Aaaaand... OPEN!

Este ejercicio tiene como objetivo implementar una clase en C++ llamada **ClapTrap**. El propósito es familiarizarse con los principios básicos de clases en C++, incluyendo atributos privados, métodos públicos, constructores y destructores, así como la gestión básica de estados y mensajes de salida.

Directorio y Archivos

- **Directorio para entrega:** ex00/
- **Archivos a entregar:**
  - Makefile: Archivo que automatiza la compilación.
  - main.cpp: Archivo principal para pruebas.
  - ClapTrap.hpp: Declaración de la clase.
  - ClapTrap.cpp: Implementación de la clase.

Detalles de la Clase ClapTrap

La clase tendrá los siguientes **atributos privados**, con valores iniciales especificados:

- **Name**: Nombre que se pasa como parámetro al constructor.
- **Hit points (10)**: Representa la salud del ClapTrap.
- **Energy points (10)**: Cantidad de energía disponible.
- **Attack damage (0)**: Daño que inflige al atacar.

También debe tener los siguientes **métodos públicos** para simular el comportamiento de un ClapTrap:

1. **void attack(const std::string& target);**
   a. El ClapTrap ataca a un objetivo, reduciendo sus puntos de vida en función de su "Attack damage".
   b. Imprime un mensaje como:

ClapTrap <nombre> attacks <target>, causing <damage> points of damage!

   c. Consume **1 punto de energía**.
2. **void takeDamage(unsigned int amount);**
   a. Simula que el ClapTrap recibe daño, reduciendo sus puntos de vida en función del valor amount.
   b. Los puntos de vida no deben ser negativos.
3. **void beRepaired(unsigned int amount);**
   a. El ClapTrap se repara a sí mismo, recuperando los puntos de vida especificados en amount.
   b. Consume **1 punto de energía**.
   c. Imprime un mensaje como:

ClapTrap <nombre> repairs itself, recovering <amount> hit points!

Reglas adicionales

- El ClapTrap **no puede atacar ni repararse** si no tiene puntos de vida (**Hit points**) o energía (**Energy points**) suficientes.
- Los **constructores y el destructor** deben imprimir mensajes que indiquen cuándo son llamados, para facilitar la evaluación.
- No se permite interacción directa entre diferentes instancias de ClapTrap (por ejemplo, un ClapTrap atacando a otro).

Cómo Enfrentar el Ejercicio en C++ 98

A continuación, te explico un enfoque claro para implementar la solución:

1. Declarar la Clase (ClapTrap.hpp)

- Define los atributos privados: nombre, puntos de vida, puntos de energía y daño de ataque.
- Declara los métodos públicos.
- Declara los constructores, incluido el destructor.

2. Implementar la Clase (ClapTrap.cpp)

- Inicializa los atributos en el constructor, usando parámetros o valores por defecto.
- Implementa cada método público asegurándote de verificar las condiciones necesarias, como los puntos de energía o vida antes de realizar acciones.
- Asegúrate de imprimir mensajes descriptivos en cada método y en los constructores/destructores.

3. Crear el Makefile

- Escribe un Makefile para compilar el programa con los comandos básicos:
  - Compilar los archivos fuente (ClapTrap.cpp y main.cpp).
  - Generar el ejecutable.
  - Añadir reglas de limpieza (clean).

4. Escribir Pruebas (main.cpp)

- Crea instancias de ClapTrap con diferentes nombres.
- Simula ataques, reparaciones y daños.
- Asegúrate de probar casos límite, como intentar realizar acciones sin puntos de vida o energía.

Resumen de Pasos Clave

1. **Define la clase ClapTrap** en un archivo de cabecera.
2. **Implementa los métodos** siguiendo las reglas descritas en el enunciado.
3. **Añade mensajes descriptivos** en cada método.
4. **Configura el Makefile** para compilar todo correctamente.
5. **Escribe un main.cpp** para probar la funcionalidad de manera exhaustiva.

Explicación detallada y sencilla del código

Este programa implementa una clase **ClapTrap** en C++ que simula un robot con habilidades básicas de atacar, recibir daño y repararse. A continuación, se explica cada sección del código:

ClapTrap.hpp

Propósito

Este archivo contiene la **declaración** de la clase ClapTrap. Define su interfaz (atributos y métodos públicos/privados) para su uso posterior.

1. **Atributos privados**:
    a. **_name**: Nombre del ClapTrap.
    b. **_hitPoints**: Puntos de vida. Si llega a 0, el ClapTrap no puede realizar ninguna acción.
    c. **_energyPoints**: Puntos de energía. Son necesarios para atacar o repararse.
    d. **_attackDamage**: Daño que causa al atacar.
2. **Constructores y destructor**:
    a. **ClapTrap()**: Constructor por defecto, inicializa valores básicos.
    b. **ClapTrap(const ClapTrap&)**: Constructor de copia, clona un ClapTrap existente.
    c. **ClapTrap(std::string name)**: Constructor parametrizado que establece el nombre del ClapTrap.
    d. **~ClapTrap()**: Destructor, ejecutado al destruir un objeto.
3. **Operador de asignación (=)**: Permite copiar los valores de un ClapTrap a otro mediante la sobrecarga del operador de asignación.
4. **Métodos públicos**:
    a. **attack**: Realiza un ataque a un objetivo.
    b. **takeDamage**: El ClapTrap recibe daño y se reducen sus puntos de vida.
    c. **beRepaired**: Repara al ClapTrap aumentando sus puntos de vida.
    d. **Setters y getters**: Métodos para establecer o consultar los atributos.

ClapTrap.cpp

Implementación

Aquí se desarrolla la lógica de los métodos declarados en el archivo de cabecera.

1. **Constructores y destructor**:
    a. **ClapTrap()**:
        i. Inicializa _attackDamage en 0, _energyPoints y _hitPoints en 10.
        ii. Imprime un mensaje indicando que se llamó al constructor por defecto.
    b. **ClapTrap(const ClapTrap&)**:
        i. Copia los atributos de otro ClapTrap.
        ii. Imprime un mensaje indicando que se llamó al constructor de copia.
    c. **ClapTrap(std::string name)**:
        i. Establece el nombre del ClapTrap y valores iniciales de los atributos.
        ii. Imprime un mensaje indicando el nombre asignado.
    d. **~ClapTrap()**:
        i. Imprime un mensaje indicando que se destruyó un ClapTrap.
2. **Operador de asignación (=)**:
    a. Copia los atributos de otro ClapTrap.
    b. Imprime un mensaje indicando la asignación.
3. **Métodos principales**:
    a. **attack(const std::string& target)**:
        i. Verifica si tiene puntos de vida y energía suficientes para atacar.
        ii. Si es posible, reduce los puntos de energía en 2 y muestra un mensaje indicando el ataque y el daño causado.
        iii. Si no tiene vida o energía, imprime un mensaje explicando por qué no puede atacar.
    b. **takeDamage(unsigned int amount)**:
        i. Reduce los puntos de vida según el daño recibido (amount).
        ii. Si el daño excede los puntos de vida, se establecen en 0.
        iii. Imprime un mensaje indicando cuánto daño se recibió.
    c. **beRepaired(unsigned int amount)**:
        i. Aumenta los puntos de vida en la cantidad especificada.
        ii. Imprime un mensaje indicando la reparación realizada.
4. **Getters y setters**:
    a. Métodos como getName, getHitPoints, etc., permiten obtener los valores de los atributos.
    b. Métodos como setName, setHitPoints, etc., permiten modificar los valores.

main.cpp

Propósito

El archivo principal contiene las pruebas para verificar que la clase ClapTrap funciona correctamente.

1. **Crear un objeto ClapTrap**:

ClapTrap Bobby("Bobby");

   a. Se crea un ClapTrap llamado "Bobby". Se llama automáticamente al constructor parametrizado, inicializando sus atributos.
2. **Configurar atributos**:

Bobby.setAttackDamage(2);

   a. Se establece el daño de ataque en 2 puntos.
3. **Mostrar estado inicial**:
   a. Se imprimen los atributos iniciales de "Bobby": puntos de vida, energía y daño de ataque.
4. **Simular acciones**:
   a. **Ataques**: Bobby.attack("Peter Pan");
   Bobby.attack("Bambi");

      i. Bobby intenta atacar, reduciendo su energía en cada ataque.
   b. **Recibir daño**: Bobby.takeDamage(4);
   Bobby.takeDamage(4);

      i. Bobby recibe daño dos veces, reduciendo sus puntos de vida.
   c. **Repararse**: Bobby.beRepaired(10);

      i. Bobby recupera puntos de vida mediante reparación.
5. **Límite de acciones**:
   a. Si "Bobby" se queda sin puntos de vida o energía, no puede atacar.
6. **Finalización**:
   a. Cuando termina el programa, se llama automáticamente al destructor, destruyendo a "Bobby".

Resumen

- **ClapTrap.hpp**: Define la clase ClapTrap, sus atributos y métodos.
- **ClapTrap.cpp**: Implementa la lógica de ClapTrap, incluyendo acciones como atacar, repararse y recibir daño.
- **main.cpp**: Prueba la funcionalidad de ClapTrap mediante una serie de acciones y muestra los resultados por consola.
- En definitiva: el flujo del programa demuestra cómo interactuar con una clase básica en C++, implementando encapsulación, constructores, destructores y métodos.