.

#### **5. Data Integrity:**

- To ensure data integrity by validating film and licensing information, thus preventing the entry of invalid data and ensuring that all necessary licensing requirements are met.

#### **6. User-Friendly Interaction:**

- To provide a clear and structured API through the `FilmManager` class, facilitating easy interaction for users or other systems that may need to manage film and licensing information.

#### **7. Documentation and Extensibility:**

- To create a well-documented codebase that allows for future enhancements and easy onboarding of new developers. The design should be modular to facilitate the addition of features or changes to compliance logic.

### **Methodology:**

#### **1. Requirements Analysis**

- Identify Stakeholders: Determine who will use the system (e.g., filmmakers, distributors, regulatory bodies).
- Gather Requirements: Collect functional and non-functional requirements, focusing on licensing management and compliance needs.

#### **2. System Design**

- **Class Diagram:** Create a class diagram to visualize the relationships between the Film, LicensingRecord, and FilmManager classes.
- **Define Attributes and Methods:**
  - **Film Class:** Attributes for film details (ID, title, director, release date) and methods for initialization and string representation.
  - **LicensingRecord Class:** Attributes for licensing details (film ID, license type, expiration date, compliance status) and methods for compliance checking and representation.
  - **FilmManager Class:** Methods for CRUD operations and compliance checks.

### **3. Implementation**

- **Code Development:** Implement the classes and methods in Python, adhering to Object-Oriented Programming principles:
  - **Encapsulation:** Keep film and licensing data private, exposing only necessary methods.
  - **Inheritance:** If applicable, create subclasses for specific types of films or licensing records.
- **Data Structures:** Use appropriate data structures (lists, dictionaries) to manage collections of films and licensing records.

### **4. Testing**

- **Unit Testing:** Write unit tests for each method in the classes to ensure they work as expected.
- **Integration Testing:** Test interactions between classes, ensuring that CRUD operations and compliance checks function correctly together.
- **User Acceptance Testing:** Engage potential users to validate that the system meets their needs and is user-friendly.

### **5. Documentation**

- Code Documentation: Use docstrings and comments to explain code functionality and usage.
- User Manual: Create a user manual detailing how to use the system, including examples of CRUD operations and compliance checks.

## **6. Deployment**

- Environment Setup: Prepare the environment for deployment, ensuring all dependencies are managed.
- Distribution: Package the application for distribution, considering options such as command-line execution or integration into a larger system.

## **7. Maintenance and Enhancement**

- Feedback Loop: Gather feedback from users to identify areas for improvement.
- Feature Enhancements: Plan for future features, such as a web interface or database integration for persistent storage.

## **Minimum Software Requirements:**

### **Programming Language:**

- Python 3.x

### **Development Environment:**

- IDE/Text Editor (e.g., PyCharm, VSCode, or Jupyter Notebook)

### **Version Control:**

- Git for version control (e.g., GitHub, GitLab for repository hosting).

### **Testing Framework:**

- unittest or pytest for unit testing to ensure code reliability.

## **Minimum Hardware Requirements:**

Processor: Dual-core CPU (Intel i3 or equivalent)

RAM: At least 4 GB (8 GB recommended for smoother performance)

Storage: At least 100 MB of free disk space for the Python environment and code.

## **Conclusions:**

### **1. Modular Design:**

- The separation of concerns through distinct classes (Film, LicensingRecord, and FilmManager) allows for a modular design, making the code easier to understand and maintain. Each class has its own responsibility, enhancing code organization.

### **2. CRUD Functionality:**

- The FilmManager class provides robust CRUD operations for managing films and their licensing records. This functionality ensures that users can easily add, update, delete, and retrieve information, which is critical for effective film management.

### **3. Licensing Management:**

- The manage\_film\_licensing\_requirements method in the FilmManager class allows for efficient tracking of licensing requirements, ensuring that all necessary details are associated with the correct film.

### **4. Compliance Assurance:**

- The ensure\_compliance\_with\_regulations method facilitates checking compliance with local and international regulations, which is vital for avoiding legal issues and maintaining industry standards.

**5. Scalability:** The device is scalable; additional features or more complex compliance checks can be integrated without major overhauls to the existing structure. New classes or methods can be added as the project grows.



## **6. Data Integrity:**

- By encapsulating film and licensing details within their respective classes, the program helps maintain data integrity. This encapsulation ensures that licensing records are always linked to their corresponding films, preventing data mismatches.

## **7. Potential for Expansion:**

- Future enhancements could include database integration for persistent storage, user authentication for access control, or even a user interface (UI) for ease of use. This allows for the system to evolve based on user needs.

## **8. Testing and Reliability:**

- Utilizing a testing framework like unittest or pytest would allow for systematic validation of the code, ensuring reliability and robustness, which is essential for **production-level applications**.

## **9. Real-World Application:**

- This solution is applicable in real-world scenarios where film licensing is crucial, such as film production companies, distribution entities, and regulatory bodies, enabling them to manage licensing processes effectively.

# **Future Enhancements:**

## **\_\_\_1. Database Integration:**

- Use a Database: Transition from in-memory data storage to a relational database (like PostgreSQL or SQLite) to persist film and licensing records. This will allow for better data management, querying, and scalability.
- ORM Framework: Implement an Object-Relational Mapping (ORM) tool like SQLAlchemy to simplify database interactions and enhance data handling.

## **2. User Authentication and Roles:**

- **Access Control:** Implement user authentication to restrict access to sensitive operations. Different roles (e.g., admin, manager) can be defined with specific permissions for managing films and licenses.
- **User Management:** Create a user management system to handle registrations, logins, and role assignments.

### **3. Web Interface:**

- **Web Application:** Develop a web-based user interface using frameworks like Flask or Django, making the system more user-friendly and accessible.
- **Responsive Design:** Ensure the UI is mobile-friendly to accommodate various devices.

#### **? Search and Filter Functionality:**

- **Advanced Search:** Implement robust search capabilities for films and licensing records, allowing users to filter by various criteria (e.g., title, license type, expiration date).
- **Sorting Options:** Provide options to sort records based on different attributes for better user experience.

#### **? Import/Export Functionality:**

- **CSV/Excel Import/Export:** Allow users to import licensing data from spreadsheets and export reports or records for external use.
- **API Integration:** Build APIs to facilitate data exchange with other systems or applications.

#### **? Localization and Internationalization:**

- **Multi-language Support:** Enhance the application to support multiple languages, catering to a global audience.
- **Regional Compliance:** Include customizable compliance checks based on different regions or countries.

#### **? Logging and Auditing:**

- **Audit Trails:** Implement logging to track changes made to film and licensing records for accountability and auditing purposes.
- **Change History:** Allow users to view the history of changes made to records.

## **References:**

### **Python Documentation:**

- The official Python documentation provides comprehensive details about Python syntax, built-in functions, and libraries.
- [Python Official Documentation](#)

### **? Object-Oriented Programming (OOP):**

- **Books:**
  - "Python 3 Object-Oriented Programming" by Dusty Phillips - A practical guide to OOP principles in Python.
- **Online Resources:**
  - Real Python - Object-Oriented Programming