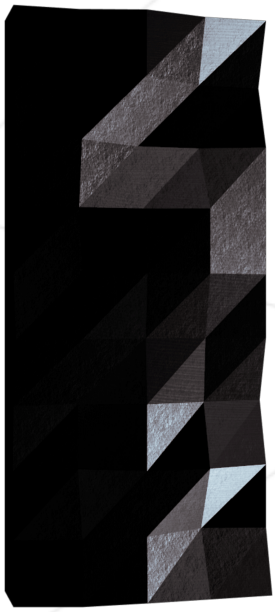

C++ - Módulo 05

Repetición y Excepciones

Resumen: Este documento contiene la evaluación del módulo 05 de los módulos C++ de 42.

Índice general

Capítulo I

Reglas Generales

- La declaración de una función en un header (excepto para los templates) o la inclusión de un header no protegido conllevará un 0 en el ejercicio.
- Salvo que se indique lo contrario, cualquier salida se mostrará en stdout y terminará con un newline.
- Los nombres de ficheros impuestos deben seguirse escrupulosamente, así como los nombres de clase, de función y de método.
- Recordatorio : ahora está codificando en `C++`, no en `C`. Por eso :
  - Las funciones siguientes están **PROHIBIDAS**, y su uso conllevará un 0: `*alloc`, `*printf` et `free`
  - Puede utilizar prácticamente toda la librería estándar. NO OBSTANTE, sería más inteligente intentar usar la versión para C++ que a lo que ya está acostumbrado en C, para no basarse en lo que ya ha asimilado. Y no está autorizado a utilizar la STL hasta que le llegue el momento de trabajar con ella (módulo 08). Eso significa que hasta entonces no se puede utilizar Vector/List/Map/etc... ni nada similar que requiera un include <algorithm>.
- El uso de una función o de una mecánica explícitamente prohibida será sancionado con un `0`
- Tenga también en cuenta que, a menos que se autorice de manera expresa, las palabras clave `using namespace` y `friend` están prohibidas. Su uso será castigado con un `0`.
- Los ficheros asociados a una clase se llamarán siempre `ClassName.cpp` y `ClassName.hpp`, a menos que se indique otra cosa.
- Tiene que leer los ejemplos en detalle. Pueden contener prerrequisitos no indicados en las instrucciones.
- No está permitido el uso de librerías externas, de las que forman parte `C++11`, `Boost`, ni ninguna de las herramientas que ese amigo suyo que es un figura le ha recomendado.
- Probablemente tenga que entregar muchos ficheros de clase, lo que le va a parecer repetitivo hasta que aprenda a hacer un script con su editor de código favorito.

- Lea cada ejercicio en su totalidad antes de empezar a resolverlo.
- El compilador es `clang++`
- Se compilará su código con los flags `-Wall -Wextra -Werror`
- Se debe poder incluir cada include con independencia de los demás include. Por lo tanto, un include debe incluir todas sus dependencias.
- No está obligado a respetar ninguna norma en `C++`. Puede utilizar el estilo que prefiera. Ahora bien, un código ilegible es un código que no se puede calificar.
- Importante: no va a ser calificado por un programa (a menos que el enunciado especifique lo contrario). Eso quiere decir que dispone cierto grado de libertad en el método que elija para resolver sus ejercicios.
- Tenga cuidado con las obligaciones, y no sea zángano; podría dejar escapar mucho de lo que los ejercicios le ofrecen.
- Si tiene ficheros adicionales, no es un problema. Puede decidir separar el código de lo que se le pide en varios ficheros, siempre que no haya moulinette.
- Aun cuando un enunciado sea corto, merece la pena dedicarle algo de tiempo, para asegurarse de que comprende bien lo que se espera de usted, y de que lo ha hecho de la mejor manera posible.

Capítulo II

Ejercicio 00: ¡Mamá, cuando sea mayor quiero ser burócrata!

	Ejercicio : 00
¡Mamá, cuando sea mayor quiero ser burócrata!	
Directorio de entrega : $ex00/$	
Ficheros a entregar : `Bureaucrat.hpp Bureaucrat.cpp main.cpp`	
Funciones prohibidas : `Ninguna`	

Hoy, vamos a crear una pesadilla artificial compuesta por despachos, pasillos, Forms y filas de espera.
Divertido, ¿eh? ¿No? Qué pena.
En primer lugar, vamos a colocar el engranaje más pequeño en la gigantesca máquina de la burocracia moderna: el `Bureaucrat`.

Debe tener un nombre (constant) y un rango, que va del 1 (el más alto) al 150 (el más bajo). Si se intenta crear un `Bureaucrat` con un rango que no sea válido se deberá lanzar una excepción que será una `Bureaucrat::GradeTooHighException` o una `Bureaucrat::GradeTooLowException`.

También tendrá que crea getters para los atributos (`getName` y `getGrade`), y dos funciones para incrementar/decrementar el rango. Atención: el rango 1 es el más alto, si lo baja obtiene el rango 2, etc.

Se deben poder capturar las excepciones con un bloque de código de este tipo:

```
try
{
        /* do some stuff with bureaucrats */
}
catch (std::exception & e)
{
        /* handle exception */
}
```

También tendrá que hacer un overload del operador << que mostrará algo parecido a esto: `<name>, bureaucrat grade <grade>`.

Por supuesto, entregará un `main` para demostrar que todo funciona.

Capítulo III

Ejercicio 01: ¡Formen filas, gusanos!

	Ejercicio : 01
¡Formen filas, gusanos!	
Directorio de entrega : $ex01/$	
Ficheros a entregar : `Lo mismo que antes + Form.cpp Form.hpp`	
Funciones prohibidas : `Ninguna`	

Ahora que ya tenemos burócratas, debemos darles algo que hacer. ¿Y qué hay mejor que una pila de Forms para que los rellenen?

Cree la clase `Form`. Tiene un nombre, un booleano que indica si está firmada (al principio, no lo está), un rango requerido para firmarla y un rango requerido para ejecutarla. El nombre y el rango son constantes y los atributos son todos privados (y no están protegidos). Los rangos están sometidos a los mismos requisitos que `Bureaucrat` y si alguno de ellos se encuentra fuera de los límites se lanzarán excepciones con `Form::GradeTooHighException` y `Form::GradeTooLowException`.

Igual que anteriormente, cree getters para todos los atributos y un overload del operador << a `ostream` que describa completamente el estado del Form.

Añada también una función `beSigned` que reciba un `Bureaucrat` y firme el Form si el rango del burócrata es lo suficientemente alto. Recuerde que el rango 1 es más alto que el rango 2. Si el rango es demasiado bajo, lance una `Form::GradeTooLowException`.

También añada una función `signForm` a `Bureaucrat`. Si se ha conseguido firmar mostrará algo parecido a `«bureaucrat>signs <form>"`, si no mostrará algo parecido a `«bureaucrat>cant sign <form>because <razón>"`.

Añada lo que haga falta a su `main` para probar que todo funciona.

Capítulo IV

Ejercicio 02: Necesita el Form 28B, no el 28C...

	Ejercicio : 02
Necesita el Form 28B, no el 28C...	
Directorio de entrega : $ex02/$	
Ficheros a entregar : `Lo mismo que antes + ShrubberyCreationForm.[hpp,cpp] RobotomyRequestForm.[hpp,cpp] PresidentialPardonForm.[hpp,cpp]`	
Funciones prohibidas : `Ninguna`	

Ahora que ya tenemos Forms básicos, vamos a crear algunos Forms que hagan algo de verdad.

Cree los siguientes Forms específicos:

- `ShrubberyCreationForm` (Rangos requeridos: firma 145, ejecución 137). Acción: Crea un archivo que se llama <target>_shrubbery y dentro dibuja árboles con ASCII, en el directorio actual.
- `RobotomyRequestForm` (Rangos requeridos: firma 72, ejecución 45). Acción: Hace ruidos de taladradora y comunica que <target>ha sido correctamente robotomizado el 50 % de lo casos. El resto del tiempo, comunica que ha fracasado.
- `PresidentialPardonForm` (Rangos requeridos: firma 25, ejecución 5). Acción: Nos comunica que <target>ha sido perdonado por Zafod Beeblebrox.

Estos elementos recibirán un único parámetro en su constructor, que corresponderá al objetivo del Form. Por ejemplo, çasa"si desea plantar un arbusto en casa. Recuerde que los atributos del Form tienen que seguir siendo privados y dentro de la clase base.

Ahora, añada el método `execute (Bureaucrat const & executor) const` al Form base e implemente un método que ejecute la acción del Form en cada uno de los Forms específicos. Tiene que comprobar que se haya firmado el Form y que el burócrata que intente ejecutarlo tenga el rango apropiado, de lo contrario lance la excepción que corresponda. Usted decide si prefiere realizar los controles en cada clase específica o en la clase

base y después llamar a otro método para ejecutar la acción. Lo que está claro es que hay una procedimiento más elegante que el otro. En cualquier caso, el Form base tiene que ser una clase abstracta.

Termine añadiéndo la función `executeForm (Form Const & form)` al burócrata. Tendrá que intentar ejecutar el Form y, si lo logra, mostrará algo parecido a `«bureaucrat>executs <form>"`. Si no, mostrará un mensaje de error explícito.

Añada lo que haga falta a su `main` para probar que todo funciona.

Capítulo V

Ejercicio 03: At least this beats coffee-making

	Ejercicio : 03
At least this beats coffee-making	
Directorio de entrega : $ex03/$	
Ficheros a entregar : `Lo mismo que antes + Intern.hpp Intern.cpp`	
Funciones prohibidas : `Ninguna`	

Dado que rellenar Forms es bastante aburrido, sería cruel pedirles a nuestros burócratas que lo rellenasen todo ellos mismos. Por eso, le vamos a pedir a un becario que lo haga.

Cree entonces la clase `Intern`. El becario no tiene ni nombre, ni rango, ni características determinantes. Lo único que nos importa de él es que haga su trabajo.

El becario dispone de algo importante, la función `makeForm`. Hacen falta dos cadenas, la primera representa el nombre del Form y la segunda el objetivo del Form. Devolverá, como puntero a `Form`, un puntero a la clase de Form específico representada por el primer parámetro e inicializada con el segundo parámetro. Imprimirá en la salida estándar algo parecido a "`Ïntern creates <form>`". Cualquier metodo de tipo if/elseif/elseif/else no se acceptará durante la evaluación. Si el Form solicitado no es conocido, muestre un mensaje de error explícito.

Por ejemplo, para crear un `RobotomyRequestForm` que tenga como objetivo a `"Bender"`:

```
{
        Intern  someRandomIntern;
        Form*   rrf;

        rrf = someRandomIntern.makeForm("robotomy request", "Bender");
}
```

Añada lo que haga falta a su `main` para probar que todo funciona.

Capítulo VI

Ejercicio 04: Así es como nos gustan, simples y aburridos

	Ejercicio : 04
Así es como nos gustan, simples y aburridos	
Directorio de entrega : $ex04/$	
Ficheros a entregar : `Lo mismo que antes + OfficeBlock.cpp OfficeBlock.hpp`	
Funciones prohibidas : `Ninguna`	

`Este ejercicio y el que sigue no puntúan, pero resultan interesantes para su piscina. No está obligado a hacerlos.`

La Burocracia Central, refugio del orden y de la organización, está compuesta de edificios de oficinas bien habilitados. Cada uno de ellos necesita a un becario y a dos burócratas para funcionar y ser capaz de crear, firmar y ejecutar formularios, todo ello con una simple orden. Mola, ¿verdad?

Cree entonces la clase `OfficeBlock`. Será construida pasando punteros a (o referencias a, decida en función de lo que resulte más apropiado) un becario, a un burócrata signatario y a un burócrata ejecutor. También se puede construir vacía. Ninguna otra construcción debe ser posible (ni copia, ni asignación).

Sus funciones serán designar a un nuevo becario, a un burócrata signatario o a un burócrata ejecutor.

Su única función "útil"será texttt doBureaucracy que recibe el nombre de un formulario y el de un objetivo. Intentará que el becario cree el formulario pedido, que el primer burócrata lo firme y que el segundo burócrata lo ejecute. Los mensajes imprimidos por el becario y por los burócratas proporcionarán un log de todo lo ocurrido. Si se produce

un error, se debe generar una excepción a partir de esa función: puede modificar lo que ha realizado hasta ahora para hacer que la función sea elegante. Recuerde: siempre se agradece que los mensjes de error sean específicos.

Por supuesto, si no están cubiertos los tres puestos de trabajo del edificio, no se podrá realizar ninguna actividad burocrática.

Añada lo que haga falta a su `main` para probar que todo funciona.

Por ejemplo, el siguiente bloque de código podría producir el output que se muestra después de él:

```
int main()
{
        Intern      idiotOne;
        Bureaucrat  hermes = Bureaucrat("Hermes Conrad", 37);
        Bureaucrat  bob = Bureaucrat("Bobby Bobson", 123);
        OfficeBlock ob;

        ob.setIntern(idiotOne);
        ob.setSigner(bob);
        ob.setExecutor(hermes);

        try
        {
                ob.doBureaucracy("mutant pig termination", "Pigley");
        }
        catch (Some::SpecificException & e)
        {
                /* specific known error happens, say something */
        }
        catch (std::exception & e)
        {
                /* oh god, unknown error, what to do ?! */
        }
}
```

```
$> ./ex04
Intern creates a Mutant Pig Termination Form (s.grade 130, ex.grade 50) targeted on Pigley (Unsigned)
Bureaucrat Bobby Bobson (Grade 123) signs a Mutant Pig Termination Form (s.grade 130, ex.grade 50)
    targeted on Pigley (Unsigned)
Bureaucrat Hermes Conrad (Grade 37) executes a Mutant Pig Termination Form (s.grade 130, ex.grade 50)
    targeted on Pigley (Signed)
That'll do, Pigley. That'll do ...
$>
```

Capítulo VII

Ejercicio 05: Generador infinito de firmas

	Ejercicio : 05
Generador infinito de firmas	
Directorio de entrega : $ex05/$	
Ficheros a entregar : `Lo mismo que antes + CentralBureaucracy.cpp CentralBureaucracy.hpp`	
Funciones prohibidas : `Ninguna`	

Ya solo nos queda embalar todo esto en un bonito paquete.

Cree la clase `CentralBureaucracy`. Se creará sin parámetros y su creación tendrá 20 edificios de oficinas vacíos.

Se podrá .ªlimentar.ª los burócratas en el objeto. Se generán los becarios de manera automática, sin la intervención del usuario porque, seamos francos, no cuestan un duro.

Los burócratas .ªlimentados.en el objeto serán utilizados para ocupar puestos en los edificios de oficinas. Si no hay ningún puesto vacante, puede rechazarlos o almacenarlos en alguna sala de espera.

Después de esto, podrá poner los objetivos en una fila de espera en el objeto, utilizando la función texttt queueUp que recibirá una cadena y el nombre de la persona que se encuentra en la fila de espera.

Por último, cuando se llame a la función texttt doBureaucracy, haga un poco de actividad burocrática al azar. El primero en llegar será el primer servido, siguiendo el orden de los edificios creados.

He aquí la pinta que podría tener su `main`:

- Cree la Burocracia Central

- Cree 20 burócratas al azar y envíelos a la Burocracia Central
- Ponga en fila de espera a un gran número de objetivos en la Burocracia Central
- Llame a la función `doBureaucracy()` y observe cómo actúa la magia