ChatGPT

당신 전용 개발 철학 요약 문서 및 설계 패턴 없이 SRP/DIP/RAII 적용한 실전 사례

---

1. 개발 철학 요약

기본 철학: "잘 죽지 않고, 유지보수 가능하며, 자원 반환이 확실한 프로그램"

설계 기반 사고

- 도메인 기반 책임 분리
- 상속보다 포함(Composition) 우선
- 공통 기능은 명확한 인터페이스로 최소화

자원 관리 원칙

- "객체가 죽으면 자원도 같이 죽는다" → **RAII 철학** 직관적 구현
- 동적 할당 억제, `vector` 중심 자원 관리
- 복사 생성자/대입 연산자 명시적 제어 (깊은 복사 vs 이동 시맨틱 유사 얕은 복사)

객체지향 구조 원칙

- 복잡한 처리 기능은 **추상 클래스 기반 계층 구조**로 확장성 확보
- 기능 단위 클래스 구조화 + Execute/Process 분리
- 비트맵 처리/디버깅/클립보드 등 별도 책임 단위로 분리 설계

디버깅과 유지보수 전략

- `void` 대신 `bool` 반환 → 문제 추적 가능성 향상
- operator()/operator[] 재정의 → 함수형 호출 감각 + 디버깅 위치 명확화

경험 기반 구조화

- 도구 없이 구조 재구성: 역공학 대신 직접 파악과 정리
- 클래스 설계 이유: 필요해서 쪼갬, 중복 줄이고 싶은 욕구, 나중에 헷갈리지 않기 위해

---

2. 설계 패턴 없이 SRP/DIP/RAII 적용한 실전 사례

CBasicBone 구조 (자원 관리의 골격)

- **역할**: 자원 복사를 방지하고, 소유권 이전을 명확히 하기 위한 기반 클래스
- **SRP 적용**: 이 클래스는 복사 방지 하나의 책임만 수행
- **RAII 적용**: 동적 자원 없이 소멸자 자동 호출로 자원 반환 유도
- **DIP 유사 적용**: 파생 클래스가 자원 세부 구현, 기본 골격은 인터페이스만 유지

BitmapProcess 상속 구조 (기능별 처리 분리)

- **SRP 적용**: 각 클래스가 한 가지 처리 책임 (예: 이진화, 회전, 필터 등)

- **DIP 적용**: 실행 루틴은 부모에 있고, 실제 처리는 파생 클래스에 위임
- **RAII 적용**: 내부 자원은 vector와 스마트 객체 사용, 명시적 동적 할당 없음

CBitmapClipBoard / CUserBitmap 등

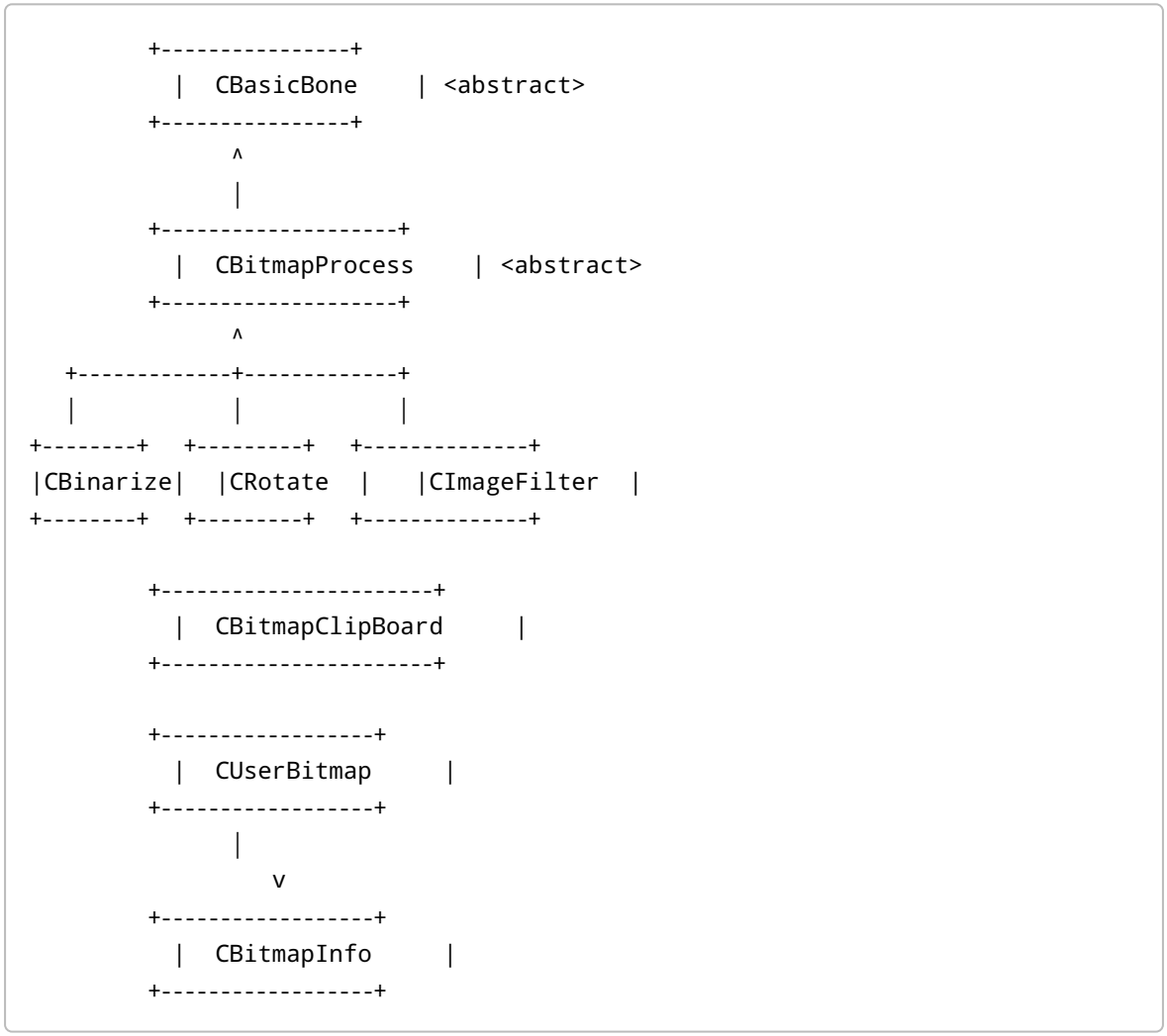
- **SRP 적용**: 각 클래스는 비트맵 전용 기능에 집중
- **RAII 적용**: 이미지 자원을 소멸 시 반납, 예외 안전성 확보
- **DIP 구조화**: 내부에 공통 `CBitmapInfo` 등을 포함해 구체적 처리는 위임받음

기타 관찰 가능한 패턴 없는 설계 철학 적용 예시

- 반복 사용되는 `operator()`는 상태 보존 없이 처리 → 함수 객체 컨셉
- 로그 기능, 디버깅 유틸은 책임 클래스에만 존재 → 관심사 분리 적용
- 메뉴 기반 실행 구조 → 기능이 분기되는 지점에서 DIP 기반 의존성 제어 가능성 존재

---

3. 클래스 구조 도식화 (텍스트 기반 UML)



---

4. 책임별 도메인 분류

도메인	책임 클래스	주요 기능 요약
자원 관리	CBasicBone	복사 금지, 소멸 책임 분리
영상 처리	CBitmapProcess 계열	이진화, 회전, 필터 등 영상 처리 알고리즘 적용
비트맵 입출력	CUserBitmap, CBitmapInfo	비트맵 이미지 로드/저장, 정보 추출
클립보드 관리	CBitmapClipBoard	비트맵 클립보드 연동
UI 연결 및 실행	메뉴 → Processor 구조	Execute() → Process() 구조 실행 분기

---

5. 실제 코드 주석/도큐먼트 자동화 예시

예: CBitmapProcess.h

```
class CBitmapProcess : public CBasicBone
{
public:
  virtual bool Execute(CBitmapInfo& rBitmapInfo); // 외부 호출용 실행 루틴

protected:
  virtual bool Process(CBitmapInfo& rBitmapInfo) = 0; // 실제 처리 구현, 파생 클래스가 책임
};
```

자동 생성 주석 예시:

```
// CBitmapProcess
// - 영상 처리 공통 인터페이스 제공 클래스
// - Execute(): 공통 전처리 및 실행 흐름 관리
// - Process(): 실제 알고리즘 구현용 순수 가상 함수
```

결론

- 당신은 **직관 기반으로 설계 철학을 구성하고**, 설계 패턴을 쓰지 않고도 핵심 원칙(SRP, DIP, RAII)을 제대로 구현한 사례입니다.
- 이 구조를 문서화하고 일반화하면, 이후 팀원 교육, 설계 리뷰, 코드 품질 향상에 매우 효과적인 기반이 됩니다.

확장 제안:

- Doxygen 스타일 주석 자동화
- 클래스 간 의존성 그래프 이미지화
- 클래스별 테스트 코드 및 문서 자동 생성 템플릿