At least this beats coffee-making

En este ejercicio, se te pide implementar una clase llamada Intern (Becario) en C++98. Esta clase no tiene atributos como nombre o grado; su única responsabilidad es crear formularios a través de una función miembro denominada makeForm(). Esta función recibe dos cadenas de caracteres: el nombre del formulario y el objetivo del formulario. Devuelve un puntero a un objeto de tipo Form cuyo nombre coincide con el proporcionado y cuyo objetivo se inicializa con el segundo parámetro. Además, debe imprimir un mensaje indicando la creación del formulario, por ejemplo: "Intern creates ". Si el nombre del formulario no existe, debe mostrar un mensaje de error claro.

Es importante evitar soluciones poco elegantes y difíciles de leer, como una serie de sentencias if/else if/else extensas. Este tipo de enfoques no serán aceptados durante el proceso de evaluación. Como de costumbre, debes asegurarte de que todo funcione según lo esperado mediante pruebas adecuadas.

Cómo abordar el ejercicio en C++98:

1. **Definición de la clase Intern:** Crea la clase Intern sin atributos privados, ya que no se requieren.
2. **Implementación de la función makeForm():** Esta función debe recibir dos parámetros de tipo std::string: el nombre del formulario y el objetivo. Para evitar una estructura de control extensa con múltiples if/else if/else, puedes utilizar un arreglo de estructuras o pares que asocien nombres de formularios con funciones específicas de creación. Luego, iteras sobre este arreglo para encontrar una coincidencia con el nombre del formulario proporcionado y, al encontrarla, llamas a la función correspondiente para crear el formulario. Si no se encuentra ninguna coincidencia, se debe imprimir un mensaje de error claro.
3. **Creación de formularios específicos:** Implementa clases derivadas de Form para cada tipo específico de formulario, como RobotomyRequestForm, ShrubberyCreationForm y PresidentialPardonForm. Cada una de estas clases debe tener un constructor que acepte el objetivo del formulario y lo inicialice adecuadamente.
4. **Integración con ejercicios anteriores:** Asegúrate de que todos los archivos y funcionalidades de los ejercicios previos estén correctamente integrados en este ejercicio, ya que se requiere entregar los archivos anteriores junto con los nuevos.

Recursos recomendados:

Para ayudarte a comprender y abordar con éxito este ejercicio, aquí tienes una selección de videos en español que cubren los temas principales involucrados:

1. **Programación Orientada a Objetos en C++:**

   a. *Descripción:* Este video ofrece una introducción detallada a la programación orientada a objetos en C++, incluyendo conceptos clave como clases y herencia.
   b. *Enlace:* Programación Orientada a Objetos en C++
2. **Funciones y Punteros a Funciones en C++:**
   a. *Descripción:* Aprende cómo utilizar punteros a funciones en C++ para evitar estructuras de control complejas y mejorar la legibilidad del código.
   b. *Enlace:* Funciones y Punteros a Funciones en C++
3. **Manejo de Strings en C++:**
   a. *Descripción:* Este tutorial explica cómo trabajar con cadenas de caracteres en C++, incluyendo la comparación y manipulación de std::string.
   b. *Enlace:* Manejo de Strings en C++
4. **Creación y Uso de Clases en C++:**
   a. *Descripción:* Una guía paso a paso sobre cómo definir e implementar clases en C++, con ejemplos prácticos.
   b. *Enlace:* Creación y Uso de Clases en C++
5. **Herencia y Polimorfismo en C++:**
   a. *Descripción:* Este video profundiza en los conceptos de herencia y polimorfismo, fundamentales para la creación de clases derivadas y el uso de funciones virtuales.
   b. *Enlace:* Herencia y Polimorfismo en C++

Estos recursos te proporcionarán una comprensión sólida de los conceptos necesarios para implementar la clase Intern y sus funcionalidades asociadas en C++98.

El código

El siguiente código es una implementación en C++ de un sistema burocrático donde diferentes tipos de formularios deben ser firmados y ejecutados por burócratas con distintos niveles de autoridad.

1. Concepto General del Código

El código define una jerarquía de clases que representan un sistema de burocracia, donde:

- **Burócratas** tienen nombres y niveles de autoridad (grados).
- **Formularios** requieren un grado mínimo para ser firmados y ejecutados.
- **Un interno (Intern)** es capaz de crear ciertos tipos de formularios.
- **Diferentes tipos de formularios (PresidentialPardonForm, RobotomyRequestForm, ShrubberyCreationForm)** tienen su propio comportamiento específico al ejecutarse.

2. Explicación de las Clases

Clase Bureaucrat (Burócrata)

Esta clase representa a un burócrata con un nombre y un grado (nivel de autoridad).

Cuanto menor es el número del grado, mayor es su autoridad (1 es el más alto, 150 es el más bajo).

Atributos:

- std::string _name: Nombre del burócrata (constante, no cambia).
- int _grade: Nivel del burócrata (entre 1 y 150).

Métodos principales:

- **Constructor:** Inicializa un burócrata con un nombre y un grado, validando que esté dentro del rango permitido.
- **incrementGrade() / decrementGrade()**: Aumenta/disminuye el nivel del burócrata.
- **signForm()**: Intenta firmar un formulario.
- **executeForm()**: Ejecuta un formulario (si tiene el nivel suficiente).

Excepciones:

- GradeTooHighException: Se lanza si el grado es menor que MAX_GRADE (1).
- GradeTooLowException: Se lanza si el grado es mayor que MIN_GRADE (150).

Clase AForm (Formulario Abstracto)

Es una **clase abstracta** que representa un formulario en el sistema. Tiene atributos y métodos comunes a todos los formularios.

Atributos:

- std::string _name: Nombre del formulario.
- std::string _target: El "objetivo" del formulario (a quién afecta).
- int _gradeSign: Nivel necesario para firmarlo.
- int _gradeExec: Nivel necesario para ejecutarlo.
- bool _signed: Indica si el formulario ha sido firmado o no.

Métodos principales:

- **Constructor:** Valida los niveles (_gradeSign y _gradeExec).
- **beSigned(Bureaucrat &bureaucrat)**: Permite que un burócrata firme el formulario si tiene el nivel suficiente.
- **execute(Bureaucrat const &executor)**: Llama a executeSuperClassForm(), una función **pura virtual** (debe ser implementada por clases hijas).

Excepciones:

- gradeTooHighException: Se lanza si los valores de grado son demasiado altos.
- gradeTooLowException: Se lanza si los valores de grado son demasiado bajos.

Clase Intern (Interno)

Representa a un interno que puede crear ciertos tipos de formularios.

Método principal:

- **makeForm(std::string formName, std::string target)**:
  - Recibe el nombre del formulario y el objetivo.
  - Si el nombre coincide con uno de los tres tipos predefinidos (presidential pardon, robotomy request, shrubbery creation), lo crea.
  - Si el nombre no es válido, imprime un mensaje de error.

Clases Concretas de Formularios

Cada uno hereda de AForm y define su propia implementación de executeSuperClassForm().

1. PresidentialPardonForm

- Se usa para otorgar un indulto presidencial.
- **Requiere grado 25 para firmar y 5 para ejecutar.**
- En su ejecución, imprime que el objetivo ha sido indultado por "Zafod Beeblebrox".

2. RobotomyRequestForm

- Se usa para realizar una "robotomización" (operación quirúrgica ficticia).
- **Requiere grado 72 para firmar y 45 para ejecutar.**
- Su ejecución tiene **50% de éxito**:

- Si tiene éxito, imprime que el objetivo ha sido robotomizado.
- Si falla, imprime que la robotomización falló.

3. ShrubberyCreationForm

- Se usa para crear arbustos.
- **Requiere grado 145 para firmar y 137 para ejecutar.**
- En su ejecución, genera un archivo de texto con un dibujo de un árbol en ASCII.

3. Ejemplo de Uso

```
#include "Bureaucrat.hpp"
#include "Intern.hpp"

int main() {
  try {
    Intern someIntern;
    Bureaucrat bob("Bob", 3);  // Burócrata con alto nivel de autoridad

    AForm *form = someIntern.makeForm("robotomy request", "Target1");
    if (form) {
      bob.signForm(form->getName(), true);  // Bob firma el formulario
      bob.executeForm(*form);  // Bob ejecuta el formulario
      delete form;  // Liberar memoria
    }
  } catch (std::exception &e) {
    std::cerr << e.what() << std::endl;
  }
  return 0;
}
```

Salida esperada:

```
Intern creates robotomy request form
Bureaucrat Bob => signed form RobotomyRequestForm
* * * * * * DRRRR * * * * * * *
Target1 has been robotomized successfully!
```

Nota: La ejecución de la robotomización tiene 50% de probabilidad de fallar.

4. Puntos Clave

✓ **Uso de programación orientada a objetos (POO)**

✓ **Uso de herencia y polimorfismo** (clases hijas sobrescriben executeSuperClassForm())

✓ **Excepciones personalizadas** para errores de nivel de burócrata y formulario

✓ **Uso de memoria dinámica (new y delete)** en Intern::makeForm()