

# Photo Gallery Application - Project Report

## 1. Introduction

**Project Title:** Smart Photo Gallery Application

**Team Members:** Nurdan Z. , Elkham M. , Yernur S.

### **Project Overview:**

The Smart Photo Gallery application is a Java-based software designed to manage multimedia content efficiently. It allows users to capture photos and videos, apply filters, store images, sort them using multiple criteria, and receive real-time notifications.

### **Objective:**

The primary objective of this project is to **implement at least six design patterns** learned in the course and integrate them into a cohesive application. The project demonstrates the practical use of **Facade, Observer, Singleton, Strategy, Factory, and Decorator patterns**.

### **Key Features:**

- Simplified camera interface using **Facade**.
- Real-time notifications using **Observer**.
- Centralized image repository using **Singleton**.
- Dynamic image sorting using **Strategy**.
- Filter creation using **Factory**.
- Image filter stacking using **Decorator**.

## 2. Body

### 2.1 Architecture and Design Patterns

#### 2.1.1 *Facade Pattern*

- **Class:** CameraFacade
- **Description:** Provides a simplified interface to capture photos and record videos, integrating the BasicCamera with subsystems (FocusSystem, FlashSystem, StorageSystem).

#### **Code Snippet:**

```
CameraFacade camera = new CameraFacade();  
camera.takePhoto("holiday.jpg");  
camera.recordMovie();
```

#### 2.1.2 *Observer Pattern*

- **Classes:** ImageGallery (Subject), UserObserver (Observer)
- **Description:** Users subscribe to image gallery updates. When a new image is added, all observers are notified automatically.

#### **Code Snippet:**

```
gallery.addObserver(new UserObserver("Alice"));  
gallery.addNewImage("sunset.jpg"); // triggers notifications
```

### 2.1.3 *Singleton Pattern*

- **Class:** AppManager
- **Description:** Ensures a single instance of the image repository exists, managing all images centrally.

#### **Code Snippet:**

```
AppManager manager = AppManager.getInstance();  
manager.addImage(new GalleryImage("holiday.jpg", LocalDate.now(), 5));
```

### 2.1.4 *Strategy Pattern*

- **Classes:** ImageSorter, SortByDate, SortByName, SortByRating
- **Description:** Sorting behavior is interchangeable at runtime, allowing images to be sorted by multiple criteria.

#### **Code Snippet:**

```
ImageSorter sorter = new ImageSorter();  
sorter.setStrategy(new SortByRating());  
sorter.sortImages(manager.getImages());
```

### 2.1.5 *Factory Pattern*

- **Class:** FilterFactory
- **Description:** Creates filter objects dynamically based on a filter type (GRAYSCALE, SEPIA).

#### **Code Snippet:**

```
DisplayableImage gray = FilterFactory.create(FilterType.GRAYSCALE, originalImage);
```

### 2.1.C Decorator Pattern

- **Classes:** ImageFilterDecorator, GrayscaleFilter, SepiaFilter
- **Description:** Allows dynamic addition of filter behaviors to images without modifying the original image class.

#### Code Snippet:

```
DisplayableImage sepiaOnGray = new SepiaFilter(grayImage);  
sepiaOnGray.display();
```

### 2.1.7 Visitor Pattern

**Classes:** ImageVisitor, AbstractImageVisitor, AverageRatingVisitor, BatchExportVisitor, GalleryImage

**Description:** Allows operations to be performed on GalleryImage objects without exposing their internal state. Supports calculating average rating, batch exporting, or future operations.

#### Code Snippet - Average Rating Visitor:

```
AverageRatingVisitor avgVisitor = new AverageRatingVisitor();  
for (GalleryImage img : manager.getImages()) {  
    img.accept(avgVisitor);  
}  
System.out.println("Average Rating: " + avgVisitor.getAverage());
```

#### Code Snippet - Batch Export Visitor:

```
BatchExportVisitor exportVisitor = new BatchExportVisitor();  
for (GalleryImage img : manager.getImages()) {
```

```

img.accept(exportVisitor);
}

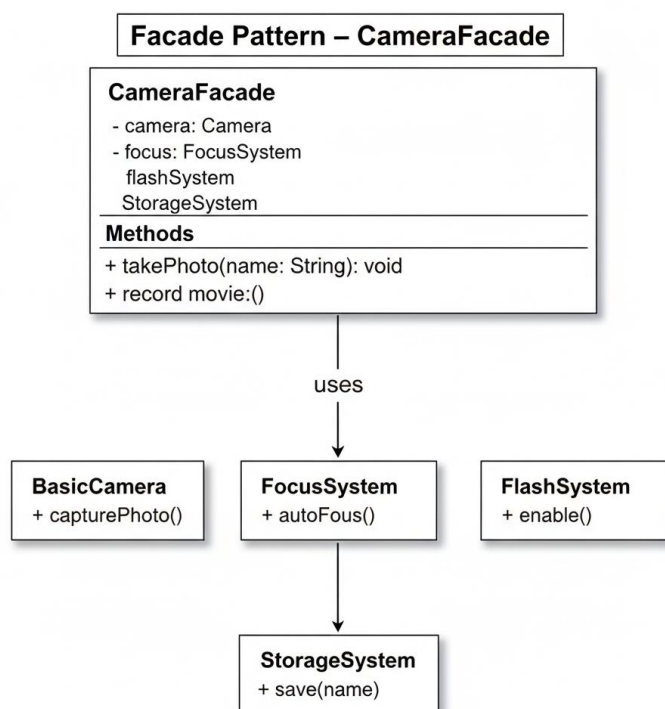
```

### Integration with Other Patterns:

- Works seamlessly with **Singleton** AppManager to access all images.
- Independent of **Observer**, **Facade**, **Strategy**, **Decorator**, and **Factory**, but can be applied after sorting or filtering.
- Maintains **encapsulation**, demonstrating clean separation of concerns.

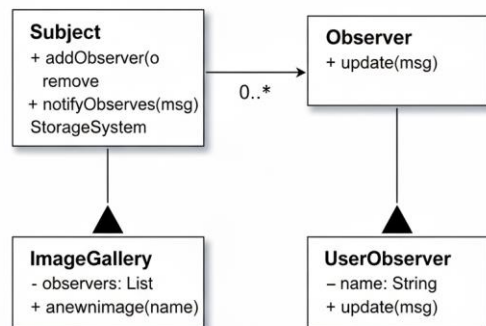
## 2.2 UML Diagrams

### 1. Facade Pattern - CameraFacade



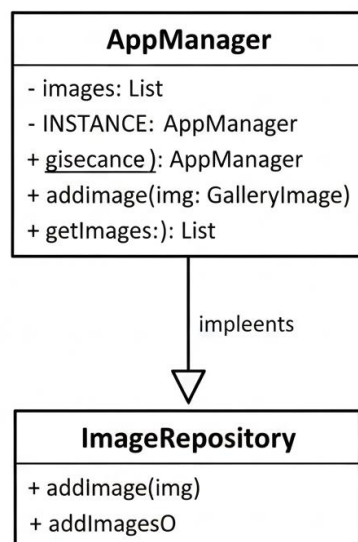
### 2. Observer Pattern - ImageGallery / UserObserver

### Observer Pattern – ImageGallery / UserObserver

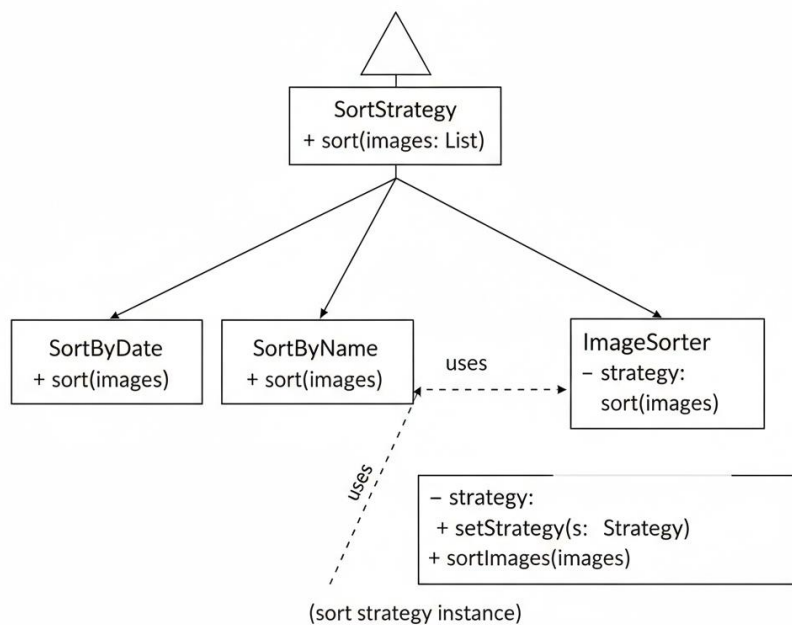


### 3. Singleton Pattern - AppManager

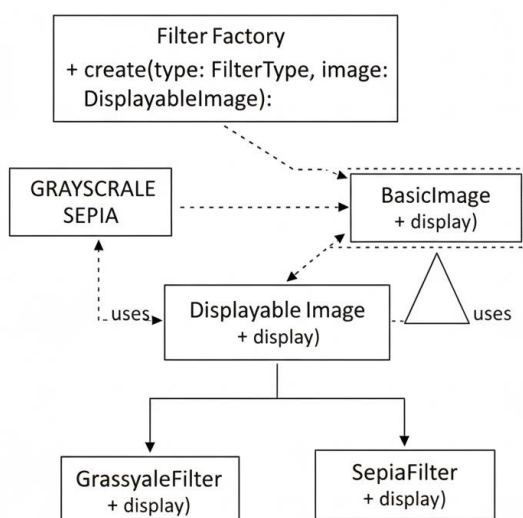
#### Singleton Pattern – AppManager



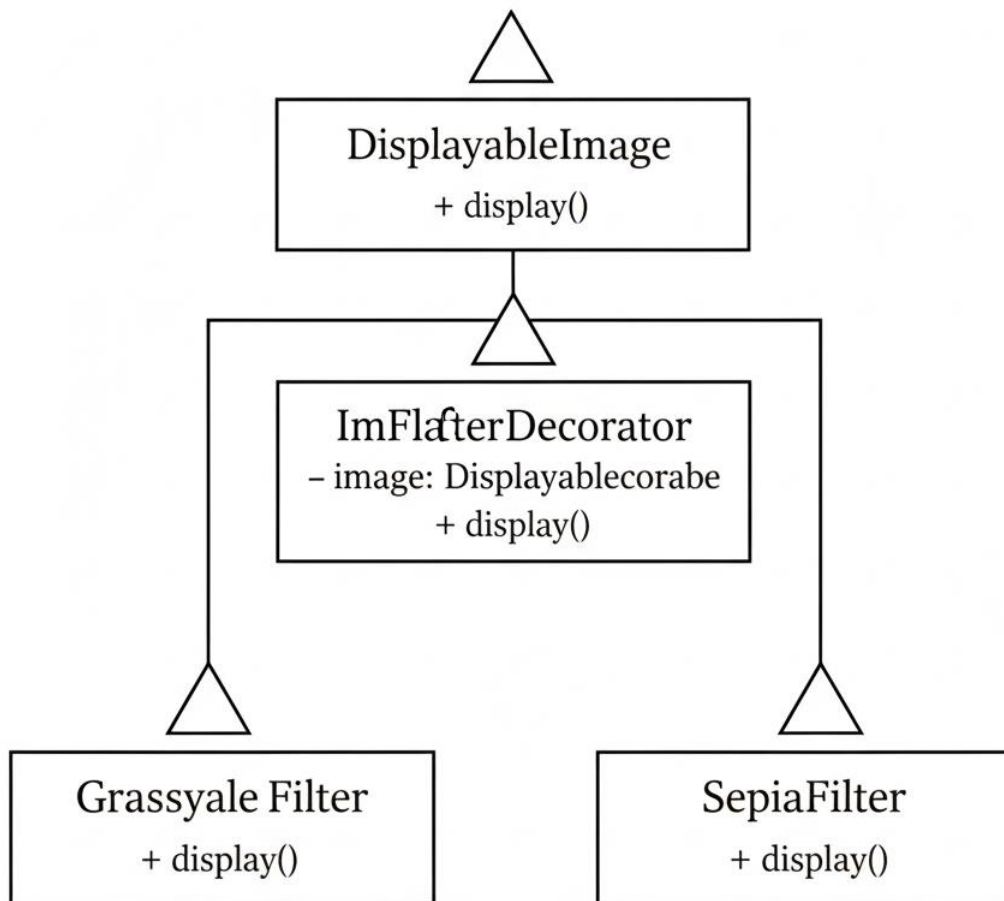
#### 4. Strategy Pattern - ImageSorter / SortByDate / SortByName / SortByRating



#### 5. Factory Pattern - FilterFactory



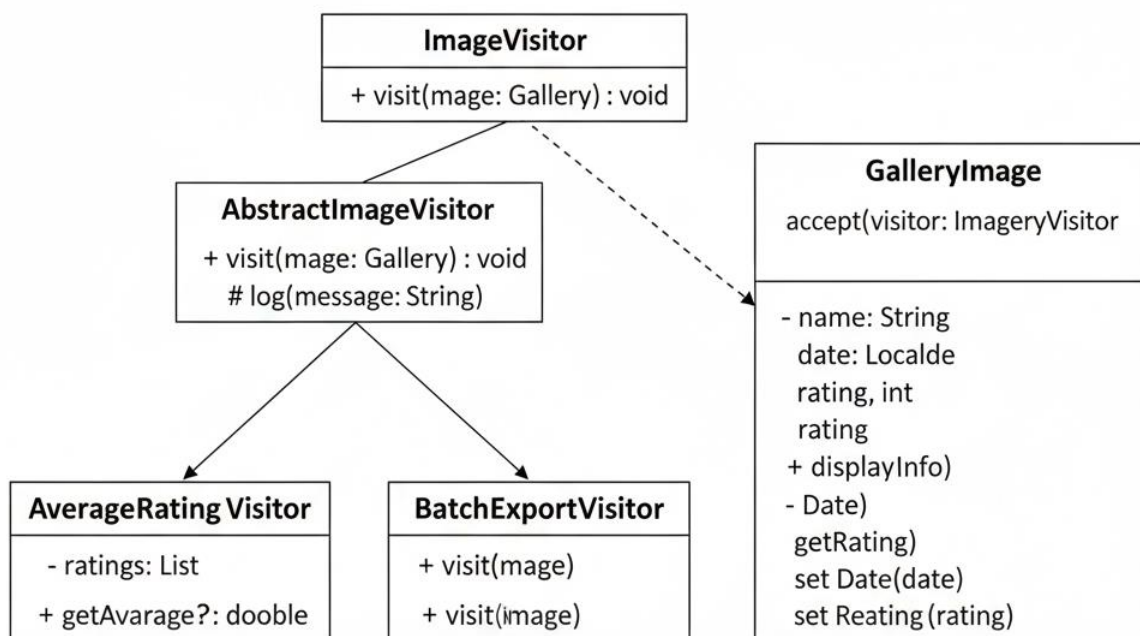
## 6. Decorator Pattern - GrayscaleFilter / SepiaFilter





- Facade: CameraFacade simplifies camera usage and uses subsystems.
- Observer: ImageGallery (Subject) notifies UserObserver instances.
- Singleton: AppManager manages the image repository.
- Strategy: ImageSorter can swap sorting strategies dynamically.
- Factory: FilterFactory creates filter objects from a type.
- Decorator: ImageFilterDecorator adds filter functionality without changing original image.

## 7. Visitor pattern



## 2.3 Screenshots

## User Registration and Observer Notifications

```
[INFO 2025-11-15 21:01:25] User1 subscribed to gallery updates.  
[INFO 2025-11-15 21:01:25] User2 subscribed to gallery updates.
```

[INFO 2025-11-15 21:01:25] User1 subscribed to gallery updates.

[INFO 2025-11-15 21:01:25] User2 subscribed to gallery updates.

- Shows users subscribing to gallery notifications.

## Taking Photos (Facade + Manager + Observer)

```
FocusSystem: Auto-focused.  
FlashSystem: Flash enabled.  
BasicCamera: Photo captured!  
StorageSystem: Saved Holiday  
[INFO 2025-11-15 21:01:25] Facade: Photo process simplified.  
AppManager: Image added to repository.  
ImageGallery: New image added: Holiday  
[User1] Notification: New image added: Holiday  
[User2] Notification: New image added: Holiday
```

FocusSystem: Auto-focused.

FlashSystem: Flash enabled.

BasicCamera: Photo captured!

StorageSystem: Saved Holiday

[INFO 2025-11-15 21:01:25] Facade: Photo process simplified.

AppManager: Image added to repository.

ImageGallery: New image added: Holiday

[User1] Notification: New image added: Holiday

[User2] Notification: New image added: Holiday

- Demonstrates **camera subsystem via Facade**, **image management via Singleton**, and **Observer notifications**.
- Repeat for other photos like Family and Birthday.

## Applying Filters (Decorator + Factory)

```
[INFO 2025-11-15 21:01:25] Applying Grayscale:  
Displaying original image: unnamed  
Applying Grayscale Filter  
[INFO 2025-11-15 21:01:25] Applying Sepia on top of grayscale:  
Displaying original image: unnamed  
Applying Grayscale Filter  
Applying Sepia Filter
```

[INFO 2025-11-15 21:01:25] Applying Grayscale:

Displaying original image: unnamed

Applying Grayscale Filter

[INFO 2025-11-15 21:01:25] Applying Sepia on top of grayscale:

Displaying original image: unnamed

Applying Grayscale Filter

Applying Sepia Filter

- Demonstrates **Decorator stacking filters** and **Factory creating filter objects**.

## Sorting Images (Strategy Pattern)

```
--- Sort by Date ---  
Images sorted by date.  
Image: Holiday, Date: 2025-11-09, Rating: 5  
Image: Family, Date: 2025-10-20, Rating: 4  
Image: Birthday, Date: 2025-09-15, Rating: 3  
  
--- Sort by Rating ---  
Images sorted by rating.  
Image: Holiday, Date: 2025-11-09, Rating: 5  
Image: Family, Date: 2025-10-20, Rating: 4  
Image: Birthday, Date: 2025-09-15, Rating: 3  
  
--- Sort by Name ---  
Images sorted by name.  
Image: Holiday, Date: 2025-11-09, Rating: 5  
Image: Family, Date: 2025-10-20, Rating: 4  
Image: Birthday, Date: 2025-09-15, Rating: 3
```

--- Sort by Date --- Images sorted by date. Image: Holiday, Date: 2025-11-09, Rating: 5  
Image: Family, Date: 2025-10-20, Rating: 4 Image: Birthday, Date: 2025-09-15, Rating: 3

--- Sort by Rating --- Images sorted by rating. Image: Holiday, Date: 2025-11-09, Rating: 5  
Image: Family, Date: 2025-10-20, Rating: 4 Image: Birthday, Date: 2025-09-15, Rating: 3

--- Sort by Name --- Images sorted by name. Image: Holiday, Date: 2025-11-09, Rating: 5  
Image: Family, Date: 2025-10-20, Rating: 4 Image: Birthday, Date: 2025-09-15, Rating: 3

- Demonstrates **Strategy pattern** by swapping sorting criteria dynamically.

## Visitor output:

```
--- Visitor: Average Rating ---  
Average Rating: 4.0  
  
--- Visitor: Export All Images ---  
Exporting image info: Holiday | Rating: 5  
Exporting image info: Family | Rating: 4  
Exporting image info: Birthday | Rating: 3
```

## 2.4 Application Workflow

1. User registers in the gallery → UserObserver subscribes.
2. User takes photo or records video → CameraFacade manages the process.
3. Image is added to AppManager → Singleton ensures central management.
4. Observers are notified → ImageGallery triggers update() in UserObserver.
5. User applies filters → Factory creates filter, Decorator applies it.
6. User sorts images → Strategy pattern dynamically changes sorting behavior
7. Visitor pattern can analyze image properties or perform batch operations → Visitors (AverageRatingVisitor, BatchExportVisitor) traverse GalleryImage objects without exposing internal state.

## 3. Conclusion

The project demonstrates a **well-integrated system using six design patterns**. Each pattern is **operational and contributes to the functionality**:

- **Facade** simplifies complex camera operations.

- **Observer** ensures real-time updates to users.
- **Singleton** centralizes image management.
- **Strategy** allows flexible sorting options.
- **Factory** provides dynamic filter creation.
- **Decorator** enables stacking multiple filters dynamically.
- **Visitor** allows batch operations and analysis without breaking encapsulation.

The architecture is modular, maintainable, and extensible for future features.

## 4. Further Work

- **Visitor Pattern Integration:** Analyze image properties or perform batch operations without exposing internal state.
- **Additional Filters:** Add more filters like Blur, Brightness, Contrast using Factory + Decorator.
- **UI Enhancement:** Create GUI for real-time interaction instead of console output. (Currently, the application runs in the console. For future work, we plan to implement a GUI using **JavaFX** or **Swing**, which would allow users to click buttons for capturing photos, applying filters, sorting images, and viewing notifications visually. This will improve usability and provide a real-time interactive experience.)
- **Video Processing:** Extend filters and sorting to videos.
- **Persistence:** Save gallery state to a database for permanent storage.
- **AI Features:** Integrate AI-based image tagging and recommendation system.

## 5. Source Code

- `com.project.camera` → Camera subsystem and Facade
- `com.project.controller` → GalleryController
- `com.project.filters` → Factory and Decorators
- `com.project.manager` → Singleton repository
- `com.project.observer` → Observer pattern classes
- `com.project.sorting` → Strategy pattern classes
- `com.project.visitor` → Visitor pattern classes (ImageVisitor, AbstractImageVisitor, AverageRatingVisitor, BatchExportVisitor)
- `com.project.util` → Logging utility

