Serena, my love!

Explicación detallada del ejercicio

Este ejercicio tiene como objetivo la creación de una clase derivada llamada ScavTrap basada en la clase base ClapTrap, que probablemente hayas implementado en ejercicios anteriores. Aquí tienes un desglose claro y completo de lo que se pide:

1. Crear una nueva clase ScavTrap:

- **Herencia:** ScavTrap debe derivar de la clase base ClapTrap. Esto significa que ScavTrap hereda todas las propiedades y métodos de ClapTrap, pero también puede tener propiedades y métodos propios.
- **Constructores y destructor:**
  - Los constructores y destructor de ScavTrap deben invocar (automáticamente o explícitamente) los constructores y destructor de ClapTrap.
  - Además, deben imprimir mensajes únicos para indicar que son específicos de ScavTrap.
  - Es importante demostrar que el proceso de construcción/destrucción sigue el orden correcto:
    - Cuando creas un objeto ScavTrap, primero se construye un ClapTrap.
    - Cuando destruyes un objeto ScavTrap, primero se destruye el ScavTrap y luego el ClapTrap.

2. Modificar y usar atributos heredados de ClapTrap:

- Actualiza la clase ClapTrap para permitir que ScavTrap use sus atributos de manera adecuada.
- Al crear un objeto ScavTrap, sus atributos se inicializan con los siguientes valores:
  - **Nombre:** Se pasa como parámetro al constructor de ScavTrap.
  - **Hit Points:** 100 (representan la salud del robot).
  - **Energy Points:** 50.
  - **Attack Damage:** 20.

3. Añadir una función específica para ScavTrap:

- Define una función miembro llamada guardGate que no recibe parámetros ni devuelve un valor.
- Esta función imprime un mensaje indicando que ScavTrap está ahora en "Gate Keeper Mode" (Modo Guardián de la Puerta).

4. Implementar el método attack:

- ScavTrap debe sobrescribir (override “anular”) el método attack de ClapTrap, pero imprimir un mensaje diferente para reflejar la individualidad del robot.

5. Realizar pruebas:

- Asegúrate de incluir pruebas que verifiquen el correcto funcionamiento de los constructores, destructor, inicialización de atributos, el método attack y la función guardGate.
- Las pruebas también deben demostrar el encadenamiento correcto de construcción y destrucción (llamadas en orden correcto).

Cómo afrontar este ejercicio en C++98

1. **Definir la clase ScavTrap:**
    a. Usa public como especificador de acceso en la herencia para que los miembros públicos y protegidos de ClapTrap también sean accesibles en ScavTrap.
    b. Incluye los constructores y destructor:
        i. **Constructor por defecto:** Inicializa los atributos heredados con los valores indicados.
        ii. **Constructor con parámetros:** Permite inicializar el nombre y establece los demás atributos con los valores predeterminados.
        iii. **Destructor:** Asegúrate de que libere recursos si es necesario (aunque aquí no parece necesario).
2. **Sobrescribir el método attack:**
    a. Implementa una versión específica para ScavTrap que imprima un mensaje diferente.
3. **Añadir la función guardGate:**
    a. Implementa esta función para mostrar el mensaje de que el ScavTrap está en modo "Gate Keeper".
4. **Actualizar ClapTrap:**
    a. Si la clase base ClapTrap no permite un fácil acceso a sus atributos (como hitPoints, energyPoints, attackDamage), debes asegurarte de que estén protegidos (protected) en lugar de privados (private).
5. **Escribir el código de pruebas:**
    a. Crea objetos de ScavTrap y verifica que los constructores y destructor se llaman en el orden correcto.
    b. Llama a attack y guardGate para asegurarte de que imprimen los mensajes esperados.
    c. Comprueba que los atributos se inicializan correctamente.

Sigue las restricciones de C++98, evitando características modernas como nullptr o inicializadores en las definiciones de clase. Las pruebas necesarias se pueden realizar en el archivo main.cpp.

Explicación detallada del código

El código proporciona una implementación de dos clases en C++98: **ClapTrap** (una clase base) y **ScavTrap** (una clase derivada de ClapTrap). Estas clases modelan "robots" con habilidades de combate y características especiales.

1. Clase ClapTrap (Base)

Archivo ClapTrap.hpp (Encabezado)

La clase **ClapTrap** representa un robot con atributos y métodos básicos:

Atributos protegidos (protected)

Estos atributos son accesibles por ScavTrap (la clase derivada):

- _name: Nombre del ClapTrap.
- _hitPoints: Puntos de vida (inicialmente 100).
- _energyPoints: Puntos de energía (inicialmente 50).
- _attackDamage: Daño de ataque (inicialmente 20).

Constructores y destructor

- **Constructor por defecto:** Inicializa los valores predeterminados.
- **Constructor con parámetros:** Permite establecer el nombre al crearlo.
- **Constructor de copia:** Crea un nuevo objeto copiando los valores de otro ClapTrap.
- **Destructor:** Indica cuando un ClapTrap es destruido.

Operador de asignación (=)

Permite copiar valores de un ClapTrap a otro.

Métodos principales

- attack(target): Ataca a un objetivo si tiene suficiente energía y vida.
- takeDamage(amount): Reduce los puntos de vida según el daño recibido.
- beRepaired(amount): Aumenta los puntos de vida.

Getters y setters

Permiten acceder y modificar los atributos _name, _hitPoints, _energyPoints y _attackDamage.

Archivo ClapTrap.cpp (Implementación)

Implementa los métodos definidos en ClapTrap.hpp, incluyendo:

- **Los constructores y destructor**, que imprimen mensajes para mostrar el flujo de creación y destrucción de objetos.
- **El método attack**, que:
  - Verifica si el ClapTrap tiene suficiente energía y vida antes de atacar.
  - Resta 2 puntos de energía si ataca con éxito.
- **El método takeDamage**, que:
  - Reduce los puntos de vida o los deja en 0 si el daño es mayor.
- **El método beRepaired**, que:
  - Aumenta los puntos de vida del ClapTrap.

2. Clase ScavTrap (Derivada de ClapTrap)

Archivo ScavTrap.hpp (Encabezado)

La clase **ScavTrap hereda públicamente** de **ClapTrap**, lo que significa que los atributos y métodos public y protected de ClapTrap están disponibles en ScavTrap.

Nuevos elementos en ScavTrap

- **Constructores y destructor:**
  - Se invoca automáticamente el constructor de ClapTrap antes de inicializar ScavTrap.
- **Método attack sobrescrito (override)**:
  - Similar al de ClapTrap, pero con diferente mensaje y un costo mayor de energía (5 en lugar de 2).
- **Nuevo método guardGate**:
  - Muestra un mensaje indicando que el ScavTrap está en "Gate Keeper Mode".

Archivo ScavTrap.cpp (Implementación)

- **Los constructores llaman al constructor de ClapTrap** antes de mostrar sus propios mensajes.
- **El método attack**:
  - Verifica si hay suficiente energía (mínimo 5 puntos).
  - Reduce la energía y muestra el mensaje de ataque.

- **El método guardGate**:
  - Simplemente imprime que el ScavTrap está en "Gate Keeper Mode".
- **El destructor imprime un mensaje al destruirse**.

3. Archivo main.cpp (Pruebas del programa)

El main.cpp crea objetos de **ScavTrap** y prueba sus funcionalidades:

Flujo del programa

1. Se crea un **ScavTrap llamado "Damien"**.
2. Se crea otro ScavTrap **temporal** (tmp), se le asignan **666 puntos de energía** y luego se copia en "Damien".
3. Se imprimen los atributos de **"Damien"** tras la asignación.
4. **"Damien" ataca varios objetivos** (usando el método attack).
5. **"Damien" recibe y se repara de daño** (takeDamage y beRepaired).
6. **"Damien" entra en modo "Gate Keeper"** (guardGate).
7. Se indica que "Damien" quedará bajo la tutela del Presidente.

4. Orden de Construcción y Destrucción

Cuando se crea un ScavTrap:

1. **Se llama primero al constructor de ClapTrap**.
2. Luego, **se llama al constructor de ScavTrap**.

Cuando se destruye un ScavTrap:

1. **Se ejecuta primero el destructor de ScavTrap**.
2. **Luego se ejecuta el destructor de ClapTrap** (porque un ScavTrap es también un ClapTrap).

Resumen

- **ClapTrap es la clase base** que define los atributos y funciones esenciales de un robot de combate.
- **ScavTrap hereda de ClapTrap** y añade nuevas funcionalidades como guardGate.
- **El main.cpp demuestra el uso de estas clases** mediante la creación de objetos y ejecución de pruebas.

- **Se respeta el ciclo de vida de los objetos**, con constructores y destructores ejecutándose en el orden correcto.

Este código sigue las normas de **C++98**, evitando características modernas como nullptr y asegurando compatibilidad con compiladores antiguos.