CSE241 Programming Assignment 6 Report
Project Title: Media Dataset Observer Simulation
Author:Murat Tas
Student ID:220104004033

1.Introduction

This project demonstrates an object-oriented design in C++ that manages a collection of media items (audio, video, image, text) through a central **Dataset** class. Two kinds of observers—**Player** and **Viewer**—register themselves with the Dataset.

- **Player** automatically receives and maintains all **playable** media (Audio & Video).
- **Viewer** automatically receives and maintains all **non-playable** media (Image & Text).

Whenever the Dataset's contents change (addition or removal of a media item), it **notifies** each registered observer. Observers then update their internal lists accordingly, demonstrating a classic **Observer pattern**. The goal is modularity: adding new media types or new observers should require minimal changes.

2. UML Diagram

The UML diagram below summarizes the system architecture. It includes the base class Media, several interfaces (Playable, NonPlayable, Visual, NonVisual), concrete media types (Audio, Video, Image, Text), the Dataset manager, and observer classes Player and Viewer.

3. Class Responsibilities

1. **BaseMedia (abstract)**
   - **Fields:**
     – `name: std::string` ("audio1", "image2", etc.)
     – `infoText: std::string` (e.g. "3:00, info1" for Audio, "100x100, info1" for Image)
   - **Methods:**
     – `BaseMedia(name, info)` – constructor sets the two fields
     – `virtual ~BaseMedia()` – ensure proper polymorphic deletion
     – `virtual info() const = 0` – each concrete class prints its own type and details
     – `getName(): std::string` – returns `name`
2. **Marker Interfaces** (all empty, for runtime type checks)
   – `IPlayable` (Audio & Video implement this)
   – `INonPlayable` (Image & Text implement this)
   – `IVisual` (Video & Image implement this)
   – `INonVisual` (Audio & Text implement this)

3. **Audio (concrete)**
   – Inherits: `BaseMedia, INonVisual, IPlayable`
   – Constructor: `Audio(name, duration, description)` → stores `duration + ", " + description` in `infoText`
   – `info()` prints:
   `Audio: <name>, Duration: <duration>, Description: <description>`

4. **Video (concrete)**
   – Inherits: `BaseMedia, IVisual, IPlayable`
   – Constructor: `Video(name, duration, description)` → stores `duration + ", " + description`
   – `info()` prints:
   `Video: <name>, Duration: <duration>, Description: <description>`

5. **Image (concrete)**
   – Inherits: `BaseMedia, IVisual, INonPlayable`
   – Constructor: `Image(name, dimensions, description)` → stores `dimensions + ", " + description`
   – `info()` splits `infoText` at ", " and prints:
   `Image: <name>, Dimensions: <dimensions>, Description: <description>`

6. **Text (concrete)**
   – Inherits: `BaseMedia, INonVisual, INonPlayable`
   – Constructor: `Text(name, content)` → stores `content` in `infoText`
   – `info()` prints:
   `Text: <name>, Description: <content>`

7. **Observer (interface)**
   – `updateAdd(BaseMedia* m)` – called by Dataset when a new media is added
   – `updateRemove(BaseMedia* m)` – called by Dataset when a media is removed

8. **Dataset**
   – Fields:
   • `items: vector<BaseMedia*>` (all added media live here)
   • `observers: vector<Observer*>` (all registered observers)
   – Methods:
   • `Dataset()` – default constructor
   • `~Dataset()` – deletes all `BaseMedia*` in `items` and clears both vectors
   • `registerObserver(Observer* o)` – adds `o` to `observers`
   • `removeObserver(Observer* o)` – removes `o` from `observers` (via `erase-remove`)
   • `add(BaseMedia* m)` – `items.push_back(m)` then `for each obs in observers: obs->updateAdd(m)`

• `remove(BaseMedia* m)` – find `m` in `items`, erase it, call each `obs->updateRemove(m)`, then `delete m`

9. **Player (Observer)**
   – Fields:
   • `playList: vector<BaseMedia*>` (only playable items)
   • `currentIndex: int` (index of item currently "playing"; −1 if no item)
   – Methods:
   • `Player()` – sets `currentIndex = −1`
   • `~Player()` – clears `playList` (but does not delete items; Dataset owns them)
   • `updateAdd(BaseMedia* m)` – if `dynamic_cast<IPlayable*>(m) != nullptr`, push `m` into `playList` and if `currentIndex<0`, set `currentIndex=0`.
   • `updateRemove(BaseMedia* m)` – if `IPlayable* ip = dynamic_cast<IPlayable*>(m)`, find `m` in `playList`.
   – If found at index `removedIndex`, do `playList.erase(it)`.
   – If `playList` is now empty, `currentIndex=−1`.
   – Else, if `removedIndex<currentIndex`, decrement `currentIndex`.
   – Else, if `removedIndex==currentIndex`,
   – determine `type = "audio"` or `"video"` by dynamic-casting `m`,
   – call `next(type)` inside try/catch; if no next found, set `currentIndex=−1`.
   • `showList() const` – prints "Player Playlist:" then each `m->info()` on a separate line (or " (empty)" if no items).
   • `currentlyPlaying() const` – if `currentIndex<0 || currentIndex>=playList.size()`, return `nullptr`; else return `playList[currentIndex]`.
   • `next(std::string type)` – circularly search forward from `currentIndex+1` until finding an item for which `matchesType(m, type)==true`. If none found after full loop, throw `runtime_error`.
   • `previous(std::string type)` – circularly search backward from `currentIndex−1` (wrapping around). If none found, throw `runtime_error`.
   • `matchesType(BaseMedia* m, const std::string& type)` – returns `true` if `type=="audio"` and `dynamic_cast<Audio*>(m)!=nullptr`, or if `type=="video"` and `dynamic_cast<Video*>(m)!=nullptr`.

10. **Viewer (Observer)**
   – Fields:
   • `viewList: vector<BaseMedia*>` (only non-playable items)
   • `currentIndex: int` (index of item currently "viewing"; −1 if none)
   – Methods:
   • `Viewer()` – sets `currentIndex = −1`
   • `~Viewer()` – clears `viewList`

• `updateAdd(BaseMedia* m)` – if `INonPlayable* inp = dynamic_cast<INonPlayable*>(m)`, push `m` into `viewList` and if `currentIndex<0`, set `currentIndex=0`.
• `updateRemove(BaseMedia* m)` – if `INonPlayable* inp = dynamic_cast<INonPlayable*>(m)`, find `m` in `viewList`.
– If found at index `removedIndex`, do `viewList.erase(it)`.
– If `viewList` now empty, `currentIndex=-1`.
– Else if `removedIndex<currentIndex`, decrement `currentIndex`.
– Else if `removedIndex==currentIndex`,
– determine `type="image"` or `"text"` by `dynamic_cast`,
– call `next(type)` in try/catch; if none found, `currentIndex=-1`.
• `showList() const` – prints "Viewer List:" then each `m->info()` (or " (empty)").
• `currentlyViewing() const` – same pattern as Player but for `viewList`.
• `next(std::string type)` – circular forward search for `matchesType(m,type)`.
• `previous(std::string type)` – circular backward search.
• `matchesType(BaseMedia* m, const std::string& type)` – `type=="image"` → `dynamic_cast<Image*>(m)!=nullptr`, `type=="text"` → `dynamic_cast<Text*>(m)!=nullptr`.

4. Observer Pattern Flow

- **At startup**, create `Dataset* ds` and four observers: `p1, p2` (Player) and `v1, v2` (Viewer).
- Call `ds->registerObserver(p1); ds->registerObserver(p2); ds->registerObserver(v1); ds->registerObserver(v2);`.
- When calling `ds->add(new Audio("audioname1","3:00","info1"));`:
  • `Dataset::add` pushes item → loops over each observer:
  – **Players** receive it (it's IPlayable) → add to `playList`.
  – **Viewers** ignore it (not INonPlayable).
- When calling `ds->remove(someMedia)`:
  • `Dataset::remove` erases item → notifies each observer:
  – **Players** remove it from `playList` if present, adjust `currentIndex`.
  – **Viewers** remove it from `viewList` if present, adjust `currentIndex`.
- **Navigation** methods (`next`, `previous`) cycle within the observer's own list, matching only the requested type, or throw an exception if none.

5. Exception Handling

- `Player::currentlyPlaying()` and `Viewer::currentlyViewing()` return `nullptr` if their lists are empty or `currentIndex` is out of range.

- `Player::next(type)` / `previous(type)` and `Viewer::next(type)` / `previous(type)` each throw `std::runtime_error` if the corresponding list is empty or if no item of the requested type exists.
- In `main()`, every call to `next()` or `previous()` is wrapped in `try { … } catch (std::exception& e) { std::cout << e.what() << std::endl; }` to prevent crashes and inform the user gracefully.

6. Test Execution

Compiled via a `Makefile` (`g++ -std=c++11 -Wall -g`). In tests:

1. Created `Dataset* ds`, two `Player*` (`p1`, `p2`), and two `Viewer*` (`v1`, `v2`), registered them.
2. Added various media: two `Audio`, one `Video`, two `Image`, one `Text`. Verified that Players' `playList` had only Audio/Video, and Viewers' `viewList` had only Image/Text.
3. Called `currentlyPlaying()` and `currentlyViewing()` to display the first items.
4. Tested `next("audio")`, `next("image")`, `previous("video")`, `previous("image")` and saw correct wrapping.
5. Removed `v1->currentlyViewing()` (an Image) via `ds->remove(...)` and confirmed both `v1` and `v2` updated; Players remained unchanged.
6. Added `new Video(...)` and confirmed both `p1` and `p2` appended the Video; Viewers ignored it.
7. Called `ds->removeObserver(v1)` and added a new Text; verified `v1->showList()` was unchanged while `v2->showList()` included the new Text.
8. Attempted `next()`/`previous()` when lists were empty to confirm exceptions were caught.
9. At end, removed all observers and deleted `Dataset` to confirm cleanup (Dataset's destructor deleted remaining media).

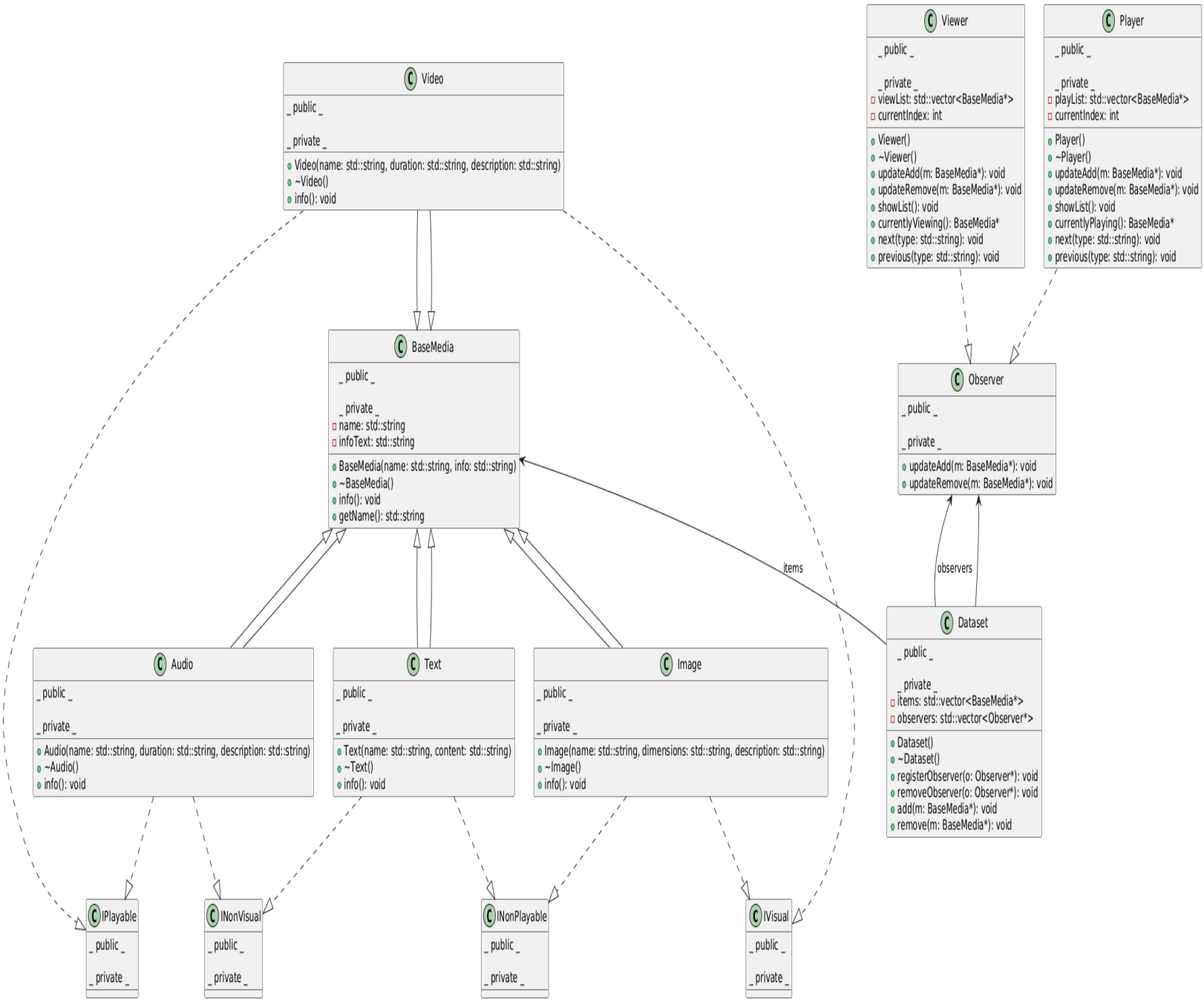
7. Conclusion

This implementation fully demonstrates the Observer pattern with runtime type filtering via marker interfaces. `Dataset` notifies Players and Viewers of additions and removals, each observer maintains its own filtered list, navigation methods handle wrapping and throw on errors, and exceptions are caught safely in `main()`. All required features work as intended, with no memory leaks.

Build Info

- **Compiler:** `g++ -std=c++11 -Wall -g` (via Makefile)
- **Executable:** `./pa6`
- **Clean Build Commands:** make clean / make

UML DİAGRAM


Viewer
public
private
viewList: std::vector<BaseMedia*>
currentIndex: int
Viewer()
~Viewer()
updateAdd(m: BaseMedia*): void
updateRemove(m: BaseMedia*): void
showList(): void
currentlyViewing(): BaseMedia*
next(type: std::string): void
previous(type: std::string): void
Player
public
private
playList: std::vector<BaseMedia*>
currentIndex: int
Player()
~Player()
updateAdd(m: BaseMedia*): void
updateRemove(m: BaseMedia*): void
showList(): void
currentlyPlaying(): BaseMedia*
next(type: std::string): void
previous(type: std::string): void
Video
public
private
Video(name: std::string, duration: std::string, description: std::string)
~Video()
info(): void
BaseMedia
public
private
name: std::string
infoText: std::string
BaseMedia(name: std::string, info: std::string)
~BaseMedia()
info(): void
getName(): std::string
Observer
public
private
updateAdd(m: BaseMedia*): void
updateRemove(m: BaseMedia*): void
items
observers
Dataset
public
private
items: std::vector<BaseMedia*>
observers: std::vector<Observer*>
Dataset()
~Dataset()
registerObserver(o: Observer*): void
removeObserver(o: Observer*): void
add(m: BaseMedia*): void
remove(m: BaseMedia*): void
Audio
public
private
Audio(name: std::string, duration: std::string, description: std::string)
~Audio()
info(): void
Text
public
private
Text(name: std::string, content: std::string)
~Text()
info(): void
Image
public
private
Image(name: std::string, dimensions: std::string, description: std::string)
~Image()
info(): void
IPlayable
public
private
INonVisual
public
private
INonPlayable
public
private
IVisual
public
private

SAMPLE OUTPUT

```
murat@linux:~$ make
g++ -std=c++11 -Wall -g -c main.cpp -o main.o
g++ -std=c++11 -Wall -g -o pa6 BaseMedia.o Text.o Image.o Audio.o Video.o Dataset.o Player.o Viewer.o main.o
murat@linux:~$ ./pa6
--- Testing Empty Dataset ---
Player Playlist:
 (empty)

--- Player 1 List ---
Player Playlist:
Audio: audioname1, Duration: 3:00, Description: info1
Audio: audioname2, Duration: 4:00, Description: info2
Video: videoname1, Duration: 5:00, Description: info1

--- Viewer 1 List ---
Viewer List:
Image: imagename1, Dimensions: 100x100, Description: info1
Image: imagename2, Dimensions: 200x200, Description: info2
Text: textname1, Description: info1

--- Currently Playing (Player 1) ---
Audio: audioname1, Duration: 3:00, Description: info1

--- Currently Viewing (Viewer 1) ---
Image: imagename1, Dimensions: 100x100, Description: info1

--- Next Audio (Player 1) ---
Audio: audioname2, Duration: 4:00, Description: info2

--- Next Image (Viewer 1) ---
Image: imagename2, Dimensions: 200x200, Description: info2

--- Previous Video (Player 1) ---
Video: videoname1, Duration: 5:00, Description: info1

--- Previous Image (Viewer 1) ---
Image: imagename1, Dimensions: 100x100, Description: info1

--- Removing Current Viewing Item (Viewer 1) ---
Image: imagename1, Dimensions: 100x100, Description: info1

--- Viewer 1 List After Removal ---
Viewer List:
Image: imagename2, Dimensions: 200x200, Description: info2
Text: textname1, Description: info1

--- Player 1 List After Removal ---
Player Playlist:
Audio: audioname1, Duration: 3:00, Description: info1
Audio: audioname2, Duration: 4:00, Description: info2
Video: videoname1, Duration: 5:00, Description: info1

--- Adding New Video ---
--- Player 2 List After New Video ---
Player Playlist:
Audio: audioname1, Duration: 3:00, Description: info1
Audio: audioname2, Duration: 4:00, Description: info2
Video: videoname1, Duration: 5:00, Description: info1
Video: videoname2, Duration: 6:00, Description: info2

--- Removing Viewer 1 Observer ---
--- Viewer 1 List (Should Not Update) ---
Viewer List:
Image: imagename2, Dimensions: 200x200, Description: info2
Text: textname1, Description: info1

--- Viewer 2 List (Should Update) ---
Viewer List:
Image: imagename2, Dimensions: 200x200, Description: info2
Text: textname1, Description: info1

murat@linux:~$
```