INF224 - Travaux Pratiques

Author

Torres Robles Daniel Felipe

Summary

This project is a two-part endeavor that focuses on distinct aspects of software development, utilizing different programming languages and frameworks:

1. **Backend Development with C++ and Object-Oriented Programming**: This phase is centered around the development of a prototype for a multimedia set-top box software, with an emphasis on object-oriented design principles using C++. It involves the creation of base and derived classes for various multimedia objects such as photos, videos, and films. Additionally, it includes the implementation of testing programs and effective management of multimedia content.
2. **Frontend Development with Java Swing for GUI Creation**: The project's second phase expands into Java, developing a graphical user interface (GUI) with Java Swing. This GUI is engineered to interface with the C++ software developed in the previous stage, enabling the management and playback of multimedia objects. This phase encompasses the creation of a main window with interactive components, the addition of menu bars and toolbars, and the setup of client-server communication for retrieving and playing back multimedia objects.

Project Hierarchy

- `Torres_Robles__Daniel_Felipe/`: Root directory of the project.
  - `cpp/`: This directory houses all the C++ related resources, such as source code files and Makefile.
  - `swing/`: This directory is dedicated to all Java Swing related assets, including source code files and Makefile.

C++ Component - Compilation and Execution

Navigate to the `cpp/` directory and run `make run` to compile and execute the C++ program.

Java Swing Componentv - Compilation and Execution

Navigate to the `swing/` directory and run `make run` to compile and execute the Java Swing program.

Makefile

Both the C++ and Java Swing sections of the project come with a Makefile to simplify the process of compilation and execution. It's important to verify that the required compilers and JDK are installed on your system.

Documentation

The code is thoroughly annotated using Doxygen. For an in-depth understanding of the codebase, please refer to the generated documentation, available in the *Documentation.pdf* file.

Questions C++

4e Etape: Photos et videos

1. The methods mentioned, which are declared in the base class and overridden in the subclasses without any implementation in the base class, are referred to as **pure virtual functions**. These functions are declared by assigning 0 in their declaration within the base class, which makes the base class abstract.

As a result, objects of the base class cannot be directly instantiated, and only objects of its subclasses can be created.

To declare such a method:

```
virtual void play() const = 0;
```

2. The addition of a pure virtual function to a base class prevents the instantiation of objects from that class because it becomes abstract. An abstract class is designed to establish a common interface and shared functionality for its subclasses. However, it's considered incomplete by itself due to the existence of one or more pure virtual functions. Therefore, we can only instantiate objects from those classes that offer concrete implementations for all the pure virtual functions they inherit from their base class.

5e Etape: Traitement uniforme (en utilisant le polymorphisme)

1. The key feature of object-oriented programming that enables a list of photos and videos to be handled uniformly, irrespective of their specific types, is **polymorphism**. Polymorphism allows objects from different classes to be managed through a common interface, mainly by using pointers or references of the base class to access objects of the derived classes. This functionality is vital for enforcing consistent behavior across a collection of objects that may belong to various classes but are linked through inheritance.
2. To leverage polymorphism in C++, it is particularly required to:
   - Declare methods in the base class as virtual, enabling them to be overridden in derived classes. This encompasses methods for both displaying attributes and "playing" the objects.
   - Utilize pointers or references of base class types to refer to objects of the derived classes. This is necessary because C++ needs to determine the size of the objects it's dealing with at compile time. Using pointers or references allows the program to manage objects of varying sizes, which may arise due to different data members in derived classes.
3. In `main.cpp`, the array elements should be pointers (or smart pointers like `std::unique_ptr` or `std::shared_ptr`) to the base class, not base class objects. This setup allows the array to hold references to both Photo and Video objects, interacting with them through their shared base interface while executing the unique behavior of each derived class.
   - This contrasts with Java, where all non-primitive types are inherently managed via references. In Java, you can directly store derived class objects in an array or collection of the base class type, without the explicit need for pointers.
   - The use of pointers in C++ instead of objects is tied to the slicing problem. If derived class objects were directly assigned to elements of a base class array, any data or functions specific to the derived classes would be "sliced" off, leaving only the base class part of the objects. Using pointers circumvents slicing by enabling the program to access the complete derived class objects.

7e étape. Destruction et copie des objets

1. **Which classes need modifications to prevent memory leaks when objects are destroyed?**
   - Any class that utilizes the 'new' keyword to allocate dynamic memory must also use 'delete' to release that memory, in order to avoid memory leaks. Specifically, the Film class allocates memory for an array to store the durations of chapters, hence it requires a destructor to free up that allocated memory.
2. **What is the problem with copying objects that have instance variables which are pointers, and what are the solutions?**
   - The issue with copying such objects arises from the shallow copy behavior of the default copy constructor and assignment operator provided by C++. A shallow copy replicates the pointer values but not the data they point to, leading to scenarios where multiple objects point to the

same memory location. This can result in undefined behavior when one object alters the data or deallocates the memory, impacting all objects sharing that pointer.

- **Possible solutions are:**
  - Implementing a deep copy, where both the pointers and the data they point to are duplicated. This ensures that each object has its own copy of the data.
  - Using smart pointers (e.g., `std::unique_ptr`, `std::shared_ptr`) that automatically manage memory and can correctly handle copying semantics. However, std::unique_ptr requires explicit handling for copy operations as it exclusively owns the resource.

8e étape. Créer des groupes

1. **Why use a list of object pointers?**
   - Employing a list of pointers, particularly pointers to the base class, permits the list to accommodate objects from any derived class, thus facilitating polymorphic behavior. This is essential as the actual objects may vary in size and behavior, and our goal is to interact with them through a unified interface provided by the base class. This method is akin to Java's practice of using references for all objects, where Java inherently manages objects polymorphically without the explicit need for pointers.
2. **Why objects are not destroyed when a group is destroyed?**
   - Given that an object can be part of several groups, eliminating an object when a group is deleted could result in undefined behavior when other groups attempt to access the removed object. This is mitigated by ensuring that the responsibility for the objects' lifetimes is managed outside the group, usually by the code that creates the objects. This strategy necessitates meticulous memory management to avoid memory leaks, as it's crucial to delete objects when they are no longer in use.

10e étape. Gestion cohérente des données

1. **How to ensure objects are only created through the MediaManager class to maintain database integrity?**
   - Set the constructors of the multimedia and group classes to private or protected, which prevents them from being directly instantiated with 'new' by external code. Subsequently, designate MediaManager as a friend class of these classes, which grants MediaManager the ability to access their constructors, while other classes are restricted from doing so.

Questions Java

1ere Etape: Fenêtre principale et quelques interacteurs

1. When we launch the program and interact with it by clicking the buttons to add text and resizing the window:
   1. **Text Insertion:** Each press of a button appends a particular line of text to the `JTextArea`, as per the programming. The two buttons insert their respective lines of text, showcasing how the actions of buttons can influence other components within the UI.
   2. **Window Adjustment and Text Area Functionality:** In the absence of a `JScrollPane`, the `JTextArea` may not effectively manage text overflow. When we input more text than the visible area can accommodate, we may not be able to view all the text via scrolling. This is due to the fact that `JTextArea` does not inherently possess scrolling functionality.
   3. **The Requirement for JScrollPane:** To ensure the `JTextArea` is completely functional, particularly when the volume of text surpasses the visible area, it's essential to integrate it into a `JScrollPane`. The `JScrollPane` offers a scrollable view of the `JTextArea`, enabling navigation

through all the inputted text irrespective of its quantity. This resolves usability issues that become evident through interaction and window resizing.