Instituto Politecnico Nacional

ESCOM “ESCUELA SUPERIOR DE CÓMPUTO”

ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

TAREA1: ATRIBUTOS FUNDAMENTALES ORIENTADO A OBJETOS

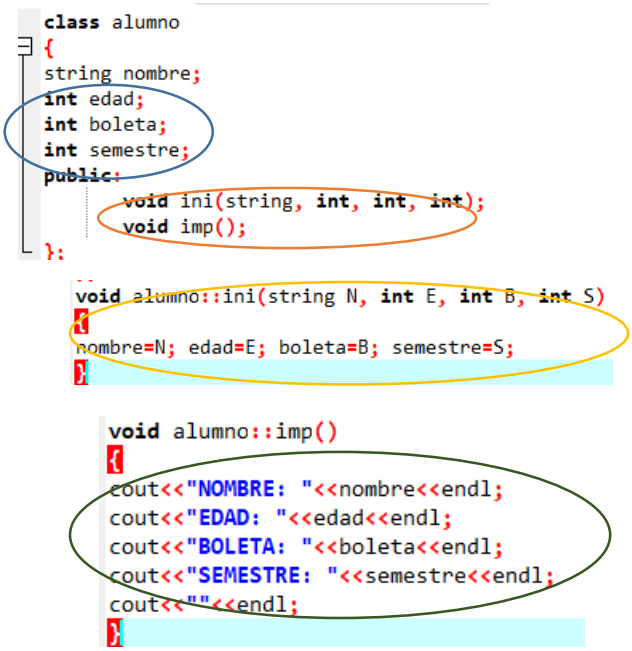
PROFA: Reyna Melara Abarca

ALUMMNO: Rojas Alvarado Luis Enrique

GRUPO: 2CM9

Objeto:

Se trata de un ente abstracto usado en programación que permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras. Los objetos integran, a diferencia de los métodos procedurales, tanto los procedimientos como las variables y datos referentes al objeto. A los objetos se les otorga ciertas características en la vida real. Cada parte del programa que se desea realizar es tratado como objeto, siendo así estas partes independientes las unas de las otras. Los objetos se componen de 3 partes fundamentales: métodos, eventos y atributos.



```
class alumno
{
string nombre;
int edad;
int boleta;
int semestre;
public:
    void ini(string, int, int, int);
    void imp();
};
```

```
void alumno::ini(string N, int E, int B, int S)
{
nombre=N; edad=E; boleta=B; semestre=S;
}
```

```
void alumno::imp()
{
cout<<"NOMBRE: "<<nombre<<endl;
cout<<"EDAD: "<<edad<<endl;
cout<<"BOLETA: "<<boleta<<endl;
cout<<"SEMESTRE: "<<semestre<<endl;
cout<<""<<endl;
}
```

Métodos:

Son aquellas funciones que permite efectuar el objeto y que nos rinden algún tipo de servicio durante el transcurso del programa. Determinan a su vez como va a responder el objeto cuando recibe un mensaje.

Eventos:

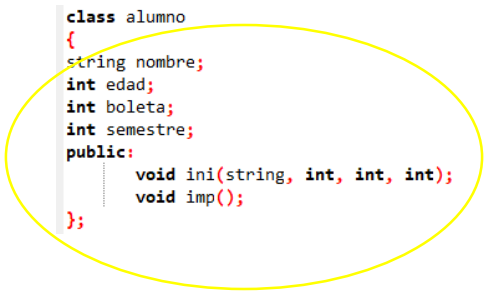
Son aquellas acciones mediante las cuales el objeto reconoce que se está interactuando con él. De esta forma el objeto se activa y responde al evento según lo programado en su código.

Atributos:

Características que aplican al objeto solo en el caso en que él sea visible en pantalla por el usuario; entonces sus atributos son el aspecto que refleja, tanto en color, tamaño, posición, si está o no habilitado.

Clase:

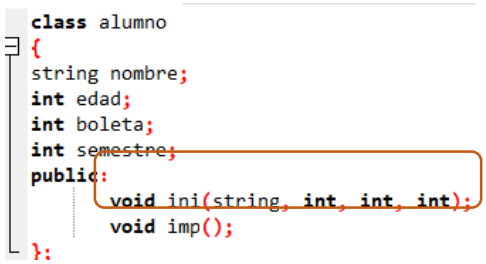
Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje. Cada clase es un modelo que define un conjunto de variables -el estado, y métodos apropiados para operar con dichos datos -el comportamiento. Cada objeto creado a partir de la clase se denomina instancia de la clase.



```
class alumno
{
string nombre;
int edad;
int boleta;
int semestre;
public:
        void ini(string, int, int, int);
        void imp();
};
```

Encapsulación:

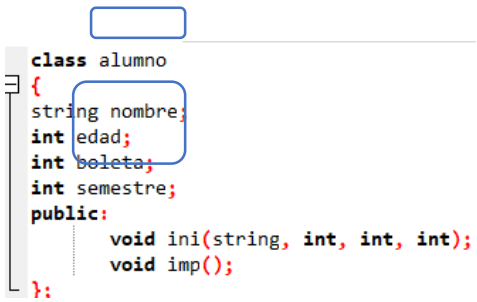
Define el comportamiento de una clase u objeto que tiene dentro de él todo tipo de métodos y datos pero que solo es accesible mediante el paso de mensajes. y los datos a través de los métodos del objeto/clase.



```
class alumno
{
string nombre;
int edad;
int boleta;
int semestre;
public:
        void ini(string, int, int, int);
        void imp();
};
```

Abstracción:

Las características específicas de un objeto, aquellas que lo distinguen de los demás tipos de objetos y que logran definir límites conceptuales respecto a quien está haciendo dicha abstracción del objeto.

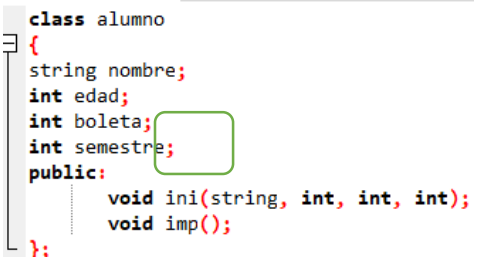


```
class alumno
{
string nombre;
int edad;
int boleta;
int semestre;
public:
        void ini(string, int, int, int);
        void imp();
};
```

Cohesión

La medida que indica si una clase tiene una función bien definida dentro del sistema. El objetivo es enfocar de la forma más precisa posible el propósito de la clase. Cuanto más enfoquemos el propósito de la clase, mayor será su cohesión.

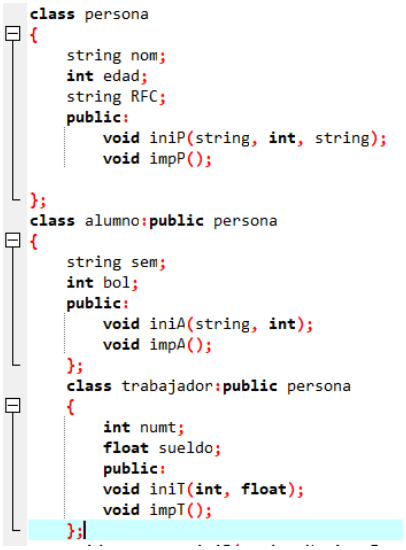
Herencia



```
class alumno
{
string nombre;
int edad;
int boleta;
int semestre;
public:
        void ini(string, int, int, int);
        void imp();
};
```

La herencia es la transmisión del código entre unas clases y otras. Para soportar un mecanismo de herencia tenemos dos clases: la clase padre y la/s clase/s hija/s. La clase padre es la que transmite su código a las clases hijas. En muchos lenguajes de programación se declara la herencia con la palabra "extends". El siguiente ejemplo también puede abarcar las clases abstractas.

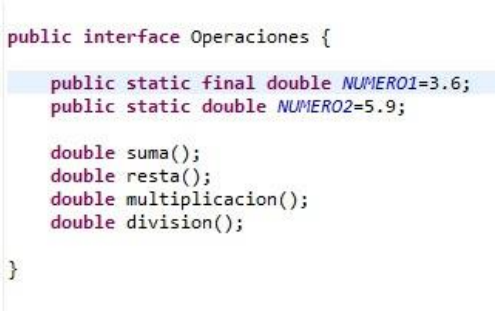
Interfaz



```
class persona
{
    string nom;
    int edad;
    string RFC;
    public:
        void iniP(string, int, string);
        void impP();

};
class alumno:public persona
{
    string sem;
    int bol;
    public:
        void iniA(string, int);
        void impA();
    };
    class trabajador:public persona
    {
        int numt;
        float sueldo;
        public:
        void iniT(int, float);
        void impT();
    };
```

Una interfaz es un conjunto de métodos abstractos y de constantes cuya funcionalidad es la de determinar el funcionamiento de una clase, es decir, funciona como un molde o como una plantilla. Al ser sus métodos abstractos estos no tiene funcionalidad alguna, sólo se definen su tipo, argumento y tipo de retorno.



```
public interface Operaciones {

    public static final double NUMERO1=3.6;
    public static double NUMERO2=5.9;

    double suma();
    double resta();
    double multiplicacion();
    double division();

}
```

Ésta interfaz tiene: 2 Atributos constantes NUMERO1 y NUMERO2. Se observa que un atributo es declarado como "final" y otro no, pues bien, aunque no se lo pongamos, java nos lo incluye implícitamente, con lo cual, aunque o esté puesto, sigue siendo "final". 4 métodos. Dijimos anteriormente que los métodos de una interfaz deben ser "public" y "abstract", yo en este ejemplo no los he puesto, ya que, como en el caso de los atributos, java nos pone los métodos "public" y "abstract" implícitamente.

Polimorfismo:

Los objetos responden a los mensajes que se les envían. Un mismo mensaje puede ser interpretado o dar paso a distintas acciones según que objeto es el destinatario.
Con este sistema el emisor se desentiende de los detalles de la ejecución (aunque el programador ha de saber en todo momento cuales son las consecuencias de ese mensaje).

```
void funcion(int)          { std::cout << "Entero\n"; }
void funcion(unsigned int) { std::cout << "Entero sin signo\n"; }
```

La función *función* sería polimórfica ya que tomaría *una forma diferente* según cómo sea llamada.

```
funcion(0123);    // Muestra "Entero"
funcion(0xcafeu); // Muestra "Entero sin signo"
```

Acoplamiento:

Es el nivel de dependencia que tiene una clase de los detalles de implementación de otra. Así, cuanto más necesite saber una clase sobre cómo hace otra las cosas internamente más acopladas estarán.

```
    cout<<"Elige una opcion"<<endl;
    cout<<"1.- Persona "<<endl;
    cout<<"2.- Alumno "<<endl;
    cout<<"3.- Trabajador"<<endl;
    cout<<"4.- SALIR..."<<endl;
    cin>>opc;
    system("cls");
    switch (opc)
    {
    case 1:
    sujeto.iniP("Veronica", 18, "VERO140818");
    sujeto.impP();
       system("pause");
       system("cls");
       break;
       case 2:
    estudiante.iniP("Ramon", 19, "RAM201023");
    estudiante.iniA("2CV7", 11481212);
    estudiante.impP();
    estudiante.impA();
system("pause");
       system("cls");
       break;
       case 3:
    empleado.iniP("Jorge", 20, "JOR630221");
    empleado.impP();
    empleado.iniT(200, 12000);
    empleado.impT();
       system("pause");
       system("cls");
```