1 Introducción al desarrollo de software

Índice

Índice

Introducción

I *

Frédéric CHOPIN (1810-1849)

Sonata n. 3 en si menor Op. 58

I. Allegro maestoso
II. Scherzo: Molto vivace
III. Largo
IV. Finale: Presto, non tanto

3 Mazurkas Op. 59

I. n. 36 en la menor (Moderato)
II. n. 37 en la bemol mayor (Allegretto)
III. n. 38 en fa sostenido menor (Vivace)

Maurice RAVEL (1875-1937)

Gaspard de la nuit

Trois poèmes pour piano d'après Aloysius Bertrand

I. Ondine: Lent
II. Le Gibet: Très lent
III. Scarbo: Modéré

Introducción

Programa

Programa: conjunto de eventos ordenados de manera que se suceden de forma secuencial en el tiempo, uno tras otro.

Programa en TI

Un programa en TI

Un programa en TI

En los electrodomésticos, lo que se sucede son un conjunto de **órdenes** que la máquina sigue ordenadamente.

Una vez seleccionado el programa que queremos, el electrodoméstico hace todas las tareas correspondientes de manera autónoma.

Una receta es un programa

1. Espera que introduzca las zanahorias bien limpiadas, una patata y especias al gusto.
2. Gira durante 1 minuto, avanzando progresivamente hasta la velocidad 5.
3. Espera que introduzca leche y sal.
4. Gira durante 30 segundos a velocidad 7.
5. Gira durante 10 minutos a velocidad 3 mientras cuece a una temperatura de 90 grados.
6. Se detiene. La crema de zanahoria está lista!

Un programa en TI

Si no se sigue el orden … =(=(=(

Debe producir el mismo resultado

Un programa en TI

En el **mundo de TI**, la forma en que se estructuran las tareas que deben ser ejecutadas es similar a los programas de electrodomésticos.

En este caso la computadora transforma información o datos.

Un programa en TI

Un **programa informático** son una serie de **órdenes** que se llevan a cabo **secuencialmente**, aplicadas sobre un conjunto de **datos**.

Un programa en TI

Qué datos procesa un programa informático?

Un programa en TI

Qué datos procesa un programa informático?

Depende del tipo de programa

Un programa en TI

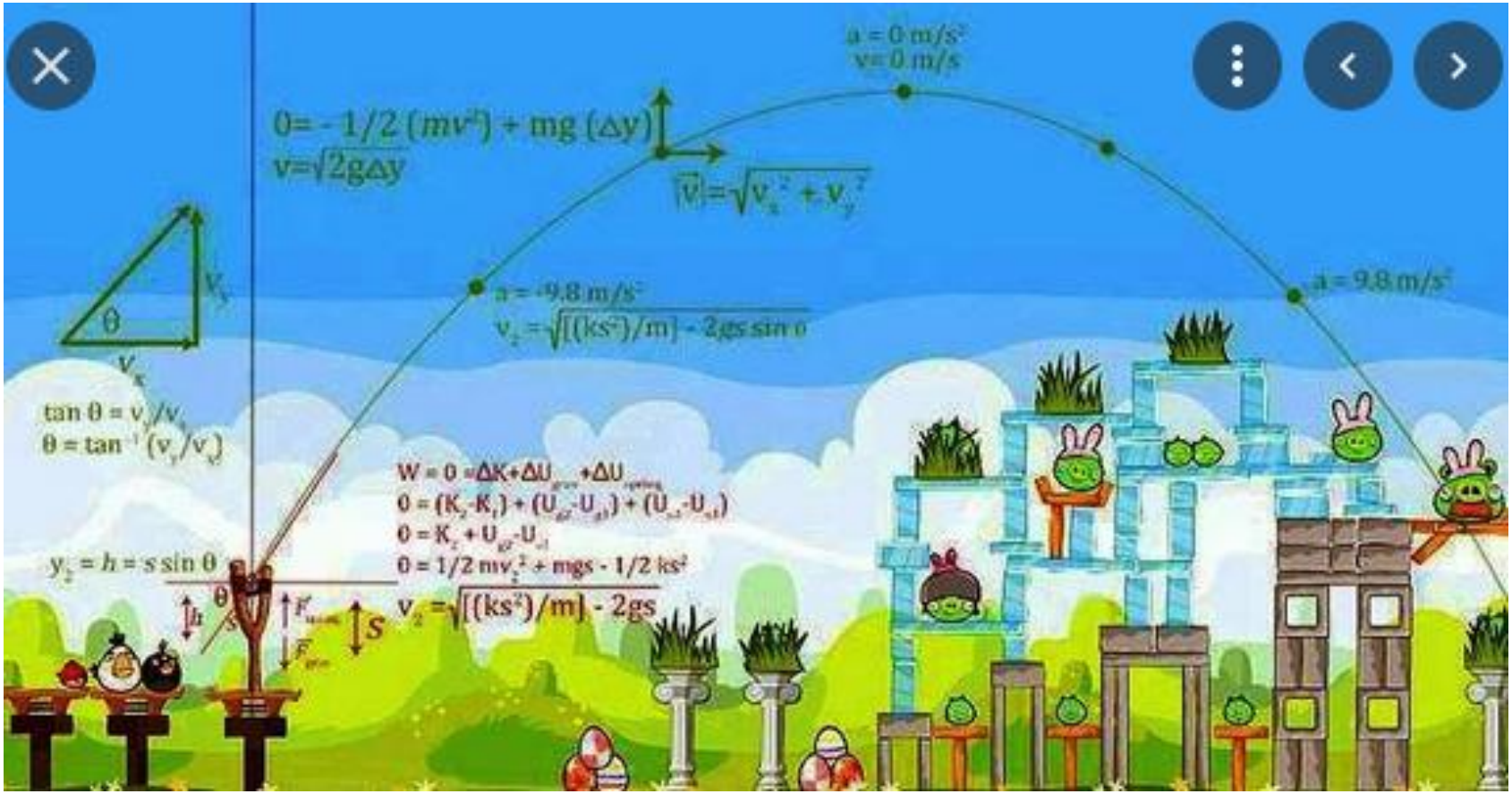
Un **editor** procesa los datos de un documento de **texto**.

Una **hoja de cálculo** procesa datos **numéricos** ubicadas en un fichero.

Un programa en TI

Un **videojuego** procesa los datos que hacen referencia a la forma y ubicación de enemigos y jugadores, las interfaces gráficas donde se encontrará el jugador, los puntos conseguidos ...

Un programa en TI



Un programa en TI

¿Qué datos podríamos almacenar en esta partida?

Un programa en TI

Un **navegador web** procesa las órdenes del usuario y los datos que recibe desde de un servidor ubicado en internet.

Un **reproductor de vídeo** procesa los fotogramas almacenados en un archivo y el audio relacionado.

Un programa en TI

La tarea de un programador informático es:

- escoger qué órdenes constituirán un programa de ordenador
- en qué orden se deben llevar a cabo
- y sobre qué datos hay que aplicarlas para que el programa lleve a cabo la tarea que tiene que resolver.

Un programa en TI

¿Programar es difícil?

Un programa en TI

¿Programar es difícil?

Depende del programa

Un programa en TI

```
pxblic class Main
{
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

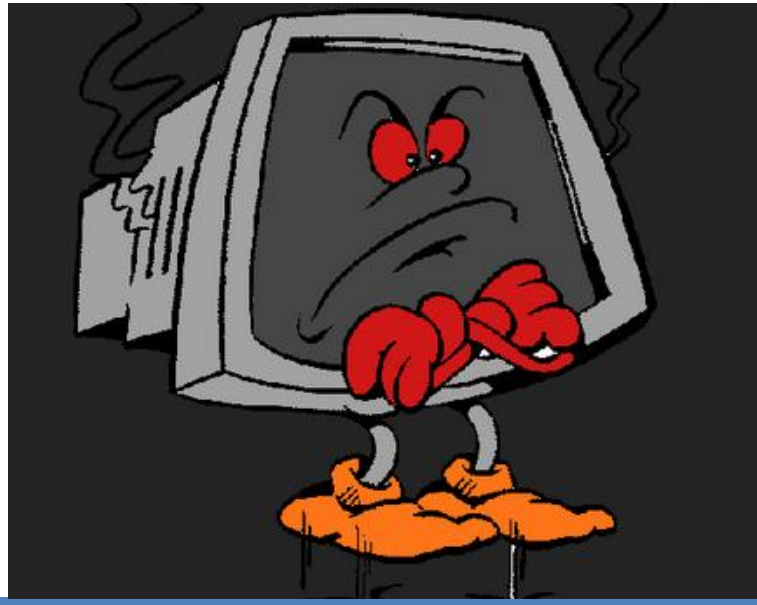
Un programa en TI

Una vez hecho el programa, cada vez que se ejecute, el ordenador cumplirá todas las órdenes del programa.

Por "ejecutar un programa" se entiende que el ordenador siga todas sus órdenes, desde la primera hasta la última.

Un programa en TI

Un ordenador es incapaz de hacer absolutamente nada por sí mismo, siempre hay que decirle qué debe hacer, mediante la ejecución de programas.



Índice

Un ordenador es una máquina binaria y solo puede trabajar con 0 y 1 : **Lenguaje máquina**

00010100110 011 101

Un ordenador es una máquina binaria y solo puede trabajar con 0 y 1: **Lenguaje máquina**

00010100110 011 101

Sumar 3 + 5 y guardar el resultado en un registro concreto

Código fuente, código objeto y código ejecutable: Máquinas virtuales

Para **crear un programa** lo que se hará será **crear un archivo** y **escribir** en un fichero la serie de **instrucciones** que se quiere que el ordenador ejecute.

Estas instrucciones deberán **seguir unas pautas determinadas en función del lenguaje de programación escogido**. Además, deberían seguir un orden determinado que dará sentido al programa escrito.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

El **código fuente** es el conjunto de archivos de texto resultantes, donde se encuentran las instrucciones

```
public class Exemple1 {


    public static void main(String[] args) {
        int i = 10;
        int j = 20;
        int res = 0;

        res = i+j;
        System.out.println("El resultado es: " + res);
    }

}
```

Código fuente, código objeto y código ejecutable: Máquinas virtuales

Este código fuente puede ser desde un **nivel muy alto**, muy cerca del lenguaje humano, hasta un **nivel más bajo**, más cercano al código de las máquinas, como ahora el código ensamblador.

00010100110 011 101

Sumar 3 + 5 y guardar el resultado en un registro concreto

```
public class Exemple1 {


    public static void main(String[] args) {
        int i = 10;
        int j = 20;
        int res = 0;

        res = i+j;
        System.out.println("El resultado es: " + res);
    }

}
```

Código fuente, código objeto y código ejecutable: Máquinas virtuales

La tendencia actual es hacer uso de lenguajes de alto nivel, es decir, cercanos al lenguaje humano.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

¡¡¡**PROBELMA**!!!

Código fuente, código objeto y código ejecutable: Máquinas virtuales

Los archivos de código fuente no contienen el lenguaje máquina que entenderá el ordenador.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

Por lo tanto, resultan **incomprensibles** para el **procesador**.

Para poder generar código máquina hay que hacer un proceso de **traducción** desde los mnemotécnicos que contiene cada archivo a las secuencias binarias que entiende el procesador.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

Por lo tanto, resultan **incomprensibles** para el **procesador**.

Para poder generar código máquina hay que hacer un proceso de **traducción** desde los mnemotécnicos que contiene cada archivo a las secuencias binarias que entiende el procesador.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

compilación : traducción del código fuente de los archivos del programa en ficheros en formato binario que contienen las instrucciones en un formato que el procesador puede entender.

El programa que hace este proceso se denomina **compilador**.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

El **código objeto** es el código fuente traducido (por el compilador) a código máquina, pero este código aún **no puede ser ejecutado por el ordenador**.

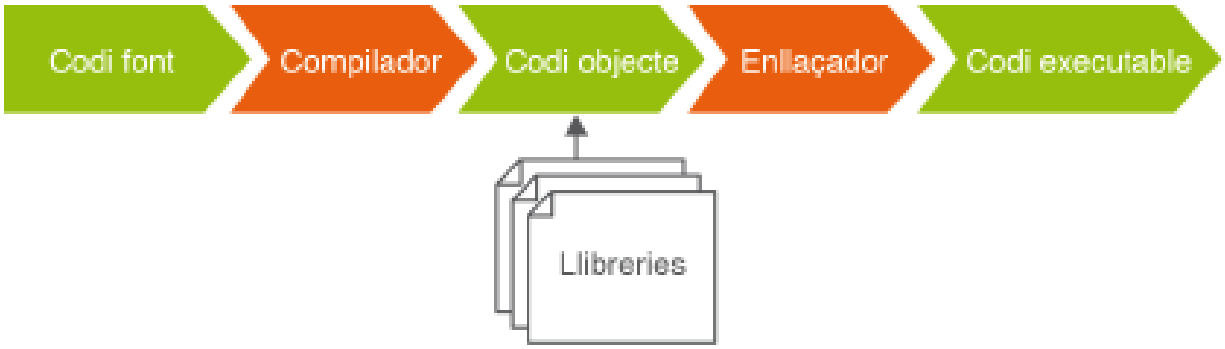
Código fuente, código objeto y código ejecutable: Máquinas virtuales

El **código ejecutable** es la traducción completa en código máquina, llevada a cabo por el enlazador.

El **código ejecutable** es **interpretado** directamente por el ordenador.

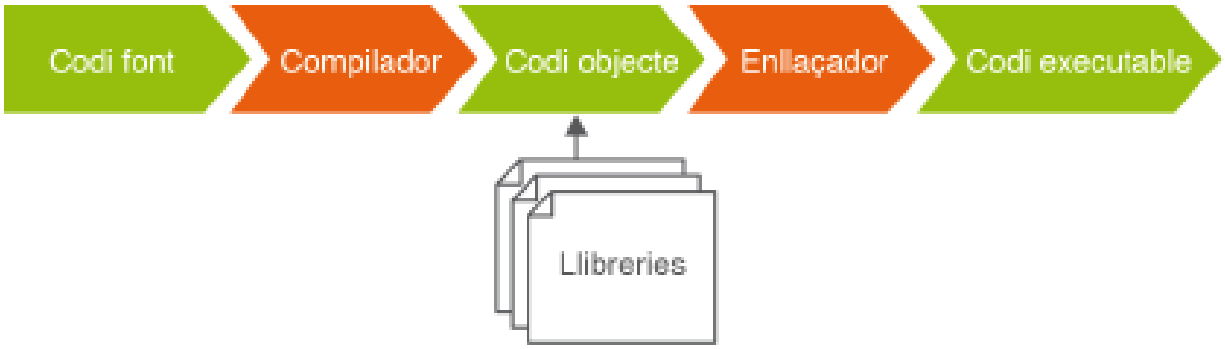
Código fuente, código objeto y código ejecutable: Máquinas virtuales

El **enlazador** es el encargado de insertar al código objeto las funciones de las librerías que son necesarias para el programa y de llevar a cabo el proceso de montaje generando un archivo ejecutable.



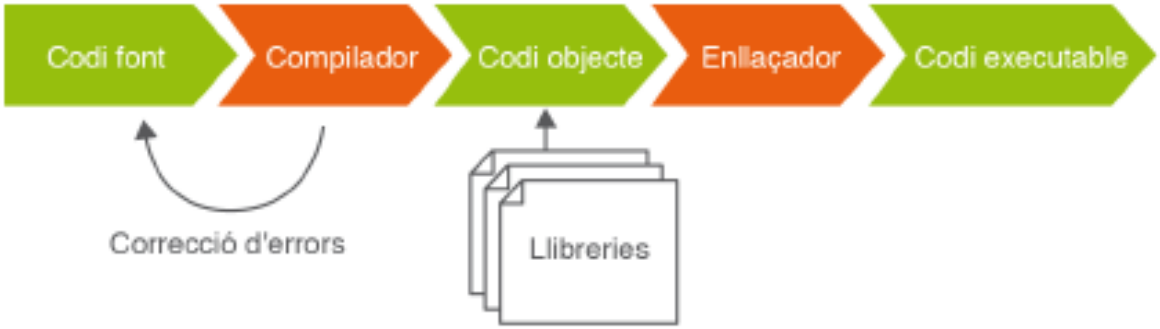
Código fuente, código objeto y código ejecutable: Máquinas virtuales

Una **librería** (library en inglés) es un colección de código predefinido que facilita la tarea del programador a la hora de codificar un programa.



Código fuente, código objeto y código ejecutable: Máquinas virtuales

El **código fuente** desarrollado por los programadores se convertirá en **código objeto** con la ayuda del **compilador**. Este ayudará a localizar los errores de sintaxis o de compilación que se encuentren en el código fuente. Con el **enlazador**, que recogerá el código objeto y las **librerías**, se generará el **código ejecutable**.



Código fuente, código objeto y código ejecutable: Máquinas virtuales

El concepto de **máquina virtual** surge con el objetivo de facilitar el desarrollo de compiladores que generan código para **diferentes procesadores**.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

La compilación consta de dos fases:

•La primera parte del código fuente en un lenguaje intermedio obteniendo un programa equivalente con un menor nivel de abstracción que el original y que no puede ser directamente ejecutado.

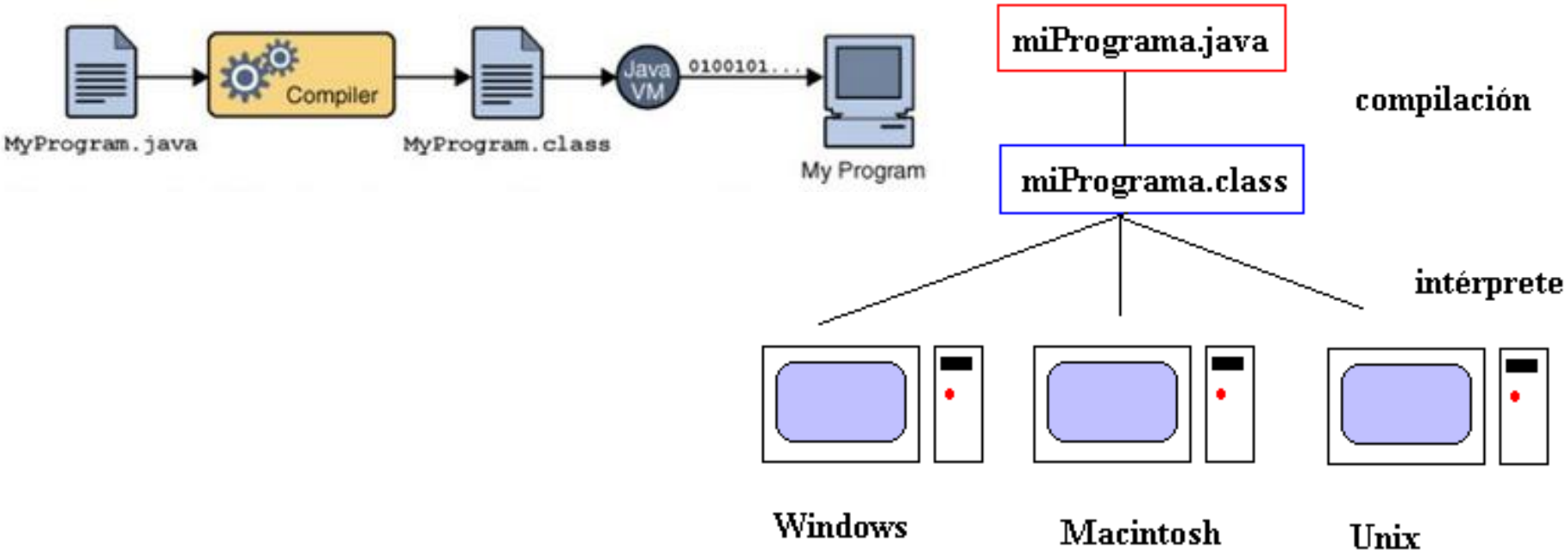
•La segunda fase traduce el lenguaje intermedio a un lenguaje comprensible por la máquina.

Código fuente, código objeto y código ejecutable: Máquinas virtuales

¿por qué dividir la compilación en dos fases?

El objetivo es que el código de la primera fase, el código intermedio, sea común para cualquier procesador, y que el código generado en la segunda fase sea el específico para cada procesador.

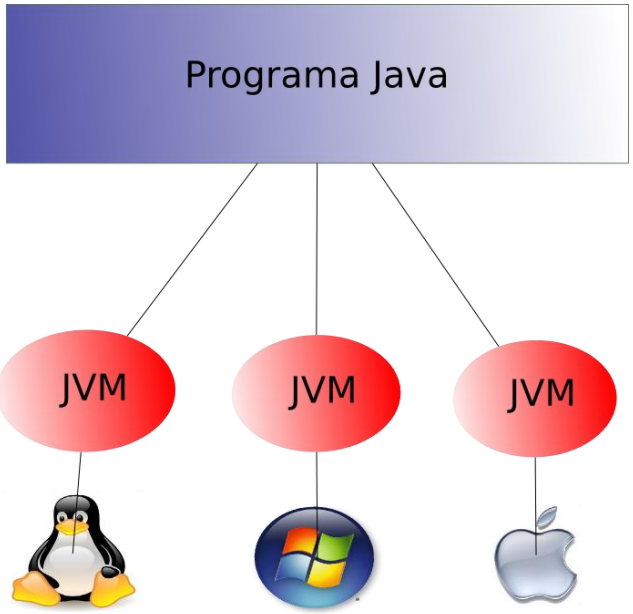
La traducción del lenguaje intermedio al lenguaje máquina no se suele hacer mediante compilación sino mediante un intérprete



La Java Virtual Machine (JVM)

La máquina virtual Java se sitúa en un nivel superior al hardware sobre el que se desea ejecutar la aplicación y actúa como un puente entre el código a ejecutar y el sistema.

La JVM será la encargada, al ejecutar la aplicación, de convertir el código de bytes a código nativo de la plataforma física.



Ventaja: la portabilidad de la aplicación a diferentes plataformas con el único requerimiento de disponer de la JVM para el sistema correspondiente.

Índice

Tipos de lenguajes de programación

Un **lenguaje de programación** es un lenguaje que permite establecer una comunicación entre el hombre y la máquina.

El lenguaje de programación identificará el código fuente, que el programador desarrollará para indicar a la máquina qué pasos debe dar, una vez este código se haya convertido en código ejecutable.

Tipos de lenguajes de programación

Los diferentes tipos de lenguajes son:

Lenguaje de 1ª generación o lenguaje máquina.

Lenguajes de 2ª generación o lenguajes de ensamblador.

Lenguajes de 3ª generación o lenguajes de alto nivel.

Lenguajes de 4ª generación o lenguajes de propósito específico.

Lenguajes de 5ª generación.

Tipos de lenguajes de programación

El primer tipo de lenguaje que se desarrolló es el llamado **lenguaje de primera generación o lenguaje máquina**. Es el único lenguaje que entiende el ordenador directamente.

00010100110 011 101

Sumar 3 + 5 y guardar el resultado en un registro concreto

Tipos de lenguajes de programación

- Las instrucciones se expresan en **código binario**

- Esto hace que la programación en este lenguaje resulte tediosa y complicada, ya que se requiere un **conocimiento** profundo de la **arquitectura** física del **ordenador**.

Tipos de lenguajes de programación

- El programador utiliza la totalidad de recursos del hardware, con lo cual se pueden obtener programas muy eficientes.

- Prácticamente en desuso

El segundo tipo de lenguaje de programación son los lenguajes de segunda generación o **lenguajes de ensamblador**.

Se trata del primer lenguaje de programación que utiliza códigos **mnemotécnicos** para indicar a la máquina las operaciones que debe llevar a cabo. Estas operaciones, muy básicas, han sido diseñadas a partir del conocimiento de la estructura interna de la propia máquina.

Tipos de lenguajes de programación

000101001100 011 101

Sumar 3 + 5 y guardar el resultado en un registro concreto

SUMA A0 3 5

Sumar 3 + 5 y guardar el resultado en un registro concreto

Cada instrucción en lenguaje ensamblador corresponde a una instrucción en lenguaje máquina. Estos tipos de lenguajes dependen totalmente del procesador que utilice la máquina, por eso se dice que están orientados a las máquinas.

Tipos de lenguajes de programación

A partir del código escrito en lenguaje ensamblador, el programa traductor (ensamblador) lo convierte en código de primera generación, que será interpretado por la máquina.

En general se utiliza este tipo de lenguajes para **programar controladores (drivers)** o **aplicaciones de tiempo real**, ya que requiere un uso muy **eficiente** de la velocidad y de la memoria.

Tipos de lenguajes de programación

Como **ventajas** de los lenguajes de primera y segunda generación se pueden establecer:

•Permiten escribir **programas muy optimizados** que aprovechan al máximo el hardware (hardware) disponible.

•Permiten al programador especificar exactamente qué instrucciones quiere que se ejecuten.

Los **inconvenientes** son los siguientes:

- Los programas escritos en lenguajes de bajo nivel están completamente **ligados al hardware** donde se ejecutarán y no se pueden trasladar fácilmente a otros sistemas con un hardware diferente.

Tipos de lenguajes de programación

- Hay que **conocer a fondo la arquitectura** del sistema y del procesador para escribir buenos programas.

- No permiten expresar de forma directa conceptos habituales a nivel de algoritmo.

- Son **difíciles** de codificar, documentar y mantener.

Tipos de lenguajes de programación

El siguiente grupo de lenguajes se conoce como lenguajes de **tercera generación** o **lenguajes de alto nivel**. Estos lenguajes, más evolucionados, utilizan palabras y frases relativamente fáciles de entender y proporcionan también facilidades para expresar alteraciones del flujo de control de una forma bastante sencilla e intuitiva.

Tipos de lenguajes de programación

Los lenguajes de tercera generación o de alto nivel **se utilizan cuando se quiere desarrollar aplicaciones grandes y complejas,** donde se prioriza el hecho de facilitar y comprender cómo hacer las cosas (lenguaje humano) por encima del rendimiento del software o del uso de la memoria.

Tipos de lenguajes de programación

- Surgen de los esfuerzos encaminados a hacer la tarea de **programación independiente de la máquina** donde se ejecutarán.

- Son normalmente fáciles de aprender porque están formados por elementos de lenguajes naturales

Factorial de un número en Basic

```
1  ' ____________________________________
2  ' Funció Factorial
3  ' ____________________________________
4  Public Function Factorial(num As Integer)As String
5  Dim i As Integer
6      For i = 1 To num - 1
7        num = num * i
8        Factorial = num
9      Next
10 End Function
```

Tipos de lenguajes de programación

Ejemplos de lenguajes de programación de tercera generación: **C, C ++, Java, Pascal, FORTRAN, ALGOL, COBOL, BASIC, Python**

Tipos de lenguajes de programación

Como consecuencia de este alejamiento de la máquina y acercamiento a las personas, los programas escritos en lenguajes de programación de tercera generación no pueden ser interpretados directamente por el ordenador, sino que es necesario llevar a cabo previamente su **traducción a lenguaje máquina**. Hay dos tipos de traductores: los **compiladores** y los **intérpretes.**

COMPILADORES

Son programas que **traducen el programa escrito con un lenguaje de alto nivel al lenguaje máquina**.

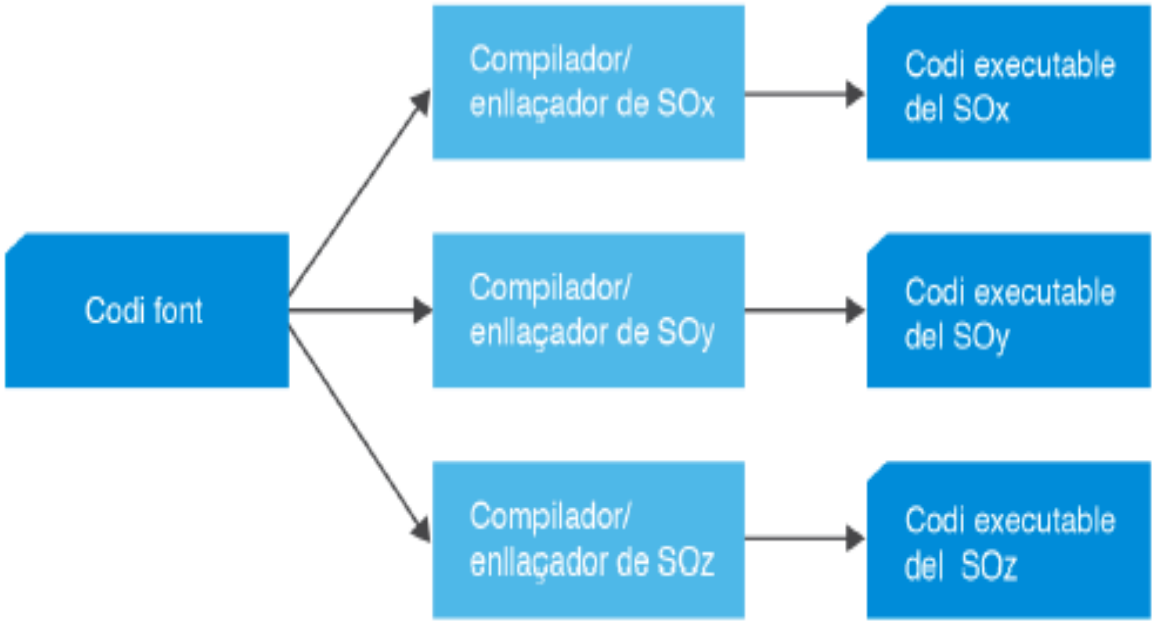
El compilador detectará los posibles errores del programa fuente para conseguir un programa ejecutable depurado.

Lenguajes Compilados: Pascal, C, C ++

Tipos de lenguajes de programación

- Crear el código fuente.
- Crear el código ejecutable en uso de compiladores y enlazadores.
- El código ejecutable depende de cada SO. Para cada SO hay un compilador, es decir, si se quiere ejecutar el código con SO debe recompilar el código fuente.
- El programa resultante se ejecuta directamente desde SO

Dependencia del sistema operativo a la hora de elegir y utilizar compilador



INTÉRPRETES

El **intérprete** también es un programa que **traduce el código de alto nivel al lenguaje máquina**, pero, a diferencia del compilador, lo hace **en tiempo de ejecución.** Es decir, no se hace un proceso previo de traducción de todo el programa fuente a código de bytes, sino que se **va traduciendo y ejecutando instrucción por instrucción.**

Tipos de lenguajes de programación

Algunas **características** de los lenguajes interpretados son:

•El código interpretado no es ejecutado directamente por el sistema operativo, sino que hace uso de un intérprete.

•Cada sistema tiene su propio intérprete

Lenguajes interpretados : JavaScript, PHP, ASP, BASIC, **Perl**, **Python**, **Ruby**

Compiladores vs intérpretes

El intérprete es más lento que el compilador, ya que lleva a cabo la traducción vez que la ejecución. Además, esta traducción se hace siempre que ejecuta el programa, mientras que el compilador sólo la lleva a cabo una vez.

La ventaja de **los intérpretes** es que **hacen que los programas sean más portables**. Así, un programa compilado en un ordenador con sistema operativo Windows no funcionará en un Macintosh, o en un ordenador con sistema operativo Linux, a menos que se vuelva a compilar el programa fuente en el nuevo sistema.

Tipos de lenguajes de programación

Los lenguajes de tercera generación son aquellos que son capaces de contener y ejecutar, en una sola instrucción, el equivalente a varias instrucciones de un lenguaje de segunda generación.

Tipos de lenguajes de programación

Las **ventajas** de los **lenguajes de tercera generación** son:

•El **código** de los programas es mucho **más sencillo** y **comprensible**.

•Son **independientes del hardware** (no hacen ninguna referencia). Por este motivo es posible "llevar" el programa entre diferentes ordenadores / arquitecturas / Sistemas operativos (siempre que en el sistema de destino exista un compilador para este lenguaje de alto nivel).

•**Es más fácil y rápido escribir los programas y más fácil mantenerlos.**

Tipos de lenguajes de programación

Los **inconvenientes** de los lenguajes de tercera generación son:

•Su **ejecución** en un ordenador puede resultar **más lenta** que el mismo programa escrito en lenguaje de bajo nivel, aunque esto depende mucho de la calidad del compilador que haga la traducción.

Tipos de lenguajes de programación

Los lenguajes de **cuarta generación o lenguajes de propósito específico**. Aportan un nivel muy alto de abstracción en la programación, permitiendo desarrollar aplicaciones sofisticadas en un espacio corto de tiempo, muy inferior al necesario para los lenguajes de 3ª generación.

Tipos de lenguajes de programación

Se automatizan ciertos aspectos que antes había que hacer a mano.

Incluyen herramientas orientadas el desarrollo de aplicaciones (IDE) que permiten definir y gestionar bases de datos, realizar informes (p.ej .: Oracle reports), consultas (p.ej .: informix 4GL), módulos ..., escribiendo muy pocas líneas de código o ninguna.

Permiten la creación de prototipos de una aplicación rápidamente. Los prototipos permiten tener una idea del aspecto y del funcionamiento de la aplicación antes de que el código esté terminado. Esto facilita la obtención de un programa que reúna las necesidades y expectativas del cliente.

Ventajas 4ª generación:

- Mayor abstracción.
- Menor esfuerzo de programación.
- Menor coste de desarrollo del software.
- Basados en generación de código a partir de especificaciones de nivel muy alto.

Tipos de lenguajes de programación

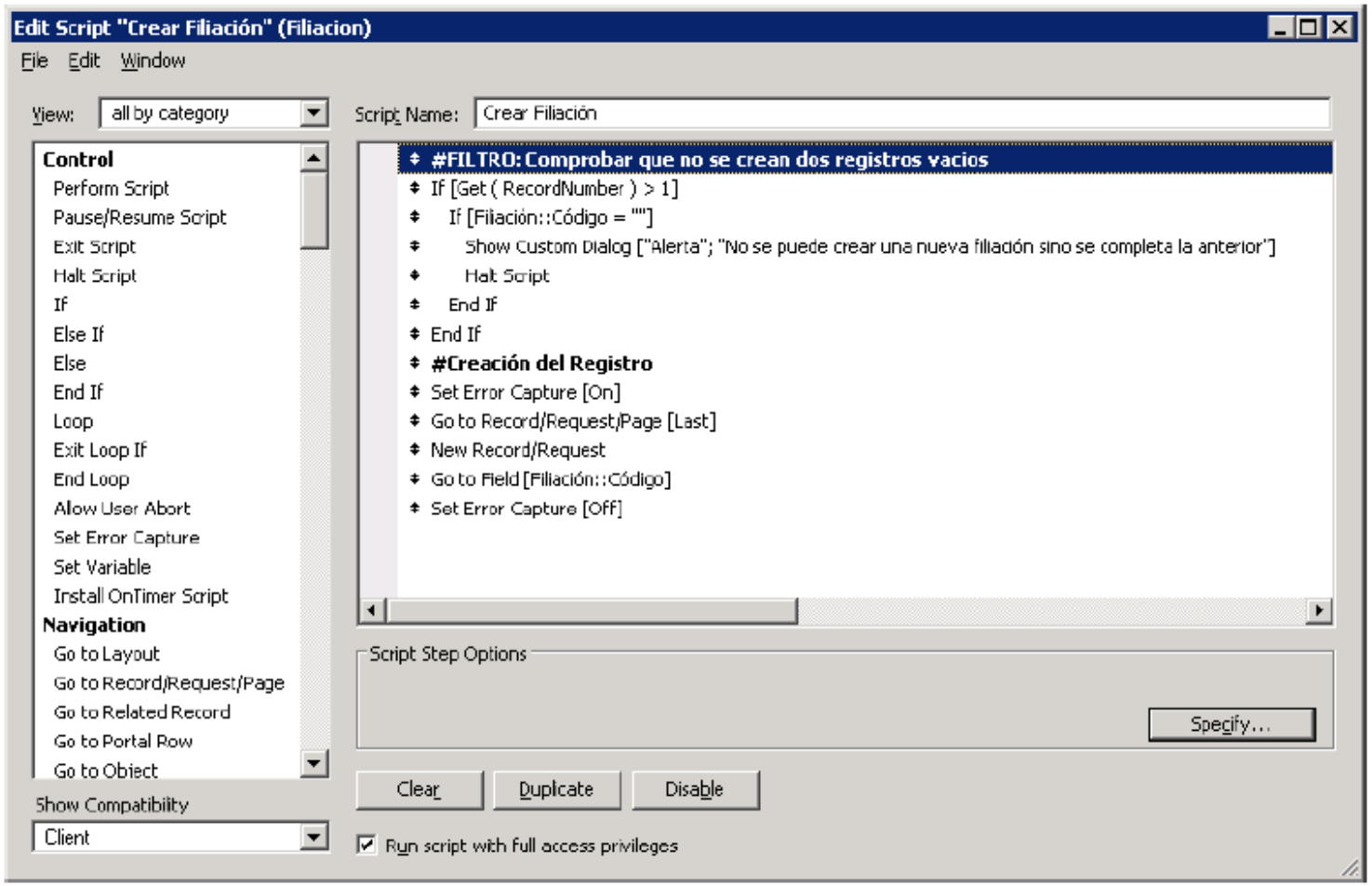
Ventajas 4ª generación:

- Se pueden llevar a cabo aplicaciones sin ser un experto en el lenguaje.
- Suelen tener un conjunto de instrucciones limitado.
- Son específicos del producto que les ofrece.

Estos lenguajes de programación de cuarta generación están orientados, básicamente, a las **aplicaciones de negocio y el manejo de bases de datos.**

Lenguajes de 4ª generación: Visual Basic, Visual Basic .NET, ABAP de SAP, FileMaker, PHP, ASP, 4D

Ejemplo FileMaker



Tipos de lenguajes de programación

Los lenguajes de **quinta generación** son lenguajes específicos para el tratamiento de problemas relacionados con la inteligencia artificial y los sistemas expertos.

Tipos de lenguajes de programación

En lugar de ejecutar sólo un conjunto de comandos, el objetivo de estos sistemas es "Pensar" y anticipar las necesidades de los usuarios. Estos sistemas se encuentran todavía en desarrollo.

Algunos ejemplos de lenguajes de quinta generación son **Lisp** o **Prolog**.

Índice

Es difícil establecer una clasificación general de los lenguajes de programación, ya que existe un gran número de lenguajes y, a veces, diferentes versiones de un mismo lenguaje. Esto provocará que en cualquier clasificación que se haga un mismo lenguaje pueda pertenecer a más de uno de los grupos establecidos.

Paradigmas de programación

Una clasificación muy extendida, atendiendo a la forma de trabajar de los programas y la filosofía con la que fueron concebidos, es la siguiente:

- **Paradigma imperativo / estructurado.**
- **Paradigma de objetos.**
- **Paradigma funcional.**
- **Paradigma lógico.**

El **paradigma imperativo / estructurado** debe su nombre al papel dominante que ejercen las **sentencias imperativas**, es decir aquellas que indican llevar a cabo una determinada operación que modifica los datos guardados en memoria.

Algunos de los lenguajes imperativos son **C, Basic, Pascal, Cobol** ...

La técnica seguida en la programación imperativa es la **programación estructurada**.

La idea es que **cualquier programa**, por complejo y grande que sea, puede ser representado mediante **tres tipos de estructuras de control**: **Secuencia, Selección e Iteración**.

Características de la programación estructurada

Las **características** de la **programación estructurada** son la **claridad**, el **teorema de la estructura** y el **diseño descendente**.

Claridad

Deberá haber suficiente información al código para que el programa pueda ser entendido y verificado: **comentarios, nombres de variables comprensibles y procedimientos comprensibles** ...

Todo programa estructurado puede ser leído desde el principio al fin sin interrupciones en la secuencia normal de lectura.

Teorema de la estructura

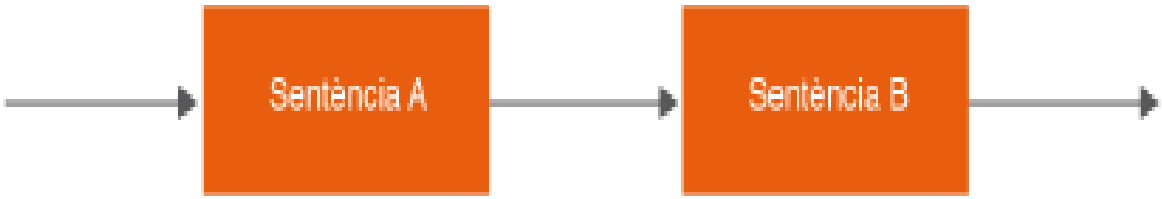
Demuestra que todo programa se puede escribir utilizando únicamente las tres estructuras básicas de control:

- Secuencia
- Selección
- Iteración

Características de la programación estructurada

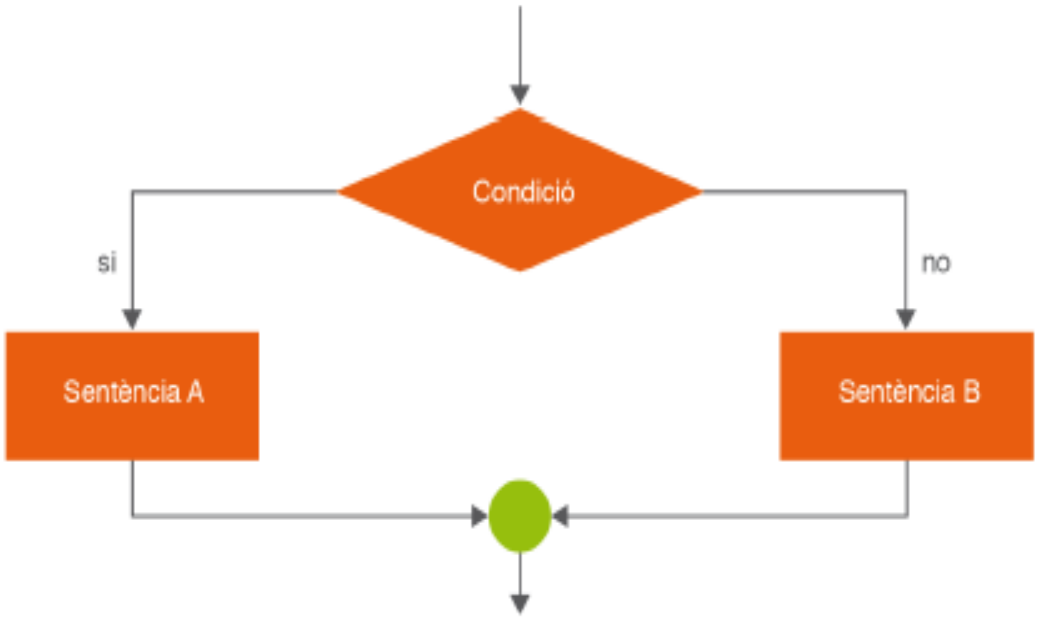
Secuencia: instrucciones ejecutadas sucesivamente, una tras otra.

FIGURA 1.7. Exemple de seqüència



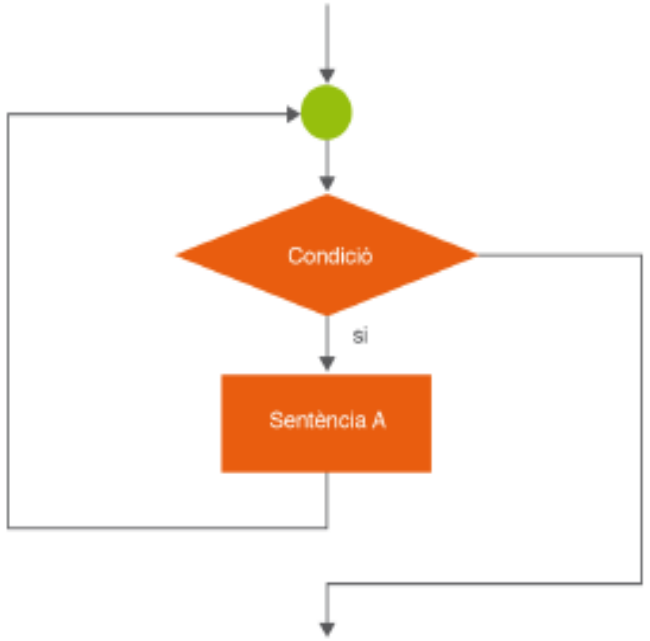
Selección: la instrucción condicional con doble alternativa, de la forma "Si condición, entonces SentenciaA, sino SentenciaB".

FIGURA 1.8. Exemple de selecció



Iteración: el bucle condicional "mientras condición, haz sentencias", que ejecuta las instrucciones repetidamente mientras la condición se cumpla.

FIGURA 1.9. Exemple d'iteració

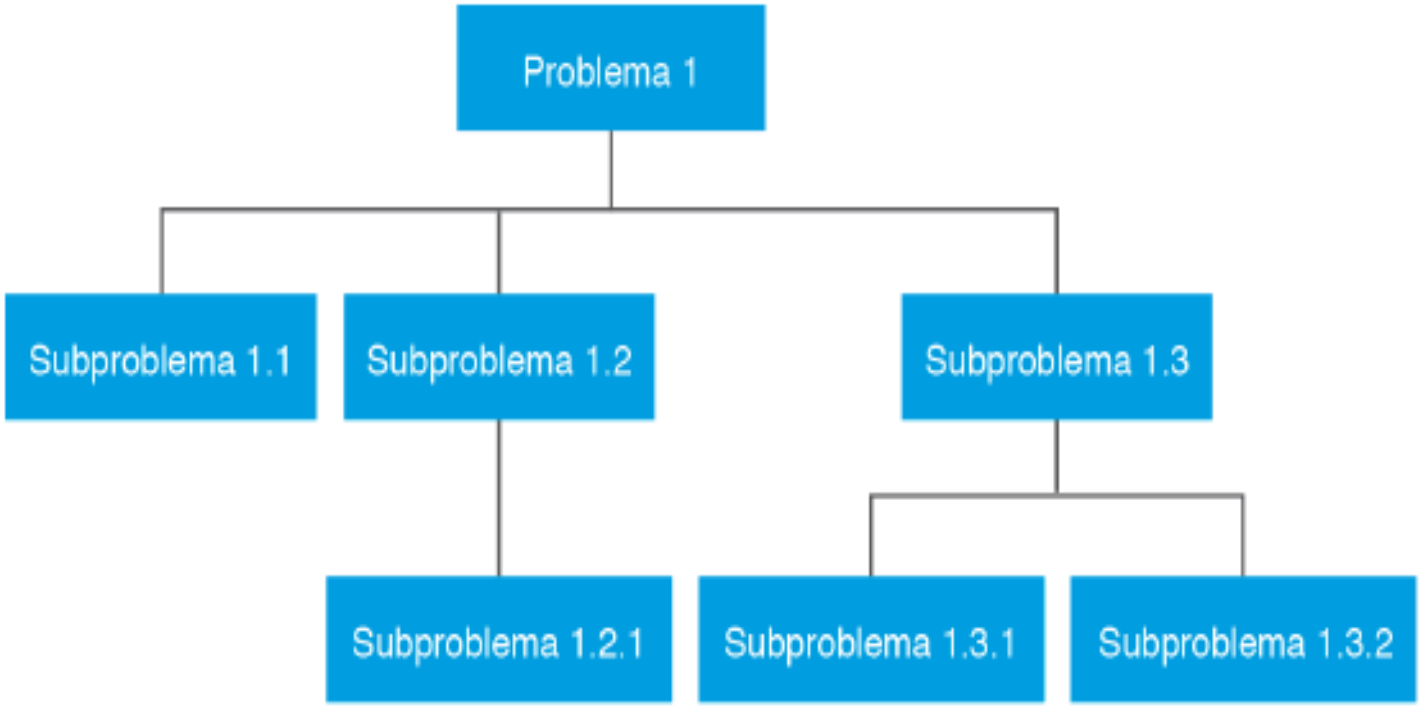


Características de la programación estructurada

Por otra parte, también se propone desarrollar el programa con la técnica de **diseño descendente** (top-down). Es decir, **modular el programa creando porciones más pequeñas de programas con tareas específicas**, que se subdividen en otros subprogramas, cada vez más pequeños. La idea es que estos subprogramas típicamente llamados funciones o procedimientos deben resolver un único objetivo o tarea.

Características de la programación estructurada

FIGURA 1.10. Disseny descendent



Características de la programación estructurada

La visión moderna de la programación estructurada introduce las características de **programación modular y tipos abstractos de datos (TAD)**.

Programación modular

La programación modular está basada en la filosofía del diseño descendente, donde cada subproblema corresponde con un módulo que se resuelve de manera independiente

Tipos Abstractos de Datos (TAD)

El programador puede definir un nuevo tipo de datos y sus posibles operaciones

El **paradigma de objetos**, típicamente conocido como Programación Orientada a Objetos (**POO**, o OOP en inglés), es un paradigma de construcción de programas **basado en una abstracción del mundo real**.

En un programa orientado a objetos, **la abstracción** no **son** procedimientos ni funciones sino los **objetos**. Estos objetos son una representación directa de algo del mundo real, como un libro, una persona, un pedido, un empleado ...

Un **objeto** es una combinación de datos (llamadas **atributos**) y **métodos** (funciones y procedimientos) que nos permiten interactuar con él. En este tipo de programación, por lo tanto, los programas son conjuntos de objetos que interactúan entre ellos a través de mensajes (llamadas a métodos).

Algunos de los lenguajes **POO** son **C ++, Java, C #** ...

Características de la POO

La programación orientada a objetos se basa en la integración de 5 conceptos: **abstracción, encapsulación, modularidad, jerarquía y polimorfismo**

Abstracción

Es el proceso en el que se separan las propiedades más importantes de un objeto de las que no lo son. Es decir, por medio de la abstracción **se definen las características esenciales de un objeto del mundo real, los atributos y comportamientos que lo definen como tal, para luego modelar en un objeto de software**. En el proceso de abstracción no debe ser preocupante la implementación de cada método o atributo, basta definirlos.

Abstracción

En la tecnología orientada a objetos la herramienta principal para soportar la abstracción es la **clase**. Se puede definir una clase como una descripción genérica de un grupo de objetos que comparten características comunes, las cuales son especificadas en sus atributos y comportamientos.

Encapsulación

Permite a los objetos elegir qué información es publicada y qué información es escondida en el resto de los objetos. Por eso los objetos suelen presentar sus **métodos** como interfaces **públicas** y sus **atributos** como datos **privados o protegidas**, siendo **inaccesibles desde otros objetos.**

Encapsulación

Las características que se pueden otorgar son:

- **Público**: **cualquier** clase puede acceder a cualquier atributo o método declarado como público y utilizarlo.
- **Protegido**: cualquier clase **heredada** puede acceder a cualquier atributo o método declarado como protegido a la clase madre y utilizarlo.
- **Privado**: **ninguna** clase puede acceder a un atributo o método declarado como privado y utilizarlo.

Modularidad

Permite modificar las características de cada una de las clases que definen un objeto, de forma independiente del resto de clases en la aplicación. En otras palabras, si una aplicación se puede dividir en módulos separados, normalmente clases, y estos módulos se pueden compilar y modificar sin afectar a los demás, entonces esta aplicación ha sido implementada en un lenguaje de programación que soporta la modularidad.

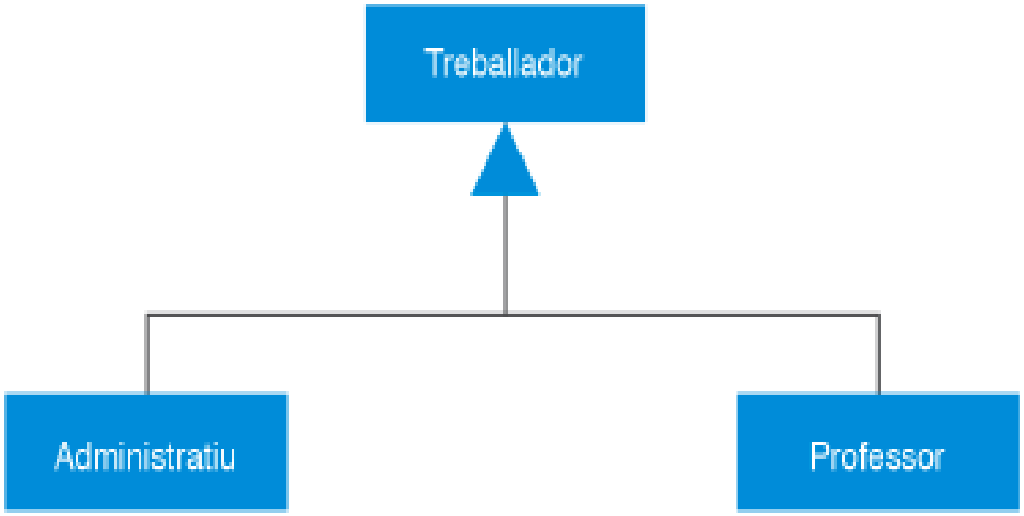
Jerarquía

Permite la ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son la **herencia** y la **agregación**.

Jerarquía

La **herencia** también se puede ver como una forma de compartir código, por lo que cuando se utiliza la herencia para definir una nueva clase **sólo se debe añadir lo que sea diferente**, es decir, reaprovecha los métodos y variables, y especializa el comportamiento.

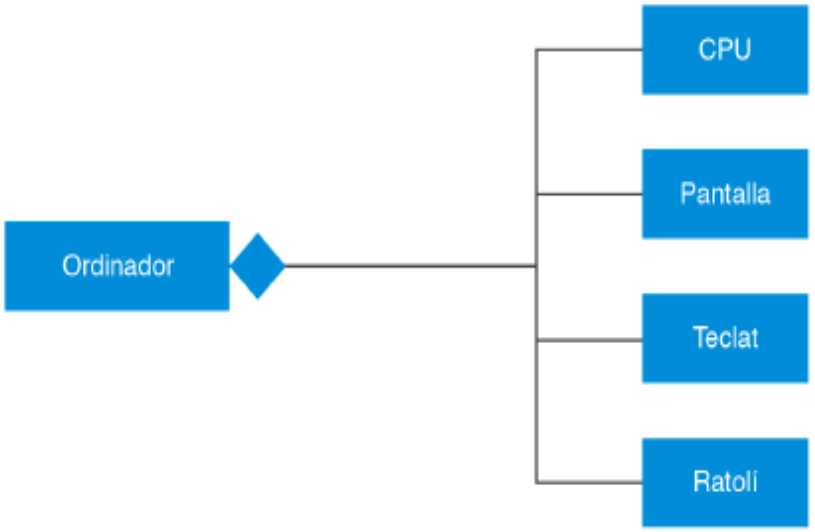
FIGURA 1.11. Exemple d'herència



Jerarquía

La **agregación** es un objeto que está formado de la combinación de otros objetos o componentes.

FIGURA 1.12. Exemple d'agregació



El polimorfismo

Es una característica que permite dar diferentes formas a un método, ya sea en la definición como en la implementación.

La **sobrecarga (overload)** de métodos consiste en implementar varias veces un mismo método pero con parámetros diferentes, de modo que, en invocarlo, el compilador decide cuál de los métodos se debe ejecutar, en función de los parámetros de la llamada.

- `print(Object o)`
- `print(String string)`
- `print(boolean bln)`
- `print(char c)`
- `print(char[] chars)`
- `print(double d)`
- `print(float f)`
- `print(int i)`
- `print(long l)`

```
49          System.out.
```

El polimorfismo

La **sobreescritura (override)** de métodos consiste en reimplementar un método heredado de una superclase exactamente con la misma definición (incluyendo nombre de método, parámetros y valor de retorno).

```
@Override
public void paintComponent(Graphics g) {
    for (int i = 0; i < DIMENSION; i++) {
        for (int j = 0; j < DIMENSION; j++) {
            t[i][j].paintComponent(g);
```

Característiques de la POO

```
1  class Treballador {
2     private:
3        string nom;
4        string DNI;
5     protected:
6        static const float SOU_BASE = 1.000;
7     public:
8        string GetNom() {return this.nom;}
9        void SetNom (string n) {this.nom = n;}
10       string GetDNI() {return this.DNI;}
11       void SetDNI (string dni) {this.DNI = dni;}
12       virtual float salari() = 0;
13 }
14
15 class Administratiu: public Treballador {
16    public:
17       float Salari() {return SOU_BASE * 10};
18 }
19
20 class Professor: public Treballador {
21    private:
22       int numHores;
23    public:
24       float Salari() {return SOU_BASE + (numHores * 15);}
25 }
```

```
1  class Treballador {
2     private:
3        string nom;
4        string DNI;
5     protected:
6        static const float SOU_BASE = 1.000;
7     public:
8        string GetNom() {return this.nom;}
9        void SetNom (string n) {this.nom = n;}
10       string GetDNI() {return this.DNI;}
11       void SetDNI (string dni) {this.DNI = dni;}
12       virtual float salari() = 0;
13 }
14
15 class Administratiu: public Treballador {
16    public:
17       float Salari() {return SOU_BASE * 10};
18 }
19
20 class Professor: public Treballador {
21    private:
22       int numHores;
23    public:
24       float Salari() {return SOU_BASE + (numHores * 15);}
25 }
```

CLASES

Atributos privados

Atributos que se heredan

Métodos públicos

El **paradigma funcional** está basado en un modelo matemático. La idea es que el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produzca el resultado deseado.

Los creadores de los primeros **lenguajes funcionales** pretendían convertirlos en lenguajes de uso universal para el procesamiento de datos en todo tipo de aplicaciones, pero, con el paso del tiempo, se ha utilizado **principalmente en ámbitos de investigación científica y aplicaciones matemáticas.**

Uno de los lenguajes más típicos del paradigma funcional es el **Lisp**

```
> (defun factorial (n)
(if (= n 0)
1
(* n (factorial (- n 1)))))
FACTORIAL
> (factorial 3)
6
```

El **paradigma lógico** tiene como característica principal la aplicación de las reglas de la lógica para inferir conclusiones a partir de datos.

Uno de los lenguajes más típicos del paradigma lógico es el **Prolog**.

Un **programa lógico contiene una base de conocimiento sobre la que se llevan a cabo consultas**. La base de conocimiento está formada por hechos, que representan la información del sistema expresada como relaciones entre los datos y reglas lógicas que permiten deducir consecuencias a partir de combinaciones entre los hechos y, en general, otras reglas.

El **paradigma lógico** es ampliamente utilizado en las aplicaciones que tienen que ver con la **Inteligencia Artificial**, particularmente en el campo de sistemas expertos y procesamiento del lenguaje humano.

Un sistema experto es un programa que imita el comportamiento de un experto humano. Por lo tanto contiene información (es decir una base de conocimientos) y una herramienta para comprender las preguntas y encontrar la respuesta correcta examinando la base de datos (un motor de inferencia).