

1.0 Presentación

Metodologías y Técnicas de Programación II Programación Orientada a Objeto (POO)

C++

Profesor:

José Luis Marina
jmarina@nebrija.es

Laboratorios:

José Luis Marina
jmarina@nebrija.es
Borland Builder 6.0

Puntuación:

Prácticas Laboratorio:	20%
Diarias	20%
Trabajo Laboratorio	80%

Exámen Parcial	15%
----------------	------------

Exámen Final	65%
--------------	------------

Exámen Final Extraordinario	70%
------------------------------------	------------

1.0 Presentación: Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

C++
Mi primera Clase

Repaso de Conceptos

Estándares de Programación
Punteros y Memoria

Ficheros y E/S
Control y Operadores

Clases y Objetos en C++

Uso y aplicación
Constructores

Funciones Amigas
Constantes e "inline"

Sobrecarga

De Operadores

De Funciones

Herencia.

Tipos de Visibilidad

Herencia Múltiple

Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

Plantillas

Contenedores

Iteradores

1.1 Historia de la Programación

¿Qué es programar?

Programar es tener en cuenta



- Un **problema**:
Quiero saber mi nota final en la asignatura.
- Un Conjunto de **Datos**

En las prácticas diarias tengo un	:	8
En la práctica final tengo un	:	9
En el exámen parcial un	:	6
En el exámen final un	:	7
- Unas **funciones** o algoritmos
$$\text{Nota Prácticas} = (20\% \text{ Prac.D}) + (80\% * \text{Prac.F})$$
$$\text{Nota Final} = 20\% \text{ Nota.Prac.} + 15\% \text{ Parcial} + 65\% \text{ Final}$$

Y entonces...

Aplicar las funciones para resolver el problema (7,21)

1.1 Historia de la Programación

¿Qué es programar?



Queremos por supuesto que nuestro programa:

- Sea **Correcto**:
Funciona bien y de acuerdo a los requisitos.
El resultado se corresponde con unas especificaciones.

Pero que además:

- Sea **Eficaz**
Lo hace en un tiempo admisible.
Especificaciones No Funcionales o de Calidad.
- Sea **Adaptable o Flexible**
¿Qué pasa si cambian las especificaciones?
- Sea **Reutilizable**
Sería ideal que ante problemas parecidos, no tenga que volver a escribir todo el código y pueda aprovechar la mayor parte de él.

1.1 Historia de la Programación

Evolución

Código Binario

Con “ceros” y “unos”. 0010101 --> SUMAR

```
CS:A0 BBB702    MOV  BX,02B7
CS:A3 C746FE5100 MOV  WORD PTR[BP-2],0051
CS:A8 EB07      JMP  B1
CS:AA 8BC3      MOV  AX,BX
CS:AC 0346FE    ADD  AX,[BP-2]
CS:AF 8BD8      MOV  BX,AX
CS:B1 83FB01    CMP  BX,1
```

Código Máquina. Ensamblador

Tengo Nemotécnicos y puedo llamar a rutinas (funciones)

Lenguajes Procedimentales

Puedo pasar argumentos a funciones, devolver valores.
Separo interfaz de implementación.

Programación Orientada a Objeto

Todo es un objeto.
Ayuda a abstraer, imaginarme las cosas del mundo real simplificando.
Puedo encapsular los datos para evitar operaciones no permitidas.

Patrones de Diseño

Pares de solución/problema

Librerías de Clases, Frameworks

Reutilización masiva

Programación Orientada a Aspectos, a Servicios, a Modelos,...

1.2 Reutilización

Informe de Standish Group de 2004:

Sólo el **29%** de los proyectos software satisfacen los requisitos de los clientes, se entregan a tiempo y se ajustan a los presupuestos.

El **53%** acaban incumpliendo algún requisito, se entregan fuera de plazo o cuestan más de lo presupuestado.

El **18%** se cancelan antes de su finalización o jamás llegan a utilizarse.

Reutilizar ayuda a **sacar más con menos esfuerzo**: Copio o reutilizo código, estructuras o ideas.

Menos Equivocaciones:

Reutilizo algo que ya funciona. Se ha corregido sus errores.

Escribo menos código nuevo.

Menos Pruebas

Pruebo sólo lo nuevo y su integración.

Más Creativo

No rehago siempre lo mismo. Codifico nuevas funcionalidades.

Mejoran las Probabilidades de Éxito

1.2 Tipos de Reutilización

Reutilización Oportunista

En general se da porque un programador copia código y lo pega en el suyo. Está bien pero **tiene problemas a la larga**: genero líneas diferentes para lo mismo y si algo está mal en el origen hay que buscar todos los sitios en donde se ha pegado para corregirlo.

Reutilización Sistemática - (POO)

Buscamos reutilizar y con un plan calculado. Queremos escribir sólo el código necesario, equivocarnos menos, poner y quitar funcionalidad de forma fácil. Queremos **aprovechar los conocimientos y experiencia** de otros que han estado antes “por aquí”.

1.3 Programación Orientada a Objeto

Conceptos

```
int i;           ClaseEnteros mi_entero;  
i=33;          mi_entero.asignar(33);  
cout << i;     mi_entero.imprime_lo_que_vales();
```

Todo es un Objeto:

Un objeto es como una variable mejorada.

Cogemos un componente conceptual de nuestro problema y lo representamos como un objeto en nuestro programa (perro, edificio, tuerca, ventana,...)

Los objetos se comunican mediante mensajes:

Haremos programas que serán grupos de objetos enviando mensajes a otros para decirles qué hacer. Petición de invocación a una función que pertenece a un objeto en particular.

Datos Propios:

Podemos crear un nuevo tipo de objeto haciendo un paquete con otros objetos. Oculto la complejidad. Coche (ruedas, motor,etc)

Cada objeto es de un tipo:

Un clase define el tipo de un objeto. Clase Enteros -> un_entero.

Un objeto es una instancia de una clase determinada.

1.3 Programación Orientada a Objeto

Conceptos II

Todos los Objetos de un tipo en particular pueden recibir los mismos mensajes:

Esto es muy potente. Nos abre un montón de posibilidades para REUTILIZAR CÓDIGO.

Clase Figura

Clase Círculo

Clase Cuadrado

Clase Triángulo

Un objeto del tipo o clase Círculo también es del tipo Figura. Está garantizado que un círculo recibirá los mensajes de Figura.

Si hacemos código que habla con objetos de tipo Figura, SIN TOCAR NADA, ese código también funciona con objetos de la clase Círculo.

```
mi_figura.dibujar_en_pantalla();
```

1.3 Programación Orientada a Objeto

Conceptos III

Clases:

Una clase describe un conjunto de objetos que tienen:

Las mismas características. > Datos

Los mismos comportamientos. > Funciones

Una clase es un tipo de datos. Un entero también tiene unas características y unas funciones específicas.

Crearemos nuevas clases y el lenguaje de programación las tratará como si fueran tipos de datos propios.

Una vez definida una clase podremos crear tantos objetos como queramos.

```
Circulo    circulo1, circulo2;  
Circulo    my_nuevo_circulo;
```

1.3 Programación Orientada a Objeto

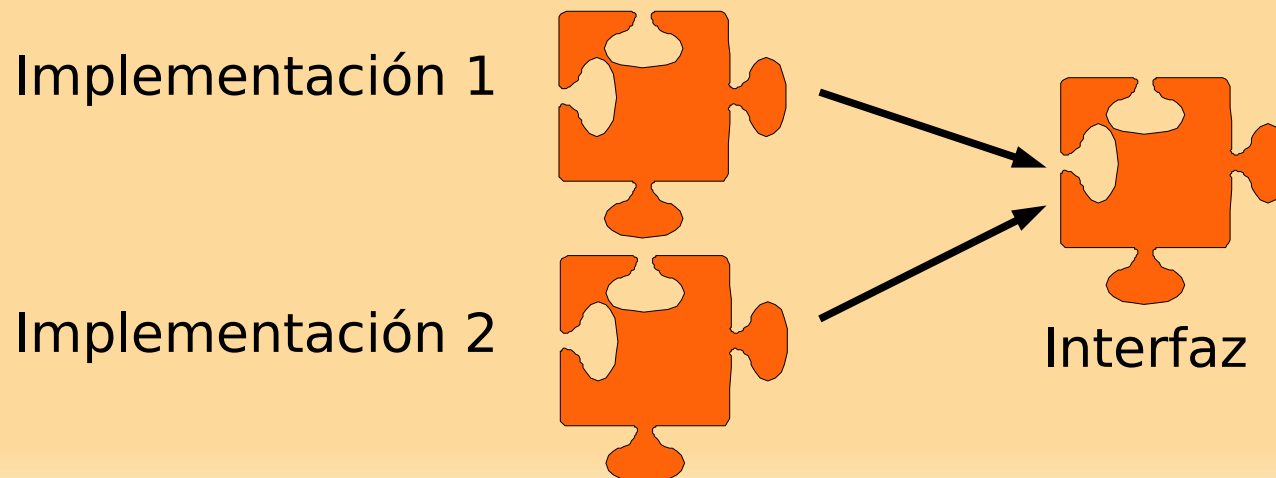
Ocultando la Implementación

Creadores de Clases:

Programadores que proporcionan un interfaz a otros programadores clientes. No dejamos que los clientes accedan a la implementación. Así podemos cambiarla sin preocuparnos. Mejoro la función “dibujar()”.

Programadores Clientes:

Utilizan clases creadas por otros para hacer programas u otros objetos sin preocuparse de la implementación. Si mejoras el método “dibujar()” yo no tengo que cambiar nada de mi código.



1.3 Programación Orientada a Objeto

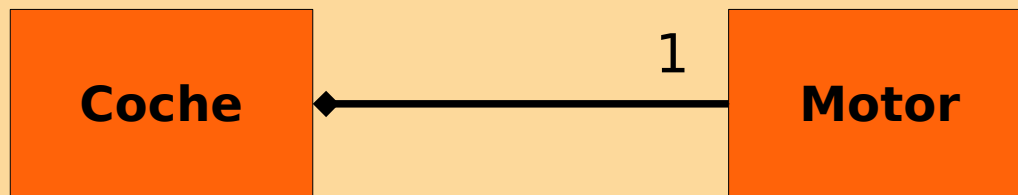
Reutilizando la implementación

Si hemos hecho bien nuestra clase se convierte en una unidad de código útil para los demás y para nosotros mismos. La utilizamos cuando nos venga bien. La REUTILIZAMOS.

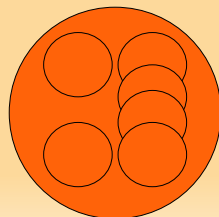
Podemos **crear un objeto** de esa clase directamente:

```
ClaseMotor    mi_motor;  
mi_motor.poner_a_punto();
```

Utilizar el objeto dentro de una clase



Abstracción
de la
complejidad



Composición o Agregación

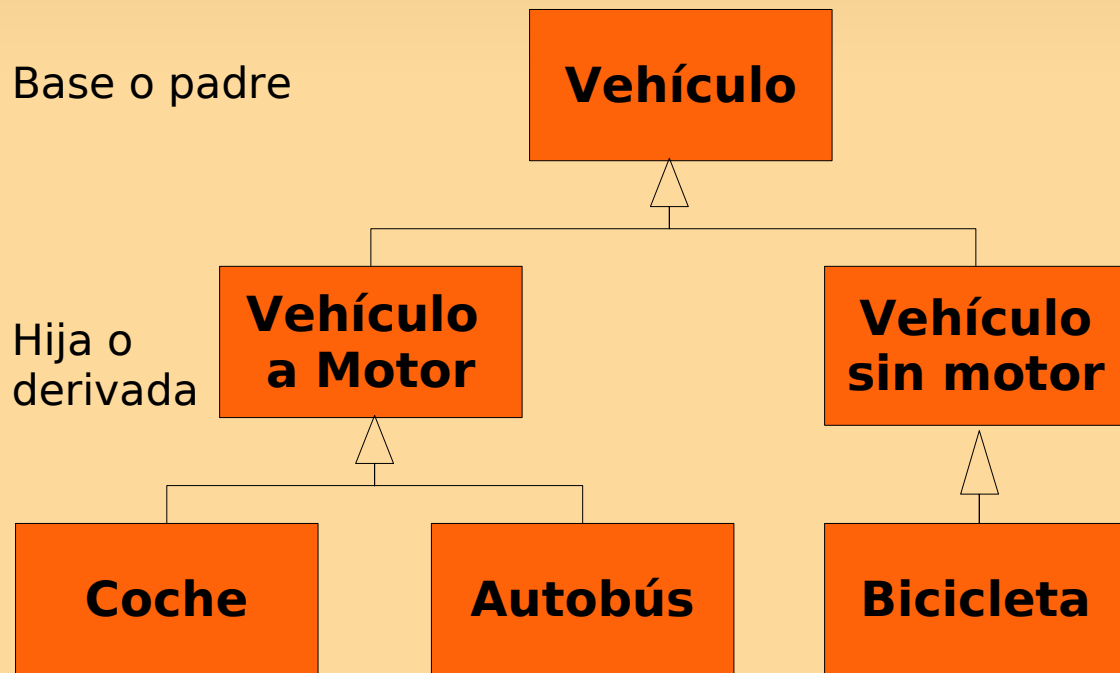
Se lee “X tiene n Y”
suelen ser privados
dinámica o estática
muy potente

1.3 Programación Orientada a Objeto

Reutilizando el Interfaz

Hay veces en que queremos crear un nuevo tipo, cuyo interfaz se parece a uno ya creado.

Es más sencillo clonar una clase y añadir y modificar cosas al clon.



Herencia

“es un” “es como un”

Pública. Permiso que lo vean mis herederos.
Privado. Sólo para mi.

1.3 Programación Orientada a Objeto

Características

Abstracción:

Nos permite representar una realidad compleja en términos de un modelo simplificado.

Encapsulación:

Nos permite ocultar cómo está construido un objeto. Otros componentes de nuestro sistema no necesitan conocer los datos y el funcionamiento interno del objeto. Datos y Funciones se empaquetan juntos. Los diferentes componentes, como no pueden acceder a sus partes internas, **se comunican mediante mensajes.**

Herencia:

Permite que un objeto incorpore todo o parte de su definición de otro objeto. Parte de la definición de Felino se corresponde con la definición de Mamífero.

Polimorfismo:

Propiedad que poseen algunas operaciones de tener un comportamiento diferente dependiendo del objeto (o tipo de dato) sobre el que se aplican

1.4 C++: El lenguaje de Programación

C++

Es un lenguaje de programación de

- Propósito general
- Orientado a Objeto.

Se basa en C. Un compilador de C++ compila también C.

Lo diseñó y lo implementó Bjarne Stroustrup y empezó a hacerlo en 1979. La primera versión se llamó “C con clases”.

La primera versión comercial salió al mercado en 1985.

Se diseñó para que cumpliera con dos requisitos:

- Eficiente
- Elegante

C++ es un lenguaje estandarizado. Se reúnen representantes de países y grandes empresas. El primer estándar sale en 1997 después de 8 años de esfuerzo. El estándar facilita enseñarlo en universidades de todo el mundo.

1.4 C++: El lenguaje de Programación

¿Alguien usa C++?

El número de programadores C++ en el mundo se estima que es de tres millones – 3.000.000 (IDS 2004)

Hay más de 400 libros sobre C++ en las librerías o en versión electrónica.

Todos los proveedores grandes tienen compiladores para C++.

Linux está en gran parte hecho en C++.

¿Es mejor C++ que Java o C#?

¿Quién es mejor programador/a?

- a) Juan que sabe 3 lenguajes
- b) Ana que sabe uno.
- c) No sabe no contesta.

Algunos datos

Aprendiendo C++ aprendes también C.

Pasar de C++ a Java o a C# es relativamente sencillo.

Hay mucho código para reutilizar.

Hay librerías que nos dan facilidades como el “Recolector de Basura”.

C++ sigue evolucionando : C++0x (Estándar ISO en 2008)

1.4 C++: El lenguaje de Programación

Portabilidad

Puedes compilar el mismo código C++ en diferentes plataformas.

Brevedad

El código escrito en C++ es muy corto comparado con el de otros lenguajes. Podemos utilizar caracteres especiales en lugar palabras evitando algo de esfuerzo al programador.

Modularidad

Podemos hacer una aplicación software usando varios ficheros de código fuente, que se compilan de manera separada y luego se enlazan para generar el ejecutable. No tenemos que recompilar todo; sólo el fichero que hemos tocado se compila y se enlaza el código objeto con el resto generado antes. Podemos enlazar código objeto de C++ con código que proviene de ensamblador.

Velocidad

El código resultante de una compilación de C++ es muy eficiente. C++ es un lenguaje que podemos tratar a muy alto nivel o a muy bajo nivel.

1.4 C++: El lenguaje de Programación

Categoría (Todas) | **Provincia** (Todas) | **Palabra clave** C++

505 ofertas encontradas | [Guarda esta búsqueda](#) | [Suscríbete a su RSS](#)

Más opciones

▼ **Provincia**

- [Madrid \(300\)](#)
- [Barcelona \(108\)](#)
- [Vizcaya \(20\)](#)
- [Valencia/València \(11\)](#)

[\[Mostrar más\]](#)

▼ **Categoría**

- [Tecnología y telecomunicaciones \(413\)](#)
- [Ingenieros y técnicos \(67\)](#)
- [Calidad, producción e I+D \(13\)](#)
- [Finanzas y banca \(5\)](#)

[\[Mostrar más\]](#)

▼ **Estudios**

- [Ingeniero Técnico \(212\)](#)
- [Ingeniero Superior](#)

Ofertas de empleo | **Formación relacionada**

Fecha <input checked="" type="checkbox"/>	Puesto vacante <input checked="" type="checkbox"/>	Población <input checked="" type="checkbox"/>	Empresa <input checked="" type="checkbox"/>
13/02	Ingenieros de Desarrollo C/C++	Madrid	
13/02	Analista/Programador Office Automation	Madrid	
13/02	Programadores C++.	Madrid	
13/02	ANALISTA / PROGRAMADOR - ORACLE	ALGETE	
13/02	Programador sin experiencia para Multinacional	Madrid	
13/02	Ingeniero de ventas sector sanitario con C++	Madrid	Multinacional inglesa
13/02	Ingeniero Superior Electrónica (Recién Titulado)	Barcelona	
13/02	Programadores y AP C, C++	Madrid	
13/02	Desarrolladores C++ Proyectos I+D (Tres Cantos)	Tres Cantos	
13/02	Analista Programador C++ UNIX	Alcobendas	
13/02	Ingeniero Desarrollo C++	Barcelona	
13/02	Técnico Testing Software.	Madrid	
13/02	Consultor Junior	Madrid	
13/02	A/P Y P En C/C++	Madrid	

1.5 Ejemplo de Clase

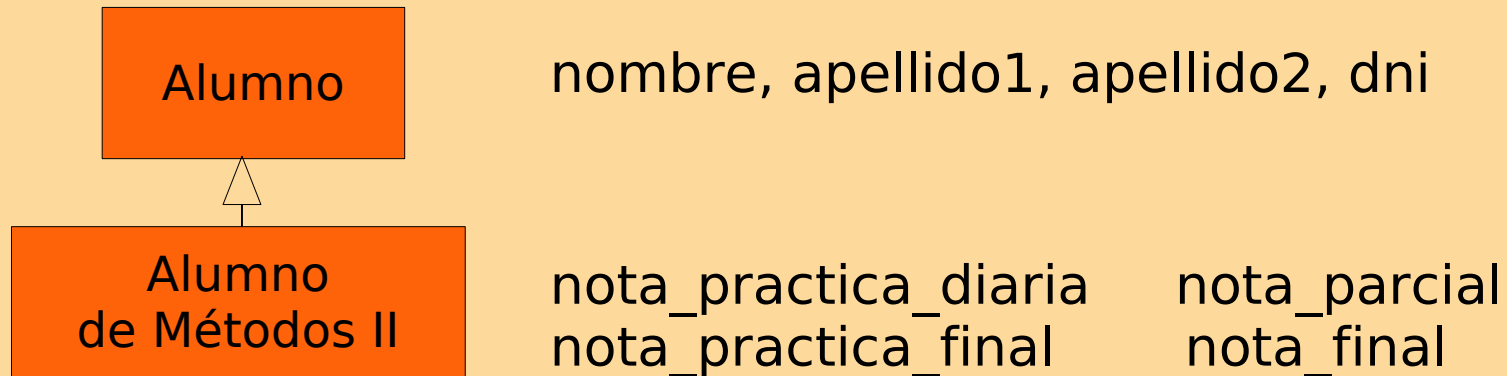
Definición

Queremos poder calcular la nota de los alumnos de la asignatura de Métodos II de programación sabiendo que la nota final se calcula:

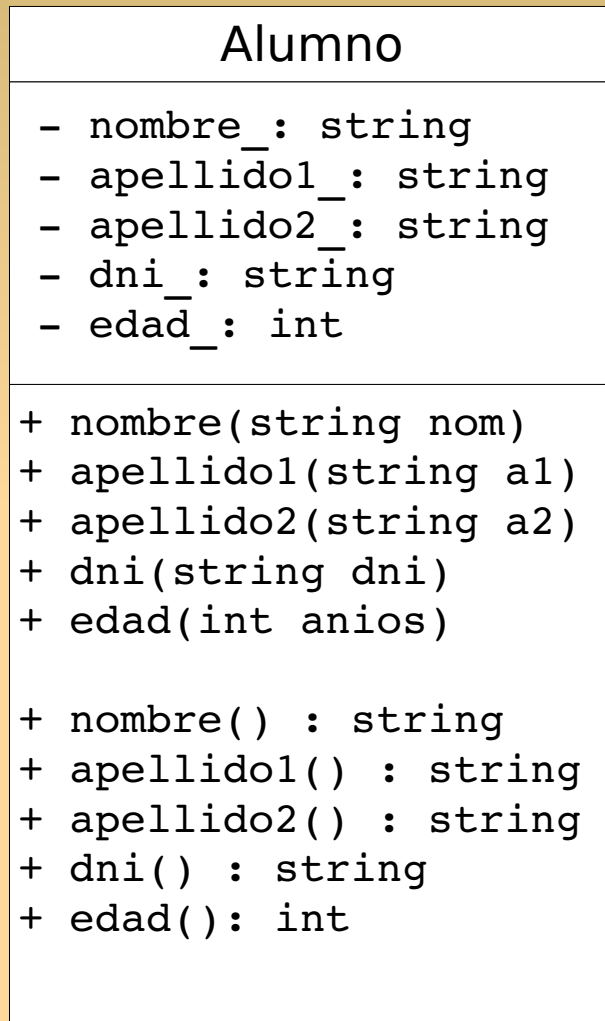
- 20 % Por las prácticas de laboratorio (20% diarias 80% trabajo final)
- 15 % Por exámen parcial.
- 65 % Por exámen final

y suponiendo que todos los alumnos aprobarán y no irán al exámen extraordinario.

Los alumnos deberán ser identificados por nombre, apellido1, apellido2 y DNI.



1.5 Ejemplo de Clase

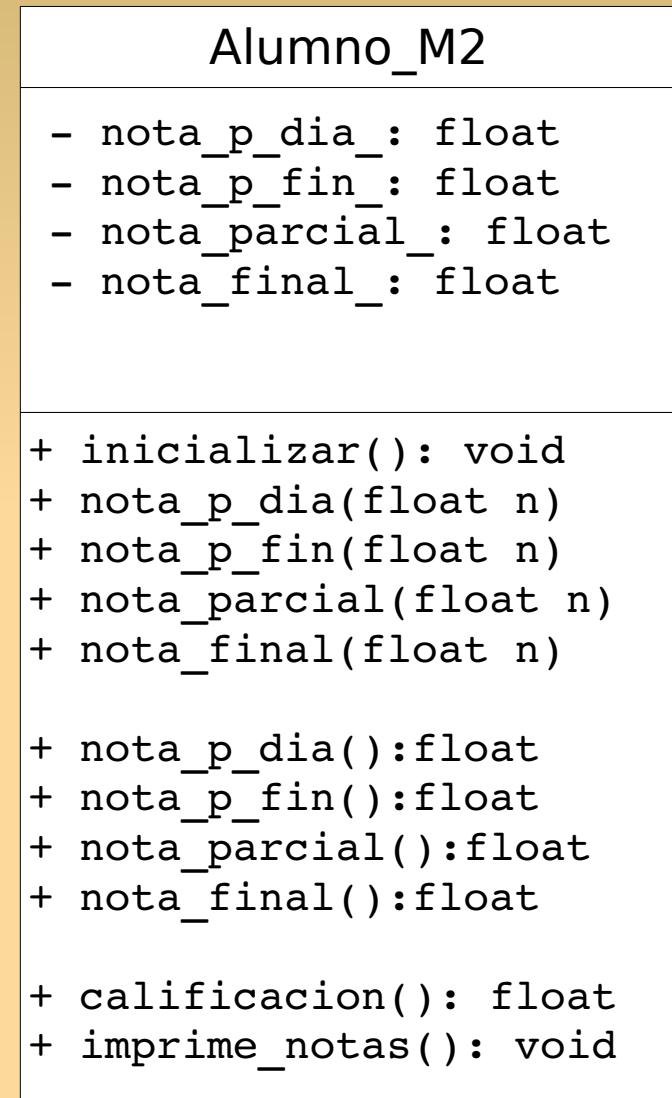


- privado
+ público

Nombre
de la Clase

Datos
Atributos

Métodos
Funciones



1.5 Ejemplo de Clase

Definición de Interfaz

```
// Clase Alumno de Métodos II
class AlumnoM2: public Alumno
{
private:
    float  nota_p_dia_;
    float  nota_p_fin_;
    float  nota_parcial_;
    float  nota_final_;

public:
    void inicializar (void);
    void nota_p_dia (float nota);
    void nota_p_fin (float nota);
    void nota_parcial (float nota);
    void nota_final (float nota);

    float nota_p_dia (void);
    float nota_p_fin (void);
    float nota_parcial (void);
    float nota_final (void);

    float calificacion(void);
    void imprime_notas();
}
```

AlumnoM2

```
- nota_p_dia_: float
- nota_p_fin_: float
- nota_parcial_: float
- nota_final_: float
```

```
+ inicializar(): void
+ nota_p_dia(float n)
+ nota_p_fin(float n)
+ nota_parcial(float n)
+ nota_final(float n)
```

```
+ nota_p_dia():float
+ nota_p_fin():float
+ nota_parcial():float
+ nota_final():float
```

```
+ calificacion(): float
+ imprime_notas(): void
```

1.5 Ejemplo de Clase

Implementación

```
void
AlumnoM2::inicializar (void)
{
    nota_p_dia_      = -1;
    nota_p_fin_      = -1;
    nota_parcial_    = -1;
    nota_final_      = -1;
}
//-----
void
AlumnoM2::nota_p_dia (float nota);
{
    nota_p_dia_      = nota;
}
//-----
void
AlumnoM2::nota_p_fin (float nota);
{
    nota_p_fin_      = nota;
}
//-----
void
AlumnoM2::nota_parcial (float nota);
{
    nota_parcial_    = nota;
}
```

```
//-----
void
AlumnoM2::nota_final (float nota);
{
    nota_final_      = nota;
}
//-----
float
AlumnoM2::nota_final (void);
{
    return nota_final_;
}
...
//-----
float
AlumnoM2::calificacion (void);
{
    flota cal=0.0;
    cal = 0.2* (0.2 * nota_p_dia_ +
                0.8 * nota_p_fin_) +
          0.15*nota_parcial_ +
          0.65*nota_final_;
    return cal;
}
```

1.5 Ejemplo de Clase

Ejercicios

1.- Método para imprimir

Codificar el método para imprimir por pantalla:

```
Nombre      :  
Apellido1   :  
Apellido2   :  
DNI         :  
Edad        :
```

```
Calificación :
```

2.- Método que devuelve la calificación en un argumento.

Codificar el método que podamos llamar así.

```
float cal;  
calificacion(cal);  
cout << "Calificacion: " << cal << "\n";
```

(*) Suponemos que un "Creador de Clases" ha codificado la clase Alumno de la que deriva la nuestra.

1.6 Para Recordar

1.- Los proyectos Software son complejos y difíciles de desarrollar

Sólo el **29% de los proyectos software satisfacen los requisitos de los clientes**, se entregan a tiempo y se ajustan a los presupuestos.

2.- La POO nos ayuda a reutilizar y equivocarnos menos.

Sus características son: Abstracción, encapsulación, herencia y polimorfismo.

3.- C++ es un lenguaje Orientado a Objeto.

Se utiliza extensamente. Podemos aprender otros lenguajes. Eficiente.

4.- Los diagramas UML son una buena ayuda para diseñar.

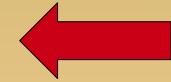
5.- Aprendemos con la Práctica.

Programa

Introducción a la POO

Historia de la Programación
Conceptos de POO

C++
Mi primera Clase



Repaso de Conceptos

Estándares de Programación
Punteros y Memoria

Ficheros y E/S
Control y Operadores

Clases y Objetos en C++

Uso y aplicación
Constructores

Funciones Amigas
Constantes e "inline"

Sobrecarga

De Operadores

De Funciones

Herencia.

Tipos de Visibilidad

Herencia Múltiple

Polimorfismo

Funciones Virtuales

Polimorfismo y Sobrecarga.

Plantillas

Contenedores

Iteradores