Husniddin Ramazanov 200104004802 PA6

Introduction

This project implements a simple media management system in C++ using the Observer design pattern. The goal is to store various types of media (audio, video, image, text) within a single central container (Dataset) and have "observer" classes (Player and Viewer) automatically update themselves when the dataset changes. Each media item inherits from a common abstract base (MediaObject) and then implements one or more interface classes to indicate whether it is visual, non-visual, playable, or non-playable. The result is a modular, extensible framework in which any addition or removal of media in the dataset is automatically propagated to all registered observers.

Design and Implementation

1. Core Abstractions

- MediaObject (abstract base class)
    - Holds two protected members: std::string name and std::string info.
    - Declares a pure virtual method virtual void info_display() const = 0; so that each concrete subclass must implement its own way of printing its identifying information.
    - Provides trivial getters (getName(), getInfo()) and a virtual destructor.
- Interface Classes (pure-virtual abstract classes)

1. Visual

- Declares virtual void display() const = 0; for any "visual" media (image or video).

2. Non_Visual

- Declares virtual void process() const = 0; for non-visual media (audio or text).

3. Playable

- Holds a protected member std::string duration and declares virtual void play() const = 0;. Any subclass is required to define how playback occurs.
- Provides getDuration() so subclasses can report duration.

4. Non_playable

- Declares virtual void view() const = 0; for non-playable media (image or text), indicating how it is "viewed."

2. Concrete Media Classes

Each of these inherits from MediaObject and one or two interfaces, depending on its category:

- **Audio**
    - Inherits MediaObject and implements Non_Visual + Playable.
    - Constructor: Audio(const std::string &name, const std::string &dur, const std::string &info) initializes MediaObject(name, info) and Playable(dur).
    - Implements:
        - info_display() → prints Audio: <name>, Duration: <duration>, Info: <info>.
        - process() → prints Processing audio: <name>.
        - play() → prints Playing audio: <name> (Duration: <duration>).
- **Video**
    - Inherits MediaObject + Visual + Playable.
    - Constructor: Video(const std::string &name, const std::string &dur, const std::string &info) calls MediaObject(name, info) and Playable(dur).
    - Implements:
        - info_display() → prints Video: <name>, Duration: <duration>, Info: <info>.
        - display() → prints Displaying video: <name>.

        - play() → prints Playing video: <name> (Duration: <duration>).

  - **Image**
    - Inherits MediaObject + Visual + Non_playable.
    - Constructor: Image(const std::string &name, const std::string &dim, const std::string &info) calls MediaObject(name, info) and stores the dimension string.
    - Implements:
        - info_display() → prints Image: <name>, Dimension: <dimension>, Info: <info>.
        - display() → prints Displaying image: <name>.
        - view() → prints Viewing image: <name>.
        - getDimension() → returns the dimension string.
  - **Text**
    - Inherits MediaObject + Non_Visual + Non_playable.
    - Constructor: Text(const std::string &name, const std::string &info) calls MediaObject(name, info).
    - Implements:
        - info_display() → prints Text: <name>, Info: <info>.
        - process() → prints Processing text: <name>.
        - view() → prints Viewing text: <name>.

3. **Observer Pattern Components**
    - Observer (interface)
        - Declares a single method virtual void update() = 0;. Any class that wants to be notified of dataset changes implements this interface.
    - Dataset (subject class)
        - Maintains two private vectors:

1. std::vector<MediaObject*> objects; — stores pointers to all media items (audio, video, image, text).

2. std::vector<Observer*> observers; — stores pointers to registered observers (Player and Viewer).

- Implements:
  - ~Dataset() destructor: iterates through objects and deletes each MediaObject*.
  - void add(MediaObject* obj): pushes obj into objects, prints Added <name> to dataset., then calls notifyObservers().
  - void remove(MediaObject* obj): searches for obj in objects; if found, prints Removed <name> from dataset., erases it, and calls notifyObservers().
  - void registerObserver(Observer* obs): pushes obs onto observers and immediately calls obs->update() so the new observer is in sync.
  - void removeObserver(Observer* obs): searches and erases obs from observers.
  - void notifyObservers(): iterates over all registered observers and calls obs->update().
  - std::vector<Playable*> getPlayableObjects() const: dynamic_casts each MediaObject* in objects to Playable*; if successful, adds it to the returned vector.
  - std::vector<Non_playable*> getNonPlayableObjects() const: similarly, dynamic_casts to Non_playable*.

4. Concrete Observer Classes

○ Player

- Implements Observer.
- Private members:
  - std::vector<Playable*> playlist; — current list of playable media.
  - int currentIndex; — index of the currently “playing” item.

        - Dataset* dataset; — pointer to the observed Dataset.
    - Constructor: initializes currentIndex = -1 (no item yet) and dataset = nullptr.
    - Methods:

1. void setDataset(Dataset* ds): stores the dataset pointer so that update() can query it later.
2. void update() override: called whenever Dataset changes.
        - Calls dataset->getPlayableObjects() and stores the result in playlist.
        - Adjusts currentIndex if it is out of range (e.g., if the playlist shrank). If playlist is non-empty and currentIndex == -1, sets it to 0.
3. void show_list() const: prints a numbered list of all items in playlist. If i == currentIndex, appends [CURRENTLY PLAYING]. If playlist is empty, prints Playlist is empty.
4. Playable* currently_playing(): returns playlist[currentIndex]. If playlist is empty or currentIndex == -1, throws std::runtime_error("No item currently playing - playlist is empty!").
5. void next(const std::string &type): advances currentIndex in a circular fashion until it finds an element whose dynamic type matches type—either “audio” (dynamic_cast<Audio*>) or “video” (dynamic_cast<Video*>). If type is neither “audio” nor “video,” it returns the next element regardless of subtype. If no match is found after a full cycle, prints No <type> found in playlist!
6. void previous(const std::string &type): same logic as next(), but moves backward in the playlist.
7. bool matchesType(Playable* obj, const std::string &type) const: helper that checks whether a given Playable* is an Audio* (if type == "audio") or Video* (if type == "video"), or returns true for any type if type is empty or unrecognized.

    - Viewer

        - Implements Observer.
        - Private members:
            - std::vector<Non_playable*> viewlist; — current list of non-playable media.
            - int currentIndex; — index of the currently “viewing” item.
            - Dataset* dataset; — pointer to the observed Dataset.
        - Constructor: initializes currentIndex = -1 and dataset = nullptr.
        - Methods:

1. void setDataset(Dataset* ds): store the dataset pointer.
2. void update() override: calls dataset->getNonPlayableObjects() and stores in viewlist. Adjusts currentIndex similarly to Player::update().
3. void show_list() const: prints viewlist in the same numbered style, marking currentIndex with [CURRENTLY VIEWING]. If empty, prints View list is empty.
4. Non_playable* currently_viewing(): returns viewlist[currentIndex] or throws std::runtime_error("No item currently viewing - view list is empty!") if empty.
5. void next(const std::string &type): advances currentIndex until it finds an element matching type—either “text” (dynamic_cast<Text*>) or “image” (dynamic_cast<Image*>). If no match is found after a full cycle, prints No <type> found in view list!.
6. void previous(const std::string &type): same logic in reverse.
7. bool matchesType(Non_playable* obj, const std::string &type) const: checks dynamic type against “text” or “image,” returning true if type is empty or unrecognized.

5. File Organization
    - MediaObject.h / MediaObject.cpp

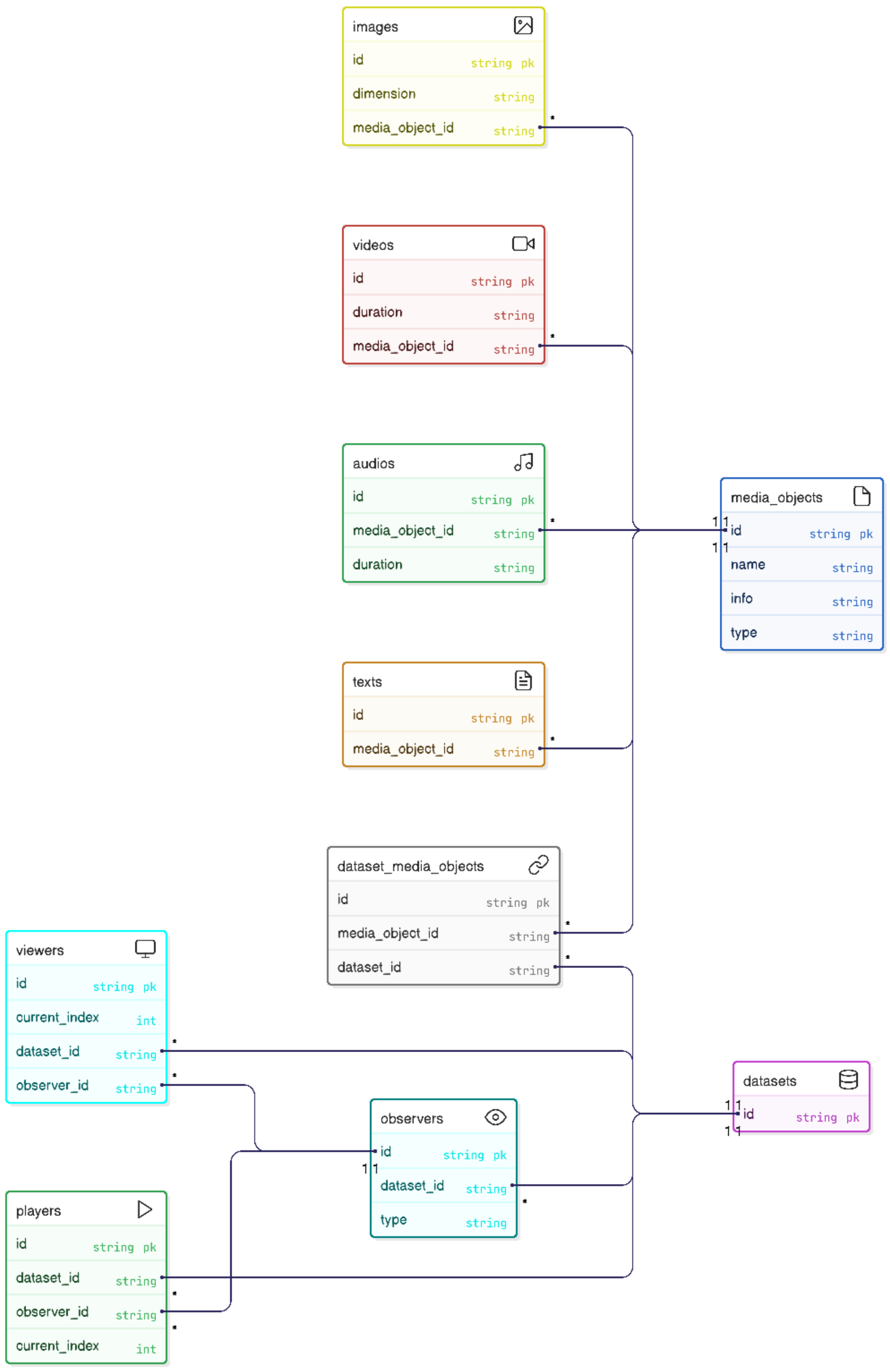
    - Declares the abstract base. Implements the constructor, destructor, and getters.
  - Visual.h, NonVisual.h, Playable.h, NonPlayable.h
    - Define the four pure-virtual interface classes with exactly one method each (display(), process(), play(), view()) plus appropriate destructors.
    - Playable.h also stores a string duration and provides a getter.
  - Audio.h / Audio.cpp
    - Declares and implements the Audio class.
  - Video.h / Video.cpp
    - Declares and implements the Video class.
  - Image.h / Image.cpp
    - Declares and implements the Image class.
  - Text.h / Text.cpp
    - Declares and implements the Text class.
  - Observer.h
    - Declares the Observer interface.
  - Dataset.h / Dataset.cpp
    - Declares and implements the Dataset class, including add(), remove(), observer management, and the two query methods (getPlayableObjects(), getNonPlayableObjects()).
  - Player.h / Player.cpp
    - Declares and implements the Player observer.
  - Viewer.h / Viewer.cpp
    - Declares and implements the Viewer observer.
  - main.cpp

- Contains a concise example that uses all components—registers two players and two viewers, adds media, demonstrates navigation, removal, observer removal, and final cleanup.

Conclusion

This report has described a C++ implementation of a media management system using the Observer pattern. We defined a common abstract base class (MediaObject) and interface classes (Visual, Non_Visual, Playable, Non_playable) to categorize media items. We then implemented concrete subclasses (Audio, Video, Image, Text) that inherit the appropriate interfaces. The Dataset class serves as the single source of truth, maintaining a list of all MediaObject* instances and a list of registered observers (Player and Viewer). Whenever the dataset changes, it calls notifyObservers(), so each Player automatically updates its playlist of playable items and each Viewer updates its viewlist of non-playable items.

UML DIAGRAM


images
id string pk
dimension string
media_object_id string
videos
id string pk
duration string
media_object_id string
audios
id string pk
media_object_id string
duration string
texts
id string pk
media_object_id string
media_objects
id string pk
name string
info string
type string
dataset_media_objects
id string pk
media_object_id string
dataset_id string
viewers
id string pk
current_index int
dataset_id string
observer_id string
observers
id string pk
dataset_id string
type string
datasets
id string pk
players
id string pk
dataset_id string
observer_id string
current_index int
*
1
eraser