### 개발자 철학 및 구조적 설계 원칙 요약

#### 🔹 핵심 철학

* **안정성 최우선**: 모든 설계와 구현의 우선순위는 시스템 안정성에 있음. 실패보다는 예외, 예외보다는 안전한 구조 선호.
* **자원은 직접 관리**: 메모리는 사용 즉시 반납. 가비지 컬렉션은 신뢰하지 않으며, 자원 소유권을 명확히 함.
* **복사와 이동은 명시적으로**: 복사 생성자/대입, 이동 생성자/대입은 반드시 명시적으로 구현. 기본 제공 방식을 지양.
* **단일 목적 원칙(SRP)**: 클래스, 함수, 변수는 단 하나의 명확한 목적만 수행해야 함. 다목적 구조는 금지.
* **전역 상태 최소화**: 전역 변수, 전역 함수는 지양. 네임스페이스 사용으로 충돌 방지.

#### 🔹 구조 설계 원칙

* **CBasicBone 구조 기반**: 모든 클래스는 CBasicBone으로부터 파생되어 안정성과 일관성을 확보. 복사/이동 금지 원칙 내장.
* **처리와 표현 분리**: 예: CUserBitmap(데이터) vs CBitmapProcess(처리), CUserRawImage vs CUserRawImageProcess 구분.
* **자료구조 대체 유연성 확보**: std::vector, 커스텀 CMatRix 등을 통해 배열/동적할당 구조 대체. .data()로 하위 호환성 고려.
* **함수 객체 활용**: 단일 기능 함수 객체 구조로 확장성 강화. 이후 메뉴 기반 확장 및 유지보수 용이.
* **명명 규칙 명확화**: 이름만 보고도 역할과 책임이 드러나도록 네이밍. 자료형 기반 접두사 사용(헝가리안 표기법 등).

#### 🔹 구현 전략 요약

* **RAII 철저 적용**: 생성자에서 자원 획득, 소멸자에서 자원 반납.
* **예외 안전성 확보**: 실패 시 롤백 가능 구조 적용. 무조건적인 void 반환 대신 bool 등으로 실패 명시.
* **테스트 이전에도 신뢰 가능한 구조**: .data() 등 표준 접근 방식 고려하여 미리 안정성 확보.

#### 🔹 철학적 관점

* **코드는 자기 문서화되어야 한다**: 문서보다 코드 자체에서 의미가 드러나야 함.
* **죽더라도 조용히, 혼자 죽을 것**: 시스템 영향 없이 실패하는 구조.
* **시스템은 내가 제어한다**: 운영체제의 자원 회수에 의존하지 않음. 내 코드가 정리한다.
* **버그는 실패보다 나쁘다**: 동작하지 않는 것이 차라리 나음.
* **실용성과 미학의 균형**: 구조는 기능적이면서도 미학적으로 일관되어야 한다.

#### 🔹 상징 구조들

* CBasicBone: 모든 객체 구조의 근간. 복사/이동 금지, 안정성과 자원 관리 전담.
* CUserBitmap, CUserRawImage: 데이터 보관 책임. 구조 교체를 고려한 유연한 인터페이스 제공.
* CBitmapProcess, CUserRawImageProcess: 처리 책임. 실제 기능 실행 책임 부여. 기능은 독립된 함수 객체로 분리 가능.
* CMatRix: 벡터 기반 사용자 정의 행렬 자료구조. .data() 등 호환성 고려된 구조.

※ 이 문서는 설계 문서가 아닌 철학 메모입니다.