PROG-3

UD 1

INTRODUCCIÓN AL PARADIGMA ORIENTADO A OBJETOS

Pedro J. Ponce de León, David Rizo Valero

Versión 1.0

lsi Departamento de Lenguajes y Sistemas Informáticos

Universitat d'Alacant
Universidad de Alicante

Indice

- **El progreso de la abstracción**
  - Definición de la abstracción
  - Lenguajes de programación y niveles de abstracción
  - Principales paradigmas de programación
  - Mecanismos de abstracción en los lenguajes de programación

- El paradigma orientado a objetos
  - Lenguajes orientados a objetos (LOO). Características básicas
  - LOO: Características opcionales
  - Historia de los LOO
  - Metas de la programación orientada a objetos (POO)

El progreso de la abstracción
Definición

- **Abstracción**
  - *Supresión intencionada (u ocultación) de algunos detalles de un proceso o artefacto, con el fin de destacar más claramente otros aspectos, detalles o estructuras*.

- En cada nivel de detalle cierta información se muestra y cierta información se omite.
  - Ejemplo: Diferentes escalas en mapas.

- Mediante la abstracción creamos **MODELOS** de la realidad.

El progreso de la abstracción

Lenguajes de programación y niveles de abstracción

- Los diferentes niveles de abstracción ofertados por un lenguaje, dependen de los mecanismos proporcionados por el lenguaje elegido:

  - Ensamblador
  - Procedimientos

  } **Perspectiva funcional**

  - Paquetes
  - Tipos abstractos de datos (TAD)

  } **Perspectiva de datos**

  - Objetos
    - TAD
    - + paso de mensajes
    - + herencia
    - + polimorfismo

  } **Perspectiva de servicios**

El progreso de la abstracción

Lenguajes de programación y niveles de abstracción

- Los lenguajes de programación proporcionan abstracciones

Espacio del problema

Lenguajes Orientados a Objetos (LOO)

LOO Puros

Smalltalk, Eiffel

Espacio de la solución

Lenguajes Ensamblador
Lenguajes Imperativos (C, Fortran, BASIC)

Lenguajes Específicos (LISP, PROLOG)

LOO Híbridos (Multiparadigma)

C++, Object Pascal, Java,…

El progreso de la abstracción
Principales paradigmas

- **PARADIGMA***:*
  - Forma de entender y representar la realidad.
  - Conjunto de teorías, estándares y métodos que, juntos, representan un modo de organizar el pensamiento.

- Principales **paradigmas de programación**:
  - Paradigma *Funcional*: El lenguaje describe procesos
    - Lisp y sus dialectos (p. ej. Scheme), Haskell, ML
  - Paradigma *Lógico*
    - Prolog
  - Paradigma *Imperativo* (o procedural)
    - C, Pascal
  - Paradigma *Orientado a Objetos*
    - Java, C++, Smalltalk, …

El progreso de la abstracción

Mecanismos de abstracción en los lenguajes de programación

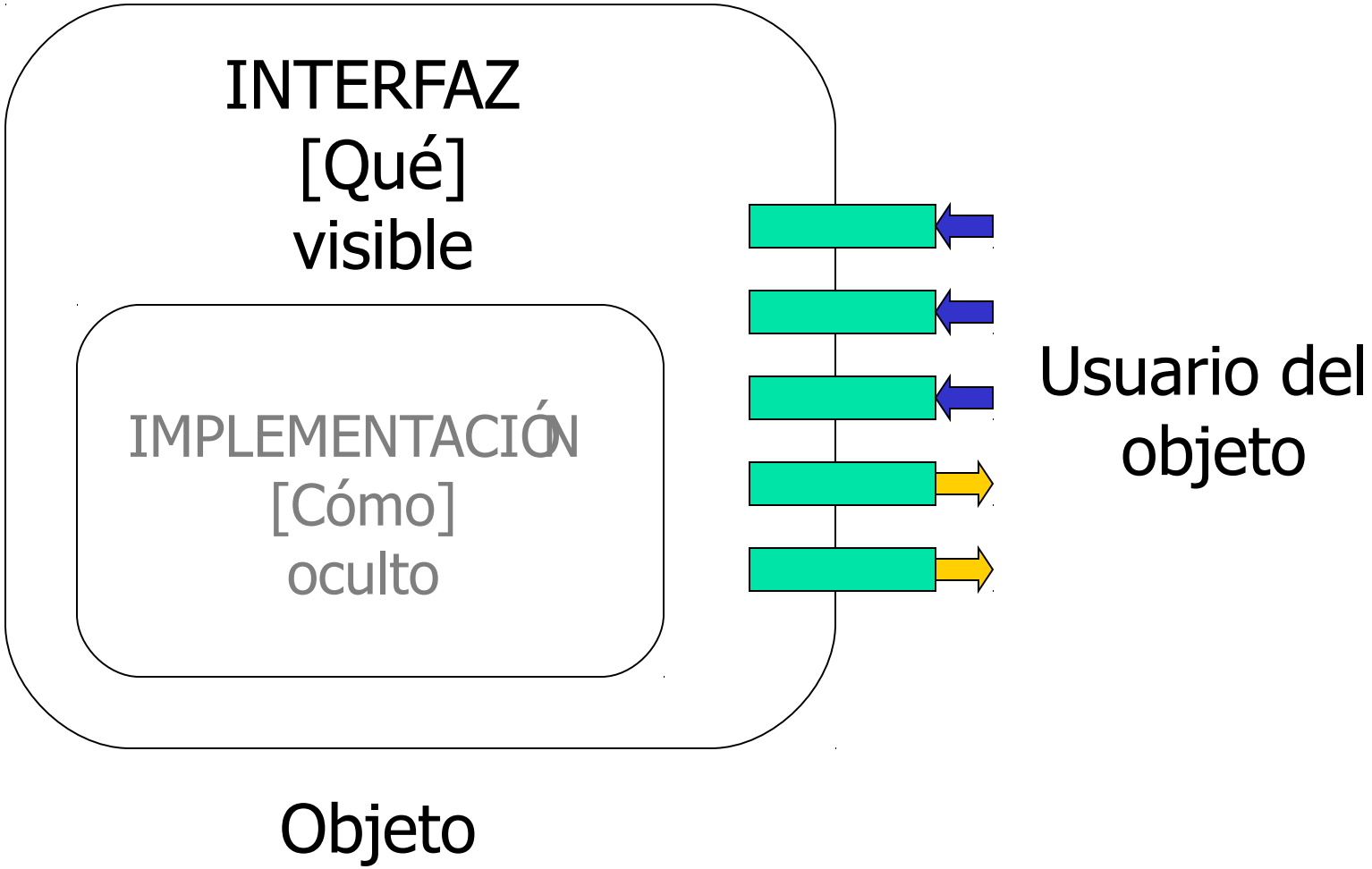
- **OCULTACIÓN DE INFORMACIÓN:**

  Omisión intencionada de detalles de implementación tras una interfaz simple.

- Cuando existe una división estricta entre la vista interna de un componente (objeto) y su vista externa hablamos de **ENCAPSULACIÓN**.
  - Estas dos vistas son:
    - **INTERFAZ**: QUÉ sabe hacer el objeto. Vista externa
    - **IMPLEMENTACIÓN**: CÓMO lo hace. Vista interna
  - Favorece la intercambiabilidad.
  - Favorece la comunicación entre miembros del equipo de desarrollo y la interconexión de los artefactos resultantes del trabajo de cada miembro.

El progreso de la abstracción

Mecanismos de abstracción en los lenguajes de programación



El progreso de la abstracción

Mecanismos de abstracción en los lenguajes de programación

Ejemplo en C++ (sin abstracción)

```
class Vector {
private:
        int [] datos;
public:
        // Método ordenación
        void burbuja();
        void insertar(int);
};

main() {
        Vector v = new Vector();
        v.insertar(28);
        // … insertamos más
elementos
        v.burbuja();
}
```

¿Quién es el usuario del objeto?

¿La operación de ordenación está abstraída?

El progreso de la abstracción

Mecanismos de abstracción en los lenguajes de programación

Ejemplo en C++ (CON abstracción)

```
class Vector {
private:
        int [] datos;
        void burbuja();
public:
        // Método ordenación
        void ordenar() {burbuja();}
        void insertar(int);
};
main() {
        Vector v;
        v.insertar(28);
        // … insertamos más
elementos
        v.ordenar();
}
```

El usuario del objeto, el **main**, usa la abstracción **ordenar** que oculta cómo realiza la operación por dentro

Indice

- El progreso de la abstracción
  - Definición de la abstracción
  - Lenguajes de programación y niveles de abstracción
  - Principales paradigmas de programación
  - Mecanismos de abstracción en los lenguajes de programación

- **El paradigma orientado a objetos**
  - Características básicas de los lenguajes orientados a objetos (LOO).
  - Características opcionales de los LOO
  - Historia de los LOO
  - Metas de la programación orientada a objetos (POO)

El paradigma orientado a objetos

- Metodología de desarrollo de aplicaciones en la cual éstas se organizan como colecciones cooperativas de **objetos,** cada uno de los cuales representan una **instancia** de alguna **clase**,y cuyas clases son miembros de **jerarquías de clases** unidas mediante relaciones de **herencia**. (Grady Booch)

- Cambia…
  - El modo de organización del programa:
    En clases (datos+operaciones sobre datos).
  - El concepto de ejecución de programa
    Paso de mensajes

- No basta con utilizar un lenguaje OO para programar orientado a objetos. Para eso hay que seguir un paradigma de programación OO.

El paradigma orientado a objetos
¿Por qué la POO es tan popular?

- POO se ha convertido durante las pasadas dos décadas en el paradigma de programación dominante, y en una herramienta para resolver la llamada *crisis del software*

- Motivos
  - POO escala muy bien.
  - POO proporciona un modelo de abstracción que razona con técnicas que la gente usa para resolver problemas (metáforas)
    - *"Es más fácil enseñar Smalltalk a niños que a programadores"* (Kay 77)

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- Ejemplo: Supongamos que Luis quiere enviar flores a Alba, que vive en otra ciudad.
  - Luis va a la floristería más cercana, regentada por un florista llamado Pedro.
  - Luis le dice a Pedro qué tipo de flores enviar a Alba y la dirección de recepción.

- El mecanismo utilizado para resolver el problema es
  - Encontrar un **agente** apropiado (Pedro)
  - Enviarle un **mensaje** conteniendo la petición (envía flores a Alba).
  - Es la **responsabilidad** de Pedro satisfacer esa petición.
  - Para ello, es posible que Pedro disponga de algún **método** (algoritmo o conjunto de operaciones) para realizar la tarea.

- Luis no necesita (ni le interesa) conocer el método particular que Pedro utilizará para satisfacer la petición: esa *información está OCULTA*.

- Así, la solución del problema requiere de la cooperación de varios individuos para su solución.

El progreso de la abstracción

Mecanismos de abstracción en los lenguajes de programación

```
class Vector {
private:
        int [] datos;
        void burbuja();
public:
        // Método ordenación
        void ordenar() {burbuja();}
        void insertar(int);
};
main() {
        Vector v = new Vector();
        v.insertar(28);
        // … insertamos más
elementos
        v.ordenar();
}
```

Quiénes son aquí:
- agente
- mensaje
- método
- responsabilidad del agente

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

Mundo estructurado en:

- Agentes y comunidades
- Mensajes y métodos
- Responsabilidades
- Objetos y clases
- Jerarquías de clases
- Enlace de métodos

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- **Agentes y comunidades**
  - Un programa OO se estructura como una comunidad de agentes que interaccionan (OBJETOS). Cada objeto juega un rol en la solución del problema. Cada objeto proporciona un servicio o realiza una acción que es posteriormente utilizada por otros miembros de la comunidad.

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- **Mensajes y métodos**
  - A un objeto se le envían mensajes para que realice una determinada acción.
  - El objeto selecciona un método apropiado para realizar dicha acción.
  - A este proceso se le denomina ***Paso de mensajes***
  - Sintaxis de un mensaje:

receptor.selector(argumentos)

```
unJuego.mostrarCarta(laCarta,42,47)
```

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- **Mensajes y métodos**
  - Un mensaje se diferencia de un procedimiento/llamada a función en dos aspectos:
    - En un mensaje siempre hay un receptor, lo cual no ocurre en una llamada a procedimiento.
    - La interpretación de un mismo mensaje puede variar en función del receptor del mismo.
      - Por tanto un nombre de procedimiento/función se identifica 1:1 con el código a ejecutar, mientras que un mensaje no.
    - Un ejemplo:

```
JuegoDeCartas juego = new Poker … ó … new Mus … ó …
juego.repartirCartas(numeroDeJugadores)
```

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- **Responsabilidades**
  - El comportamiento de cada objeto se describe en términos de responsabilidades
  - **Protocolo**: Conjunto de responsabilidades de un objeto
  - POO vs. programación imperativa

No pienses lo que puedes hacer con tus estructuras de datos.

Pregunta a tus objetos lo que pueden hacer por ti.

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- ***Entramos en la documentación de cualquier clase de una librería de C++, p.ej. Vector en https://www.sgi.com/tech/stl/Vector.html***

```
vector v;
v.push_back(3);
// v.size() == 1; v.capacity() >= 1; v[0] == 3
```

¿Cuál es la responsabilidad de Vector, al menos en cuanto a inserción?
¿Cuál es el protocolo de Vector?
(véase siguiente diapositiva)

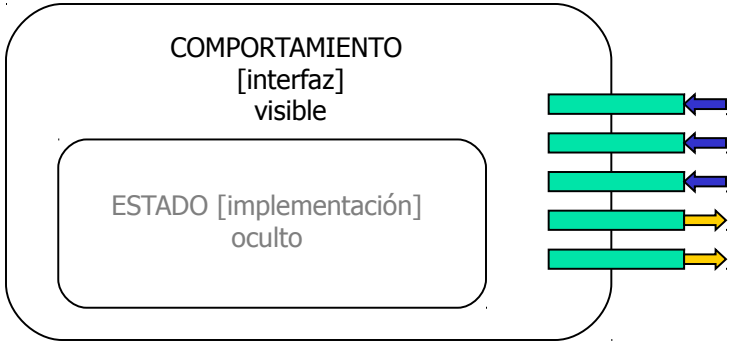
(extracto de la documentación de Vector)

`iterator begin()`	Container	Returns an `iterator` pointing to the beginning of the `vector`.
`iterator end()`	Container	Returns an `iterator` pointing to the end of the `vector`.
`const_iterator begin() const`	Container	Returns a `const_iterator` pointing to the beginning of the `vector`.
`const_iterator end() const`	Container	Returns a `const_iterator` pointing to the end of the `vector`.
`reverse_iterator rbegin()`	Reversible Container	Returns a `reverse_iterator` pointing to the beginning of the reversed vector.
`reverse_iterator rend()`	Reversible Container	Returns a `reverse_iterator` pointing to the end of the reversed vector.
`const_reverse_iterator rbegin() const`	Reversible Container	Returns a `const_reverse_iterator` pointing to the beginning of the reversed vector.
`const_reverse_iterator rend() const`	Reversible Container	Returns a `const_reverse_iterator` pointing to the end of the reversed vector.
`size_type size() const`	Container	Returns the size of the `vector`.
`size_type max_size() const`	Container	Returns the largest possible size of the `vector`.
`size_type capacity() const`	`vector`	See below.
`bool empty() const`	Container	`true` if the `vector`'s size is `0`.
`reference operator[](size_type n)`	Random Access Container	Returns the `n`'th element.
`const_reference operator[](size_type n) const`	Random Access Container	Returns the `n`'th element.
`vector()`	Container	Creates an empty vector.
`vector(size_type n)`	Sequence	Creates a vector with `n` elements.
`vector(size_type n, const T& t)`	Sequence	Creates a vector with `n` copies of `t`.
`vector(const vector&)`	Container	The copy constructor.
`template <class InputIterator> vector(InputIterator, InputIterator)` [1]	Sequence	Creates a vector with a copy of a range.
`~vector()`	Container	The destructor.
`vector& operator=(const vector&)`	Container	The assignment operator
`void reserve(size_t)`	`vector`	See below.
`reference front()`	Sequence	Returns the first element.
`const_reference front() const`	Sequence	Returns the first element.
`reference back()`	Back Insertion Sequence	Returns the last element.
`const_reference back() const`	Back Insertion Sequence	Returns the last element.
`void push_back(const T&)`	Back Insertion Sequence	Inserts a new element at the end.
`void pop_back()`	Back Insertion Sequence	Removes the last element.
`void swap(vector&)`	Container	Swaps the contents of two vectors.

….EN CÓDIGO

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- **Objetos y clases**
  - Un objeto es una **encapsulación** de un **estado** (valores de los datos) y **comportamiento** (operaciones).



  - Los objetos se agrupan en categorías (**clases**).
    - Un objeto es una **instancia** de una clase.
    - El método invocado por un objeto en respuesta a un mensaje viene determinado por la clase del objeto receptor.

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

```
class Complejo {
private:
    float real;
    float imag;
public:
    Complejo(float r, float i) {
        real = r; imag = i;
    }
    void setReal(float r){real=r;}
    float modulo() {
        return
sqrt(real*real+imag*imag);
    }
};
```

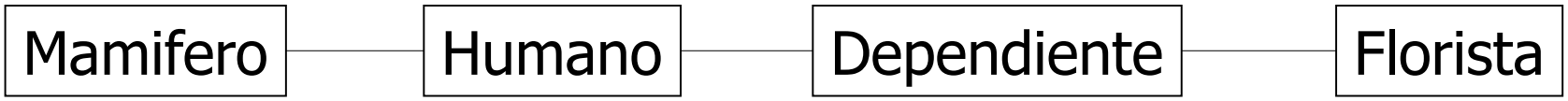
```
main() {
     Complejo a(1,2), b(5, 9);

     float m1 = a.modulo();
     float m2 = b.modulo();
     a.setReal(10);
     m1 = a.modulo();
}
```

- ¿Quiénes son objetos? ¿De qué clase instancian?
- Cuál es el estado de esos objetos. ¿Cambia en algún momento=
- Un objeto es una **encapsulación** de un **estado** (valores de los datos) y **comportamiento** (operaciones).
- ¿El método modulo es igual para ambos objetos? ¿Depende del estado de éstos?

….EN CÓDIGO

El paradigma orientado a objetos
Un nuevo modo de ver el mundo

- **Jerarquías de clases**
  - En la vida real, mucho conocimiento se organiza en términos de jerarquías. Este principio por el cual el conocimiento de una categoría más general es aplicable a una categoría más específica se denomina **generalización**, y su implementación en POO se llama **herencia**.
    - Pedro, por ser florista, es un dependiente (sabe vender y cobrar)
    - Los dependientes normalmente son humanos (pueden hablar)
    - Los humanos son mamíferos (Pedro respira oxígeno…)
  - Las clases de objetos pueden ser organizadas en una estructura jerárquica de herencia. Una clase '*hijo*' **hereda** propiedades de una clase 'padre' más alta en la jerarquía (más general):



Indice

- El progreso de la abstracción
  - Definición de la abstracción
  - Principales paradigmas de programación
  - Lenguajes de programación y niveles de abstracción
  - Mecanismos de abstracción en los lenguajes de programación

- El paradigma orientado a objetos
  - **Características básicas de los lenguajes orientados a objetos**
  - LOO: Características opcionales
  - Historia de los LOO
  - Metas de la programación orientada a objetos (POO)

Características Básicas de un LOO

- Según Alan Kay (1993), son seis:

(1) Todo es un **objeto**

(2) Cada objeto es construído a partir de otros objetos.

(3) Todo objeto es **instancia** de una **clase**

(4) Todos los objetos de la misma clase pueden recibir los mismos mensajes (realizar las mismas acciones). La clase es el lugar donde se define el **comportamiento** de los objetos y su estructura interna.

(5) Las clases se organizan en una estructura arbórea de raíz única, llamada **jerarquía de herencia**.

(6) Un programa es un conjunto de objetos que se comunican mediante el **paso de mensajes**.

Características opcionales de un LOO (1/3)

- **Polimorfismo**
  - Capacidad de una entidad de referenciar elementos de distinto tipo en distintos instantes

    p. ej., enlace dinámico
- **Genericidad**
  - Definición de clases parametrizadas (*templates en C++, generics en Java*) que definen tipos genéricos.

    p. ej.: Lista<T> : donde T puede ser cualquier tipo.

- **Gestión de Errores**
  - Tratamiento de condiciones de error mediante **excepciones**
- **Aserciones**
  - Expresiones que especifican qué hace el software en lugar de cómo lo hace
    - **Precondiciones**:propiedades que deben ser satisfechas cada vez que se invoca una servicio
    - **Postcondiciones**:propiedades que deben ser satisfechas al finalizar la ejecución de un determinado servicio
    - **Invariantes**: aserciones que expresan restricciones para la consistencia global de sus instancias.

POO
Características opcionales de un LOO (2/3)

- **Tipado estático**
  - Es la imposición de un tipo a un objeto en tiempo de compilación
    - Se asegura en tiempo de compilación que un objeto entiende los mensajes que se le envían.
  - Evita errores en tiempo de ejecución

- **Recogida de basura** (*garbage collection*)
  - Permite liberar automáticamente la memoria de aquellos objetos que ya no se utilizan.

- **Concurrencia**
  - Permite que diferentes objetos actúen al mismo tiempo, usando diferentes *threads* o hilos de control.

POO
Características opcionales de un LOO (3/3)

- **Persistencia**
  - Es la propiedad por la cual la existencia de un objeto trasciende la ejecución del programa.
    - Normalmente implica el uso de algún tipo de base de datos para almacenar objetos.

- **Reflexión**
  - Capacidad de un programa de manipular su propio estado, estructura y comportamiento.
    - En la programación tradicional, las instrucciones de un programa son 'ejecutadas' y sus datos son 'manipulados'.
    - Si vemos a las instrucciones como datos, también podemos manipularlas.

```
String instr = "System.out.println(";
ejecuta(instr + "27)");
Class c = Class.forName("String");
Method m = c.getMethod("length", null);
m.invoke(instr,null);
```

Características opcionales de un LOO: conclusiones

- Lo ideal es que un lenguaje proporcione el mayor número posible de las características mencionadas
  - Orientación a objetos no es una condición booleana: un lenguaje puede ser 'más OO' que otro.

Recomendación al alumno: las características de un LOO deberían repasarse al final de curso

Indice

- El progreso de la abstracción
  - Definición de la abstracción
  - Principales paradigmas de programación
  - Lenguajes de programación y niveles de abstracción
  - Mecanismos de abstracción en los lenguajes de programación

- El paradigma orientado a objetos
  - Características básicas de los lenguajes orientados a objetos (LOO).
  - LOO: Características opcionales
  - **Historia de los LOO**
  - Metas de la programación orientada a objetos (POO)

Historia de los L.O.O.

Año	**Lenguaje**	Creadores	Observaciones
1967	**Simula**	Norwegian Computer Center	***clase, objeto, encapsulación***
1970s	**Smalltalk**	Alan Kay	***método y paso de mensajes, enlace dinámico, herencia***
1985	**C++**	Bjarne Stroustrup	Laboratorios Bell. Extensión de C. Gran éxito comercial (1986->)
1986	**1ª Conf. OOPSLA**		**Objective C, Object Pascal, C++, CLOS**,… Extensiones de lenguajes no OO (C, Pascal, LISP,…)
'90s	**Java**	Sun	POO se convierte en el paradigma dominante. Java: Ejecución sobre máquina virtual
'00->	**C#, Python, Ruby,…**		Más de 170 lenguajes OO… Lista TIOBE (Del Top 10, 8 o 9 son OO)

Historia de los L.O.O.: **Actualidad**

- A partir de los 90’ proliferan con gran éxito la tecnología y lenguajes OO.
- Los más implantados en la actualidad son **Java**, **C++** y **PHP** (lista TIOBE)
- **C#, Python, Objective-C** son otros lenguajes OO muy utilizados
- Híbridos (OO, procedimental): PHP, C++, Visual Basic, Javascript
- Otros LOO: Ada, Delphi, Ruby, Swift, D,...

Resumen

- **El progreso de la abstracción**
  - Definición de la abstracción
  - Principales paradigmas de programación
  - Lenguajes de programación y niveles de abstracción
  - Mecanismos de abstracción en los lenguajes de programación

- **El paradigma orientado a objetos**
  - Características básicas de los lenguajes orientados a objetos (LOO).
  - LOO: Características opcionales
  - Historia de los LOO
  - Metas de la programación orientada a objetos (POO)

Indice

- El progreso de la abstracción
  - Definición de la abstracción
  - Principales paradigmas de programación
  - Lenguajes de programación y niveles de abstracción
  - Mecanismos de abstracción en los lenguajes de programación

- El paradigma orientado a objetos
  - Características básicas de los lenguajes orientados a objetos (LOO).
  - LOO: Características opcionales
  - Historia de los LOO
  - **Metas de la programación orientada a objetos (POO)**

Metas de la P.O.O.
Parámetros de Calidad (Bertrand Meyer)

- La meta última del incremento de abstracción de la POO es
  - **MEJORAR LA CALIDAD DE LAS APLICACIONES**.

- Para medir la calidad, Bertrand Meyer define unos parámetros de calidad:

  - **PARÁMETROS EXTRÍNSECOS**

  - **PARÁMETROS INTRÍNSECOS**

Metas de la P.O.O.
Principales parámetros Extrínsecos

- **Fiabilidad: corrección** + **robustez**:
  - **Corrección**: capacidad de los productos software para realizar con exactitud sus tareas, tal y como se definen en las especificaciones.
  - **Robustez**: capacidad de los sistemas software de reaccionar apropiadamente ante condiciones excepcionales.

- La corrección tiene que ver con el comportamiento de un sistema en los casos previstos por su especificación. La robustez caracteriza lo que sucede fuera de tal especificación.

Metas de la P.O.O.
Principales parámetros Intrínsecos

- **Modularidad: extensibilidad** + **reutilización**:
  - **Extensibilidad**: facilidad de adaptar los productos de software a los cambios de especificación.
  - **Reutilización**: Capacidad de los elementos software de servir para la construcción de muchas aplicaciones diferentes.
    - Las aplicaciones a menudo siguen patrones similares
- En definitiva: producir aplicaciones + fáciles de cambiar: **mantenibilidad**

Bibliografía

- Cachero et. al.
  - ***Introducción a la programación orientada a Objetos***
    - Capítulo 1
- Timothy Budd
  - ***An introduction to OO Programming. 3rd Edition***. Addison Wesley, 2002
    - Capítulos 1 y 2
- Bertrand Meyer
  - ***Object Oriented Software Construction***
- Bruce Eckel
  - ***Piensa en Java, 4ª edición***
    (Thinking in C++ / Thinking in Java, online)
    - *Capítulo 1*