Unión Europea
Fondo Social Europeo
*El FSE invierte en tu futuro*

# Desarrollo de Software. Tipos de Lenguajes. Fases.

IES Benigasló
a.micleusanu@edu.gva.es

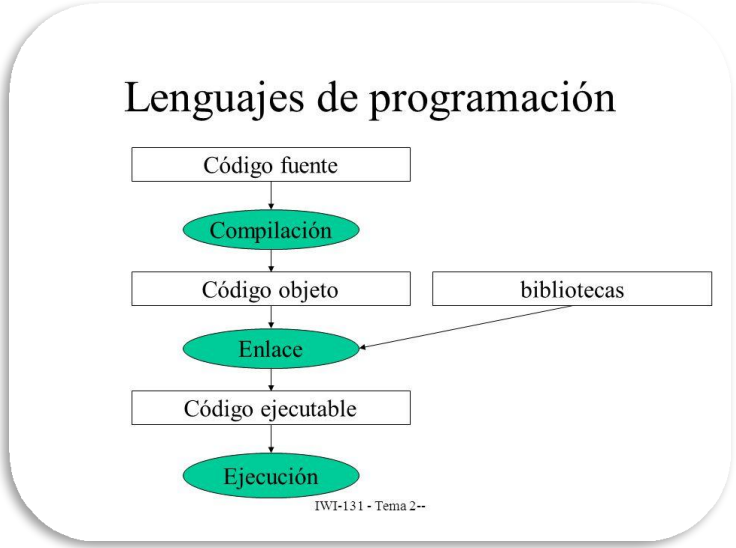
DAM Desarrollo de Aplicaciones Multiplataforma

# Antes de nada...

¿Qué es el **Software**?

# Tipos de código

- Código Fuente
- Código Objeto
- Código Ejecutable



# Código fuente

**Definición.** Conjunto de sentencias entendibles por el programador que componen el programa o una parte de ello.

Suele estar almacenado en un fichero del tipo texto como los que se pueden abrir, por ejemplo, por lo que se puede abrir con cualquier editor de texto.

El código fuente estará escrito en un lenguaje de programación determinado, elegido por el programador, como pueden ser: Basic, C, C++, C#, Java, Perl, Python, PHP.

```
public int SerieFibonacci (int limiteSerie) {
    int result=-1, a, b;
    if(limiteSerie==0 || limiteSerie==1) {
        result=limiteSerie;
    }
    else {
        a = -1;
        b = 1;
        for(int i = 0; i <= limiteSerie; i++) {
            result = a + b;
            a = b;
            b = result;
        }
    }
    return result;
}
```

# Código fuente

- Es el código escrito por los programadores con algún editor de texto.
- Utiliza sentencias y órdenes derivadas del inglés.
- Más cercano al razonamiento humano.
- Lenguaje de programación de alto nivel y contiene el conjunto de instrucciones.
- Ejemplos: Java, C, C++, Python, PHP, etc.

# Código objeto

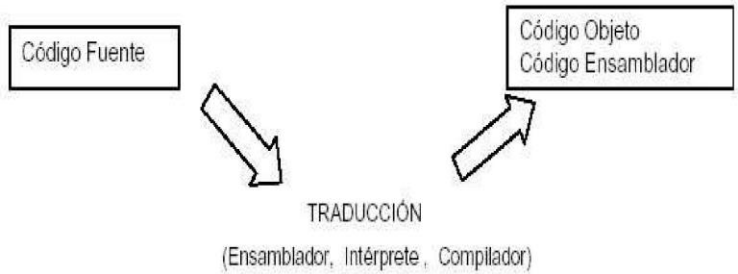
**Definición.**

Conjunto de instrucciones y datos escritos en un lenguaje que entiende el ordenador directamente: *binario* o *código máquina*.

Provienen de la traducción de cierto código fuente, es un fragmento del programa final y es específico de la plataforma de ejecución.

# Código objeto

- Es el código binario resultado de compilar el código fuente.
- La **compilación** es la traducción de una sola vez del programa y se realiza utilizando un compilador.
- La **interpretación** es la traducción y ejecución simultánea del programa línea a línea.
- El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora.
- Es un código intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un sólo paso.



# Código ejecutable

**Definición.**

Reúne diferentes códigos u objetos generados por los programadores junto con las *"librerías de uso general"* (propias del entorno o del lenguaje de programación) componiendo el programa final.

Este es el código que ejecutan los usuarios del sistema y es específico para una plataforma concreta: Windows, Linux, Mac OS, o cierto sistema Hardware.

¿Sabrías decirme distintos sistemas Hardware?

# Código ejecutable

- Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias.
- El Sistema Operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo.
- También es conocido como código máquina. Es directamente inteligible por la computadora.

# Tipos de lenguaje

El ordenador sólo entiende un lenguaje conocido como código binario o **código máquina**, consistente en ceros y unos. Es decir, sólo utiliza 0 y 1 para codificar cualquier acción.

Los lenguajes más próximos a la arquitectura hardware se denominan **lenguajes de bajo nivel** y los que se encuentran más cercanos a los programadores y usuarios se denominan **lenguajes de alto nivel**.

# Lenguaje de bajo nivel

Son lenguajes totalmente dependientes de la máquina, es decir, el programa resultante de este tipo de lenguajes no se pueden migrar o utilizar en otras máquinas. Al estar prácticamente diseñados a medida del hardware, aprovechan al máximo las características de éste.

Dentro de este grupo se encuentran:

- El **lenguaje máquina**: este lenguaje ordena a la máquina las operaciones fundamentales para su funcionamiento. Consiste en la combinación de 0's y 1's para formar las ordenes entendibles por el hardware de la máquina. Este lenguaje es mucho más rápido que los lenguajes de alto nivel. La desventaja es que son bastantes difíciles de manejar y usar, además de tener códigos fuente enormes donde encontrar un fallo es casi imposible.
- El **lenguaje ensamblador** es un derivado del lenguaje máquina y está formado por abreviaturas de letras y números llamadas mnemotécnicos. Con la aparición de este lenguaje se crearon los programas traductores para poder pasar los programas escritos en lenguaje ensamblador a lenguaje máquina. Como ventaja con respecto al código máquina es que los códigos fuentes eran más cortos y los programas creados ocupan menos memoria. Las desventajas de este lenguaje siguen siendo prácticamente las mismas que las del lenguaje máquina, añadiendo la dificultad de tener que aprender un nuevo lenguaje difícil de probar y mantener.

# Lenguaje de bajo nivel

```
00001A1E 4D 4B       LDR             R3, =(stdout_ptr - 0xC000)
00001A20 E3 58       LDR             R3, [R4,R3] ; stdout
00001A22 1B 68       LDR             R3, [R3]
00001A24 18 46       MOV             R0, R3  ; stream
00001A26 FF F7 52 EA BLX             fileno
00001A2A 03 46       MOV             R3, R0
00001A2C 18 46       MOV             R0, R3
00001A2E 4A 4B       LDR             R3, =(a1ChangeDisplay - 0x1
00001A30 7B 44       ADD             R3, PC  ; "1 ) Change displ
00001A32 19 46       MOV             R1, R3
00001A34 00 F0 34 FB BL              print
00001A38 48 4B       LDR             R3, =(stdin_ptr - 0xC000)
00001A3A E3 58       LDR             R3, [R4,R3] ; stdin
00001A3C 1B 68       LDR             R3, [R3]
00001A3E 18 46       MOV             R0, R3  ; stream
00001A40 FF F7 44 EA BLX             fileno
00001A44 02 46       MOV             R2, R0
00001A46 07 F5 43 63 ADD.W           R3, R7, #0xC30
00001A4A 10 46       MOV             R0, R2
00001A4C 19 46       MOV             R1, R3
00001A4E 4F F0 02 02 MOV.W           R2, #2
00001A52 4F F0 0A 03 MOV.W           R3, #0xA
00001A56 00 F0 77 FB BL              read
00001A5A 03 46       MOV             R3, R0
00001A5C 00 2B       CMP             R3, #0
```

```
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
11001010 11001010 11110101 00101011
11001010 11110101 00101011 00101011
11001010 00010111 11110101 00101011
00010111 11110101 00101011 00101011
11001010 11110101 00101011 00101011
```

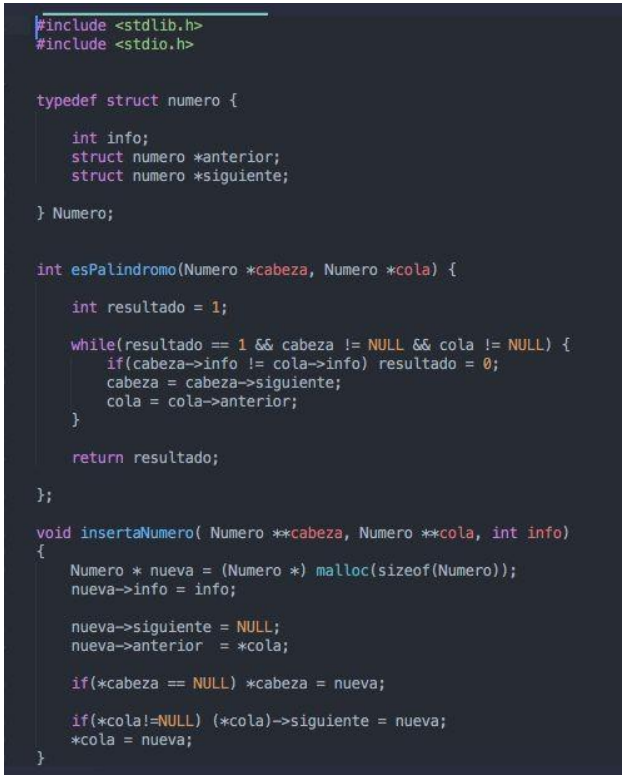
# Lenguaje de alto nivel

Se tratan de **lenguajes independientes de la arquitectura del ordenador**. Por lo que, en principio, un programa escrito en un lenguaje de alto nivel, se puede migrar de una máquina a otra sin ningún tipo de problema.

Estos lenguajes permiten al programador olvidarse por completo del funcionamiento interno de la máquina para la que está diseñando el programa. Tan solo necesitan un traductor que entienda tanto el **código fuente** como las **características de la máquina**.

# Lenguaje de alto nivel

Ejemplo lenguaje de alto nivel



```c
#include <stdlib.h>
#include <stdio.h>


typedef struct numero {

    int info;
    struct numero *anterior;
    struct numero *siguiente;

} Numero;


int esPalindromo(Numero *cabeza, Numero *cola) {

    int resultado = 1;

    while(resultado == 1 && cabeza != NULL && cola != NULL) {
        if(cabeza->info != cola->info) resultado = 0;
        cabeza = cabeza->siguiente;
        cola = cola->anterior;
    }

    return resultado;

};

void insertaNumero( Numero **cabeza, Numero **cola, int info)
{
    Numero * nueva = (Numero *) malloc(sizeof(Numero));
    nueva->info = info;

    nueva->siguiente = NULL;
    nueva->anterior  = *cola;

    if(*cabeza == NULL) *cabeza = nueva;

    if(*cola!=NULL) (*cola)->siguiente = nueva;
    *cola = nueva;
}
```

# Las Etapas de la Ingeniería del Software

- **Análisis de requerimientos:** Se extraen los requisitos del producto de software. En esta etapa la habilidad y experiencia en la ingeniería del software es crítica para reconocer requisitos incompletos, ambiguos o contradictorios. Usualmente el cliente/usuario tiene una visión incompleta/inexacta de lo que necesita y es necesario ayudarle para obtener la visión completa de los requerimientos. El contenido de comunicación en esta etapa es muy intenso ya que el objetivo es eliminar la ambigüedad en la medida de lo posible.

- **Especificación:** Es la tarea de describir detalladamente el software a ser escrito, de una forma rigurosa. Se describe el comportamiento esperado del software y su interacción con los usuarios y/o otros sistemas.

- **Diseño y arquitectura:** Determinar como funcionará de forma general sin entrar en detalles incorporando consideraciones de la implementación tecnológica, como el hardware, la red, etc. Consiste en el diseño de los componentes del sistema que dan respuesta a las funcionalidades descritas en la segunda etapa también conocidas como las entidades de negocio. Generalmente se realiza en base a diagramas que permitan describir las interacciones entre las entidades y su secuenciado.

# Las Etapas de la Ingeniería del Software

- **Programación:** Se traduce el diseño a código. Es la parte más obvia del trabajo de ingeniería de software y la primera en que se obtienen resultados «tangibles». No necesariamente es la etapa más larga ni la más compleja aunque una especificación o diseño incompletos/ambiguos pueden exigir que, tareas propias de las etapas anteriores se tengan que realizarse en esta.

- **Prueba:** Consiste en comprobar que el software responda/realice correctamente las tareas indicadas en la especificación. Es una buena praxis realizar pruebas a distintos niveles (por ejemplo primero a nivel unitario y después de forma integrada de cada componente) y por equipos diferenciados del de desarrollo (pruebas cruzadas entre los programadores o realizadas por un área de test independiente).

- **Documentación:** Realización del manual de usuario, y posiblemente un manual técnico con el propósito de mantenimiento futuro y ampliaciones al sistema. Las tareas de esta etapa se inician ya en el primera fase pero sólo finalizan una vez terminadas las pruebas.

- **Mantenimiento:** En esta etapa se realizan un mantenimiento correctivo (resolver errores) y un mantenimiento evolutivo (mejorar la funcionalidades y/o dar respuesta a nuevos requisitos).

# Ciclo de vida del Desarrollo de Software

El ciclo de vida del desarrollo Software (*SDLC* en sus siglas inglesas), es una secuencia estructurada y bien definida de las etapas en Ingeniería de software para desarrollar el producto software deseado.

# Ciclo de vida del Desarrollo de Software

- El **Paradigma de desarrollo de Software** ayuda al desarrollador a escoger una estrategia para desarrollar el software. El paradigma de desarrollo software tiene su propio set de herramientas, métodos y procedimientos, los cuales son expresados de forma clara, y define el ciclo de vida del desarrollo del software.

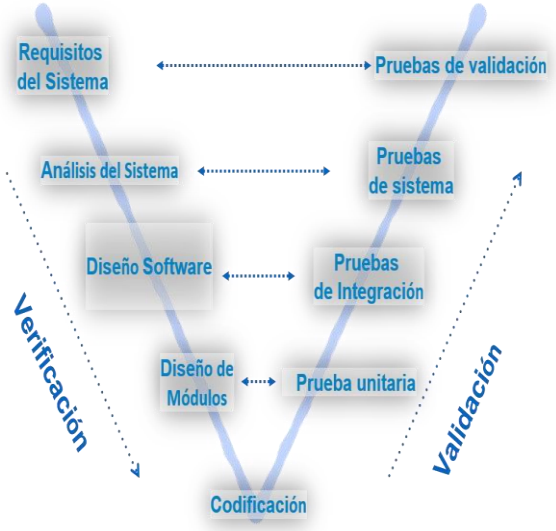
- Algunos paradigmas de desarrollo de software o modelos son los siguientes:
  - Modelo de cascada
  - Modelo en V
  - Modelo en Espiral

# Modelo de cascada

- El modelo de cascada es el modelo de paradigma más simple en desarrollo de software. Sigue un modelo en que las fases del SDLC funcionarán una detrás de la otra de forma lineal. Lo que significa que solamente cuando la primera fase se termina se puede empezar con la segunda, y así progresivamente.
- Este modelo asume que todo se lleva a cabo y tiene lugar tal y como se había planeado en la fase anterior, y no es necesario pensar en asuntos pasados que podrían surgir en la siguiente fase. Este modelo no funcionará correctamente si se dejan asuntos de lado en la fase previa. La naturaleza secuencial del modelo no permite volver atrás y deshacer o volver a hacer acciones.
- Este modelo es recomendable cuando el desarrollador ya ha diseñado y desarrollado softwares similares con anterioridad, y por eso está al tanto de todos sus dominios

# Modelo en V

- El mayor inconveniente del modelo de cascada es que solo se pasa a la siguiente fase cuando se completa la anterior, por tanto no es posible volver atrás si se encuentra algún error en las etapas posteriores. El Modelo V aporta opciones de evaluación del software en cada etapa de manera inversa.
- En cada etapa, se crea la planificación de las pruebas y los casos de pruebas para verificar y validar el producto según los requisitos de la etapa. Por ejemplo, en la etapa de recogida de requisitos, el equipo de evaluadores prepara las pruebas de caso correspondientes a los requisitos. Más tarde, cuando el producto se desarrolla y está preparado para ser evaluado, las pruebas de caso en esta etapa verifican el software y su validez según sus requisitos.
- Esto hace que tanto la verificación como la validación vayan en paralelo. Este modelo también se conoce como modelo de validación y verificación.

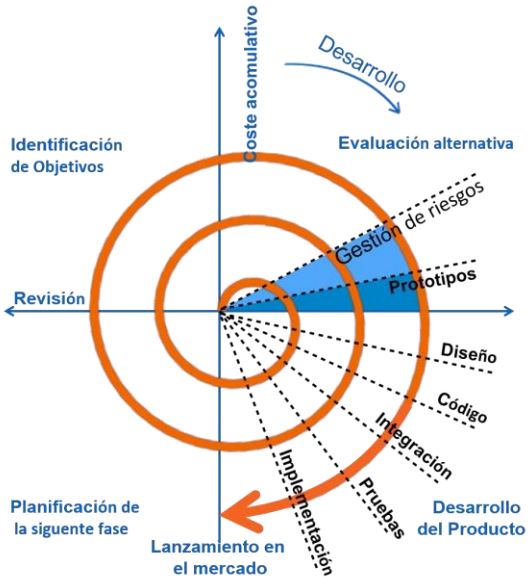


# Modelo en Espiral

- Este modelo considera el riesgo, factor que otros modelos olvidan o no prestan atención en el proceso.

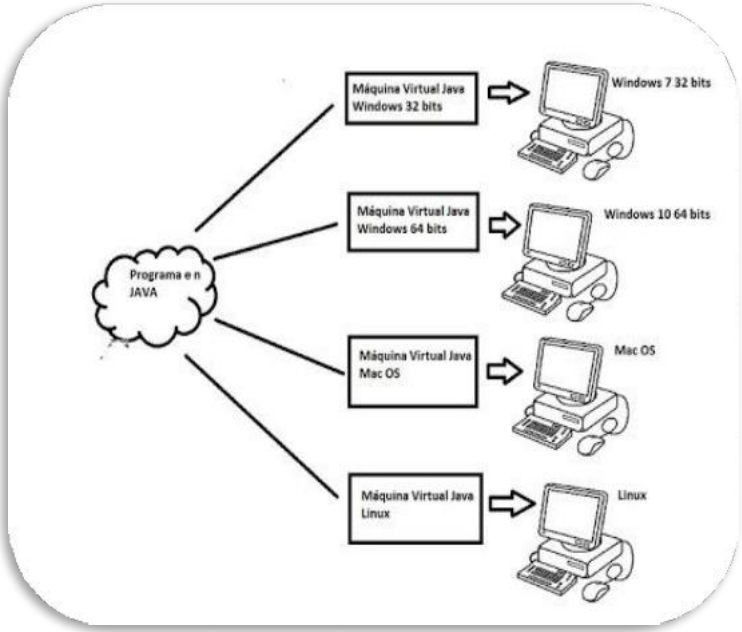
  El modelo empieza determinando los objetivos y las limitaciones del software al inicio de cada repetición.

  En la siguiente etapa se crean los modelos de prototipo del software. Esto incluye el análisis de riesgos. Luego un modelo estándar de SDLC se usa para construir el software. En la cuarta etapa es donde se prepara el plan de la siguiente repetición.

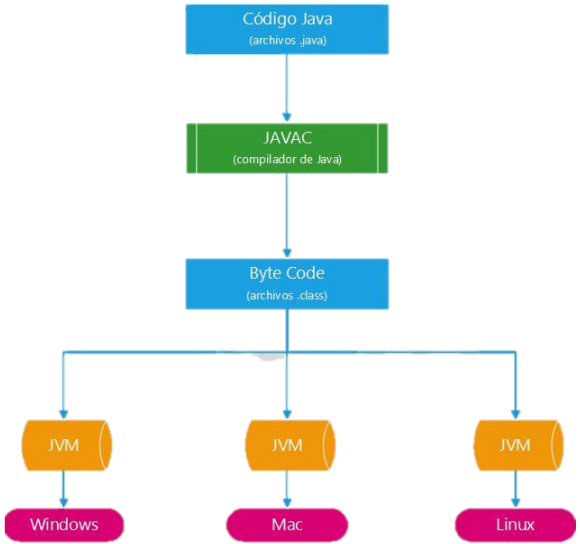


# Máquina Virtual de Java

- La máquina virtual de Java o Java Virtual Machine es un entorno de ejecución para aplicaciones de Java, cuya finalidad es la de adaptar los programas Java compilados a las características del sistema operativo donde se van a ejecutar.
- Cuando compilas una aplicación escrita en lenguaje Java, en realidad éste no se compila a lenguaje máquina, directamente entendible por el sistema operativo, sino a un lenguaje intermedio denominado **Byte Code**.
- Entre el **Byte Code** y el sistema operativo se coloca un componente especial llamado Máquina virtual que es el que realmente va a ejecutar el código.



# Proceso de MVJ o JVM



# Fin de presentación

jose.abad@iescamp.es