

Mommy, when I grow up, I want to be a bureaucrat!

El ejercicio propuesto consiste en diseñar una clase Bureaucrat que forma parte de un sistema burocrático simulado. Esta clase debe cumplir con las siguientes características:

- **Atributos:**
 - name: n nombre constante que identifica al burócrata. - grade: n grado que varía entre 1 (el más alto) y 150 (el más bajo).
- **Restricciones:**
 - Si se intenta crear un objeto Bureaucrat con un grade fuera del rango permitido, se debe lanzar una excepción específica: Bureaucrat::GradeTooHighException para valores demasiado altos (por encima de 1) o Bureaucrat::GradeTooLowException para valores demasiado bajos (por debajo de 150).
- **Métodos:**
 - getName(): Devuelve el nombre del burócrata. - getGrade(): devuelve el grado del burócrata. - incrementGrade(): incrementa el grado del burócrata en una unidad. Si el resultado supera el límite superior (1), se lanza la excepción GradeTooHighException. - decrementGrade(): decrementa el grado del burócrata en una unidad. Si el resultado supera el límite inferior (150), se lanza la excepción GradeTooLowException.
- **Sobrecarga de operadores:**
 - Sobrecargar el operador de inserción (<<) para que, al imprimir un objeto Bureaucrat, se muestre en el formato: <name>, bureaucrat grade <grade>.
- **Forma Canónica Ortodoxa:**
 - Exceptuando las clases de excepciones, todas las demás clases deben seguir la Forma Canónica Ortodoxa en C++, lo que implica implementar:
 - Constructor por defecto.
 - Constructor de copia.
 - Operador de asignación.
 - Destructor.

Enfoque para Implementar en C++98:

1. Definición de la Clase Bureaucrat:

Atributos:

- i. const std::string name;: l ser constante, debe inicializarse en el momento de la construcción del objeto. - int grade;: debe asegurarse que siempre esté dentro del rango [1, 150].

2. Constructores:

Constructor Parametrizado:

- i. Inicializa name y grade. - verifica que grade esté dentro del rango permitido; de lo contrario, lanza la excepción correspondiente. - **Constructor por Defecto:**
- ii. Aunque name es constante, se puede proporcionar un valor por defecto utilizando un constructor que inicialice name con un valor predeterminado y grade con un valor válido, como 150.

3. Gestión de Excepciones:

Definir clases de excepción dentro de Bureaucrat para GradeTooHighException y GradeTooLowException. - estas clases pueden heredar de std::exception y sobrescribir el método what() para proporcionar mensajes de error descriptivos.

4. Métodos Getters:

getName(): devuelve una referencia constante a name. - getGrade(): Devuelve el valor de grade.

5. Métodos de Incremento y Decremento:

incrementGrade():

- i. Disminuye el valor de grade en 1 (recordando que 1 es el grado más alto). - i el nuevo valor de grade es menor que 1, lanza GradeTooHighException. -

decrementGrade():

- ii. Aumenta el valor de grade en 1. - i el nuevo valor de grade es mayor que 150, lanza GradeTooLowException.

6. Sobrecarga del Operador <<:

- a. Definir una función amiga que permita acceder a los atributos privados de Bureaucrat. - Esta función debe insertar en el flujo de salida el nombre y el grado en el formato especificado.

7. Forma Canónica Ortodoxa:**a. Constructor de Copia:**

Crea un nuevo objeto Bureaucrat copiando los atributos de otro existente.

b. Operador de Asignación:

Copia los atributos de un Bureaucrat a otro, asegurándose de manejar correctamente la autoasignación y de que name permanezca constante.

c. Destructor:

Libera recursos si es necesario. En este caso, como no se utilizan recursos dinámicos, el destructor puede ser trivial.

Recursos Recomendados:

Para profundizar en los conceptos clave necesarios para abordar este ejercicio en C++98, se recomiendan los siguientes videos:

1. Excepciones en C++:

Programación en C++: Gestión de Excepciones* - Este video explica cómo manejar excepciones en C++, https://youtu.be/xKjgAnXZ_AY?si=P_3_gfSOFFdhGQS-

2. Sobrecarga de Operadores:

C++ Sobrecarga de Operadores* - Una explicación detallada sobre cómo y por qué sobrecargar operadores en C++. - Ver Video](<https://www.youtube.com>

3. Constructores y Destructores en C++:

Este video ofrece una explicación detallada sobre la creación y destrucción de objetos en C++, abordando la importancia de los constructores y destructores en la gestión de recursos. - Ver Video](<https://www.youtube.com/watch?v=KfVCP3H4wn4>)

4. Curso C++: Destructores:

En este video se profundiza en el uso de destructores en C++, incluyendo ejemplos prácticos y consejos sobre su correcta implementación. - Ver Video](<https://www.youtube.com/watch?v=oG7f7REBwcU>)

5. C++: La Regla de los Tres, Constructores y Copias:

Este video analiza la "Regla de los Tres" en C++, que es fundamental para entender la gestión de recursos y la correcta implementación de constructores de copia, destructores y operadores de asignación. - Ver Video](<https://www.youtube.com/watch?v=cFyXvERLYLQ>)

6. Programación en C++: Destructor de Objetos:

Este video explica el concepto de destructores en C++, su sintaxis y su papel en la liberación de recursos, con ejemplos claros y concisos. - Ver Video](<https://www.youtube.com/watch?v=N-3BVAvWJjk>)

7. Curso C++ No Tan Básico: Destructores:

Este video ofrece una visión más avanzada sobre los destructores en C++, incluyendo prácticas recomendadas y errores comunes a evitar. - Ver Video](https://www.youtube.com/watch?v=GnNA7_6krIA)

Estos recursos proporcionan explicaciones detalladas y ejemplos prácticos que le ayudarán a comprender y aplicar correctamente conceptos fundamentales en C++98, como la gestión de memoria, constructores, destructores y la regla de los tres, esenciales para la correcta implementación de la clase Bureaucrat y sus excepciones asociadas.

El código:

Este código implementa una clase Bureaucrat que modela a un burócrata con un **nombre constante** y un **grado** que varía entre 1 (más alto) y 150 (más bajo).

- Se usan **excepciones personalizadas** para manejar errores cuando el grado está fuera del rango permitido.
- Se sigue la **Forma Canónica Ortodoxa** en la implementación de la clase Bureaucrat.
- Se sobrecarga el operador << para imprimir información sobre un Bureaucrat.

- Se usa un archivo main.cpp para probar distintas funcionalidades y casos de error.

1. Archivo Bureaucrat.hpp (Definición de la clase)

Este archivo contiene la declaración de la clase Bureaucrat y sus métodos.

Atributos

private:

```
std::string const _name; // Nombre del burócrata (constante, no cambia)
int _grade;             // Grado del burócrata (1 = mejor, 150 = peor)
```

- `_name` es const, lo que significa que **no puede cambiar** una vez inicializado.
- `_grade` es un número entero que representa el grado del burócrata.

Constructores y operadores

public:

```
Bureaucrat();
Bureaucrat(std::string _name, int _grade);
Bureaucrat(const Bureaucrat &origin);
Bureaucrat &operator=(const Bureaucrat &src);
~Bureaucrat();
```

- **Constructor por defecto:** Inicializa un burócrata llamado "Default" con grado 150.
- **Constructor con parámetros:** Permite crear un burócrata con un nombre específico y un grado dentro del rango permitido. **Lanza excepciones si el grado es inválido.**
- **Constructor de copia:** Crea una copia de otro burócrata.
- **Operador =:** Permite asignar un burócrata a otro.
- **Destructor:** Libera recursos si fuera necesario (en este caso, no hace mucho).

Métodos principales

```
const std::string getName() const;
int getGrade() const;
void incrementGrade();
void decrementGrade();
```

```
void setGrade(int grade);
```

- getName() devuelve el nombre del burócrata.
- getGrade() devuelve el grado.
- incrementGrade() sube el rango (disminuye el número).
- decrementGrade() baja el rango (aumenta el número).
- setGrade(int grade) cambia el grado, validando que esté dentro del rango permitido.

Excepciones personalizadas

```
class GradeTooHighException : public std::exception {  
    public: virtual const char *what() const throw();  
};
```

```
class GradeTooLowException : public std::exception {  
    public: virtual const char *what() const throw();  
};
```

- GradeTooHighException: Se lanza cuando el grado es menor que 1.
- GradeTooLowException: Se lanza cuando el grado es mayor que 150.
- El método what() devuelve un mensaje de error cuando se captura la excepción.

Sobrecarga del operador <<

```
std::ostream &operator<<(std::ostream &stream, Bureaucrat &Bureaucrat);
```

- Permite imprimir un burócrata en formato: Alice, bureaucrat grade 91.



2. Archivo Bureaucrat.cpp (Implementación de la clase)

Constructor por defecto

```
Bureaucrat::Bureaucrat() : _name("Default"), _grade(150) {  
    cout << "Default constructor called 🤖" << endl;  
}
```


- Crea un burócrata con nombre "Default" y grado 150.

Constructor con parámetros (Valida el rango del grado)

```
Bureaucrat::Bureaucrat(std::string _name, int _grade) : _name(_name) {  
    cout << "Parametric constructor called  " << endl;  
    if (_grade < MAX_GRADE)  
        throw Bureaucrat::GradeTooHighException();  
    else if (_grade > MIN_GRADE)  
        throw Bureaucrat::GradeTooLowException();  
    else  
        this->_grade = _grade;  
    cout << "Bureaucrat " << this->getName() << "  created with grade " << this->_grade << endl;  
}
```


- Se almacena _name y se valida _grade.
- **Si _grade es menor que 1, lanza GradeTooHighException.**
- **Si _grade es mayor que 150, lanza GradeTooLowException.**
- Si el grado es válido, se asigna a _grade.

Constructor de copia

```
Bureaucrat::Bureaucrat(const Bureaucrat& origin) {  
    cout << "Calling to copy constructor  " << endl;  
    if (this != &origin)  
        *this = origin;  
}
```

- Llama al operador = para copiar los atributos de origin.

Operador de asignación =

```
Bureaucrat& Bureaucrat::operator=(const Bureaucrat& src) {  
    const_cast<std::string&>(this->_name) = src._name;  
    this->_grade = src._grade;  
    cout << "Bureaucrat: " << this->getName() << " -> Copy assignation operator called  " << endl;  
    return *this;  
}
```

- Se usa `const_cast` para modificar `_name`, que es `const` (esto no es recomendable en C++ moderno).
- Se copia `_grade`.

Métodos `incrementGrade()` y `decrementGrade()`

```
void Bureaucrat::incrementGrade() {  
    if (_grade - 1 < MAX_GRADE)  
        throw Bureaucrat::GradeTooHighException();  
    else  
        this->_grade--;  
}
```

```
void Bureaucrat::decrementGrade() {  
    if (_grade + 1 > MIN_GRADE)  
        throw Bureaucrat::GradeTooLowException();  
    else  
        this->_grade++;  
}
```

- **`incrementGrade()`** sube el rango. Si `_grade` ya es 1, lanza `GradeTooHighException`.
- **`decrementGrade()`** baja el rango. Si `_grade` ya es 150, lanza `GradeTooLowException`.

3. Archivo `main.cpp` (Pruebas y demostración)

Sección 1: Creación y Copia

```
Bureaucrat a("Alice", 91);  
Bureaucrat b(a);  
Bureaucrat c = b;
```

- Crea Alice con grado 91.
- Copia a en b (constructor de copia).
- Asigna b a c (operador `=`).

Sección 2: Manejo de Excepciones

```
try{  
    Bureaucrat a("A", MIN_GRADE + 1);  
} catch (std::exception &e) {  
    cerr << e.what() << endl;  
}
```

- Intenta crear un burócrata con grado 151, lo que lanza GradeTooLowException.

Sección 3: Incremento y Decremento Inválidos

```
try{  
    Bureaucrat c("Chief", MAX_GRADE);  
    c.incrementGrade(); // Lanza GradeTooHighException  
} catch (std::exception &e) {  
    cerr << e.what() << endl;  
}
```

- Chief empieza con grado 1 y al intentar mejorarlo, lanza una excepción.

Conclusión

Este código:

- ✓ Implementa la **Forma Canónica Ortodoxa** en C++98.
- ✓ Usa **excepciones personalizadas** para manejar errores.
- ✓ Controla el **rango del grado** y evita valores inválidos.
- ✓ Usa **sobrecarga de operadores** (<<).