

# Mi registro técnico de liquidación de deudas

Programación (26)    diario (4)    pensamientos (8)    libro (3)    hogar

## programación

# [42Seoul] Módulo 05 de CPP: reutilización y manejo de excepciones

Aggrodonk    2022. 9. 10. 22:59

## Introducción

✓ success

100 / 100

Solo – about 7 hours – 0 XP

subject.pdf

Rusty

youjeon's CPP Module 05

100%

This team was locked 5 days ago and closed 5 days ago

GIT REPOSITORY

git@vogsphere.42seoul.kr:vogsphere/intra-uuid-48f56e3b-7021-4752-b6b9-4e0e971060bc-4386840-youjeon

EVALUATIONS

Peer evaluations (2/2)

EVALUATED BY TAEHYKIM 5 DAYS AGO

100%

계속 c만 쓰다가 cpp module 평가가 두번째라 익숙치가 않지만 요즘에 자바를 공부해서 그런지 설명도 잘해주셔서 이해가 잘 되었습니다. !! 블로그를 잘보고 있습니다. 항상 도움 많이 받고 있습니다. ! 감사드립니다. !

YOUR FEEDBACK, 5 DAYS AGO:

다행히 기존에 cpp를 해보시거나 평가를 가보신적은 없지만 자바를 해보신 분이셔서 평가를 조금 수월하게 받을수있었습니다. 다만 제가 안드로이드 공부할때 자바를 잠깐 사용해보고 말아서 이게 자바의 어느 개념과 비슷한가요? 라는 질문에 말문이 막힌건 조금 아쉬웠던것 같습니다. 그래도 잘 설명이 되었다고 해주셔서 다행입니다. 평가 감사드립니다!

★★★★★ 4 / 4 🗨️ 4 / 4 📄 4 / 4 📁 4 / 4

EVALUATED BY PHILANTHROPIST SEONYUN 5 DAYS AGO

100%

객체 지향 언어인 C++를 라피신이나 본과정에서 처음 접하게 됐던 과제였습니다. 저는 아주 기본적인 객체 개념만 가지고 있고, 특히나 CPP Module 00이 아닌 기초 부분을 뛰어넘어선 중간 단계의 과제가 잡히게 돼서 과제를 잘 이해할 수 있을까 걱정되는 마음으로 평가에 임했습니다. 다행히 평가자 분이 C++의 가장 기초가 되는 개념부터 설명해 주셔서 차근차근 이해하려고 한 걸음 나아갈 수 있었고, 그래도 새롭게 등장하는 개념이 워낙 많아서 모두 소화하기는 어려웠지만 대강의 큰 틀을 파악할 수 있는 좋은 기회여서 유익했습니다. 다음 CPP 과제도 무사히 통과하셔서 마무리 잘 하시길 바라겠습니다~ 오랫동안 설명하느라 고생 많으셨습니다!

YOUR FEEDBACK, 5 DAYS AGO:

우선, 제가 이전 평가가 늦게 끝나서(사실 평가가 늦게끝나진 않았는데, 서버 이슈로 평가자가 입력되는걸 확인하다가 늦었습니다) 1~2분정도 늦었는데 흔쾌히 이해해주셔서 감사합니다. cpp나 다른 객체지향언어를 사용해보신적이 없었고 cpp평가도 처음이라고 하셔서 객체지향언어에 대한 기본적인 내용과 과제들이 요구하는 내용을 위주로 먼저 설명을 드리고 진행했습니다. 다음에 다른 cpp평가를 하실때나 직접 해당 과제를 하실때 도움이 되었으면 좋겠습니다. 고생하셨습니다! 좋은 평가 감사드립니다!

★★★★★ 4 / 4 🗨️ 4 / 4 📄 4 / 4 📁 4 / 4

Quizás debido a Hinnamno, fue el día en que Intra estaba a punto de saber mal.

Este es un desafío con respecto a la reutilización y el manejo de excepciones.

Puede considerarlo como una tarea para aprender a manejar excepciones mediante declaraciones try-catch y excepciones.

## ex00

Crear una clase burocrática para una enorme burocracia.

Debe tener un nombre constante y una calificación entre 1 y 150.

Si recibe una calificación incorrecta, debe manejar la excepción llamando a la función `Bureaucrat::GradeTooHighException` o `Bureaucrat::GradeTooLowException`.

Si ocurre un error en el constructor, maneje la excepción usando la misma función.

Necesitamos crear funciones `getName` y `getGrade` y también tener funciones para aumentar o disminuir la calificación. Como el grado 1 es el más alto, si subes el grado 3, pasa a ser 2.

Por sobrecarga `<<`, `<nombre>`, grado burocrático `<grado>`. Hazlo imprimible e implementa las pruebas necesarias.

```
class Bureaucrat {
private:
    const std::string name;
    int grade;
public:
    class GradeTooHighException : public std::exception
    {
        public:
            const char * what(void) const throw();
    };
    class GradeTooLowException : public std::exception
    {
        public:
            const char * what(void) const throw();
    };
};

const char * Bureaucrat::GradeTooHighException::what(void) const throw()
{
    return "Grade too high...";
}

const char * Bureaucrat::GradeTooLowException::what(void) const throw()
{

```

```

        return "Grade too low...";
    }

    Bureaucrat::Bureaucrat(std::string name, int grade) : name(name)
    {
        this->grade = grade;
        if (grade < 1)
        {
            throw Bureaucrat::GradeTooHighException();
        }
        else if (grade > 150)
        {
            throw Bureaucrat::GradeTooLowException();
        }
    }
}

```

En C++, las excepciones se lanzan con try... Procesado mediante declaración de captura y declaración de lanzamiento.

Si define una situación de excepción con anticipación y escribe los detalles del error en el retorno o de la función what() de la clase que hereda la clase de excepción, puede arrojar los detalles del error en la declaración try catch más cercana cuando ocurre una situación de excepción.

```

int main()
{
    try
    {
        try
        {
            Bureaucrat test("test", 200);
        }
        catch(const std::exception& e)
        {
            std::cerr << e.what() << '\n';
        }

        Bureaucrat a("a", 150);
        Bureaucrat b("b", 1);

        std::cout << a << std::endl;
        std::cout << b << std::endl;
        std::cout << std::endl;

        // a.decrementGrade();

        // b.incrementGrade();

        //std::cout << a << std::endl;
        //std::cout << b << std::endl;
        //std::cout << std::endl;

        a.incrementGrade();
        b.decrementGrade();
        std::cout << a << std::endl;
        std::cout << b << std::endl;
        std::cout << std::endl;
    }
}

```

```

    catch(const std::exception& e)
    {
        std::cerr << e.what() << '\n';
    }

    return (0);
}

```

A terminal window with a dark background. The prompt is 'ex00 git:(main) x ./a.out'. The program output is as follows:

```

Grade too low...
a, bureaucrat grade 150
b, bureaucrat grade 1

a, bureaucrat grade 150
b, bureaucrat grade 1

a, bureaucrat grade 149
b, bureaucrat grade 2

```

Si crea un constructor fuera de 1 ~ 150 en el código anterior, si no hay try catch en ninguna parte, se producirá una cancelación.

Si hay una declaración try catch en el archivo principal, el programa se detendrá después de que se genere el lanzamiento, pero si la situación de detención está envuelta en un try catch en otra función, la excepción solo se pasará al catch más cercano, por lo que el resto continuará después de que se genere la situación de error. Continúa.

Por lo tanto, en casos generales, se utiliza una declaración try catch en una unidad de trabajo (transacción). Si utiliza varias declaraciones try catch en una unidad de trabajo, tenga cuidado porque el trabajo que debe finalizar continúa o no se puede manejar el manejo de excepciones de otro trabajo.

La mayor ventaja de utilizar las declaraciones try catch y throw es que no tiene que preocuparse por cómo devolver una situación de error. Si el código anterior devuelve un error sin usar try catch, no tengo idea de cómo pasar y procesar el valor de retorno para que el lugar que llamó a la función sepa si ocurrió un error dentro de la función. Esto se puede hacer fácilmente simplemente usando el comando. tirar la palabra clave se puede procesar.

Otra ventaja es que el desenrollado de la pila se realiza automáticamente. Si maneja una situación de excepción sin usar try catch al ingresar una función dentro de una función varias veces, el valor de retorno especificado como situación de excepción debe continuar pasándose al lugar don

de se ejecutó la función por primera vez, pero esto también se realiza automáticamente. Sin embargo, dado que esta operación sólo funciona correctamente en el área de la pila, debe tener cuidado con las pérdidas de memoria de las variables declaradas como punteros.

<https://modoocode.com/230>

## Masticando C++ - <11. Manejo de excepci...

modoocode.com

<https://ansohxxn.github.io/cpp/chapter14-3/>

## C++ Capítulo 14.3: Clases de excepción y ...

Esta es una nota escrita después de escuchar la conferencia sobre C++ del profesor Hong Jeong-mo en Infrun. 😊 🤔 [Aprende C++ sig...

ansohxxn.github.io

<https://ansohxxn.github.io/cpp/chapter14-4/>

## C++ Capítulo 14.4: std::excepción

Esta es una nota escrita después de escuchar la conferencia sobre C++ del profesor Hong Jeong-mo en Infrun. 😊 🤔 [Aprende C++ sig...

ansohxxn.github.io

<https://wikidocs.net/229>

## 07-04 Manejo de excepciones

Al crear un programa se producen innumerables errores. Por supuesto, la razón por la que ocurren errores es la consideración de Java p...

wikidocs.net

<https://musket-ade.tistory.com/entry/C-Stack-UnwindingStack-Unwinding>

## [C++] Desenrollado de pila

En la publicación anterior, cuando se lanzaba un lanzamiento dentro de una función, la excepción se manejaba regresando al área dond...

mosquete-ade.tistory.com

## ex01

Crear formularios que sean procesados por burócratas.

Nombre constante, si se debe firmar (bool), grado constante requerido para firmar, grado constante requerido para la ejecución. Haga que estas variables miembro sean privadas.

De la misma manera que en la burocracia, si la calificación es demasiado alta o demasiado baja, trátelo como una excepción y también implemente la salida a la consola a través de <<.

Cree una función miembro beSigned que tome un burócrata como argumento y cambie el estado del signo a verdadero si la calificación del burócrata es mayor o igual a la calificación requerida por este formulario. Si es bajo, se maneja como una excepción.

Luego, agregue la función signForm() como función miembro de la burocracia e imprima el contenido en la consola cuando la firma tenga éxito o falle.

```
std::ostream& operator<<(std::ostream& out, const Form& f)
{
```

```

out << f.getName()
    << std::boolalpha <<" , signed : " << f.getSigned()
    << " , signGrade : " << f.getSignGrade()
    << " , execGrade : " << f.getExecGrade();
return (out);
}

```

```

b, bureaucrat grade 1
c, signed : true, signGrade : 50, execGrade : 50

```

Al generar el tipo de datos bool, solo se generarán 0 y 1. Para mejorar esto, agregue std::boolalpha a delante de la salida. Luego, se genera verdadero o falso según lo previsto.

<https://woodforest.tistory.com/92>

## [C++] salida bool - boolalpha

bool a = false; bool b = true; 일 때 cout< cout< 하면 0 1 이 출력됩니다 true/false로 출력하고 싶을땐 if/else를 쓰기 귀찮기도 하죠 c++에서는 방법을...

woodforest.tistory.com

## ex02

01에서 작성한 Form을 추상 클래스로 만들고, 그걸 상속받은 세 개의 클래스를 만들어서 실행할 수 있게 만들어라.(Form의 private 은 그대로 남겨둬야 한다.)

ShrubberyCreationForm - <target>\_shrubbery 파일을 만들어서 그 안에 아스키 트리를 넣어라

RobotomyRequestForm - 소음이 출력되고, <target>의 로봇화가 50% 확률로 성공했음을 알려라. 아니면 실패했음을 알려거나.

PresidentialPardonForm - Zaphod Beeblebrox가 <target>을 사면시켰음을 알려라.

모든 클래스의 생성자는 인자를 target(name) 하나만 받는다.

Form에 execute(Bureocrat const & executor) const를 추가하고 양식이 서명되어있으며, 실행하는 관료의 등급이 높거나 같은지를 확인하고 예외처리를 해야 한다.

각각 자식 클래스에서 위의 예외처리를 구현할지, 아니면 기본 클래스에서 다른 함수를 호출하여 확인할지는 사용자에게 달려있는데, 어느 한쪽이 더 좋긴 하다.

마지막으로 관료에게 executeForm(Form const & form) 멤버 함수를 만들고, 실행 후에 적절한 출력을 해라.

```
class ThisIsNotSignedException : public std::exception
{
    public:
        const char * what(void) const throw();
};
virtual void execute(const Bureaucrat& b) const = 0;
void setName(std::string name);
void setSigned(bool b);
void checkExec(const Bureaucrat& b) const;
```

Form 클래스의 execute를 순수 가상 함수로 만들어서 추상 클래스로 만들었다.

또한 Form의 private 멤버 변수들을 protected로 바꾸지 못하도록 되어있기 때문에, 변경이 필요한 멤버 변수를 변경하는 set 함수들을 만들어준다.

예외처리를 자식 클래스마다 따로 구현할지 아니면 Form에서 따로 함수를 만들어서 구현할 건지를 물어보는 데, 후자가 멤버 변수에 접근하기가 편하고 중복으로 구현할 필요가 없을 것 같아서 그렇게 구현하였다.

```
class ShrubberyCreationForm : public Form
{
    private:
        ShrubberyCreationForm(void);

    public:
        ShrubberyCreationForm(const ShrubberyCreationForm& obj);
        ShrubberyCreationForm& operator=(const ShrubberyCreationForm& obj);
        ~ShrubberyCreationForm(void);
        ShrubberyCreationForm(std::string target);
        void execute(const Bureaucrat& b) const;
};
```

그리고 나면 자식 클래스에서는 실행 부분만 구현을 해주면 된다. 다른 자식 클래스도 다 동일하게 생겼으니 생략.

ShrubberyCreationForm에서는 파일을 만들어서 그 파일에 일정한 문자열을 집어넣으면 된다. 파일을 만들거나 문자열을 만드는 과정은 01/ex04에서 이미 진행해봤을 테니 따로 설명하지는 않겠다.

안에 ASCII로 숲을 그려 넣으라길래 뭘 어떻게 해야 하나 감이 안 왔는데 그냥 구글에 ascii forest art라고 검색해서 나오는 적당한 문자열을 선택하면 된다.



[illegible]

하나의 문자열에 끝에 ‘\n’를 넣어서 계속 그리면 여러 줄을 가진 문자열을 하나의 string에 넣을 수 있다. 다른 방법도 있는데, 나는 왜인지 몰라도 내 맥에서는 다른 방법이 안돼서 이렇게 그렸다.

RobotomyRequestForm에서는 50%의 확률로 기계화에 성공하거나 실패하는 상황을 구현해야 한다. 다만 c++의 random은 c++11이기 때문에 사용할 수 없고, C의 rand를 가져와야 한다.

```
int tmp = (rand() % 2);

    std::cout << "drill say : drrrrrrrrrrrrr...!" << std::endl;

    if (tmp)
    {
        std::cout << "wow! " << this->getName() << " succeeded in becoming a robot" <<
std::endl;
    }
    else
    {
        std::cout << "sorry... " << this->getName() << " failed to becoming a robot" <
< std::endl;
    }
}
```

사실 한참 고민한 게 로봇화에 실패한 상황이 exception인지 좀 헷갈렸는데, 다른 사람들에게 물어보니 문서는 제대로 실행이 되었으니 이 관료사회에서는 그것도 성공이지 않겠냐고 해서 듣고 보니 맞는 말이라 그렇게 구현했다. 이걸 50%의 실패를 exception으로 구현했어도 이해할만한 부분인 것 같다.

```
void PresidentialPardonForm::execute(const Bureaucrat& b) const
{
    checkExec(b);

    std::cout << this->getName() << " was pardoned by President Zaphod Beeblebrox.." << std::endl;
}
```

PresidentialPardonForm에서는 따로 뭘 구현하지는 않고 그냥 문자열만 출력하도록 했다.

```
void Bureaucrat::signForm(Form& f) const
{
    try
    {
        f.beSigned(*this);
        std::cout << this->name << " signed " << f.getName() << std::endl;
    }
    catch(const std::exception& e)
    {
        std::cerr << this->name << " couldn't sign " << f.getName() << " because "<<
e.what() << std::endl;
    }
}
```

관료 클래스에서 try catch문을 만들어서 메인 문에서의 코드가 복잡하거나 중간에 중단되지 않도록 구현하였다.

<https://www.techiedelight.com/ko/create-a-multiline-string-literal-in-cpp/>

## C++에서 여러 줄 문자열 리터럴 만들기

이 게시물에서는 C++에서 여러 줄 문자열 리터럴을 만드는 방법에 대해 설명합니다. 1. 문자열 리터럴 사용 C++는 두 개 이상의 문자열 리터럴이 인접한 경...

[www.techiedelight.com](http://www.techiedelight.com)

<https://cplusplus.com/forum/beginner/79626/>

## How to do 50/50 chance of a number - C...

Sep 20, 2012 at 4:25pm UTC I am doing a Plinko program and when you drop a chip in, it has a 50/50 shot at either going left or right(w...

cplusplus.com

<https://blog.naver.com/njw1204/221079839989>

## C언어/C++ rand 함수(랜덤 함수) 사용시 주의...

시작하기 앞서 C++ 레퍼런스의 rand 함수 정보를 보고 갑시다. rand 함수는 0 이상 RAND\_MAX이하의 ...

blog.naver.com

## ex03

양식을 대신 만들어주는 인턴 클래스를 구현하라. 이름이나 성적 같은 잡다한 것은 필요 없고, makeForm()이라고 하는 멤버 함수를 가진다. 해당 함수의 인자로써 함수의 타입과 target의 이름을 가진다. 만약 양식 이름이 없다면 명시적 오류 메시지를 인쇄해야 한다.

흔한 if/elseif/else문의 범벅을 사용하지 않고 구현해야 한다.

```

Form* Intern::makeForm(std::string type, std::string name)
{
    std::string types[3] = {"shrubbery creation", "robotomy request", "presidential pardon"};

    for (int i = 0; i < 3; i++)
    {
        if (types[i] == type)
        {
            return (f[i]->clone(name));
        }
    }
    throw Intern::NoTypeException();
    return (NULL);
}

```

01/ex06에서 switch를 사용해보라며 비슷한 주제로 구현을 해본 적이 있는데, 그때처럼 함수 포인터를 사용하면 for문 하나안에 if문 하나를 넣어서 깔끔하게 만들 수 있다.

Algunas personas pueden implementarlo usando un interruptor, y otras como yo lo implementa n usando un for o un if. La implementación anterior en sí es mucho más fácil, pero la cantidad de trabajo cuando le diga al pasante que agregue un formulario adicional más adelante será menor. este último Lo pensé y lo implementé de esa manera. Sin embargo, al evaluarlo más tarde, se co mprobó si se utilizaba un puntero de función.

3

Suscribir

Otras publicaciones en la categoría '**Programación**'

[42Seúl] Módulo 07 del CPP - Plantilla (1)	2022.09.11
[42Seúl] Módulo CPP 06 - Conversión de tipo (1)	2022.09.11
[42Seoul] CPP Módulo 04 - Polimorfismo y clases abstractas (0)	2022.09.10
[42Seoul] Módulo CPP 03 - Herencia de clase (0)	2022.08.18
[42Seoul] Módulo CPP 02: creación de una clase de punto fijo (0)	2022.08.18

etiqueta

#42Seúl, #cpp

Artículos relacionados con 'Programación'

구독하기



alpha-traveler

2022.09.11 12:59 신고

댓글주소 수정/삭제 댓글쓰기

좋은 글 잘봤습니다 ㅎㅎ 다음에도 올려주세요 :)



원피스는 실존하다

2023.05.19 19:39 신고

댓글주소 수정/삭제 댓글쓰기

예외처리 클래스는 Orthodox Canonical Form 규칙 안 지켜도 상관없나요?



어그로동크

2023.07.31 14:41 신고

댓글주소 수정/삭제

아마 늦었겠지만 과제를 읽어보시면 Please note that exception classes don't have to be designed in Orthodox Canonical Form. 이라고 명시되어있습니다

nombre

contraseña

☐ Secreto

Por favor ingrese sus valiosos comentarios.

Deja un comentario

Mensajes recientes

[42Seúl] ft\_containers[4] - Mapa...

[42Seúl] ft\_containers[3] - TR...

[42 Seúl] ft\_containers[2] - Beck...

[42Seúl] ft\_containers[1] - S...

buscar

Por favor esc

visitantes totales

100.591

[42Seúl] ft\_containers[0] - y...

---

hoy	55
ayer	68