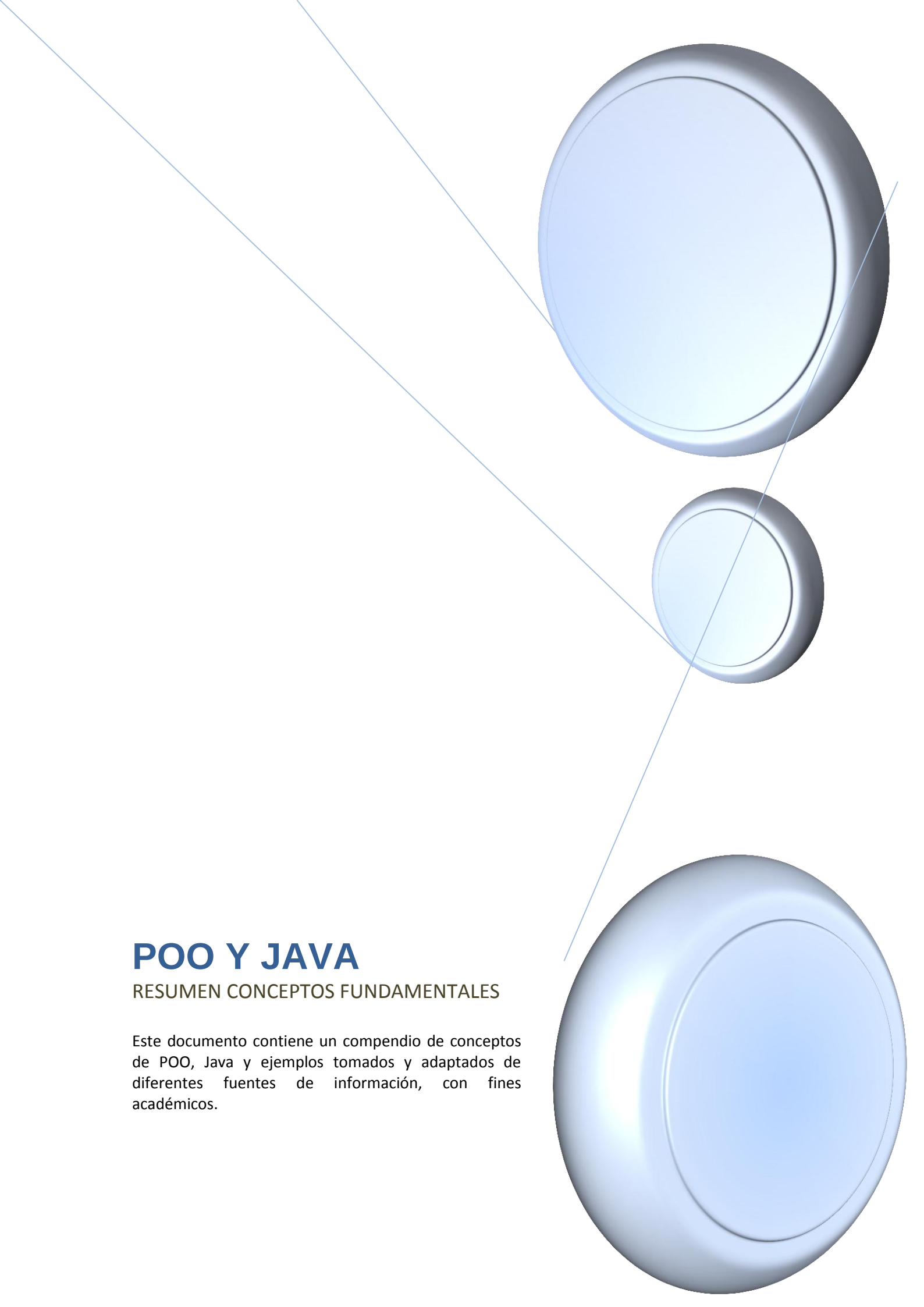

POO Y JAVA

RESUMEN CONCEPTOS FUNDAMENTALES

Este documento contiene un compendio de conceptos de POO, Java y ejemplos tomados y adaptados de diferentes fuentes de información, con fines académicos.

TABLA DE CONTENIDO

1. BREVE HISTORIA DE JAVA

Java se creó como parte de un proyecto de investigación para el desarrollo de software avanzado para una amplia variedad de dispositivos de red y sistemas embebidos. La meta era diseñar una plataforma operativa sencilla, segura, portable, distribuida y de tiempo real.

Cuando se inició el proyecto, C++ era el lenguaje del momento. Pero a lo largo del tiempo, las dificultades encontradas con C++ crecieron hasta el punto en que se pensó que los problemas podrían resolverse mejor creando una plataforma de lenguaje completamente nueva.

Se hizo uso de la arquitectura y diseño de una amplia variedad de lenguajes como Eiffel, SmallTalk, Objetive C y Cedar/Mesa. El resultado es un lenguaje que se ha mostrado ideal para desarrollar aplicaciones de usuario final seguras, distribuidas y basadas en red en un amplio rango de entornos desde los dispositivos de red embebidos hasta su uso para soluciones en Internet.

2. CARACTERÍSTICAS EN EL DISEÑO DE JAVA

a. Sencillo, orientado a objetos y familiar:
Sencillo, para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores. Orientado a objetos, porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor. Familiar, porque aunque se rechazó C++, se mantuvo Java lo más parecido posible a C++, eliminando sus complejidades innecesarias, para facilitar la migración al nuevo lenguaje.

b. Robusto y seguro:
Robusto, simplificando la administración de memoria y eliminando las complejidades del uso de apuntadores y aritmética de apuntadores del C. Seguro para que pueda operar en un entorno de red.

c. Independiente de la arquitertura y portable:
Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el compilador Java genera un código llamado 'bytecodes' o comúnmente conocido como código byte, un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software. Es además portable en el sentido de que es rigurosamente el mismo lenguaje en todas las plataformas. El 'bytecode' es traducido a código máquina y ejecutado por la Java Virtual Machine, que es la implementación Java para cada plataforma hardware-software concreta.

d. Alto rendimiento:
A pesar de ser interpretado, Java tiene en cuenta el rendimiento, y particularmente en las últimas versiones dispone de diversas herramientas para

su optimización. Cuando se necesitan capacidades de proceso intensivas, pueden usarse llamadas a código nativo.

e. Interpretado, multi-hilo y dinámico:
El intérprete Java puede ejecutar codigo byte en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi-hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

3. EL ENTORNO DE DESARROLLO DE JAVA

Existen distintos programas comerciales que permiten desarrollar código Java. La compañía Sun,creadora de Java, distribuye gratuitamente el Java(tm) Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java.Existe también una versión reducida del JDK, denominada JRE (Java Runtime Environment) destinada únicamente a ejecutar código Java (no permite compilar).

Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Los entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa.

4. LA JAVA VIRTUAL MACHINE

La existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de Sun a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se planteó la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código “neutro” el cual estuviera preparado para ser ejecutado sobre una “máquina hipotética o virtual”, denominada Java Virtual Machine (JVM). Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.

La JVM es el intérprete de Java. Ejecuta los “bytecodes” (ficheros compilados con extensión *.class) creados por el compilador de Java (javac). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado JIT (Just-In-Time Compiler), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Java está diseñado para que un programa escrito en este lenguaje sea ejecutado independientemente de la plataforma (hardware, software y sistema operativo) en la que se esté actuando. Esta portabilidad se consigue haciendo de Java un lenguaje medio interpretado medio compilado. El código fuente, se compila a un lenguaje intermedio cercano al lenguaje máquina pero independiente del ordenador y el sistema operativo en que se ejecuta (llamado en el mundo Java bytecodes).

Finalmente, se interpreta ese lenguaje intermedio por medio de un programa denominado máquina virtual de Java (JVM), que sí depende de la plataforma.
Los java bytecodes permiten el ya conocido "write once, run anywhere" (compila una sola vez y ejecútalo donde quieras). Podemos compilar nuestros programas a bytecodes en cualquier plataforma que tenga el compilador Java.
Los bytecodes luego pueden ejecutarse en cualquier implementación de la máquina virtual de Java (JVM). Esto significa que mientras el ordenador tenga un JVM, el mismo programa escrito en Java puede ejecutarse en Windows, Solaris, iMac, Linux, etc.

5. VARIABLES

Una variable es un nombre que contiene un valor que puede cambiar a lo largo del programa. De acuerdo con el tipo de información que contienen, en Java hay dos tipos principales de variables:

• Variables de tipos primitivos. Están definidas mediante un valor único que puede ser entero, de punto flotante, caracter o booleano. Java permite distinta precisión y distintos rangos de valores para estos tipos de variables (char, byte, short, int, long, float, double,boolean). Ejemplos de variables de tipos primitivos podrían ser: 123, 3456754, 3.1415, 12e-09, 'A', True, etc.

• Variables referencia. Las variables referencia son referencias o nombres de una información más compleja: arrays u objetos de una determinada clase. Desde el punto de vista de la función en el programa, las variables pueden ser:

• Variables miembro de una clase: Se definen en una clase, fuera de cualquier método; pueden ser tipos primitivos o referencias.

• Variables locales: Se definen dentro de un método o más en general dentro
De cualquier bloque entre llaves {}. Se crean en el interior del bloque y se destruyen al finalizar dicho bloque. Pueden ser también tipos primitivos o referencias.

6. TIPOS PRIMITIVOS DE VARIABLES

Java dispone de ocho tipos primitivos de variables: un tipo para almacenar valores true y false (boolean); un tipo para almacenar caracteres (char), y 6

tipos para guardar valores numéricos, cuatro tipos para enteros (byte, short, int y long) y dos para valores reales de punto flotante (float y double).

7. CÓMO SE DEFINEN E INICIALIZAN LAS VARIABLES

Una variable se define especificando el tipo y el nombre de dicha variable. Estas variables pueden ser tanto de tipos primitivos como referencias a objetos de alguna clase perteneciente al API de Java o generada por el usuario. Si no se especifica un valor en su declaración, las variable primitivas se inicializan a cero (salvo boolean y char, que se inicializan a false y '\0'). Análogamente las variables de tipo referencia son inicializadas por defecto a un valor especial: null.

Es importante distinguir entre la referencia a un objeto y el objeto mismo. Una referencia es una variable que indica dónde está guardado un objeto en la memoria del ordenador. Al declarar una referencia todavía no se encuentra "apuntando" a ningún objeto en particular (salvo que se cree explícitamente un nuevo objeto en la declaración), y por eso se le asigna el valor null. Si se desea que esta referencia apunte a un nuevo objeto es necesario crear el objeto utilizando el operador new. Este operador reserva en la memoria del ordenador espacio para ese objeto (variables y funciones). También es posible igualar la referencia declarada a otra referencia a un objeto existente previamente.

Un tipo particular de referencias son los arrays o vectores, sean éstos de variables primitivas (por ejemplo, un vector de enteros) o de objetos. En la declaración de una referencia de tipo array hay que incluir los corchetes []. En los siguientes ejemplos aparece cómo crear un vector de 10 números enteros y cómo crear un vector de elementos MyClass. Java garantiza que los elementos del vector son inicializados a null o a cero (según el tipo de dato) en caso de no indicar otro valor.

Ejemplos de declaración e inicialización de variables:

```
int x; // Declaración de la variable primitiva x. Se inicializa a 0
int y = 5; // Declaración de la variable primitiva y. Se inicializa a 5
MyClass unaRef; // Declaración de una referencia a un objeto MyClass.
// Se inicializa a null
unaRef = new MyClass(); // La referencia "apunta" al nuevo objeto creado

MyClass segundaRef = unaRef; // Declaración de una referencia a un objeto
//MyClass.
// Se inicializa al mismo valor que unaRef

int [] vector; // Declaración de un array. Se inicializa a null
vector = new int[10]; // Vector de 10 enteros, inicializados a 0

double [] v = {1.0, 2.65, 3.1}; // Declaración e inicialización de un vector de 3
// elementos con los valores entre llaves
```

En el ejemplo mostrado las referencias unaRef y segundaRef actuarán sobre el mismo objeto. Es equivalente utilizar cualquiera de las referencias ya que el objeto al que se refieren es el mismo.

8. VISIBILIDAD Y VIDA DE LAS VARIABLES

Se entiende por visibilidad, ámbito o scope de una variable, la parte de la aplicación donde dicha variable es accesible y por lo tanto puede ser utilizada en una expresión. En Java todas las variables deben estar incluidas en una clase. En general las variables declaradas dentro de unas llaves {}, es decir dentro de un bloque, son visibles y existen dentro de estas llaves.

Uno de los aspectos más importantes en la programación orientada a objetos (OOP) es la forma en la cual son creados y eliminados los objetos. En Java la forma de crear nuevos objetos es utilizando el operador new. Cuando se utiliza el operador new, la variable de tipo referencia guarda la posición de memoria donde está almacenado este nuevo objeto. La eliminación de los objetos la realiza el programa denominado garbage collector, quien automáticamente libera o borra la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a ese objeto. Lo anterior significa que aunque una variable de tipo referencia deje de existir, el objeto al cual apunta no es eliminado si hay otras referencias apuntando a ese mismo objeto.

9. OPERADORES DE JAVA

Operadores aritméticos

Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división (%).

Ejemplo #1

```
class OperMate
{
                  public static void main(String[] args )
                        {
                              System.out.println("7 + 5= "+ (7+5));
                              System.out.println("7 - 5= "+ (7-5));
                              System.out.println("7 * 5= "+ (7*5));
                              System.out.println("7 / 5= "+ (7/5));
                              System.out.println("7 % 5= "+ (7%5));
                        }
}
```

Ejemplo 1 - Operadores aritméticos

Operadores de asignación

Los operadores de asignación permiten asignar un valor a una variable. El operador de asignación por excelencia es el operador igual (=). La forma general de las sentencias de asignación con este operador es:

Operador	Utilización	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 – op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

variable = expression;

Ejemplo #2

```
class OperAsig
{
      public static void main(String[] args)
            {
                  int a,b;  // esta declaración se
                        //explicara posteriormente

                              a=5;
                              a+=20;
                              b=10;
                              System.out.println("a= "+ a+" b= "+b);

            }
}
```

Ejemplo 2 - Operadores de asignación

Operador condicional ?:

Este operador, tomado de C/C++, permite realizar bifurcaciones condicionales sencillas. Su forma general es la siguiente:

BooleanExpression ? res1 : res2

donde se evalúa booleanExpression y se devuelve res1 si el resultado es true y res2 si el resultado es false. Es el único operador ternario (tres argumentos) de Java.

Ejemplo #3

```
class OperCond
{
            public static void main(String[] args)
                  {
                        int a,b;
                        a=5;
                        b=7;
                        System.out.println(a >  b? a+" es mayor que
                        "+b:b+" es mayor que "+a);
                  }
}
```

Ejemplo 3 - Operadores Condicionales

Operadores incrementales

Java dispone del operador incremento (++) y decremento (--). El operador (++) incrementa en una unidad la variable a la que se aplica, mientras que (--) la reduce en una unidad. Estos operadores se pueden utilizar de dos formas:

- Precediendo a la variable (por ejemplo: ++i). En este caso primero se incrementa la variable y luego se utiliza (ya incrementada) en la expresión en la que aparece.

- Siguiendo a la variable (por ejemplo: i++). En este caso primero se utiliza la variable en la expresión (con el valor anterior) y luego se incrementa.

Operadores relacionales

Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor boolean (true o false) según se cumpla o no la relación considerada. La Tabla muestra los operadores relacionales de Java.

Operador	Utilización	El resultado es true
>	op1 > op2	si op1 es mayor que op2
>=	op1 >= op2	si op1 es mayor o igual que op2
<	op1 < op2	si op1 es menor que op2
<=	op1 <= op2	si op1 es menor o igual que op2
==	op1 == op2	si op1 y op2 son iguales
!=	op1 != op2	si op1 y op2 son diferentes

Operadores lógicos

Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false) o los resultados de los operadores relacionales. La Tabla muestra los operadores lógicos de Java. Debe notarse que en ciertos casos el segundo operando no se evalúa porque ya no es necesario (si ambos tienen que ser true y el primero es false, ya se sabe que la condición de que ambos sean true no se va a cumplir). Esto puede traer resultados no deseados y por eso se han añadido los operadores (&) y (|) que garantizan que los dos operandos se evalúan siempre.

Operador	**Nombre**	**Utilización**	**Resultado**
&&	AND	op1 && op2	true si op1 y op2 son true. Si op1 es false ya no se evalúa op2
||	OR	op1 || op2	true si op1 u op2 son true. Si op1 es true ya no se evalúa op2
!	negación	! op	true si op es false y false si op es true
&	AND	op1 & op2	true si op1 y op2 son true. Siempre se evalúa op2
|	OR	op1 | op2	true si op1 u op2 son true. Siempre se evalúa op2

Operador de concatenación de cadenas de caracteres (+)

El operador más (+) se utiliza también para concatenar cadenas de caracteres. Por ejemplo, para escribir una cantidad con un rótulo y unas unidades puede utilizarse la sentencia:

System.out.println("El total asciende a " + result + " unidades");

donde el operador de concatenación se utiliza dos veces para construir la cadena de caracteres que se desea imprimir por medio del método println(). La variable numérica result es convertida automáticamente por Java en cadena de caracteres para poderla concatenar. En otras ocasiones se deberá llamar explícitamente a un método para que realice esta conversión.

10. ESTRUCTURAS DE PROGRAMACIÓN

Las estructuras de programación o estructuras de control permiten tomar decisiones y realizar un proceso repetidas veces. Son los denominados bifurcaciones y ciclos. En la mayoría de los lenguajes de programación, este tipo de estructuras son comunes en cuanto a concepto, aunque su sintaxis varía de un lenguaje a otro. La sintaxis de Java coincide prácticamente con la utilizada en C/C++, lo que hace que para un programador de C/C++ no suponga ninguna dificultad adicional.

Sentencias o expresiones

Una expresión es un conjunto variables unidos por operadores. Son órdenes que se le dan al computador para que realice una tarea determinada.
Una sentencia es una expresión que acaba en punto y coma (;). Se permite incluir varias sentencias en una línea, aunque lo habitual es utilizar una línea para cada sentencia. Por ejemplo:

i = 0; j = 5; x = i + j;// Línea compuesta de tres sentencias

Comentarios

Existen tres formas diferentes de introducir comentarios entre el código de Java.
Los comentarios son útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además permite que cualquier persona distinta al programador original pueda comprender el código escrito de una forma más rápida. Se recomienda acostumbrarse a comentar el código desarrollado.
Java interpreta que todo lo que aparece a la derecha de dos barras “//” en una línea cualquiera del código es un comentario del programador y no lo tiene en cuenta. El comentario puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada.

La segunda forma de incluir comentarios consiste en escribir el texto entre los símbolos /*...*/. Este segundo método es válido para comentar más de una línea de código. Por ejemplo:

```
// Esta línea es un comentario
int a=1; // Comentario a la derecha de una sentencia

// Esta es la forma de comentar más de una línea utilizando
// las dos barras. Requiere incluir dos barras al comienzo de cada línea

/* Esta segunda forma es mucho más cómoda para comentar un número
elevado de líneas ya que sólo requiere modificar el comienzo y el final. */
```

En Java existe además una forma especial de introducir los comentarios (utilizando /**...*/ más algunos caracteres especiales) que permite generar automáticamente la documentación sobre las clases y packages desarrollados por el programador. Una vez introducidos los comentarios, el programa javadoc.exe (incluido en el JDK) genera de forma automática la información de forma similar a la presentada en la propia documentación del JDK.

Bifurcaciones

Las bifurcaciones permiten ejecutar una de entre varias acciones en función del valor de una expresión lógica o relacional. Se tratan de estructuras muy importantes ya que son las encargadas de controlar el flujo de ejecución de un programa. Existen dos bifurcaciones diferentes: if y switch.

Bifurcación if

Esta estructura permite ejecutar un conjunto de sentencias en función del valor que tenga la expresión de comparación (se ejecuta si la expresión de comparación tiene valor true). Tiene la forma siguiente:

```
if (booleanExpression) {
statements;
}
```

Las llaves {} sirven para agrupar en un bloque las sentencias que se han de ejecutar, y no son necesarias si sólo hay una sentencia dentro del if.

Bifurcación if else

Análoga a la anterior, de la cual es una ampliación. Las sentencias incluidas en el else se ejecutan en el caso de no cumplirse la expresión de comparación (false):

```
if (booleanExpression) {
statements1;
} else {
statements2;
}
```

Bifurcación if elseif else

Permite introducir más de una expresión de comparación. Si la primera condición no se cumple, se compara la segunda y así sucesivamente. En el caso de que no se cumpla ninguna de las comparaciones se ejecutan las sentencias correspondientes al else:

```
if (booleanExpression1) {
statements1;
} else if (booleanExpression2) {
statements2;
} else if (booleanExpression3) {
statements3;
} else {
statements4;
}
```

Sentencia switch

Se trata de una alternativa a la bifurcación if elseif else cuando se compara la misma expresión con distintos valores. Su forma general es la siguiente:

```
switch (expression) {
case value1: statements1; break;
case value2: statements2; break;
case value3: statements3; break;
case value4: statements4; break;
case value5: statements5; break;
case value6: statements6; break;
[default: statements7;]
}
```

Las características más relevantes de switch son las siguientes:

- Cada sentencia case se corresponde con un único valor de expression. No se pueden establecer rangos o condiciones sino que se debe comparar con valores concretos.
- Los valores no comprendidos en ninguna sentencia case se pueden gestionar en default, que es opcional.
- En ausencia de break, cuando se ejecuta una sentencia case se ejecutan también todas las case que van a continuación, hasta que se llega a un break o hasta que se termina el switch.

Ciclos

Un ciclo se utiliza para realizar un proceso repetidas veces. El código incluido entre las llaves {} (opcionales si el proceso repetitivo consta de una sola línea), se ejecutará mientras se cumpla unas determinadas condiciones. Hay que prestar especial atención a los ciclos infinitos, hecho que ocurre cuando la condición de finalizar el ciclo (booleanExpression) no se llega a cumplir nunca.

Ciclo while

Las sentencias statements se ejecutan mientras booleanExpression sea true.

```
while (booleanExpression) {
statements;
}
```

Ciclo for

La forma general del ciclo for es la siguiente:

```
for (inicialización; Expresión booleana; incremento) {
sentencias;
}
```

que es equivalente a utilizar while en la siguiente forma,

```
inicialización;
while (Expresión Booleana) {
 sentencias;
incremento;
}
```

La sentencia o sentencias inicialización se ejecutas al comienzo del for, e incremento después de sentencias. La Expresión Booleana se evalúa al comienzo de cada iteración; el ciclo termina cuando la expresión de comparación toma el valor false. Cualquiera de las tres partes puede estar vacía. La inicialización y el incremento pueden tener varias expresiones separadas por comas.

Ciclo do while

Es similar al ciclo while pero con la particularidad de que el control está al final del ciclo (lo que hace que el ciclo se ejecute al menos una vez, independientemente de que la condición se cumpla o no). Una vez ejecutados los statements, se evalúa la condición: si resulta true se vuelven a ejecutar las sentencias incluidas en el ciclo, mientras que si la condición se evalúa a false finaliza el ciclo.

```
do {
statements
} while (booleanExpression);
```

11. OBJETO

Un objeto en un sistema posee: una identidad, un estado y un comportamiento. El estado marca las condiciones de existencia del objeto dentro del programa.Lógicamente este estado puede cambiar. Un coche puede estar parado, en marcha, estropeado, funcionando, sin gasolina, etc.

El comportamiento determina como responde el objeto ante peticiones de otros objetos. Por ejemplo un objeto conductor puede lanzar el mensaje arrancar a un coche. El comportamiento determina qué es lo que hará el objeto. La identidad determina que cada objeto es único aunque tengan el mismo valor. No existen dos objetos iguales. Lo que sí existe es dos referencias al mismo objeto.

Los objetos se manejan por referencias, existirá una referencia a un objeto. De modo que esa referencia permitirá cambiar los atributos del objeto. Incluso puede haber varias referencias al mismo objeto, de modo que si una referencia cambia el estado del objeto, el resto mostrará esos cambios.

12. CLASES

Las clases son las plantillas para hacer objetos. Una clase sirve para definir una serie de objetos con propiedades (atributos), comportamientos (operaciones o métodos), y semántica comunes. Hay que pensar en una clase como un molde. A través de las clases se obtienen los objetos en sí, Es decir antes de poder utilizar un objeto se debe definir la clase a la que pertenece, esa definición incluye: Sus atributos. Es decir, los datos miembros de esa clase. Los datos pueden ser públicos (accesibles desde otra clase) o privados (sólo accesibles por código de su propia clase). También se las llama campos.