Now it’s weird!

Explicación del Ejercicio 03: Now it's weird!

En este ejercicio, se debe crear una nueva clase llamada DiamondTrap, que será una mezcla entre FragTrap y ScavTrap, ambas clases heredadas de ClapTrap. Se trata de un caso de **herencia múltiple**, lo que puede traer ciertas complejidades, especialmente con la inicialización de los atributos y la estructura de la jerarquía.

📌 Objetivos del ejercicio:

1. **Crear la clase DiamondTrap** que herede tanto de FragTrap como de ScavTrap.
2. **Gestionar correctamente la herencia múltiple** y evitar problemas como la duplicación de atributos provenientes de ClapTrap.
3. **Mantener una estructura coherente de atributos y funciones** basándose en los siguientes criterios:
    a. **Nombre (name)**: Será un atributo privado de DiamondTrap, y debe llamarse igual que el atributo de ClapTrap.
    b. **Nombre de ClapTrap (ClapTrap::name)**: Se formará concatenando el nombre de DiamondTrap con "_clap_name".
    c. **Puntos de vida (Hit Points)**: Se toman de FragTrap.
    d. **Puntos de energía (Energy Points)**: Se toman de ScavTrap.
    e. **Daño de ataque (Attack Damage)**: Se toma de FragTrap.
    f. **Método attack()**: Se toma de ScavTrap.
4. **Agregar una función especial whoAmI()**, que mostrará tanto el nombre del DiamondTrap como su ClapTrap::name.
5. **Asegurar que ClapTrap solo se inicialice una vez** en la cadena de herencia.

🚀 Cómo abordarlo en C++98

1. **Definir DiamondTrap con herencia múltiple**

2. **Implementar el constructor adecuadamente**
    a. Se debe inicializar correctamente ClapTrap, asegurando que no se llame dos veces al constructor.
    b. Se establece el nombre con el sufijo "_clap_name" para ClapTrap::name.

3. **Implementar la función whoAmI()**

4. **Usar las flags -Wshadow y -Wno-shadow**
    a. -Wshadow: Genera advertencias si una variable local o atributo oculta otra variable con el mismo nombre en una clase base.

b. -Wno-shadow: Desactiva esas advertencias si decides ignorarlas.

☑ Conclusión

Este ejercicio desafía la comprensión de la **herencia múltiple** y el manejo de constructores en C++. La clave está en **inicializar correctamente ClapTrap solo una vez** y organizar bien qué atributos provienen de cada clase base.

Aquí tienes una explicación clara y detallada del código que has proporcionado:

¿Qué hace este código?

Este código implementa una jerarquía de clases en C++ para representar diferentes tipos de robots con características y comportamientos específicos. La clase base es ClapTrap, y a partir de ella se derivan ScavTrap y FragTrap. Finalmente, DiamondTrap hereda de ambas (ScavTrap y FragTrap), combinando sus habilidades.

El programa crea un objeto DiamondTrap en el main.cpp, le asigna un nombre, y lo pone a atacar, recibir daño, repararse, y mostrar su identidad.

Explicación de cada clase y su función

1. ClapTrap (Clase base)

Representa un robot básico con las siguientes características:

- **Atributos protegidos** (pueden ser accedidos por clases derivadas):
  - _name: Nombre del robot.
  - _hitPoints: Puntos de vida.
  - _energyPoints: Puntos de energía.
  - _attackDamage: Daño de ataque.
- **Métodos clave**:
  - attack(target): Reduce los puntos de energía y ataca al objetivo.
  - takeDamage(amount): Reduce los puntos de vida al recibir daño.
  - beRepaired(amount): Recupera puntos de vida.
  - Constructores y destructor (ClapTrap(), ClapTrap(std::string name), ~ClapTrap()).

Ejemplo:

```
ClapTrap robot("Clappy");
robot.attack("Enemigo");
robot.takeDamage(20);
robot.beRepaired(10);
```

2. ScavTrap (Hereda de ClapTrap)

Un robot mejorado con más energía y la habilidad especial guardGate().

- **Herencia**: class ScavTrap : virtual public ClapTrap
- **Nuevo método**:
  - guardGate(): Muestra que ha activado su "modo guardián".
  - Redefine attack(), consumiendo más energía (5 puntos en vez de 2).

Ejemplo:

```
ScavTrap scav("Guardian");
scav.attack("Intruso");
scav.guardGate();
```

3. FragTrap (Hereda de ClapTrap)

Otro robot mejorado con la habilidad highFivesGuys().

- **Herencia**: class FragTrap : virtual public ClapTrap
- **Nuevo método**:
  - highFivesGuys(): Pide un "choca esos cinco".

Ejemplo:

```
FragTrap frag("Amistoso");
frag.highFivesGuys();
```

4. DiamondTrap (Heredero de FragTrap y ScavTrap)

El DiamondTrap es el híbrido de FragTrap y ScavTrap. Implementa:

- **Herencia múltiple**: class DiamondTrap : public FragTrap, public ScavTrap
- **Nuevo atributo**:
  - _name: Nombre propio del DiamondTrap (distinto de ClapTrap::_name).
- **Método especial**:
  - whoAmI(): Muestra su nombre y el de su ClapTrap base.
- **Importante**: Usa ScavTrap::attack(), por lo que su ataque consume 5 puntos de energía.

Ejemplo:

```
DiamondTrap dt("SuperRobot");
dt.attack("Villano");
dt.whoAmI();
dt.highFivesGuys();
dt.guardGate();
```

5. main.cpp: Uso del DiamondTrap

El main.cpp prueba las funciones de DiamondTrap:

```
DiamondTrap Diamond("Damien");

Diamond.attack("The priest");
Diamond.attack("The photographer");
Diamond.attack("The mother");

Diamond.beRepaired(6);
Diamond.takeDamage(66);
Diamond.whoAmI();
Diamond.highFivesGuys();
Diamond.guardGate();
```

Flujo de ejecución esperado

1. Se crea un DiamondTrap llamado "Damien".
2. "Damien" ataca tres veces (cada ataque consume 5 puntos de energía).
3. Se repara (aumenta su vida).
4. Recibe daño (disminuyen sus puntos de vida).
5. Muestra su identidad (whoAmI()).
6. Pide un "choca esos cinco" (highFivesGuys()).

7. Activa el "modo guardián" (guardGate()).

Detalles importantes sobre C++98

Este código está escrito en un estilo compatible con **C++98**, pero hay algunos puntos clave a considerar:

1. **Uso de herencia virtual**: FragTrap y ScavTrap heredan de ClapTrap usando virtual, lo que evita duplicación de datos en DiamondTrap.
2. **Evita std::vector o std::unique_ptr**: En C++98 no hay std::unique_ptr, y la gestión de memoria dinámica requiere new y delete.
3. **Inicialización en constructores**: En C++98, la inicialización en lista (: ClapTrap(name)) es preferible en vez de hacerlo dentro del constructor.
4. **Uso de using**:
    a. using ClapTrap::operator =; hace que DiamondTrap herede el operador = de ClapTrap sin necesidad de reescribirlo.

Conclusión

Este código demuestra cómo usar **herencia múltiple** y **herencia virtual** en C++ para combinar características de distintas clases. DiamondTrap es un híbrido con los mejores atributos de ScavTrap y FragTrap, permitiendo realizar ataques, pedir choca esos cinco y activar el modo guardián.