

intro à POO

¿que es?

Problema
Real

← →
largo Camino...

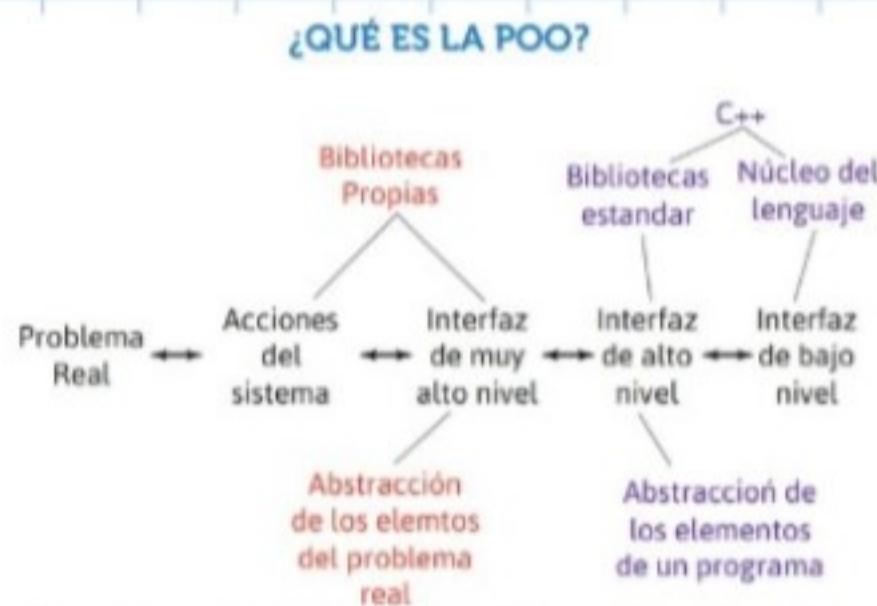
interfaz de bajo
Nivel

Mecanismos de abstracción

que es la POO?

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

El objetivo es describir el problema y plantear la solución en los términos del problema mismo y no de elementos computacionales



resumen: una simplificación de la realidad, lo mas relevante

¿Qué es la POO?

¿QUÉ ES LA POO?



¿Qué son los objetos?

objeto:

entidad que tiene un conjunto de datos y funcionalidades o comportamientos

en C++

datos = atributos o variables a miembro

comportamiento = métodos o funciones miembro

clase : definición de las propiedades y comportamientos de un tipo de objeto concreto

instanciamiento: creación de un objeto a partir de la definición de una clase

Su composición:

`class <nombre de la clase> {`

`public:`

lo que sí se ve desde afuera del objeto

`private:`

lo que no se ve desde afuera del objeto

`};`

El Principio de Ocultación:

Cada objeto está aislado del exterior, y expone solo una interfaz que especifica cuando puede interactuar

El aislamiento protege a las propiedades internas de un objeto, garantizando su integridad (invariantes).

Nota:

En la mayoría de los casos, todos los atributos de un objeto deberían ser privados, exponiendo solamente métodos públicos al exterior.

Objetos en C++

```
class <nombre de la clase> {  
  
public:  
    <métodos públicos>  
        interfaz del objeto  
    <atributos públicos>  
        mala idea  
  
private:  
    <atributos privados>  
        estado del objeto  
    <métodos privados>  
        funciones auxiliares  
  
};
```

utilización de objetos:

```
float x, y, z;  
cin >> x >> y >> z;  
// se crea el objeto y se definen sus atributos iniciales  
Ecuacion eq;  
eq.CargarCoefs(x,y,z);  
// se opera con el objeto y/o consulta su estado  
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;  
}
```

ejemplo de Realización:

```
float x, y, z;  
cin >> x >> y >> z;  
// se crea el objeto y se definen sus atributos iniciales  
Ecuacion eq;  
eq.CargarCoefs(x,y,z);  
// se opera con el objeto y/o consulta su estado  
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;  
}
```

def. de las clases:

```
class Ecuacion {  
public:  
    // interfaz para carga de datos  
    void CargarCoefs(float a, float b, float c);  
    // interfaz para consulta de resultados  
    bool TieneRaicesReales();  
    float CalcularRaiz1(); // si son reales  
    float CalcularRaiz2();  
    float CalcularParteReal(); // si son complejas  
    float CalcularParteImag();  
private:  
    // atributos cargados por el usuario  
    float m_a, m_b, m_c;  
    // método auxiliar para resolver los cálculos  
    float Discriminante();  
};
```

Yo No lo Suelo hacer así

implementacion de metodos

dentro de la clase:

```
class Ecuacion {  
public:  
    ...  
    void CargarCoefs(float a, float b, float c) {  
        m_a = a; m_b = b; m_c = c;  
    }  
    ...  
private:  
    float m_a, m_b, m_c;  
    ...  
};
```

Fuera de la clase:

```
class Ecuacion {  
public:  
    ...  
    void CargarCoefs(float a, float b, float c); //solo prototipo  
};  
...  
void Ecuacion::CargarCoefs(float a, float b, float c) {  
    m_a = a; m_b = b; m_c = c;  
}
```

Ej. de Funcion: (metodo)

EJEMPLOS DE CLASES/OBJETOS

Cosas tangibles: Auto, Arma, Disparo

Roles o papeles: Alumno, Empleado

Organizaciones/Sistema: Empresa, Universidad, Juego

Incidentes o sucesos: Liquidación, Examen

Interacciones o relaciones: Pedido, Alquiler

Abstracciones de más bajo nivel: Vector, Matriz, Archivo

EJEMPLOS DE MÉTODOS

Definir/modificar el estado:

void Alumno::ActualizarEMail(string nueva_dir_correo);

void Personaje::SumarPuntos(int pts_ganados);

void Tecla::Apretar(); void Tecla::Soltar();

void Curso::InscribirAlumno(string nombre);

Consultar el estado

string Alumno::VerTelefono();

int Personaje::VerVidas();

bool Tecla::EstaApretada();

int Curso::CantidadDeInscriptos();

string Curso::NombreAlumno(int i);

algunos ejemplos

CONSTRUCTORES Y DESTRUCTORES

Constructor:

método especial que se invoca automáticamente al crear un objeto tiene el mismo nombre que la clase puede recibir argumentos y sobrecargarse si no se especifica ninguno, c++ otorga dos por defecto:

constructor nulo: no hace " nada " (también llamado " por defecto")

constructor de copia: copia uno por uno los atributos desde otro objeto de la misma clase

Destructor:

método que se invoca automáticamente al destruir un objeto

tiene por nombre el carácter ~ más el nombre que la clase

no puede recibir parámetros

por defecto no hace

"

nada

"

Los constructores y destructores generados por el compilador equivalen aproximadamente a:

```
class Ecuacion {  
public:  
    Ecuacion() { /* naaada */ }  
    Ecuacion(const Ecuacion &o) {  
        /** copia atributo por atributo **/  
        m_a=o.m_a; m_b=o.m_b; m_c=o.m_c;  
    }  
    ~Ecuacion() { /* naaada */ }  
    // ... varios métodos públicos...  
private  
    float m_a, m_b, m_c;  
};
```

► constructor
Por defecto

► destructor
Por
defecto

