Unidad Didáctica 1

Tema 1

Apuntes para apoyar el estudio

Fundamentos de Informática

Equipo docente de la asignatura:

M.F. Verdejo

E. Amigó

V. Fresno

R. Centeno

Tabla de contenido

Tema 1- Introducción

1.1 ¿qué es un sistema de computación?

El origen del computador se debe a la necesidad de realizar cálculos de forma automática. Sin embargo, el procesamiento numérico no es su única utilidad. La posibilidad de realizar operaciones lógicas le dota de la capacidad de ser usado para el procesamiento general de información. La informática, es precisamente el cuerpo de conocimiento que se encarga de todo lo relacionado con el desarrollo y uso de los sistemas basados en computador.

Todos hemos escuchado alguna vez la frase que dice que vivimos en la sociedad de la información. La tecnología afecta nuestra vida diaria, en casa, en el trabajo, en los centros educativos, en nuestras relaciones sociales, en las actividades de ocio y negocio.... Las nuevas tecnologías hacen posible que tengamos disponible una gran cantidad de información a nuestra disposición y además nos facilitan los medios de comunicarnos, de trabajar de forma colaborativa, así como la automatización de una enorme

variedad de procesos.

Por mencionar algunos aspectos, los computadores nos permiten:

- Mantener un sofisticado sistema de seguridad en nuestro domicilio
- Encontrar todo tipo de información multimedia a través de Internet y compartir documentos, fotos, videos y opiniones con otros, en redes sociales, portales temáticos o generalistas.
- Organizar nuestra información personal, y poder acceder a ella desde cualquier punto en el que nos encontremos.
- Comunicarnos rápida y fácilmente con otras personas, a través de correo electrónico, videoconferencia, o telefonía IP.
- Comprar entradas para espectáculos, obtener billetes y reservar hoteles, así como una gama cada vez más amplia de productos.
- Realizar tramites como ciudadanos, contribuyentes o estudiantes

Por ello podemos afirmar que todos nosotros somos usuarios. De aquí que nuestra primera definición, nos incluya.

1.1.1 Definición de sistema de computación

Un sistema de computación está formado por cuatro partes: el hardware, el software, los datos, y los usuarios.

1.1.2 Tipos de sistemas atendiendo a los usuarios

Sin duda el sistema de computación más común es el basado en el **Computador Personal**, el cual puede tener la forma de un portátil como el de la figura 1, o el de un computador de sobremesa como el de la figura 3, aunque también hay más variedad como por ejemplo tabletas (figura 2) , PDAs, etc.

Figura 1: un usuario interaccionando con un portátil

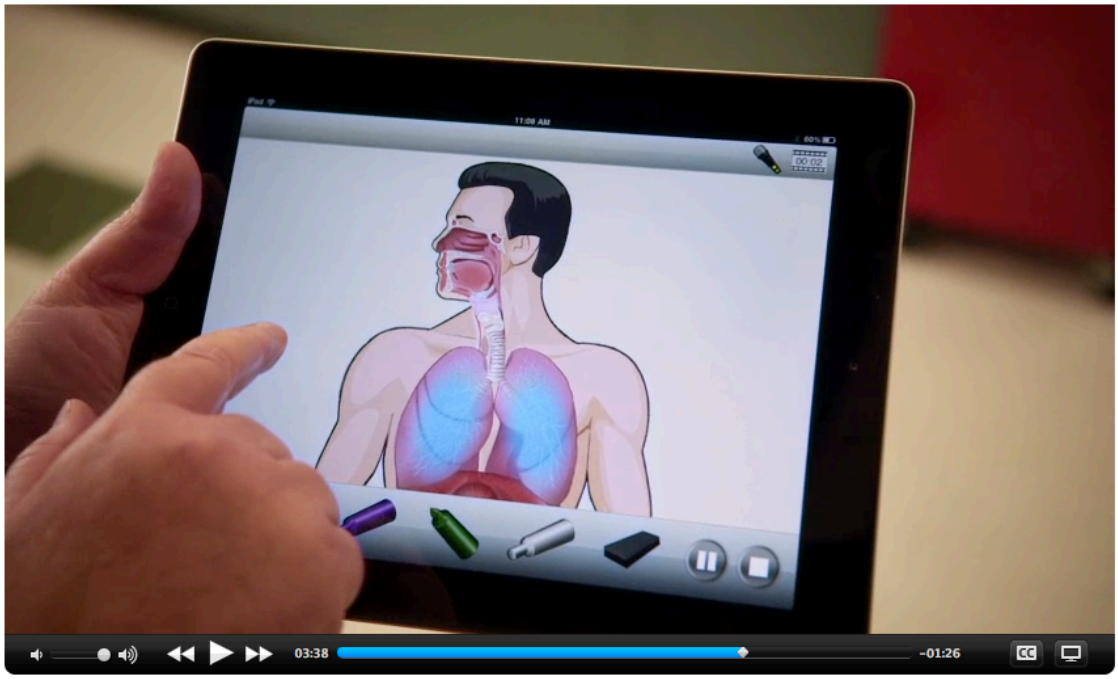


Figura 2: un usuario interaccionando con una aplicación educativa, en una tableta

Figura 3: un usuario interaccionando con un computador de sobremesa

En un siguiente nivel, tenemos las estaciones de trabajo, con computadores más potentes, y que como en el caso de la figura 4, está siendo utilizada

para captar el movimiento real de una persona, e incorporarlo a un personaje virtual.

Figura 4: captando el movimiento de una persona para incorporarlo en un personaje virtual.

Y en un nivel de más potencia, y atendiendo a comunidades de usuarios, tenemos las configuraciones de supercomputadores, como el de la figura 5, con capacidad para modelar fenómenos complejos, por ejemplo la evolución del clima terrestre , la secuenciación del ADN, o dar soporte a los buscadores para indexar y encontrar el enorme volumen de la información en Internet.

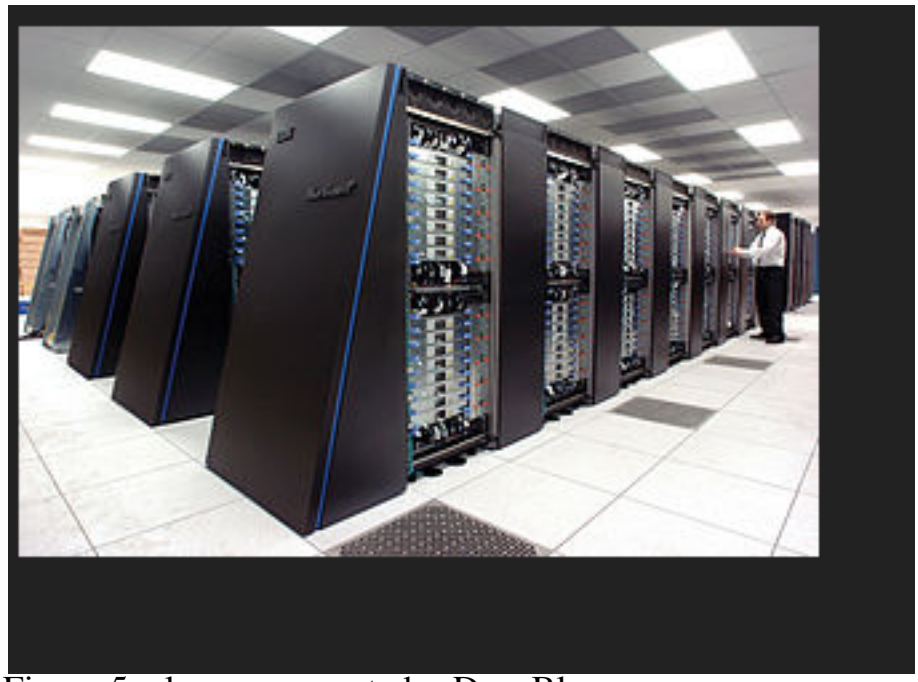


Figura 5: el supercomputador DeepBlue

En esta dirección se puede encontrar mas información sobre los diez supercomputadores más potentes (en noviembre de 2011)

http://www.gizmocrazed.com/2011/11/top-10-supercomputers-in-the-world-november-2011/

En España hay varios, uno de los mas conocidos es el *Marenostrum* del Barcelona Supercomputing Center
http://www.bsc.es/

y aquí pueden verse, algunos de los centros de Google, las granjas de servidores que permanentemente monitorizan e indexan la información en Internet.

http://googleespana.blogspot.com.es/search/label/Centros%20de%20datos

Veamos a continuación, en qué consiste cada uno de los elementos de un sistema de computación.

1.1.3 Elementos

1.1.3.1 Hardware

El hardware son los dispositivos electrónicos. La figura 6 es una muestra representativa del aspecto de un PC con la carcasa abierta.
Un sistema basado en PC (Personal Computer) tiene cuatro tipos de componentes básicos

1. Entrada : Teclado, Ratón, Micrófono
2. Salida: Monitor, Altavoces, Impresora
3. Procesamiento: Microprocesador
4. Almacenamiento
   - Temporal: Memoria RAM (Random Access Memory)
   - Permanente: disco duro, CD, DVD...

Figura 6: un computador con la carcasa abierta

Los elementos principales del hardware son el procesador y la memoria, que se encuentran insertados en lo que se llama la placa madre, y tiene el aspecto típico que se muestra en la Figura 7

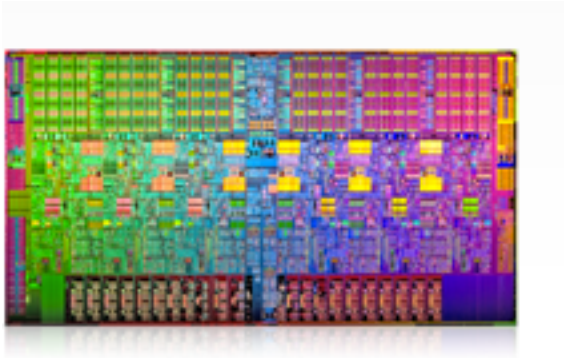


Figura 7: placa madre de uno de los modelos portátiles de Apple.

ACTIVIDAD- En el curso virtual de esta asignatura, en el material de estudio interactivo, se puede encontrar una presentación multimedia de las partes de un computador de sobremesa.

Cualquier computador ofrece al usuario información acerca de su hardware, por ejemplo para el portátil que se muestra en la Figura 8

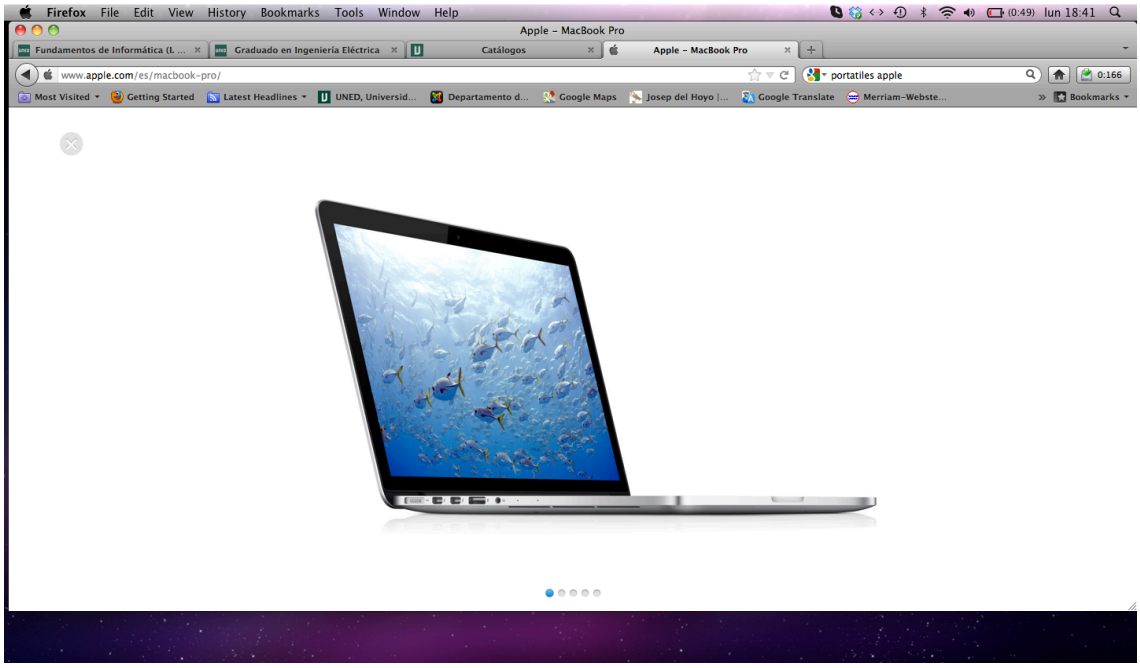


Figura 8: un portátil de la familia Apple

Si se consulta mediante la selección del menú (acerca de este mac) qué hardware tiene se obtiene la descripción de la figura 9

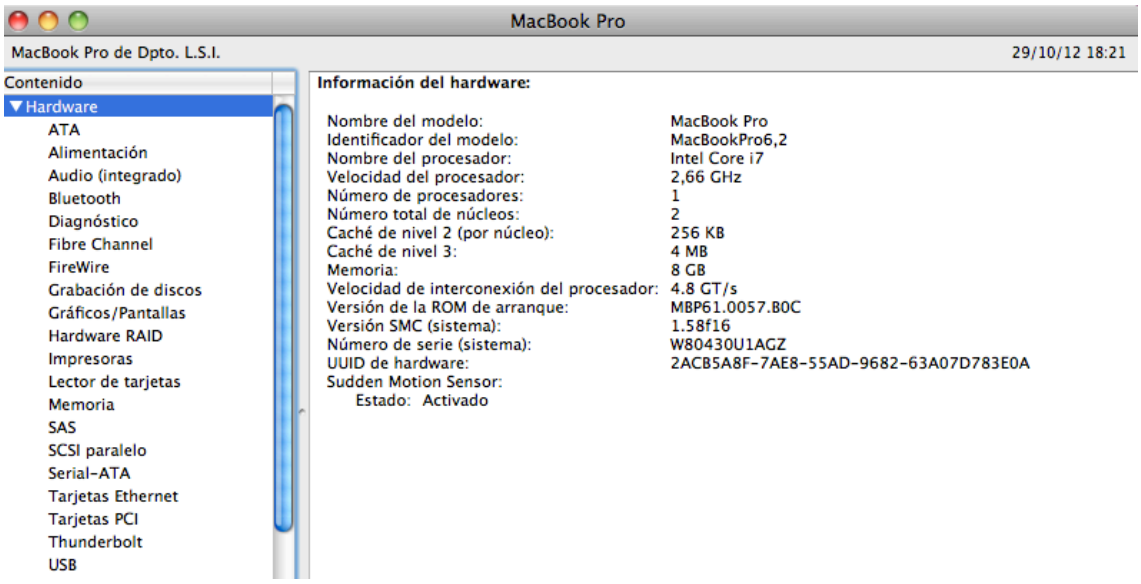


Figura 9: información del hardware de un determinado portátil

1.1.3.2 Software

El software consiste en la parte lógica, las instrucciones que hacen funcionar el computador. Hay dos tipos básicos de software llamados software del sistema y software de aplicaciones respectivamente.

La información tanto de uno como del otro, es también accesible al usuario, mediante una opción de menú. La figura 9 muestra el software del sistema instalado en el portátil anterior.

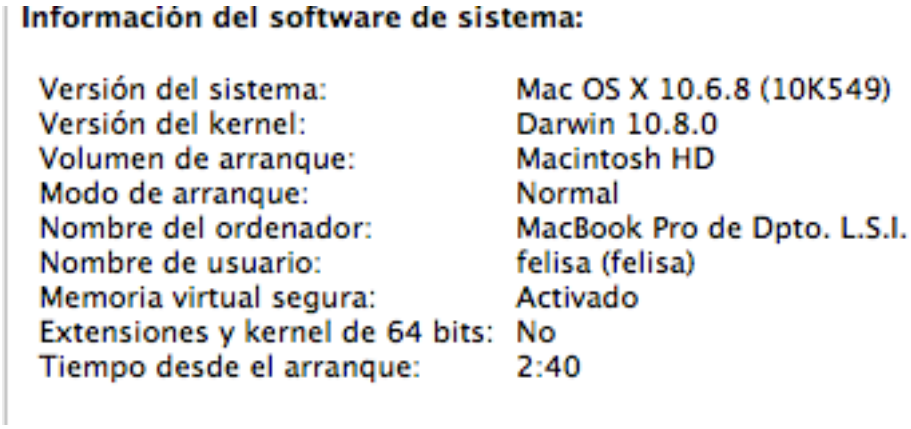
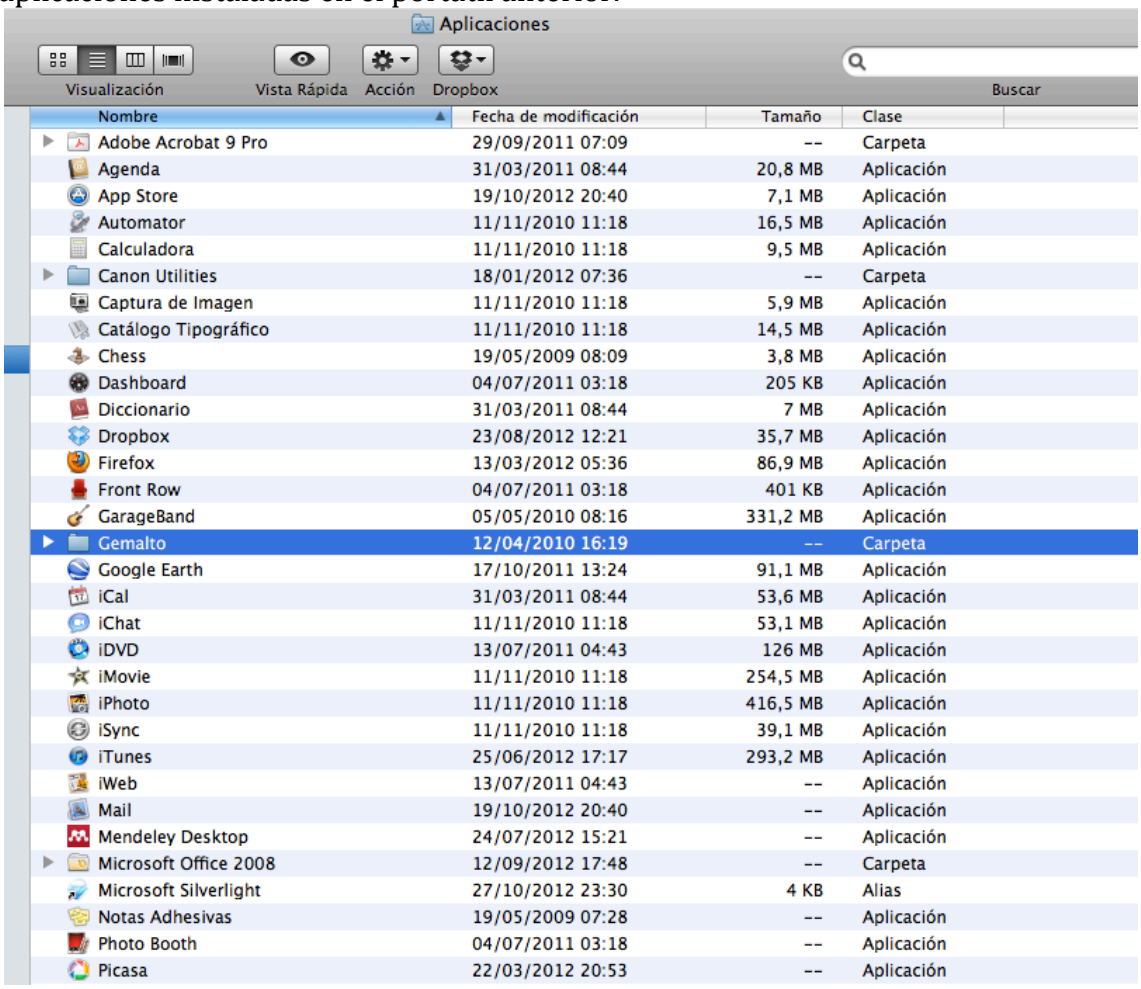


Figura 10: información del software de un determinado portátil

Las aplicaciones instaladas , dependen de las necesidades de los usuarios: navegadores, editores de texto, correo electrónico, … y la información sobre las

mismas también es accesible, en este caso la figura 11 nos muestra una parte de la aplicaciones instaladas en el portátil anterior.



Nombre	Fecha de modificación	Tamaño	Clase
Adobe Acrobat 9 Pro	29/09/2011 07:09	--	Carpeta
Agenda	31/03/2011 08:44	20,8 MB	Aplicación
App Store	19/10/2012 20:40	7,1 MB	Aplicación
Automator	11/11/2010 11:18	16,5 MB	Aplicación
Calculadora	11/11/2010 11:18	9,5 MB	Aplicación
Canon Utilities	18/01/2012 07:36	--	Carpeta
Captura de Imagen	11/11/2010 11:18	5,9 MB	Aplicación
Catálogo Tipográfico	11/11/2010 11:18	14,5 MB	Aplicación
Chess	19/05/2009 08:09	3,8 MB	Aplicación
Dashboard	04/07/2011 03:18	205 KB	Aplicación
Diccionario	31/03/2011 08:44	7 MB	Aplicación
Dropbox	23/08/2012 12:21	35,7 MB	Aplicación
Firefox	13/03/2012 05:36	86,9 MB	Aplicación
Front Row	04/07/2011 03:18	401 KB	Aplicación
GarageBand	05/05/2010 08:16	331,2 MB	Aplicación
Gemalto	12/04/2010 16:19	--	Carpeta
Google Earth	17/10/2011 13:24	91,1 MB	Aplicación
iCal	31/03/2011 08:44	53,6 MB	Aplicación
iChat	11/11/2010 11:18	53,1 MB	Aplicación
iDVD	13/07/2011 04:43	126 MB	Aplicación
iMovie	11/11/2010 11:18	254,5 MB	Aplicación
iPhoto	11/11/2010 11:18	416,5 MB	Aplicación
iSync	11/11/2010 11:18	39,1 MB	Aplicación
iTunes	25/06/2012 17:17	293,2 MB	Aplicación
iWeb	13/07/2011 04:43	--	Aplicación
Mail	19/10/2012 20:40	--	Aplicación
Mendeley Desktop	24/07/2012 15:21	--	Aplicación
Microsoft Office 2008	12/09/2012 17:48	--	Carpeta
Microsoft Silverlight	27/10/2012 23:30	4 KB	Alias
Notas Adhesivas	19/05/2009 07:28	--	Aplicación
Photo Booth	04/07/2011 03:18	--	Aplicación
Picasa	22/03/2012 20:53	--	Aplicación

Figura 11: visualización de las aplicaciones instaladas

Actividad: clasificar los diferentes dispositivos que aparecen en la siguiente figura:

Figura 12: dispositivos

1.1.3.3 Datos

Los datos, son la información en forma textual, numérica, imágenes, sonidos, señales... que se procesan en una computadora. En la figura 13 se muestra un

escenario de captura de datos, a través de sensores que captan actividad cerebral en un experimento interactivo.

Figura 13: recogiendo datos en un experimento

En la figura 14, un grupo de usuarios trabaja con la visualización de un conjunto de datos epidemiológicos, que se presentan geolocalizados.

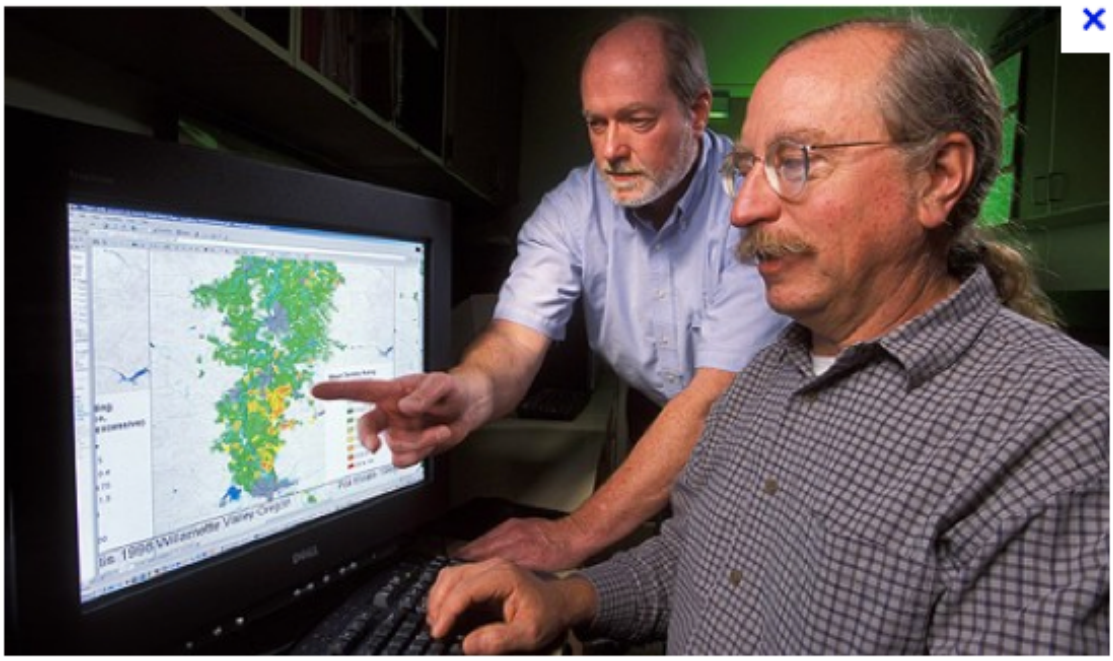


Figura 14

La figura 15 muestra algunos de los diferentes formatos de datos, generados con distintas aplicaciones, que puede procesar un computador : por ejemplo documentos (doc, pdf, txt, ..), imágenes (jpg), música (mp3), ect

Figura 15: diferentes formatos de archivos de datos

El ciclo de procesamiento de datos incluye entrada, procesamiento, salida, y almacenamiento, tal y como se ilustra en la figura 16

Figura 16: ciclo de procesamiento de datos

Vamos a verlo con un ejemplo, la imagen de la figura 17 muestra el catalogo de la Biblioteca de la Uned. Hay una caja en donde se puede expresar lo que se desea

buscar, utilizando por ejemplo palabras clave.

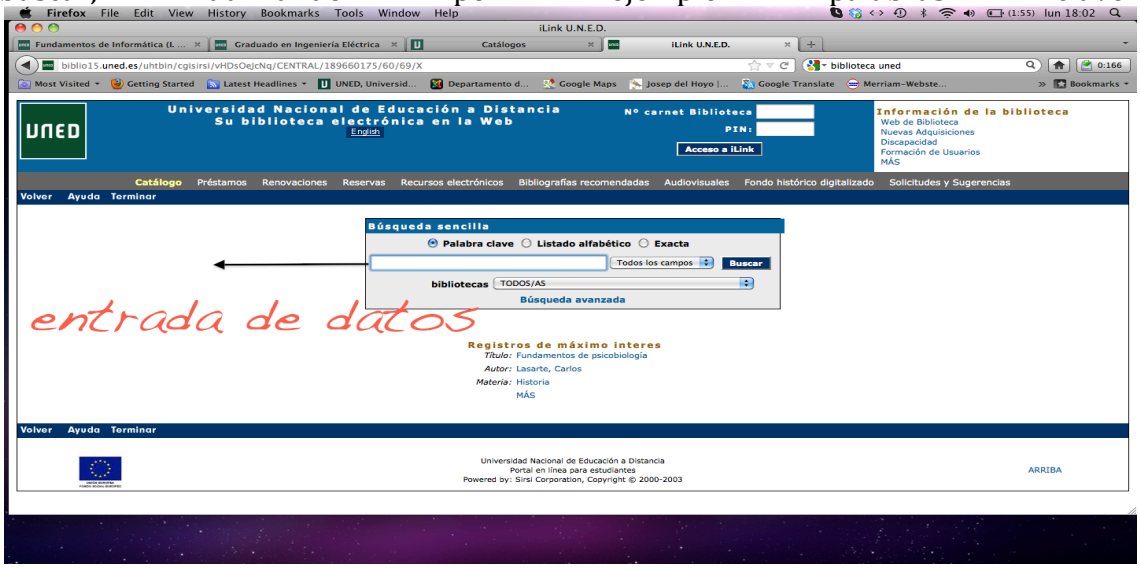


Figura 17: ventana para entrada de datos

En esta caja la entrada de datos que proporcionará el usuario de forma interactiva, se indican mediante los botones de selección y el cuadro en blanco a rellenar.

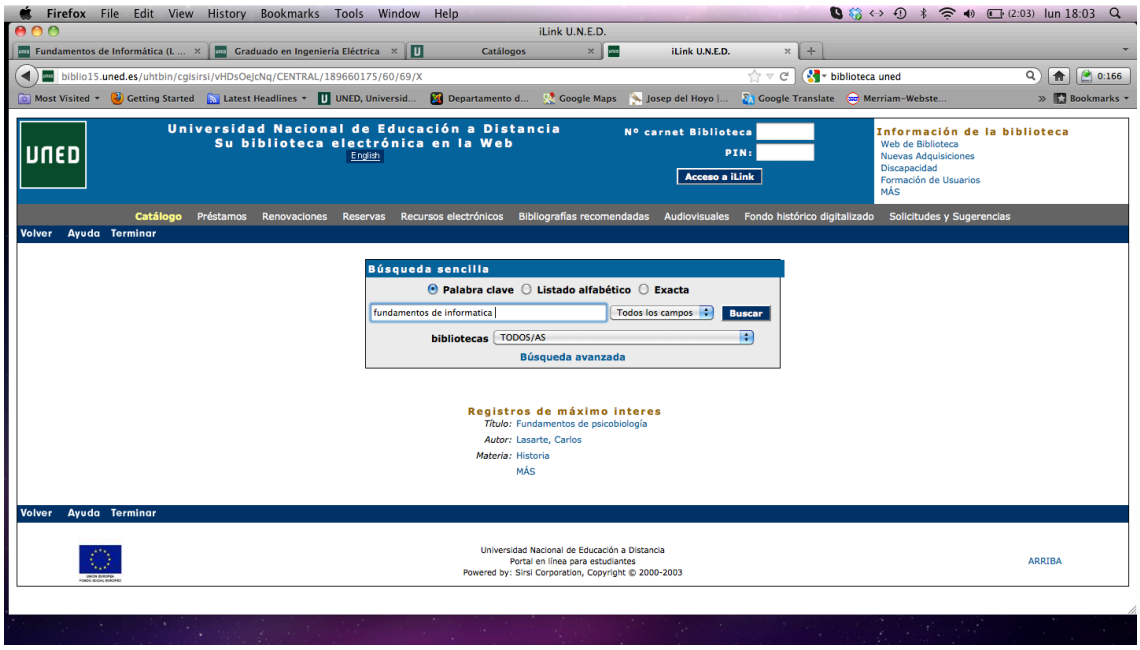


Figura 18: ventana mostrando los datos de búsqueda introducidos por el usuario

Una vez completados los datos, tal y como aparecen en la figura 18, el proceso, (búsqueda de documentos que responden a los datos de entrada) se ejecutará al pulsar sobre el botón de búsqueda señalado en la figura 19.

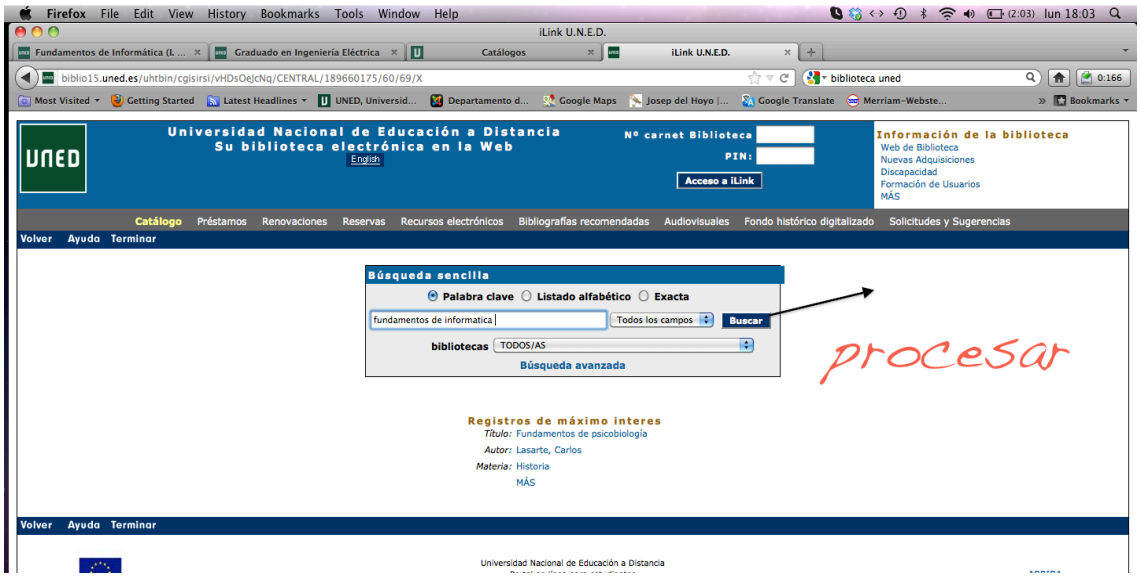


Figura 19: el botón de buscar, es el que inicia el proceso

Y en nuestro ejemplo de búsqueda, obtenemos una salida, que consiste en una lista de documentos, que se muestran en pantalla, parte de ellos aparecen en la figura 20

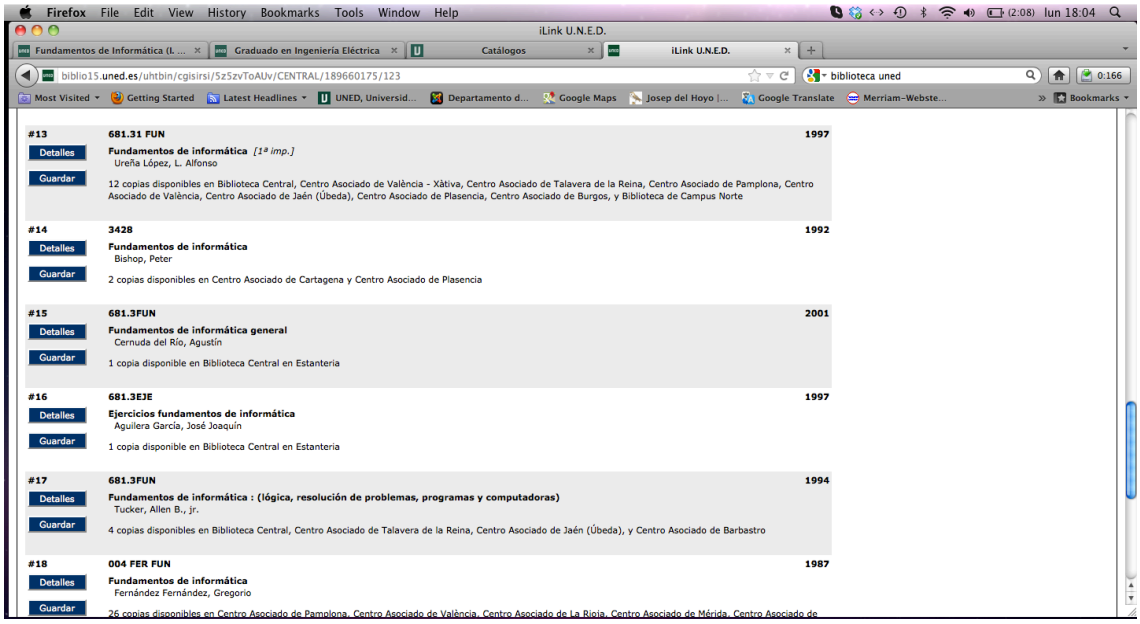


Figura 20: resultado de la búsqueda en el catálogo de la biblioteca

También existe la posibilidad de Almacenar nuestros resultados, bien para proseguir refinando nuestra búsqueda, o para realizar otra operación con los resultados, por ejemplo imprimirlos o enviar un correo electrónico, como se muestra en la figura 21.

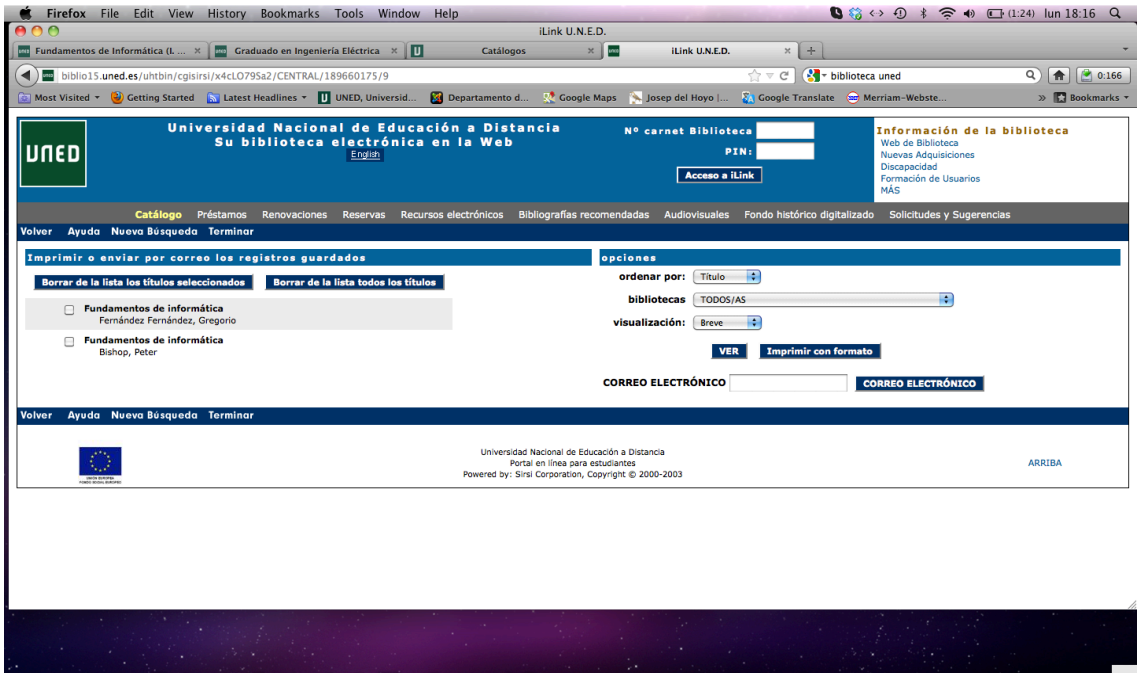


Figura 21 : diversas opciones para almacenar/tratar los datos de salida

1.1.3.4 Los usuarios

Podemos clasificarlos en los siguientes grupos

1. Los profesionales informáticos que diseñan, y programan software del sistema o aplicaciones.
2. Los profesionales que usan el computador como herramienta de trabajo, y que diseñan, adaptan o modifican sus propias aplicaciones o servicios.
3. Los profesionales que usan el computador como herramienta de trabajo, normalmente utilizando aplicaciones existentes.
4. Las personas que usan la computadora como herramienta personal de trabajo, de comunicación, de ocio…

Actividad

A continuación podrás ver varias fotos que ilustran la clasificación anterior

¿en qué grupo las clasificarías?
¿qué ejemplos puedes proporcionar para cada una de las clases?

Figura 22 : un equipo médico inspeccionando resultados de diferentes pruebas ecográficas y radiológicas

Figura 23 a: un paleontólogo creando una simulación del movimiento de un dinosaurio.

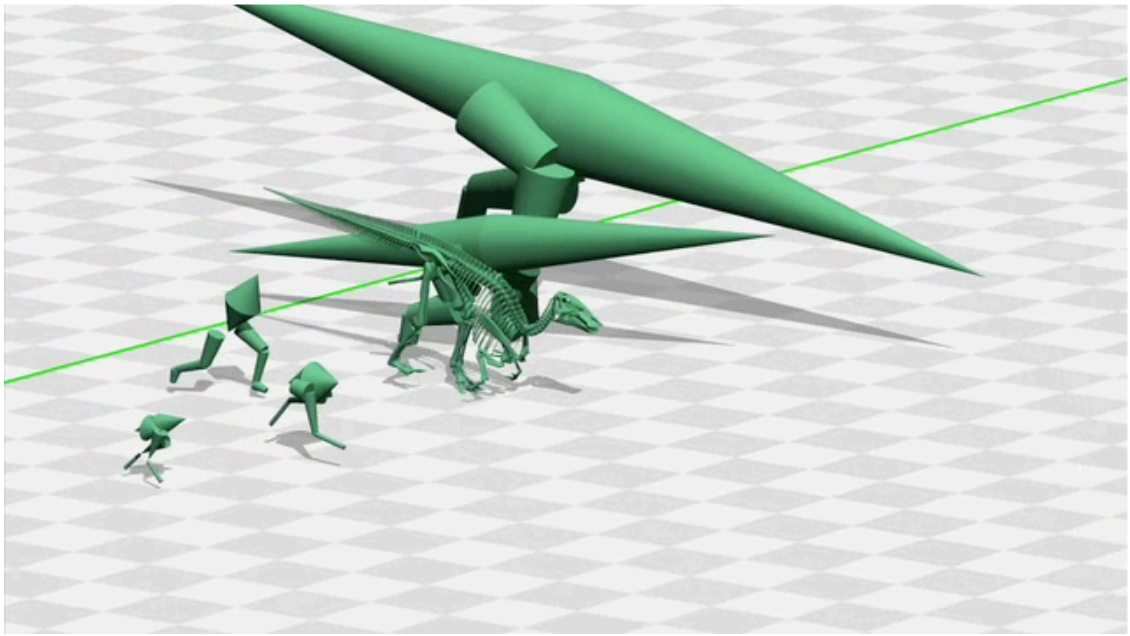


Figura 23 b: simulando el movimiento de un dinosaurio

Figura 24 a: un alumno de odontología practicando un tipo de intervención

Figura 24 b: monitorizando diversas constantes durante una practica de intervención

Todos los usuarios, también los de los grupos 2 y 3, deben de ser capaces de realizar una variedad de tareas como por ejemplo en un computador personal:

1. Configurar el sistema

   En la siguiente figura se muestra el panel de preferencias del sistema en un Mac, en donde se pueden especificar para cada uno de los elementos, las diferentes opciones de configuración.

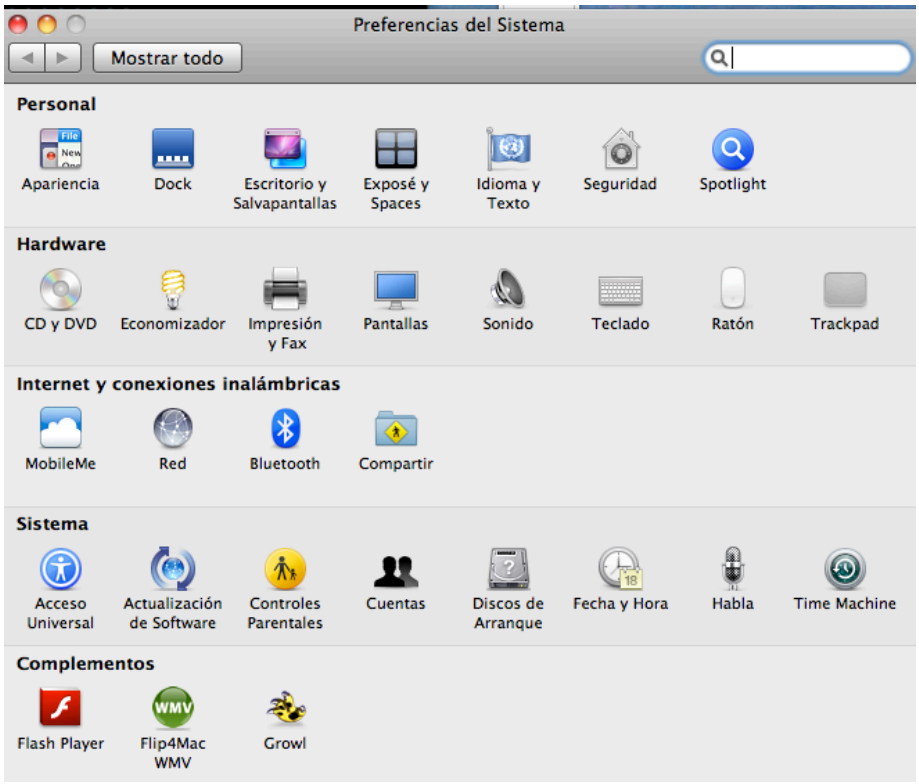


Figura 25: panel de configuración

2. Instalar software

Por ejemplo, para realizar las prácticas de esta asignatura , hay que instalar la maquina virtual de Java, que se hace desde el entorno de la asignatura, en donde existe además un pequeño tutorial de ayuda.

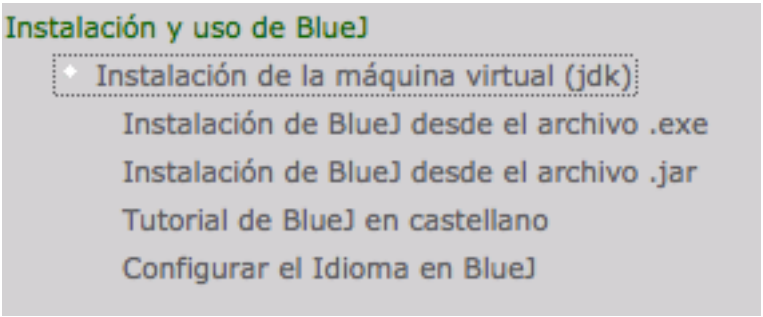


Figura 26: fragmento de la página de la práctica del curso virtual

3. Ejecutar programas

La siguiente figura nos muestra el aviso que nos aparece en la pantalla de la computadora cuando se descarga un archivo que es ejecutable, (.exe)

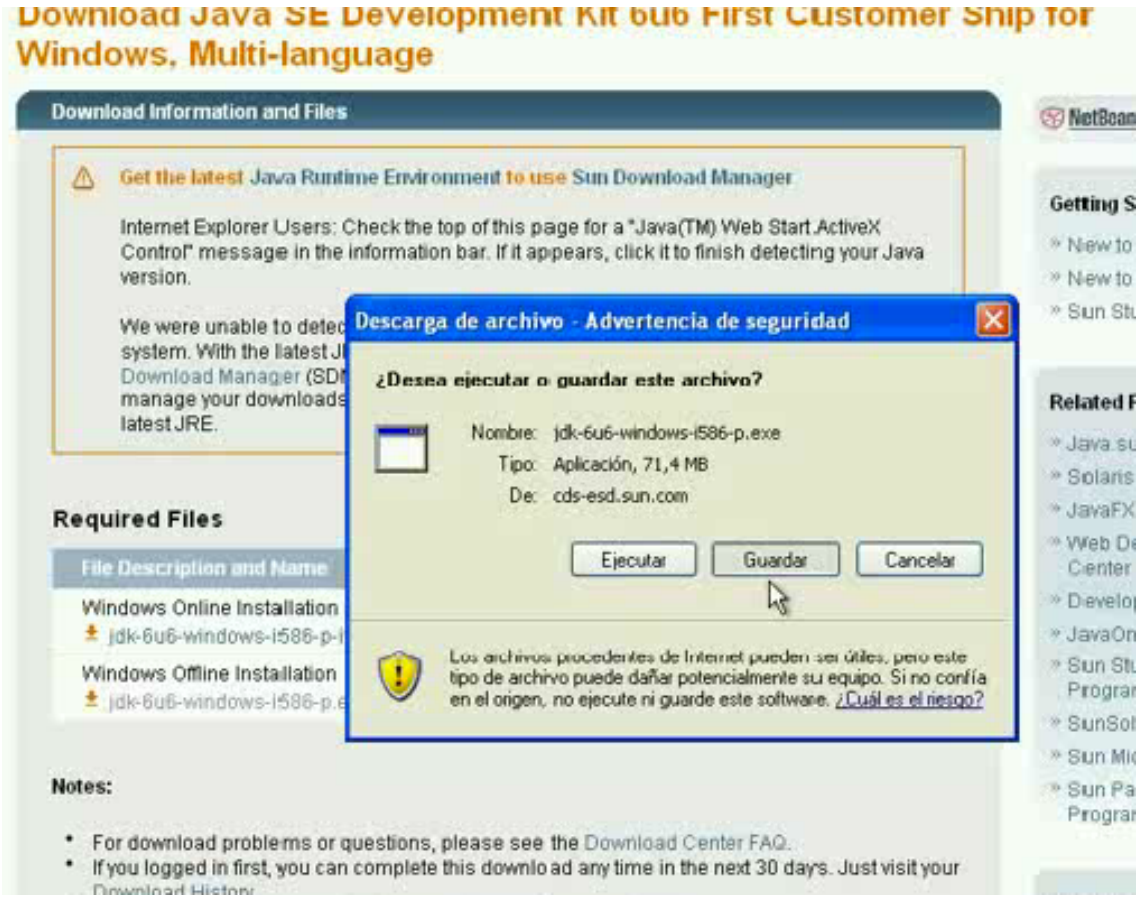


Figura 27: avisos de advertencia relacionado con un ejecutable

4. Administrar archivos

5. Mantener el sistema

La siguiente pantalla es un ejemplo del aviso que nos alerta sobre actualizaciones de nuestras aplicaciones

Figura 28: aviso automático de actualizaciones pendientes

Al seleccionar *Mostar detalles* podemos ver la información precisa de las actualizaciones sugeridas

Figura 29: detalles sobre una actualización de software

Actividad

Probablemente usarás un computador del tipo ilustrado por la figura 30, (1) identifica los elementos que hemos descrito (2) consulta las instrucciones de prácticas del curso virtual, e instala el JDK y el BlueJ, el entorno de programación con el que vamos a trabajar en esta asignatura

Figura 30:PC de sobremesa

1.2 . Nociones básicas de Hardware y Software

1.2.1. Representación de la información y su almacenamiento

La información es un conjunto de datos procesados y organizados, por tanto, implica tanto un conjunto de datos como su interrelación. Para representar los datos existen dos formas de hacerlo: la representación analógica y la digital. En los computadores toda la información está almacenada digitalmente, desde los números al texto pasando por imágenes de vídeo o registros de audio.

Es importante conocer las siguientes *definiciones* (utilizando su denominación anglosajona):

bit
La unidad más pequeña de datos. Su valor es 0 ó 1. También se denomina dígito binario.
byte
Bloque compuesto por 8 bits. Generalmente, es el mínimo bloque de información que fluye por el computador. Por ejemplo, cada letra del lenguaje natural es representada por un byte.

Generalizando,
un **Kilobyte** =1,024 bytes.
un **Megabyte =** 1,048,576 bytes.
Un **Gigabyte** = 1024 Megabytes
Un **Terabyte** =1024 Gigabytes

Ejemplo de representación de número decimal en binario:
$52 = 110100 = 1*2^5+1*2^4+0*2^3+1*2^2+0*2^1+0*2^0$

Para más información sobre representación, se puede consultar la bibliografía complementaria de la asignatura.

1.2.2 Definir el término Hardware

Se denomina **hardware** o **soporte físico** al conjunto de elementos materiales que componen un computador. Hardware también son los componentes físicos de un computador tales como el disco duro, CD–ROM, DVD, disquetera (floppy), etc..

En dicho conjunto se incluyen los dispositivos electrónicos y electromecánicos, circuitos, cables, tarjetas, placa madre, armarios o cajas, periféricos de todo tipo y otros elementos físicos.

1.2.2.1 Hardware: componentes básicos

Dos son los componentes básicos para el procesamiento en el computador: (1) la unidad central de proceso o CPU y (2) la memoria, como se ilustra en la figura 31. Ambos se ubican en la tarjeta madre del computador.

Definiciones:

(1) Unidad central de procesamiento (CPU)

El procesador, también llamado CPU (Central Processing Unit), es el computador real, la ″inteligencia″ de un sistema de computación.
Los microchips utilizados – Intel Pentium, AMD Athlon, etc **–** realizan funciones básicas hasta millones de veces por segundo.
Tiene dos partes principales:

- La unidad de control (CU: Control Unit) y
- La unidad aritmético-lógica (ALU: Arithmetic-Logic Unit).

El procesador trabaja en contacto directo con la memoria. (ver figura 31)

Unidad de control (CU)

Funciones:
• leer e interpretar instrucciones de programa
• dirigir la operación de los componentes de procesado interno
• controlar el flujo de programa y los datos de entrada y salida en la RAM

Los programas se encuentran normalmente almacenados en un fichero en la memoria permanente (disco duro u otros discos externos). Cuando se arranca el programa, lo primero que se hace es pasarlo de la memoria permanente a la memoria RAM.

Al ejecutar un programa (secuencia de instrucciones), la primera de esas instrucciones se mueve de la memoria RAM a la unidad de control, donde se decodifica en el decodificador.

La unidad de control utiliza además registros (de acceso muy rápido) para almacenar información y facilitar el intercambio de datos entre la memoria RAM y el procesador.

- El registro de instrucción contiene la instrucción que está siendo ejecutada.
- El registro de programa contiene la dirección en la RAM de la próxima instrucción a ser ejecutada.
- Los registros de propósito general almacenan datos para su procesamiento inmediato.

Unidad aritmético-lógica (ALU)
Funciones:
• computaciones aritméticas (suma, resta, multiplicación y división)
• computaciones lógicas (comparaciones)
Los resultados se almacenan en el acumulador

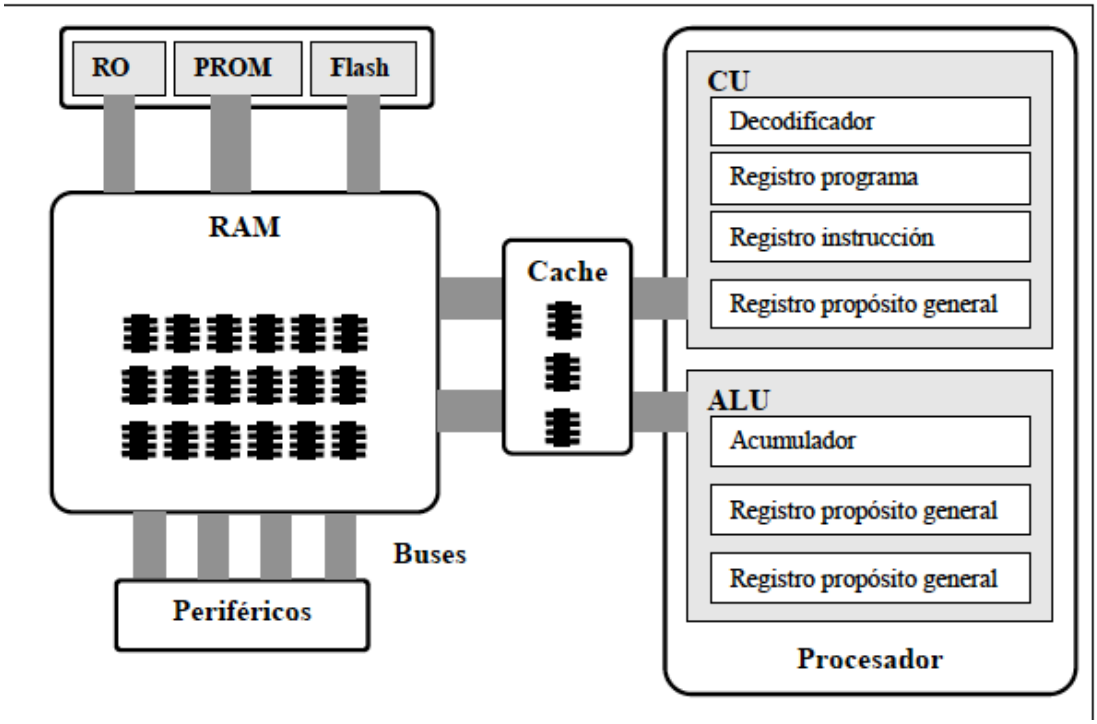


Figura 31: hardware, componentes básicos

(2) Memoria En la placa madre del computador está situado además del microchip, varios chips de memoria que componen la memoria primaria del computador (RAM, caché, etc).

- La memoria caché guarda la memoria "más inmediata" que utiliza el microchip, es decir, datos o instrucciones recientemente accedidos. Es más rápida que la RAM.

- La RAM (Random Access Memory) es una memoria de lectura y

escritura. Requiere de energía eléctrica para cargar datos. Los chips de la RAM tienen puntos cargados electrónicamente que representan un bit (unidad mínima de información, de valor 1 ó 0). Los bits se agrupan en bytes (conjunto de 8 bits). La información en la RAM se distribuye en direcciones: cada instrucción o dato de un programa está en una dirección. La memoria se accede mediante buses: el bus de direcciones (selecciona la posición de la que se va a leer/escribir) y el bus de datos (datos leídos/escritos). La memoria RAM **es volátil:** cuando se quita la corriente eléctrica, se pierden los datos. Los programas y los datos se transfieren a la RAM desde dispositivos de entrada o desde el disco duro. Cuando se deja de usar un programa, el espacio en memoria se asigna a otro.

Otro tipo de memorias

Memoria no volátil- almacenan datos incluso cuando el computador está apagado.

- **Memoria ROM** (Read-Only Memory) según las siglas en inglés. Los datos sólo se leen y no se modifican, por tanto se llama memoria de solo lectura. Suele utilizarse en el arranque del computador. Almacena información permanente, cómo por ejemplo, quién es el computador y que componentes tiene al arrancar (soy un Pentium y tengo un monitor LCD, grabadora de DVD, etc.) La ROM contiene un conjunto de instrucciones de inicio, conocidas como el sistema básico de entrada y salida (BIOS).

- Los chips que no pueden modificarse se conocen con el nombre de PROM (memoria programable de solo lectura), y se encuentran normalmente en las impresoras y las unidades de disco duro. Contienen las instrucciones que hacen funcionar esos dispositivos.

- La memoria flash es un tipo de memoria no volátil, que se utiliza en dispositivos digitales portátiles como por ejemplo las cámaras digitales o los llaveros USB

Velocidad de proceso de la CPU

Existen 3 conceptos básicos relacionados con la velocidad de funcionamiento de un procesador.

Ciclo de reloj- el periodo de la señal de reloj a la entrada del procesador. (frecuencia de reloj).

Los ciclos se miden en:

- Megahercio (MHz) = 1 millón, (10^6) ciclos por segundo
- Gigahercio (GHz) = Mil millones, (10^9) de ciclos por segundo

Ciclo de máquina- periodo de ejecución de una operación completa por el procesador. Suele ser múltiplo entero del ciclo de reloj.
Ciclo de instrucción- el tiempo que requiere el procesador para ejecutar una determinada instrucción. Suele especificarse para cada instrucción en función del número de ciclos de máquina. Una instrucción sencilla puede requerir solo un ciclo de máquina, pero una compleja decenas de ciclos.
El ciclo de ejecución de una instrucción se divide en dos fases principales:
1. Fase de búsqueda
    - Captura de la instrucción desde la RAM al registro de instrucción de la CU
    - Decodificación e interpretación de la instrucción
2. Fase de ejecución
    - Ejecución de la instrucción (normalmente usando la ALU)
    - Almacenamiento de resultados en memoria

Por ejemplo en un procesador Intel 8085, fabricado por Intel a mediados de los 70, estas dos fases son los ciclos de máquina. Cada fase es pues un ciclo máquina, y a su vez cada ciclo máquina son 4 ciclos de reloj, por lo que cada ciclo de instrucción real serían 8 ciclos de reloj.

Otros dispositivos de almacenamiento
Las unidades externas de almacenamiento, componen la llamada memoria secundaria, y son bien conocidos: discos duros, memorias de pequeño tamaño tipo flash, etc.

Periféricos de entrada
Son los que permiten que el usuario aporte información exterior. Entre ellos podemos encontrar: teclado, ratón (mouse), escáner, cámara digital, micrófono, SAI (Sistema de Alimentación Ininterrumpida), etc.

Periféricos de salida
Son los que muestran al usuario el resultado de las operaciones realizadas por el PC. En este grupo podemos encontrar: monitor, impresora, altavoces, etc.
Periféricos de comunicación
Son los dispositivos que pueden aportar simultáneamente información exterior al PC y al usuario. Aquí se encuentran: módem, tarjetas de red local, tarjetas de conexión inalámbrica,...

En el tema 2 se tratan con mas detalle estos elementos.

1.2.3 Software

Se denomina **software** (también **soporte lógico**) a todos los componentes intangibles de un computador, es decir, al conjunto de programas y procedimientos necesarios para hacer posible la realización de una tarea específica, en contraposición a los componentes físicos del sistema (hardware). Esto incluye aplicaciones informáticas tales como un procesador de textos, que permite al usuario confeccionar un documento, y software de sistema como un sistema operativo, que permite al resto de programas funcionar adecuadamente, facilitando la interacción con los componentes físicos y el resto de aplicaciones.
El software es intangible, existe como ideas, conceptos, símbolos, pero no tiene sustancia. Una buena metáfora sería un libro: las páginas y la tinta son el hardware, mientras que las palabras, oraciones, párrafos y el significado del texto son el software. Un computador sin software sería tan inútil como un libro con páginas en blanco.
Probablemente la definición más formal de software es la atribuida a la IEEE en su estándar 729: «la suma total de los programas de cómputo, procedimientos, reglas documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo. Bajo esta definición el concepto de software va más allá de los programas de cómputo en sus distintas formas: código fuente o código máquina, binario o ejecutable, además de su documentación: es decir, todo lo intangible.
Comúnmente se considera el software como sinónimo de programa, y sin embargo, en informática es un concepto mucho más general que abarca no sólo el código generado sino toda la documentación asociada. Existen dos grandes grupos, a saber: **software de sistemas y de aplicación**. El primero es el software utilizado por otro software, como los sistemas operativos, que es el programa que permite comunicar con los elementos hardware. El segundo, es el software destinado a ser utilizado por un usuario: una hoja de cálculo, un procesador de textos, un juego, etc.
La comunicación del usuario con el software se realiza hoy en día de forma generalizada a través de las denominadas interfaces gráficas de usuario. En éstas, las capacidades del programa están representadas por iconos o símbolos gráficos a los que el usuario accede mediante un dispositivo apuntador. Con un buen diseño, es fácil independizar interfaz y parte funcional del programa con lo que se consigue un mejor mantenimiento y mayor reutilización de sus elementos.

Definiciones:

• **Software de sistema**, que permite funcionar al hardware. Su objetivo es

aislar tanto como sea posible al programador de aplicaciones de los detalles del computador particular que se use, especialmente de las características físicas de la memoria, dispositivos de comunicaciones, impresoras, pantallas, teclados, etcetera. Incluye entre otros:

- Sistemas operativos
- Controladores de dispositivo
- Herramientas de diagnóstico
- Servidores
- Sistemas de ventanas
- Utilidades

• **Software de programación**, que proporciona herramientas para ayudar al programador a escribir programas informáticos y a usar diferentes lenguajes de programación de forma práctica. Incluye entre otros:

- Editores de texto
- Compiladores
- Intérpretes
- Enlazadores
- Depuradores

Los entornos integrados de desarrollo (IDE) agrupan estas herramientas de forma que el programador no necesite introducir múltiples comandos para compilar, interpretar, depurar, etcétera, gracias a que habitualmente cuentan con una interfaz gráfica de usuario (GUI) avanzada.

• **Software de aplicación**, que permite a los usuarios llevar a cabo una o varias tareas más específicas, en cualquier campo de actividad susceptible de ser automatizado o asistido, con especial énfasis en los negocios. Incluye entre otros:

- Aplicaciones de automatización industrial
- Aplicaciones ofimáticas
- Software educativo
- Software médico
- Bases de datos
- Videojuegos

1.2.3.1 Sistema Operativo (SO)

Es un programa que gestiona el hardware de un computador, proporcionando la base para los programas de aplicación y sirviendo como intermediario entre el usuario del computador y el hardware del computador. La figura 30 ilustra estas ideas.

• El hardware (CPU, memoria y dispositivos de entrada/salida (I/O:

input/output) proporciona los recursos básicos de computación.

• Los programas de aplicación (procesadores de texto, hojas de cálculo, compiladores, correo electrónico, navegadores, etc.) definen la forma en la que estos recursos se utilizan para resolver los problemas de computación de los usuarios.

• El sistema operativo controla y coordina el uso del hardware entre las distintas aplicaciones de los distintos usuarios.

Los dos objetivos principales de un SO son:

• eficiencia en el uso de recursos

• facilidad de uso para el usuario

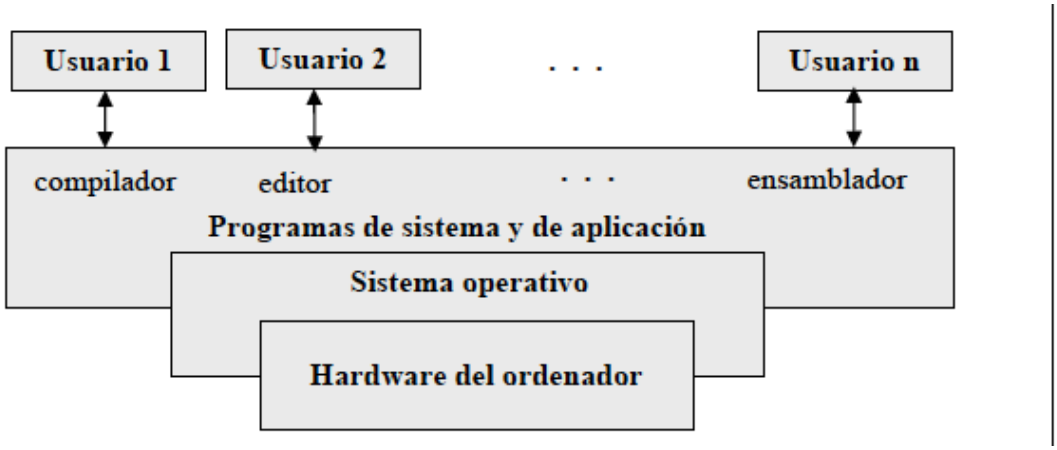


Figura 32: Esquema de función del sistema operativo

1.2.3.2 Elementos de programación y lenguajes

1.2.3.2.1 Lenguaje de programación

Un lenguaje de programación es un lenguaje formal, definido por un conjunto de normas «lingüísticas» que permiten escribir un programa y que éste sea entendido por el computador y sea ejecutado para conseguir el procesamiento deseado. La definición de un lenguaje de programación incluye especificar

• vocabulario: qué elementos léxicos están permitidos

• sintaxis: cómo se construyen las frases

• semántica: qué significan las frases

Por ejemplo una instrucción típica de asignación es

A:= B+C

Los elementos léxicos serían los símbolos A, B, C que pueden representar variables, y los símbolos : + que representan la operación suma, y := el operador de asignar un valor.
La regla sintáctica para la instrucción de asignación sería:

> Parte izquierda (variable), seguida del símbolo de asignación, seguida de una expresión (en este caso, variable, operador de suma, variable)

La interpretación semántica sería:

> Las instrucciones a realizar son calcular la suma de los valores de B y C, y el resultado, asignarlo a la variable A.

Existen diferentes formalismos para definir la sintaxis y la semántica de los lenguajes de programación. Para la sintaxis uno de los más usados es la notación BNF. Para saber más

> http://es.wikipedia.org/wiki/Notaci%C3%B3n_de_Backus-Naur

1.2.3.2.2 Problema, Algoritmo, Datos y Programa

Un problema algorítmico es cualquier problema cuya solución pueda expresarse mediante un algoritmo.
Un algoritmo es una lista de instrucciones que realizan una descripción paso a paso y precisa de un proceso que al llevarlo a cabo resuelve cualquier problema que pertenezca a una determinada clase.
En nuestra vida diaria encontramos ejemplos de algoritmos, la figura siguiente ilustra el paso 1 del algoritmo de montaje de un mueble de Ikea

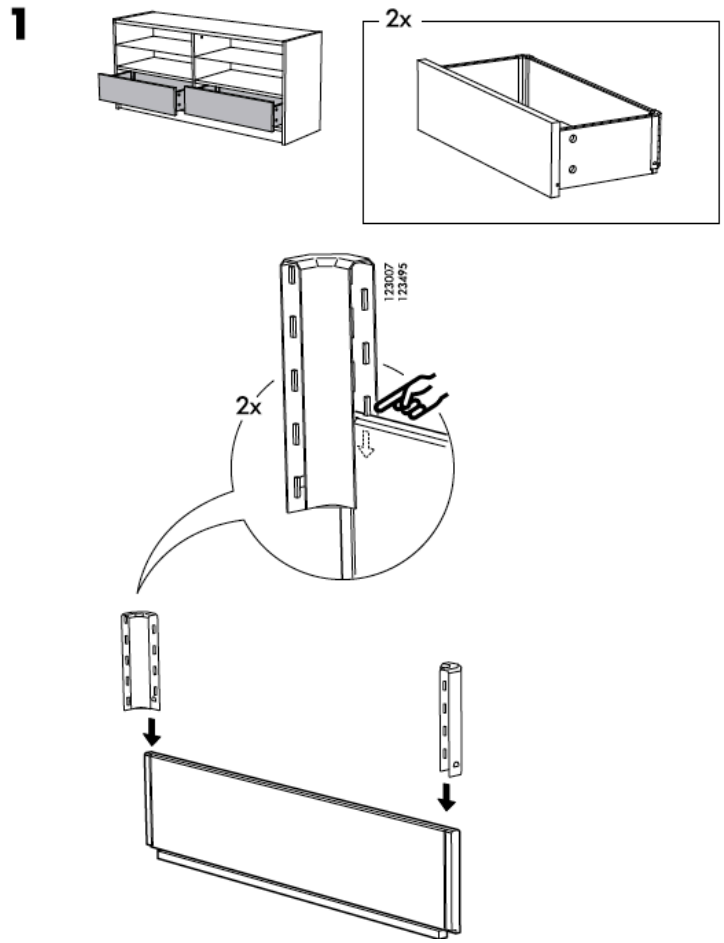


Figura: instrucción para montar un cajón

En informática, un algoritmo describe un proceso que se ejecutará por un computador. Expresa por tanto una serie precisa, *ordenada* y finita de instrucciones elementales que trabajan sobre los datos, posiblemente modificándolos.

Los *datos* son estructuras algebraicas que consisten, en general, en un conjunto de valores, con sus propiedades y con las operaciones que soportan.

Un *programa* es una transcripción de un algoritmo, mediante un lenguaje de programación. Es decir un conjunto de instrucciones escritas en un lenguaje susceptibles de ser ejecutadas por un computador para realizar una determinada tarea.

La *programación* es el proceso de diseñar, escribir, depurar y mantener el código fuente de programas ejecutables en una computadora.

- El código fuente se escribe en un lenguaje de programación entendible por el programador. Dicho código no se puede ejecutar directamente en una computadora.

• El código fuente debe someterse a un proceso de traducción para convertirlo en lenguaje máquina directamente ejecutable por computador.

1.2.3.2.3 Niveles en los lenguajes de programación

Lenguajes de Bajo Nivel
Lenguaje máquina: En un principio, el lenguaje de programación estaba directamente orientado al tipo de instrucciones que el procesador del computador era capaz de entender y ejecutar (escrito en binario!).

Lenguaje ensamblador: versión "humanizada" del lenguaje máquina que utiliza nombres de instrucciones (ADD, LOAD, etc). Todavía muy cercano a la ejecución de la máquina.

Por ejemplo, en el lenguaje ensamblador para un procesador x86:
La sentencia

- `MOV AL, 061h`

Asigna el valor hexadecimal `61` (97 decimal) al registro "`AL`".
El programa ensamblador lee la sentencia de arriba y produce su equivalente binario en lenguaje de máquina

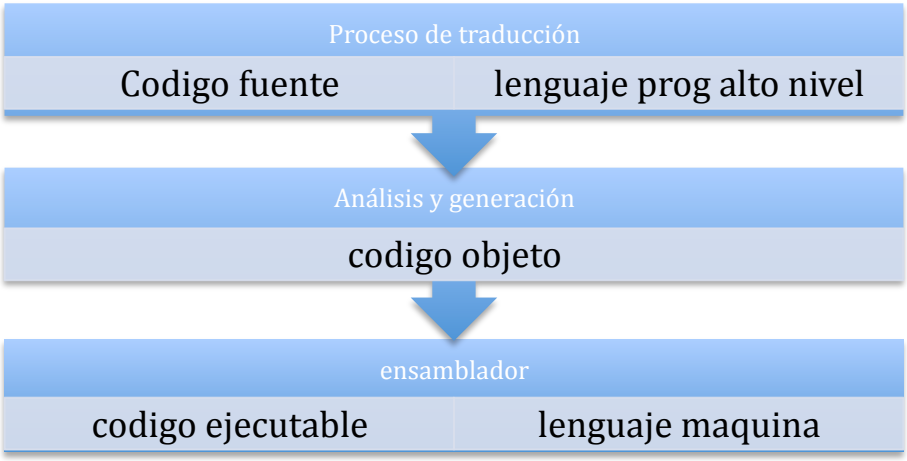
- Binario: `10110000 01100001` (hexadecimal: `B061`)

Para saber más:
http://es.wikipedia.org/wiki/Lenguaje_ensamblador

Lenguajes de Alto Nivel
Abstraen el problema a resolver de la máquina y procesador concretos que lo van a ejecutar el programa. Ejemplos: C, Pascal, C++, Java, etc.

1.2.3.2.4 ¿Cómo se ejecuta un programa?
Los programas escritos en un lenguaje de alto nivel (código fuente) se tienen que traducir a un lenguaje ejecutable por el procesador de la máquina. Existen dos variantes principales:

• *Compilador*: traduce el código fuente a las instrucciones básicas del computador (código máquina), que luego se ejecutan. El proceso tiene dos fases principales, una de análisis y generación para obtener el código objeto, y otra de paso al lenguaje máquina para tener el código ejecutable.



• *Intérprete*: toma una instrucción del código fuente, la traduce a código máquina, la ejecuta, y realiza lo mismo con las siguientes instrucciones. Algunos ejemplos significativos son los interpretes de comandos de los sistemas operativos o los de los lenguajes basados en scripts (Perl, JavaScript)

Como veremos más adelante, hay también soluciones intermedias entre la compilación y la interpretación, como es el caso del lenguaje JAVA que es el que aprenderemos en este curso. En Java se compila a un lenguaje intermedio para una maquina virtual, y después se interpreta ese lenguaje en la maquina física concreta que se utilice.

1.2.3.3 Paradigmas de programación

Todos los lenguajes de programación proporcionan abstracciones. La complejidad de los problemas que se pueden resolver depende del tipo y la calidad de la abstracción.
El programador debe establecer una asociación entre el "espacio de la solución" (el lugar donde se modela el problema, en este caso el computador) y el "espacio del problema" (por ejemplo, si se quiere programar una aplicación bancaria, los elementos que conforman el entorno bancario). En este sentido, el lenguaje ensamblador es una primera abstracción de la máquina subyacente.
Un paradigma de programación es un modelo básico de diseño y desarrollo de programas. Según el tipo de abstracción que se realice, tendremos los distintos tipos de *paradigmas de programación*.

Una breve visión histórica de la evolución de los lenguajes de programación

1ª generación: Los lenguajes de esta generación se caracterizan por poseer

una codificación muy compleja, lo cual los hace difíciles de aprender, entender y aplicar. Más que lenguajes en el sentido de aportar algún grado de abstracción, son más bien códigos de máquina y no requieren traducción alguna porque el computador es capaz de leerlos directamente.

2ª generación: Se llama lenguajes de segunda generación a los lenguajes ensamblador. Requieren una traducción, aunque ésta es muy simple porque cada instrucción corresponde a un código solamente. Son mucho más fáciles de manejar para los profesionales que los códigos hexadecimales, por lo que se les considera lenguajes codificados y a cada palabra le corresponde una instrucción del microprocesador.

3ª generación: Estos son los más utilizados y están diseñados para ser usados por programadores profesionales. Se caracterizan por su transportabilidad, es decir, que están implementados sobre varias máquinas de forma que un programa puede ser fácilmente llevado de una máquina a otra sin una revisión sustancial.

Aunque no son fundamentalmente declarativos, estos lenguajes permiten que los algoritmos se expresen en un nivel y estilo de escritura fácilmente legible y comprensible por otros programadores. Los códigos pueden ser difíciles de leer, entender, mantener y depurar.

4ª generación: Según el IEEE (1990), se puede definir un lenguaje de esta generación como aquél desarrollado para mejorar la productividad de un lenguaje de la tercera generación, que muchas veces, permite a los no-programadores aprovechar la funcionalidad disponible en el computador sin escribir programas complejos. Los rasgos de un lenguaje de esta generación son: un sistema de gestión de base de datos integrado, un lenguaje de consulta, un generador de informes y gráficos, la posibilidad de producir modelos financieros, la funcionalidad de una hoja de cálculo y algunas funciones estadísticas.

5ª generación: Según el IEEE (1990), se puede definir un lenguaje de esta generación como el que incorpora los conceptos de sistemas basados en el conocimiento, sistemas expertos y el procesamiento del lenguaje natural. Todavía no existe ningún lenguaje que realmente se pueda definir como de quinta generación de propósito general.

1.2.3.3.1 Principales paradigmas de programación

En esta sección se resumen los principales paradigmas de programación y se realiza una breve caracterización de los mecanismos de algunos lenguajes de programación. A la hora de describir las características de un lenguaje de programación, compararlo con otros o situarlo dentro de un paradigma, se hace referencia a los mecanismos que tiene. En algunos casos (como en el paradigma imperativo), muchos de los mecanismos

vienen impuestos directamente por la arquitectura de la máquina y en otros casos (como en el paradigma funcional), los mecanismos vienen marcados por la teoría en la que se basan. Esto no quiere decir que la mayoría de los lenguajes de programación no tengan la mayoría de los mecanismos, sino solamente que el grado de acceso a ellos que tiene el programador, y su visibilidad y naturaleza implícita o explícita, dependen de cada uno.
Hay que tener en cuenta que la lista de mecanismos no ha sido siempre fija, sino que ha ido evolucionando y creciendo. Y no hay ninguna razón para pensar que no va a seguir expandiéndose con avances en investigación y la implantación de nuevos lenguajes e incluso nuevos paradigmas basados en nuevas metáforas de computación.

Programación imperativa

En este paradigma un programa se construye mediante instrucciones. La programación imperativa nació junto con los primeros computadores y lenguajes en los años 50 y 60 y reflejaba directamente la arquitectura moderna del computador, basada en lo que propuso Von Neumann. Se puede caracterizar un lenguaje como imperativo si dispone de las siguientes propiedades (Sethi, 1992; Tucker y Noonan,2002): el concepto básico es el **estado** de la máquina; un programa consiste en un conjunto de instrucciones que se ejecuta y que cambia el valor de alguna variable o posición de memoria; tipos de datos para números reales, caracteres, cadenas, boléanos y sus operadores; enunciados de control de flujo, como por ejemplo: *for*, *while*, *case* e *if*; estructuras de datos que se puede asignar; comandos para controlar la entrada y salida de datos; punteros y procedimientos y funciones.
Casi todos los lenguajes de propósito general son de este tipo. Como ejemplos representativos en esta categoría se pueden destacar: FORTRAN, Algol, COBOL, Pascal, Modula 2, PL/I, C y Ada. A continuación se presentará un breve resumen de los aspectos más notables, basado en varias fuentes bibliográficas (Pratt y Zelkowitz, 2001; Sethi, 1992; Wilson y Clark, 2001; Sebesta, 2002)

FORTRAN: (**For**mula **Tran**slating System) uno de los primeros, y más usados lenguajes de calculo científico. Desarrollado por J. Backus en Octubre del 56. Ya en la versión del 58, incluía la noción de subprograma. Sigue activo, con sucesivas extensiones, la última la de 2008, soporta incluso programación concurrente.
Para saber más: http://es.wikipedia.org/wiki/Fortran

COBOL: *Common Business Oriented Language* (lenguaje común orientado a los negocios) fue desarrollado a finales de los años 50 y aparecieron compiladores para él a principios de los años 60. Difería

considerablemente de otros LP de la época como FORTRAN y Algol, por estar muy orientado a los negocios. Una buena parte de aplicaciones bancarias están aún hoy día en COBOL.

ALGOL: diseñado por un comité internacional a principios de los 60. En el ALGOL aparecen por primera vez muchos de los conceptos de los lenguajes algorítmicos que se desarrollaron posteriormente:

Definición de la sintaxis en notación BNF (Backus-Naur Form).
Declaración explícita de tipo para todos los identificadores.
Estructuras iterativas más generales.
Recursividad.
Paso de parámetros por valor y por nombre.
Estructura de bloques, lo que determina la visibilidad de los identificadores.

Pascal y **Modula 2**: Pascal fue desarrollado a finales de los años 60 y principios de los 70 por Niklaus Wirth, basado en su experiencia con Algol W. Quiso producir un LP que se pudiera implementar eficientemente en la mayoría de los computadores y que sirviera para la enseñanza de la programación estructurada. Pascal es un lenguaje tipado (lo que facilita la detección de errores en tiempo de compilación) y estructurado (mediante funciones y procedimientos) .
Wirth desarrolló **Modula** y luego **Modula 2** a partir de Pascal, con una mejora en la sintaxis y semántica respecto a Pascal, donde la mayor extensión consistía en la existencia de módulos para facilitar la reutilización del código del lenguaje.
C: Es un lenguaje orientado a la implementación de software de sistemas, apreciado por la eficiencia del código que genera. Solucionó los problemas de resistencia a lenguajes de alto nivel por parte de los programadores de sistemas. Aunque originalmente estaba muy vinculado al sistema operativo UNIX, en los años 80 salieron versiones de compiladores para otros sistemas y quizás hoy en día sea el lenguaje con compiladores para la mayor gama de SO y máquinas.
Para saber más
http://es.wikipedia.org/wiki/C_%28lenguaje_de_programaci%C3%B3n%29

Ada: El Departamento de Defensa de los EEUU decidió que quería un nuevo lenguaje que se pudiera usar para controlar sistemas complejos, grandes, con cierto grado de paralelismo, y que cambiaron con el tiempo. La definición del nuevo lenguaje, llamado Ada, apareció en 1983 y es el único en la historia que fue un estándar antes de la aparición del primer compilador (que apareció en 1986). Fue la primera vez que aparecieron pruebas oficiales para un compilador, pero años más tarde, la empresa Sun

hizo algo parecido para el Java, pero con un pequeño cambio de énfasis: lo hizo para comprobar qué programas no utilizan elementos que forman parte de la versión oficial del lenguaje.

Características de los lenguajes imperativos

Un programa en un lenguaje imperativo aparece como una lista de instrucciones u órdenes elementales que han de ejecutarse una tras otra, en el orden en que aparecen en el programa.

Las instrucciones de un programa imperativo utilizan datos almacenados en la memoria del computador, llamados variables. Para realizar algún cálculo, se parte de ciertos datos almacenados y se realizan diversas operaciones (instrucciones); al final, el resultado está almacenado en alguna celda de memoria.

Los elementos principales de la programación imperativa son:

• *Concepto de variable*. El componente principal es la memoria, compuesto por un gran número de celdas donde se almacenan los datos y que son referenciadas por medio de su nombre (variable). El conjunto de valores de todas las variables del programa en un momento dado representa el estado del programa.

• *Operaciones de asignación*. Cada valor calculado debe ser asignado a la variable mediante operaciones de asignación. De esta forma se modifica el estado del programa.

• *Repetición*. Un programa imperativo, normalmente realiza su tarea ejecutando repetidamente una secuencia de pasos elementales.

El paradigma imperativo es una abstracción del lenguaje ensamblador. En los términos mencionados anteriormente, el modelado se realiza más cerca del "espacio de la solución" que del "espacio del problema".

Ejemplo: programa en C para conversión de temperaturas.

```
#include <stdio.h>
/* imprime la tabla Fahrenheit-Celsius
para fahr=0,20,...,3000*/
main ()
{
int fahr, Celsius;
int lower, upper, step;
lower = 0;
upper = 300;
step = 20;
fahr = lower;
while (fahr <= upper){
Celsius = 5*(fahr–32)/9;
printf("%d\t%d\n", fahr, Celsius);
```

```
fahr = fahr + step;
}
}
```

Programación Declarativa

Los lenguajes declarativos están orientados a buscar la solución del problema, sin preocuparse por la forma de llegar a ello; es decir, el programador debe concentrarse en la lógica del algoritmo, más que en el control de la secuencia, por ello los programas están formados por un conjunto de definiciones o ecuaciones, las cuales describen lo que debe ser calculado, no en sí la forma de hacerlo, mientras que los lenguajes imperativos describen paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado un programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar un problema.

Los programas se construyen mediante descripciones de funciones o expresiones lógicas. Las variables sólo pueden tener asignado un solo valor a lo largo de la ejecución del programa, lo cual implica que no puede existir asignación destructiva. Debido a esto, cobra especial importancia el uso del anidamiento y la recursividad. Las listas representan la estructura fundamental de datos. El orden de la ejecución no resulta importante debido a que no existen efectos colaterales; es decir, que al calcular un valor, resulta imposible afectar el cálculo de otros y con esto se puede afirmar que cualquier secuencia de ejecución deberá conducir al mismo resultado. Las expresiones o definiciones pueden ser usadas como valores y por lo tanto se pueden tratar como argumentos de otras definiciones. El control de la ejecución no es responsabilidad del programador.

Hay tres familias de lenguajes declarativos:

- algebraicos (Maude, SQL) ,

para saber más de este último, importante para las bases de datos http://es.wikipedia.org/wiki/SQL

- lógicos (Prolog) y
- funcionales (Haskell),

así como diversas variantes de combinación entre ellos.

Programación funcional

El paradigma funcional se basa en el concepto matemático de función. Una función es una regla de correspondencia que asocia a cada elemento de un conjunto origen un elemento de un conjunto destino. Los conjuntos origen y destino suelen llamarse, respectivamente, dominio y rango de la función.
Los programas escritos en un lenguaje funcional están constituidos

únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos colaterales.
Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas). Otro elemento básico es la composición funcional (el resultado de un cálculo es el argumento del siguiente).
Los elementos principales de la programación funcional son:

- Tipos de datos
- Expresiones condicionales
- Recursión

El paradigma funcional, en lugar de abstraer la máquina subyacente, trata de abstraer el problema a solucionar, modelándolo como una composición de funciones. En el caso de los programas funcionales, se modela el problema que se quiere resolver, y no una abstracción de la máquina que lo va a resolver. Es decir, estamos más cercanos al "espacio del problema" que al "espacio de la solución". Este paradigma tiene una visión particular del problema a resolver basada en funciones.
Algunos ejemplos de lenguajes de programación funcional son LISP, Haskell, Gofer, etc.

Ejemplo: implementación del algoritmo de ordenación quicksort en Haskell

```
qsort [] = []
qsort (x:xs) = qsort elts_lt_x ++ [x] ++ qsort elts_greq_x
where
elts_lt_x = [y | y <- xs, y < x]
elts_greq_x = [y | y <- xs, y >= x]
```

Si sientes curiosidad por la programación funcional , consulta más información en

http://www.haskell.org/haskellwiki/Es/Por_que_Haskell_importa

Programación orientada a objetos

En los años 70 había un acuerdo sobre la necesidad de facilitar la reutilización de código. Y como resultado hubo un esfuerzo para extender el concepto de la abstracción de los programas imperativos (abstracción de procedimientos y funciones) para incluir datos y tipos de datos abstractos.

Se consiguió proporcionar abstracción de datos con un mecanismo de encapsulamiento que limitó tanto la visibilidad como el ámbito de los datos y las funciones definidas por ellos. Aun así, con los módulos y paquetes, todavía hubo problemas porque no existían mecanismos para la inicialización y terminación automática de instancias de estos tipos, ni tampoco para extender los tipos de datos abstractos para añadir nuevas funciones. La programación orientada a objetos (a partir de ahora POO) nació de las ideas originales del lenguaje Simula (tratado a continuación) y el concepto de "clase" formaba una base para solucionar los problemas anteriores.

La POO recibió mucho interés en los años 80 y se convirtió en el paradigma dominante a finales de los años 90. Al principio había muchos lenguajes orientados a objetos (a partir de ahora, LPOO), muchos de los cuales incorporaron aspectos de la programación funcional. A continuación se hará un resumen de algunos de los principales LPOO, es decir, Simula, Smalltalk, Eiffel, C++ y Java:

Simula: Al principio de los años 60 un grupo en Noruega estaba trabajando en un lenguaje para llevar a cabo simulaciones de sistemas. Eventualmente su trabajo dio a luz Simula 67. Se puede decir que este lenguaje fue el padre de lo que es el paradigma de la programación orientada a objetos (a partir de ahora, OO) porque definió el concepto de clase como un objeto que empaqueta tanto los datos como los procedimientos y las funciones que los manipulan.

Smalltalk: fue el resultado del trabajo de Alan Kay y sus colegas en Xerox para producir un computador personal para no expertos, llevado a cabo al principio de los años 70. El lenguaje en sí formaba parte de un entorno de desarrollo visual y era difícil distinguir entre los límites del lenguaje y el entorno. Smalltalk seguía el desarrollo del concepto de clase de Simula aunque recibió mucha influencia también de Lisp. Hasta la mitad de los años 80 cuando la programación OO empezó a ser muy popular, Smalltalk era el LPOO por defecto.

Eiffel: Fue diseñado por Betrand Meyer a mediados de los años 80 con el objetivo de ser un lenguaje que apoyara la creación de sistemas grandes y robustos. La gran diferencia con Smalltalk era su sistema de tipos estáticos y la comprobación de tipos durante el proceso de compilación, algo importante para producir sistemas robustos.

C++: Fue desarrollado por Bjarne Stroustrup desde 1979 a 1985 para añadir el mecanismo de clases de Simula al lenguaje C. Ha tenido un impacto muy importante en la evolución de C y en la forma de compilar programas escritos en C en un compilador de C++ con pocos cambios. Ha servido como un puente entre el mundo de la programación OO y el mundo industrial. No es un lenguaje completamente OO, pero ha resultado uno de los más utilizados.

Java: Fue desarrollado por Sun Microsystems para programar dispositivos electrónicos en 1990/1. Como lenguaje es más sencillo y pequeño que C++, y se desarrolló sobre la base de lo que los diseñadores pensaban que eran las peores características de C++ (en el sentido de causar problemas a la hora de programar). Como características importantes del lenguaje, adicionales a las de la orientación a objetos, se pueden destacar dos: en primer lugar, el papel central otorgado a las bibliotecas del LP a través de la API (*Application Programming Interface*) de Java. En segundo lugar, se puede destacar el hecho de que no se compila un programa en este lenguaje a código de máquina, sino a un código intermedio que se puede considerar como un lenguaje de máquina para una máquina virtual de Java. El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

Programación orientada a objetos

La programación orientada a objetos (POO) permite representar directamente en el programa los elementos del espacio del problema. Los elementos del espacio del problema y su representación en el espacio de solución se denominan *objetos*. La idea principal es que el código que describe la solución se escribe en los términos del problema.

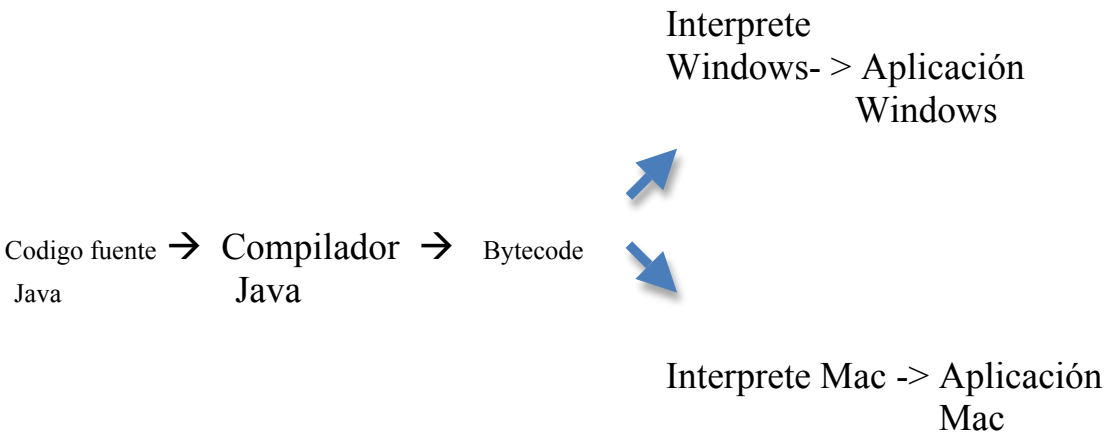
Cada objeto tiene un estado, representado por unos atributos (parecido a las variables) y unas operaciones que puede realizar.

Las principales características de la programación orientada a objetos son las siguientes:

1. *Todo es un objeto.* Cualquier elemento conceptual de problema (una persona, una cuenta bancaria, un circuito electrónico, un servicio...) puede representarse como un objeto en el programa.
2. *Un programa es un conjunto de objetos que se hacen peticiones los unos a los otros mandándose mensajes*. Mandarse un mensaje es equivalente a hacer una llamada a una de las operaciones ofrecidas por el objeto.
3. *Cada objeto tiene su propia memoria compuesta de otros objetos*. Esos otros objetos son los atributos del objeto (por ejemplo, un coche tiene una matrícula). De esa forma se pueden crear agrupaciones complejas que resulten sencillas de manejar a través del objeto.
4. *Cada objeto es de una clase determinada*. Una clase es como un tipo. Los objetos son instancias de la clase. La clase determina cómo está compuesto el objeto (qué atributos tiene) y cómo se comporta (qué mensajes se le pueden mandar, es decir, qué operaciones ofrece).
5. *Todos los objetos de una clase pueden recibir el mismo tipo de mensajes*.

Java es un lenguaje que se caracteriza por ser independiente de la

plataforma. Por plataforma se entiende el conjunto de hardware (computador con su procesador) y sistema operativo. Esto se consigue realizando una mezcla entre compilación e interpretación
Primero se compila el programa fuente a un lenguaje intermedio llamado *bytecode* (códigos de byte), que es parecido al lenguaje máquina, pero independiente de una máquina concreta.
Cada máquina sobre la que se ejecute Java va a tener una máquina virtual capaz de interpretar el bytecode. Por ello, el mismo fichero bytecode va a poder interpretarse / ejecutarse en distintas máquinas.



Además de los paradigmas mencionados hay otros como la programación guiada por eventos, la programación visual, o la programación concurrente. Para saber más:

http://es.wikipedia.org/wiki/Programaci%C3%B3n_dirigida_por_eventos

y los libros clásicos de Lenguajes de Programación mencionados en la sección de referencias.

1.2.3.4 Entornos de desarrollo integrados

Los entornos de trabajo facilitan un editor de código con acceso a las bibliotecas del lenguaje, un compilador y un depurador, todos integrados de manera que desde el editor se puede dar el comando para compilar el código y depurar simbólicamente (es decir, ir desde el editor línea por línea en el programa, inspeccionando las variables y estructuras de datos: lo que se llama IDE [*Integrated Development Environment* – entorno integrado de desarrollo]).
El gran adelanto que tuvo lugar en los años 90 con estos entornos de

desarrollo fue la inclusión de herramientas visuales para el diseño de la interfaz gráfica de la aplicación. Así que un programador podía dibujar la interfaz que quería, con botones, cajas de textos, menús despegables, etc., y a la vez el entorno produciría el código fuente. Se llamó a dichos entornos *herramientas de desarrollo rápido* (RAD –*rapid application development*) y hoy en día casi siempre es la herramienta de elección por parte de los programadores. Hay que tener un poco de cuidado con ellos, porque en el caso de las herramientas (y lenguajes) como Delphi, Visual BASIC y Visual C++, los API son para el SO Microsoft Windows y por lo tanto los programas generados (especialmente la interfaz gráfica) no funcionará en otras plataformas. Esto no ocurre con las herramientas para Java porque, como ya se ha dicho, Java fue desarrollado para funcionar en muchas plataformas, incluso con la misma interfaz gráfica.

Un Entorno integrado de desarrollo, ofrece al programador las siguientes herramientas:

•Editor de texto
- Desarrollo de código fuente

•Compilador y enlazador
- Traducción de código fuente a código máquina o código intermedio

•Depurador para la detección y corrección de errores en los programas

Los diferentes tipos de errores que se detectan son
- Sintácticos
  El código fuente es sintácticamente incorrecto
  (El compilador es capaz de identificar este tipo de errores)
- en tiempo de ejecución: se produce un error durante la ejecución del programa y aborta (Por ejemplo División por cero),
- Semánticos: El programa no se comporta como debiera

El depurador constituye una herramienta muy útil para ayudar al programador a resolver el problema

El entorno de desarrollo recomendado por el equipo docente en esta asignatura, BLUEJ, es sencillo y versátil, diseñado especialmente para una aprendizaje introductorio de la POO en JAVA.

1.2.3.5 Ingeniería del Software

En la primera época de los computadores el desarrollo del software era puramente artesanal. Pero a mediados de los años setenta en el siglo XX el hardware ofrecía cada vez mayores posibilidades. En los nuevos sistemas software que se comenzaban a desarrollar, la complejidad era ya un factor a tener en cuenta y las técnicas artesanales ya no servían y era necesario

hacer una planificación de cara a rentabilizar el esfuerzo. Pero la falta de control sobre el desarrollo de estos sistemas desembocó en lo que hoy se conoce como crisis del software (Gibbs, 1994). El problema alcanzó tal magnitud que en la reunión del Comité Científico de la OTAN que tuvo lugar en 1968 se propuso una solución que consistía en tratar el desarrollo de software con las técnicas ingenieriles utilizadas por las industrias fabricantes de productos "físicos". Así, la ingeniería del software se puede definir como la aplicación de una aproximación sistemática, disciplinada y cuantificable al desarrollo, operación y mantenimiento del software (Pressman, 2002).

El proceso de construcción del software requiere, como cualquier otra ingeniería, identificar las tareas que se han de realizar sobre el software y aplicar esas tareas de una forma ordenada y efectiva. Adicionalmente el desarrollo del software se debe realizar por un conjunto coordinado de personas simultáneamente, y por lo tanto sus esfuerzos deben estar dirigidos por una misma metodología que permita estructurar las diferentes fases del desarrollo. Algunas empresas y universidades han desarrollado a lo largo del tiempo varias metodologías, es decir, de modos sistemáticos de producir software. Todas ellas tienen en común la intención de automatizar el proceso de desarrollo para que el producto resultante cumpla una serie de requisitos (tiempo de desarrollo, calidad, eficiencia, precio, mantenibilidad). Las metodologías dividen la realización de un proyecto en una serie de etapas que son: especificación, diseño, codificación, pruebas y mantenimiento. Las distintas formas de ordenar el esfuerzo a lo largo del tiempo son los ciclos de vida.

En concreto, el software adopta varias formas en distintos momentos de su ciclo de vida:

• Código fuente: escrito por programadores. Contiene el conjunto de instrucciones destinadas a la computadora.

• Código objeto: resultado de compilar el código fuente, como si fuera una traducción de éste último. El código objeto no es directamente inteligible por el ser humano, pero tampoco es directamente entendible por la computadora.

• Código ejecutable: resultado de enlazar uno o varios fragmentos de código objeto. Constituye un archivo binario con un formato tal que el sistema operativo es capaz de cargarlo en la memoria de un computador, y proceder a su ejecución. El código ejecutable es directamente inteligible por la computadora.

Por último, cabe mencionar dos tipos de herramientas que se han producido que marcan nuevos tipos de sistemas de programación que podrían ser muy importantes en el futuro de la programación. En primer lugar, las herramientas que permiten a un usuario no programador construir

programas utilizando una herramienta visual y paletas de componentes que se pueden conectar para especificar el flujo del programa sin escribir una línea de código (por ejemplo, JavaStudio). En segundo lugar, las herramientas que permiten el diseño de programas utilizando **UML** (*Unified Modelling Language* o Lenguaje Unificado de Modelado) y después producen automáticamente el código fuente del programa.
Estas herramientas ofrecen la posibilidad de mantener el problema del desarrollo de programas a un nivel abstracto donde es posible llevar a cabo análisis sobre el diseño sin bajar al nivel del código.
Como resumen de las etapas para la creación de un software, se pueden mencionar:

- Análisis.
- Desarrollo.
- Construcción.
- Pruebas (unitarias e integradas).
- Paso a Producción.

Dentro de estas etapas, existen sub-etapas (para algunos son otras etapas, como por ejemplo, paso a ambiente beta/rc).
Para saber más sobre UML:
http://www.ibm.com/developerworks/rational/library/769.html

1.2.3.6 Clasificación del software según la licencia de uso

Una primera clasificación nos permite caracterizar el software como propietario, libre, y con licencia GPL.

Software propietario

Se dice del software que pertenece a una persona o empresa, está sujeto a derechos de autor, y su distribución, reproducción, modificación y comercialización está controlada por el propietario. Frecuentemente el usuario sólo tiene acceso al código ejecutable, no pudiendo conocer el código fuente.

Software libre (open source)
Se dice del software que pertenece al usuario. En este caso el autor conserva la propiedad intelectual pero suele renunciar a cobrar por su distribución. El autor permite su distribución, reproducción, modificación y comercialización normalmente en los mismos términos.

Licencia GPL
El código fuente está disponible

Atendiendo a la forma de distribución, podemos establecer la siguiente clasificación:

•Adquisición de licencia
La mayoría del software propietario tiene este tipo de distribución

•Freeware
Software liberado por el autor para su uso gratuito
Debe ser utilizado en las formas expresamente permitidas por el autor. La mayoría del software libre es gratuito

- Shareware y demo, en esta categoria encontramos:
  Software sujeto a derechos de autor
  Distribuido sin cargo como versión de evaluación
  Prototipos o versiones incompletas (beta)

- Abandonware
  Software sujeto a derechos de autor aunque cedido por el autor para su uso gratuito. Debido a su antigüedad no se comercializa

Para saber más:

http://es.wikipedia.org/wiki/Software_libre

http://es.tldp.org/Otros/gples/gples.html

Referencias

Pratt , Zelkowitz, 2001. Programming Languages Design And Implementation. Prentice Hall.

Robert W.Sebesta, 2002. Concepts of Programming Languages (5th Edition) Addison-Wesley.

Sethi, 1992-(Sethi, 1992) Sethi, R. Lenguajes de programación: conceptos y constructores. Addison- Wesley, 1992

Tucker, Noonan, Lenguajes de programación, principios y paradigmas McGraw-Hill, 2002

Wilson, Clark, 2001; Comparative Programming Languages. Addison-Wesley, 3 edition.

Créditos y Nota importante

Este material se ha creado para apoyar el estudio de la asignatura de Fundamentos de Informática en los grados de Ingeniería de la UNED. Algunas de las fotos (figura 1 a 8, 12 a 15 y 22 a 24) incluidas, han sido obtenidas en la web y por tanto no deben reutilizarse fuera de este contexto. Parte del material es una reelaboración de unos apuntes previos elaborados por los profesores C. Rodrigo & J.L. Delgado.