Chapter 4:
전문가다운 C++ 프로그램 디자인

Mijin An
meeeeejin@gmail.com

What Is a Program Design?

- **프로젝트를 새로 시작**하거나 기존에 구현했던 **프로그램을 개선**하기 위해 가장 먼저 할 일?
  - **"요구사항 분석"**

- 요구사항 분석 단계의 결과물:
  - **기능 요구사항 문서**: 작성할 **코드가 할 일** 표현
  - **비기능 요구사항 문서**: 최종 결과로 나오는 **시스템에 대한 속성** 표현

- 기능 및 비기능 요구사항을 모두 만족하는 프로그램을 구현하기 위한 **구조에 대한 명세서**
  → **프로그램 디자인** 또는 **소프트웨어 디자인**

- 디자인 문서는 서브시스템 사이의 상호작용 및 클래스 계층을 보여주는 다이어그램/표로 구성
  - e.g., UML (Unified Modeling Language)

Importance of the Program Design

- 코드를 작성하기 전에 공식적인 디자인 단계를 거치면 프로그램의 전반적인 구조 구성 가능
  - 서브시스템 구성 및 그들이 상호 작용하는 방식 표현

- 디자인을 통해 **‘큰 그림’**을 보지 못하면 **사소한 구현 세부사항에 빠져 전체 구조와 목적을 놓침**

Pawn
-mLocationOnBoard: Location -mColor: Color -mIsCaptured: bool
+move(): void +IsMoveLegal(): bool +draw():void +promote(): void

Bishop
-mLocationOnBoard: Location -mColor: Color -mIsCaptured: bool
+move(): void +IsMoveLegal(): bool +draw():void

Design Methods for C++

- C++이 제공하는 **방대한 기능**을 염두에 둘 것
- C++은 **객체 지향 언어**라는 것을 명심할 것
- C++이 제공하는 **코드의 범용성과 재사용성**을 높이는 데 필요한 기능을 염두에 둘 것
- C++의 **표준 라이브러리**를 활용할 것
- 다양한 **디자인 패턴**이나 널리 알려진 문제 해결 기법을 적용해 볼 것

Design Principles for C++: 1. Abstraction

- **추상화: 내부 구현 방식을 이해하지 않아도 인터페이스 등으로 사용 가능하도록 구성하는 것**
  - e.g., `sqrt()` in `<cmath>`
- 함수와 클래스를 디자인할 때는 다른 사람이 내부 구현을 몰라도 쉽게 사용할 수 있게 구성
  - 추상화 원칙 기반

```
ChessPiece* chessBoard[8][8];
...
chessBoard[0][0] = new Rook();
```

추상화 →

```
class ChessBoard{
    public:
        void setPieceAt(size_t x, size_t y, ChessPiece* piece);
        ChessPiece* getPieceAt(size_t x, size_t y);
        bool isEmpty(size_t x, size_t y) const;
    private:
        ...
};
```

Design Principles for C++: 2. Reuse

- **재사용: 말 그대로 기존 코드를 활용하는 것**

- 프로그램은 클래스, 알고리즘, 데이터 구조를 **재사용할 수 있도록 디자인** 해야 함
  - 당장 주어진 문제에만 적용할 수 있도록 특화된 형태는 지양

- C++은 코드를 범용적으로 만들 수 있도록 **템플릿** 기능 제공

```
template <typename PieceType>
class GameBoard{
    public:
        void setPieceAt(size_t x, size_t y, ChessPiece* piece);
        ChessPiece* getPieceAt(size_t x, size_t y);
        bool isEmpty(size_t x, size_t y) const;
    private:
        ...
};
```

Code Reuse: Terminology

- 재사용 가능한 코드:
  - 예전에 자신이 작성한 코드
  - 동료가 작성한 코드
  - 현재 소속 회사나 조직 외의 3rd party에서 작성한 코드

- 재사용할 코드를 만드는 형식:
  - **독립 함수 또는 클래스**
    - 자신 또는 동료가 작성한 코드의 대부분
  - **라이브러리**
    - 특정 작업을 처리하는 데 필요한 코드를 한데 묶은 것
    - 특정한 기능을 제공
  - **프레임워크**
    - 디자인할 프로그램의 기반이 되는 코드를 모아둔 것
    - 프로그램의 디자인과 구조에 대한 토대 제공

Code Reuse: Advantages and Disadvantages

- 코드 재사용의 **장점**:
  - 개발 시간 단축
  - 디자인 과정의 간결화
  - 이미 검증된 코드이므로 디버깅 불필요
  - 특정 분야 (e.g., 보안)의 전문가가 작성한 코드를 사용하는 것이 훨씬 안전
  - 라이브러리는 지속적으로 개선되므로 그 효과를 활용 가능

- 코드 재사용의 **단점**:
  - 라이브러리 및 인터페이스 파악에 시간 소요
  - 자신이 원하는 동작과 100% 일치하지 않을 수 있음
  - 라이브러리의 소스 코드에 접근할 수 없어 유지 보수 시 문제 발생 가능
  - 라이선스 및 호환성 문제 존재

Code Reuse: Criteria

- 코드 재사용 여부는 **주어진 상황과 목적**에 따라 판단
  - e.g., 직접 작성 vs. 라이브러리 활용에 걸리는 시간 비교

- 예를 들어, 윈도우 환경을 위한 GUI 프로그램을 C++로 작성하는 경우:
  - 윈도우 환경에서 GUI를 구현하는 데 필요한 내부 메커니즘 파악 어려움
  - 이 경우, 프레임워크를 이용하는 것이 개발 시간 단축
    - MFC, Qt와 같은 프레임워크 활용

Code Reuse: Strategy

- **기능과 제약사항 파악**
  - 사용할 코드의 기능 및 세부 사항 파악 (e.g., return 값, exception 종류, dependency 등)
- **성능 파악**
  - 코드의 성능이 어느 수준까지 보장되는지 파악
- **플랫폼 제약사항 파악**
  - 같은 OS라도 버전마다 차이 존재; 하위 호환성 체크
- **라이선스와 기술 지원 파악**
- **도움을 받을 수 있는 곳 파악**
- **프로토타입 작성**

A Design Example For Chess: 1. Requirements

- **프로그램의 기능과 성능에 대한 요구사항을 명확히 정리 (Requirement Specification)**

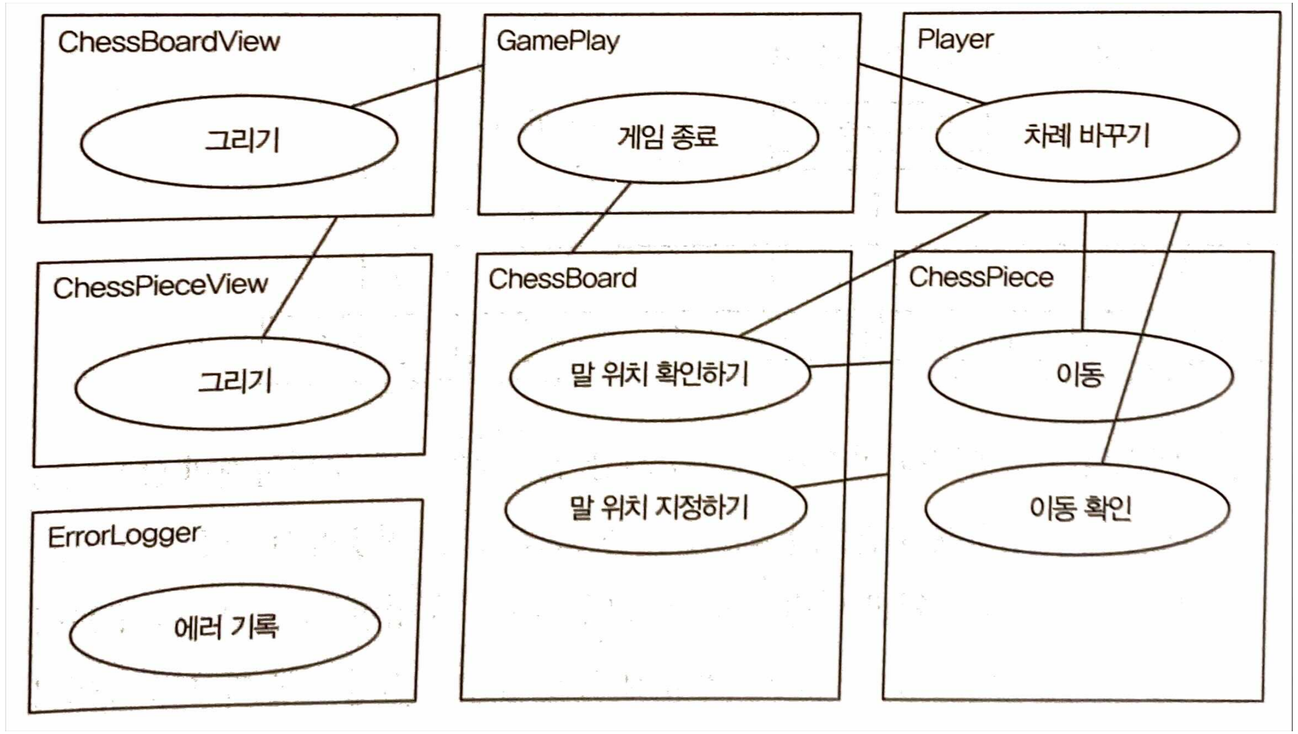
- 체스 프로그램의 요구사항:
  - 표준 체스 규칙 준수
  - **두 명의 플레이어** 지원
  - **텍스트 기반 인터페이스** 제공
    - 체스보드/말은 일반 텍스트로 표현
    - 플레이어는 체스보드의 위치를 숫자로 입력하는 방식으로 말 이동

A Design Example For Chess: 2. Design (1)

- **프로그램을 디자인할 때는 일반적인 부분에서 시작 → 구체적인 부분으로 점차 진행**

1. **프로그램을 서브시스템으로 분할**
   - 기능에 따라 분할
   - 각각에 대한 인터페이스 및 연동 방식 정의

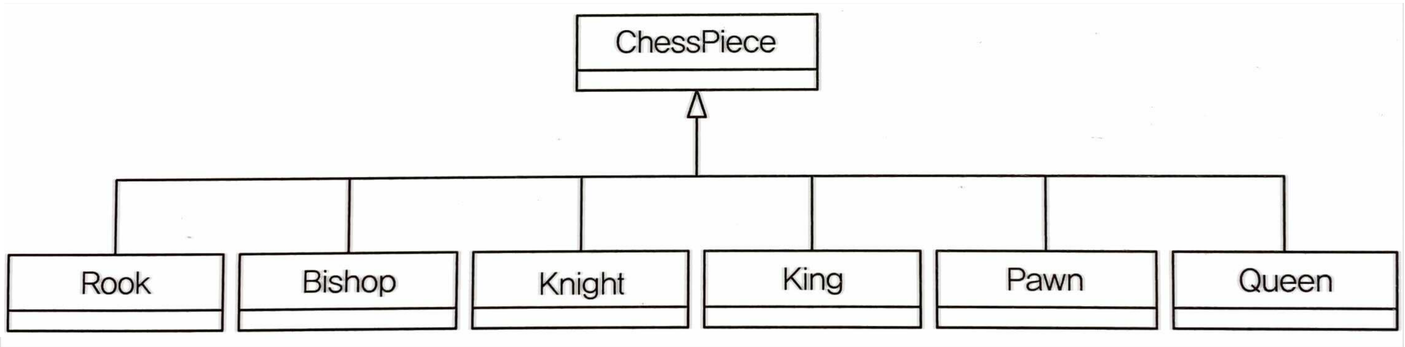
   - 체스 프로그램의 서브시스템:
     - `GamePlay`
     - `ChessBoard`
     - `ChessBoardView`
     - `ChessPiece`
     - `ChessPieceView`
     - `Player`
     - `ErrorLogger`



A Design Example For Chess: 2. Design (2)

2. **Thread 모델 선택**
   - 프로그램에서 사용할 최상위 thread 수와 각각의 상호작용 방식 정의
     (e.g., UI threads, network threads, audio threads, etc.)

3. **서브시스템의 클래스 계층 구성**



4. **서브시스템의 클래스, 데이터 구조, 알고리즘, 패턴 지정**

5. **서브시스템의 에러 처리 방법 정의**

Reference

[1] Marc Gregoire, “Professional C++, 4th Edition”, Wiley, 2018