ELEMENTOS DE UN PROGRAMA INFORMÁTICO EN JAVA.

Contenidos:

1) Estructura y bloques fundamentales.
2) Variables.
3) Tipos de datos.
4) Literales.
5) Constantes.
6) Operadores y expresiones.
7) Conversiones de tipo.
8) Comentarios.
9) Estructuras: selección y control.

Comentarios.

- // Comentario de una línea

```
/* Inicio del comentario de varias líneas
Comentario1
Comentario2
Comentario3
Comentario4
Fin del comentario de varias líneas*/
```

- Los comentarios no tienen efecto como instrucciones para el ordenador, simplemente sirven para que cuando una persona lea el código pueda comprender mejor lo que lee

Estructura y bloques fundamentales.

- El código fuente de un programa en Java debe estar escrito en un fichero de texto con extensión **".java"**.

```
public class NombreClase
 {
      public static void main(String args[])
      { //Inicio del método
            System.out.println("Hola Mundo!");
      } //Fin del método
}
```

- El nombre de la clase debe corresponder exactamente con el nombre del fichero de texto que contiene el código fuente.

Estructura y bloques fundamentales.

- **public static void main(String args[])**
- La ejecución del programa Java comenzará por el código contenido en este método.

- System.out.println("Hola Mundo!");
- La llamada a System.out.println() que permite mostrar en pantalla una serie de caracteres.

Variables

- Las **variables** identifican datos mediante un nombre simbólico, haciendo referencia a una **dirección de memoria principal** en los que se sitúan los datos.

- Los nombres utilizados para identificar a las variables deben cumplir una serie de condiciones:
  - ✓ No pueden empezar por un dígito numérico.
  - ✓ No pueden utilizarse espacios, y los únicos caracteres especiales válidos son el guión bajo (_) y el símbolo del dólar ($).
  - ✓ Son sensibles a las mayúsculas y minúsculas, es decir, las variables "suma" y "Suma" se consideran variables distintas.

Variables

- No podemos utilizar como nombres las **palabras reservadas** de **JAVA**:

```
abstract  default  goto
assert    do       if
boolean   double   implements
break     else     import
byte      enum     instanceof
case      extends  int
catch     false    interface
char      final    long
class     finally  native
const     float    new
continue  for      null
package   synchronized
private   this
protected throw
public    throws
return    transient
short     true
static    try
strictfp  void
super     volatile
switch    while
```

Variables

- Nombenclatura:
  - ✓ Los nombres de las variables han de empezar por una letra minúscula.
  - ✓ Cuando el nombre de una variable está formado por más de una palabra, se suele utilizar una letra mayúscula para distinguir el comienzo de las palabras. Por ejemplo: sumaTotal.
  - ✓ Es **recomendable** utilizar nombres que hagan referencia al contenido que va a almacenar para facilitar la comprensión del código. Es mucho más claro utilizar el nombre "suma" que "s".
  - ✓ Ejemplos de nombres de variables válidos: indice, ventas, compras, saldoGeneral, importetotal, contador_lineas, $valor, num2.
  - ✓ Ejemplos de nombres de variables no válidos: 3valores, suma&total, super, edad media.

Variables

- Podemos definir variables:
  - tipoDato nombreVariable;
  - tipoDato nombreVariable1, nombreVariable2, nombreVariable3;

- Ejemplos de declaraciones de variables:
  - int num1, num2, suma;
  - char letraNIF;
  - String texto;
  - boolean mayorEdad;

Variables

- Asignaciones:
  - Inicialización: Se le asigna un valor a la variable en el momento de su inicialización
    - int num1 = 34;
    - int doble = num1 * 2;
    - String saludo = "Hola";
    - char letraA = 'A', letraB = 'B';
  - PostDefinición: Se le asignará un valor a la variable anteriormente definida, modificando cualquier valor previo:
    - nombreVariable = valor;
    - int a=5, b=0, c;
    - b = a * 3; // Se cambia el valor de b a 15
    - c = a; // Se guarda en c el valor de a que es 5
    - a = a + 6; // Se suma 6 al valor que tenía a.
    - // Ahora vale 11
    - b = a - c; // b guarda 11 - 5 que es 6

Variables

- Ámbito de una variable:
  - Las variables pueden ser utilizadas en el bloque de código en el que han sido definidas,es decir:

```
public class AmbitoVariables
{
    static int variableGlobal;
    public static void main(String[] args)
    {
        int variableDelMain = 10;
        /*Aquí se pueden usar variableGlobal y  variableDelMain. No se puede usar  variableDeOtroMetodo */
        System.out.println("variableGlobal " + variableGlobal);
        System.out.println("variableDelMain " + variableDelMain);
        otroMetodo();
    }
    static void otroMetodo()
     {
        int variableDeOtroMetodo=90;
        /* Aquí se pueden usar variableGlobal y  variableDeOtroMetodo. No se puede usar  variableDelMain */
        System.out.println("variableGlobal " + variableGlobal);
        //System.out.println("variableDelMain " + variableDelMain);
        System.out.println("variableDeOtroMetodo " + variableDeOtroMetodo);
    }
}
```

Tipos de datos en JAVA

- **Números enteros:** Representan a los números enteros (sin parte decimal) con signo (pueden ser positivos o negativos). Se dispone de varios tipos de datos, ocupando cada uno de ellos un espacio distinto en memoria. Cuanta más capacidad de almacenamiento, más grande es el rango de valores permitidos, aunque ocupará más espacio de memoria principal. Se dispone de los siguientes tipos:
  - ✓ **byte**: Ocupan 8 bits (1 byte), permitiendo almacenar valores entre -128 y 127.
  - ✓ **short**: Ocupan 16 bits (2 bytes), permitiendo almacenar valores entre -32.768 y 32.767.
  - ✓ **int**: Ocupan 32 bits (4 bytes), permitiendo almacenar valores entre -2.147.483.648 y 2.147.483.647. Es el tipo de datos por defecto para los valores numéricos enteros. Este tipo de datos es lo suficientemente grande para almacenar los valores numéricos que vayan a usar tus programas. solo se suelen usar los tipos anteriores si se pueden producir problemas con el espacio de memoria.
  - ✓ **long**: Ocupan 64 bits (8 bytes), permitiendo almacenar valores entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.

- **Números reales**: Representan a los números reales con parte decimal y signo positivo o negativo. Hay dos tipos de datos numéricos reales que permiten obtener mayor o menor precisión. Utilizan un método para almacenar los datos que puede ocasionar que el valor original varíe levemente del valor almacenado realmente. Cuanta más precisión se utilice, habrá menor variación.
  - ✓ **float**: Ocupan 32 bits (4 bytes). Se le denomina de simple precisión. Almacenan valores desde -3.40282347E+38 a + 3.40282347E+38
  - ✓ **double**: Ocupan 64 bits (8 bytes). Se le denomina de doble precisión. Es el tipo de datos por defecto para los valores numéricos reales. Almacenan valores desde: - 1.79769313486231570E+308 a +1.79769313486231570E+308

Operadores

- **Operadores Aritméticos**:
  - ✓ + (suma)
  - ✓ - (resta)
  - ✓ * (multiplicación)
  - ✓ / (división entera o con decimales según operandos)
  - ✓ % (resto de la división)

- Ejemplos:
  - ✓ 4 + 3
  - ✓ 8 - 5 + 2
  - ✓ 6 * 2 / 3
  - ✓ 8.5 - 3 + 4.3

Operadores

- **Operadores Relacionales**
  - Los operadores relacionales permiten comparar dos valores numéricos.
  - > (mayor que)
  - >= (mayor o igual que)
  - < (menor que)
  - <= (menor o igual que)
  - == (igual que)
  - != (distinto de)
- Ejemplos:
  - 4 > 3 resulta true.
  - 7 <= 2 resulta false.
  - 5 + 2 == 4 + 3 resulta true.
  - 4 * 3 != 12 resulta false.

Operadores

- **Operadores Lógicos**
  - Los operadores lógicos permiten unir valores o expresiones lógicas, obteniendo como resultado si es verdadera o falsa la expresión combinada.
  - && (Y lógico - conjunción) => Resulta true solo si ambos operandos son true.
  - || (O lógico - disyunción) => Resulta true si al menos uno de los operandos son true.
  - ! (NO lógico) => Resulta true si el operando es false.

- Ejemplos:
  - 4 > 3 && 5 <= 5 resulta true, porque las dos expresiones son true.
  - 4 > 3 && 5 != 5 resulta false, porque al menos una expresión es false.
  - 4 > 3 || 5 != 5 resulta true, porque al menos una expresión es true.
  - 4 > 3 && !(5 != 5) resulta true, porque las dos expresiones son true.

Operadores

- **Operadores Incrementales**
  - ✓Nos permiten incrementar las variables en una unidad
  - ✓++ (incrementa en 1 el valor de una variable).
  - ✓-- (decrementa en 1 el valor de una variable).

- Ejemplo:
  - ✓int x=5, y=5, z;
  - ✓z=x++; /* z vale 5, x vale 6 porque primero se asigna
  - ✓el valor de x a z después se incrementa x */
  - ✓z=++y; /* z vale 6, y vale 6 porque primero incrementa
  - ✓la variable y, después se asigna el valor a z */

Operadores

- **Operador Condicional.**
  - Permite asignar a una variable un valor u otro dependiendo de una expresión condicional. El formato es el siguiente:
  - variable = condición ? valor1 : valor2;
  - La condición que se indica debe ser una variable booleana o una expresión condicional que resulte un valor de tipo booleano (true o false).
- Ejemplo:
  - mensaje = (num1 >= 0) ? "Positivo" : "Negativo";

Operadores: Prioridad

- Los operadores del mismo nivel, fila, tienen la misma prioridad.
- Los operadores de distinto nivel, inferior, tienen menor prioridad.

()	**[]**	**.**			
-(unario)	**--**	**++**	**!**		
new (tipo)					
*****	**/**	**%**			
+	**-**				
>	**>=**	**<**	**<=**		
==	**!=**				
&&					
||					
? :					
=	**+=**	**-=**	***=**	**/=**	**%=**

Conversiones de tipo

- **Conversiones implícitas**:
  - Se realizan de manera automática, es decir, el valor o expresión que se va a asignar a una variable es convertido automáticamente por el compilador.

- Ejemplos:

```
// Declaraciones.
int k = 5, p;
short s = 10;
char c = 'ñ';
float h;

// Conversiones implícitas.
p = c; // Conversión implícita de char a int.
h = k; // Conversión implícita de int a float.
k = s; // Conversión implícita de short a int.
```

Conversiones de tipo

- **Conversiones explícita**:
  - Cuando no se cumplan las condiciones para una conversión implícita, ésta podrá realizarse de manera explícita utilizando la expresión:
  - variable_destino = (tipo_destino) dato_origen;

- Ejemplos:
  - // Declaraciones.
  - char c;
  - byte k;
  - int p = 400;
  - double d = 34.6;
  - // Conversiones explícitas.
  - c = (char)d; // Se elimina la parte decimal (trunca), no se redondea.
  - k = (byte)p; // Se provoca una pérdida de datos, pero la conversión es posible.

Introducción a la POO.

PROGRAMACIÓN

Contenidos:

1) Antecedentes y características de Java.
2) Historia de Java.
3) La Programación Orientada a Objetos (POO) y Java.
4) Fortalezas de Java.
5) Los Bytecodes.
6) Tipos de Aplicaciones en Java.
7) Principios de la Programación Orientada a Objetos.
8) Características de la Programación Orientada a Objetos.
9) Los objetos
10) Clases, atributos y métodos
11) Encapsulación y visibilidad
12) Herencia

Antecedentes y características de Java: ¿Qué 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 (1995) 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 *Maquina 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”.

Antecedentes y características de Java: ¿Qué es Java?

Las características principales de lenguaje Java se resumen a continuación:

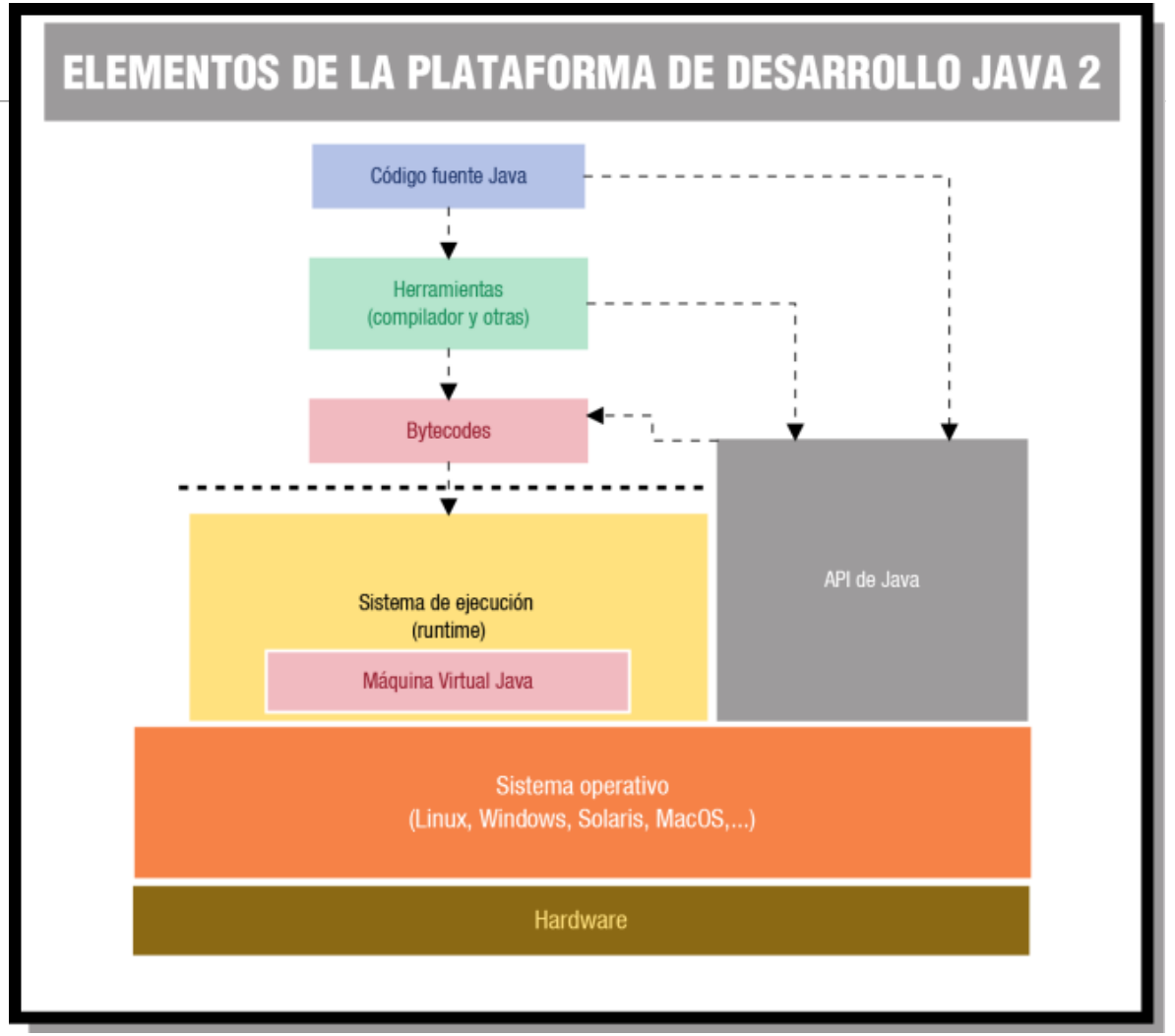
- El código generado por el compilador Java es independiente de la arquitectura.
- Está totalmente orientado a objetos.
- Su sintaxis es similar a C y C++.
- Es distribuido, preparado para aplicaciones TCP/IP.
- Dispone de un amplio conjunto de bibliotecas.
- Es robusto, realizando comprobaciones del 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

Historia de Java

Java 2 es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluyendo 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.

Historia de Java



La Programación Orientada a Objetos (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** (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.), **herencia** (mecanismo que permite derivar una clase de otra, de manera que extienda su funcionalidad) y **polimorfismo** (capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente). Por ejemplo, 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 (los peces se mueven nadando, mientras que las aves lo hacen volando).

La Programación Orientada a Objetos (POO) y Java

Los patrones o tipos de objetos se denominan clases (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) y los objetos que utilizan estos patrones o pertenecen a dichos tipos, se identifican con el nombre de instancias.

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.)

Fortalezas de Java

Las dos principales características que distinguen a Java de otros lenguajes son la *independencia de la plataforma y la posibilidad de trabajar en red* o, mejor, la posibilidad de crear aplicaciones que trabajan en red. Adicionalmente, Java ofrece dos características muy valiosas, su seguridad y simplicidad:

a) **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.

Fortalezas de Java

b) **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.

c) **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.

Fortalezas de Java

b) **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.

c) **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.

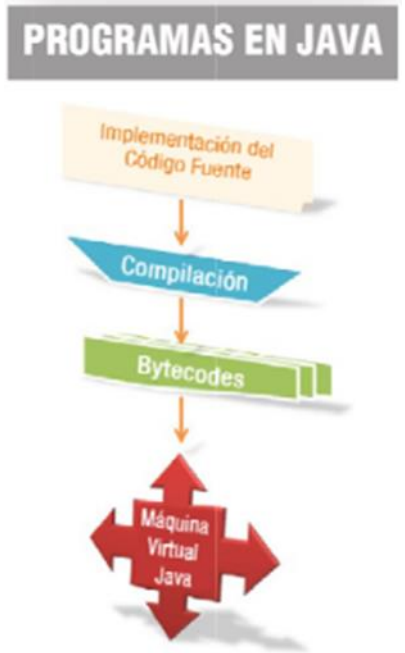
Fortalezas de Java

d) **Simplicidad**: Aunque Java *no 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. En relación a C o C++, Java es algo más sencillo de entender porque elimina la aritmética de **punteros** (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), los registros, la definición de tipos, la gestión de memoria, etc., lo que reduce la posibilidad de cometer errores comunes en los programas. Muy relacionado con la simplicidad que aporta Java está la incorporación de un elemento muy útil: 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 o memoria que queda desperdiciada al usar los métodos de gestión de memoria. Puede ser interna o externa). 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.

Los Bytecodes

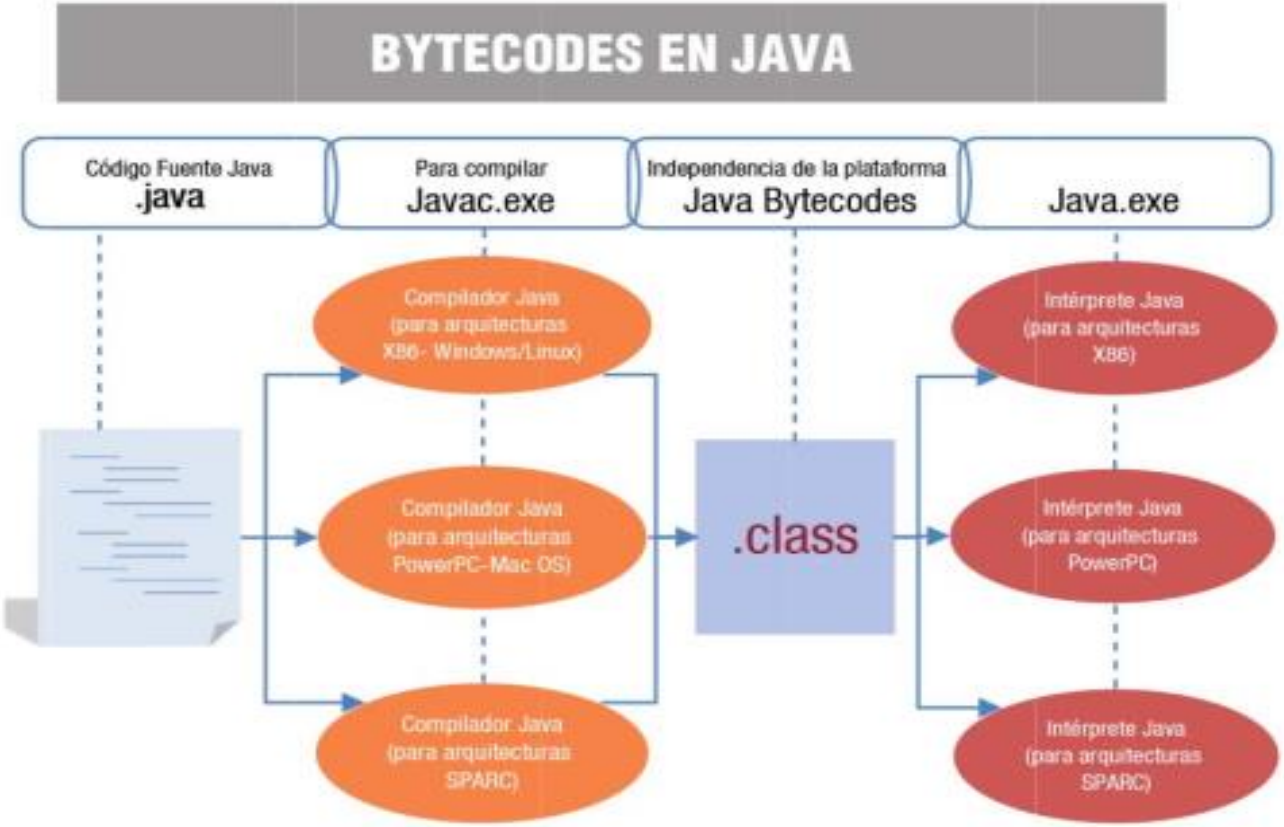
Un programa escrito en Java no es directamente ejecutable. Es necesario que el código fuente sea interpretado por la Máquina Virtual Java. A continuación se detallan los pasos que sigue este proceso:

- Una vez escrito el código fuente (archivo con extensión .Java), se precompila generando el Bytecode o archivo compatible con el intérprete de la JVM. Este intérprete genera el código nativo para la plataforma sobre la que se ejecuta el programa.
- En el proceso de precompilación, existe un verificador de código de bytes que se asegura de que se cumplen ciertas 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.

Los Bytecodes



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, como los creados con 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 (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.

Tipos de Aplicaciones en Java

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.

Tipos de Aplicaciones en 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 como dispositivos móviles. Los juegos Java creados para teléfonos móviles son midlets.

Principios de la Programación Orientada a Objetos

Dentro de las distintas formas de hacer las cosas en programación, distinguimos dos paradigmas fundamentales:

- Programación Estructurada, en la que se crean funciones y procedimientos que definen las acciones a realizar, y que posteriormente forman los programas.
- Programación Orientada a Objetos, que considera los programas en términos de objetos y todo gira alrededor de ellos.

Pero ¿en qué consisten realmente estos paradigmas? Veamos estos dos modelos de programación con más detenimiento.

Inicialmente se programaba aplicando las técnicas de programación tradicional, también conocidas como Programación Estructurada. El problema se descomponía en unidades más pequeñas hasta llegar a acciones o verbos muy simples y fáciles de codificar. Por ejemplo, en la resolución de una ecuación de primer grado, lo que hacemos es descomponer el problema en acciones más pequeñas o pasos diferenciados:

- Pedir valor de los coeficientes.
- Calcular el valor de la incógnita.
- Mostrar el resultado.

Si nos damos cuenta, esta serie de acciones o pasos diferenciados no son otra cosa que verbos; por ejemplo el verbo pedir, calcular, mostrar, etc.

Principios de la Programación Orientada a Objetos

Sin embargo, la Programación Orientada a Objetos aplica de otra forma diferente la técnica de programación "divide y vencerás". Para ello lo que hace es descomponer, en lugar de acciones, en objetos, propiedades y métodos. El principal objetivo sigue siendo descomponer el problema en problemas más pequeños, que sean fáciles de manejar y mantener, fijándonos en el escenario del problema e intentando reflejarlo en nuestro programa. Por ejemplo, podríamos pensar en un objeto "ecuación" con las propiedades "coeficientes" y el método "calcular resultado". Se dice que la Programación Orientada a Objetos aborda los problemas de una forma más natural, entendiendo como natural que está más en contacto con el mundo que nos rodea.

La Programación Estructurada se centra en el conjunto de acciones a realizar en un programa, haciendo una división de procesos y datos. La Programación Orientada a Objetos se centra en la relación que existe entre los datos y las acciones a realizar con ellos, encerrándolos en el concepto de **objeto**, tratando de realizar una abstracción lo más cercana al mundo real.

Beneficios de la Programación Orientada a Objetos

Las principales ventajas de la POO son:

- **Comprensión:** Los conceptos del espacio del problema se hayan reflejados en el código del programa, por lo que la mera lectura del código nos describe la solución del problema en el mundo real.
- **Modularidad:** Facilita la modularidad del código, al estar las definiciones de objetos en módulos o archivos independientes, hace que las aplicaciones estén mejor organizadas y sean más fáciles de entender.
- **Fácil mantenimiento:** Cualquier modificación en las acciones queda automáticamente reflejada en los datos, ya que ambos están estrechamente relacionados. Esto hace que el mantenimiento de las aplicaciones, así como su corrección y modificación sea mucho más fácil. Por ejemplo, podemos querer utilizar un algoritmo más rápido, sin tener que cambiar el programa principal. Por otra parte, al estar las aplicaciones mejor organizadas, es más fácil localizar cualquier elemento que se quiera modificar y/o corregir. Esto es importante ya que se estima que los mayores costes de software no están en el proceso de desarrollo en sí, sino en el mantenimiento posterior de ese software a lo largo de su vida útil.
- **Seguridad:** La probabilidad de cometer errores se ve reducida, ya que no podemos modificar los datos de un objeto directamente, sino que debemos hacerlo mediante las acciones definidas para ese objeto. Imaginemos un objeto lavadora. Se compone de un motor, tambor, cables, tubos, etc. Para usar una lavadora no se nos ocurre abrirla y empezar a manipular esos elementos, ya que lo más probable es que se estropee. En lugar de eso utilizamos los programas de lavado establecidos. Pues algo parecido con los objetos, no podemos manipularlos internamente, sólo utilizar las acciones que para ellos hay definidas.

Beneficios de la Programación Orientada a Objetos

- **Reusabilidad:** Los objetos se definen como entidades reutilizables, es decir, que los programas que trabajan con las mismas estructuras de información, pueden reutilizar las definiciones de objetos empleadas en otros programas, e incluso las acciones definidas sobre ellos. Por ejemplo, podemos crear la definición de un objeto de tipo persona para una aplicación de negocios y deseamos construir a continuación otra aplicación, digamos de educación, en donde utilizamos también personas, no es necesario crear de nuevo el objeto, sino que por medio de la reusabilidad podemos utilizar el tipo de objeto persona previamente definido

Características de la Programación Orientada a Objetos

Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción**: Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la clase. Básicamente, una clase es un tipo de dato que agrupa características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global del conjunto. Por ejemplo, si pensamos en una clase Vehículo que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como Coche y Camión. Entonces se dice que Vehículo es una abstracción de Coche y de Camión.
- **Modularidad**: Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.

Características de la Programación Orientada a Objetos

- **Encapsulación**: También llamada "ocultamiento de la información". La encapsulación o encapsulamiento es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto Persona y otro Coche. Persona se comunica con el objeto Coche para llegar a su destino, utilizando para ello las acciones que Coche tenga definidas como por ejemplo conducir. Es decir, Persona utiliza Coche pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones

Características de la Programación Orientada a Objetos

- **Jerarquía**: Mediante esta propiedad podemos definir relaciones de jerarquías, entre clases y objetos. Las dos jerarquías más importantes son la jerarquía "es un", llamada generalización o especialización, y la jerarquía "es parte de", llamada agregación. Conviene detallar algunos aspectos:
  - ✓ La generalización o especialización, también conocida como herencia, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase CochedeCarreras a partir de la clase Coche, y así sólo tendremos que definir las nuevas características que tenga.
  - ✓ La agregación, también conocida como inclusión, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un Coche está formado por Motor, Ruedas, Frenos y Ventanas. Se dice que Coche es una agregación y Motor, Ruedas, Frenos y Ventanas son agregados de Coche.
- **Polimorfismo**: Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase Animal y la acción de expresarse. Nos encontramos que cada tipo de Animal puede hacerlo de manera distinta, los Perros ladran, los Gatos maullan, las Personas hablamos, etc.

Los objetos

Un objeto es un conjunto de datos con las operaciones definidas para ellos. Los objetos tienen un estado y un comportamiento.

Los objetos tienen unas características fundamentales que los distinguen:

- **Identidad:** Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- **Estado:** El estado de un objeto viene determinado por parámetros o atributos que lo describen y los valores de éstos. Por ejemplo, si tenemos un objeto Coche, el estado estaría definido por atributos como Marca, Modelo, Color, Cilindrada, etc.
- **Comportamiento:** Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto Coche, el el comportamiento serían acciones como: arrancar(), parar(), acelerar(), frenar(), etc.

Los objetos: Propiedades y métodos

Todo objeto tiene un estado y un comportamiento. Concretando un poco más, las partes de un objeto son:

- **Campos, Atributos o Propiedades**: Parte del objeto que almacena los datos. También se les denomina Variables Miembro. Estos datos pueden ser de cualquier tipo primitivo (boolean, char, int, double, etc) o ser a su vez ser otro objeto. Por ejemplo, un objeto de la clase Coche puede tener un objeto de la clase Ruedas.
- **Métodos o Funciones Miembro:** Parte del objeto que lleva a cabo las operaciones sobre los atributos definidos para ese objeto.

La idea principal es que el objeto reúne en una sola entidad los datos y las operaciones, y para acceder a los datos privados del objeto debemos utilizar los métodos que hay definidos para ese objeto. La única forma de manipular la información del objeto es a través de sus métodos. Es decir, si queremos saber el valor de algún atributo, tenemos que utilizar el método que nos muestre el valor de ese atributo. De esta forma, evitamos que métodos externos puedan alterar los datos del objeto de manera inadecuada. Se dice que los datos y los métodos están encapsulados dentro del objeto.

Los objetos: Interacción entre objetos

Dentro de un programa los objetos se comunican llamando a sus métodos. Los métodos están dentro de los objetos y describen el comportamiento de un objeto cuando recibe una llamada. En otras palabras, cuando un objeto, objeto1, quiere actuar sobre otro, objeto2, tiene que ejecutar uno de sus métodos. Entonces se dice que el objeto2 recibe un mensaje del objeto1.

Un **mensaje** es la acción que realiza un objeto. Un método es la función o procedimiento al que se llama para actuar sobre un objeto. Los distintos mensajes que puede recibir un objeto o a los que puede responder reciben el nombre de **protocolo** de ese objeto.

El proceso de interacción entre objetos se suele resumir diciendo que se ha "enviado un mensaje" (hecho una petición) a un objeto, y el objeto determina "qué hacer con el mensaje" (ejecuta el código del método). Cuando se ejecuta un programa se producen las siguientes acciones:

- Creación de los objetos a medida que se necesitan.
- Comunicación entre los objetos mediante el envío de mensajes unos a otros, o el usuario a los objetos.
- Eliminación de los objetos cuando no son necesarios para dejar espacio libre en la memoria del computador.

Clases, atributos y métodos

Un programa informático se compone de muchos objetos, algunos de los cuales comparten la misma estructura y comportamiento. Si tuviéramos que definir la estructura y comportamiento cada vez que queremos crear un objeto, estaríamos utilizando mucho código redundante. Por ello lo que se hace es crear una clase, que es una descripción de un conjunto de objetos que comparten una estructura y un comportamiento común.

Y a partir de la clase, se crean tantas "copias" o "instancias" como necesitemos. Esas copias son los objetos de la clase.

Las **clases** constan de datos y métodos que resumen las características comunes de un conjunto de objetos. Un programa informático está compuesto por un conjunto de clases, a partir de las cuales se crean objetos que interactúan entre sí.
En otras palabras, una clase es una plantilla o prototipo donde se especifican:

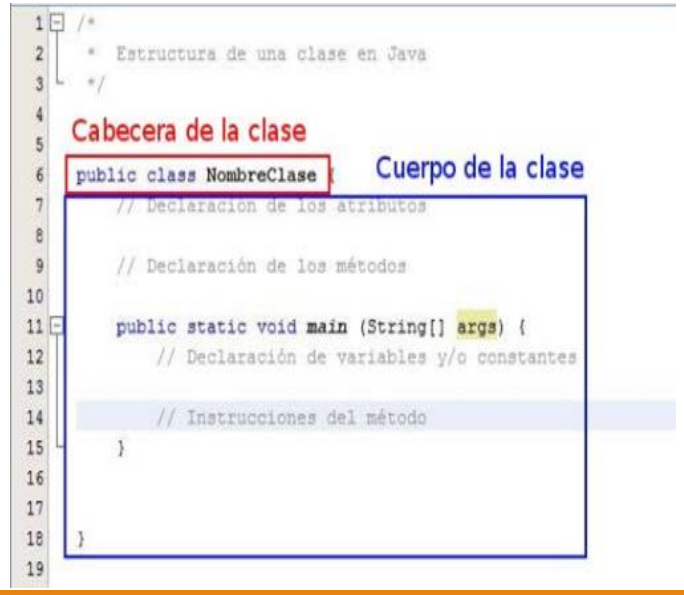
- Los **atributos** comunes a todos los objetos de la clase.
- Los **métodos** que pueden utilizarse para manejar esos objetos.

Clases, atributos y métodos

Para declarar una clase en Java se utiliza la palabra reservada class. La declaración de una clase está compuesta por:

- **Cabecera de la clase**: La cabecera es un poco más compleja que como aquí definimos, pero por ahora sólo nos interesa saber que está compuesta por una serie de modificadores, en este caso hemos puesto public que indica que es una clase pública a la que pueden acceder otras clases del programa, la palabra reservada class y el nombre de la clase.
- **Cuerpo de la clase**: En él se especifican encerrados entre llaves los atributos y los métodos que va a tener la clase.

El método main() se utiliza para indicar que se trata de una clase principal, a partir de la cual va a empezar la ejecución del programa. Este método no aparece si la clase que estamos creando no va a ser la clase principal del programa



Encapsulación y visibilidad

La **encapsulación** permite agrupar funcionalidades (métodos) y estado (datos) de una forma cohesiva. Los métodos proporcionan los mecanismos adecuados para modificar el estado y en algunos casos también serán la puerta de acceso a este. En lenguajes como Java, la forma de acceder al estado será mediante métodos tipo getXXX().

Cuando hablamos de **visibilidad** nos referimos a la ocultación o publicación tanto de la información del estado como de la implementación de la funcionalidad de una clase. La visibilidad nos permite ocultar al exterior detalles del estado o de la implementación que no queremos que sean expuestos fuera de nuestra "unidad de encapsulación".

Un ejemplo muy sencillo sería el siguiente: en un juego no queremos que se sepa la vida de los personajes, únicamente queremos que se sepa si el personaje está vivo, herido o muerto. Podríamos recurrir a una propiedad privada que almacene los puntos de vida del personaje y un método público que nos permita obtener su estado. El método calcularía el estado en función de los puntos de vida del jugador pero los puntos de vida no quedarían expuestos en ningún momento.

El concepto de visibilidad suele acompañar al de encapsulación hasta el punto de que a veces se asume que encapsulación y visibilidad son sinónimos.

Relaciones entre clases. Herencia

Las clases, igual que los objetos, no existen de modo aislado. La Orientación a Objetos (POO) intenta modelar aplicaciones del mundo real tan fielmente como sea posible y por lo tanto debe reflejar estas relaciones entre clases y objetos.

De todas las relaciones posibles entre las distintas clases y objetos, hay que destacar por su importancia en O.O. la relación de herencia. La relación de herencia es una relación entre clases que comparten su estructura y el comportamiento.

- Se denomina **herencia simple** a la relación en que una clase comparte la estructura y comportamiento de una sola clase.
- Se denomina **herencia múltiple** a la relación en que una clase comparte la estructura y comportamiento de varias clases.

Para que un lenguaje de programación pueda ser considerado orientado a objetos, debe implementar el mecanismo de herencia.

La clase superior de la jerarquía, en cada relación, se denomina **superclase**, clase base ó clase padre y, la clase que hereda de la superclase, se denomina **subclase**, clase derivada ó clase hija.

Relaciones entre clases. Herencia

La herencia es:
HERENCIA = TRANSMISIÓN + REDEFINICIÓN + ADICIÓN
Las implicaciones de la herencia sobre los objetos de las clases involucradas son las siguientes:

- Sobre los objetos de la superclase A: ninguna
- Sobre los objetos de la subclase B:
  - ✓ Contienen todos los atributos que contienen los objetos de la superclase.
  - ✓ Contiene los atributos añadidos de la subclase.
  - ✓ Responden a los mensajes que corresponden con métodos transmitidos a la subclase; es decir, como dicta el método de la superclase.
  - ✓ Responden a los mensajes que corresponden con métodos añadidos a la subclase; es decir, como dicta el método de la subclase.
  - ✓ Responden a los mensajes que corresponden con métodos redefinidos en la subclase; es decir, como dicta el método de la subclase anulando el método de la superclase.

Sintaxis de la herencia en Java.

```
modificador NombreSubClase extends NombreSuperClase{
                    …
}
```