Form up, maggots!

Explicación detallada del ejercicio

Este ejercicio consiste en ampliar el código de la clase Bureaucrat (burocrático) que ya has implementado en el ejercicio anterior, añadiendo una nueva clase llamada Form. Esta clase representa un formulario que los burócratas podrán firmar, pero solo si cumplen ciertos requisitos de nivel.

El objetivo es modelar un sistema burocrático en el que los formularios tienen diferentes niveles de acceso y los burócratas deben tener suficiente autoridad para poder firmarlos. Se implementarán reglas de validación para asegurarse de que el sistema funcione correctamente.

¿Qué elementos tiene la clase Form?

La clase Form debe contener los siguientes atributos:

1. **Nombre (name)** → Es una constante, lo que significa que su valor no cambiará una vez que el formulario sea creado.
2. **Estado (isSigned)** → Un booleano que indica si el formulario ha sido firmado o no. Cuando se crea, su estado inicial es *no firmado*.
3. **Nivel requerido para firmarlo (gradeToSign)** → Un valor constante que define el nivel mínimo que necesita un burócrata para firmarlo.
4. **Nivel requerido para ejecutarlo (gradeToExecute)** → Un valor constante que indica el nivel necesario para ejecutar el formulario.

Nota: Todos estos atributos son privados (*private*), lo que significa que solo pueden ser accedidos desde dentro de la clase Form y no desde clases derivadas.

Reglas sobre los niveles (grades)

Al igual que en la clase Bureaucrat, los niveles siguen una jerarquía en la que **1 es el nivel más alto** y **los números más grandes representan niveles más bajos**.

Se deben manejar dos excepciones cuando un nivel esté fuera del rango permitido:

- Form::GradeTooHighException → Se lanza cuando el nivel es demasiado alto.
- Form::GradeTooLowException → Se lanza cuando el nivel es demasiado bajo.

Métodos a implementar

1. **Getters**
    a. Implementa funciones *getter* para acceder a cada uno de los atributos de la clase.
2. **Operador << sobrecargado**
    a. Implementa la sobrecarga del operador << para poder imprimir fácilmente la información de un formulario.
3. **Método beSigned(Bureaucrat)**
    a. Este método permite que un burócrata intente firmar el formulario.
    b. Si el nivel del burócrata es igual o superior al nivel requerido (gradeToSign), el estado del formulario cambiará a *firmado*.
    c. Si el burócrata tiene un nivel insuficiente, se lanzará la excepción Form::GradeTooLowException.
4. **Modificar el método signForm() de Bureaucrat**
    a. Este método, que ya existe en la clase Bureaucrat, debe ser modificado para intentar firmar el formulario utilizando Form::beSigned().
    b. Dependiendo del resultado, se imprimirá un mensaje indicando si la firma fue exitosa o no:
        i. Si la firma tiene éxito: <nombre del burócrata> signed <nombre del formulario>
        ii. Si la firma falla (porque el burócrata tiene un nivel insuficiente): <nombre del burócrata> couldn't sign <nombre del formulario> because <razón>.
5. **Tests**
    a. Es necesario crear pruebas para comprobar que todo el sistema funciona correctamente.
    b. Se deben probar casos en los que la firma sea exitosa y casos en los que se lance la excepción de nivel insuficiente.

¿Cómo afrontar este ejercicio?

Para completar este ejercicio de manera eficiente, sigue estos pasos:

1. **Crea la clase Form** con los atributos mencionados y sus respectivos *getters*.
2. **Implementa el método beSigned(Bureaucrat)**, asegurándote de manejar la excepción si el nivel del burócrata es insuficiente.
3. **Sobrecarga el operador <<** para imprimir correctamente la información de un formulario.
4. **Modifica el método signForm() en la clase Bureaucrat** para que intente firmar el formulario y genere los mensajes adecuados.
5. **Escribe pruebas** para verificar que todo funcione según lo esperado.

Si sigues estos pasos de forma ordenada y pruebas cada parte antes de avanzar, lograrás completar el ejercicio sin grandes dificultades.

Recursos

Para abordar con éxito este ejercicio, es fundamental profundizar en varios conceptos clave de C++98. A continuación, se presentan algunos recursos en español que abordan estos temas:

1. **Sobrecarga de operadores en C++98:**
   Curso C++ No Tan Básico. Sobrecarga de los operadores Input y Output.
   Este video explica cómo sobrecargar los operadores de inserción (<<) y extracción (>>), lo cual es esencial para imprimir la información de objetos de clases personalizadas.
   Ver video
2. **Sobrecarga de operadores con funciones amigas:**
   Curso C++ No Tan Básico. Sobrecarga de operadores con funciones amigas.
   Este video muestra cómo utilizar funciones amigas para sobrecargar operadores, permitiendo el acceso a miembros privados de una clase.
   Ver video
3. **Sobrecarga de operadores unarios:**
   Curso C++ No Tan Básico. Sobrecarga de operadores unarios.
   Este video se enfoca en la sobrecarga de operadores unarios como +, - y !, y cómo implementarlos en clases personalizadas.
   Ver video
4. **La Forma Canónica Ortodoxa en C++:**
   La Forma Canónica Ortodoxa.
   Este artículo describe el concepto de la Forma Canónica Ortodoxa, una práctica estándar en C++ para definir clases que se comporten de manera consistente y segura.
   Leer artículo
5. **Gestión de memoria en C++98:**
   Curso C++. Destructores. Vídeo 59.
   Este video aborda el uso de destructores en C++ para la gestión adecuada de la memoria y la liberación de recursos.
   Ver video

Estos recursos proporcionan una comprensión sólida de los conceptos necesarios para implementar la clase Form y sus interacciones con la clase Bureaucrat en C++98.

El código

Este código representa un sistema en C++ que modela el comportamiento de **burócratas** y **formularios** en una estructura jerárquica donde los burócratas pueden firmar formularios

dependiendo de su **grado**. Se implementan reglas estrictas sobre qué formularios pueden ser firmados y ejecutados por burócratas de distintos niveles.

◇ Bureaucrat (Burócrata)

Clase que representa a un burócrata con las siguientes características:

- **Nombre** (constante, no se puede cambiar después de su creación).
- **Grado** (un número entre 1 y 150 que representa su autoridad, donde **1 es el nivel más alto y 150 el más bajo**).
- Puede **incrementar o disminuir su grado**.
- Puede **firmar formularios** si su grado es lo suficientemente alto.
- Maneja excepciones si el grado es **demasiado alto** o **demasiado bajo**.

📌 Funciones clave en Bureaucrat

1. **Constructores y operador de asignación**
    a. Un **constructor por defecto** crea un burócrata con nombre "Default" y grado 150.
    b. Un **constructor parametrizado** permite asignar nombre y grado, pero lanza excepciones si el grado es inválido.
    c. Un **constructor de copia** y un **operador de asignación** permiten copiar burócratas.
2. **Getters**
    a. getName(): Devuelve el nombre del burócrata.
    b. getGrade(): Devuelve su grado.
3. **Modificación del grado**
    a. incrementGrade(): **Sube** el grado (se acerca a 1). Si el grado se vuelve **demasiado alto**, lanza una excepción.
    b. decrementGrade(): **Baja** el grado (se acerca a 150). Si el grado se vuelve **demasiado bajo**, lanza una excepción.
    c. setGrade(int _grade): Asigna un nuevo grado validando los límites.
4. **Firmar formularios**
    a. signForm(std::string formName, bool wasSigned): Muestra si el burócrata ha firmado un formulario o no.
5. **Excepciones**
    a. GradeTooHighException: Se lanza si el grado es demasiado alto (<1).
    b. GradeTooLowException: Se lanza si el grado es demasiado bajo (>150).
6. **Sobrecarga del operador <<**
    a. Permite imprimir información del burócrata en consola de forma elegante.

◇ Form (Formulario)

Clase que representa un formulario con:

- **Nombre** (constante).
- **Grado mínimo requerido para firmarlo** (constante).
- **Grado mínimo requerido para ejecutarlo** (constante).
- **Estado de firma** (bool, indica si el formulario ha sido firmado).

📌 Funciones clave en Form

1. **Constructores y operador de asignación**
    a. El **constructor parametrizado** permite crear un formulario validando que los grados requeridos estén en el rango permitido.
    b. Se implementan **constructor de copia** y **operador de asignación**.
2. **Getters**
    a. getName(): Obtiene el nombre del formulario.
    b. getGradeToSign(): Obtiene el grado requerido para firmarlo.
    c. getGradeToExecute(): Obtiene el grado requerido para ejecutarlo.
    d. getSigned(): Indica si el formulario ha sido firmado.
3. **Firmar un formulario**
    a. beSigned(Bureaucrat &bureaucrat): Permite a un burócrata firmar el formulario si su grado es suficiente. En caso contrario, lanza una excepción.
4. **Excepciones**
    a. GradeTooHighException: Se lanza si los grados requeridos son menores que el permitido.
    b. GradeTooLowException: Se lanza si los grados requeridos son mayores que el permitido.
5. **Sobrecarga del operador <<**
    a. Permite imprimir información del formulario en consola.

◇ Main (Pruebas)

El archivo Main.cpp prueba el funcionamiento de los burócratas y formularios mediante varios escenarios.

📌 Escenarios de prueba

1. **Crear formularios con grados inválidos**

   a. Se intentan crear formularios con grados **fuera del rango permitido**, lo que genera excepciones.
2. **Intentar firmar un formulario con un burócrata de menor grado**
   a. Un burócrata con grado **100** intenta firmar un formulario que requiere grado **50**.
   b. Como su grado es **demasiado bajo**, no puede firmarlo y se lanza una excepción.
3. **Intentar ejecutar un formulario con un burócrata de menor grado**
   a. Se intenta ejecutar un formulario con un burócrata cuyo grado es **demasiado bajo**.
   b. Se detecta que no tiene autoridad para ejecutarlo y se lanza una excepción.

◇ Resumen Final

Este código simula un sistema de burocracia con reglas estrictas:

- Los burócratas tienen **niveles de autoridad (grados)** y pueden firmar documentos si tienen el rango suficiente.
- Los formularios requieren un **grado mínimo** para ser firmados y ejecutados.
- Se **lanzan excepciones** si se intentan acciones que no cumplen con las reglas establecidas.

Este sistema ayuda a entender mejor la programación orientada a objetos en C++, el uso de **constructores**, **herencia (excepciones)** y **sobrecarga de operadores**. 🚀