

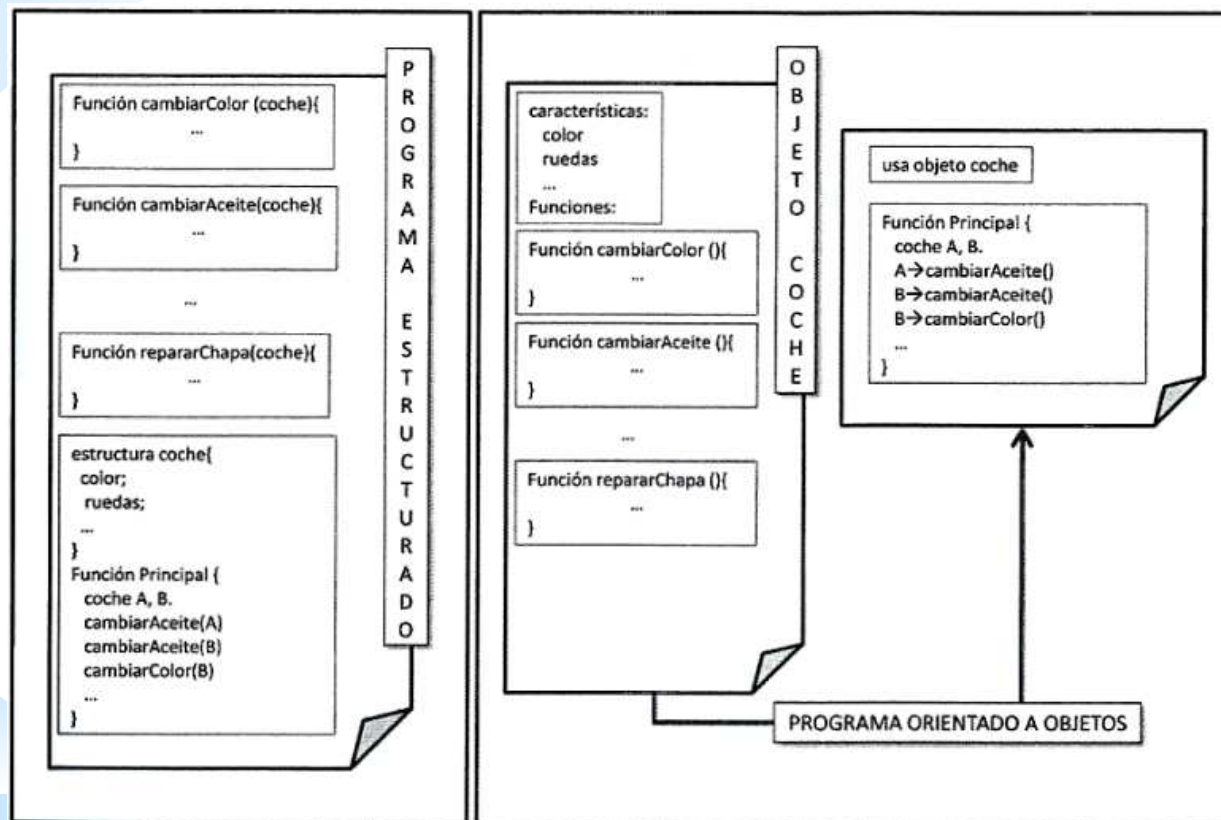


PROGRAMACIÓN

4ª TUTORÍA COLECTIVA

PROGRAMACIÓN ORIENTADA A OBJETOS - POO

- * Refleja la realidad, de forma que los elementos de un programa se ajustan a elementos de la vida cotidiana.
- * Permite la creación de software cada vez más complejo a partir de unidades o bloques de código reutilizable.



Taller coches:

Modelo Estructurado
Y
Modelo Orientado a
Objetos

PROGRAMACIÓN ORIENTADA A OBJETOS - POO

* Propiedades de la Orientación a objetos:

- ABSTRACCIÓN
- ENCAPSULACIÓN
- MODULARIDAD
- JERARQUÍA
- POLIMORFISMO

PROGRAMACIÓN ORIENTADA A OBJETOS - POO

ABSTRACCIÓN

Permitirá recolectar y representar las características esenciales de un objeto, dejando atrás aquellas que no tienen tanta importancia.

Generalización de las características y comportamientos de un grupo de objetos.

Si pensamos en una clase **Vehículo** que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como **Coche** y **Camión**. Entonces se dice que **Vehículo** es una abstracción de **Coche** y de **Camión**.

PROGRAMACIÓN ORIENTADA A OBJETOS - POO

ENCAPSULACIÓN o ENCAPSULAMIENTO

Es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación a fin de evitar ser manipulado de forma inadecuada.

Sabemos qué operaciones realiza un objeto, pero no necesitamos conocer cómo las realiza.

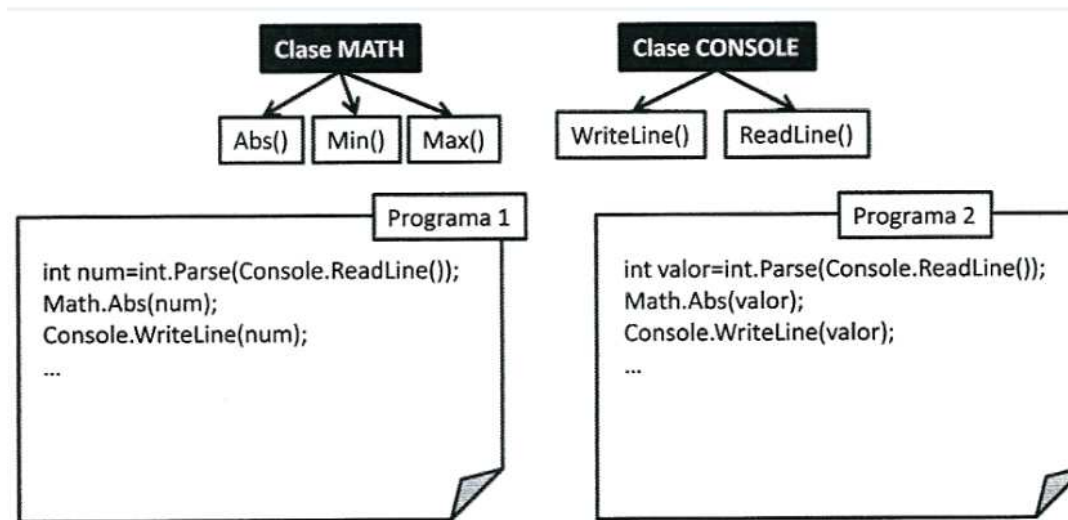
Pensemos en un programa con dos objetos, un objeto **Persona** y otro **Coche**. **Persona** se comunica con **Coche** para llegar a su destino, utilizando para ello las acciones que **Coche** tenga definidas como por ejemplo conducir. Es decir, **Persona** utiliza **Coche** pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.

PROGRAMACIÓN ORIENTADA A OBJETOS - POO

MODULARIDAD

Se refiere a la forma en la que los elementos se encuentran organizados en módulos, facilitando así la encapsulación y abstracción de la información.

Permite dividir la aplicación en partes (módulos) independientes que podremos **reutilizar**.

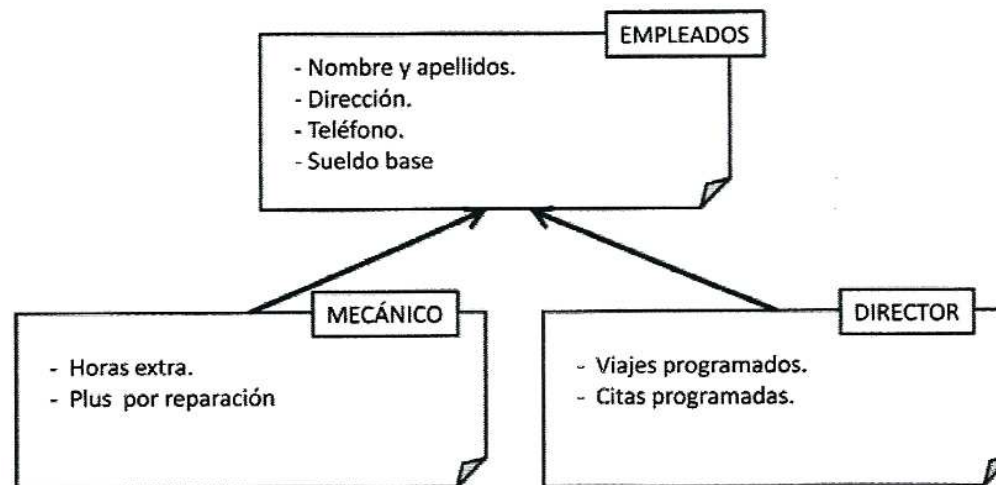


PROGRAMACIÓN ORIENTADA A OBJETOS - POO

JERARQUÍA

Proceso de estructuración por el que se establece una organización en niveles.

Podemos hablar de **herencia**: un objeto puede adquirir las propiedades y comportamientos de otro.



PROGRAMACIÓN ORIENTADA A OBJETOS - POO

POLIMORFISMO

Podemos tener métodos con igual nombre pero con implementaciones diversas tal que dependiendo del objeto que se use, ese método realizará una u otra operación.

Se usa bastante junto con la herencia.



CLASES Y OBJETOS

¿Qué es lo que tienen en común?



Se podría encontrar una forma de definir “algo” que **encapsule** las características y comportamiento comunes

Modelo
Marca
Color
Velocidad

Acelerar
Desacelerar
Apagar
Arrancar

CLASES Y OBJETOS

¿Qué es una **clase**?

Es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de ciertas características comunes.

Es una plantilla genérica para un conjunto de objetos de similares características.

Contiene:

- Conjunto de **atributos** comunes
- **Comportamiento** por medio de métodos

CLASES Y OBJETOS

Representación de la clase Auto:

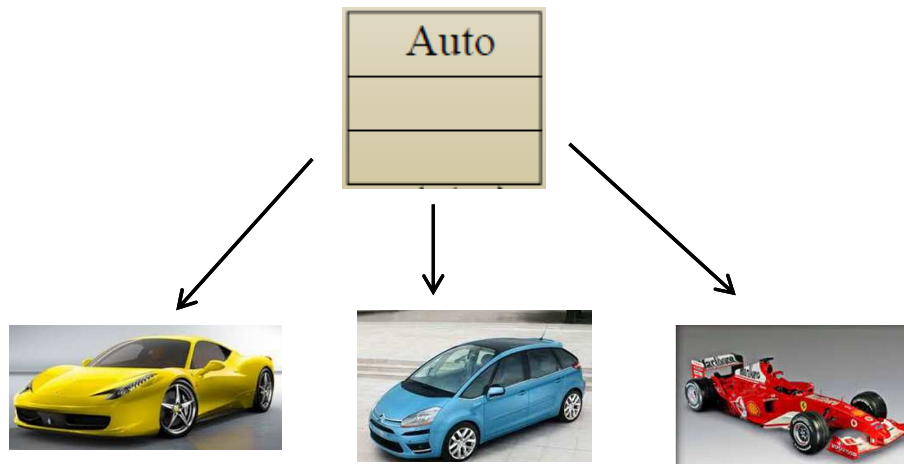


CLASES Y OBJETOS

Cada uno de los diferentes automóviles tiene características comunes pero con valores diferentes. Por ejemplo, los tres tienen color, pero cada uno un color diferente.



Ejemplares de una clase (instancias) = **OBJETOS**



CLASES Y OBJETOS

MENSAJES

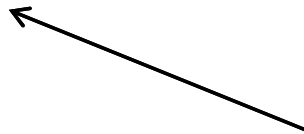
Son invocaciones a la funcionalidad de un objeto. “Se lanzan mensajes” para pedir a los objetos que hagan cosas.



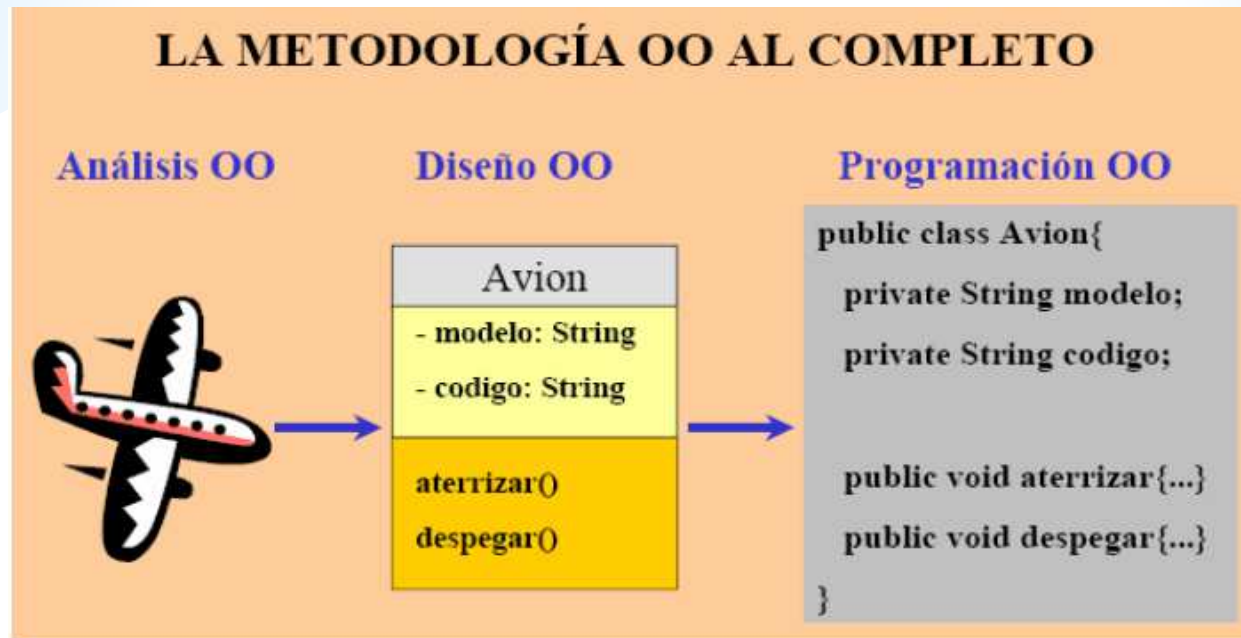
arrancar()



acelerar()

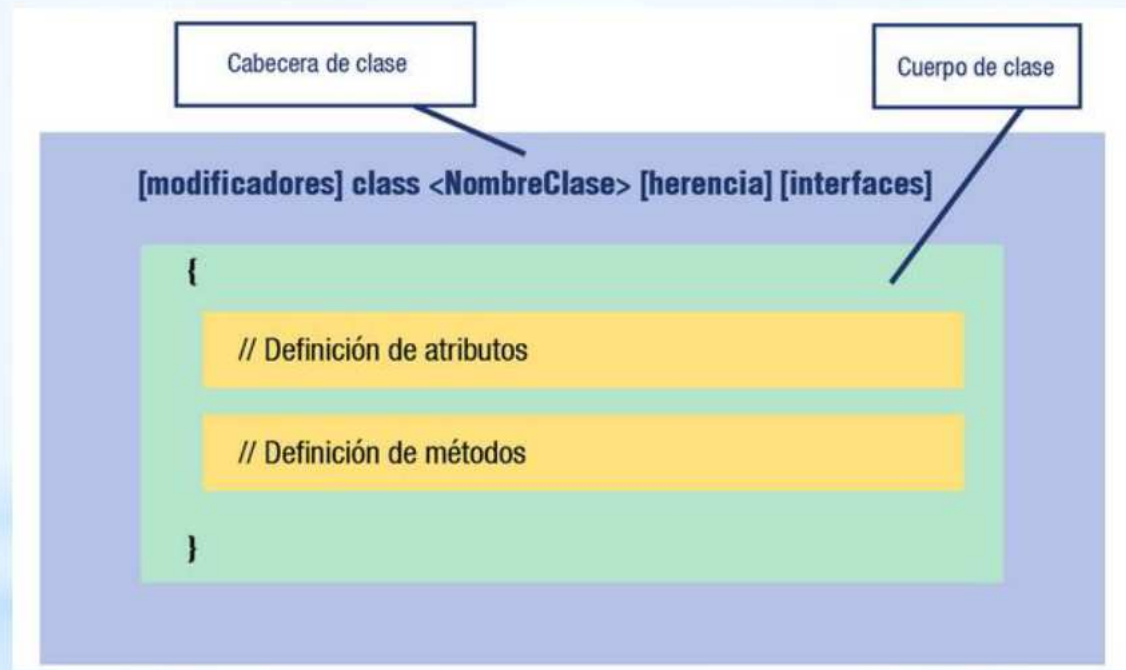


ETAPAS DE LA POO



CLASES

La estructura de una clase suele seguir el siguiente esquema:



CLASES

- * En caso de ser una clase distinta de la clase principal ejecutable, el método main() no aparece.
- * Un programa se compone de un conjunto de clases que crean objetos que interactúan entre sí.
- * Una vez creada una clase, el programador podrá crear objetos o instancias de esa clase, simplemente para ello debe definir una variable/objeto de ese tipo de clase y crear después el objeto con la instrucción new.

Alumno.java

```
public class Alumno {  
    ....  
}
```

UsaAlumno.java

```
Alumno alum1;    // declaro la variable de ese tipo de clase  
  
alum1 = new Alumno()    //creo el objeto con el operador new
```


CLASES - Modificadores de acceso

```
[modificador] class  NombreClase {  
    <conjunto atributos>  
    <conjunto de métodos>  
}
```

El **modificador** indica la forma de acceder a la clase:

- **public**: cualquier otra clase puede acceder a ella.
- **final**: la clase no puede tener hijos, no va a poder crear herencia.
- **abstract**: no se pueden instanciar objetos a partir de ella.
- Si no se indica ningún modificador, se entiende que la clase es “**amistosa**” y solo podrán acceder a ella las que estén en el mismo paquete.

ATRIBUTOS

- * Son el conjunto de datos que los objetos de una clase almacenan cuando son creados. Variables de objeto.
- * Los atributos son variables de tipo primitivo o pertenecientes a otras clases.
- * Se definen dentro del cuerpo de la clase, y normalmente antes de los métodos.

```
[modificador] tipo nom_atributo [= valorInicial];
```

```
private static int cantidad = 0;  
private String nombre;
```

ATRIBUTOS - Modificadores de acceso y de contenido

Si se indica un modificador, podrá ser:

- **public:** cualquier otra clase puede acceder al atributo.
- **protected:** Sólo la propia clase y sus hijas pueden acceder al atributo.
- **private:** Sólo la propia clase puede acceder al atributo.
- Si no se indica modificador, solo las clases del paquete pueden acceder al atributo.

Además, junto con los anteriores se pueden indicar:

- **static:** el valor del atributo será el mismo para todas las instancias (objeto) de la clase.
Si un objeto cambia todos los objetos instanciados por la misma clase cambiarán también.
- **final:** se utiliza para indicar que la propiedad es una constante. Una vez inicializada no se podrá modificar.

ATRIBUTOS - Ejemplo

Alumno
-nombre: String -apellidos: String -añoNacimiento: int -numMatricula: int -grupo: String -horario: String

```
class Alumno {  
    private String  
    private String  
    private int  
    private int  
    private String  
    private String  
}
```

```
nombre;  
apellidos;  
añoNacimiento;  
numMatricula;  
grupo;  
horario = "MAÑANA";
```

→ atributos

// atributo con valor inicial

MÉTODOS

- * Permiten definir el comportamiento de un objeto en sus interacciones con otros objetos.
- * Un método es:
 - * Un bloque de código que tiene un nombre.
 - * Recibe unos parámetros o argumentos opcionalmente.
 - * Contiene instrucciones o sentencias para realizar algo.
 - * Devuelve un valor de algún tipo conocido.
- * Para definir un método:

```
[modificador] tipoDevuelto nomMétodo([ tipo param1, ...]{  
    <bloque de código> return <expresión>  
}
```

MÉTODOS - Modificadores de acceso

Si se indica un modificador, podrá ser:

- **public**: cualquier otra clase puede acceder al método.
- **protected**: Sólo la propia clase y sus hijas pueden acceder al método.
- **private**: Sólo la propia clase puede acceder al método.
- Si no se indica modificador, será un método amistoso, al que podrán acceder todas las clases del mismo paquete.

Además, junto con los anteriores se pueden indicar:

- **static**: Para poder utilizar el método, no es necesario crear una instancia del objeto. Los métodos static sólo pueden acceder a propiedades static.
- **final**: Ninguna clase que herede este método, podrá sobrescribirlo.

MÉTODOS - Ejemplo

Alumno
-nombre: String -apellidos: String -añoNacimiento: int -numMatricula: int -grupo: String -horario: String
+devuelveGrupo(): String +setGrupo(String nuevoGrupo) +setGrupo(String nuevoGrupo, String turno)

```
class Alumno {  
    private String nombre;  
    private String apellidos;  
    private int añoNacimiento;  
    private int numMatricula;  
    private String grupo;  
    private String horario = "MAÑANA"; // atributo con valor inicial  
  
    public String devuelveGrupo ( ) { .....}  
    public void setGrupo (String nuevoGrupo) {.....}  
    public void setGrupo (String nuevoGrupo, String turno) {.....}  
}
```

} → atributos

} → métodos


THIS

La auto referencia **this** se utiliza para referirse a los atributos desde un método del propio objeto.


Se utiliza cuando existe ambigüedad entre los nombres de los parámetros de un método y los atributos del objeto

```
public String getGrupo ( ) {  
    return grupo;  
}
```

```
public void setGrupo (String grupo) {  
    this.grupo = grupo;  
}
```



atributo



parámetro

MÉTODOS GETTER Y SETTER

Getter y Setter son métodos que se suelen definir normalmente para obtener o asignar valores a los atributos **privados** de la clase.

Normalmente existe un método getter y un setter para cada atributo.

```
class Alumno{
    private String nombre;
    private String grupo;
    public String getNombre{
        return nombre;
    }
    public String getNombre{
        return nombre;
    }
    public void setGrupo(String paramGrupo){
        grupo=paramGrupo;
    }
    public void setNombre(String paramNom){
        nombre=paramNom;
    }
}
```

OBJETOS

- * Los objetos en Java no son más que variables de tipo complejo.
- * El tipo de dato de un objeto es el de la clase de la que se ha instanciado.
- * La declaración de un objeto es idéntica a la de una variable, donde el *tipo_identificador* será el nombre de la clase que posteriormente creará el objeto.

tipo_identificador nombre_objeto

Alumno alum1; //creo una variable objeto de tipo Alumno, tendrá todos los atributos
 // y métodos de la clase Alumno

Para inicializar un objeto se utiliza uno de los métodos constructores definidos en la clase, o en su caso el constructor por defecto, precedido del operador new.

alum1 = new Alumno();

Manejo de OBJETOS

- * El trabajo con un objeto consiste en acceder a:
 - * Sus atributos.
 - * Sus métodos.
- * El acceso a estos elementos se realiza mediante.
 - * objeto.atributo
 - * objeto.metodo(param1, param2)

```
Alumno alum1 = new Alumno();
```

```
alum1.setGrupo("DAW1");
```

```
String g = alum1.getGrupo();
```

MÉTODOS - SOBRECARGA

Consiste en tener varios métodos con el mismo nombre pero con distinto número de parámetros. Según los parámetros que se pasen se invocará a un método o a otro.

```
public void setGrupo (String grupo){  
    this.grupo = grupo;  
}
```

```
public void setGrupo(String grupo, String horario){  
    this.grupo = grupo;  
    horario=turno;  
}
```

MÉTODOS CONSTRUCTORES

- * Son métodos utilizados para instanciar/crear los objetos.
- * Son métodos que se emplean para:
 - inicializar los valores de los atributos de los objetos.
 - realizar las operaciones que sean necesarias para la generación de este objeto.
- * Se definen con el mismo nombre que la clase, es decir, con la primera letra en mayúscula:

NombreClase(); //Constructor por defecto de la clase si no se define ninguno.

NombreClase(param1);

NombreClase(param1,....., paramN);

MÉTODOS CONSTRUCTORES

Lo normal es escribir uno o más constructores (sobrecarga) permitiendo dar flexibilidad a la hora de inicializar un objeto.

```
Class Alumno{
```

```
...
```

```
Alumno(String nombre, String apellidos, int año)
```

```
{  
    this.nombre = nombre;  
    this.apellidos = apellidos;  
    this.añoNacimiento = año;  
}
```

```
Alumno(String nombre, String apellidos, int año, String grupo, String turno) {
```

```
    this.nombre = nombre;  
    this.apellidos = apellidos;  
    this.añoNacimiento = año;  
    this.grupo = grupo;  
    horario = turno;  
}
```


MÉTODOS CONSTRUCTORES

Podríamos utilizar cualquiera de los dos constructores, definidos en la clase:

```
Alumno alum2 = new Alumno("Aitor", "Garcia", 1989);  
Alumno alum3 = new Alumno("María", "Comillas", 1982, "DAI2", "tarde");
```

No podríamos usar el constructor por defecto puesto que hemos definido dos nuevos constructores.

No podríamos utilizar:

```
Alumno alum1 = new Alumno();
```

ERROR.- Este constructor no está definido.

RESUMEN MODIFICADORES

modificador	clases	Variables	Metodos	interfaces
public	Visible en todas partes	Visible donde su clase lo sea	Visible donde su clase lo sea	Visible donde su clase lo sea
protected	No se aplica a clases	Visible en el mismo paquete y en todas las subclases	Visible en el mismo paquete y en todas las subclases	Visible en el mismo paquete y en todas las subclases
ninguno	Visible solo en su paquete	Visible en el paquete de su clase	Visible en el paquete de su clase	Visible en el paquete de su clase
private	No se aplica	Visible solo en su clase	Visible solo en su clase	Visible solo en su clase
abstract	La clase tiene métodos no implementados	No se aplica	No hay implementación solo se da el encabezado	Todas las interfaces son abstractas aunque no se declaren así
final	No pueden crearse subclases	Su valor no se puede cambiar	No se puede sobrescribir	No se aplica
static	No se aplica	Es una variable de la clase, solo hay una para todos los objetos de la clase y por lo tanto en todos los objetos tiene el mismo valor	Es un método de la clase, solo puede usar las variables de clase. Se puede invocar con el nombre de la clase	No se aplica