C++ Korea

“Design Pattern”

Builder, Singleton

이 상 형

개요

참고 서적

- GoF 디자인 패턴! 이렇게 활용 한다.

- 문제 사례를 통해 디자인 패턴을 알아본다.

C++ Korea

Design Pattern

목차

01 최대 N개로 객체 생성을 제한하는 문제

02 부분 부분 생성을 통한 전체 객체 생성 문제

01 최대 N개로 객체 생성을 제한하는 문제

01 최대 N개로 객체 생성을 제한하는 문제

생성되는 객체 개수를 제한 하는 문제.

목표 : 생성되는 객체의 최대 개수를 제한하는 데 있어 객체의 생성을 요청하는 쪽에서는 일일이 신경쓰지 않아도 되도록 만들어 주는 것

01 문제 사례 설명

수비와 공격을 수행하는 유닛들로 구성된 게임.

생성 가능한 유닛의 최대 개수를 제한

01 다양한 접근 방법 및 Singleton 패턴

1. 전역 변수에 의한 객체 생성, 관리
2. Singleton 패턴

01 전역 변수에 의한 객체 생성, 관리

소스코드 : Singleton2.cpp

게임 유닛에 대한 생성 요청에 대해 전역 변수인 pUnitArray 배열 변수에 빈공간이 있을 때에만 게임 유닛을 생성해주는 방식

01 전역 변수에 의한 객체 생성, 관리

최대 N개의 객체만 생성되게 제한하기 위해 이와 같은 방식은 문제가 없을까?

01 전역 변수에 의한 객체 생성, 관리

최대 N개의 객체만 생성되게 제한하기 위해 이와 같은 방식은 문제가 없을까?

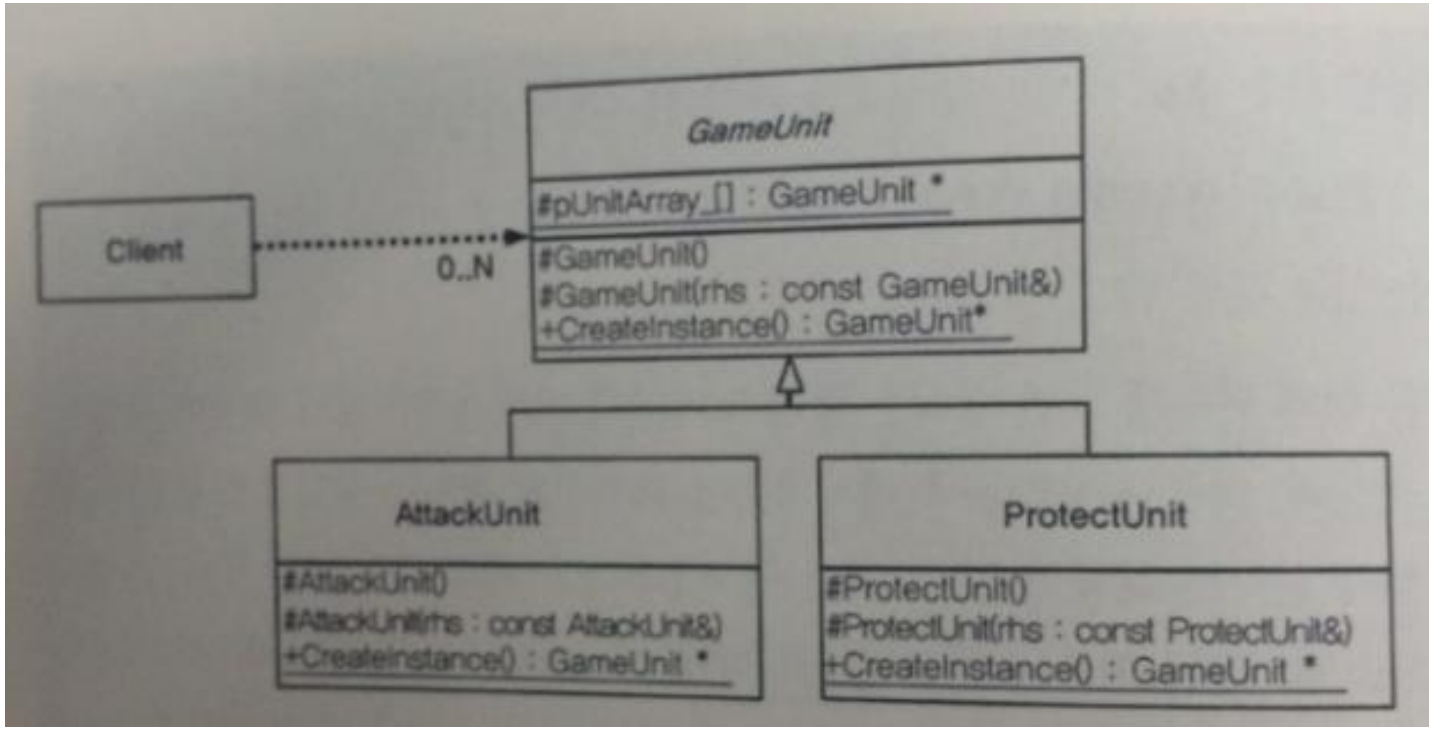
AttackUnit, ProtectionUnit 클래스 생성 가능.

01 Singleton 패턴

- 객체가 임의로 생성될 수 있는 경로 없애야함.
- 생성된 객체들은 전체적으로 관리 해야함.

01 Singleton 패턴

- 객체가 임의로 생성될 수 있는 경로 없애야함.
- 생성된 객체들은 전체적으로 관리 해야함.



01 Singleton 패턴

객체가 생성되는 개수를 제한하는 형태의 설계를 Singleton 패턴이라고 함.

01 Singleton 패턴

소스코드 : Singleton3.cpp

- 각 클래스의 생성자, 복사생성자 는 모두 protected

02 부분 부분 생성을 통한 전체 객체 생성 문제

02 부분 부분 생성을 통한 전체 객체 생성 문제

객체는 한번에 생성 된다 ?

```
Human* human = new Human();
```

02 부분 부분 생성을 통한 전체 객체 생성 문제

객체는 한번에 생성 된다 ?

```
Human* human = new Human();
```

아니다!

02 부분 부분 생성을 통한 전체 객체 생성 문제

실제 객체의 생성은
객체를 구성하는 데이터 멤버들을 일일이 생성하는 과정을 거침.

```
class Human
{
public:
    Human()
    {
        _arm = new Arm();
        _leg = new Leg();
        _face = new Face();
    }
private:
    Face* _face;
    Arm* _arm;
    Leg* _leg;
};
```

02 부분 부분 생성을 통한 전체 객체 생성 문제

실제 객체의 생성은
객체를 구성하는 데이터 멤버들을 일일이 생성하는 과정을 거침.

```
class Human
{
public:
    Human()
    {
        _arm = new Arm();
        _leg = new Leg();
        _face = new Face();
    }
private:
    Face* _face;
    Arm* _arm;
    Leg* _leg;
};
```

객체는 한 번에 생성되는 것이 아님

02 부분 부분 생성을 통한 전체 객체 생성 문제

때때로 클래스의 생성자를 불러서 객체를 생성 하는 것이 아니라
객체를 구성하는 부분 부분을 따로 생성하고
이를 조합해서 전체 객체를 생성해주는 것이 효율적일때가 있다.

02 부분 부분 생성을 통한 전체 객체 생성 문제

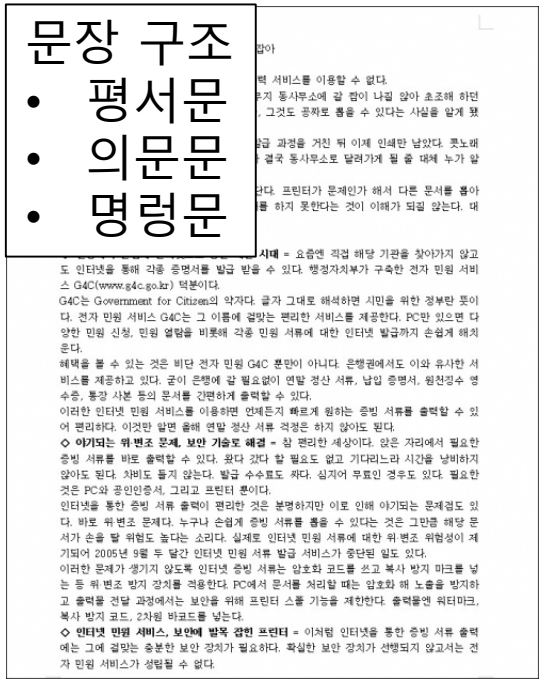
때때로 클래스의 생성자를 불러서 객체를 생성 하는 것이 아니라
객체를 구성하는 부분 부분을 따로 생성하고
이를 조합해서 전체 객체를 생성해주는 것이 효율적일때가 있다.

패킷 들어 왔을때 패킷을 생성 해야함.

- 패킷을 모았다가 한꺼번에 생성
- 패킷이 왔을때 바로 생성

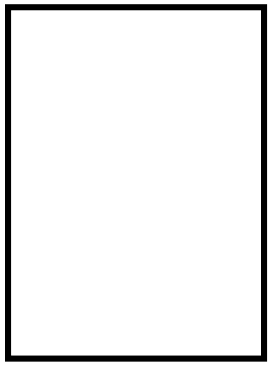
02 문제 사례 설명

문장 구조
- 평서문
- 의문문
- 명령문

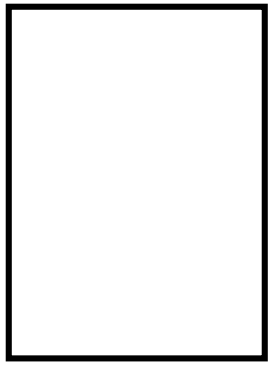


한국어 매뉴얼

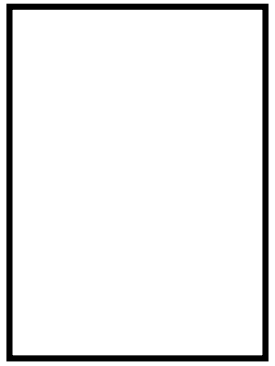
번역 소프트웨어



일어 메뉴얼



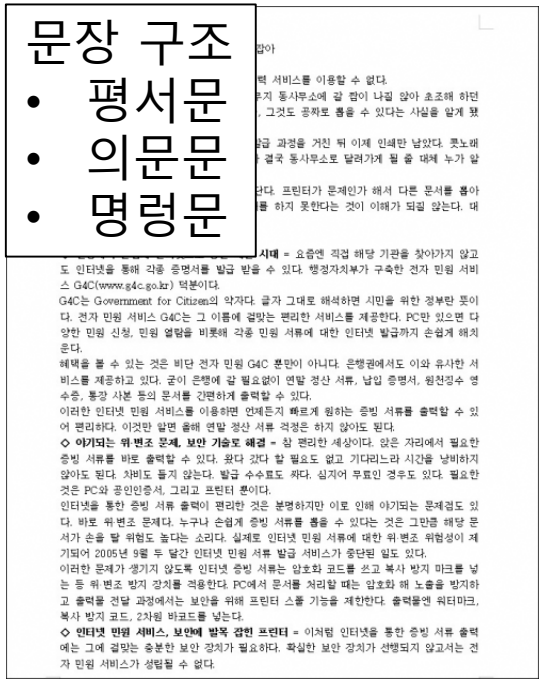
영어 매뉴얼



프랑스어 메뉴얼

02 문제 사례 설명

문장 구조
- 평서문
- 의문문
- 명령문



시대 = 요즘엔 직접 해당 기관을 찾아가지 않고도 인터넷을 통해 각종 증명서를 발급 받을 수 있다. 행정자치부가 구축한 전자 민원 서비스 G4C(www.g4c.go.kr) 덕분이다.

G4C는 Government for Citizen의 약자다. 글자 그대로 해석하면 시민을 위한 정부란 뜻이다. 전자 민원 서비스 G4C는 그 이름에 걸맞는 편리한 서비스를 제공한다. PC만 있으면 다양한 민원 신청, 민원 열람을 비롯해 각종 민원 서류에 대한 인터넷 발급까지 손쉽게 해치운다.

혜택을 볼 수 있는 것은 비단 전자 민원 G4C 분만이 아니다. 은행권에서도 이와 유사한 서비스를 제공하고 있다. 굳이 은행에 갈 필요없이 연말 정산 서류, 납입 증명서, 원천징수 영수증, 통장 사본 등의 문서를 간편하게 출력할 수 있다.

이러한 인터넷 민원 서비스를 이용하면 언제든지 **빠르게** 원하는 증빙 서류를 **출력할** 수 있어 편리하다. 이것만 알면 올해 연말 정산 서류 걱정은 하지 않아도 된다.

◇ **아기되는 위 변조 문제, 보안 기술로 해결** = 참 편리한 세상이다. 앉은 자리에서 필요한 증빙 서류를 바로 출력할 수 있다. 왔다 갔다 할 필요도 없고 기다리느라 시간을 낭비하지 않아도 된다. 차비도 들지 않는다. 발급 수수료도 **싸다**. 심지어 무료인 경우도 있다. 필요한 것은 PC와 공인인증서, 그리고 프린터 뿐이다.

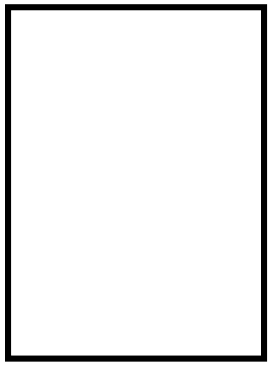
인터넷을 통한 증빙 서류 출력이 편리한 것은 분명하지만 이로 인해 야기되는 문제점도 있다. 바로 위 변조 문제다. 누구나 손쉽게 증빙 서류를 **뽑을** 수 있다는 것은 그만큼 해당 문서가 손을 탈 위험도 높다는 소리다. 실제로 인터넷 민원 서류에 대한 위 변조 위험성이 제기되어 2005년 9월 두 달간 인터넷 민원 서류 발급 서비스가 중단된 일도 있다.

이러한 문제가 생기지 않도록 인터넷 증빙 서류는 암호화 코드를 쓰고 복사 방지 마크를 넣는 등 위 변조 방지 장치를 적용한다. PC에서 문서를 처리할 때는 암호화 해 노출을 방지하고 출력을 전달 과정에서는 보안을 위해 프린터 스풀 기능을 제한한다. 출력물엔 워터마크, 복사 방지 코드, 2차원 바코드를 넣는다.

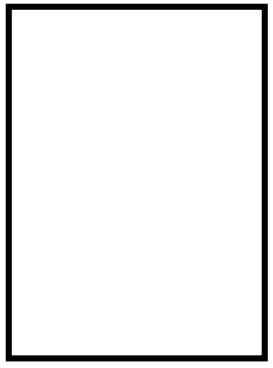
◇ **인터넷 민원 서비스, 보안에 발목 잡힌 프린터** = 이처럼 인터넷을 통한 증빙 서류 출력에는 그에 걸맞는 충분한 보안 장치가 필요하다. 확실한 보안 장치가 선행되지 않고서는 전자 민원 서비스가 성립될 수 없다.

한국어 매뉴얼

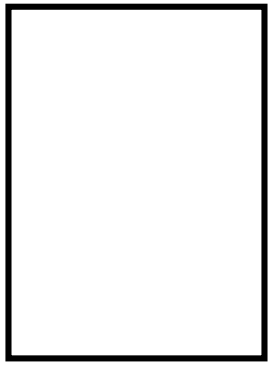
번역 소프트웨어



일어 메뉴얼



영어 매뉴얼



프랑스어 메뉴얼

문장 단위로 번역을 해서 문장을 조합 한다.

02 다양한 접근 방법 및 Builder 패턴

- 우리가 생성 해야 할 영어, 일본어, 프랑스어로 된 매뉴얼 각각 객체.
- 매뉴얼을 구성하는 개별문장을 단위로 객체의 부분 부분을 생성 한 후 이를 조합해주는 방식.
- 접근 방법
    1. 함수 형태 접근 방식
    2. 전담 클래스 활용 방식
    3. Builder 패턴

02 함수 형태 접근 방식

소스 코드 : Builder2.cpp

1. 한국어로 된 매뉴얼을 한 문장씩 읽어서 문장의 종류를 구분(평서문인지, 명령문인지, 의문문인지)
2. 구분된 문장 종류별로 각 국가의 언어로 변역시켜주는 함수를 작성해서 사용.

02 함수 형태 접근 방식

부분 부분 나누어서 객체(영어, 일본어, 프랑스 메뉴얼)를 생성하겠다는
원래의 의도를 수용하는데 큰 문제 없음.

02 함수 형태 접근 방식

부분 부분 나누어서 객체(영어, 일본어, 프랑스 메뉴얼)를 생성하겠다는
원래의 의도를 수용하는데 큰 문제 없음.

그러나

1. 요구사항의 추가, 변경 시 영향을 받는 범위가 불분명
2. 모듈의 재 사용 할 경우에도 그 재 사용할 모듈의 경계가 불확실

02 함수 형태 접근 방식

부분 부분 나누어서 객체(영어, 일본어, 프랑스 메뉴얼)를 생성하겠다는
원래의 의도를 수용하는데 큰 문제 없음.

그러나

1. 요구사항의 추가, 변경 시 영향을 받는 범위가 불분명
2. 모듈의 사용 시에도 재 사용할 모듈의 경계가 불확실

➔ 전담 클래스 활용 방식

02 전담 클래스 활용 방식

소스 코드 : Builder3.cpp

1. Director 클래스가 번역할 문장을 선별하는 역할을 수행
2. Translator 클래스는 문장의 종류별로 실제 번역을 수행하는 역할을 담당.

02 전담 클래스 활용 방식

소스 코드 : Builder3.cpp

1. Director 클래스가 번역할 문장을 선별하는 역할을 수행
2. Translator 클래스는 문장의 종류별로 실제 번역을 수행하는 역할을 담당.

요구 사항

1. 한국어 매뉴얼 중 아시아 지역에서만 유용한 문장은
영어나 프랑스어 매뉴얼 작성시에는 번역하지 않도록 요구 사항
2. 형태소 분석기 모듈을 새로 변경시켜 적용

02 전담 클래스 활용 방식

요구사항

중국어 매뉴얼을 만들어야 한다.

02 전담 클래스 활용 방식

요구사항

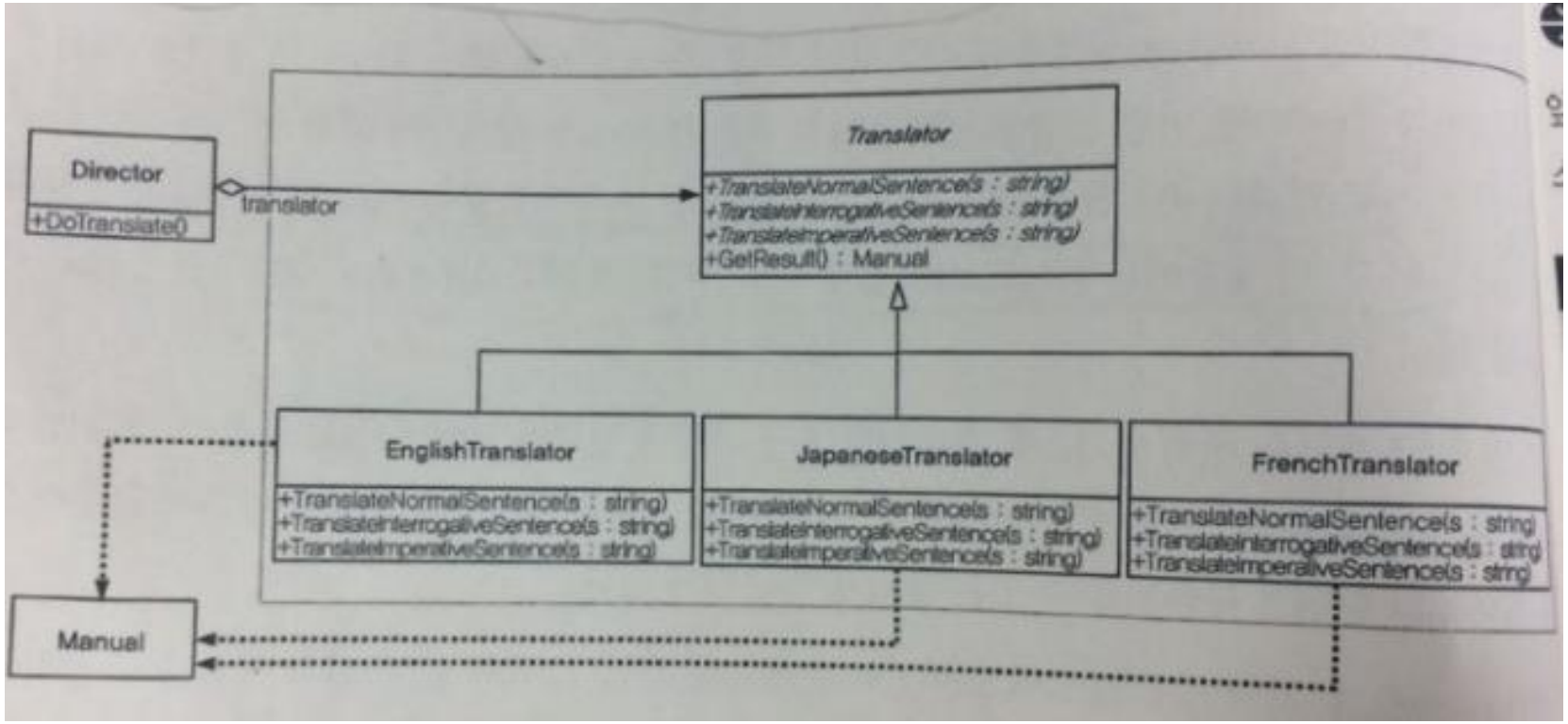
중국어 매뉴얼을 만들어야 한다.

02 전담 클래스 활용 방식

어떻게 접근해야 번역 대상 언어(중국어)의 추가와 같은 새로운 요구사항을
기존의 소스코드와는 독립적으로 쉽게 반영 할 수 있을까?

➔새로운 요구 사항을 기존의 설계와는
독립적으로 추가하기 위한 대표적인 도구는 클래스 상속이다.

02 전담 클래스 활용 방식



02 Builder 패턴

이처럼 객체를 생성하되, 그 객체를 구성하는 부분 부분 먼저 생성하고,
이를 조합함으로써 전체 객체를 생성하며, 생성할 객체의 종류가
손쉽게 추가, 확장이 가능하게 고안된 설계를 Builder라고 한다.

소스 코드 : Builder4.cpp

02 Builder 패턴

이처럼 객체를 생성하되, 그 객체를 구성하는 부분 부분 먼저 생성하고,
이를 조합함으로써 전체 객체를 생성하며, 생성할 객체의 종류가
손쉽게 추가, 확장이 가능하게 고안된 설계를 Builder라고 한다.

소스 코드 : Builder4.cpp

Thank U