Tema 1. Introducción a la programación

Principales conceptos que se abordan en este tema

- Conceptos sobre la programación de ordenadores
- Evolución histórica de los lenguajes de programación
- Razones por las que la programación a objetos se aplica en el desarrollo del software

Número de sesiones

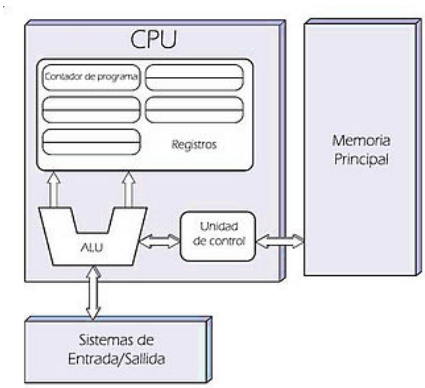
Se estiman un total de 4 horas

1.1 Estructura de un ordenador

La arquitectura básica de un ordenador digital se basa en la arquitectura de una máquina descrita por Von Newman en 1944. La arquitectura Von Newmann divide a un ordenador en cuatro partes:

- **Unidad de control** - es el componente básico de un ordenador.
  - Controla la ejecución de las operaciones y dirige el funcionamiento de todos los demás componentes, de tal forma que el trabajo conjunto de todos conduzca a la consecución de las tareas específicas programadas en el sistema.
- **Unidad Aritmético-Lógica** – es la parte encargada de realizar las operaciones aritméticas (suma, resta….) y las operaciones lógicas (comparación….). También realiza otras funciones más complejas (raíces, funciones trigonométricas...).
  - Al conjunto Unidad de Control y Unidad Aritmético-Lógica se le conoce como **CPU** (**Central Process Unit – Unidad Central de proceso**). Así tenemos los procesadores Intel, Motorola, AMD.
- **Memoria principal** – es la memoria de almacenamiento interno. Opera a gran velocidad. Aquí se ubican los programas: las instrucciones junto con los datos sobre los que actúan.
  - La memoria principal está formada por una serie de ***celdas*** o posiciones identificadas por una ***dirección*** de memoria.
  - Por otro lado está la ***memoria secundaria*** o ***memoria externa*** que permite resolver los problemas de volatilidad y capacidad de la memoria principal (discos duros, CD…).
- **Dispositivos de Entrada / Salida** – son los que facilitan la interacción del usuario (el mundo exterior) con la máquina (teclado, monitor, impresora….).

Los ordenadores digitales utilizan internamente el ***código binario***. La mínima información manipulable es un dígito binario, 0 y 1, el ***bit***. Tanto los datos como las instrucciones que ejecuta la CPU han de expresarse en este código para poder ser almacenados en memoria. Al conjunto de instrucciones codificadas en binario se le conoce como ***lenguaje máquina*** y es el lenguaje más básico y el único que entiende el ordenador.



Maquina de Von Newman

Lenguaje máquina: es el lenguaje propio del ordenador, basado en el ***código binario.***

Para saber más

En el siguiente enlace puedes aprender más sobre el código binario:

El código binario. Explicación

1.2 Algortimos y programas

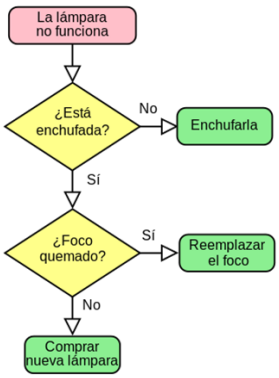
Un ***algoritmo*** es un conjunto de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien lo ejecute.

En todo algoritmo se distingue:

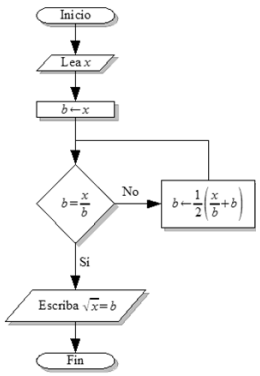
- el *procesador* – es el que entiende los pasos del algoritmo y los lleva a cabo (por ej. un cocinero / un ordenador)
- el *entorno* – los materiales necesarios para la ejecución del algoritmo (por ej. huevos, patatas, cebolla / *datos* en un programa)
- las *acciones* – los actos del procesador sobre el entorno (cascar, batir, freír / sumar, restar, comparar, asignar)

Para que un ordenador pueda ejecutar un algoritmo ha de expresarse en forma de ***programa*** a través de un lenguaje de programación.

Los diagramas de flujo sirven para representar algoritmos de forma gráfica.



Algoritmo encender lámapra



Algoritmo raiz cudrada

> Un *algoritmo* es una secuencia ordenada y finita de pasos a seguir para resolver un problema.

Un ***programa*** es un conjunto de instrucciones que indican a un procesador, que puede o no estar en el ordenador, las acciones que debe ejecutar. Estas expresiones están expresadas en lenguaje o código máquina, que es el único lenguaje que entienden los procesadores (es como el idioma: si no me hablan en mi idioma no me entero)

Estas instrucciones de código máquina son una serie de secuencias de bytes. Cada byte se compone de 8 bits y un bit puede ser un 0 o un 1. Un bit es la unidad mínima de información con la que es capaz de trabajar un procesador.

En resumen: un procesador se nutre de bytes o chorros de 0 y 1 agrupados de 8 en 8, que representan las instrucciones de un programa. Cuando se interpretan y procesan se obtiene el resultado deseado por el programador. ¿Cómo se crea un programa o aplicación? Con un lenguaje de programación.

Un *programa* es un conjunto de instrucciones que actúan sobre unos datos y que están expresados en un lenguaje de programación

Los ***lenguajes de programación,*** en términos coloquiales, son programas que sirven para crear otros programas. Al igual que el lenguaje natural consta de sintaxis, semántica y vocabulario, un lenguaje de programación se compone de un conjunto de símbolos (léxico), un conjunto de reglas de sintaxis (que indican cómo construir correctamente las instrucciones) y una semántica (reglas que determinan el significado de las construcciones del lenguaje), que el ordenador es capaz de entender y procesar.

Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

Cuando un algoritmo se expresa en un lenguaje de programación tenemos un programa comprensible por un ordenador. Al hecho de expresar el algoritmo en un lenguaje de programación dado se denomina ***codificar*** un programa.

Un *lenguaje de programación* es un lenguaje artificial que puede ser usado para controlar el comportamiento de una máquina.

1.3 Evolución de los lenguajes de programación

Una de las clasificaciones que podemos hacer de los lenguajes de programación es utilizando el *criterio de proximidad* del lenguaje con la máquina o con nuestro lenguaje natural:

a. Lenguaje máquina - es el lenguaje básico y el único que entiende el ordenador.

Está formado por un conjunto de instrucciones codificadas en binario, que consta de dos únicos símbolos: 0 y 1 denominados bits (abreviatura inglesa de Binary digit. -Dígito binario-), dependientes totalmente del hardware del ordenador.

Inicialmente se programaba así, pero dejó de utilizarse por su dificultad y complicación, siendo sustituido por otros lenguajes más fáciles de aprender y utilizar, que además reducen la posibilidad de cometer errores.

b. Lenguaje de bajo nivel o lenguaje ensamblador – es una versión simbólica del lenguaje máquina.

Cada instrucción lleva asociado un símbolo (una palabra nemotécnica – ADD, SUB…) para que resulte más fácil la programación. Requiere una fase de traducción al lenguaje máquina (con un traductor – *assembler*).

El lenguaje ensamblador, como el lenguaje máquina, es dependiente del hardware, sólo funciona en un tipo de ordenador y no en otro.

```
;-------------------------------------------------------------------
; Prueba para el PIC 16F876. Programa para recibir datos continuamente de
una señal
; analógica y enviarlos de modo digital en formato USART.
; 3/9/2003 Version
;-------------------------------------------------------------------
;-------------------------------------------------------------------
; RUTINAS A UTILIZAR.
;-------------------------------------------------------------------

     ORG  0x04            ; Comienzo del vector de interrupciones.
     goto Intconv

     ORG  0x05
Recepcion
     BANKSEL    RCREG     ; Seleccionamos el banco 0.
     movf RCREG, W        ; Se lleva al acumulador W el byte recibido.
     movwf dato_REC       ; Se almacena el byte recibido en dato_REC
     RETURN
;-------------------------------------------------------------------
```

Ejemplo rutina lenguaje ensamblador

c. Lenguaje de alto nivel – es independiente de la máquina y, por tanto, portable.

Es más sencillo de aprender ya que es más cercano a nuestro lenguaje natural. Las modificaciones y puestas a punto son más sencillas.

Los programas escritos en un lenguaje de alto nivel necesitan una etapa de traducción al lenguaje máquina que realiza un programa ***compilador***. El programa resultante de la traducción ocupa más memoria y su velocidad de ejecución aumenta en relación a los programas escritos directamente en código máquina (Ejemplos: Pascal, C, C++, Java…).

Otra posible clasificación de los lenguajes de programación puede hacerse en relación al *paradigma de programación* que utilizan:

1. ***Lenguajes imperativos o procedimentales*** – están orientados a instrucciones (Pascal, C)
2. ***Lenguajes orientados a objetos*** – se centran más en los datos y su estructura (SmallTalk, Java, C#, Eiffel)

3. ***Lenguajes declarativos (funcionales y lógicos)*** – se construyen mediante descripciones de funciones matemáticas (Haskell, Lisp) o expresiones lógicas (Prolog)

1.4 Traductores: compiladores e intérpretes

Un ***traductor*** es un programa que recibe un *programa fuente* escrito en un lenguaje de alto nivel y obtiene como salida un programa traducido a código máquina.

Los traductores se dividen en:

a) **Intérpretes** – toman un programa fuente y lo traducen de forma simultánea a su ejecución.

b) **Compiladores** – traducen un programa fuente a lenguaje máquina.

Al proceso de traducción se le denomina en este caso *compilación* y al programa resultado de la compilación se le denomina *programa objeto (*con extensión .**OBJ**). Los compiladores detectan además los errores de sintaxis en el programa fuente.

En la mayoría de los lenguajes compilados, el programa objeto no es directamente ejecutable. Es preciso efectuar el *montaje* o *linkediton* con ayuda del programa *montador* o *enlazador (linker)* para obtener un *programa ejecutable* (**.EXE**).

En este proceso de montaje se enlazan algunas rutinas del sistema o subprogramas compilados separadamente.

1.5 Paradigmas de programación

Un ***paradigma de programación*** es un conjunto de teorías, estándares, que indican la forma de organizar los programas sobre la base de algún modelo conceptual y un lenguaje apropiado que lo soporte.

Además de los paradigmas funcional y lógico (no utilizados ampliamente), los dos paradigmas fundamentales de programación son:

- ***Paradigma imperativo o procedural****
- ***Paradigma orientado a objetos****
- ***Paradigma funcional***
- ***Paradigma lógico***

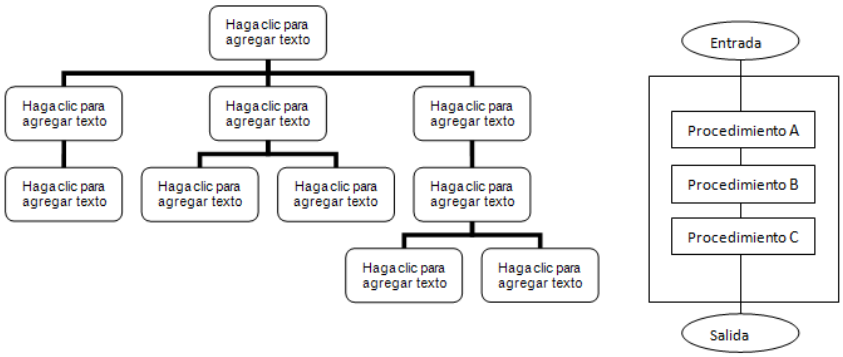
**Ambos utilizan la descomposición (“divide y vencerás”) para afrontar la complejidad de los problemas.*

1.5.1 Paradigma procedural

Ampliamente utilizado en la década de los 70. Los lenguajes Pascal y C son los máximos exponentes de este tipo de programación. Se basa en la descomposición funcional (o algorítmica).

El bloque principal de desarrollo es el ***procedimiento o función***. Los algoritmos que implementan estos procedimientos utilizan para su construcción las tres estructuras básicas de la programación estructurada: ***secuencia, selección e iteración***.

Aquí, una aplicación está formada por una jerarquía de módulos que se organizan en torno a un programa principal. Los datos tienen un papel secundario. La programación estructurada se resume en la expresión: "*Algoritmos + Estructuras de datos = Programas*"



Este paradigma tiene sus ventajas cuando se trata de resolver tareas sencillas.

1.5.2 Paradigma orientado a objetos (programación orientada a objetos - POO)

Tuvo su gran impacto en la década de los 90. Hoy día es ampliamente utilizado en el desarrollo de software, tanto en análisis como en diseño y programación. Vino a resolver los problemas de complejidad y tamaño en el desarrollo de software.

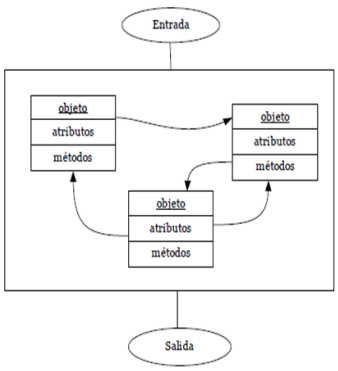
Este paradigma utiliza para resolver la complejidad de un problema la descomposición orientada a objetos. Un programa es un conjunto de objetos que cooperan entre sí para resolver una determinada tarea. El bloque principal de construcción del programa es el ***objeto***, entidad extraída del espacio (o dominio) del problema, con una identidad, unos atributos y un comportamiento. Los datos son el punto central de atención de los programas orientados a objetos. En la POO: ***"Objetos*** + *Flujo de mensajes* = *Programas".*

Este paradigma ofrece ventajas sobre el paradigma procedural:

- modela mejor el mundo real, es más intuitivo, describe el problema en términos similares a los que utiliza la mente humana
- maneja mejor la complejidad del software
- permite la reutilización
- obtiene programas más robustos y estables
- facilita la extensibilidad, la escalabilidad de las aplicaciones

La POO se basa en los siguientes principios:

- *Abstracción* – mecanismo que maneja la complejidad. Permite captar lo esencial ignorando los detalles
- *Encapsulación* – permite mantener oculta la implementación de la abstracción
- *Jerarquía* - la jerarquización ordena las abstracciones, las organiza, simplificando así su desarrollo. Permite la implementación de la herencia y el polimorfismo
- *Modularización* – división del programa en módulos para facilitar su diseño, mantenimiento y reutilización



1.5.3 Paradigma funcional

La programación funcional es un paradigma de programación declarativa basado en la utilización de funciones aritméticas.

Los lenguajes funcionales ofrecen al programador un buen número de recursos que permiten resolver problemas complejos mediante programas pequeños y robustos.

- Lenguajes: LISP, Scheme, Haskell, Scala, Clojure

1.5.4 Paradigma lógico

Se basa en el uso de la lógica como un lenguaje de programación.

Es declarativo: se especifican hechos, reglas y objetivo sin indicar cómo se obtiene éste último a partir de los primeros

- Lenguajes: Prolog, Mercury, Oz

1.6 Calidad de los programas

Todo buen programa debe ser:

- *Correcto* – sin errores y cumpliendo los requisitos que satisfacen el problema
- *Robusto* - capaz de funcionar incluso en situaciones anormales
- *Legible* - claro y fácil de leer
- *Modificable* - fácil de modificar y mantener
- *Eficiente* - utiliza adecuadamente los recursos de la máquina (memoria y tiempo de ejecución)
- *Reutilizable* – fácil de reutilizar, todo o parte, en otros programas
- *Modular* – dividido en partes cada una de ellas resolviendo una determinada tarea

1.7 Lenguajes orientados a objetos

Los lenguajes de programación orientada a objetos son aquellos que soportan las características de la POO. Dentro de ellos se distinguen:

a) ***Lenguajes puros orientados a objetos*** – proceden del lenguaje Simula (Simula 67 fue el primer lenguaje OO diseñado en el año 67).

Trabajan exclusivamente con objetos y clases.

Entre ellos están SmallTalk, Eiffel, Java, C#.

b) ***Lenguajes híbridos*** – están basados en lenguajes procedimentales. Soportan por tanto, la programación procedural (estructurada) y la POO.

Estos lenguajes se construyen a partir de otros ya existentes, como C o Pascal.

Entre ellos destaca: Ada, Modula y Object Pascal proceden del lenguaje Pascal, C++ procede del lenguaje C.

1.8 Desarrollo de software orientado a objetos

El proceso de desarrollo de software es un conjunto de actividades que se necesitan para transformar los requerimientos de un problema en el producto software deseado. Describe un enfoque para la construcción, desarrollo y mantenimiento del software. RUP (*Rational Process Unified* – Proceso Unificado de Rational) es una de las metodologías de desarrollo de software de gran éxito en la construcción de software que utiliza el desarrollo iterativo e incremental y el lenguaje UML.

El desarrollo *iterativo* e *incremental* que utilizan las metodologías de desarrollo de software orientado a objetos consiste en:

- dividir un proyecto en mini proyectos más fáciles de manejar
- cada mini proyecto se denomina *iteración*
- en cada iteración se cubre el ciclo entero de desarrollo de una aplicación informática (análisis OO, diseño OO, programación OO, pruebas e integración y mantenimiento)
- cada iteración genera una versión parcialmente completa del sistema
- las sucesivas iteraciones se construyen unas sobre otras hasta que el sistema se ha completado
- la diferencia entre una y otra iteración se denomina *incremento*

Veamos brevemente las diferentes etapas en el desarrollo OO de una aplicación informática (*Ciclo de desarrollo del software):*

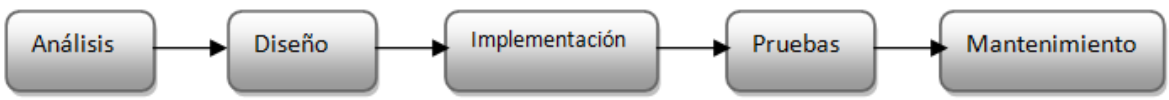
- *Análisis OO* – se investiga el problema y los requisitos en vez de solucionarlo. Se habla de análisis de requisitos o análisis de objetos (estudio de los objetos del dominio). Se presta especial atención a encontrar y describir los objetos (las clases) en el dominio del problema y se construye un modelo (Libro, Biblioteca, Socio… en el sistema de información de una biblioteca).

- *Diseño OO* – hay que encontrar una solución que satisfaga los requisitos, sin realizar todavía la implementación. Se diseñan los objetos (las clases), prestando especial atención a los objetos software y estudiando cómo van a colaborar para satisfacer los requisitos. En el sistema de información de la biblioteca un objeto de la clase Libro podría tener un atributo *título* y un método numeroVecesPrestado*().*

- *Implementación – (Programación OO - codificación)* – Se traduce el diseño a un lenguaje de programación concreto.

- *Pruebas* - se verifica que el producto construido hace lo deseado. Se preparan tests con datos de prueba para comprobar el correcto funcionamiento.

- *Mantenimiento* - actividades que incluyen modificaciones del producto, tanto del código como de la documentación, debido a errores o a la necesidad de mejora o/y adaptación



Ciclo de vida en cascada del software

Para saber más

Puedes consultar información acerca del ciclo de vida del software:

Ciclo de vida del software

1.8.1 El lenguaje UML

UML (*Unified Modeling Language* – Lenguaje Unificado de Modelado) en un lenguaje utilizado para describir modelos. Un *modelo* es una descripción abstracta de un sistema, una representación simplificada que permite comprenderlo y simularlo. UML no es una metodología de desarrollo de software OO sino una notación para especificar, construir, visualizar y documentar los elementos de un sistema de software.

UML define una serie de diagramas o representaciones gráficas de elementos interconectados. Entre otros diagramas, UML permite representar *diagramas de clases*, que muestran la estructura estática de un modelo, es decir, las clases que lo componen y sus relaciones.

Para saber más

UML fue desarrollado a mediado de los 90s por Grady Boch, James Rumbaugh e Ivar Jacobson, en la Rational Software Corp., que ahora forma parte de IBM. Actualmente UML es mantenido por el la Object Management Group (OMG) organización sin fines lucro que promueve la estandarización de las tecnologías orientadas a objetos emitiendo lineamientos y especificaciones como UML.

En términos muy generales UML define una notación que se expresa con diagramas. Para mostrar las diferentes perspectivas del modelado, UML define 9 tipos de diagramas:

- Diagramas de casos de uso
- Diagramas de clases
- Diagramas de componentes
- Diagramas de despliegue
- Diagramas de objetos
- Diagramas de colaboración
- Diagramas de estados y transiciones
- Diagramas de actividades

1.9 El lenguaje de programación JAVA

Java es un lenguaje de programación de alto nivel totalmente orientado a objetos desarrollado por el equipo de James Gosling de Sun Microsystems en 1995. Curiosamente, este lenguaje fue diseñado antes que diese comienzo la era wold wide web, puesto que fue diseñado para dispositivos electrónicos como calculadoras, microondas y la televisión interactiva.

Inicialmente Java se llamó Oak (roble en inglés), aunque tuvo que cambiar de denominación debido a que dicho nombre ya estaba registrado por otra empresa.

El proyecto Green fue el primero en que se aplicó Java, y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. En dicho proyecto aparecía ya ***Duke***, la actual mascota de Java.

Icono de Duke, la mascota de java

Lo cierto es que los primeros proyectos no tuvieron éxito, y Java entró en un letargo olvido.

No fue, hasta la expansión universal de Internet, que se produjo el resurgimiento de Java.

Java es idóneo para desarrollar aplicaciones Internet, para la web, principal motivo por el que ganó rápidamente muchos adeptos, principalmente por su neutralidad respecto a la plataforma de ejecución y porque los programas Java podían ejecutarse dentro de un navegador (los ***applets***).

Sin embargo hay muchas otras características que hacen de Java un lenguaje atractivo cuya evolución ha sido muy rápida.

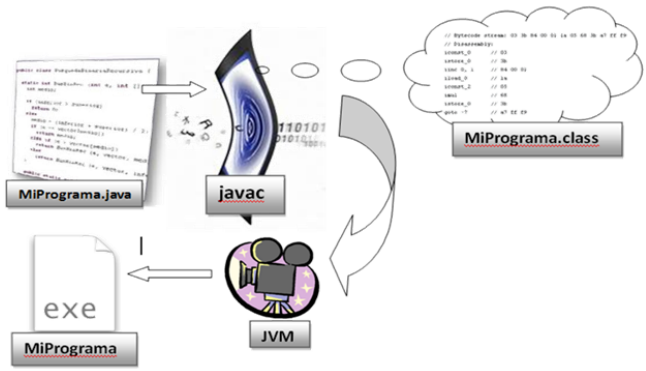
Curiosidad

Pero ¿por qué se llama Java? Aunque no hay un origen 100% oficial sobre el nombre, el logo actual de Java es una taza de café. Esto se debe a que sus creadores amaban tomar café de la isla de Java. También dicen que significa **J**ust **A**nother **V**ague **A**cronym.

Logo oficial de la plataforma JAVA

1.9.1 Características de JAVA

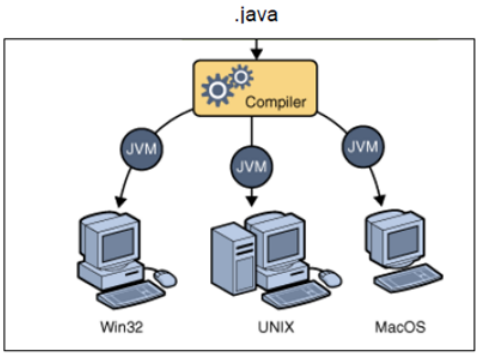
- *Sencillo* - Java es un lenguaje sencillo (en comparación con su predecesor C++) y elegante. Java elimina los punteros de C++, no permite la herencia múltiple. El *garbage collector* (recolector de basura) permite la gestión automática de la memoria dinámica. Proporciona una sintaxis sencilla, elegante, hay pocas construcciones de programa.
- *Orientado a objetos* – Java fue diseñado desde el principio para ser un lenguaje OO, lo que facilita la construcción de software siguiendo el paradigma de la POO.
- *Distribuido* – la adaptabilidad de Java para trabajar en entornos de red es inherente a su arquitectura ya que fue diseñado para ello. Las clases que conforman una aplicación pueden estar ubicadas en distintas máquinas de la red.
- *Robusto* – confiable, no permite construcciones peligrosas (uso de punteros), pone especial énfasis en la verificación temprana de errores (en tiempo de compilación), gestiona excepciones en tiempo de ejecución.
- *Seguro* – proporciona mecanismos de seguridad que protegen el sistema.
- *Interpretado* – Java no funciona como la mayoría de los lenguajes de programación compilados donde el compilador traduce el código fuente al lenguaje máquina concreto del procesador. Java es compilado e interpretado a la vez. Se necesita un intérprete para ejecutar los programas Java.



Los programas fuente escritos en java (**.java*) se compilan (con el compilador *javac*) para obtener ficheros con extensión *.class*. Estos ficheros ya pueden ser distribuidos, no hay proceso de enlace (no se necesita un montador).

Los ficheros **.class* no contienen código máquina comprensible por ningún procesador sino que contiene *bytecodes*, una especie de código de bajo nivel (un lenguaje intermedio) que será interpretado por la máquina virtual de Java. Esta máquina abstracta es la que ha de estar instalada en una máquina real para poder ejecutar la aplicación java. El código de *bytecodes* es el mismo para todos los sistemas.

- *De arquitectura neutral y portable* – ya que Java es interpretado permite que sea independiente de la plataforma en la que se ejecuta. El lema de Java es "*escribe una vez, ejecuta en cualquier lugar*" (*write once, run anywhere*). El programa se escribe y compila una vez y se ejecuta en cualquier plataforma que tenga instalada la JVM.

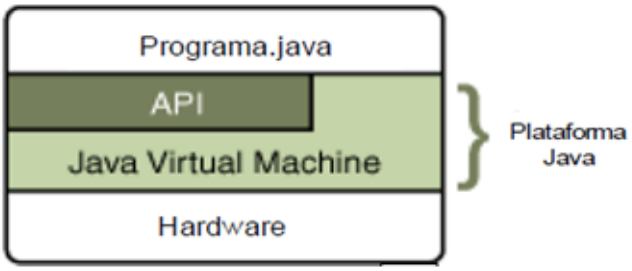


Gracias a la JVM la misma aplicación puede ejecutarse en múltiples plataformas

- *Multihilo – permite la ejecución de varias tareas simultáneamente. Esta característica es especialmente útil en la programación gráfica (GUI – animaciones) y en la programación en red (un servidor puede atender a múltiples clientes al mismo tiempo)*

1.10 La plataforma JAVA

Java no es sólo un lenguaje de programación sino toda una plataforma de desarrollo que, además del lenguaje que da nombre a la plataforma, consta de:

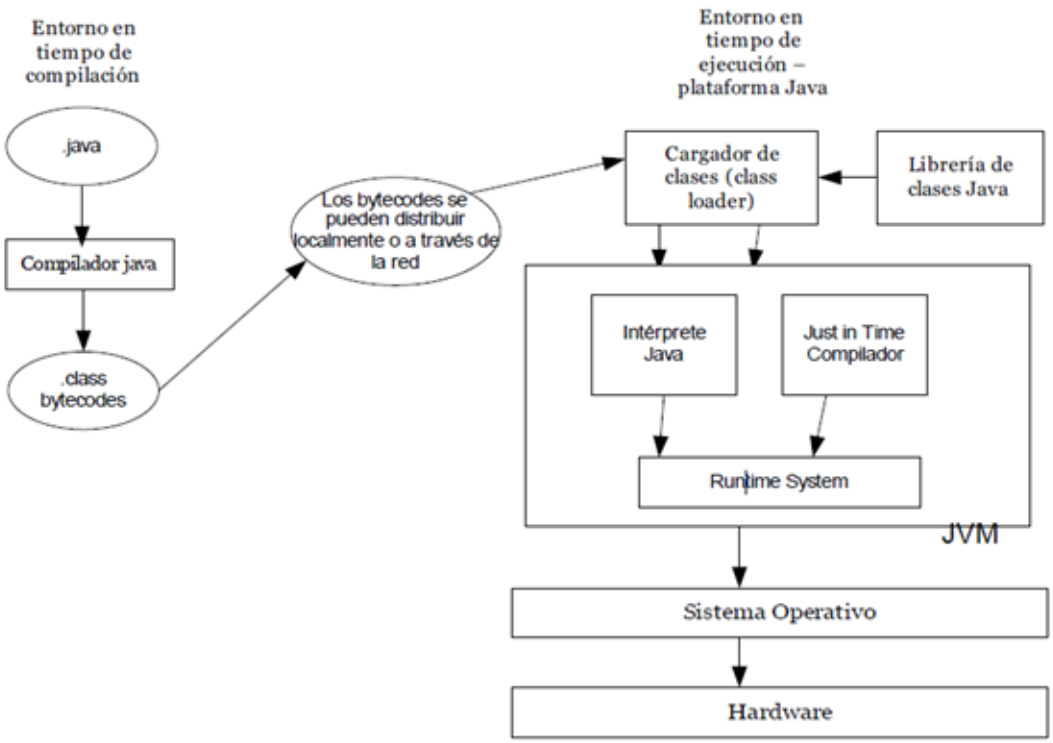


a. La JVM – Java Virtual Machine – la máquina virtual de Java

Es la máquina abstracta (en realidad es software) que lee los ficheros *.class* que contienen *bytecodes* y los traduce a código máquina.

La JVM:

- se requiere para ejecutar cualquier programa Java
- es dependiente de la máquina (JVM para Windows, JVM para Linux.....)
- existe también dentro de los navegadores (para poder ejecutar los *applets*)



Justo antes de ejecutar un programa, Java utiliza un cargador de clases (*class loader*) para ubicar los bytecodes de todas las clases a utilizar por el programa en la memoria del ordenador.

b. La API de Java – Java Application Programming Interface - es una colección de componentes software, clases, que el programador puede incluir en sus programas. La API ofrece

capacidades de todo tipo: gráficas, matemáticas, componentes de red, etc. Todos estos componentes se agrupan en librerías de clases relacionadas (paquetes).

- La API junto con la máquina virtual (JVM) conforman el **JRE (Java Runtime Environment**). Para ejecutar programas Java simplemente (no para desarrollarlos) es suficiente tener instalado el JRE en el sistema.

1.10.1 Afinando la configuración

Para que podamos compilar y ejecutar ficheros Java es necesario que realicemos unos pequeños ajustes en la configuración del sistema. Vamos a indicarle dónde encontrar los ficheros necesarios para realizar las labores de compilación y ejecución, en este caso Javac.exe y Java.exe. así como las librerías contenidas en la API de Java y las clases del usuario.

La variable **PATH**: Como aún no disponemos de un IDE (Integrated Development Environment - Entono Integrado de Desarrollo) la única forma de ejecutar programas es a través de línea de comandos. Pero sólo podremos ejecutar programas directamente si la ruta hacia ellos está indicada en la variable PATH del ordenador. Es necesario que incluyamos la ruta hacia estos programas en nuestra variable PATH. Esta ruta será el lugar donde se instaló el JDK hasta su directorio bin.

Para saber más

Si deseas conocer más sobre la configuración de variables en entornos Windows y Linux, te ofrecemos los siguientes enlaces:

¿Cómo establecer la variable PATH en Windows?

¿Cómo establecer la variable PATH en Linux?

PATH and CLASSPATH (inglés)

La variable **CLASSPATH**: esta variable de entorno establece dónde buscar las clases o bibliotecas de la API de Java, así como las clases creadas por el usuario. Es decir, los ficheros .dass que se obtienen una vez compilado el código fuente de un programa escrito en Java. Es posible que en dicha ruta existan directorios y ficheros comprimidos en los formatos zip o jar que pueden ser utilizados directamente por el JDK, conteniendo en su interior archivos con extensión class.

(Por ejemplo: C:\Program Files\Java\jdk\1.8.0_161\bin)

Si no existe la variable CLASSPATH debes crearla, para modificar su contenido sigue el mismo método que hemos empleado para la modificación del valor de la variable PATH anteriormente descrito. Ten en cuenta que la ruta que debes incluir será el lugar donde se instaló el JDK hasta su directorio lib.

(Por ejemplo: C:\Program Files\Java\jdk\1.8.0_161\lib)

1.11 Herramientas de desarrollo (JDK o Java SDK – Java Software Development Kit)

Además del JRE para ejecutar programas si queremos desarrollarlos Java proporciona un conjunto de herramientas de desarrollo tales como:

- **javac** – compilador a bytecodes
- **javadoc** – generador de documentación
- **java** –llamada a la máquina virtual (lanzador de aplicaciones Java)
- **jdb** – depurador de consola
- **jar** – herramienta para crear archivos *.jar*

Estas herramientas junto con el JRE constituyen el J*ava SE Development Kit (JDK).*

El JDK puede descargarse desde la página de Sun, http://java.sun.com

1.12 Versiones de JAVA

Java ha evolucionado muy rápidamente desde su aparición en 1995.

La versión inicial fue **Java 1.0**. **Java 1.1** introdujo cambios significativos. A partir de aquí la numeración de las nuevas versiones es algo confusa.

La siguiente versión fue **Java 2** v.1.2.

Java 2 v. 1.5, conocida también como **Java 5.0**, supuso un cambio cualitativo en cuanto a introducción de mejoras en el lenguaje (colecciones genéricas, bucle ***for-each***, enumerados, autoboxing/unboxing....).

Java 6 (lanzada en Diciembre 2006), fue la primera versión de Java para trabajar con Windows Vista. Se mejoró el rendimiento de ejecución de las aplicaciones, ciertos elementos de la edición empresarial (J2EE) se han trasladado a la versión de escritorio (J2SE), así elJ2SE incorpora un servidor web de desarrollo, el paquete Swing ha sido optimizado, incluye soporte para lenguajes de script (por ejemplo JavaScript)

Java SE 7 Su lanzamiento fue en julio de 2011.

- Soporte para XML dentro del propio lenguaje.
- Un nuevo concepto de superpaquete.Soporte para closures.
- Introducción de anotaciones estándar para detectar fallos en el software.

No oficiales:

- NIO2.
- Java Module System.
- Java Kernel.
- Nueva API para el manejo de Días y Fechas, la cual reemplazara las antiguas clases Date y Calendar.
- Posibilidad de operar con clases BigDecimal usando operandos.

Java SE 8 — lanzada en marzo de 2014. Cabe destacar:

- Incorpora de forma completa la librería JavaFX.
- Diferentes mejoras en seguridad.
- Diferentes mejoras en concurrencia.
- Añade funcionalidad para programación funcional mediante expresiones Lambda.
- Mejora la integración de JavaScript.
- Nuevas API para manejo de fechas y tiempo (date - time).

Java SE 9 - Si en Java 8 la característica más destacada fue la incorporación al lenguaje de las ***lambdas*** y los ***streams*** en Java 9 la característica que más destaca es la definición de los módulos que proporciona varios importantes beneficios

Java SE 10 - Esta es la primera versión del nuevo mecanismo de publicación de versiones más frecuente y con una cadencia, más o menos, fija. Así que las novedades, para ser un cambio de versión mayor, no son muchas. No es tampoco una de las importantes versiones LTS con soporte extendido. Algunas de las novedades de Java 10:

- Inferencia de tipos para variables locales
- Cambios internos
    - Java 10 introduce también algunos cambios internos como la paralelización del colector de basura G1 o la compartición de datos a nivel aplicación (*Application Class-Data Sharing*).

Java SE 11 - El cambio fundamental en Java 11, es sin duda JavaFX que ha sido eliminado de la implementación estándar de la tecnologí, para convertirse a pasar en un módulo independiente. Y con muchas novedades que abarcan posibilidades que permiten mejorar el desempeño y la seguridad:

- Acceso a controles de acceso basados en Nest (nestmates), que permiten la implementación de las clases internas y elimina la necesidad de insertar métodos por parte de los compiladores.
- La actualización de las constantes dinámicas de archivo de clase a nuevos enfoques centrados en la plataforma y el rendimiento.
- Se ofrece soporte experimental de ZGC, el nuevo recolector de basura diseñado para tiempos de pausa inferiores a 10 milisegundos y cuyo objetivo que la penalización no supere el 15% del rendimiento.
- La disponibilidad de Flight Recorder, un framework de recolección de datos de bajos recursos para resolver problemas detectados en las aplicaciones de Java y la herramienta HotSpot JVM.
- Una nueva biblioteca completamente estándar HTTP se encarga de estandarizar la API incubada y habilita el soporte para permitir flujos basados en HTTP/1.1 y HTTP/2.
- Implementación de TLS 1.3, cuyo estándar ha sido aprobado este año.
- La sintaxis de variable local para los Parámetros Lambda, actualizando así la sintaxis Lambda para usar la inferencia de tipo de variable introducida en Java 10.
- En la distribución de GNU/Linux, se cargarán por defecto las bibliotecas de GTK3 en lugar de las de GTK2. Un movimiento hecho para adaptarse a los entornos de escritorios basados en GTK más modernos en GNU/Linux, posiblemente con GNOME en mente.
- De manera oficial se ha eliminado características como Web Start, se eliminó los Applets que ahora son obsoletos cuyas funciones están ya ampliamente cubiertas por HTML, CSS y JavaScript y el módulo JavaFX.

Han sido desarrolladas diferentes ediciones de Java para diferentes tipos de aplicaciones:

- **Java Platform Standard Edition (Java SE) -** es Java SDK para escribir, desarrollar y ejecutar applets y aplicaciones en Java
- **Java Platform Edition Empresarial (Java EE) -** Java Edición Empresarial, versión ampliada de la Java SE que incluye las API necesarias para construir aplicaciones para arquitectura distribuidas multicapa.
- **Java Platform MicroEdition (Java ME) –** edición especial para dispositivos móviles.

Java SE 12 - Las características destacadas de Java 12 son la incorporación de forma experimental las expresiones ***switch*** y mejoras en el recolector de basura para mayor rendimiento.

Java SE 13 - Incorpora algunas nuevas características interesantes que mejoran un facilitan la lectura del código, entre las mas destacadas están los bloques de texto y las expresiones ***switch*** mejoradas.

Java SE 14 - Esta nueva versión de **Java SE 14 se clasifica como un período de soporte regular** para el que se publicarán actualizaciones antes de la próxima versión ya que la actual rama estable LTS «Java SE 11», contara con actualizaciones hasta 2026, mientras que la rama anterior de Java 8 LTS será compatible hasta diciembre de 2020.

Dentro de las principales novedades de esta versión **se menciona el soporte experimental de *instanceof*, *record* y** soporte experimental par**a bloques de texto se ha ampliado.**

- **instanceof:** es utilizado para la coincidencia de patrones en el operador que permite determinar de inmediato la variable local para acceder al valor verificado.
- **record:** proporciona una forma compacta para definir clases, evitando la definición explícita de varios métodos de bajo nivel, como ***equals()***, ***hashCode()*** y ***toString()***, en los casos en que los datos se almacenan solo en campos.
- **La ampliación en los bloques de texto:** proporciona una nueva forma de literales de cadena que permite incluir datos de texto de varias líneas en el código fuente sin usar caracteres de escape y preservar el formato de texto original en el bloque. El encuadre de

bloque se realiza con tres comillas dobles.
En Java 14, los bloques de texto admiten la secuencia de escape «\s» para definir un espacio único y «\» para concatenar con la siguiente línea.

También podremos encontrar que **se implementó una versión preliminar de la utilidad jpackage,** que **permite crear paquetes para aplicaciones Java autónomas**. La utilidad se basa en javapackager de JavaFX y permite crear paquetes en formatos nativos de varias plataformas (msi y exe para Windows, pkg y dmg para macOS, deb y rpm para Linux).

Por otro lado se menciona qu**e se ha agregado un nuevo mecanismo de asignación de memoria al recolector de basura G1**, teniendo en cuenta las características específicas de trabajar en sistemas grandes utilizando la arquitectura NUMA. El nuevo asignador de memoria se habilita utilizando el indicador «+ XX: + UseNUMA» y puede aumentar significativamente el rendimiento en los sistemas NUMA.

Se implementó una **versión preliminar de la API de acceso a memoria externa**, que **permite a las aplicaciones Java acceder de manera segura y eficiente a áreas de memoria fuera** del montón de Java mediante la manipulación de nuevas abstracciones de MemorySegment, MemoryAddress y MemoryLayout.

Los ports para los procesadores Solaris OS y SPARC se declaran obsoletos con la intención de eliminar estos en el futuro. La transferencia de estos ports a los obsoletos permitirá a la comunidad acelerar el desarrollo de nuevas funciones OpenJDK sin perder el tiempo manteniendo las funciones específicas de Solaris y SPARC.

Ademas **se eliminó el recolector de basura CMS** (Concurrent Mark Sweep), que fue obsoleto hace dos años y no se acompañó. Por otra parte, se declaró uso obsoleta de una combinación de algoritmos de recolección de basura y ParallelScavenge SerialOld.

De los demás cambios que se mencionan:

- Se eliminaron herramientas y API para comprimir archivos JAR utilizando el algoritmo Pack200.
- API agregada para rastrear eventos JFR sobre la marcha (JDK Flight Recorder), por ejemplo, para organizar el monitoreo continuo.
- Se ha agregado el módulo jdk.nio.mapmode, que ofrece nuevos modos (READ_ONLY_SYNC, WRITE_ONLY_SYNC) para crear buffers de bytes mapeados (MappedByteBuffer) que hacen referencia a la memoria no volátil (NVM).

La siguiente tabla muestra el número de la versión, el nombre y la fecha de liberación de la misma:

Versión	Nombre	Fecha de liberación
JDK 1.0	Oak	Enero 1996
JDK 1.1	(none)	Febrero 1997
J2SE 1.2	Playground	Diciembre 1998
J2SE 1.3	Kestrel	Mayo 2000
J2SE 1.4	Merlin	Febrero 2002
J2SE 5.0	Tiger	Septiembre 2004
Java SE 6	Mustang	Diciembre 2006
Java SE 7	Dolphin	Julio 2011

Java SE 8	Marzo 2014
Java SE 9	21 de septiembre de 2017
Java SE 10	20 de marzo de 2018
Java SE 11	25 de septiembre de 2018
Java SE 12	Marzo 2019
Java SE 13	Septiembre 2019
Java SE 14	Marzo 2020
Java SE 15	Septiembre 2020
Java SE 16	Marzo 2021
Java SE 17	Septiembre 2021

1.13 Tipos de aplicaciones JAVA

Aplicaciones normales – programas autónomos que se ejecutan en un ordenador que tiene la JVM. Pueden producir salida de texto en consola o utilizar un interface gráfico (GUI).

Applets – es un programa diseñado para ejecutarse dentro de un navegador compatible con Java. La salida es gráfica.

Aplicaciones web – aplicación que se ejecuta en un servidor. Las aplicaciones web utilizan tecnologías llamadas ***servlets*** y JSP – ***Java Server Pages***. Las aplicaciones web interactúan normalmente con bases de datos.

Servlets - Son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones web que interactúen con los clientes.

Midlets - Aplicaciones creadas en Java para su ejecución en sistemas de propósito simple o dispositivos móviles. Los juegos Java creados para teléfonos móviles son midlets. Son programas creados para dispositivos embebidos (se dedican a una sola actividad)

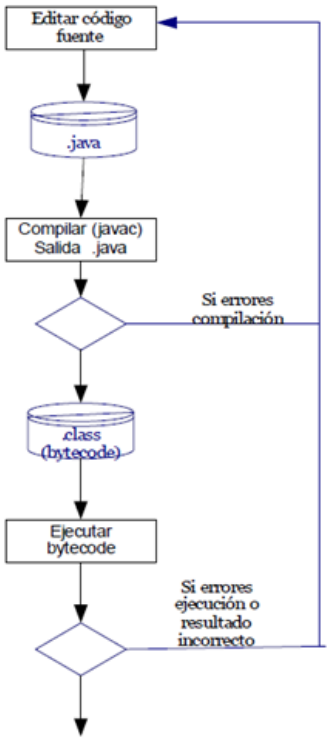
1.14 Editar, compilar y ejecutar un programa Java

El único software que se necesita para desarrollar y ejecutar un programa Java es el JDK y un editor de textos.

El fichero fuente se escribe con el editor y se salva como un fichero con extensión *.java*.

El compilador java, *javac*, se invoca desde la línea de comandos para producir un fichero con extensión *.class*.

El fichero *.class* se invoca también desde línea de comandos llamando a *java*, la máquina virtual.



1.15 Entornos integrados de desarrollo (IDE)

El JDK incluye un conjunto de herramientas de línea de comandos para compilar y ejecutar su código Java, que incluye una copia completa del JRE. Aunque ciertamente se pueden usar estas herramientas para desarrollar aplicaciones, la mayoría de los desarrolladores valoran la funcionalidad adicional, la gestión de tareas y la interfaz visual de un IDE.

Existen numerosos entornos de desarrollo (IDE – Integrated Development Environment) que facilitan la tarea de editar, compilar y ejecutar un programa Java.

Entre otros: Eclipse, Dr. Java, JCreator, Borland JBuilder.

El curso lo iniciaremos usando BlueJ, y a medida que vayamos asentando los conceptos, podremos empezar a usar cualquier otro IDE comercial, como NetBeans o Eclipse.

a) **BlueJ** - es un excelente entorno de desarrollo diseñado por las universidades de Kent (Reino Unido), Deakin (Australia) y Southern (Dinamarca) específicamente para la enseñanza y aprendizaje de la POO en Java. Incluye herramientas educacionales tales como visualización e interacción de objetos que permiten aprender desde el inicio los conceptos de la POO de una manera sencilla. Es gratuito. Se puede descargar desde http://www.bluej.org

b) **NetBeans** – es un entorno de desarrollo integrado profesional gratuito de código abierto. Incluye una gran variedad de herramientas que ayudan en el desarrollo de cualquier tipo de aplicación Java: aplicaciones web, applets, aplicaciones con GUI, JSP….

Hay una versión especial NetBeans/BlueJ que permite abrir y trabajar con proyectos que han sido creados en BlueJ. BlueJ es un entorno para aprender Java pero para desarrollar grandes proyectos es preciso hacer una transición hacia un IDE más profesional como puede ser NetBeans. Con esta nueva edición esta transición desde BlueJ hacia NetBeans puede ser muy sencilla.

NetBeans/BlueJ Edition en una variante de la versión standard de NetBeans con dos diferencias significativas:

- puede abrir y trabajar con proyectos creados en BlueJ
- la interface standard de NetBeans ha sido simplificada

Puede descargarse de la página de Java la última versión del JDK junto con NetBeans.

Actualmente el equipo de desarrollo de BlueJ ha elaborado junto al equipo de NetBeans un plugin que puede añadirse a NetBeans. De esta forma es posible desarrollar y ejecutar proyectos simultáneamente en BlueJ y en NetBeans.

Más información del plugin en el sitio http://www.bluej.org/netbeans/

(Tanto el plugin como la versión NetBeans/BlueJ son versiones bastante antiguas. Todavía lo podremos descargar de la red, (quizá de páginas no oficiales) pero las versiones más modernas y de más fácil acceso no permiten abrir archivos BlueJ)

c) **Eclipse** - Es un IDE de código abierto popular para el desarrollo Java. Maneja las tareas básicas, tales como la compilación de códigos y la configuración de un entorno de depuración, para que pueda centrase en escribir y probar códigos. Además, puede usar Eclipse para organizar archivos de códigos de origen en proyectos, compilar y probar esos proyectos y almacenar archivos de proyectos en cualquier cantidad de repositorios de origen.

Más información en el sitio web: https://eclipse.org/home/index.php