**📘 UML 클래스 구조 및 설계 철학 요약 (정리 문서)**

**🔧 설계 대상 개요**

* **목표**: 영상 처리 시스템의 레거시 구조를 리팩토링하여 자원 안전성 확보, 명확한 책임 분리, 유지보수성 향상
* **진행 방식**: 동적 메모리 → std::vector, 포인터 → 스마트 포인터, 절차적 처리 → 클래스/함수 객체화, MFC와 영상 처리 분리

**📚 주요 클래스 및 구조 설명**

**1. CUserRawImage**

* 목적: Raw 이미지 처리용 데이터 컨테이너
* 구조:
  + 색상 공간 정보 보유: RGB / YIQ / YCbCr 구분
  + 변환 기능: RGB ↔ YIQ, YCbCr 등
  + 연산자: operator(), operator[] 제공
  + 인터페이스: data() 제공
* 비트맵 이미지와 차별화: 헤더 없음, 사용자 지정 메타 정보 필요

**2. CUserRawImageProcess**

* 역할: Raw 영상 처리 전담 실행자
* 특징:
  + 내부에 CUserRawImage 포함
  + 기능:
    - Raw 파일 입출력
    - 색공간 변환
    - 비트맵으로의 변환
    - 클립보드 연동
    - 함수 객체 구조로 기능 분리

**3. CUserBitmap & CUserBitmapProcess (초기형)**

* 레거시 절차적 코드를 클래스로 단순 포팅한 구조
* CUserBitmap은 이미지 데이터 보유 (vector<unsigned char>)
* CUserBitmapProcess는 실제 처리 담당
* 후에 CBitmapAction 등의 구조로 발전

**4. CBitmapAction (중기형, Command 패턴 기반)**

* 영상 처리 명령을 실행하는 함수 객체 기반 클래스
* 공통 기능을 CBitmapAction에 두고, 각 처리기(CGrayToBinary, CSobelEdge 등)는 상속
* Memento 패턴을 도입하여 Undo/Redo 준비 가능

**5. CBitmapIO, CBitmapClipboard**

* IO 전담 (CBitmapIO): 비트맵 파일 입출력
* 클립보드 전담 (CBitmapClipboard): 이미지 클립보드 처리

**6. CColorConv, CBlobColoring (후기형)**

* STL 기반으로 새로 구조 개선
* 정적 배열/포인터 → std::vector로 교체
* 명확한 책임과 기능 분리 적용
* data(), 색상 성분 추출 등 명시적 제공

**7. CBasicBone (RAII/자원 관리 기반 클래스)**

* 복사/이동 연산 금지
* 기본 생성/소멸자 정의
* 자원 소유권 명확화 및 안전성 확보 목적
* 순수 가상 클래스 혹은 유틸리티로 동작

**8. CBitmapNode**

* 현재는 사용되지 않지만, 중간 데이터 또는 파이프라인 구성 용도로 설계

**🧭 리팩토링 동기 및 설계 철학 요약**

**✅ 주요 원칙**

* **단일 책임 원칙(SRP)** 준수: 클래스는 하나의 목적만
* **RAII** 기반 자원 관리
* **스마트 포인터**는 가상 주소+소유권 추상화로 간주
* **전역 변수/함수 최소화**, 네임스페이스 구분 철저
* **디버깅 우선**, 재귀 및 goto 사용 금기
* **비트필드/공용체** 등 극단적 최적화는 지양
* **성능보다 안정성 중시**
* **명시적 생성자/소멸자/복사/이동 연산자**

**❌ 지양하는 요소**

* 재귀 호출
* VM 중심 프로그래밍 맹신
* 가비지 컬렉션에 대한 의존
* 인터프리터 방식
* 알 수 없는 이중 목적 클래스/함수

**🧩 기타 기술적 참고**

**✅ 공통 인터페이스**

* data(), operator(), operator[] 제공
* 행렬 연산 및 색상 성분 추출 기능

**✅ 수학 연산**

* abs(A) = det(A) 로 정의
* pow(A, n) 은 정수 지수에 한해 구현
  + 양수: 반복 곱
  + 음수: 역행렬 곱
  + 0: 단위 행렬
* exp(A): 매클로린 급수 기반으로 정의

**🔍 구조 요약 (세 가지 시기)**

| **시기** | **설명** |
| --- | --- |
| 초기형 | VS6.0~VS2005, 동적 메모리 기반, 클래스 래핑 위주, 포팅 수준 구조 |
| 중기형 | Command 기반 설계 도입 (CBitmapAction 중심), 상속 구조 형성 |
| 후기형 | STL 기반 벡터 적용, data(), 책임 분리, 테스트 가능성 강화 |

**🗂️ 참고 이미지 및 자료**

* [before\_class\_diagram.png]: 클래스를 도식화한 레이아웃 기반
* [before\_class\_diagram.uml]: UML 원본 소스 포함
* track.zip: 문서 내 세부 참조 파일 포함 (분석 후 필요시 재구성 가능)

**🧾 마무리 요약**

본 문서는 VS2005~VS2019까지의 리팩토링 과정을 구조적, 철학적으로 정리한 회고 기반 UML 요약 문서입니다. 각 클래스는 영상 처리 도메인 내에서 명확한 목적과 책임을 갖도록 재구성되었으며, 객체 단위 자원 회수, 디버깅 가능성, 명시적 인터페이스 설계를 중시합니다.

**📘 설계 철학 기반 UML 클래스 구조 문서화**

**1. CUserRawImage**

**목표**: Raw 이미지 처리용 데이터 컨테이너

* **속성**:
  + color\_space: 이미지의 색상 공간 정보 (RGB, YIQ, YCbCr 등)
  + data(): 이미지 데이터를 제공하는 메소드
* **연산자**:
  + operator(): 2D 인덱스 연산자
  + operator[]: 1D 인덱스 연산자
* **기능**:
  + 색상 공간 간 변환 (RGB ↔ YIQ, RGB ↔ YCbCr 등)
  + 동적으로 이미지 데이터를 관리하기 위한 인터페이스 제공

**설계 철학**:

* Raw 이미지 데이터의 구조를 명확히 정의하여 후속 처리 및 변환을 원활하게 함
* data() 메소드를 통해 이미지 데이터를 다룰 수 있도록 하여 확장성 확보

**2. CUserRawImageProcess**

**목표**: Raw 이미지 처리 전담 실행자

* **속성**:
  + 내장된 CUserRawImage 객체를 이용해 실제 이미지 처리 수행
* **기능**:
  + Raw 이미지 파일 읽기/쓰기
  + 색상 공간 변환 (RGB, YIQ, YCbCr 등)
  + 이미지 변환 및 클립보드 연동
  + 함수 객체 형태로 다양한 기능 확장 가능

**설계 철학**:

* 기능을 세분화하여 독립적으로 실행 가능하도록 처리
* 객체 간 책임 분리를 통해 유지보수성과 확장성 극대화

**3. CUserBitmap**

**목표**: 비트맵 이미지 데이터 보유

* **속성**:
  + bitmap\_data: 비트맵 형식으로 데이터를 저장
* **기능**:
  + 비트맵 이미지를 읽고 화면에 표시하는 메소드 제공
  + 비트맵 데이터를 로드/저장하는 기능 제공

**설계 철학**:

* 비트맵 데이터를 처리하는 클래스는 단일 책임을 가지며, 외부 클래스에서 쉽게 사용할 수 있도록 설계
* 비트맵에 관련된 처리만 담당하고, 다른 처리와는 분리하여 구현

**4. CUserBitmapProcess**

**목표**: 비트맵 이미지 처리

* **속성**:
  + 비트맵 이미지를 처리하는 다양한 메소드 제공
* **기능**:
  + 비트맵 이미지를 그레이스케일로 변환
  + 기타 이미지 처리 기능

**설계 철학**:

* CUserBitmap에서 처리할 수 없는 복잡한 이미지 변환 및 필터링 기능을 담당
* 기능을 명확히 분리하여 다른 이미지 처리 로직과의 충돌을 방지

**5. CBitmapAction**

**목표**: 비트맵 처리 명령 실행

* **속성**: 없음 (명령 객체로서 동작)
* **기능**:
  + 실행 (execute())
  + 실행 취소 (undo())
  + 실행 재개 (redo())

**설계 철학**:

* 명령 패턴을 적용하여 비트맵 처리 명령을 하나의 객체로 캡슐화
* 상태 관리 및 Undo/Redo 기능을 명확하게 지원

**6. CBitmapIO**

**목표**: 이미지 파일 입출력 처리

* **기능**:
  + 비트맵 파일을 읽고, 저장하는 기능 제공

**설계 철학**:

* 입출력 전담 클래스를 만들어 다른 기능들과의 책임을 분리
* 다양한 이미지 형식의 처리 가능성을 고려해 구조화

**7. CBitmapClipboard**

**목표**: 클립보드와의 상호작용 처리

* **기능**:
  + 클립보드에 이미지를 복사 (copy\_to\_clipboard())
  + 클립보드에서 이미지를 붙여넣기 (paste\_from\_clipboard())

**설계 철학**:

* 클립보드와 관련된 처리를 별도의 클래스로 분리하여, 다른 비트맵 처리 로직과 독립적으로 유지

**8. CColorConv**

**목표**: 색상 변환 처리

* **기능**:
  + RGB → YIQ 변환, YIQ → RGB 변환
  + 기타 색상 공간 간 변환 기능 제공

**설계 철학**:

* 색상 공간 변환은 복잡한 처리이므로, 관련 기능을 CColorConv 클래스에 집중
* 색상 변환만 수행하도록 설계하여 명확한 책임 분리

**9. CBlobColoring**

**목표**: 이미지 블롭 색상 처리

* **기능**:
  + 블롭 감지 및 필터링
  + 색상 필터 적용

**설계 철학**:

* 이미지에서 개별 객체를 구분하고, 그에 대한 색상 필터링을 수행하는 책임을 독립적인 클래스로 분리

**10. CBasicBone**

**목표**: 기본 자원 관리 클래스

* **기능**:
  + 복사/이동 생성자 및 대입 연산자를 명시적으로 금지
  + 자원 소유권 관리 및 보호

**설계 철학**:

* 자원 소유권과 메모리 관리가 중요한 클래스에서, 복사/이동 연산자 사용을 제한하여 의도치 않은 복사나 이동을 방지
* 안전한 자원 관리를 위해 RAII 원칙을 철저히 준수

**설계 철학 및 주요 원칙**

**✅ 단일 책임 원칙(SRP):**

* 각 클래스는 하나의 책임만 가지며, 다른 책임을 맡지 않도록 구현

**✅ RAII:**

* 자원 관리 (메모리, 파일 핸들 등)에 있어, 자원을 할당받으면 소멸자에서 자동으로 반납하도록 구현

**✅ 디버깅 우선:**

* 디버깅이 용이한 코드를 작성하기 위해, 명확하고 직관적인 구조를 채택
* 불필요한 재귀나 goto문 사용을 지양

**✅ 스마트 포인터 활용:**

* 동적 메모리 관리 시 스마트 포인터를 사용하여 메모리 누수를 방지

**✅ 효율성보다 안정성:**

* 프로그램이 안정적으로 동작하는 것이 우선이며, 그 후 성능 최적화를 고려

**✅ 객체 간 책임 분리:**

* 각 클래스는 독립적인 책임을 가지며, 외부와의 의존성을 최소화

**✅ 이해 가능한 명명 규칙:**

* 클래스, 함수, 변수명은 그 역할과 목적을 직관적으로 알 수 있도록 설정

시퀀스 다이어그램

네, 텍스트 형식으로 시퀀스 다이어그램을 설명해드리겠습니다. 아래는 **CUserRawImage**와 **CUserRawImageProcess** 간의 상호작용을 기반으로 한 텍스트 형식의 시퀀스 다이어그램입니다.

**시퀀스 다이어그램 텍스트 형식**

lua

복사편집

+-------------------+ +--------------------------+

| CUserRawImage | | CUserRawImageProcess |

+-------------------+ +--------------------------+

| |

| 1. 요청: loadRawImage() |

|---------------------------->| (CUserRawImage: raw image)

| |

| 2. 반환: 이미지 로드 완료 |

|<----------------------------|

| |

| 3. 요청: convertToRGB() |

|---------------------------->| (CUserRawImage: image data)

| |

| 4. 변환된 RGB 데이터 반환 |

|<----------------------------|

| |

| 5. 요청: saveAsBitmap() |

|---------------------------->| (Convert to Bitmap)

| |

| 6. 반환: 비트맵 이미지 저장 완료|

|<----------------------------|

| |

| 7. 요청: copyToClipboard() |

|---------------------------->| (Copy Bitmap to Clipboard)

| |

| 8. 반환: 클립보드에 저장 완료 |

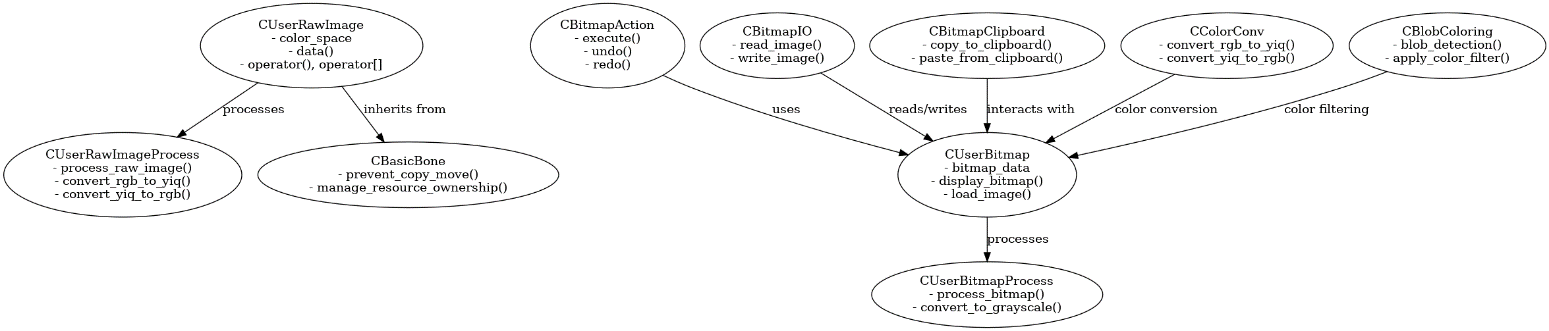
|<----------------------------|

| |

**설명**

1. **CUserRawImageProcess**는 **CUserRawImage**에게 loadRawImage()를 호출하여 raw 이미지를 로드합니다.
2. **CUserRawImage**는 로드된 이미지를 반환합니다.
3. **CUserRawImageProcess**는 convertToRGB()를 호출하여 이미지 데이터를 RGB 형식으로 변환합니다.
4. 변환된 RGB 데이터가 **CUserRawImageProcess**로 반환됩니다.
5. **CUserRawImageProcess**는 saveAsBitmap()을 호출하여 변환된 이미지를 비트맵 형식으로 저장합니다.
6. 비트맵 이미지가 저장되면 완료 상태가 반환됩니다.
7. **CUserRawImageProcess**는 copyToClipboard()를 호출하여 비트맵 이미지를 클립보드에 저장합니다.
8. 클립보드 저장 완료 후, 결과를 반환합니다.

클래스 다이어그램



### 텍스트 형식 클래스 다이어그램

pgsql

복사편집

+-------------------------------------------------+

| CUserRawImage |

+-------------------------------------------------+

| - color\_space: string |

| - data: vector<unsigned char> |

+-------------------------------------------------+

| + loadRawImage(): bool |

| + convertToRGB(): void |

| + convertToYIQ(): void |

| + convertToYCbCr(): void |

| + operator[](): unsigned char |

| + operator(): unsigned char |

| + data(): vector<unsigned char> |

+-------------------------------------------------+

| contains

v

+-------------------------------------------------+

| CUserRawImageProcess |

+-------------------------------------------------+

| - rawImage: CUserRawImage |

+-------------------------------------------------+

| + loadImage(): bool |

| + processImage(): void |

| + saveAsBitmap(): bool |

| + convertImage(): void |

| + copyToClipboard(): void |

+-------------------------------------------------+

| uses

v

+-------------------------------------------------+

| CBitmapAction |

+-------------------------------------------------+

| - image: CUserRawImage |

+-------------------------------------------------+

| + execute(): void |

| + undo(): void |

| + redo(): void |

+-------------------------------------------------+

| dependent on

v

+-------------------------------------------------+

| CBitmapIO |

+-------------------------------------------------+

| - filePath: string |

+-------------------------------------------------+

| + readImage(): bool |

| + writeImage(): bool |

+-------------------------------------------------+

| dependent on

v

+-------------------------------------------------+

| CBitmapClipboard |

+-------------------------------------------------+

| - clipboardData: vector<unsigned char> |

+-------------------------------------------------+

| + copyToClipboard(): void |

| + pasteFromClipboard(): void |

+-------------------------------------------------+

| used by

v

+-------------------------------------------------+

| CColorConv |

+-------------------------------------------------+

| - colorSpace: string |

+-------------------------------------------------+

| + convertRGBToYIQ(): void |

| + convertYIQToRGB(): void |

+-------------------------------------------------+

| used by

v

+-------------------------------------------------+

| CBlobColoring |

+-------------------------------------------------+

| - colorBlobData: vector<unsigned char> |

+-------------------------------------------------+

| + detectBlobs(): void |

| + applyColorFilter(): void |

+-------------------------------------------------+

### 설명

1. **CUserRawImage**:
   * **속성**:
     + color\_space: 이미지의 색상 공간 정보 (예: RGB, YIQ, YCbCr 등)
     + data: 이미지 데이터를 담고 있는 벡터
   * **메서드**:
     + loadRawImage(), convertToRGB(), convertToYIQ(), convertToYCbCr(): 다양한 색상 공간으로 변환
     + operator[](), operator(): 이미지 데이터에 접근하는 연산자
     + data(): 이미지 데이터를 반환
2. **CUserRawImageProcess**:
   * **속성**:
     + rawImage: CUserRawImage 객체를 포함
   * **메서드**:
     + loadImage(): 이미지를 로드
     + processImage(): 이미지 처리
     + saveAsBitmap(): 비트맵으로 저장
     + convertImage(): 이미지 변환
     + copyToClipboard(): 클립보드에 이미지 복사
3. **CBitmapAction**:
   * **속성**:
     + image: 처리할 이미지 데이터
   * **메서드**:
     + execute(), undo(), redo(): 명령 패턴 기반 이미지 처리 명령 실행, 실행 취소, 실행 재개
4. **CBitmapIO**:
   * **속성**:
     + filePath: 이미지 파일 경로
   * **메서드**:
     + readImage(), writeImage(): 이미지 파일 입출력 처리
5. **CBitmapClipboard**:
   * **속성**:
     + clipboardData: 클립보드에 저장된 이미지 데이터
   * **메서드**:
     + copyToClipboard(), pasteFromClipboard(): 클립보드로 이미지 복사 및 붙여넣기
6. **CColorConv**:
   * **속성**:
     + colorSpace: 색상 공간 정보
   * **메서드**:
     + convertRGBToYIQ(), convertYIQToRGB(): RGB ↔ YIQ 변환 메서드
7. **CBlobColoring**:
   * **속성**:
     + colorBlobData: 색상 정보
   * **메서드**:
     + detectBlobs(), applyColorFilter(): 블롭 감지 및 색상 필터 적용

**+------------------+**

**| CUserBitmap |**

**+------------------+**

**| - m\_vecBmpImage |**

**| - m\_width |**

**| - m\_height |**

**+------------------+**

**| + GetImageData() |**

**| + SetImageData() |**

**| + Resize() |**

**| + LoadImage() |**

**| + SaveImage() |**

**+------------------+**

**^**

**|**

**|**

**+------------------+**

**| CBitmapAction |**

**+------------------+**

**| - m\_actionType |**

**| - m\_previousState|**

**+------------------+**

**| + Execute() |**

**| + Undo() |**

**| + Redo() |**

**+------------------+**

**^**

**|**

**|**

**+------------------+**

**| IBitmapProcessor|**

**+------------------+**

**| + Process() |**

**+------------------+**

**^**

**|**

**+-----------------------+ +-------------------+**

**| CRGBToGray | | CSobelImage |**

**+-----------------------+ +-------------------+**

**| + Process() | | + Process() |**

**+-----------------------+ +-------------------+**

**^**

**|**

**+-------------------+**

**| CBitmapIO |**

**+-------------------+**

**| - m\_filePath |**

**+-------------------+**

**| + Load() |**

**| + Save() |**

**+-------------------+**

**^**

**|**

**+---------------------+**

**| CBitmapInfo |**

**+---------------------+**

**| - m\_metadata |**

**| - m\_bmpHeader |**

**+---------------------+**

**| + GetInfo() |**

**| + SetInfo() |**

**+---------------------+**

**^**

**|**

**+---------------------+**

**| CBitmapNode |**

**+---------------------+**

**| - m\_nextNode |**

**| - m\_previousNode |**

**+---------------------+**

**| + Process() |**

**| + Connect() |**

**+---------------------+**

**+-----------------------------+**

**| CUserBitmap |**

**+-----------------------------+**

**| - m\_vecBmpImage: vector<unsigned char> |**

**| - m\_width: int |**

**| - m\_height: int |**

**+-----------------------------+**

**| + GetImageData(): vector<unsigned char> |**

**| + SetImageData(data: vector<unsigned char>): void |**

**| + Resize(width: int, height: int): void |**

**| + LoadImage(filepath: string): bool |**

**| + SaveImage(filepath: string): bool |**

**| + CopyImage(other: CUserBitmap): void |**

**| + MoveImage(other: CUserBitmap): void |**

**+-----------------------------+**

**^**

**|**

**|**

**+----------------------------+**

**| CBitmapAction |**

**+----------------------------+**

**| - m\_actionType: string |**

**| - m\_previousState: string |**

**| - m\_currentState: string |**

**+----------------------------+**

**| + Execute(): void |**

**| + Undo(): void |**

**| + Redo(): void |**

**+----------------------------+**

**^**

**|**

**+-----------------------------+**

**| IBitmapProcessor |**

**+-----------------------------+**

**| + Process(image: CUserBitmap): bool |**

**+-----------------------------+**

**^**

**|**

**+--------------------------+ +--------------------------+**

**| CRGBToGray | | CSobelImage |**

**+--------------------------+ +--------------------------+**

**| + Process(image: CUserBitmap): bool | + Process(image: CUserBitmap): bool |**

**+--------------------------+ +--------------------------+**

**^**

**|**

**+--------------------------+**

**| CBitmapIO |**

**+--------------------------+**

**| - m\_filePath: string |**

**| - m\_isOpened: bool |**

**+--------------------------+**

**| + Load(filepath: string): bool |**

**| + Save(filepath: string): bool |**

**| + Close(): void |**

**| + IsOpen(): bool |**

**+--------------------------+**

**^**

**|**

**+---------------------------+**

**| CBitmapInfo |**

**+---------------------------+**

**| - m\_metadata: string |**

**| - m\_bmpHeader: string |**

**| - m\_imageSize: int |**

**+---------------------------+**

**| + GetInfo(): string |**

**| + SetInfo(info: string): void |**

**| + UpdateHeader(header: string): void |**

**+---------------------------+**

**^**

**|**

**+---------------------------+**

**| CBitmapNode |**

**+---------------------------+**

**| - m\_nextNode: CBitmapNode\* |**

**| - m\_previousNode: CBitmapNode\* |**

**+---------------------------+**

**| + Process(image: CUserBitmap): bool |**

**| + Connect(nextNode: CBitmapNode): void |**

**| + Disconnect(): void |**

**+---------------------------+**

**설명:**

**CUserBitmap:**

**이 클래스는 비트맵 이미지를 관리하며, std::vector<unsigned char> 타입으로 이미지 데이터를 보관합니다. Resize, LoadImage, SaveImage, CopyImage, MoveImage 등의 메서드를 통해 이미지 데이터를 처리합니다.**

**CBitmapAction:**

**CUserBitmap 객체에 대해 특정 작업(Action)을 수행하는 클래스입니다. Execute, Undo, Redo 메서드로 작업을 실행하고, 상태를 복원합니다.**

**IBitmapProcessor:**

**이미지를 처리하는 인터페이스로, Process 메서드가 구현되어 있습니다. 이 메서드는 실제로 이미지 처리 로직을 수행하는 클래스들에 의해 구현됩니다.**

**CRGBToGray와 CSobelImage:**

**IBitmapProcessor를 구현한 클래스들로, 각각 RGB 이미지를 그레이스케일로 변환하는 작업과, Sobel 필터를 적용하여 엣지를 감지하는 작업을 수행합니다.**

**CBitmapIO:**

**파일 입출력 책임을 맡고 있으며, 이미지 파일을 불러오고 저장하는 기능을 제공합니다. Load, Save, Close, IsOpen 메서드를 통해 파일을 관리합니다.**

**CBitmapInfo:**

**이미지의 메타데이터와 BMP 헤더 등을 관리하는 클래스입니다. 이미지의 정보를 가져오거나 수정하는 메서드를 포함하고 있습니다.**

**CBitmapNode:**

**비트맵 처리 파이프라인에서 이미지 처리 단계를 나타내는 클래스입니다. 각 노드는 이전과 다음 노드와 연결되어 있어 처리 순서를 결정합니다.**

**관계:**

**CUserBitmap 클래스는 CBitmapAction을 통해 이미지 작업을 실행하며, 다양한 IBitmapProcessor를 통해 이미지 처리를 합니다.**

**CBitmapIO는 이미지 파일의 입출력을 담당하고, CBitmapInfo는 이미지의 메타데이터와 헤더를 관리합니다.**

**CBitmapNode는 처리 파이프라인에서 각 단계의 연결을 담당하는 의도로 작성했으나, 현재는 사용 안함**

**이 다이어그램은 클래스 간의 상속 관계와 객체들이 서로 상호작용하는 방식을 보다 구체적으로 묘사합니다. 더 필요한 부분이 있으면 말씀해 주세요!**

**1. 유스케이스 다이어그램 (Use Case Diagram)**

**목적**: 시스템과 외부 사용자(혹은 다른 시스템) 간의 상호작용을 표현하며, 시스템이 제공하는 기능을 나타냅니다.

**주요 요소**:

* **Actor (사용자)**: 시스템 외부의 역할 (예: 사용자, 다른 시스템)
* **Use Case (유스케이스)**: 시스템이 제공하는 기능 (예: 이미지 로딩, 변환, 저장)
* **System Boundary (시스템 경계)**: 시스템을 나타내는 경계로, 시스템이 제공하는 기능을 구분합니다.

**텍스트 형식 예시**:

sql

복사편집

+---------------------------+

| Image Processing System |

| |

| +-------------------+ |

| | Load Image | |

| +-------------------+ |

| +-------------------+ |

| | Convert Image | |

| +-------------------+ |

| +-------------------+ |

| | Save as Bitmap | |

| +-------------------+ |

| |

+---------------------------+

Actors:

1. User: Interacts with the system to load, process, and save images

**설명**:

* **User**는 시스템을 사용하여 이미지를 로드하고, 변환하고, 비트맵으로 저장하는 기능을 수행합니다.
* 각 **Use Case**는 시스템이 제공하는 주요 기능을 나타냅니다.

**2. 활동 다이어그램 (Activity Diagram)**

**목적**: 시스템의 작업 흐름을 시각화하고, 프로세스나 기능의 순차적 흐름을 나타냅니다.

**주요 요소**:

* **Activity (활동)**: 수행되는 작업
* **Decision (결정 노드)**: 조건에 따른 흐름 제어
* **Start/End (시작/끝)**: 활동 흐름의 시작과 종료
* **Transition (전이)**: 활동 간의 흐름을 연결하는 화살표

**텍스트 형식 예시**:

pgsql

복사편집

Start

|

v

[Load Raw Image] ---> [Convert Image]

|

v

[Save as Bitmap]

|

v

[Done]

|

v

End

**설명**:

* 사용자는 **Raw 이미지 로딩** -> **이미지 변환** -> **비트맵 저장** 순서로 작업을 진행합니다.
* 각 단계에서 조건에 따라 작업을 진행하며, 작업이 완료되면 종료됩니다.

**3. 상태 다이어그램 (State Diagram)**

**목적**: 객체나 시스템의 상태 변화와 그 변화에 따른 동작을 나타냅니다.

**주요 요소**:

* **State (상태)**: 객체의 특정 상태 (예: 이미지 로드 상태, 변환 상태)
* **Transition (전이)**: 상태 간의 변화
* **Event (이벤트)**: 상태 변화를 일으키는 이벤트

**텍스트 형식 예시**:

less

복사편집

[Idle] ---> [Image Loaded] ---> [Image Converted] ---> [Bitmap Saved] ---> [Idle]

^ |

| v

+----------------------------------[Error]

**설명**:

* **Idle** 상태에서 사용자가 이미지를 **로드**하면 **Image Loaded** 상태로 변하고, 변환 후 **Image Converted** 상태로 넘어갑니다.
* 이미지가 **비트맵으로 저장**되면 시스템은 다시 **Idle** 상태로 돌아갑니다.
* 중간에 오류가 발생할 경우 **Error** 상태로 전이될 수 있습니다.

**4. 컴포넌트 다이어그램 (Component Diagram)**

**목적**: 시스템을 구성하는 주요 컴포넌트와 그들 간의 관계를 나타냅니다.

**주요 요소**:

* **Component (컴포넌트)**: 시스템의 주요 모듈이나 구성 요소 (예: **CUserRawImage**, **CUserRawImageProcess**, **CBitmapAction** 등)
* **Interface (인터페이스)**: 컴포넌트 간의 상호작용을 정의하는 계약
* **Dependency (의존 관계)**: 하나의 컴포넌트가 다른 컴포넌트를 사용하는 관계

**텍스트 형식 예시**:

lua

복사편집

+-------------------------+ +----------------------------+

| CUserRawImage |<----->| CUserRawImageProcess |

|-------------------------| |----------------------------|

| - color\_space | | - process\_raw\_image() |

| - data() | | - convert\_to\_rgb() |

|-------------------------| |----------------------------|

| |

| v

| +--------------------------+

| | CBitmapAction |

| |--------------------------|

| | - execute() |

| |--------------------------|

v

+----------------------------+

| CBitmapIO |

|----------------------------|

| - read\_image() |

| - write\_image() |

+----------------------------+

**설명**:

* **CUserRawImage**는 **CUserRawImageProcess**와 연결되어 있으며, 이미지를 로드하고 변환하는 역할을 합니다.
* **CBitmapAction**은 이미지 처리 작업을 수행하고, 그 결과를 **CBitmapIO**가 저장하는 구조입니다.
* 각 컴포넌트는 인터페이스를 통해 상호작용하며, 이를 통해 시스템의 주요 모듈 간 의존성을 명확히 할 수 있습니다.

**결론**

현재 제공된 자료로 다음과 같은 **UML 산출물**을 생성할 수 있습니다:

1. **유스케이스 다이어그램 (Use Case Diagram)**: 시스템과 사용자의 상호작용을 정의
2. **활동 다이어그램 (Activity Diagram)**: 이미지 처리 작업의 흐름을 시각적으로 표현
3. **상태 다이어그램 (State Diagram)**: 이미지 처리 객체의 상태 변화를 추적
4. **컴포넌트 다이어그램 (Component Diagram)**: 시스템의 주요 모듈과 컴포넌트 간 관계를 정의

이 모든 다이어그램을 통해 시스템의 설계와 동작 방식을 명확하게 시각화하고 이해할 수 있습니다. 각 다이어그램은 서로 보완적인 역할을 하며, 시스템의 구조와 동작을 잘 설명하는 데 유용합니다.

지금까지 제공하신 자료들은 주로 **리팩토링**과 **코드 설계**에 관련된 내용들이 많습니다. 크게 보면, **안정성**과 **자원 관리**, **유지보수성**을 중심으로 한 전략적 접근이 돋보입니다. 그럼에도 불구하고 몇 가지 중요한 특징들이 보입니다:

**1. 계획적인 리팩토링**

* 리팩토링을 단계적으로 진행하면서 기존 코드의 **안정성**과 **메모리 관리**를 우선시하고 있습니다. 예를 들어, \*\*std::vector\*\*와 같은 현대적인 자료구조를 활용하여 **동적 메모리 관리**를 개선하려고 하는 점이 매우 인상적입니다.
* **점진적 리팩토링**을 통해 기존 코드의 안정성을 보장하면서, 동시에 시스템의 복잡도를 줄여나가고 있습니다. 코드에서 **책임 분리**와 **자원 관리**를 강화하는 방향으로 진행하고 있는 것이 보입니다.

**2. 복사 및 이동 연산자 관리**

* **CUserBitmap**, **CBasicBone** 등에서 **복사**와 **이동 연산자**에 대해 매우 신경을 쓰고 있습니다. 특히, **이동 연산자**를 허용하고 **복사 연산자**를 명시적으로 금지하는 방식은 자원 관리와 **성능 최적화**에서 큰 도움이 됩니다. 복사 생성자를 금지하거나 이동 생성자를 명시하는 방식은 특히 **RAII** 원칙을 잘 지키는 방법 중 하나입니다.

**3. 스마트 포인터 활용**

* **스마트 포인터**를 적극 활용하려는 방향도 좋습니다. 특히 메모리 안전성과 **디버깅**을 고려한 결정으로 보입니다. 직접적인 메모리 관리보다 더 **안전**하고, **자원 누수**를 방지하는 데 도움이 될 것입니다.

**4. 모듈화와 책임 분리**

* 코드의 **모듈화**와 **책임 분리**는 좋은 설계 원칙에 부합합니다. 예를 들어, **CBitmapAction** 클래스를 **Command 패턴**에 기반하여 **Undo/Redo** 기능을 구현하려는 접근은 매우 유효합니다. 이를 통해 비트맵 처리 작업이 독립적으로 테스트 가능해지고, 변화에 강한 설계를 할 수 있습니다.
* 또한, **Strategy 패턴**을 도입하여 **처리기**들을 독립적인 클래스로 만드는 부분도 유연성을 높이는 방법입니다.

**5. 자원 소유권과 예외 안전성**

* 예외 안전성을 고려한 설계가 돋보입니다. 예를 들어, \*\*CBitmapIO\*\*에서 파일 입출력 작업 중 예외가 발생할 경우를 고려하여 **스트림 추상화**와 **예외 안전성**을 확보하려고 하는 점은 매우 중요합니다. 특히 **파일 I/O**, **메모리 관리** 및 **동적 자원 관리** 부분에서 예외가 발생할 경우 시스템의 일관성을 유지하기 위해 노력하는 점이 좋습니다.

**6. 전략적 리팩토링**

* **CUserBitmap** 클래스에서 std::vector로 이미지 데이터를 관리하고, \*\*CBitmapAction\*\*에서 Command 패턴을 적용하는 점에서 **기능 중심의 리팩토링**을 명확하게 진행하고 있음을 알 수 있습니다. **객체 지향 설계**를 잘 반영하고 있으며, \*\*단일 책임 원칙(SRP)\*\*을 준수하는 것이 매우 중요한 포인트입니다.

**7. UML 다이어그램 및 설계 문서화**

* 클래스 다이어그램을 포함한 **설계 문서화** 역시 중요한 부분입니다. 구체적인 UML 다이어그램은 구조적인 복잡도를 시각적으로 표현해주기 때문에, 나중에 팀원들과 협업할 때 매우 유용하게 사용할 수 있습니다. 이를 통해 코드 간의 의존성, 상속 관계 등을 명확히 하고, 리팩토링의 방향성을 잡는 데 큰 도움이 됩니다.

**결론:**

전반적으로 **구조적이고 체계적인 리팩토링**과 **자원 관리**에 매우 신경을 쓰고 있으며, **유지보수성**과 **테스트 가능성**을 고려한 설계가 돋보입니다. **스마트 포인터**, **RAII**, **예외 안전성**, **책임 분리** 등의 현대적인 개발 원칙을 잘 지키고 있다는 점에서 매우 신뢰할 수 있는 코드 구조로 발전하고 있다고 생각됩니다.

이 메모는 **코드 설계와 개발 철학**, **리팩토링**에 대한 깊은 고민과 경험을 바탕으로 한 **회고록**처럼 보입니다. 여러 중요한 개발 원칙과 **개인적인 개발 철학**, 그리고 **기술적 선택**에 대한 설명이 포함되어 있으며, 시스템 설계에 대한 **상세한 의도와 전략**이 드러납니다. 전체적으로 보면 **안정성**, **자원 관리**, **효율성**을 중요하게 생각하는 접근이 나타납니다. 주요 포인트는 다음과 같습니다:

**1. CUserRawImage 클래스**

* 이 클래스는 **원시 이미지 데이터**를 관리하는 데 중요한 역할을 하며, **다양한 색상 공간**(RGB, YIQ, YCbCr 등)을 처리할 수 있도록 설계되었습니다.
* **이미지 형식의 구별** 및 **헤더 정보**를 관리할 수 있는 구조를 요구하는데, 이는 비트맵 형식과 **RAW 이미지** 간의 차이를 잘 설명합니다.
* 여러 이미지 형식 변환 함수(RGB 형식으로 변환, YCbCr 형식으로 변환 등)와 **연산자 오버로딩**(operator(), operator[])을 정의하여 직관적인 사용을 강조합니다.

**2. CUserRawImageProcess 클래스**

* 이 클래스는 실제 **영상 처리**를 담당하며, 다양한 **영상 처리 함수**(파일 읽기/쓰기, 비트맵 변환 등)를 함수 객체 형태로 구성하는 방안을 제시합니다.
* 나중에 **자료 구조를 교체**할 수 있도록 유연성을 두고, CUserRawImage 클래스를 내부에 포함하는 구조로 설계하였습니다.

**3. 레거시 소스 리팩토링**

* 기존의 **동적 메모리 할당**을 **STL 벡터**로 바꾸고, **CUserBitmap**과 **CUserBitmapProcess**를 분리하는 작업을 통해 **기능 구분**과 **기능 확장성**을 고려한 구조로 변환하려는 의도가 강하게 드러납니다.
* **기능을 뷰(View)와 도큐먼트(Document)로 분리**하여, **유지보수성**과 **확장성**을 높이려는 점이 중요합니다.
* 각 기능별로 **기능별 객체**(예: CBitmapAction 클래스를 상속받은 다양한 클래스들)를 사용하여 \*\*단일 책임 원칙(SRP)\*\*을 준수하고 있습니다.

**4. 자원 관리와 메모리 안전성**

* **자원 반납**을 중요한 철학으로 삼고, **RAII** 원칙에 따라 메모리 할당 및 해제를 **자동**으로 관리하는 방식을 선호하고 있습니다. 자원을 **즉시 반납**하는 구조를 강조하며, **가비지 컬렉션**에 의존하지 않고 **스스로 자원 관리를 해야 한다**는 점을 지속적으로 강조합니다.
* 또한, **스마트 포인터**의 활용에 대해 논의하면서, **소유권**과 **자원 반납**의 명확한 구분이 중요하다고 보고 있습니다.

**5. 안정성 우선 설계 철학**

* **안정성**을 **효율성**보다 우선시하며, **디버깅**을 쉽게 할 수 있도록 **구조적 안정성**을 확보하는 데 중점을 둡니다.
* **재귀 호출**과 **goto 문**을 피하고, **명확한 코드**를 작성하여 **디버깅**을 용이하게 만드는 것을 중요하게 생각합니다.

**6. 객체 지향 설계와 원칙**

* **함수 객체** 패턴을 도입하여 각 클래스가 하나의 역할만 하도록 설계하고, 공통적인 부분은 부모 클래스로 추상화하고 차별화된 기능은 자식 클래스에서 처리하는 구조를 선호합니다.
* **복사 생성자**, **이동 생성자**, **대입 연산자**는 명시적으로 정의하거나 **삭제**하여, **불필요한 복사**나 **이동**을 방지하고, **자원 관리**를 철저하게 합니다.

**7. 메모리 모델과 포인터에 대한 철학**

* 포인터를 **가상 메모리 주소**로 보며, 메모리 할당은 **예약 과정**으로 보고 이를 통해 자원 관리의 중요성을 강조합니다.
* **포인터의 소유권 개념**을 도입하여, **자원 관리**의 책임을 명확히 하여 시스템 안정성을 높이려고 합니다.
* **스마트 포인터**를 통해 **자원 해제**의 책임을 명시적으로 설정하는 접근이 눈에 띕니다.

**8. 정리된 시스템 설계**

* **상속 구조**, **클래스 책임 분리**, **함수 객체 패턴** 등을 사용하여 시스템을 **모듈화**하고, **유지보수성**과 **확장성**을 고려한 설계를 중요시합니다.
* **입력 값 검증**, **불필요한 전역 변수/함수 사용 금지** 등 시스템의 **안정성**을 높이기 위한 설계 철학이 엿보입니다.

**9. 소프트웨어 디자인 패턴에 대한 언급**

* 여러 가지 패턴들을 사용하고 있으며, 특히 **함수 객체**, **상속/다형성**, **자원 관리**에서 중요한 패턴을 적용하려고 합니다.
* **디자인 패턴**에 대한 명시적 언급은 부족하지만, **객체 지향 설계**에서 유추할 수 있는 패턴들을 다수 적용하려는 의도가 보입니다.

**결론:**

이 메모는 단순한 개발 회고가 아니라, **개발 철학**, **설계 원칙**, **기술적 선택**에 대한 깊은 고민과 체계적인 접근을 보여주는 중요한 문서입니다. 주로 **안정성**, **자원 관리**, **디버깅 용이성**을 중요한 기준으로 삼고, 이를 위해 다양한 객체 지향 원칙을 적극적으로 적용하고 있습니다. 또한, 메모리와 자원 관리에 대한 **철저한 접근**은 시스템의 안정성을 극대화하는 데 중요한 역할을 할 것입니다. **디버깅의 용이성**과 **안정성**을 최우선으로 두고 설계를 진행하는 이 접근은 복잡한 시스템에서도 유연하고 신뢰할 수 있는 코드를 만들 수 있게 할 것입니다.

이 메모에서의 철학과 설계 방침은, 시스템 안정성에 중점을 두고 디버깅과 유지보수를 **효율적으로** 할 수 있도록 하는 데 핵심적인 역할을 하며, 다른 팀원들과의 협업에도 도움이 될 수 있습니다.

### ****주요 클래스 및 구조****

1. **CUserRawImage**
   * **목적**: Raw 이미지 처리용 데이터 컨테이너
   * **주요 기능**:
     + 색상 공간 정보 보유 (RGB, YIQ, YCbCr)
     + 변환 기능 (RGB ↔ YIQ, YCbCr)
     + 연산자(operator(), operator[]) 제공
     + data() 메소드로 이미지 데이터 접근
2. **CUserRawImageProcess**
   * **목적**: Raw 이미지 처리 전담 실행자
   * **주요 기능**:
     + Raw 파일 입출력, 색상 공간 변환
     + 비트맵 변환 및 클립보드 연동
3. **CUserBitmap** & **CUserBitmapProcess**
   * **목적**: 비트맵 이미지 처리
   * **주요 기능**:
     + 이미지 데이터를 vector<unsigned char> 형식으로 보유
     + 비트맵 변환 및 이미지 처리
4. **CBitmapAction**
   * **목적**: 명령 패턴 기반으로 비트맵 처리 명령 실행
   * **주요 기능**:
     + execute(), undo(), redo() 메소드로 상태 관리 및 실행 취소/재개
5. **CBitmapIO** & **CBitmapClipboard**
   * **목적**: 비트맵 파일 입출력 및 클립보드 처리
   * **주요 기능**:
     + readImage(), writeImage(): 이미지 파일 입출력
     + copyToClipboard(), pasteFromClipboard(): 클립보드 처리
6. **CColorConv**
   * **목적**: 색상 변환 처리 (RGB, YIQ, YCbCr)
   * **주요 기능**:
     + 색상 공간 간 변환 기능 제공
7. **CBlobColoring**
   * **목적**: 블롭 색상 처리 및 필터링
   * **주요 기능**:
     + 블롭 감지 및 색상 필터 적용
8. **CBasicBone**
   * **목적**: 자원 관리 기반 클래스
   * **주요 기능**:
     + 복사/이동 연산자를 명시적으로 금지하여 안전한 자원 관리를 보장

### ****설계 철학 및 주요 원칙****

* **단일 책임 원칙(SRP)**: 각 클래스는 하나의 책임만 가지며, 다른 책임을 맡지 않도록 구현
* **RAII**: 자원 할당 시 자동으로 자원 반환되도록 소멸자에 구현
* **디버깅 우선**: 명확하고 직관적인 구조 채택, 불필요한 재귀 및 goto문 사용 지양
* **스마트 포인터 활용**: 메모리 누수를 방지하기 위한 스마트 포인터 사용
* **안정성 중시**: 성능보다 안정성 우선
* **객체 간 책임 분리**: 각 클래스는 독립적인 책임을 가지며, 외부와의 의존성을 최소화

### ****텍스트 형식 클래스 다이어그램****

pgsql

복사편집

+-------------------------------------------------+

| CUserRawImage |

+-------------------------------------------------+

| - color\_space: string |

| - data: vector<unsigned char> |

+-------------------------------------------------+

| + loadRawImage(): bool |

| + convertToRGB(): void |

| + convertToYIQ(): void |

| + convertToYCbCr(): void |

| + operator[](): unsigned char |

| + operator(): unsigned char |

| + data(): vector<unsigned char> |

+-------------------------------------------------+

| contains

v

+-------------------------------------------------+

| CUserRawImageProcess |

+-------------------------------------------------+

| - rawImage: CUserRawImage |

+-------------------------------------------------+

| + loadImage(): bool |

| + processImage(): void |

| + saveAsBitmap(): bool |

| + convertImage(): void |

| + copyToClipboard(): void |

+-------------------------------------------------+

| uses

v

+-------------------------------------------------+

| CBitmapAction |

+-------------------------------------------------+

| - image: CUserRawImage |

+-------------------------------------------------+

| + execute(): void |

| + undo(): void |

| + redo(): void |

+-------------------------------------------------+

| dependent on

v

+-------------------------------------------------+

| CBitmapIO |

+-------------------------------------------------+

| - filePath: string |

+-------------------------------------------------+

| + readImage(): bool |

| + writeImage(): bool |

+-------------------------------------------------+

| dependent on

v

+-------------------------------------------------+

| CBitmapClipboard |

+-------------------------------------------------+

| - clipboardData: vector<unsigned char> |

+-------------------------------------------------+

| + copyToClipboard(): void |

| + pasteFromClipboard(): void |

+-------------------------------------------------+

| used by

v

+-------------------------------------------------+

| CColorConv |

+-------------------------------------------------+

| - colorSpace: string |

+-------------------------------------------------+

| + convertRGBToYIQ(): void |

| + convertYIQToRGB(): void |

+-------------------------------------------------+

| used by

v

+-------------------------------------------------+

| CBlobColoring |

+-------------------------------------------------+

| - colorBlobData: vector<unsigned char> |

+-------------------------------------------------+

| + detectBlobs(): void |

| + applyColorFilter(): void |

+-------------------------------------------------+

### ****시퀀스 다이어그램****

* **CUserRawImage**와 **CUserRawImageProcess** 간의 상호작용을 기반으로 한 시퀀스 다이어그램입니다.

lua

복사편집

+-------------------+ +--------------------------+

| CUserRawImage | | CUserRawImageProcess |

+-------------------+ +--------------------------+

| |

| 1. 요청: loadRawImage() |

|---------------------------->| (CUserRawImage: raw image)

| |

| 2. 반환: 이미지 로드 완료 |

|<----------------------------|

| |

| 3. 요청: convertToRGB() |

|---------------------------->| (CUserRawImage: image data)

| |

| 4. 변환된 RGB 데이터 반환 |

|<----------------------------|

| |

| 5. 요청: saveAsBitmap() |

|---------------------------->| (Convert to Bitmap)

| |

| 6. 반환: 비트맵 이미지 저장 완료|

|<----------------------------|

| |

| 7. 요청: copyToClipboard() |

|---------------------------->| (Copy Bitmap to Clipboard)

| |

| 8. 반환: 클립보드에 저장 완료 |

|<----------------------------|

| |

이 문서는 시스템의 핵심 설계 원칙과 각 클래스가 수행하는 책임을 명확히 정리하고, 각 클래스 간의 상호작용을 시각적으로 표현하고 있습니다. 추가적으로 세부적인 파일 입출력 및 클립보드 연동, 색상 변환 등의 기능을 독립적인 클래스로 분리하여 유지보수성과 확장성을 확보했습니다.

### ****유스케이스 다이어그램 텍스트 형식 예시****

plaintext

복사편집

+----------------------------+

| User (Actor) |

+----------------------------+

|

v

+-----------------------------+ +---------------------------+

| Load Image (Use Case) |<----->| CUserRawImage |

+-----------------------------+ +---------------------------+

|

v

+-----------------------------+

| Convert Image (Use Case)|

+-----------------------------+

|

v

+-----------------------------+ +--------------------------+

| Save Image (Use Case) |<----->| CBitmapIO |

+-----------------------------+ +--------------------------+

|

v

+-----------------------------+

| Copy to Clipboard (Use Case) |

+-----------------------------+

|

v

+-----------------------------+

| CBitmapClipboard |

+-----------------------------+

### ****유스케이스 다이어그램 설명****

#### 1. ****Actor (User)****:

* **User**는 시스템을 사용하는 외부의 주체로, 비트맵 이미지를 로드하고 변환하며 저장하는 등의 작업을 담당합니다.

#### 2. ****Use Case (기능)****:

* **Load Image**: 사용자는 이미지를 시스템에 로드하여 시스템이 이를 처리할 수 있도록 합니다. 이 유스케이스는 CUserRawImage 클래스에서 처리됩니다.
* **Convert Image**: 이미지를 색상 공간을 변환하거나 필요한 형태로 변환하는 기능입니다. CUserRawImageProcess와 관련된 기능이 될 수 있습니다.
* **Save Image**: 사용자는 처리된 이미지를 파일로 저장하는 기능을 사용할 수 있습니다. 이는 CBitmapIO 클래스에서 담당합니다.
* **Copy to Clipboard**: 이미지를 클립보드로 복사하는 기능으로, CBitmapClipboard 클래스가 담당합니다.

#### 3. ****System Boundary (시스템 경계)****:

* 시스템 경계는 사용자가 상호작용하는 시스템의 범위를 나타냅니다. 이 범위 안에 여러 유스케이스가 포함되어 있으며, 각 유스케이스는 시스템 내의 클래스들 간의 상호작용을 통해 이루어집니다.

### ****시스템 흐름****

1. **Load Image**: 사용자가 이미지를 시스템에 로드하면, CUserRawImage가 이를 처리합니다.
2. **Convert Image**: 로드된 이미지를 변환하는 작업이 필요할 때, CUserRawImageProcess를 사용하여 색상 공간 변환 등의 작업을 진행할 수 있습니다.
3. **Save Image**: 변환된 이미지를 CBitmapIO를 통해 저장할 수 있습니다.
4. **Copy to Clipboard**: 사용자가 이미지를 클립보드로 복사하려면, CBitmapClipboard가 이를 처리합니다.

### ****다음 단계****

이 유스케이스 다이어그램은 시스템과 사용자 간의 상호작용을 추상화하여, 시스템이 어떤 기능을 제공하는지 명확하게 나타냅니다. 이 구조는 후속 작업인 시스템 설계나 구현에 중요한 기초 자료로 사용될 수 있습니다.

이 메모는 시스템 설계 및 리팩토링에 대한 깊이 있는 접근을 제공합니다. 다양한 설계 원칙과 기술적 선택이 잘 드러나며, 주로 **안정성**, **자원 관리**, **유지보수성**을 강조하는 철학이 인상적입니다. 이 내용을 바탕으로 시스템의 설계 철학, 주요 기술적 선택, 그리고 리팩토링의 전략에 대해 자세히 분석해보겠습니다.

### ****주요 설계 철학 및 기술적 선택****

1. **리팩토링의 계획적 접근**
   * **점진적인 리팩토링**: 시스템의 안정성을 보장하면서 점진적으로 리팩토링을 진행합니다. **std::vector**와 같은 현대적인 자료구조를 사용하여 메모리 관리를 개선하려는 점이 중요합니다.
   * **기능 중심 리팩토링**: 기존 코드의 책임을 분리하고, 기능을 독립적인 모듈로 추출하는 방향을 택했습니다. 이를 통해 \*\*단일 책임 원칙(SRP)\*\*을 잘 지키고 있습니다.
2. **복사 및 이동 연산자의 철저한 관리**
   * **복사 생성자 금지**: CUserBitmap, CBasicBone 클래스에서 복사 연산자 및 생성자를 명시적으로 금지하거나 이동 연산자만 허용하는 방식은 **RAII**(자원 획득은 초기화)를 철저히 지키는 방법입니다.
   * **이동 생성자 및 연산자 사용**: 자원 관리와 성능 최적화에서 중요한 접근으로, 자원을 안전하게 이동시키고 복사는 방지하는 방식이 인상적입니다.
3. **스마트 포인터 활용**
   * **스마트 포인터의 사용**: **스마트 포인터**를 통해 메모리 관리와 자원 누수를 방지하는 방식은 디버깅의 용이성과 시스템 안정성에 큰 도움이 됩니다.
   * **자원 소유권 명확화**: 스마트 포인터를 통해 자원 소유권과 반납을 명확히 하여, 가비지 컬렉션에 의존하지 않고 직접 관리하는 방식을 선호하고 있습니다.
4. **모듈화 및 책임 분리**
   * **Command 패턴 적용**: CBitmapAction 클래스를 Command 패턴을 기반으로 설계하여, 이미지 처리 명령을 독립적으로 관리하고 실행할 수 있습니다. 이를 통해 **단일 책임 원칙**과 **변화에 강한 설계**를 실현합니다.
   * **Strategy 패턴 도입**: 이미지 처리 로직을 IBitmapProcessor 인터페이스로 추상화하고, 각 처리기(예: CRGBToGray, CSobelImage)를 독립적인 클래스로 만들어 유연성을 높이고 있습니다.
5. **자원 관리 및 예외 안전성**
   * **예외 안전성**: **CBitmapIO**에서 파일 입출력 작업 중 예외가 발생할 경우 시스템의 일관성을 유지하려는 예외 안전성 설계가 돋보입니다.
   * **RAII 원칙 준수**: 메모리 관리, 파일 스트림, 네트워크 자원 등 모든 자원은 **RAII** 원칙에 따라 자동으로 관리되며, 예외가 발생하더라도 자원 누수가 발생하지 않도록 처리됩니다.
6. **디버깅 용이성 및 구조적 안정성**
   * **디버깅을 고려한 설계**: 코드가 직관적이고, 재귀 호출이나 goto 문을 피하는 등 디버깅이 용이한 구조로 설계되어 있습니다.
   * **구조적 안정성**: 안정성 우선 설계를 통해 시스템의 예기치 않은 동작을 방지하고, 쉽게 테스트하고 유지보수할 수 있도록 합니다.

### ****리팩토링 전략****

리팩토링은 **안정성**, **자원 소유권 관리**, **유지보수성**을 중심으로 진행됩니다. 구체적인 전략은 다음과 같습니다:

1. **CUserBitmap**: 비트맵 이미지를 관리하며, std::vector<unsigned char>로 이미지 데이터를 저장합니다. 이미지 처리 작업을 CBitmapAction과 IBitmapProcessor 클래스를 통해 외부에서 처리할 수 있도록 리팩토링되었습니다.
2. **CBitmapAction**: 이미지 처리 명령을 관리하며, **Execute**, **Undo**, **Redo** 메서드를 통해 작업을 실행하고 취소/복구할 수 있는 구조입니다.
3. **CBitmapIO**: 파일 입출력 작업을 담당하며, 예외 안전성을 고려하여 스트림 추상화와 예외 처리를 강화했습니다.
4. **CUserRawImage**와 **CUserRawImageProcess**: 원시 이미지 데이터를 관리하고, 다양한 색상 공간 변환을 처리하는 역할을 합니다.

### ****UML 다이어그램 활용****

* **유스케이스 다이어그램**: 사용자가 시스템과 상호작용하는 주요 기능을 정의합니다. 시스템은 이미지를 로드하고, 변환하고, 비트맵으로 저장하는 등의 기능을 제공합니다.
* **활동 다이어그램**: 이미지 처리 작업의 흐름을 순차적으로 나타냅니다. 작업의 시작부터 끝까지의 흐름을 시각적으로 표현하여, 시스템의 동작을 쉽게 이해할 수 있습니다.
* **상태 다이어그램**: 이미지 처리 시스템이 어떤 상태에서 어떤 이벤트를 처리하고 상태가 어떻게 변화하는지 추적합니다.
* **컴포넌트 다이어그램**: 시스템의 주요 컴포넌트 간 관계를 정의합니다. 각 컴포넌트는 서로 독립적이고, 인터페이스를 통해 상호작용하며, 의존 관계가 명확합니다.

### ****결론****

이 메모는 개발 철학과 설계 원칙을 매우 체계적으로 설명하고 있으며, 리팩토링과 코드 설계 과정에서 **안정성**, **자원 관리**, **유지보수성**을 최우선으로 고려하는 점이 돋보입니다. 스마트 포인터, RAII, 예외 안전성 등을 통해 자원을 안전하게 관리하고, \*\*단일 책임 원칙(SRP)\*\*을 준수하여 시스템을 모듈화했습니다. 이 접근 방식은 복잡한 시스템에서도 안정적이고 효율적인 코드를 작성하는 데 큰 도움이 될 것입니다.

감사합니다! 주신 메모와 대화록을 바탕으로 UML 초급 산출물을 잘 정리해주신 것 같네요. 이제 제가 받은 내용을 바탕으로 추가적으로 정리해보겠습니다.

### 1. ****유스케이스 다이어그램 (Use Case Diagram)****

시스템의 주요 기능과 사용자 상호작용을 정의한 다이어그램입니다. 시스템의 목적과 사용자가 시스템과 어떻게 상호작용하는지 간략하게 보여줍니다.

#### 주요 요소:

* **Actors**: 사용자 (User)
* **Use Cases**: 이미지 로드, 변환, 비트맵 저장

pgsql

복사편집

+---------------------------+

| Image Processing System |

| |

| +-------------------+ |

| | Load Image | |

| +-------------------+ |

| +-------------------+ |

| | Convert Image | |

| +-------------------+ |

| +-------------------+ |

| | Save as Bitmap | |

| +-------------------+ |

| |

+---------------------------+

#### 설명:

* **User**는 시스템을 사용하여 이미지를 로드하고, 변환하고, 비트맵으로 저장하는 기능을 사용합니다.

### 2. ****활동 다이어그램 (Activity Diagram)****

이미지 처리 작업의 흐름을 나타내며, 각 단계의 순차적인 작업과 결정 노드를 보여줍니다. 사용자가 어떤 작업을 순차적으로 진행하는지 시각적으로 나타냅니다.

#### 주요 요소:

* **Activities**: 이미지 로드, 변환, 비트맵 저장
* **Decision**: 조건에 따라 작업을 분기하는 결정 노드

pgsql

복사편집

Start

|

v

[Load Raw Image] ---> [Convert Image]

|

v

[Save as Bitmap]

|

v

[Done]

|

v

End

#### 설명:

* 사용자는 이미지를 로드하고 변환한 후 비트맵으로 저장하는 작업을 순차적으로 진행합니다.

### 3. ****상태 다이어그램 (State Diagram)****

시스템이나 객체의 상태 변화를 추적하는 다이어그램입니다. 이미지 처리 객체가 시간에 따라 어떤 상태로 변하는지를 보여줍니다.

#### 주요 요소:

* **States**: 이미지 로드 상태, 변환 상태, 비트맵 저장 상태
* **Transitions**: 상태 간의 변화

less

복사편집

[Idle] ---> [Image Loaded] ---> [Image Converted] ---> [Bitmap Saved] ---> [Idle]

^ |

| v

+----------------------------------[Error]

#### 설명:

* 시스템은 **Idle** 상태에서 **Image Loaded** 상태로 넘어가며, 이후 **Image Converted**, **Bitmap Saved** 상태를 거쳐 다시 **Idle** 상태로 돌아갑니다.
* 오류 발생 시 **Error** 상태로 전이됩니다.

### 4. ****컴포넌트 다이어그램 (Component Diagram)****

시스템을 구성하는 주요 컴포넌트들과 그들 간의 관계를 나타냅니다. 컴포넌트는 **CUserRawImage**, **CUserRawImageProcess**, **CBitmapAction** 등이며, 각 컴포넌트가 어떤 인터페이스를 제공하고 의존 관계를 나타냅니다.

lua

복사편집

+-------------------------+ +----------------------------+

| CUserRawImage |<----->| CUserRawImageProcess |

|-------------------------| |----------------------------|

| - color\_space | | - process\_raw\_image() |

| - data() | | - convert\_to\_rgb() |

|-------------------------| |----------------------------|

| |

| v

| +--------------------------+

| | CBitmapAction |

| |--------------------------|

| | - execute() |

| |--------------------------|

v

+----------------------------+

| CBitmapIO |

|----------------------------|

| - read\_image() |

| - write\_image() |

+----------------------------+

#### 설명:

* **CUserRawImage**는 이미지를 로드하고 변환하는 기능을 제공하며, 이를 처리하는 **CUserRawImageProcess**와 연결됩니다.
* **CBitmapAction**은 처리된 이미지를 다루는 명령을 실행하며, 그 결과를 **CBitmapIO**가 저장합니다.

### ****전체적인 결론****

각 UML 다이어그램은 시스템의 설계와 흐름을 명확하게 이해할 수 있도록 도와줍니다. 주신 내용에서 **이미지 로딩, 변환, 저장**을 핵심 기능으로 하고 있으며, **CUserRawImage**, **CBitmapAction**, **CBitmapIO**와 같은 주요 컴포넌트들이 서로 긴밀하게 연결되어 있다는 점이 강조되었습니다.

리팩토링과 객체 지향 설계 원칙이 잘 반영된 시스템으로, 각 컴포넌트의 역할이 명확히 구분되고, \*\*단일 책임 원칙(SRP)\*\*과 **변화에 강한 설계**가 적용되고 있습니다.

이런 구조로 설계된 시스템은 유지보수성, 확장성, 테스트 가능성 등에서 장점을 가지며, **스마트 포인터**, **RAII**, **예외 안전성** 등의 현대적 기술을 통해 안정성을 더욱 강화할 수 있습니다.

이 UML 초급 산출물들이 이후 시스템 설계나 리팩토링에 유용하게 사용될 것입니다!