C.F.G.S. DESARROLLO DE APLICACIONES WEB

MÓDULO

PROGRAMACIÓN

Unidad 1:

Introducción a la Programación.

ÍNDICE DE CONTENIDOS

OBJETIVOS

En esta primera unidad se comienza con los conceptos generales y fundamentales de la programación. Se estudian los diferentes enfoques a la hora de desarrollar software, las fases genéricas del proceso de programación, el ciclo de vida del software, así como una descripción de los diferentes lenguajes de programación.

Se conocerá el lenguaje de programación que se va a emplear para desarrollar todos los contenidos de este módulo profesional, el lenguaje Java. Tras describir sus particularidades, se aprenderá cómo crear, compilar y ejecutar programas escritos en Java, de manera básica.

Finalizando la unidad se realizará un análisis y selección de un entorno de desarrollo integrado fiable, gratuito y profesional como es Eclipse, cuyo uso facilitará la labor al futuro programador.

1.- Introducción.

¿Cuántas acciones de las que has realizado hoy, crees que están relacionadas con la programación? Hagamos un repaso de los primeros instantes del día:

- ✓ Te ha despertado la alarma de tu teléfono móvil o radio-despertador.
- ✓ Has preparado el desayuno utilizando el microondas.
- ✓ Mientras desayunabas has visto u oído las últimas noticias a través de tu receptor de televisión digital terrestre.
- ✓ Te has vestido y puede que hayas utilizado el ascensor para bajar al portal y salir a la calle, etc.

Quizá no es necesario que continuemos más para darnos cuenta de que casi todo lo que nos rodea, en alguna medida, está relacionado con la programación, los programas y el tratamiento de algún tipo de información.

El volumen de datos que actualmente manejamos y sus innumerables posibilidades de tratamiento constituyen un vasto territorio en el que los programadores tienen mucho que decir.

En esta primera unidad realizaremos un recorrido por los conceptos fundamentales de la programación de aplicaciones. Iniciaremos nuestro camino conociendo con qué vamos a trabajar, qué técnicas podemos emplear y qué es lo que pretendemos conseguir. Continuando con el análisis de las diferentes formas de programación existentes, identificaremos qué fases conforman el desarrollo de un programa, avanzaremos detallando las características relevantes de cada uno de los lenguajes de programación disponibles, para posteriormente, realizar una visión general del lenguaje de programación Java. Finalmente, tendremos la oportunidad de conocer con qué herramientas podríamos desarrollar nuestros programas, escogiendo entre una de ellas para ponernos manos a la obra utilizando el lenguaje Java.

2.- Programas y programación.

2.1.- Buscando una solución.

Generalmente, la primera razón que mueve a una persona hacia el aprendizaje de la programación es utilizar el ordenador como herramienta para resolver problemas concretos. Como en la vida real, la búsqueda y obtención de una solución a un problema determinado, utilizando medios informáticos, se lleva a cabo siguiendo unos pasos fundamentales.

En la siguiente tabla podemos ver estas analogías.

Resolución de problemas	
En la vida real...	**En Programación...**
Observación de la situación o problema.	**Análisis del problema:** requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle.
Pensamos en una o varias posibles soluciones	**Diseño o desarrollo de algoritmos:** procedimiento paso a paso para solucionar el problema dado.
Aplicamos la solución que estimamos más adecuada.	**Resolución del algoritmo elegido en la computadora:** consiste en convertir el algoritmo en programa, ejecutarlo y comprobar que soluciona verdaderamente el problema.

Qué virtudes debería tener nuestra solución?

- ✓ **Corrección y eficacia:** si resuelve el problema adecuadamente.
- ✓ **Eficiencia:** si lo hace en un tiempo mínimo y con un uso óptimo de los recursos del sistema.

Para conseguirlo, cuando afrontemos la construcción de la solución tendremos que tener en cuenta los siguientes conceptos:

1.- **Abstracción:** se trata de realizar un análisis del problema para descomponerlo en problemas más pequeños y de menor complejidad, describiendo cada uno de ellos de manera precisa. Divide y vencerás, esta suele ser considerada una filosofía general para resolver problemas y de aquí que su nombre no sólo forme parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.

2.- **Encapsulación:** consiste en ocultar la información para poder implementarla de diferentes maneras sin que esto influya en el resto de elementos.

3.- **Modularidad:** estructuraremos cada parte en módulos independientes, cada uno de ellos tendrá su función correspondiente.

Citas Para Pensar

Roger Pressman: "El comienzo de la sabiduría para un ingeniero de software es reconocer la diferencia entre hacer que un programa funcione y conseguir que lo haga correctamente."

2.2.- Algoritmos y programas.

Después de analizar en detalle el problema a solucionar, hemos de diseñar y desarrollar el algoritmo adecuado. Pero, **¿Qué es un algoritmo?**

> **Algoritmo:** Secuencia ordenada de pasos, descrita sin ambigüedades, que conducen a la solución de un problema dado.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

La diferencia fundamental entre **algoritmo** y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado **lenguaje de programación** para que puedan ser ejecutados en el ordenador y así obtener la solución.

> **Programa:** Es implementar el algoritmo en un determinado **lenguaje de programación** concreto para que pueda ser ejecutado en un ordenador.

> **Lenguaje de programación:** Es una forma de expresar un algoritmo que introducido en el ordenador será capaz de ejecutarlo

El diseño de los algoritmos será una tarea que necesitará de la creatividad y conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a algoritmos diferentes, igualmente válidos.

En esencia, todo problema se puede describir por medio de un algoritmo y las características fundamentales que éstos deben cumplir son:

- ✓ Debe ser **preciso** e indicar el orden de realización paso a paso.
- ✓ Debe estar **definido**, si se ejecuta dos o más veces, debe obtener el mismo resultado cada vez.
- ✓ Debe ser **finito**, debe tener un número finito de pasos.

Pero cuando los problemas son complejos, es necesario descomponer éstos en subproblemas más simples y, a su vez, en otros más pequeños. Estas estrategias reciben el nombre de **diseño descendente** o **diseño modular** (**top-down design**). Este sistema se basa en el lema **divide y vencerás**.

Para representar gráficamente los algoritmos que vamos a diseñar, tenemos a nuestra disposición diferentes herramientas que ayudarán a describir su comportamiento de una forma precisa y genérica, para luego poder codificarlos con el lenguaje que nos interese. Entre otras tenemos:

- ✓ **Diagramas de flujo:** Esta técnica utiliza símbolos gráficos para la representación del algoritmo. Suele utilizarse en las fases de análisis.
  - **De sistema**: Flujo de datos.
  - **Ordinograma:** Secuencia de acciones.
- ✓ **Pseudocódigo:** Esta técnica se basa en el uso de palabras clave en lenguaje natural, **constantes**, **variables**, otros objetos, instrucciones y estructuras de programación que expresan de forma escrita la solución del problema. Es la técnica más utilizada actualmente.
- ✓ **Tablas de decisión:** En una tabla son representadas las posibles condiciones del problema con sus respectivas acciones. Suele ser una técnica de apoyo al pseudocódigo cuando existen situaciones condicionales complejas.

Ver Anexo I

3.- Paradigmas de la programación.

¿Cuántas formas existen de hacer las cosas? Supongo que estarás pensando: varias o incluso, muchas. Pero cuando se establece un patrón para la creación de aplicaciones nos estamos acercando al significado de la palabra paradigma.

Paradigma de programación: es un modelo básico para el diseño y la implementación de programas. Este modelo determinará como será el proceso de diseño y la estructura final del programa.

El paradigma representa un enfoque particular o filosofía para la construcción de software. Cada uno tendrá sus ventajas e inconvenientes, será más o menos apropiado, pero no es correcto decir que exista uno mejor que los demás.

Resumen de los diferentes paradigmas de programación:

- **Programación Declarativa**: Se considera opuesta a la programación imperativa, se basa en el desarrollo de programas realizando una especificación o "declaración" de un conjunto de condiciones, proposiciones, afirmaciones, restricciones y transformaciones que describen el problema y detallan su solución. Las sentencias que se utilizan, lo que hacen es **describir el problema** que se quiere solucionar, pero no las instrucciones necesarias para llevarlo a cabo.

  - **Programación Funcional**: Considera al programa como una función matemática, donde el dominio representaría el conjunto de todas las entradas posibles (inputs) y el rango sería el conjunto de todas las salidas posibles (outputs). La forma en que funciona puede ser entendida como una caja negra, esta tiene una serie de entradas, un procesamiento interno y una salida.
    No existe el concepto de variable, además el funcionamiento de cada función es independiente del resto.
    El lenguaje **LIPS** es un ejemplo de este paradigma de programación

  - **Programación lógica**: En este paradigma, se especifica **que hacer y no como hacerlo**. Se utiliza en inteligencia artificial
    El lenguaje **Prolog** es un ejemplo claro que se ajusta a este paradigma.

- **Programación Imperativa**: Consiste en una serie de comandos que una computadora ejecutará. Estos comandos detallan de forma clara y especifica el cómo hacer las cosas y llevaran al programa a través de distintos estados. Utiliza variables, tipos de datos, expresiones y estructuras de control de flujo del programa.

- **Programación convencional**: También llamada, no estructurada, no existían en los lenguajes de programación iniciales herramientas que permitieran estructurar el código para su modificación o reutilización. Los programas eran líneas y líneas de código que hacían difícil su lectura o modificación. Todo estaba basado en instrucciones de salto (goto) que modifican el flujo del programa. Un programa era un único archivo, eran difíciles de mantener y difíciles de representar.

- **Programación Estructurada**: Nació para solventar los problemas y limitaciones de la programación convencional. Permite utilizar estructuras que facilitan la modificación, reutilización y lectura del código. Permite el uso del concepto de función (programación modular), que agrupan código para la realización de una determinada tarea. Estas funciones pueden ser utilizadas en cualquier parte del programa, el número de veces que se necesiten.
  Permite una representación más clara del funcionamiento de los programas mediante diagramas, en los que se representan estructuras condicionales, bucles, etc. Todo se ve como módulos que pueden colaborar entre ellos. Este tipo de programación tenía una serie de limitaciones que provocan la aparición de otros paradigmas.

- **Programación Orientada a Objetos**: El mundo se ve desde el punto de vista de los objetos que hay en él. Los objetos tienen características (propiedades) y con ellos pueden realizarse acciones (métodos). En esta filosofía se busca encontrar los objetos, en vez de las funciones como en programación estructurada.
  Está basada en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento.
  Lenguajes como Java, C, C++, PHP,.. son claros ejemplos que se basan en este paradigma.

- **Visual, orientada a aspectos, orientada a eventos,…:** Son un conjunto de paradigmas que pueden ser relevantes, aunque gran parte de ellos incluyen los conceptos desarrollados en el resto de paradigmas anteriores.

Como se puede apreciar, existen múltiples paradigmas, incluso puede haber lenguajes de programación que no se clasifiquen únicamente dentro de uno de ellos. Un lenguaje como **Smalltalk** es un lenguaje basado en el paradigma orientado a objetos. El lenguaje de programación **Scheme**, en cambio, soporta sólo programación funcional. **Python**, soporta múltiples paradigmas.

PARA SABER MÁS

En el siguiente enlace encontrarás información adicional sobre los diferentes paradigmas de programación.

Paradigmas de programación y lenguajes

Diferencias entre paradigmas de programación

¿Cuál es el objetivo que se busca con la aplicación de los diferentes enfoques? Fundamentalmente, reducir la dificultad para el mantenimiento de las aplicaciones, mejorar el rendimiento del programador y, en general, mejorar la productividad y calidad de los programas.

4.- Fases de la programación.

Sea cual sea el estilo que escojamos a la hora de automatizar una determinada tarea, debemos realizar el proceso aplicando un método a nuestro trabajo. Es decir, sabemos que vamos a dar solución a un problema, aplicando una filosofía de desarrollo y lo haremos dando una serie de pasos que deben estar bien definidos.

El proceso de creación de software puede dividirse en diferentes fases:

- **Fase de resolución del problema.**
- **Fase de implementación.**
- **Fase de explotación y mantenimiento.**

A continuación, analizaremos cada una de ellas.

4.1.- Resolución del problema.

Para el comienzo de esta fase, es necesario que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. A su vez, la fase de resolución del problema puede dividirse en dos etapas:

- **Análisis.**

  Por lo general, el análisis indicará la especificación de requisitos que se deben cubrir. Los contactos entre el analista/programador y el cliente/usuario serán numerosos, de esta forma podrán ser conocidas todas las necesidades que precisa la aplicación. Se especificarán los procesos y estructuras de datos que se van a emplear. La creación de prototipos será muy útil para saber con mayor exactitud los puntos a tratar.

El análisis inicial ofrecerá una idea general de lo que se solicita, realizando posteriormente sucesivos refinamientos que servirán para dar respuesta a las siguientes cuestiones:

- ✓ ¿Cuál es la información que ofrecerá la resolución del problema?
- ✓ ¿Qué datos son necesarios para resolver el problema?

La respuesta a la primera pregunta se identifica con los resultados deseados o las salidas del problema. La respuesta a la segunda pregunta indicará qué datos se proporcionan o las entradas del problema.

En esta fase debemos aprender a analizar la documentación de la empresa, investigar, observar todo lo que rodea el problema y recopilar cualquier información útil.

Ejercicio Resuelto

Realizar el análisis del siguiente problema:

"Leer el radio de un círculo y calcular e imprimir su superficie y circunferencia."

Solución

- La entrada:
  - ✓ Variable RADIO: será el radio del círculo.
  - ✓ Estas variables RADIO, SUPERFICIE y CIRCUNFERENCIA podrán ser de tipo real (números con parte entera y parte decimal, por ejemplo: 3,57)
- Las salidas:
  - ✓ Variable de salida SUPERFICIE: será la superficie del círculo. (¿Te acuerdas? El número Pi por el radio al cuadrado).
  - ✓ Variable de salida CIRCUNFERENCIA: será la longitud de la circunferencia del círculo. (¿Y de ésta? Dos por el número Pi y por el radio)

■ **Diseño.**

En esta etapa se convierte la especificación realizada en la fase de análisis en un diseño más detallado, indicando el comportamiento o la secuencia lógica de instrucciones capaz de resolver el problema planteado. Estos pasos sucesivos, que indican las instrucciones a ejecutar por la máquina, constituyen lo que conocemos como algoritmo.

Consiste en plantear la aplicación como una única operación global, e ir descomponiéndola en operaciones más sencillas, detalladas y específicas. En cada nivel de refinamiento, las operaciones identificadas se asignan a módulos separados.

Hay que tener en cuenta que antes de pasar a la implementación del algoritmo, hemos de asegurarnos que tenemos una solución adecuada. Para ello, todo diseño requerirá de la realización de la **prueba o traza** del programa.

Este proceso consistirá en un seguimiento paso a paso de las instrucciones del algoritmo utilizando datos concretos. Si la solución aportada tiene errores, tendremos que volver a la fase de análisis para realizar las modificaciones necesarias o tomar un nuevo camino para la solución. Sólo cuando el algoritmo cumpla los requisitos y objetivos especificados en la fase de análisis se pasará a la fase de implementación.

4.2.- Implementación.

Si la fase de resolución del problema requiere un especial cuidado en la realización del análisis y el posterior diseño de la solución, la fase de implementación cobra también una especial relevancia. Llevar a la realidad nuestro algoritmo implicará cubrir algunas etapas más que se detallan a continuación.

- **Codificación o construcción**

Esta etapa consiste en transformar o traducir los resultados obtenidos a un determinado lenguaje de programación. Para comprobar la calidad y estabilidad de la aplicación se han de realizar una serie de pruebas que comprueben las funciones de cada módulo (pruebas unitarias), que los módulos funcionan bien entre ellos (pruebas de interconexión) y que todos funcionan en conjunto correctamente (pruebas de integración).

Cuando realizamos la traducción del algoritmo al lenguaje de programación debemos tener en cuenta las reglas gramaticales y la sintaxis de dicho lenguaje. Obtendremos entonces el código fuente, lo que normalmente conocemos por programa.

Pero para que nuestro programa comience a funcionar, antes debe ser traducido a un lenguaje que la máquina entienda. Este proceso de traducción puede hacerse de dos formas, compilando o interpretando el código del programa.

Compilación: Es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje que la máquina es capaz de interpretar.

Compilador: programa informático que realiza la traducción. Recibe el código fuente, realiza un análisis lexicográfico, semántico y sintáctico, genera un código intermedio no optimizado, optimiza dicho código y finalmente, genera el código objeto para una plataforma específica.

Intérprete: programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.

Una vez traducido, sea a través de un proceso de compilación o de interpretación, el programa podrá ser ejecutado.

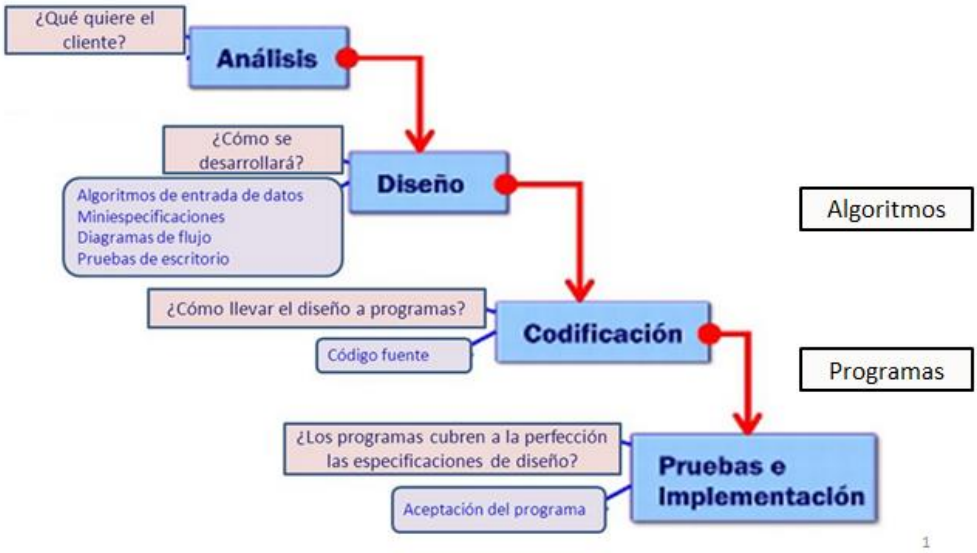
- **Prueba de ejecución y validación.**

  Para esta etapa es necesario implantar la aplicación en el sistema donde va a funcionar, debe ponerse en marcha y comprobar si su funcionamiento es correcto. Utilizando diferentes datos de prueba se verá si el programa responde a los requerimientos especificados, si se detectan nuevos errores, si éstos son bien gestionados y si la interfaz es amigable. Se trata de poner a prueba nuestro programa para ver su respuesta en situaciones difíciles.

  Mientras se detecten errores y éstos no se subsanen no podremos avanzar a la siguiente fase. Una vez corregido el programa y testeado se documentará mediante:

  - ✓ **Documentación interna:** Encabezados, descripciones, declaraciones del problema y comentarios que se incluyen dentro del código fuente.
  - ✓ **Documentación externa:** Son los manuales que se crean para una mejor ejecución y utilización del programa.

Esquema-resumen



4.3.- Explotación.

Cuando el programa ya está instalado en el sistema y está siendo de utilidad para los usuarios, decimos que se encuentra en fase de explotación.

Periódicamente será necesario realizar evaluaciones y, si es necesario, llevar a cabo modificaciones para que el programa se adapte o actualice a nuevas necesidades, pudiendo también corregirse errores no detectados anteriormente. Este proceso recibe el nombre de mantenimiento del software.

> **Mantenimiento del software:** es el proceso de mejora y optimización del software después de su entrega al usuario final. Involucra cambios al software en orden de corregir defectos y dependencias encontradas durante su uso, así como la adición de nuevas funcionalidades para mejorar la usabilidad y aplicabilidad del software.

Será imprescindible añadir una documentación adecuada que facilite al programador la comprensión, uso y modificación de dichos programas.

5.- Ciclo de vida del software.

Sean cuales sean las fases en las que realicemos el proceso de desarrollo de software, y casi independientemente de él, siempre se debe aplicar un modelo de ciclo de vida.

> **Ciclo de vida del software:** es una sucesión de estados o fases por las cuales pasa un software a lo largo de su "vida".

El proceso de desarrollo puede involucrar siempre las siguientes etapas mínimas:

- ✓ Especificación y Análisis de requisitos.
- ✓ Diseño.
- ✓ Codificación.
- ✓ Pruebas.
- ✓ Instalación y paso a Producción.
- ✓ Mantenimiento.

Existen varios tipos de ciclos de vida del software, los más importantes son los siguientes:

- **Modelo en cascada:** Es también conocido por “modelo clásico”, modelo tradicional” o “modelo lineal secuencial”. Para comenzar una fase ha de finalizarse la anterior. Es rígido y en la práctica presenta algunos problemas de aplicación. Es el más utilizado por su escasa complejidad.

- **Modelo por prototipos**: Este ciclo de vida se basa en la creación de prototipos que irán mejorando el conocimiento del problema, tanto para el usuario como para los desarrolladores
  La fase de especificación de requerimientos está compuesta por las siguientes subfases:

  - ✓ Pequeño análisis y especificación.
  - ✓ Diseño y realización.
  - ✓ Evaluación.
  - ✓ Modificación.
  - ✓ Finalización de los requerimientos.

- **Modelo evolutivo:** Se emplea para facilitar la creación de aplicaciones flexibles y escalables, que permitan incorporar modificaciones muy rápidamente una vez que se finalice su desarrollo.
  Este modelo permite adaptarse a requisitos que varíen en el tiempo. Es un modelo iterativo, permite desarrollar versiones cada vez más completas y complejas, hasta llegar al objetivo final deseado; incluso evolucionar más allá, durante la fase de explotación.
  Los modelos "iterativo incremental" y "espiral" son del tipo evolutivo.

  - ✓ **Modelo incremental:** El funcionamiento del modelo iterativo incremental permite la entrega de versiones parciales a medida que se va construyendo el producto final.
    Por ejemplo, un procesador de texto podría incluir inicialmente funciones básicas. En un incremento posterior podría incorporar funciones para previsualización y paginación. En los siguientes incrementos podría disponer de funciones de corrección ortográfica, etc. Tras sucesivos incrementos lograríamos obtener el procesador de texto final.

  - ✓ **Modelo espiral**: Este modelo emplea lo mejor de los modelos convencional y por prototipos. Está dividida en cuatro fases:

    - ✓ Planificación.
    - ✓ Análisis de riesgo.
    - ✓ Desarrollo
    - ✓ Evaluación del cliente

    El paso por cada una de estas fases se repetirá tantas veces como sea necesario hasta que se cumplan todos los requerimientos del usuario.

6.- Lenguajes de programación.

En todo el proceso de resolución de un problema mediante la creación de software, después del análisis del problema y del diseño del algoritmo que pueda resolverlo, es necesario traducir éste a un lenguaje que exprese claramente cada uno de los pasos a seguir para su correcta ejecución. Este lenguaje recibe el nombre de lenguaje de programación.

Hay que tener en cuenta que pueden existir sentencias sintácticamente correctas, pero semánticamente incorrectas. Por ejemplo, ***"Un avestruz dio un zarpazo a su cuidador"*** está bien construida sintácticamente, pero es evidente que semánticamente no.

Una característica relevante de los lenguajes de programación es, precisamente, que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos. A través de este conjunto se puede lograr la construcción de un programa de forma colaborativa.

Lenguaje de programación: Conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas. Es un lenguaje artificial, una construcción mental del ser humano para expresar programas.

Gramática del lenguaje: Reglas aplicables al conjunto de símbolos y palabras especiales del lenguaje de programación para la construcción de sentencias correctas.

Léxico: Es el conjunto finito de símbolos y palabras especiales, es el vocabulario del lenguaje.

Sintaxis: Son las posibles combinaciones de los símbolos y palabras especiales. Está relacionada con la forma de los programas.

Semántica: Es el significado de cada construcción del lenguaje, la acción que se llevará a cabo.

Los lenguajes de programación pueden ser clasificados en función de lo cerca que estén del lenguaje humano o del lenguaje de los computadores. El lenguaje de los computadores son códigos binarios, es decir, secuencias de unos y ceros. Detallaremos seguidamente las características principales de los lenguajes de programación.

6.1.- Lenguaje máquina.

Este es el lenguaje utilizado directamente por el procesador, consta de un conjunto de instrucciones codificadas en binario. Es el sistema de códigos directamente interpretable por un **circuito microprogramable.**

Este fue el primer lenguaje utilizado para la programación de computadores. De hecho, cada máquina tenía su propio conjunto de instrucciones codificadas en ceros y unos. Cuando un algoritmo está escrito en este tipo de lenguaje, decimos que está en código máquina.

Programar en este tipo de lenguaje presentaba los siguientes inconvenientes:

- ✓ Cada programa era válido sólo para un tipo de procesador u ordenador.
- ✓ La lectura o interpretación de los programas era extremadamente difícil y, por tanto, insertar modificaciones resultaba muy costoso.
- ✓ Los programadores de la época debían memorizar largas combinaciones de ceros y unos, que equivalían a las instrucciones disponibles para los diferentes tipos de procesadores.
- ✓ Los programadores se encargaban de introducir los códigos binarios en el computador, lo que provocaba largos tiempos de preparación y posibles errores.

A continuación, se muestran algunos códigos binarios equivalentes a las operaciones de suma, resta y movimiento de datos en lenguaje máquina.

Algunas operaciones en lenguaje máquina.	
Operación	**Lenguaje máquina**
SUMAR	00101101
RESTAR	00010011
MOVER	00111010

Dada la complejidad y dificultades que ofrecía este lenguaje, fue sustituido por otros más sencillos y fáciles de utilizar. No obstante, hay que tener en cuenta que todos los programas para poder ser ejecutados, han de traducirse siempre al lenguaje máquina que es el único que entiende la computadora.

PARA SABER MÁS

Información sobre cómo funciona el sistema binario.

Sistema binario

6.2.- Lenguaje Ensamblador.

La evolución del lenguaje máquina fue el lenguaje ensamblador. Las instrucciones ya no son secuencias binarias, se sustituyen por códigos de operación que describen una operación elemental del procesador. Es un lenguaje de bajo nivel, al igual que el lenguaje máquina, ya que dependen directamente del hardware donde son ejecutados.

Mnemotécnico: son palabras especiales, que sustituyen largas secuencias de ceros y unos, utilizadas para referirse a diferentes operaciones disponibles en el juego de instrucciones que soporta cada máquina en particular.

En ensamblador, cada instrucción (mnemotécnico) se corresponde a una instrucción del procesador. En la siguiente tabla se muestran algunos ejemplos.

Algunas operaciones y su mnemotécnico en lenguaje Ensamblador.	
Operación	Lenguaje Ensamblador
MULTIPLICAR	MUL
DIVIDIR	DIV
MOVER	MOV

En el siguiente gráfico puedes ver parte de un programa escrito en lenguaje ensamblador. En color rojo se ha resaltado el código máquina en **hexadecimal**, en magenta el código escrito en ensamblador y en azul, las direcciones de memoria donde se encuentra el código.

```
-u 100 1a
0CFD:0100 BA0B01      MOV     DX,010B
0CFD:0103 B409        MOV     AH,09
0CFD:0105 CD21        INT     21
0CFD:0107 B400        MOV     AH,00
0CFD:0109 CD21        INT     21
-d 10b 13f
0CFD:0100                                               48 6F 6C 61 2C
0CFD:0110  20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67
0CFD:0120  72 61 6D 61 20 68 65 63-68 6F 20 65 6E 20 61 73
0CFD:0130  73 65 6D 62 6C 65 72 20-70 61 72 61 20 6C 61 20
0CFD:0140  57 69 6B 69 70 65 64 69-61 24
```

Pero aunque ensamblador fue un intento por aproximar el lenguaje de los procesadores al lenguaje humano, presentaba múltiples dificultades:

- ✓ Los programas seguían dependiendo directamente del hardware que los soportaba.
- ✓ Los programadores tenían que conocer detalladamente la máquina sobre la que programaban, ya que debían hacer un uso adecuado de los recursos de dichos sistemas.
- ✓ La lectura, interpretación o modificación de los programas seguía presentando dificultades.

Todo programa escrito en lenguaje ensamblador necesita de un intermediario, que realice la traducción de cada una de las instrucciones que componen su código al lenguaje máquina correspondiente. Este intermediario es el programa ensamblador. El programa original escrito en lenguaje ensamblador constituye el código fuente y el programa traducido al lenguaje máquina se conoce como programa objeto que será directamente ejecutado por la computadora.

6.3.- Lenguajes compilados.

Para paliar los problemas derivados del uso del lenguaje ensamblador y con el objetivo de acercar la programación hacia el uso de un lenguaje más cercano al humano que al del computador, nacieron los lenguajes compilados. Algunos ejemplos de este tipo de lenguajes son: Pascal, Fortran, Algol, C, **C++**, etc.

Al ser lenguajes más cercanos al humano, también se les denomina **lenguajes de alto nivel**. Son más fáciles de utilizar y comprender, las instrucciones que forman parte de estos lenguajes utilizan palabras y signos reconocibles por el programador.

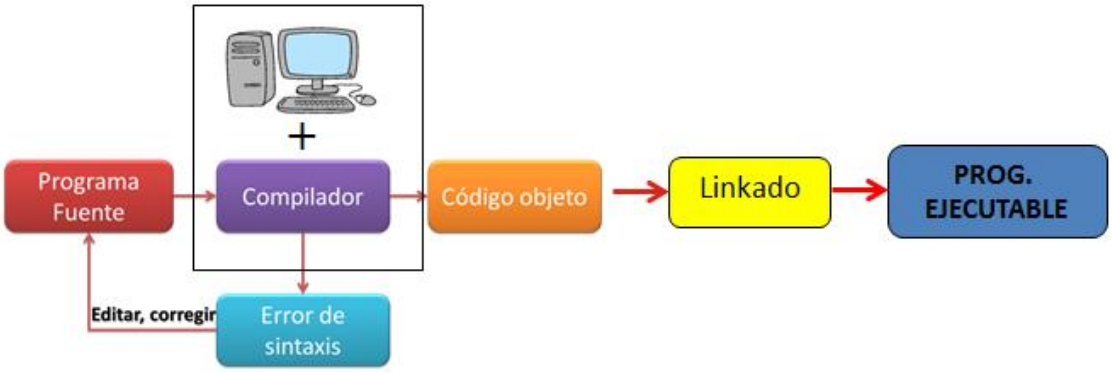
¿Cuáles son sus **ventajas**?

- Son mucho más fáciles de aprender y de utilizar que sus predecesores.
- Se reduce el tiempo para desarrollar programas, así como los costes.
- Son independientes del hardware, los programas pueden ejecutarse en diferentes tipos de máquina.
- La lectura, interpretación y modificación de los programas es mucho más sencilla.

Pero un programa que está escrito en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores.

Compilador: Es un programa cuya función consiste en traducir el código fuente de un programa escrito en un lenguaje de alto nivel a lenguaje máquina. Al proceso de traducción se le conoce con el nombre de compilación.

El compilador realizará la traducción y además informará de los posibles errores. Una vez subsanados, se generará el programa traducido a código máquina, conocido como **código objeto**. Este programa aún no podrá ser ejecutado hasta que no se le añadan los módulos de enlace o bibliotecas, durante el proceso de enlazado. Una vez finalizado el enlazado, se obtiene el **código ejecutable**.



PARA SABER MÁS

Para ilustrar el proceso de compilación de programas te proponemos el siguiente enlace:

Proceso de compilación en varias plataformas

6.4.- Lenguajes interpretados.

Se caracterizan por estar diseñados para que su ejecución se realice a través de un **intérprete**. Cada instrucción escrita en un lenguaje interpretado se analiza, traduce y ejecuta tras haber sido verificada. Una vez realizado el proceso por el intérprete, la instrucción se ejecuta, pero no se guarda en memoria.

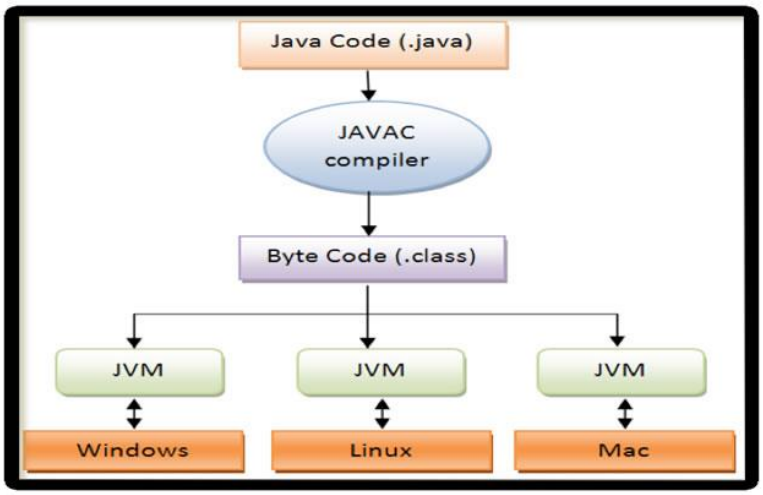
> **Intérprete:** Es un programa traductor de un lenguaje de alto nivel en el que el proceso de traducción y de ejecución se llevan a cabo simultáneamente, es decir, la instrucción se pasa a lenguaje máquina y se ejecuta directamente. No se genera programa objeto, ni programa ejecutable.

Los lenguajes interpretados generan programas de menor tamaño que los generados por un compilador, al no guardar el programa traducido a código máquina. Pero presentan el inconveniente de ser algo más lentos, ya que han de ser traducidos durante su ejecución. Por otra parte, necesitan disponer en la máquina del programa intérprete ejecutándose, algo que no es necesario en el caso de un programa compilado, para los que sólo es necesario tener el programa ejecutable para poder utilizarlo.

Ejemplos de lenguajes interpretados son: **Perl, PHP, Python, JavaScript,** etc.

A medio camino entre los lenguajes compilados y los interpretados, existen los lenguajes que podemos denominar **pseudo-compilados o pseudo-interpretados**, es el caso del **Lenguaje Java**.

Java puede verse como compilado e interpretado a la vez, ya que su código fuente se compila para obtener el código binario en forma de bytecodes, que son estructuras parecidas a las instrucciones máquina, con la importante propiedad de no ser dependientes de ningún tipo de máquina (se detallarán más adelante). La Máquina Virtual Java se encargará de interpretar este código y, para su ejecución, lo traducirá a código máquina del procesador en particular sobre el que se esté trabajando.



PARA SABER MÁS

Puedes entender por qué Java es un lenguaje compilado e interpretado a través del siguiente esquema.

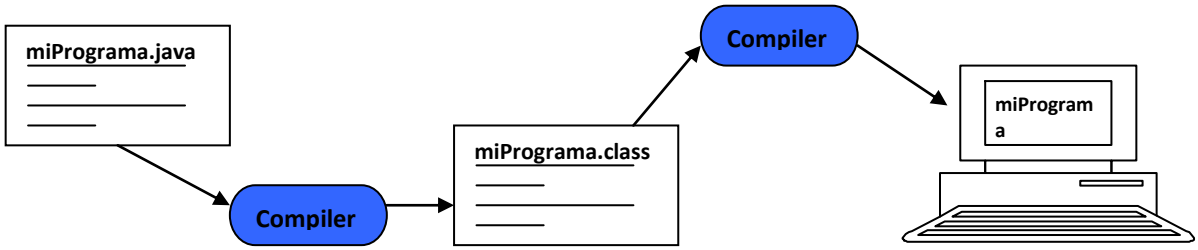
El lenguaje Java es compilado e interpretado.

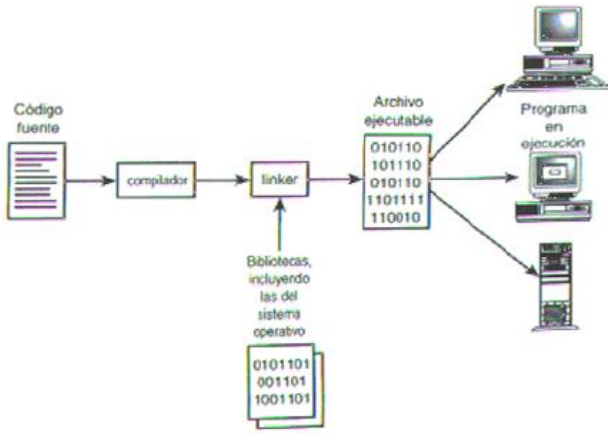
7.- El lenguaje de programación Java.

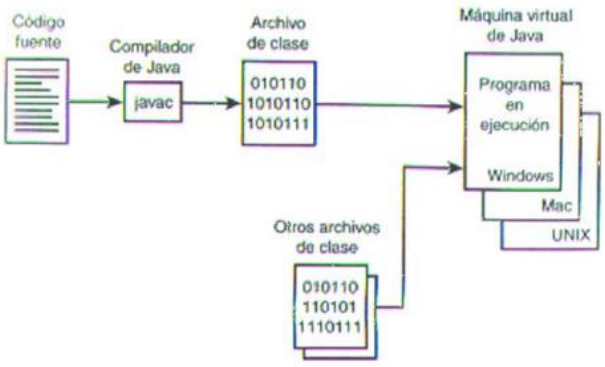
7.1.- ¿Qué y cómo es Java?

Java es un lenguaje sencillo de aprender, con una sintaxis parecida a la de C++, pero en la que se han eliminado elementos complicados y que pueden originar errores. Java es orientado a objetos, con lo que elimina muchas preocupaciones al programador y permite la utilización de gran cantidad de bibliotecas ya definidas, evitando reescribir código que ya existe.
Es un lenguaje de programación creado para satisfacer nuevas necesidades que los lenguajes existentes hasta el momento no eran capaces de solventar.

Una de las principales virtudes de Java es su independencia del hardware, ya que el código que se genera es válido para cualquier plataforma. Este código será ejecutado sobre una máquina virtual denominada **Máquina Virtual Java** (MVJ o JVM – Java Virtual Machine), que interpretará el código convirtiéndolo a código específico de la plataforma que lo soporta. De este modo el programa se escribe una única vez y puede hacerse funcionar en cualquier lugar. Lema del lenguaje: ***"Write once, run everywhere".***







Antes de que apareciera Java, el lenguaje C era uno de los más extendidos por su versatilidad. Pero cuando los programas escritos en C aumentaban de volumen, su manejo comenzaba a complicarse. Mediante las técnicas de programación estructurada y programación modular se conseguían reducir estas complicaciones, pero no era suficiente.

Fue entonces cuando la Programación Orientada a Objetos (POO) entra en escena, aproximando notablemente la construcción de programas al pensamiento humano y haciendo más sencillo todo el proceso. Los problemas se dividen en objetos que tienen propiedades e interactúan con otros objetos, de este modo, el programador puede centrarse en cada objeto para programar internamente los elementos y funciones que lo componen.
Las características principales de lenguaje Java se resumen a continuación:

- El código generado por el compilador Java es independiente de la arquitectura.
- Está totalmente orientado a objetos.
- Su sintaxis es similar a C y C++.
- Es distribuido, preparado para aplicaciones **TCP/IP**.
- Dispone de un amplio conjunto de bibliotecas.
- Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
- La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema.

PARA SABER MÁS

En este enlace hay una descripción detallada de las características reseñadas anteriormente a través del siguiente artículo:

Características detalladas del lenguaje Java

7.2.- Breve historia.

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que rescribir el código. Por eso la empresa Sun quería crear un lenguaje **independiente del dispositivo.**

Pero no fue hasta 1995 cuando pasó a llamarse **Java**, dándose a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esa filosofía y su facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

El factor determinante para su expansión fue la incorporación de un intérprete Java en la versión 2.0 del navegador Web Netscape Navigator, lo que supuso un gran revuelo en Internet.

A principios de 1997 apareció **Java 1.1**que proporcionó sustanciales mejoras al lenguaje. **Java 1.2**, más tarde rebautizado como **Java 2,** nació a finales de 1998.

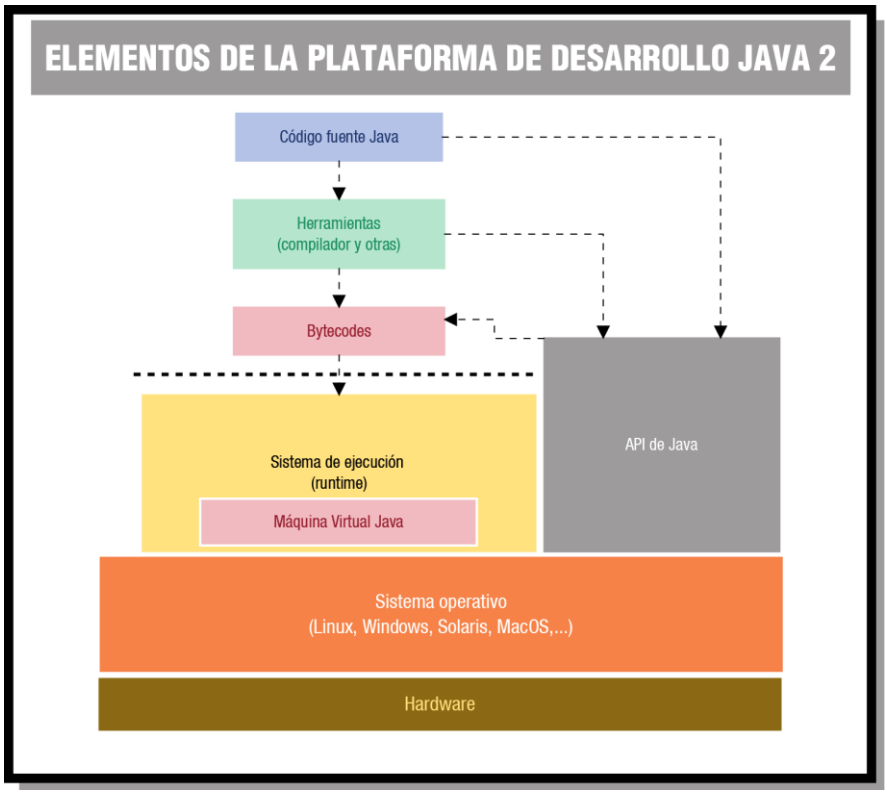
El principal objetivo del lenguaje Java es llegar a ser el **nexo universal** que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor Web, en una base de datos o en cualquier otro lugar.

Para el desarrollo de programas en lenguaje Java es necesario utilizar un entorno de desarrollo denominado **JDK** (Java Development Kit), que provee de un compilador y un entorno de ejecución (**JRE** – Java Run Environment) para los bytecodes generados a partir del código fuente. Al igual que las diferentes versiones del lenguaje han incorporado mejoras, el entorno de desarrollo y ejecución también ha sido mejorado sucesivamente.

Java 2 es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluye los siguientes elementos:

- Un lenguaje de programación: Java.
- Un conjunto de bibliotecas estándar que vienen incluidas en la plataforma y que son necesarias en todo entorno Java. Es el Java Core.
- Un conjunto de herramientas para el desarrollo de programas, como es el compilador de bytecodes, el generador de documentación, un depurador, etc.
- Un entorno de ejecución que en definitiva es una máquina virtual que ejecuta los programas traducidos a bytecodes.

El siguiente esquema muestra los elementos fundamentales de la plataforma de desarrollo Java 2.



Actualmente hay tres ediciones de la plataforma Java 2:

- **J2SE**: Entorno de Sun relacionado con la creación de aplicaciones y **applets** en lenguaje Java.
- **J2EE**: Pensada para la creación de aplicaciones Java empresariales y del lado del servidor.
- **J2ME**: Pensada para la creación de aplicaciones Java para dispositivos móviles.

PARA SABER MÁS

Si deseas conocer más sobre los orígenes del lenguaje Java, aquí te ofrecemos más información:

Los orígenes de Java

Historia de Java

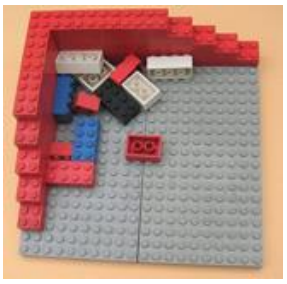
Línea de tiempo de la historia de Java

7.3.- La POO y Java.

En Java, los datos y el código (funciones o métodos) se combinan en entidades llamadas **objetos**. El objeto tendrá un comportamiento (su código interno) y un estado (los datos). Los objetos permiten la reutilización del código y pueden considerarse, en sí mismos, como piezas reutilizables en múltiples proyectos distintos. Esta característica permite reducir el tiempo de desarrollo de software.

Por simplificar un poco las cosas, un programa en Java será como una representación teatral en la que debemos preparar primero cada personaje, definir sus características y qué va a saber hacer. Cuando esta fase esté terminada, la obra se desarrollará sacando personajes a escena y haciéndoles interactuar.

Al emplear los conceptos de la Programación Orientada a Objetos (POO), Java incorpora las tres características propias de este paradigma: **encapsulación**, **herencia** y **polimorfismo**. Los patrones o tipos de objetos se denominan **clases** y los objetos que utilizan estos patrones o pertenecen a dichos tipos, se identifican con el nombre de **instancias**. Pero, no hay que alarmarse, estos conceptos se verán más adelante en sucesivas unidades.



Otro ejemplo para seguir aclarando ideas, piensa en los bloques de juegos de construcción. Suponemos que conoces los cubos de plástico en varios colores y tamaños. Por una de sus caras disponen de pequeños conectores circulares y en otra de sus caras pequeños orificios en los que pueden conectarse otros bloques, con el objetivo principal de permitir construir formas más grandes. Si usas diferentes piezas del lego puedes construir aviones, coches, edificios, etc. Si te fijas bien, cada pieza es un objeto pequeño que puede unirse con otros objetos para crear objetos más grandes.

Pues bien, aproximadamente así es **como funciona la programación dirigida a objetos**: unimos elementos pequeños para construir otros más grandes. Nuestros programas estarán formados por muchos componentes (objetos) independientes y diferentes; cada uno con una función determinada en nuestro software y que podrá comunicarse con los demás de una manera predefinida.

7.4.- Independencia de la plataforma y trabajo en red.

Existen dos características que distinguen a **Java** de otros lenguajes, como son la **independencia de la plataforma** y la posibilidad de trabajar en red o, mejor, la posibilidad de **crear aplicaciones que trabajan en red**.

Estas características las vamos a explicar a continuación:

- **Independencia:** Los programas escritos en Java pueden ser ejecutados en cualquier tipo de hardware. El código fuente es compilado, generándose el código conocido como **Java Bytecode** (instrucciones máquina simplificadas que son específicas de la plataforma Java), el bytecode será interpretado y ejecutado en la **Máquina Virtual Java (MVJ o JVM – Java Virtual Machine)** que es un programa escrito en código nativo de la plataforma destino entendible por el hardware. Con esto se evita tener que realizar un programa diferente para cada CPU o plataforma.
  Por tanto, la parte que realmente es dependiente del sistema es la Máquina Virtual Java, así como las librerías o bibliotecas básicas que permiten acceder directamente al hardware de la máquina.

- **Trabajo en red:** Esta capacidad del lenguaje ofrece múltiples posibilidades para la comunicación vía TCP/IP. Para poder hacerlo existen librerías que permiten el acceso y la interacción con protocolos como **http**, **ftp**, etc., facilitando al programador las tareas del tratamiento de la información a través de redes.

7.5.- Seguridad y simplicidad.

Junto a las características diferenciadoras del lenguaje Java relacionadas con la independencia y el trabajo en red, han de destacarse dos virtudes que hacen a este lenguaje uno de los más extendidos entre la comunidad de programadores: su seguridad y su simplicidad.

- **Seguridad:** En primer lugar, los posibles accesos a zonas de memoria "sensibles" que en otros lenguajes como C y C++ podían suponer peligros importantes, se han eliminado en Java.

  En segundo lugar, el código Java es comprobado y verificado para evitar que determinadas secciones del código produzcan efectos no deseados. Los test que se aplican garantizan que las operaciones, operandos, conversiones, uso de clases y demás acciones son seguras.

  Y en tercer lugar, Java no permite la apertura de ficheros en la máquina local, tampoco permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra.

  En definitiva, podemos afirmar que Java es un lenguaje seguro.

- **Simplicidad:** Aunque Java es tan potente como C o C++, es bastante más sencillo. Posee una curva de aprendizaje muy rápida y, para alguien que comienza a programar en este lenguaje, le resulta relativamente fácil comenzar a escribir aplicaciones interesantes.

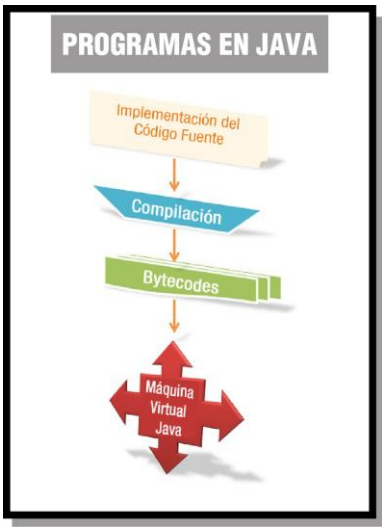
  Si has programado alguna vez en C o C++ encontrarás que Java te pone las cosas más fáciles, ya que se han eliminado: la aritmética de **punteros**, los registros, la definición de tipos, la gestión de memoria, etc. Con esta simplificación se reduce bastante la posibilidad de cometer errores comunes en los programas. Un programador experimentado en C o C++ puede cambiar a este lenguaje rápidamente y obtener resultados en muy poco espacio de tiempo.

  Muy relacionado con la simplicidad que aporta Java está la incorporación de un elemento muy útil como es el **Recolector de Basura (Garbage collector)**. Permite al programador liberarse de la gestión de la memoria y hace que ciertos bloques de memoria puedan reaprovecharse, disminuyendo el número de huecos libres (**fragmentación de memoria**).

  Cuando realicemos programas, crearemos objetos, haremos que éstos interaccionen, etc. Todas estas operaciones requieren de uso de memoria del sistema, pero la gestión de ésta será realizada de manera transparente al programador. Todo lo contrario que ocurría en otros lenguajes. Podremos crear tantos objetos como solicitemos, pero nunca tendremos que destruirlos. El entorno de Java borrará los objetos cuando determine que no se van a utilizar más. Este proceso es conocido como recolección de basura.

7.6.- Java y los Bytecodes.

Un programa escrito en Java no es directamente ejecutable, es necesario que el código fuente sea interpretado por la Máquina Virtual Java. ¿Cuáles son los pasos que se siguen desde que se genera el código fuente hasta que se ejecuta?



A continuación se detallan cada uno de ellos.

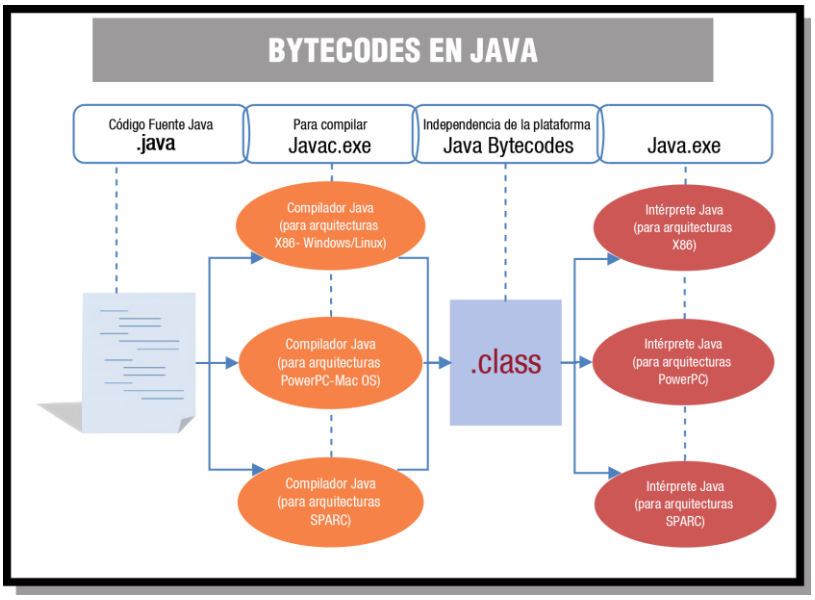
- Se escribe el código fuente (archivos con extensión **.Java**).
- Este es precompilado generándose los códigos de bytes, Bytecodes o Java Bytecodes (archivos con extensión **.class**)
- Estos serán interpretados directamente por la Máquina Virtual Java y traducidos a código nativo de la plataforma sobre la que se esté ejecutando el programa.

Bytecode: Son un conjunto de instrucciones en lenguaje máquina que no son específicas a ningún procesador o sistema de cómputo. Un intérprete de código de bytes (bytecodes) para una plataforma específica será quien los ejecute. A estos intérpretes también se les conoce como Máquinas Virtuales Java o intérpretes Java de tiempo de ejecución.

En el proceso de precompilación, existe un verificador de códigos de bytes que se asegurará de que se cumplen las siguientes condiciones:

- El código satisface las especificaciones de la Máquina Virtual Java.
- No existe amenaza contra la integridad del sistema.
- No se producen desbordamientos de memoria.
- Los parámetros y sus tipos son adecuados.
- No existen conversiones de datos no permitidas.

Para que un bytecode pueda ser ejecutado en cualquier plataforma, es imprescindible que dicha plataforma cuente con el intérprete adecuado, es decir, la máquina virtual específica para esa plataforma. En general, la Máquina Virtual Java es un programa de reducido tamaño y gratuito para todos los sistemas operativos.



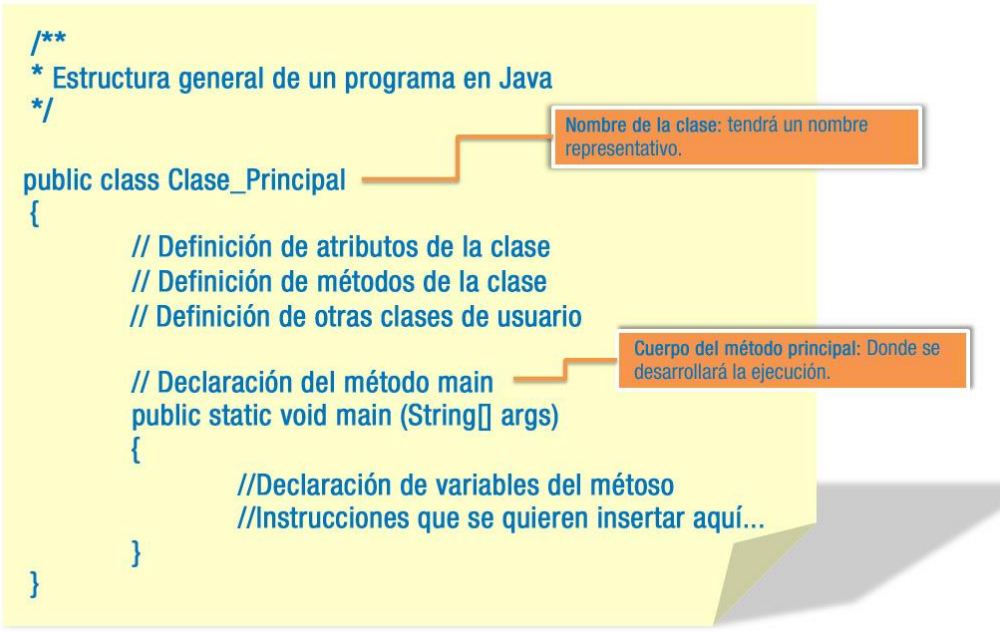
8.- Programas en Java.

Hasta ahora, hemos descrito el lenguaje de programación Java, hemos hecho un recorrido por su historia y nos hemos instruido sobre su filosofía de trabajo, pero te preguntarás ¿Cuándo empezamos a desarrollar programas? ¿Qué elementos forman parte de un programa en Java? ¿Qué se necesita para programar en este lenguaje? ¿Podemos crear programas de diferente tipo?

No te impacientes, cada vez estamos más cerca de comenzar la experiencia con el lenguaje de programación Java. Iniciaremos nuestro camino conociendo cuales son los elementos básicos de un programa Java, la forma en que debemos escribir el código y los tipos de aplicaciones que pueden crearse en este lenguaje.

8.1.- Estructura de un programa.

En siguiente figura se presenta la estructura general de un programa realizado en un lenguaje orientado a objetos como es Java.



Recomendación.

Ten en cuenta que **Java distingue entre mayúsculas y minúsculas**. Si le das a la clase principal el nombre PrimerPrograma, el archivo **.Java** tendrá como identificador **PrimerPrograma.Java**, que es totalmente diferente a primerprograma.Java. Además, para Java los elementos PrimerPrograma y primerprograma serían considerados dos clases diferentes dentro del código fuente.

Vamos a analizar cada uno de los elementos que aparecen en dicho gráfico:

public class Clase_Principal: Todos los programas han de incluir una clase como esta. Es una clase general en la que se incluyen todos los demás elementos del programa. Entre otras cosas, contiene el método o función **main()** que representa al programa principal, desde el que se llevará a cabo la ejecución del programa. Esta clase puede contener a su vez otras clases del usuario, pero sólo una puede ser **public**. El nombre del fichero **.Java** que contiene el código fuente de nuestro programa, coincidirá con el nombre de la clase que estamos describiendo en estas líneas.

- ✓ **public static void main (String[] args)**: Es el método que representa al programa principal, en él se podrán incluir las instrucciones que estimemos oportunas para la ejecución del programa. Desde él se podrá hacer uso del resto de clases creadas. Todos los programas Java tienen un método **main**.

- ✓ **Comentarios**: Los comentarios se suelen incluir en el código fuente para realizar aclaraciones, anotaciones o cualquier otra indicación que el programador estime oportuna. Estos comentarios pueden introducirse de dos formas, **con //** y **con /* */**. Con la primera forma estaríamos estableciendo una única línea completa de comentario y, con la segunda, con **/*** comenzaríamos el comentario y éste no terminaría hasta que no insertáramos ***/**.

- ✓ **Bloques de código**: son conjuntos de instrucciones que se marcan mediante la apertura y cierre de llaves **{ }**. El código así marcado es considerado interno al bloque.

- ✓ **Punto y coma**: aunque en el ejemplo no hemos incluido ninguna línea de código que termine con punto y coma, hay que hacer hincapié en que cada línea de código ha de terminar con punto y coma (**;**). En caso de no hacerlo, tendremos errores sintácticos.

8.2.- El entorno básico de desarrollo Java.

Ya conoces cómo es la estructura de un programa en Java, pero, ¿qué necesitamos para llevarlo a la práctica? La herramienta básica para empezar a desarrollar aplicaciones en Java es el **JDK (Java Development Kit o Kit de Desarrollo Java)**, que incluye un compilador y un intérprete para línea de comandos. Estos dos programas son los empleados en la precompilación e interpretación del código.

Como veremos, existen diferentes entornos para la creación de programas en Java que incluyen multitud de herramientas, pero por ahora nos centraremos en el entorno más básico, extendido y gratuito, el Java Development Kit (**JDK**). Según se indica en la propia página web de Oracle, JDK es un entorno de desarrollo para construir aplicaciones, applets y componentes utilizando el lenguaje de programación Java. Incluye herramientas útiles para el desarrollo y prueba de programas escritos en Java y ejecutados en la Plataforma Java.

Así mismo, junto a JDK se incluye una implementación del entorno de ejecución Java, el **JRE (Java Runtime Environment)** para ser utilizado por el JDK. El JRE incluye la Máquina Virtual de Java (MVJ ó JVM – Java Virtual Machine), bibliotecas de clases y otros ficheros que soportan la ejecución de programas escritos en el lenguaje de programación Java.

Para poder utilizar JDK y JRE es necesario realizar la descarga e instalación de éstos. Puedes seguir los pasos del proceso a continuación:

Ver Anexo II

Para poder desarrollar nuestros primeros programas en Java sólo necesitaremos un editor de texto plano y los elementos que acabamos de instalar a través de Java SE.

8.3.- La API de Java.

Junto con el kit de desarrollo que hemos descargado e instalado anteriormente, vienen incluidas gratuitamente todas las bibliotecas de la API (Aplication Programming Interface – Interfaz de programación de aplicaciones) de Java, es lo que se conoce como Bibliotecas de Clases Java. Este conjunto de bibliotecas proporciona al programador paquetes de clases útiles para la realización de múltiples tareas dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas **semánticamente**.

En décadas pasadas una biblioteca era un conjunto de programas que contenían cientos de rutinas (una rutina es un procedimiento o función bien verificados, en determinado lenguaje de programación). Las rutinas de biblioteca manejaban las tareas que todos o casi todos los programas necesitaban. El programador podía recurrir a esta biblioteca para desarrollar programas con rapidez.

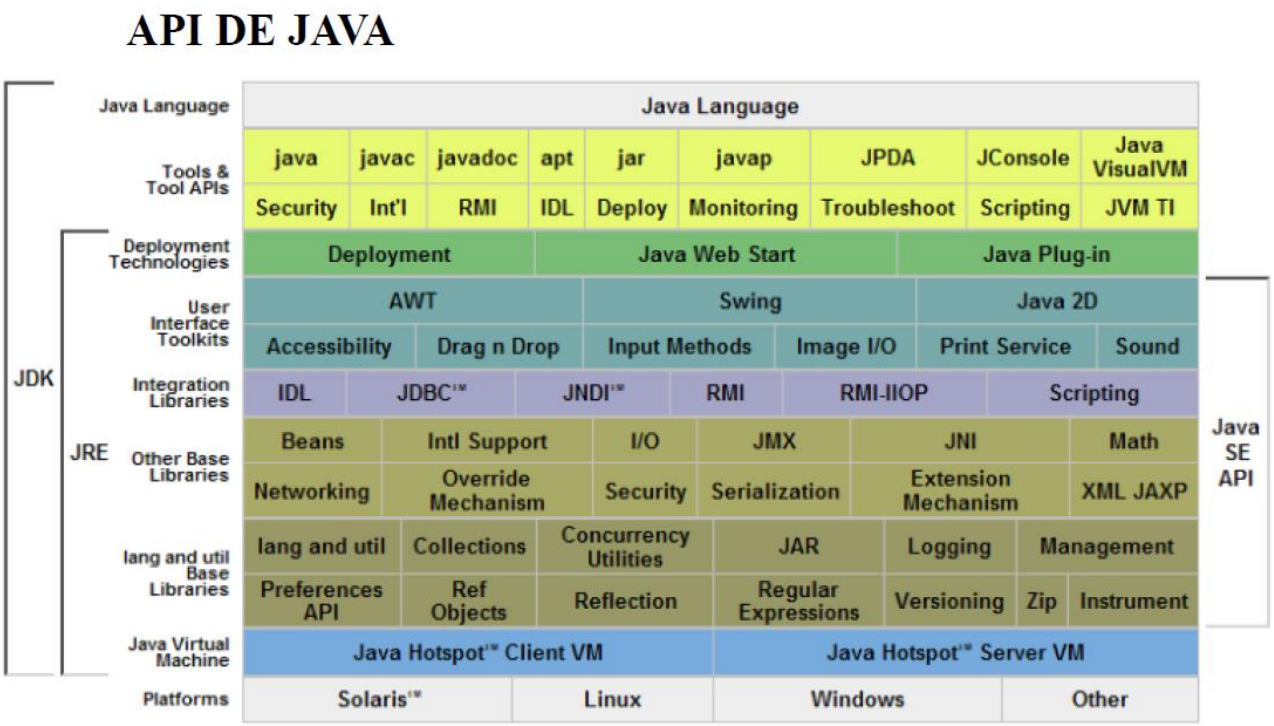
Una biblioteca de clases es un conjunto de clases de programación orientada a objetos. Esas clases contienen métodos que son útiles para los programadores. En el caso de Java cuando descargamos el JDK obtenemos la biblioteca de clases API. Utilizar las clases y métodos de las APIs de Java reduce el tiempo de desarrollo de los programas. También, existen diversas bibliotecas de clases desarrolladas por terceros que contienen componentes reutilizables de software, y están disponibles a través de la Web.

PARA SABER MÁS

Si quieres acceder a la información oficial sobre la API de Java, te proponemos el siguiente enlace

Información oficial sobre la API de Java

Documentación de la API de Java



8.4.- 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 ello, sigue las indicaciones que te mostramos a continuación:
En el siguiente anexo aprenderás como configurar la variable PATH en Windows.

Ver Anexo III

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 **.class** 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\jdk1.6.0_25\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\jdk1.6.0_25\lib**)

PARA SABER MÁS

Si deseas conocer más sobre la configuración de variables de entorno en sistemas Windows y Linux, te proponemos los siguientes enlaces:

Configurar el PATH en Windows

Configurar variables de entorno

8.5.- Codificación, compilación y ejecución de aplicaciones.

Una vez que la configuración del entorno Java está completada y tenemos el código fuente de nuestro programa escrito en un archivo con extensión **.Java**, la compilación de aplicaciones se realiza mediante el programa **Javac** incluido en el software de desarrollo de Java.

Para llevar a cabo la compilación desde la línea de comandos, escribiremos:

Javac archivo.Java

Donde **Javac** es el compilador de Java y **archivo.Java** es nuestro código fuente.

El resultado de la compilación será un archivo con el mismo nombre que el archivo Java pero con la **extensión class**. Esto ya es el archivo con el código en forma de bytecode. Es decir con el código precompilado. Si en el código fuente de nuestro programa figuraran más de una clase, veremos como al realizar la compilación se generarán tantos archivos con extensión **.class** como clases tengamos. Además, si estas clases tenían método **main** podremos ejecutar dichos archivos por separado para ver el funcionamiento de dichas clases.

Para que el programa pueda ser ejecutado, siempre y cuando esté incluido en su interior el método main, podremos utilizar el intérprete incluido en el kit de desarrollo.

> La ejecución de nuestro programa desde la línea de comandos podremos hacerla escribiendo:
>
> **Java archivo.class**
>
> Donde **Java** es el intérprete y **archivo.class** es el archivo con el código precompilado.

8.6.- Tipos de aplicaciones en Java.

La versatilidad del lenguaje de programación Java permite al programador crear distintos tipos de aplicaciones. A continuación, describiremos las características más relevantes de cada uno de ellos:

- **Aplicaciones de consola:**
  - ✓ Son programas independientes al igual que los creados con los lenguajes tradicionales.
  - ✓ Se componen como mínimo de un archivo class que debe contar necesariamente con el método main.
  - ✓ No necesitan un navegador web y se ejecutan cuando invocamos el comando Java para iniciar la Máquina Virtual de Java (JVM). De no encontrarse el método main la aplicación no podrá ejecutarse.
  - ✓ Las aplicaciones de consola leen y escriben hacia y desde la entrada y salida estándar, sin ninguna interfaz gráfica de usuario.

- **Aplicaciones gráficas:**
  - ✓ Aquellas que utilizan las clases con capacidades gráficas, como **Swing** que es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.
  - ✓ Incluyen las instrucciones **import**, que indican al compilador de Java que las clases del paquete **Javax.swing** se incluyan en la compilación.

- **Applets:**
  - ✓ Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador. Cuando el navegador carga una web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.
  - ✓ Se pueden descargar de Internet y se observan en un navegador. Los applets se descargan junto con una página HTML desde un servidor web y se ejecutan en la máquina cliente.
  - ✓ No tienen acceso a partes sensibles (por ejemplo: no pueden escribir archivos), a menos que uno mismo le dé los permisos necesarios en el sistema.
  - ✓ No tienen un método principal.
  - ✓ Son multiplataforma y pueden ejecutarse en cualquier navegador que soporte Java.

- **Servlets:**
  - ✓ Son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.
  - ✓ Los servlets, al contrario de los applets, son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones Web que interactúen con los clientes.

- **Midlets:**
  - ✓ Son 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), más específicamente para la máquina virtual Java MicroEdition (Java ME).
  - ✓ Generalmente son juegos y aplicaciones que se ejecutan en teléfonos móviles.

9.- Entornos Integrados de Desarrollo.

En los comienzos de Java la utilización de la línea de comandos era algo habitual. El programador escribía el código fuente empleando un editor de texto básico, seguidamente, pasaba a utilizar un compilador y con él obtenía el código compilado. En un paso posterior, necesitaba emplear una tercera herramienta para el ensamblado del programa. Por último, podía probar a través de la línea de comandos el archivo ejecutable. El problema surgía cuando se producía algún error, lo que provocaba tener que volver a iniciar el proceso completo.

Estas circunstancias hacían que el desarrollo de software no estuviera optimizado. Con el paso del tiempo, se fueron desarrollando aplicaciones que incluían las herramientas necesarias para realizar todo el proceso de programación de forma más sencilla, fiable y rápida. Para cada lenguaje de programación existen múltiples entornos de desarrollo, cada uno con sus ventajas e inconvenientes. Dependiendo de las necesidades de la persona que va a programar, la facilidad de uso o lo agradable que le resulte trabajar con él, se elegirá entre unos u otros entornos.

Para el lenguaje de programación Java existen múltiples alternativas, siendo los principales entornos de desarrollo **NetBeans** (que cuenta con el apoyo de la empresa Sun), **Eclipse y JCreator**. Los dos primeros son gratuitos, con soporte de idiomas y multiplataforma (Windows, Linux, MacOS).

¿Y cuál será con el que vamos a trabajar? El entorno que hemos seleccionado llevar a cabo nuestros desarrollos de software en este módulo profesional será **Eclipse** al haber sido construido por la misma compañía que creó Java, ser de código abierto y ofrecer capacidades profesionales.

9.1.- ¿Que son?.

Son aplicaciones que ofrecen la posibilidad de llevar a cabo el proceso completo de desarrollo de software a través de un único programa. Podremos realizar las labores de edición, compilación, depuración, detección de errores, corrección y ejecución de programas escritos en Java o en otros lenguajes de programación, bajo un entorno gráfico (no mediante línea de comandos). Junto a las capacidades descritas, cada entorno añade otras que ayudan a realizar el proceso de programación, como por ejemplo: código fuente coloreado, plantillas para diferentes tipos de aplicaciones, creación de proyectos, etc.

Hay que tener en cuenta que un entorno de desarrollo no es más que una fachada para el proceso de compilación y ejecución de un programa. ¿Qué quiere decir eso? Pues que si tenemos instalado un IDE y no tenemos instalado el compilador, no tenemos nada.

PARA SABER MÁS

Si deseas conocer algo más sobre lo que son los Entornos Integrados de Desarrollo (IDE) accede a la información del siguiente enlace:

Definición de Entorno Integrado de Desarrollo en Wikipedia

9.2.- IDE´S actuales.

Existen en el mercado multitud de entornos de desarrollo para el lenguaje Java, los hay de libre distribución, de pago, para principiantes, para profesionales, que consumen más recursos, que son más ligeros, más amigables, más complejos que otros, etc.

Entre los que son gratuitos o de libre distribución tenemos:

- ✓ **NetBeans**.
- ✓ **Eclipse.**
- ✓ **BlueJ.**
- ✓ **Jgrasp**.
- ✓ **Jcreator LE**.

Entre los que son propietarios o de pago tenemos:

- ✓ **IntelliJ IDEA.**
- ✓ **Jbuilder**.
- ✓ **Jcreator.**
- ✓ **JDeveloper**

PARA SABER MÁS

Cada uno de los entornos nombrados más arriba posee características que los hacen diferentes unos de otros, pero para tener una idea general de la versatilidad y potencia de cada uno de ellos, accede a la siguiente tabla comparativa:

Comparativa entornos para Java

PARA SABER MÁS

Si quieres conocer la situación actual de uso y comparar los diferentes entornos integrados de desarrollo para el lenguaje de programación Java, puedes ampliar datos en el siguiente artículo:

Artículo con comparativa sobre utilización de entornos Java.

Pero, ¿cuál o cuáles son los más utilizados por la comunidad de programadores Java? El puesto de honor se lo disputan entre **Eclipse, IntelliJ IDEA** y **NetBeans**. En los siguientes epígrafes haremos una descripción de **Eclipse**.

Ver Anexo IV

GLOSARIO

APPLET

Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador.

C++

Es el lenguaje de programación C ampliado para poder utilizar los mecanismos que permitan la manipulación de objetos. Es un lenguaje multiparadigma.

CIRCUITO MICROPROGRAMABLE

Dispositivo o conjunto de dispositivos de propósito general, que, según sea necesario, se programan para resolver distintos problemas.

CLASE

Es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y el comportamiento que todos los objetos de la clase comparten.

CONSTANTE

Estructura de datos que se utiliza en los lenguajes de programación que no puede cambiar su contenido en el transcurso del programa.

DISEÑO DESCENDENTE

Metodología de diseño de programas, consistente en la descomposición del problema en problemas más sencillos de resolver.

DISEÑO MODULAR

Metodología de diseño de programas, que consiste en dividir la solución a un problema en módulos más pequeños o subprogramas. Las soluciones de los módulos se unirán para obtener la solución general del problema.

ENCAPSULACIÓN.

En programación modular y más específicamente en programación orientada a objetos, se denomina así al ocultamiento de los datos y elementos internos de un objeto. Sólo se puede modificar un objeto a través de las operaciones definidas para éste.

FRAGMENTACIÓN DE MEMORIA

La fragmentación es la memoria que queda desperdiciada al usar los métodos de gestión de memoria. Puede ser interna o externa.

FTP

Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en arquitectura cliente-servidor.

HERENCIA

Mecanismo que permite derivar una clase de otra, de manera que extienda su funcionalidad.

HEXADECIMAL

Sistema numérico en base 16, esto significa que contiene 16 símbolos únicos para representar datos: los números del 0 al 9 y las letras de la A a la F.

HTTP

Es el protocolo utilizado para la transacción de la World Wide Web. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

INSTANCIA

Una instancia se produce con la creación de un objeto perteneciente a una clase (se dice que se instancia la clase). El objeto que se crea tiene los atributos, propiedades y métodos de la clase a la que pertenece. Los objetos y sus características se usan en la construcción de programas, ya sea como contenedores de datos o como partes funcionales del programa.

JAVASCRIPT

Lenguaje de programación interpretado. Se utiliza principalmente en su forma del lado del cliente (clientside), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web.

PHP

Lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Es usado principalmente para la interpretación del lado del servidor (serverside scripting), pero actualmente puede ser utilizado desde una interfaz de línea de comandos, o en la creación de otros tipos de programas, incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.

POLIMORFISMO

Capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente. Podemos crear dos clases distintas: Pez y Ave que heredan de la superclase Animal. La clase Animal tiene el método abstracto mover que se implementa de forma distinta en cada una de las subclases (peces y aves se mueven de forma distinta).

PUNTERO

Un puntero o apuntador es una variable que referencia una región de memoria; en otras palabras es una variable cuyo valor es una dirección de memoria.

PYTHON

Lenguaje de programación de alto nivel, cuya filosofía hace hincapié en una sintaxis limpia y que favorezca un código legible. Es multiparadigma, ya que soporta orientación a objetos, programación imperativa y programación funcional. Es un lenguaje interpretado y está fuertemente tipado. Es multiplataforma.

SCHEME

Lenguaje de programación funcional, dialecto de Lisp. Desarrollado en la década de los setenta. Scheme proporciona el mínimo número posible de nociones primitivas, construyendo todo lo demás a partir de un reducido número de abstracciones. Scheme ofrece también gestión automática de memoria. Las listas son la estructura de datos básica del lenguaje, que también ofrece arrays entre sus tipos predefinidos.

SEMÁNTICA

Referencia a los aspectos del significado, sentido o interpretación del significado de un determinado elemento, símbolo, palabra, expresión o representación formal. En principio cualquier medio de expresión (lenguaje formal o natural) admite una correspondencia entre expresiones de símbolos o palabras, y situaciones o conjuntos de cosas que se encuentran en el mundo físico o abstracto, que puede ser descrito por dicho medio de expresión.

SMALLTALK

Lenguaje de programación que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

SWING

Es una biblioteca gráfica para Java. Incluye elementos para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

TCP/IP

Familia de protocolos de Internet. Es un conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre redes de ordenadores.

VARIABLE

Estructura de datos que, como su nombre indica, puede cambiar de contenido a lo largo de la ejecución de un programa.