U.D.6

Programación orientada a objetos. Objetos

(1ª parte: Introducción a JAVA)

PRO – IES EL MAJUELO

ÍNDICE

1. Introducción a JAVA

- Lenguaje de POO, desarrollado por *Sun Microsystems* (principios de los 90).
- No creado originalmente para *internet* Objetivo: lenguaje independiente de la plataforma para el desarrollo de electrónica de consumo.
- Mucha de su sintaxis es de C y C++, el modelo de objetos es más simple, elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.
- Aplicaciones precompiladas en un *bytecode* (JVM), la compilación en código máquina nativo es posible.
- La implementación original y de referencia del compilador, la máquina virtual y las bibliotecas de clases de Java fueron desarrollados por *Sun Microsystems* en 1995. Desde entonces, *Sun* ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del *Java Community Process*, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de *Sun*, algunas incluso bajo licencias de software libre.
- Entre diciembre de 2006 y mayo de 2007, *Sun* liberó tecnologías bajo GNU GPL. En 2009 *Oracle* compra *Sun*.

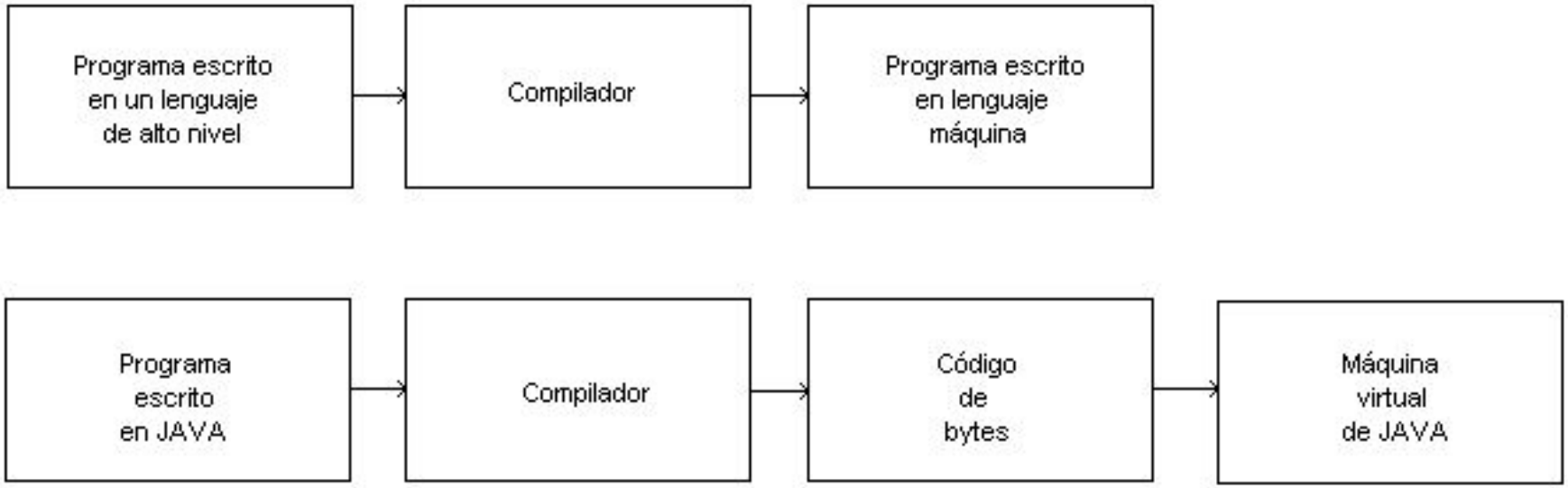
Características de Java

- Sencillo.
- Orientado a objetos.
- Distribuido.
- Interpretado.
- Robusto.
- Seguro.
- Arquitectura neutral.
- Portable.
- Alto rendimiento.
- Multihilo.
- Dinámico.

Java interpretado

- El código fuente escrito por un programador se traduce a código máquina.
  - El código fuente está escrito en un lenguaje de programación, y el traductor (compilador, intérprete) se encarga de la transformación.
- Los traductores son programas específicos para un lenguaje de programación.
  - No es posible compilar un programa escrito en Java con un compilador de C.
- Los traductores transforman el programa fuente en un programa ejecutable por la máquina destino.

- El lenguaje máquina que genera Java es un lenguaje intermedio interpretable por una máquina virtual instalada en el ordenador donde se va a ejecutar.
  - Los programas Java se componen de una serie de ficheros `.class`, que son ficheros *bytecode* que contienen las clases.


Programa escrito en un lenguaje de alto nivel
Compilador
Programa escrito en lenguaje máquina
Programa escrito en JAVA
Compilador
Código de bytes
Máquina virtual de JAVA

Aplicaciones y *Applets*

- Aplicaciones:
  - Programas autónomos independientes, tal y como cualquier programa escrito utilizando lenguajes de alto nivel.
  - Se pueden ejecutar en cualquier computadora con un traductor de Java.
- *Applets*:
  - Tipo especial de programas en Java que se pueden ejecutar directamente en un navegador web compatible Java.
  - Adecuados para desarrollar proyectos web.
  - Programas incrustados (empotrados, "*embedded*") dentro del código HTML.

Limitaciones de los *applets* (seguridad)

- No se pueden leer o escribir en el sistema de archivos.
- No pueden establecer conexiones entre computadoras (excepto con el servidor donde están alojados).
- No pueden ejecutar programas de la computadora donde reside el navegador.

JDK: Java Development Kit

- Contiene aplicaciones de consola y herramientas de compilación, documentación y depuración.
- Incluye el JRE (*Java Runtime Environment*) que consta de los mínimos componentes necesarios para ejecutar una aplicación Java (máquina virtual, librerías de clases).

Herramientas de consola JDK

- `java`: Máquina virtual de Java.
- `javac`: Compilador de Java.
- `javap`: Desensamblador de clases.
- `jdp`: Depurador de consola de Java.
- `javadoc`: Generador de documentación.
- `appletviewer`: Visor de *Applets*.

Bibliotecas de clases

- Cada JDK, se acompaña de una serie de bibliotecas con clases estándar que valen como referencia para todos los programadores en Java.
- Estas clases se pueden incluir en los programas Java. Portabilidad.
- Bien documentadas (*web*), y organizadas en paquetes y en un gran árbol de herencia.
- Conjunto de paquetes (o bibliotecas) API de Java (*Application Programming Interface*).

Paquete	**Descripción**
java.applet	Contiene clases para la creación de applets.
java.awt	Contiene clases para crear interfaces de usuario con ventanas.
java.io	Contiene clases para manejar la entrada/salida.
java.lang	Contiene clases variadas pero imprescindibles para el lenguaje, como Object, Thread, Math...
java.net	Contiene clases para soportar aplicaciones que acceden a redes TCP/IP.
java.util	Contiene clases que permiten el acceso a recursos del sistema, etc.
java.swing	Contiene clases para crear interfaces de usuario mejorando la AWT.

Herramientas gráficas

- AWT.
  - *Abstract Window Toolkit* es un *kit* de herramientas de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma, original de Java.
- Swing.
  - Biblioteca gráfica para Java.
  - Independiente de la plataforma.

Creación de aplicaciones utilizando Java

Entornos de desarrollo integrado (IDE) para desarrollar aplicaciones JAVA:

- Eclipse.
  - IBM, actualmente código abierto.
  - Multilenguaje (*plugins*).
- NetBeans.
  - Sun (actualmente Oracle), código abierto.

2. POO como base de JAVA

2.1. Clases, atributos, métodos y visibilidad

- Lo que caracteriza a la POO es el uso de clases.
- Una clase es un tipo de dato definido por el programador, donde se agrupan los datos que nos interesan sobre algo, así como las operaciones que se pueden realizar sobre dichos datos.
- Por ello una clase está formada por:
  - Atributos: Datos que contiene la clase.
    - Puede tener cualquier número de atributos o no tener ninguno.
    - Se declaran con un nombre y el tipo de dato correspondiente.
  - Métodos: Definen el comportamiento de la clase.
    - Permiten cambiar los datos de los atributos que forman la clase.
- A los atributos y métodos se les conoce como “miembros de una clase”.
  - Atributos: “Variables miembro”.
  - Métodos: “Funciones miembro”.

Acceso a miembros

- Cualquier método de una clase puede acceder a cualquier miembro de dicha clase.
- El acceso a miembros de otra clase dependerá de los modificadores de visibilidad que tengan. Existen cuatro niveles de acceso:
  - Público (*public*): Cualquier clase puede acceder a ellos.
  - Paquete (*package*, no se pone): Cualquier clase del mismo paquete puede acceder a ellos.
  - Protegido (*protected*): Cualquier clase del mismo paquete o que herede de ésta puede acceder a ellos.
  - Privado (*private*): No son accesibles fuera de la clase en la que están definidos.
- Las clases también pueden establecer permisos de visibilidad (por defecto, si no se indica nada, *package*).

2. POO como base de JAVA

2.2. Objetos, estado, comportamiento e identidad. Mensajes.

- El objeto es la unidad básica de la POO. Es la declaración de una variable donde el tipo es una clase.
- Son considerados “instancias de una clase”:
  - El estado son los valores concretos que tiene un objeto en cada uno de sus atributos.
  - El comportamiento está formado por los métodos que forman la clase (fijan las operaciones que puede realizar un objeto).
  - La identidad es el nombre que se le ha dado a un objeto (identificador).
- Se denomina “mensaje” a la llamada de métodos de una clase.
- Un POO se basa fundamentalmente en el uso de objetos, enviando mensajes para llevar a cabo determinadas tareas.

2. POO como base de JAVA

2.3. Encapsulado y visibilidad.

- Un objeto es el encapsulamiento de un conjunto de operaciones (métodos) y de un estado (datos).
- El encapsulamiento consigue que los programadores que vayan a utilizar dicha clase sólo tengan que conocer cómo llamar a los métodos sin tener que saber cómo están implementados.
- La encapsulación está asociada con la visibilidad, determinando qué miembros de la clase son visibles fuera de la clase (o incluso fuera del paquete).

- Lo aconsejable es que los datos sean privados y que se puedan acceder a ellos a través de los métodos de la propia clase.

Estructura de un programa-aplicación en JAVA

- Una sentencia de paquete (*package*), opcional.
- Una, ninguna o varias sentencias de importación (*import*).
- Comentarios.
- Declaraciones de las clases privadas deseadas (puede no haber ninguna).
- Una declaración de clase pública.
  - Contendrá el método *main*, que es por donde empezará la ejecución del programa.

3. Elementos intervinientes en un programa JAVA

- Identificadores.
- Palabras reservadas.
- Variables.
- Tipos de datos. Conversiones.
- Literales.
- Constantes.
- Operadores y expresiones.
- Comentarios.

Identificadores

- El primer carácter tiene que ser letra, número o algún carácter especial permitido ($, Ç, _).
- Admiten caracteres acentuados y ñ.
- No admiten espacios en blanco.
- No pueden coincidir con palabras reservadas.
- Distinguen entre mayúsculas y minúsculas.
- No hay límite de tamaño.
- Convenio: Si tienen más de una palabra se pone la primera letra en mayúsculas para distinguirlas (excepto la primera).
- Recomendación: Deben tener un nombre significativo.

Palabras reservadas

- Identificadores usados por el lenguaje. Tienen un significado especial para el compilador. Sólo se pueden usar para lo que se ha establecido en el lenguaje.

```
abstract, boolean, break, byte, case, catch,
   char, class, const, continue, default, do,
   double, else, extends, false, final, floar,
   for, goto, if, implements, import, instanceof,
   int, interface, long, new, null, private,
   protected, public, return, short, static, supe,
   switch, this, throw, true, try, void, while,
   ……..
```

Variables

- Espacio de memoria identificado por un nombre que sirven para guardar un dato (que puede cambiar a lo largo de la ejecución del programa).
- Las variables se deben:
  - Declarar.
  - Inicializar.
  - Utilizar.
- Declaración: `tipo nombre;`
- Pueden ser:
  - Variables miembro de una clase:
    - Dentro de una clase pero fuera de cualquier método.
    - Pueden ser de tipo primitivo o referencias.
    - También se las llama variables atributo.
  - Variables locales:
    - Dentro de un método.
    - Deben estar inicializadas (Java no permite trabajar con datos “basura”).

Tipos de datos

- Proporciona una descripción al compilador de:
  - Cuánta memoria se debe asignar.
  - Qué tipo de datos se pueden almacenar (números, letras, …).
  - Qué operaciones se pueden realizar con esos datos.
- Tipos:
  - Variables de tipo primitivo: `boolean, char, byte, short, int, long, float, double`.
  - Variables de tipo referencia: No guardan un valor real sino que almacenan una dirección de memoria, informando con ello dónde están situados los datos a los que hace referencia (ejemplos: objetos, *arrays*, …).

Literales

- Numéricos enteros:

```
int edad = 24;
```

- Numéricos con decimales (se consideran `double`):

```
float sueldo = 12.45;
double altura = 1.65f; // Almacenado como float.
```

- Booleanos:

```
boolean casado = true;
```

- Caracteres:

```
char letraInicNombre = 'A';
```

- Cadenas (clase `String`):

```
String nombre="Juan González";
```

Constantes

- Espacios de memoria identificados por un nombre cuyo valor permanece invariable a lo largo de la ejecución de un programa.
- Convenio: Mayúsculas.

```
final float PI = 3.14159;
```

Operadores y expresiones

- Tipos de expresiones:
  - Numéricas: Producen un resultado de tipo numérico.
  - Alfanuméricas: Producen un resultado de tipo *String*.
  - Booleanas o lógicas: Producen un resultado de tipo *boolean*.
- Tipos de operadores:
  - Aritméticos: +, -, *, /, %.
  - De asignación: =.
  - Incrementales: ++, -- (prefijo o sufijo).
  - Relacionales: ==, !=, >, >=, <, <=.
  - Lógicos: && ó & (AND), || ó | (OR), ! (NOT).
  - Concatenación de cadenas: +.

Orden de prioridad

1. `()`
2. `++ --`
3. `new, casting`
4. `* / %`
5. `+ - !`
6. `< > <= >= == !=`
7. `instanceof`
8. `&`
9. `|`
10. `&&`
11. `||`
12. `-= += *= %= /=`

Arriba abajo, izquierda a derecha

Conversiones de tipo (*casting*)

- Implícita:
  - Se realiza de forma automática cuando se va a asignar un valor de un tipo determinado a una variable de un tipo diferente.
- Explícita:
  - Se realiza indicando al compilador el tipo al que vamos a convertir el dato "fuente".

Conversión implícita

- Puede hacerse si:
  - Los dos tipos son compatibles.
  - El tipo de la variable donde se va a guardar el dato tiene un rango de valores igual o superior que el tipo del dato que se quiere insertar, es decir, no se produce pérdida de información.
- En expresiones con variables de diferente tipo, JAVA convierte automáticamente todos los tipos a uno, que siempre será el mayor de todos (la conversión se realiza antes de realizar las operaciones).

Conversión explícita

- Pueden producir pérdida de información.

```
variable_donde_se_guarda = (tipo_de_variable_donde_se_guarda)
                dato_que_conviene_convertir;
```

Comentarios

- De bloque:
  - Pueden ocupar más de una línea.
  - Empiezan por `/*` y terminan por `*/`.
- De línea:
  - Ocupan únicamente una línea.
  - Empiezan por `//` y terminan al final de la línea.

4. Programación de la consola: entrada y salida de información

- Entrada (captura) de datos a través del teclado.
- Salida (visualización) de la información obtenida a través de la pantalla.
- JAVA controla la entrada y la salida mediante clases que tiene definidas en el paquete `java.io`.

Entrada de datos (teclado)

- Hay muchas clases de JAVA que permiten la entrada de datos a través del teclado.
- Dos formas principales:
  - Usando las clases *InputStreamReader* y *BufferedReader*.
  - Usando la clase *Scanner*.

InputStreamReader y BufferedReader (I)

```
InputStreamReader entrada = new InputStreamReader(System.in);
BufferedReader teclado = new BufferedReader(entrada);
```

- `System.in` indica la entrada estándar de datos (teclado).
- Para usar estas dos clases hay que importar `java.io.*`.
- La clase `InputStreamReader` controla la entrada de datos a través del teclado.
  - El objeto `entrada` recoge la información del teclado.
- La clase `BufferedReader` reserva memoria para guardar los datos que se insertan a través del teclado y los deja en dicha memoria.
  - El objeto `teclado` es capaz de guardar información en la memoria.

InputStreamReader y BufferedReader (II)

- El objeto de tipo `BufferedReader` tiene un método que es `readLine()`. Dicho método se encarga de tomar todo lo que se inserte a través del teclado hasta que se presione la tecla Intro.

Llamada a dicho método: `teclado.readLine()`

- Lo que toma a través del teclado lo hace en formato *String* (siempre).

```
String dato = teclado.readLine();
```

InputStreamReader y BufferedReader (III)

- Si la entrada de datos no es de tipo cadena, hay que convertir la información:
  - Si se quiere convertir a *int*:

    ```
    int datoEntero=Integer.parseInt(dato);
    ```
  - Si se quiere convertir a *short*:

    ```
    short datoCorto=Short.parseShort(dato);
    ```
  - Si se quiere convertir a *long*:

    ```
    long datoLargo=Long.parseLong(dato);
    ```
  - Si se quiere convertir a *float*:

    ```
    float datoReal=Float.parseFloat(dato);
    ```
  - Si se quiere convertir a *double*:

    ```
    double datoRealLargo=Double.parseDouble(dato);
    ```

Scanner

```
Scanner objetoTipoScanner = new Scanner(System.in);
```

- A través de `objetoTipoScanner` podemos usar los métodos de la clase *Scanner* (que permiten la entrada de datos).
- Esta clase tiene un método para cada tipo básico de Java con lo cual, si usamos esta clase, no hace falta hacer ninguna conversión.

```
String nombre;
float numero;
nombre = objetoTipoScanner.nextLine();
numero = objetoTipoScanner.nextFloat();
```

Salida de información (por pantalla)

`System.out.println()` □ Salta de línea

`System.out.print()`

5. Estructuras de control

- Selectivas:
  - *if*, *if-else*.
  - *switch-case-default*.
- Repetitivas:
  - *for*, *while*, *do-while*.
  - Rupturas y saltos: *break*, *continue*.
- Salida de métodos:
  - *return*.
    - Sirve `return;` para salir de métodos que no devuelven nada.