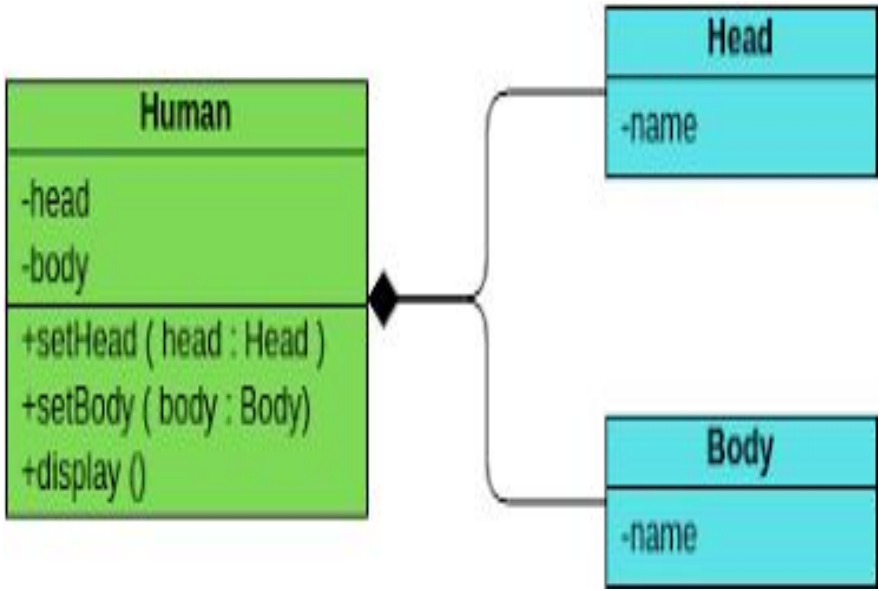
Composición

Composición

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.



Composición

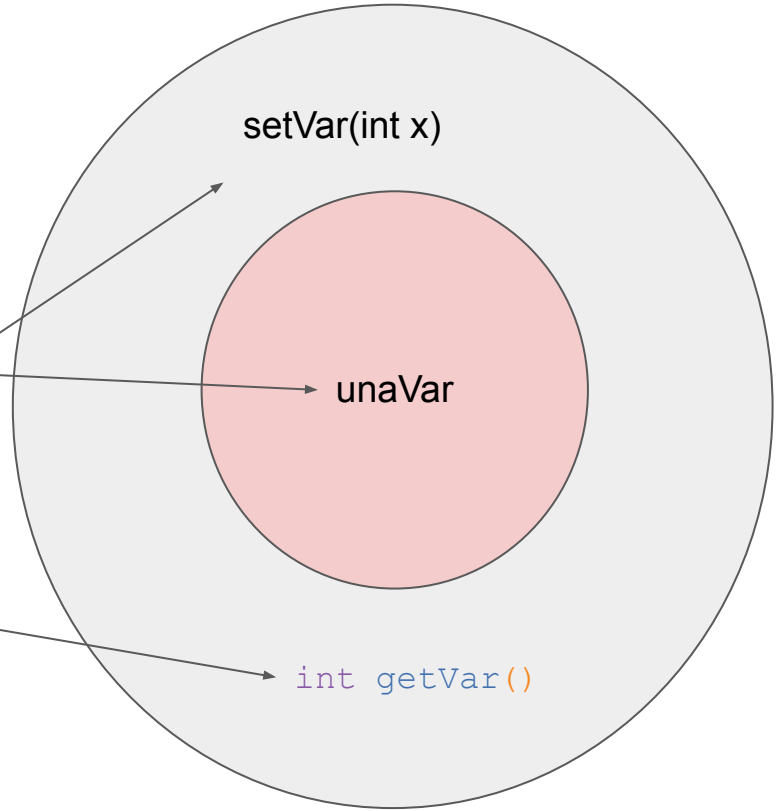
Otra forma conocida de reutilizar código que se ha implementado previamente en otras clases es a través de lo que se conoce como Composición de Clases. Es similar a la herencia con la principal excepción (entre algunas otras) de que no se utiliza una sintaxis limpia para indicar que se está implementando una composición, simplemente se declaran objetos de distintas clases (comúnmente conocidos como partes) al interior de la definición de una nueva clase (conocida como el todo). De esta forma y de manera indirecta la nueva clase adhiere las propiedades y métodos de las clases que la componen.

Composición

Se puede observar que la composición no solo contribuye a la reusabilidad de código existente sino que también hace que el código sea escalable ya que al no existir una relación estrecha de dependencia entre las clases se pueden implementar cambios en clases individuales para que se vean reflejados en las clases compuestas por objetos de esas clases.

Ejemplo Clase UnaClase

```
class UnaClase
{
    private:
    int unaVar;
    public:
    void setVar(int x){unaVar = x;}
    int getVar(){return unaVar;}
};
```



OtraClase

```
class OtraClase

{

    private:

    UnaClase unObj;

    public:

    void setVar(int x){unObj.setVar(x);}

    int getVar(){return unObj.getVar();}


};
```

Clase Otra Clase Mas

```
class OtraClaseMas

{

    private:

    UnaClase otroObj;

    public:

    void setVar(int x){otroObj.setVar(x);}

    int getVar(){return otroObj.getVar();}


};
```

Función Principal

```
int main()
{    OtraClase obj;
     OtraClaseMas ojt;
     obj.setVar(5);
     ojt.setVar(25);
     cout<<obj.getVar()<<endl;
     cout<<ojt.getVar()<<endl;
     return 0;}
```

Problema 1:

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositada.

Cliente
- atributos
  - nombre
  - monto
- métodos
  - constructor
  - depositar
  - extraer
  - retornarMonto

Banco
- atributos
  - 3 Cliente (3 objetos de la clase Cliente)
- métodos
  - constructor
  - operar
  - depositosTotales

Problema 2:

Plantear un programa que permita jugar a los dados. Las reglas de juego son: se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "gano", sino "perdió"

Dado
- atributos
  - valor
- métodos
  - tirar
  - imprimir
  - retornarValor

JuegoDeDados
- atributos
  - 3 Dado (3 objetos de la clase Dado)
- métodos
  - jugar

Paso de objetos temporales como parámetros de entrada de funciones (métodos).

Const

El paso de parámetros por referencia con el calificador const, tal cual como se pasa un objeto a un constructor de copia, permite hacer el paso de objetos temporales. Un objeto temporal es un objeto que no se declara ya que solo se invoca a través de una expresión o como parámetro de entrada de una función y cuyo tiempo de vida es muy corto. En particular la duración de un objeto temporal es el mismo de la ejecucion de la expresion que lo invoca. Lo anterior quiere decir que un objeto temporal solo existe mientras la operación o función que lo invoca toma su valor y luego es destruido en memoria.

objetos temporales

Los objetos temporales solo se pueden pasar como parámetros de entrada a funciones que tienen un paso por referencia a objetos de cierta clase calificados con const. Observe el siguiente ejemplo en el que se pasa un objeto temporal al constructor de otro objeto de una clase compuesta, ponga especial atención en la ejecución del constructor y el destructor del objeto temporal de la clase UnaClase (sí, son las dos líneas que se imprimen antes del "5". de la clase "compuesta")

Ejemplo

```
class UnaClase
{
    public:
    int unaVar;
    UnaClase(int x):unaVar(x){cout<<"UnaClase construido"<<endl;}
    ~UnaClase(){cout<<"UnaClase destruido"<<endl;}
};
```

Ejemplo

```
class OtraClase
{
    private:
    UnaClase unObj;
    public:
    /* Inicialización del miembro unObj, utilizando el constructor de la clase UnaClase */
    OtraClase(const UnaClase& oVar):unObj(oVar.unaVar){}
    int getVar(){return unObj.unaVar;}
};
```

Ejemplo

```
int main()

{

    OtraClase obj(UnaClase(5)); /* Paso por referencia de un objeto temporal de la clase UnaClase */

    cout<<obj.getVar()<<endl;


    return 0;

}
```

Bibliografia

https://www.codingame.com/playgrounds/50747/herencia-en-c-practica-3/composicion-de-clases

https://titiushko.github.io/Tutoriales-Ya/www.tutorialesya.com.ar/cmasmasya/detalleconcepto953a.html?punto=26&codigo=157&inicio=15

https://titiushko.github.io/Tutoriales-Ya/www.tutorialesya.com.ar/cmasmasya/detalleconcepto1fc5.html?punto=23&codigo=154&inicio=15