UNIVERSIDAD NACIONAL DEL ALTIPLANO
Facultad de Ingeniería Mecánica Eléctrica,
Electrónica y Sistemas
Escuela Profesional de Ingeniería de Sistemas

EVALUACIÓN DE PROGRAMACIÓN ORIENTADA A OBJETOS EN C++

Docente: Mg. Aldo Hernan Zanabria Galvez
Alumno: Yoel Nhelio Canaza Chagua

Curso:
Programación Orientada a Objetos II

CICLO III – SEMESTRE 2023 – II
PUNO, PERÚ
2023

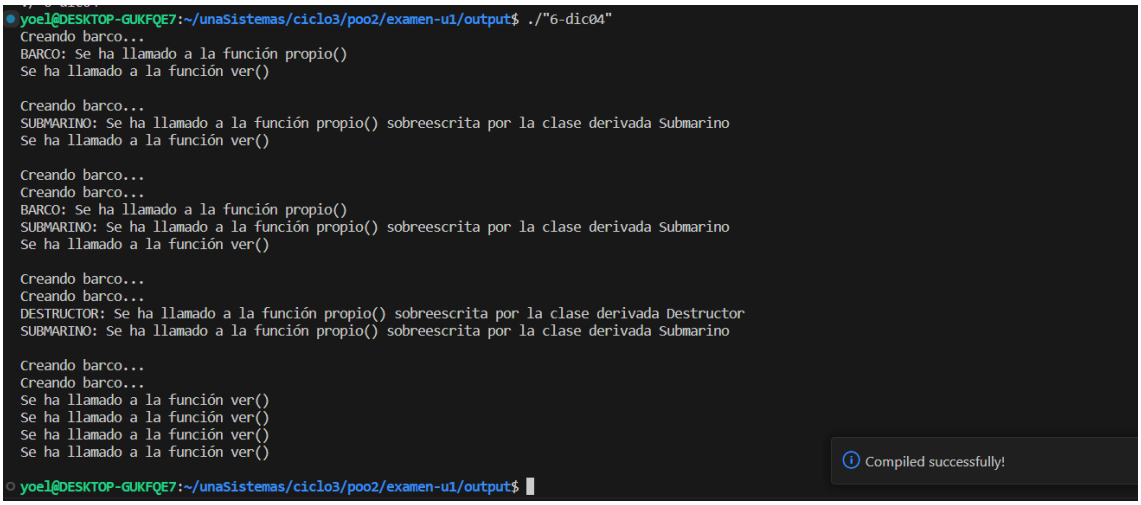
1. Ejercicio 6. DIC04

1.1. 1.a Especifica qué pasará en los siguientes casos, suponiendo que propio() está sobreescrito en las clases derivadas

```
#include <iostream>

using namespace std;

class Barco
{
public:
    Barco();
    virtual ~Barco();
    virtual void propio() const;
    void ver();

    static int obtenerNumeroBarcos();
protected:
    static int numeroBarcos;
};

int Barco::numeroBarcos = 0;

Barco::Barco()
{
    cout << "Creando barco... " << endl;
    numeroBarcos += 1;
}

Barco::~Barco()
{
    cout << "Eliminando barco... " << endl;
    numeroBarcos -= 1;
}

void Barco::propio() const
{
    cout << "BARCO: Se ha llamado a la funcion propio()" << endl;
}

int Barco::obtenerNumeroBarcos()
{
    return numeroBarcos;
}

void Barco::ver()
{
    cout << "Se ha llamado a la funcion ver()" << endl;
}



class Submarino : public Barco
{
public:
    Submarino() {}
    void propio() const override;

protected:

};

```

```
59 void Submarino::propio() const
60 {
61     cout << "SUBMARINO: Se ha llamado a la funcion propio() sobreescrita
       por la clase derivada Submarino" << endl;
62 }
63
64
65
66 class Destructor : public Barco
67 {
68 public:
69     Destructor() {}
70     void propio() const override;
71 protected:
72 };
73
74 void Destructor::propio() const{
75     cout << "DESTRUCTOR: Se ha llamado a la funcio1n propio() sobreescrita
       por la clase derivada Destructor" << endl;
76 }
77
78 int main()
79 {
80     // 1.a Especifica que pasara en los siguientes casos, suponiendo que
       propio() esta sobreescrito en las clases derivadas
81     // a
82     Barco* a = new Barco();
83     a->propio();
84     a->ver();
85     cout << endl;
86     // b
87     Barco* b = new Submarino();
88     b->propio();
89     b->ver();
90     cout << endl;
91
92     // c
93     Barco* c[] = {new Barco(), new Submarino()};
94     c[0]->propio();
95     c[1]->propio();
96     c[0]->ver();
97     cout << endl;
98
99     // d
100    Barco* d[] = {new Destructor(), new Submarino()};
101    d[0]->propio();
102    d[1]->propio();
103    cout << endl;
104
105    // e: supongamos que anadimos ver() heredado del padre tanto en
       submarino como en destructor;
106
107    Barco* e[] = {new Destructor(), new Submarino()};
108    ((Destructor *)e[0])->ver();
109    ((Submarino *)e[1])->ver();
110    e[0]->ver();
111    e[1]->ver();
112    cout << endl;
113
114
115
116 }
```

Listing 1: 6-DIC04



1.1.1. a.

Se crea un objeto a se la clase Barco y se llama a las funciones propio y ver, como estas funciones pertenecen a la clase base se muestran las funciones sin modificar.

1.1.2. b.

Se crea un objeto a se la clase Submarino y se llama a las funciones propio y ver, como estas funciones pertenecen a la clase derivada Submarino se muestra la funcion modificada de subamarino.

1.1.3. c.

Se crea un array de objetos a se la clase Barco y Submarino y se llama a las funciones propio y ver, como funciones pertenecen a la clase derivada Submarino se muestra la funcion modificada de subamarino. Sin embargo en el caso del objeto de la clase Barco, se muestra la función por defecto

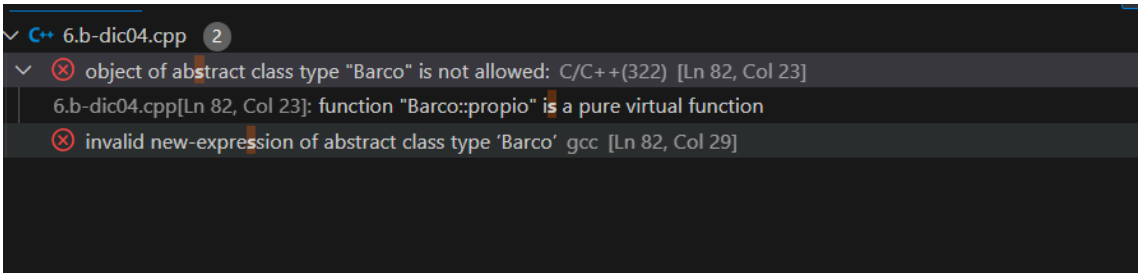
1.1.4. d.

Se crea un array de objetos a se la clase Barco y Submarino y se llama a las funciones propio y ver, como funciones pertenecen a clases derivadas, se muestran las implementaciones que estos tienen para dichas funciones.

1.2. 1.b Supongamos que la asignatura del método propio es la siguiente: virtual void Barco::propio()=0;

```
int main()
{
    // 1.b: Supongamos que la asignatura del método propio es la siguiente:
    // virtual void Barco::propio()=0;
    Barco* f[] = {new Barco(), new Submarino()};
    f[0]->propio();
    f[0]->ver();
    f[1]->propio();
    f[1]->ver();
    cout << endl;
}
```

Si agregamos el anterior codigo teniendo void Barco::propio()=0; obtendremos el siguiente error:



Este error generalmente se produce cuando intentamos crear una instancia (un objeto) de una clase abstracta directamente. Las clases abstractas son aquellas que tienen al menos una función virtual pura (una función virtual que se declara con = 0 y no se implementa en la clase base). No se pueden crear instancias de clases abstractas, ya que no tienen una implementación completa de todas sus funciones virtuales.

2. 12 SEP06: Define una funcion genérica llamada 'Intercambio que permita intercambiar el valor de dos objetos del mismo tipo. Indica qué restricciones debe cumplir el tipo de los objetos para poder usar a función de intercambio con ellos

```
1  //Ejercicio 12-sep06
2  //Ejercicio 12-sep06
3  #include <iostream>
4  #include <string>
5
6  using namespace std;
7
8
9  class Carta {
10 public:
11     Carta(string valor, string palo) : valor_(valor), palo_(palo) {}
12
13
14     friend ostream& operator<<(ostream& os, const Carta& carta) {
15         os << "Carta: " << carta.valor_ << " de " << carta.palo_;
16         return os;
```

```
    }

private:
    string valor_;
    string palo_;
};

template <typename T>
void Intercambio(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    Carta As1("As", "Corazones");
    Carta As2("As", "Treboles");

    cout << "Antes del intercambio: " << endl;
    cout << As1 << endl;
    cout << As2 << endl;


    Intercambio(As1, As2);

    cout << "Despues del intercambio: " << endl;
    cout << As1 << endl;
    cout << As2 << endl;

    return 0;
}
```

Listing 2: 12 SEPT06

Los objetos deben ser del mismo tipo

```
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/examen-u1/output$ ./"12-sep06"
Antes del intercambio:
Carta: As de Corazones
Carta: As de Treboles
Despues del intercambio:
Carta: As de Treboles
Carta: As de Corazones
yoel@DESKTOP-GUKFQE7:~/unaSistemas/ciclo3/poo2/examen-u1/output$
```