Programación Orientada a Objetos

Unidad 2: Introducción a la POO

Teoría 02 - 25/08/2011 - Pablo Novara

Un poco de historia...

¿Qué es el software?

Orígenes, ensamblador

Surgimiento de los lenguajes

Evolución del hardware
Aumento de la complejidad
Crisis del Software

Del arte a la industria
Ingeniería de Software

Presente
¿arte o industria?
¿claridad o eficiencia?

¿Qué es la POO?

La Programación Orientada a Objetos es un paradigma que utiliza objetos como elementos fundamentales en la construcción de la solución.

Ejemplo 1: de PME a POO

Escriba un programa que lea los datos de un alumno: nombre, apellido, y una lista de calificaciones correspondiente a todos sus exámenes finales (se desconoce a priori el número de exámenes rendidos).

El programa debe calcular el promedio del alumno en la carrera en informarlo en pantalla.

PME vs POO

Prog. Modular y Estructurada:

Algoritmos + Estructuras de datos

Prog. Orientada a Objetos:

Objetos = datos+funcionalidades

¿Qué son los objetos?

Un **objeto** es una **abstracción de algún hecho o ente** del mundo real que **tiene atributos** que representan sus características o propiedades y **métodos** que representan su comportamiento o acciones que realizan.

Todas las propiedades y métodos comunes a los objetos se encapsulan o se agrupan en **clases**.

Una **clase** es una plantilla o un prototipo para crear objetos, por eso se dice que los objetos son instancias de clases.

PME vs POO

+ Requiere y permite mayor nivel de abstracción

+ Representa más fielmente los entes del mundo real

+ Facilita y fomenta la reutilización de código

+ Mejora la productividad y la comunicación

- Es más sensible al diseño inicial

Características de la POO

Modularidad: división en Clases/Objetos

Abstracción: modelo simplificado

Ocultación: cada objeto es responsable de su estado

Jerarquía: Herencia y Composición

Polimorfismo: un método varía según el tipo de objeto

Principio de ocultación:

Cada objeto está aislado del exterior, **y expone una interfaz** a otros objetos que especifica cómo pueden interactuar.

El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado.

¿Qué son los objetos?

Objeto: entidad provista de un conjunto de datos/atributos y comportamiento/funcionalidad

En C++: datos=atributos, comportamiento=metodos

Clase: definición de las propiedades y comportamientos de un tipo de objeto concreto.

Instanciación: creación de un objeto a partir de la definición de una clase.

Interacción entre objetos

Los objetos se comunican entre sí mediante mensajes.

Un objeto provee servicios. Otro objeto, el cliente, solicita el servicio enviando un mensaje.

En C++, el mensaje consiste en la invocación de un método.

¿Cómo identificar los objetos en un problema?

Todos los nombres del enunciado son candidatos:

- Cosas tangibles (Auto,Arma)
- Roles o papeles (Alumno,Empleado)
- Organizaciones (Empresa,Facultad)
- Incidentes o sucesos (Liquidacion,Examen)
- Interacciones o relaciones (Pedido,Alquiler)

¿Cómo identificar atributos y métodos?

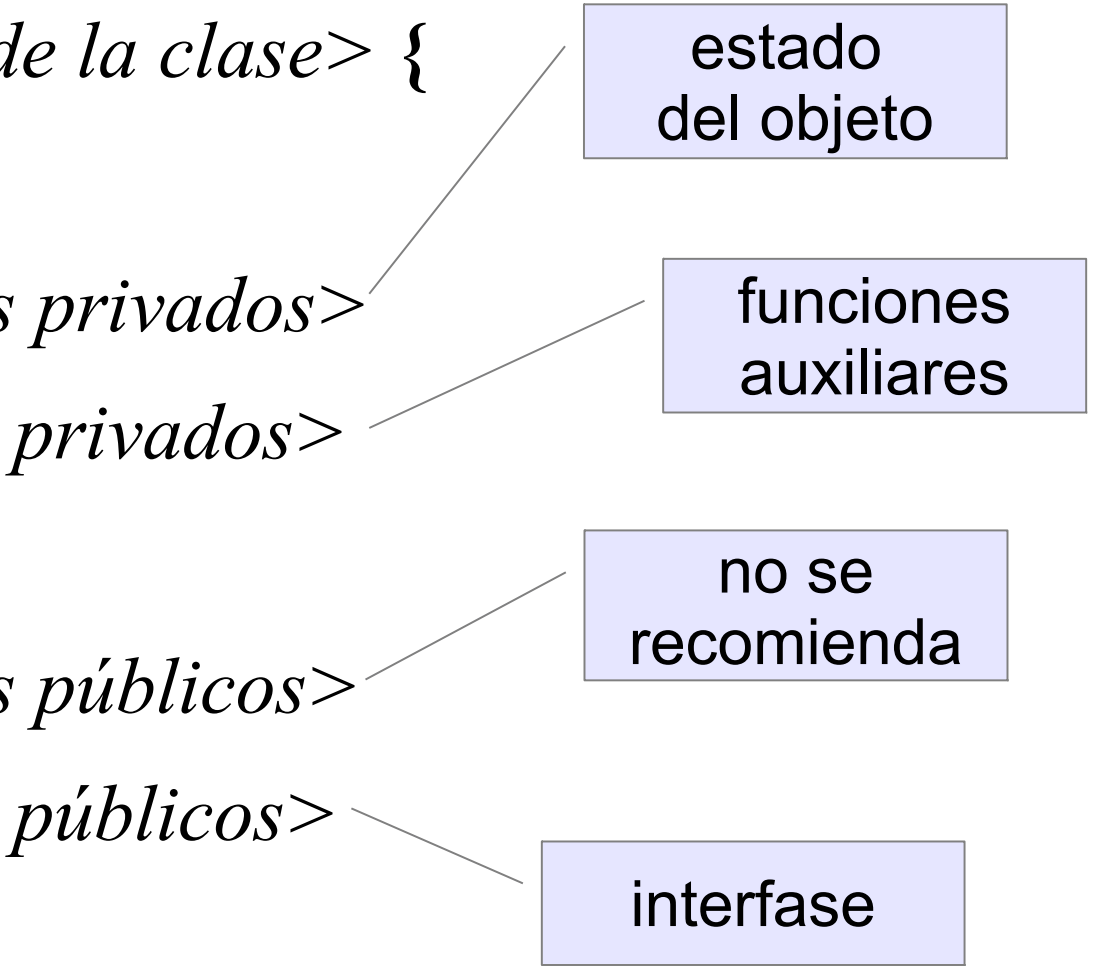
Atributos: características individuales, (adjetivos y complementos del verbo en el enunciado).

Métodos: verbos

- Constructor
- Destructor
- Modificadores del estado
- Selectores
- Mezcladores
- Cálculos o Procesamiento

Objetos en C++

```
class <nombre de la clase> {
    private:
        <atributos privados>
        <métodos privados>
    public:
        <atributos públicos>
        <metodos públicos>
};
```



Objetos en C++

```
class Ecuacion {
    float a,b,c;
    float r1,r2;
    bool reales;
public:
    void CargarCoefs(int _a, int _b, int _c);
    void Calcular();
    bool TieneRaicesRelaes();
    float VerRraiz1();
    float VerRaiz2();
    float VerParteReal();
    float VerParteImag();
};
```

atributos cargados por el cliente

atributos calculados

interfaz para cargar datos

método para procesar los datos

métodos para obtener los resultados

```
void Ecuacion::CargarCoefs(int _a, int _b, int _c) {
    a=_a; b=_b; c=_c;
}
void Ecuación::Calcular() {
    float d = b*b-4*a*c;
    if (d>=0) {
        reales=true;
        r1=(-b+sqrt(d))/(2*a); r2=(-b-sqrt(d))/(2*a);
    } else {
        reales=false;
        r1=-b/(2*a); r2=sqrt(-d)/(2*a);
    }
}
```

Objetos en C++

```
int main() {
    Ecuación eq;
    float x,y,z;  cin>>x>>y>>z;
    eq.CargarCoefs(x,y,z);
    eq.Calcular();
    if (eq.TieneRaicesReales()) {
        cout<<"r1="<<eq.VerRaiz1()<<endl;
        cout<<"r2="<<eq.VerRaiz2()<<endl;
    } else {
        cout<<"r1="<<eq.VerParteReal()<<"+"
            <<eq.VerParteImag()<<"i"<<endl;
        cout<<"r1="<<eq.VerParteReal()<<"-"
            <<eq.VerParteImag()<<"i"<<endl;
}
```

se crea la instancia

se definen sus atributos

se pide que realice el cálculo

se consultan los resultados

Constructores y Destructores

Constructor:

- método que se invoca automáticamente al crear un objeto.
- pueden recibir argumentos y sobrecargarse
- si no se especifica ninguno, c++ otorga por defecto un constructor nulo y un constructor de copia

Destructor:

- método que se invoca automáticamente al destruir un objetos.
- no puede recibir parámetros

Constructores y Destructores en C++

```
class Ecuación {
    …
public:
    Ecuación(float _a, float _b, float _c);

    ...
};


int main() {
    float x,y,z; cin>>x>>y>>z;
    Ecuación eq(x,y,z);

    ...
```

definición de un constructor

invocación del constructor

Constructores y Destructores en C++

```
class Vector {
    int *x, n; // puntero a los datos y cantidad
public:
    Vector() { x=NULL; n=0; }
    Vector(int _n) { x=new int[_n]; n=_n; }
    Vector(Vector &v2) {
        x=new int[v2.n];
        n=v2.n;
        for(int i=0;i<n;i++) x[i]=v2.x[i];
    }
    …otros métodos...
    ~Vector() { delete [] x; }
};
```

constructor nulo

constructor que recibe el tamaño

constructor de copia

destructor