

Programación Orientada a Objetos

Tema 2: Fundamentos de la programación orientada a objetos

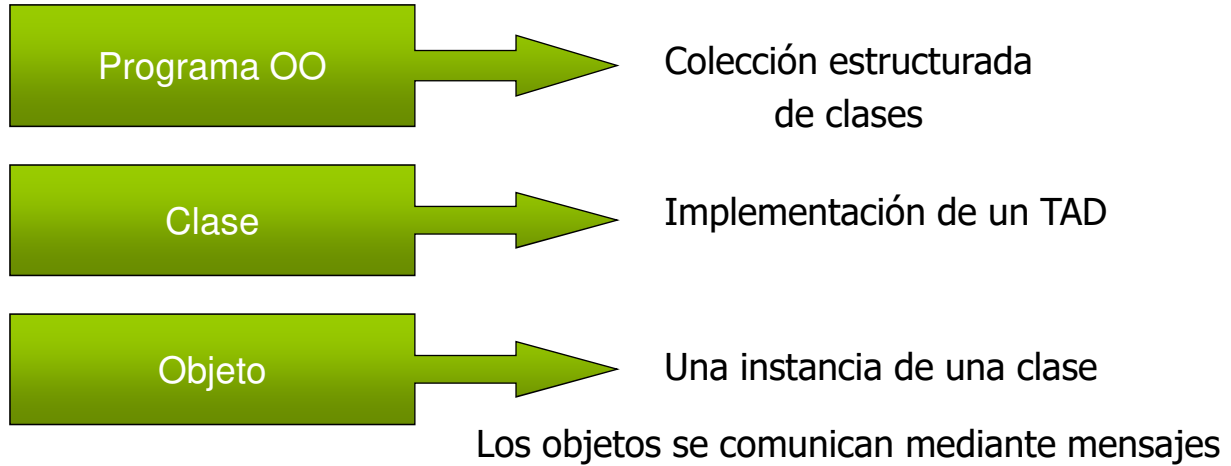
Tema 2-3: Conceptos básicos de POO 1

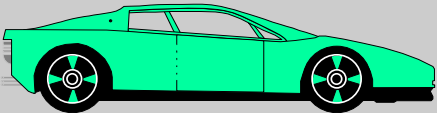
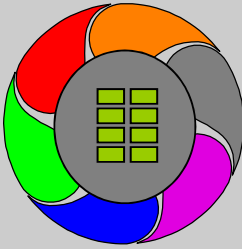
- Tema 2-3: Conceptos básicos de POO 1
- 1. Introducción
- 2. Objetos
- 3. Clases
- 4. Instancias
- 5. Diseño estructurado vs. OO



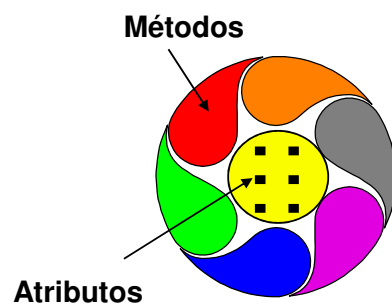
- **¿Qué es la POO?**

“Un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada una de las cuales representan una instancia de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de herencia”. Booch.



MUNDO REAL	SIMULACIÓN INFORMÁTICA
<p>Un elemento de un sistema físico se caracteriza por :</p> <p>Una IDENTIDAD, unas CARACTERÍSTICAS, un ESTADO, unos COMPORTAMIENTOS</p> 	<p>Un objeto se caracteriza por :</p> <p>Una IDENTIDAD, unos DATOS, unos PROCEDIMIENTOS</p> 

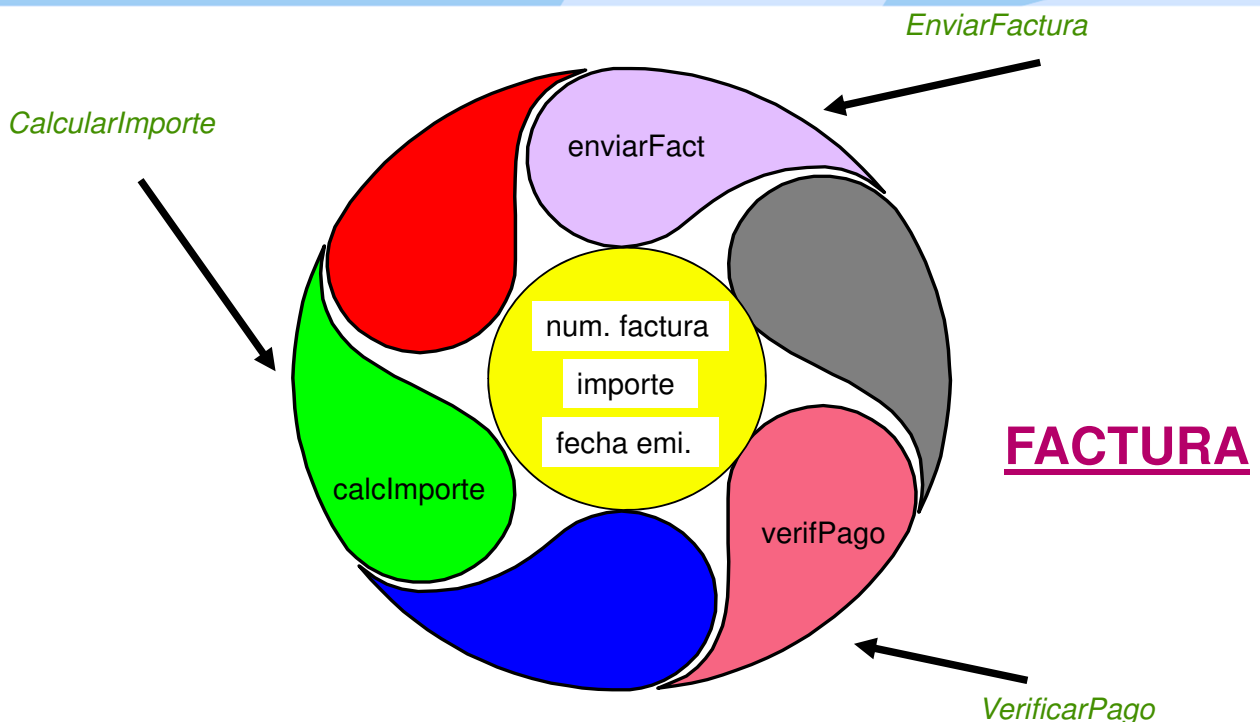
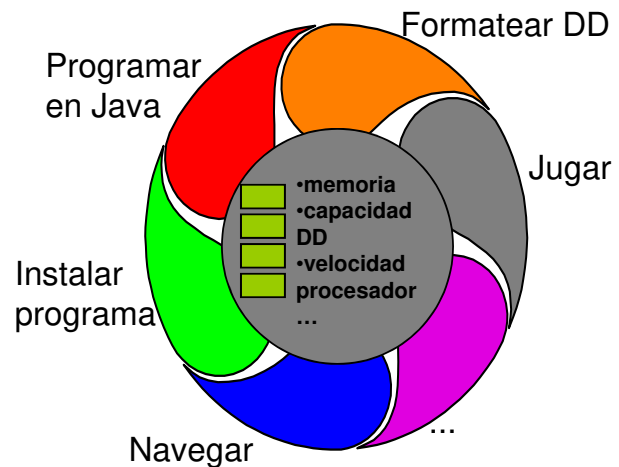
- Un “paquete” de software independiente formado por un conjunto de datos junto con los procedimientos que operan sobre estos datos.
- Un objeto puede ser real o abstracto.

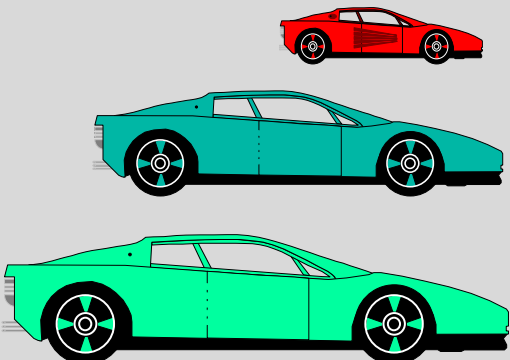
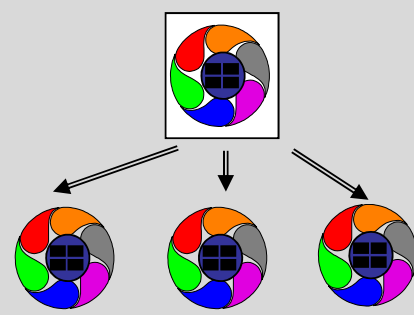


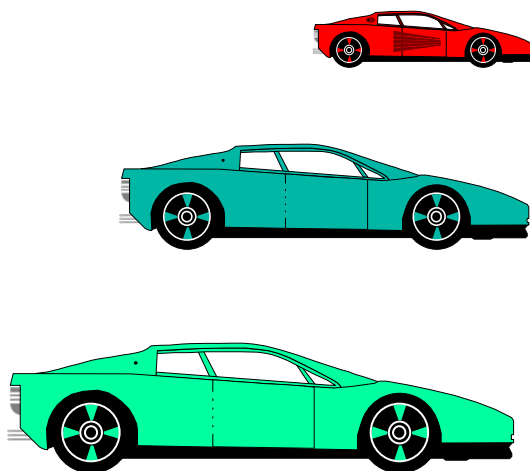
Estructura de un objeto

- Objeto = Módulo de software.
- Cada objeto constituye un universo cerrado bien definido.
- Todo lo que un objeto “sabe” se expresa en sus atributos.
- Todo lo que un objeto “puede hacer” se expresa por sus operaciones (métodos).

Ejemplo: Ordenador



MUNDO REAL	SIMULACIÓN INFORMÁTICA
<p>Los elementos de un sistema físico pueden ser agrupados en CATEGORIAS :</p> 	<p>Los objetos que muestran la misma estructura y comportamiento se agrupan en CLASES</p> 



Cualquier tipo de
coche puede
agruparse dentro
de la Clase
AUTOMOVIL



- Automóvil

Atributos	Operaciones
Matrícula	poner en marcha
marca	parar
modelo	acelerar
estado	frenar
potencia	pintar
color	encenderLuces
luces	apagarLuces
velocidad	
peso	...

Esta lista de atributos y de operaciones asegura el hecho de que el conjunto de instancias de la clase Automóvil dispondrá de estos atributos y de los valores asociados, así como de la facultad de ejecutar las operaciones citadas.



- Un Automóvil

Atributos
Matrícula: 5671BCD
marca: Ford
modelo: Focus
estado: en marcha
potencia: 130 c.
color: rojo
luces: encendidas
Velocidad: 60 km/h
peso: 1500 kg

- Se especifican sus atributos.
- Este coche tiene ahora una “existencia informática”.
- Está en condiciones de participar en las secuencias de interacciones.



Definición

“Implementación total o parcial de un tipo abstracto de dato (TAD)”

- Una clase es un tipo definido que determina la estructura de datos y las operaciones asociadas a ese tipo.
- Una clase se puede ver como una plantilla que describe objetos que van a tener la misma estructura y el mismo comportamiento.
- Un programa OO es una colección estructurada de clases.



- **Doble naturaleza de las clases:**

Una clase es un **módulo** y un **tipo de dato**:

- **Módulo** (concepto sintáctico)
 - Mecanismo para organizar el software (sintáctico)
 - Encapsula componentes software
- **Tipo** (concepto semántico)
 - Mecanismo de definición de nuevos tipos de datos: describe una estructura de datos (objetos) para representar valores de un dominio y las operaciones aplicables.

“Los servicios proporcionados por una clase, vista como un módulo, son precisamente las operaciones disponibles sobre las instancias de la clase, vista como un tipo”.



Componentes de una clase:

- **Atributos**

- Determinan una estructura de almacenamiento para cada objeto de la clase.

- **Operaciones (Métodos)**

- Operaciones aplicables a los objetos.
- Único modo de acceder a los atributos.

Ejemplo: Al modelar un banco, encontramos objetos “*cuenta*”.
Todos los objetos “*cuenta*” tienen propiedades comunes:

- atributos: número, saldo, titular, ...
- operaciones: ingresar, retirar ...

Definimos una clase CUENTA.



- La clase es la unidad fundamental de programación en Java.
- La clase define el interfaz y la implementación de los objetos de esa clase.
- Todo objeto es instancia de alguna clase.
- La declaración de una clase tiene la forma básica:

```
[public] [final] [abstract] class NombreClase {  
    ...  
}
```

Cuerpo de la clase
Aquí declaramos los
miembros de la clase:
atributos y métodos

public es visible desde otros paquetes
final esta clase no puede tener subclases
abstract clase abstracta



- Una clase en Java puede contener atributos (variables) y métodos. Los atributos pueden ser tipos primitivos como int, char, etc. u otros objetos. Los métodos son funciones. Tanto los métodos como los atributos se definen dentro del bloque de la clase. Java no soporta métodos o variables globales. Ejemplo:

```
public class MiClase {
    int i;                //atributo i
    public MiClase() {    //constructor
        i = 10;
    }
    public void suma_a_i( int j ) {    //método
        i = i + j;
    }
}
```

i = 21

mc1

i = 16

mc2

```
...
MiClase mc1 = new MiClase();
mc1.i++; // incrementa la instancia de i de mc1
mc1.suma_a_i( 10 );
MiClase mc2 = new MiClase();
mc2.i++; // incrementa la instancia de i de mc2
mc2.suma_a_i( 5 );
```



• Atributos:

La sintaxis básica para declarar atributos es:

[final] [static] acceso tipo lista-de-identificadores

acceso: **public** / **private** / **protected** / -

tipo: es cualquier tipo primitivo o clase o array de ellos

final: significa que el valor de ese campo no puede ser modificado (constante)

static: significa que todas las instancias comparten la misma variable (atributo de clase)

Ejemplos:

```
private double minimo, maximo;
final static int maxInstancias = 10;
static float[] valores;
```



- El ocultamiento de información tiene varios grados en Java.
- Todos los atributos y métodos de una clase son accesibles desde el código de la propia clase. Para controlar el acceso desde otras clases, se utilizan **modificadores de acceso**:

public	Accesible desde cualquier lugar en que sea accesible la clase
protected	Accesible por las subclases y clases del mismo paquete
private	Sólo accesible por la propia clase
- (friendly)	Accesible por las clases del mismo paquete



• Inicialización de los atributos:

Java asigna un valor por defecto a los atributos que no se inicializan que depende del tipo de éstos:

Tipo del campo	Valor Inicial
boolean	0
char	'\u0000'
byte, short, int, long	0
float, double	+0.0f ó +0.0d
referencia a objeto	null

Sin embargo, no asigna ningún valor por defecto inicial a las variables locales de métodos o constructores.

Debemos asignar algún valor a una variable local antes de usarla.



• Inicialización de los atributos:

Los atributos pueden:

- **No inicializarse**, con lo que tendrán los valores por defecto vistos anteriormente, según el tipo del campo.
- **Inicializarse en la declaración**, asignándoles una expresión. No se pueden invocar métodos que pueden lanzar excepciones.

```
private String miNombre = "Juan " + "Carlos";  
private int edad = obtenerMiEdad();
```



• Métodos:

La sintaxis básica para declarar métodos es:

[final] [static] [abstract] acceso tipo identificador(args) {...}

final: Significa que el método no puede ser sobrecargado por las clases que hereden de esta.

static: Significa que es un método de clase.

abstract: Sólo se proporciona la signatura del método. Sólo es posible en clases abstractas y no puede coincidir con los modificadores **final**, **static**, **private**.

acceso: es igual que para los atributos

tipo: el tipo de retorno del método. Puede ser cualquier tipo primitivo, clase, array o **void**

args: lista de 0 o más argumentos separados por comas, donde cada argumento tiene la forma: **tipo identificador**



• Métodos:

Ejemplos:

```
public static void main(String args[]) {
    ...
}

public long[] getSueños() {
    ...
}

public double potencia(double base, double exponente) {
    ...
}

abstract public void hacerAlgo();
```



• Métodos:

- Todo método debe finalizar con una sentencia **return** (excepto los métodos de tipo **void**) seguida de una expresión del mismo tipo que el valor de retorno.
- Los métodos estáticos sólo pueden acceder a miembros estáticos.
- Podemos tener métodos con el mismo nombre pero diferente número y/o tipo de los argumentos. En ese caso el compilador escoge el método a invocar en función del número y tipo de los argumentos suministrados. Esto se conoce como **sobrecarga** de métodos.

```
class X {
    public int producto(int a, int b) {
        return a * b;
    }
}

X x = new X();
x.producto(2, 3);
x.producto(2.0, 3.5);
```

Arrows indicate method calls:
 - Arrow from `x.producto(2, 3);` points to the `int` version of `producto`.
 - Arrow from `x.producto(2.0, 3.5);` points to the `double` version of `producto`.

```
public double producto(double a, double b) {
    return a * b;
} ...
```



```
public class Persona {

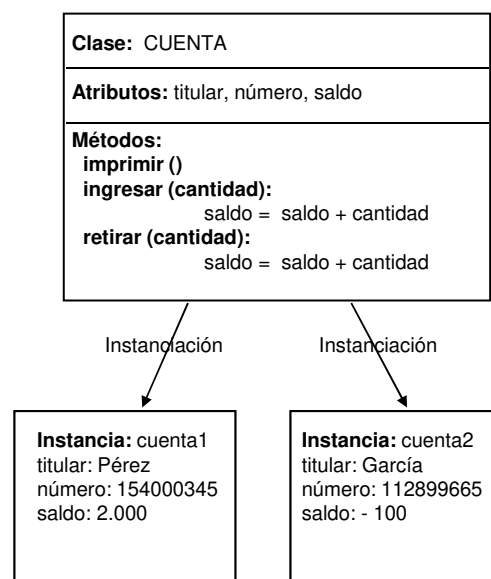
    //Atributos (privados)
    private String DNI;
    private String nombre;
    private int edad;
    private String estado;

    //Constructor
    public Persona(String DNI, String nombre,
                   int edad, String estado) {
        this.DNI = DNI;
        this.nombre = nombre;
        this.edad = edad;
        this.estado = estado;
    }

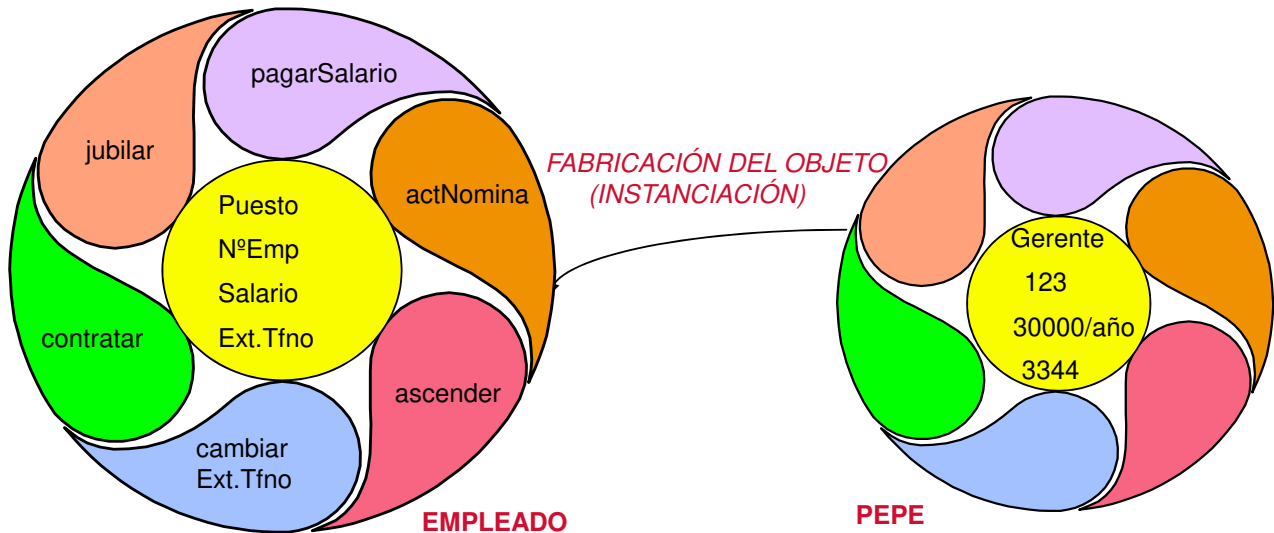
    //Métodos (públicos)
    public String getDNI() {
        return DNI;
    }
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public void cumpleaños() {
        edad++;
    }
    public String getEstado() {
        return estado;
    }
    public void setEstado(String estado) {
        this.estado = estado;
    }
    @Override
    public String toString() {
        return "Persona{" + "DNI=" + DNI
            + ", nombre=" + nombre + ", edad=" + edad
            + ", estado=" + estado + '}';
    }
}
```



- Una clase es un generador de objetos (las instancias de la clase).
- Una instancia es una estructura constituida por los atributos descritos para la clase.
- La creación de un objeto a partir de una clase se llama instanciación.
- Muchas instancias de una misma clase pueden existir simultáneamente en memoria.



La clase Empleado describe a todos los empleados, a partir de ella podemos dirigirnos a un empleado concreto dando valores a sus datos (atributos).



Definición

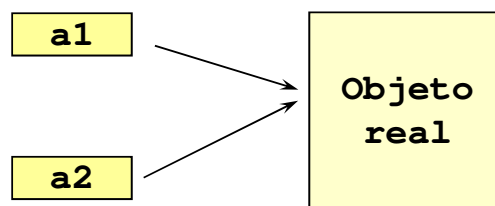
**“Es una instancia de una clase,
creada en tiempo de ejecución”**

- Es una estructura de datos formada por tantos **atributos** como tiene la clase.
- El **estado** de un objeto viene dado por el valor de los atributos. El estado suele cambiar con el paso del **tiempo**.
- Los **métodos** permiten consultar y modificar el estado del objeto.
- Durante la ejecución de un programa OO se crearán un conjunto de objetos.

- Las referencias a objetos tienen valor **null** por defecto. A diferencia de otros lenguajes, la declaración de una variable (referencia) de una cierta clase, **no implica la creación de un objeto asociado**. La vinculación entre una referencia y un objeto se hace después mediante el operador **new** o asignando el valor de una referencia a otra del mismo tipo.

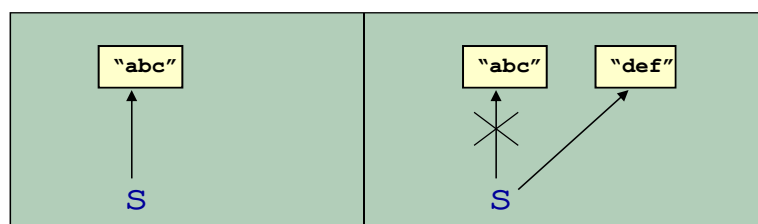
Cualquier objeto ha de ser creado explícitamente mediante el operador **new**.

```
...  
A a1, a2;  
...  
a1 = new A();  
a2 = a1;
```



- Los objetos tienen un tiempo de vida y consumen recursos durante el mismo. Cuando un objeto no se va a utilizar más, debería liberar el espacio que ocupaba en la memoria de forma que las aplicaciones no la agoten.
- En Java, la recolección y liberación de memoria es responsabilidad de un thread llamado automatic garbage collector (recolector automático de basura). Este thread monitoriza el alcance de los objetos y marca los objetos que se han salido de alcance. Veamos un ejemplo:

```
String s;           // no se ha asignado memoria todavía  
s = new String( "abc" ); // memoria asignada  
s = "def";          // se ha asignado nueva memoria  
                    // (nuevo objeto)
```





```
public class PruebaPersonas {

    public static void main(String args[]) {
        //Creamos personas
        Persona per1 = new Persona("153647458S", "Pepe", 35, "casado");
        Persona per2 = new Persona("452697419Z", "Maria", 25, "soltera");

        //Aplicamos métodos
        System.out.println("Nombre: " + per2.getNombre());
        per2.cumpleaños();
        System.out.println("Edad: " + per2.getEdad());
        System.out.println("Información completa: " + per2.toString());
    }
}
```

```
run:
Nombre: Maria
Edad: 26
Información completa: Persona{DNI=452697419Z, nombre=Maria, edad=26,
estado=soltera}
```



ATRIBUTOS Y MÉTODOS ESTÁTICOS



- Si queremos crear una clase en la que el valor de un atributo sea el mismo para todos los objetos instanciados a partir de esa clase debemos utilizar la palabra clave **static**. Para manejar este tipo de atributos necesitamos métodos estáticos.

```
public class Documento {
    private static int version;
    private int numero_de_capitulos;

    public Documento(int num_cap) {
        numero_de_capitulos = num_cap;
    }

    public void añade_un_capitulo() {
        numero_de_capitulos++;
    }

    public static void modifica_version(int i) {
        version = i;
    }

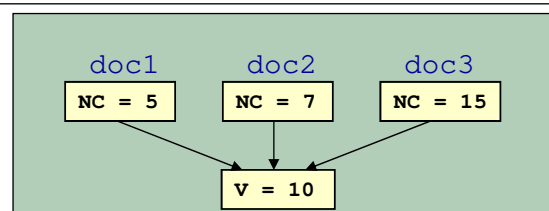
    public static int getVersion() {
        return Documento.version;
    }

    public int getNumCaps() {
        return this.numero_de_capitulos;
    }
}
```

```
public class PruebaDocumentos {

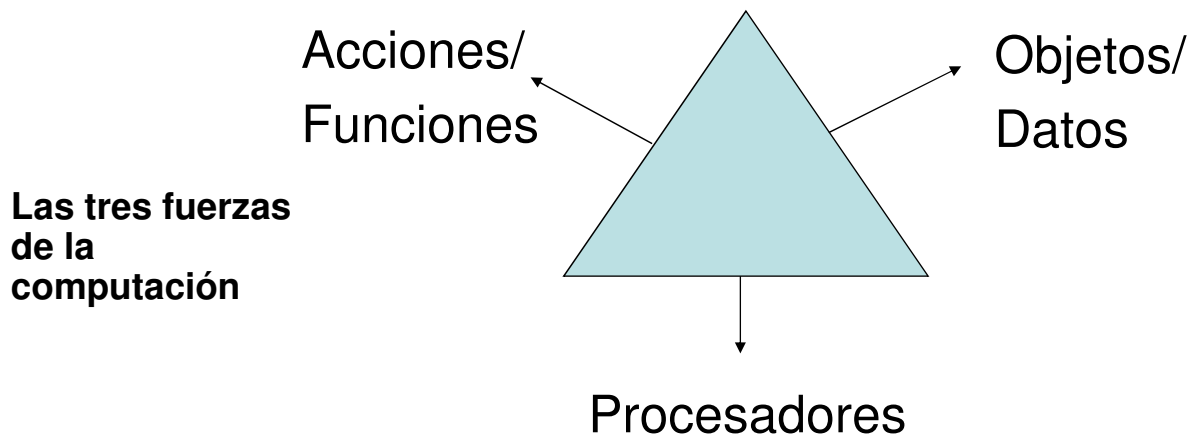
    public static void main(String args[]) {
        //Creamos documentos
        Documento doc1 = new Documento(5);
        Documento doc2 = new Documento(7);
        Documento doc3 = new Documento(15);

        //Aplicamos método de objeto
        doc1.añade_un_capitulo();
        System.out.println("Número capítulos: "
            + doc1.getNumCaps());
        //Aplicamos método de clase
        Documento.modifica_version(10);
        System.out.println("Versión: "
            + Documento.getVersion());
    }
}
```



Diseño estructurado vs. OO

- ¿Qué criterio usamos para encontrar los módulos?

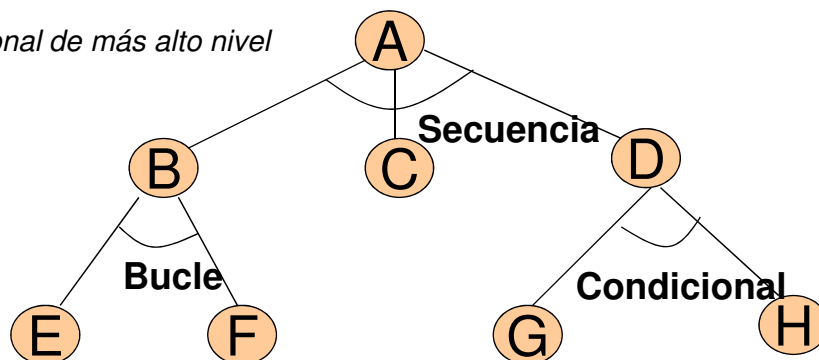


- A) Unidades de descomposición funcional → Enfoque tradicional
- B) Basándose en los principales tipos de datos → Enfoque OO

Diseño estructurado vs. OO

- **Descomposición funcional:**
- Respuesta tradicional a la cuestión de modularización.
- ¿Responde a los requisitos de modularidad?

Abstracción funcional de más alto nivel



Refinamientos sucesivos



Diseño estructurado vs. OO

- **Inconvenientes de la descomposición funcional:**
- Se basa en propiedades poco estables que dificulta la **extensibilidad**:
- ⊗ **Función principal: “Cima del sistema”**
 - El “programa principal” es una propiedad volátil.
 - Sistemas reales no tienen “cima”.
 - Mejor la visión de un “conjunto de servicios”.
- ⊗ **Centrado en la interfaz externa**
 - Primera pregunta: **¿Que hará el sistema?**
 - La arquitectura del software debe basarse en propiedades más profundas.
- ⊗ **Ordenación temporal prematura**



Diseño estructurado vs. OO

- **Inconvenientes de la descomposición funcional:**
- No promueve la **reutilización**:
- ⊗ Se desarrollan elementos software para satisfacer necesidades específicas de otro elemento del nivel superior.
- ⊗ “Cultura del proyecto actual”.
- Las estructuras de datos son descuidadas.
 - Funciones y datos deben jugar un papel complementario.
- Cuando un sistema evoluciona los datos son más estables que los procesos.



Diseño estructurado vs. OO

➤ **Ventajas de la descomposición funcional:**

- ☺ Disciplina de pensamiento lógica y bien organizada.
- ☺ Técnica simple, fácil de aplicar.
- ☺ Útil para pequeños programas y algoritmos individuales.
- ☺ Buena para documentar diseños (describir algoritmos).
- ☺ Promueve el desarrollo ordenado de sistemas.
- ☺ Adecuada para dominar la complejidad.



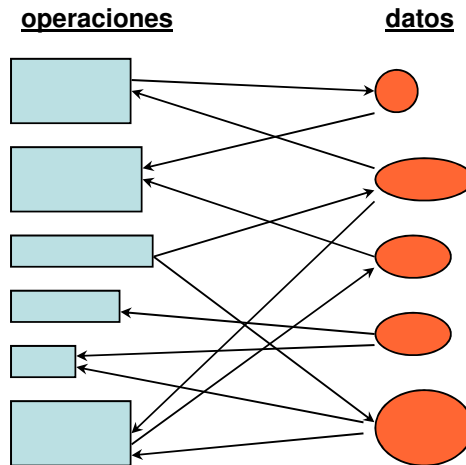
Diseño estructurado vs. OO

➤ **Descomposición modular basada en los datos:**

- Los datos son más estables que las funciones lo que favorece la extensibilidad.
- Los tipos de datos equipados con las operaciones asociadas proporcionan unidades estables para la reutilización.
- ¿Qué significa Orientación a Objetos?
 - El software se organiza como una colección de objetos que contienen tanto estructura como comportamiento.
- ¿Qué es el desarrollo Orientado a Objetos?
 - Una nueva forma de pensar acerca del software basándose en abstracciones que existen en el mundo real.

Diseño estructurado vs. OO

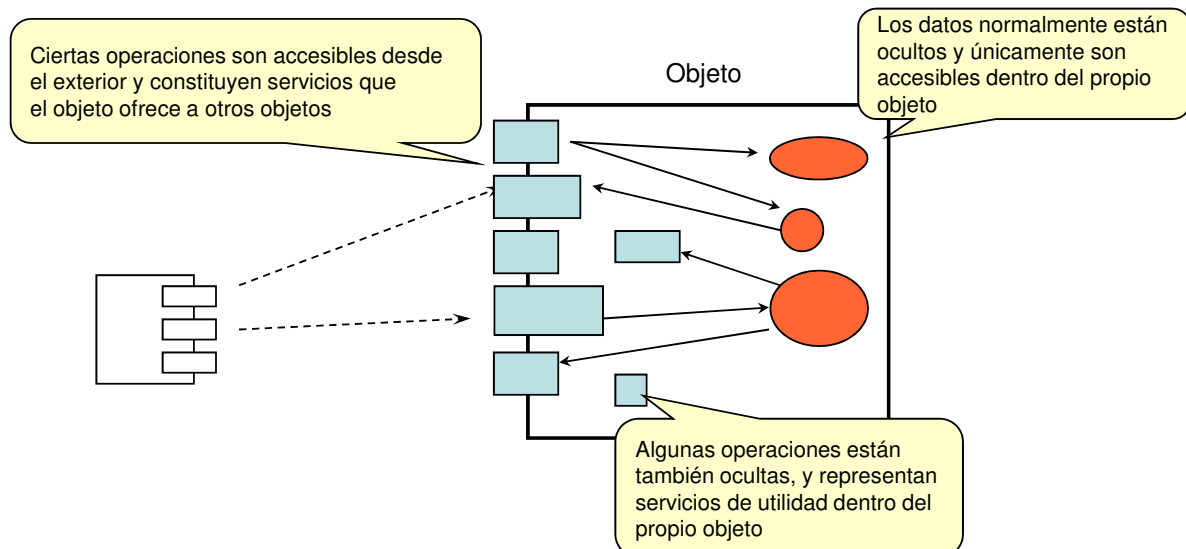
En la programación tradicional tenemos por un lado los **datos** y por otro las **operaciones sobre esos datos**, pero son entidades independientes.



39

Diseño estructurado vs. OO

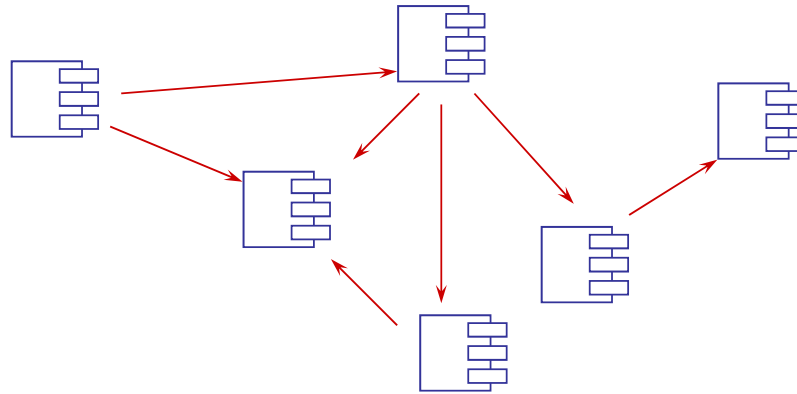
En la programación OO agrupamos (encapsulamos) **conjuntos de datos** relacionados entre sí y **operaciones sobre esos datos** en entidades que llamamos objetos.



40

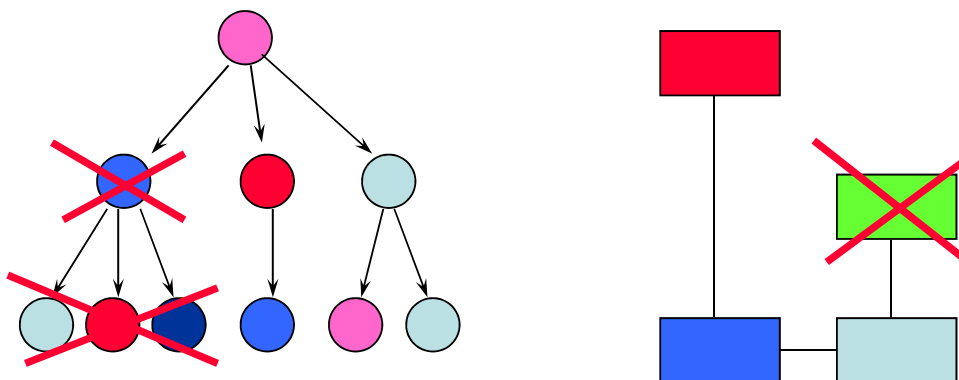
Diseño estructurado vs. OO

- Con la orientación a objetos, construimos pequeños **modelos software** de la realidad y simulamos ésta.
- Un sistema O.O. es un **conjunto de objetos** que interactúan entre sí enviándose **mensajes** mediante los cuáles se solicitan servicios unos a otros.



41

Diseño estructurado vs. OO



Las abstracciones funcionales son más volátiles que las de datos.

Esa es una de las ventajas de la OO.

42



Diseño estructurado vs. OO

- **Desarrollo de software orientado a objetos :**

Definición

- ♦ Método de desarrollo de software que basa la arquitectura del sistema en módulos deducidos de los tipos de objetos que se manipulan (en lugar de basarse en la función o funciones a las que el sistema está destinado a asegurar).
- ♦ Hay que centrar la atención no sobre lo que **HACE** el sistema, sino principalmente sobre lo que **ES** el sistema, en términos de datos, de componentes, en término de manejo de entidades, de reacción a las solicitudes.