UT1 Identificación de los elementos de un programa informático

RESULTADO DE APRENDIZAJE
1. Reconoce la estructura de un programa informático, identificando y relacionando los elementos propios del lenguaje de programación utilizado.

Tabla de contenido

1. Introducción

En el proceso de resolución de un problema mediante un ordenador, después de plantear el análisis de dicho problema y el diseño del algoritmo para su resolución, es necesario traducirlo a un lenguaje que expresase de forma clara y sin ambigüedades los pasos que se van a seguir para que el ordenador llevase a cabo dicho algoritmo. A este lenguaje es lo que se conoce como lenguaje de programación.

Un lenguaje de programación es un conjunto de caracteres, reglas para la combinación de esos caracteres y las reglas que definen sus efectos cuando los ejecuta un ordenador. Consta de los siguientes elementos:

- Un alfabeto o vocabulario: formado por el conjunto de símbolos permitidos
- Una sintaxis: son las reglas que indican cómo realizar las construcciones con los símbolos del lenguaje
- Una semántica: son las reglas que determinan el significado de cualquier construcción del lenguaje.

Existen distintas clasificaciones de los lenguajes de programación. Una de ellas es según su forma de ejecución. Tendremos

Lenguajes compilados

Un programa que se escribe en un lenguaje de alto nivel tiene que traducirse a un código que pueda utilizar la máquina.

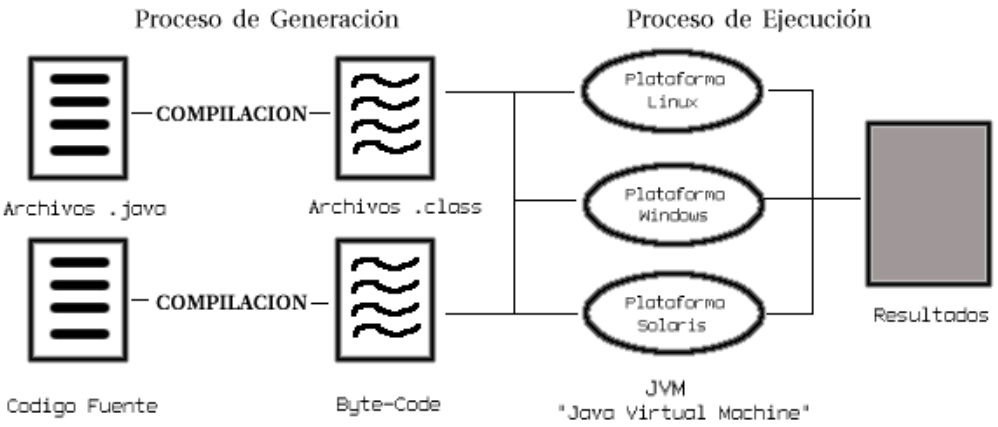
Un compilador es un programa que puede leer un programa escrito en un determinado lenguaje (un lenguaje fuente) y traducirlo en un programa equivalente en otro lenguaje (lenguaje destino). Ejemplos de lenguajes compilados son C o Pascal.

Lenguajes interpretados

En vez de producir un programa destino como resultado del proceso de traducción, el intérprete da la apariencia de ejecutar directamente las operaciones especificadas en el programa fuente con las entradas proporcionadas por el usuario. Cada vez que se ejecuta una instrucción, se debe interpretar y traducir a lenguaje máquina.

Algunos ejemplos de lenguajes interpretados son: PHP, JavaScript, Python, Perl, Logo, Ruby,ASP.

Los procesadores del lenguaje Java combinan la compilación y la interpretación, como se muestra a continuación. Un programa fuente en Java puede compilarse primero en un formato intermedio, llamado bytecode, para después ser interpretado por la máquina virtual.



2. Lenguaje de programación Java

Surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. Los electrodomésticos tenían

Reducida potencia de cálculo

Poca memoria

Se creó un lenguaje **sencillo** capaz de generar **código** de tamaño muy **reducido**

Algunas características de Java son las siguientes:

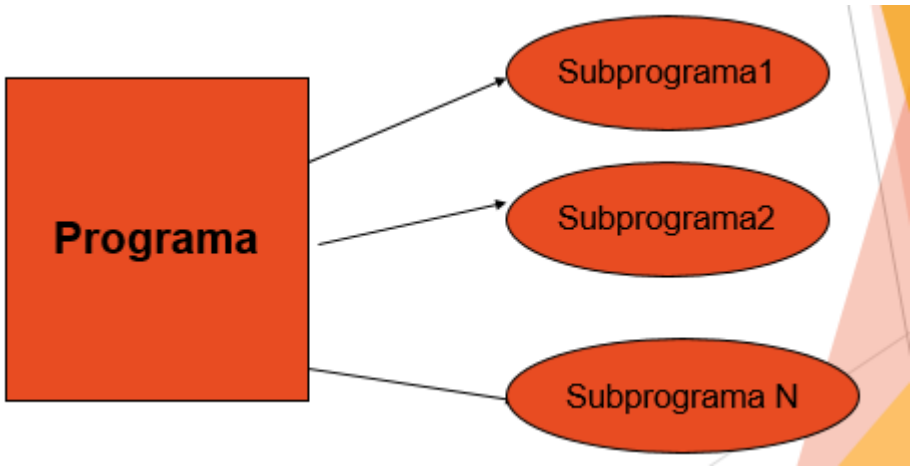
Sun describe el lenguaje como:

- Simple
- Orientado a objetos
- Distribuido
- Robusto
- Seguro
- Arquitectura neutra
- Portable
- Altas prestaciones
- Multitarea
- Dinámico

3. Programación Orientada a Objetos

Programación Estructurada

En los años 60 la programación se hacía de modo clásico. Existía un programa principal y ese programa hacía llamadas a otros subprogramas para resolver subproblemas concretos.



El diseño descendente o top-down consiste en una serie de descomposiciones sucesivas del problema inicial. La utilización de esta técnica tiene los siguientes objetivos:

- Simplificación del problema y de los subprogramas resultantes de cada descomposición.
- Las diferentes partes del programa pueden ser programadas de modo independiente e incluso por diferentes personas
- El programa final queda estructurado en forma de bloques o módulos, lo que hace más sencilla su lectura y mantenimiento.

Cuando los programas se fueron haciendo más y más grandes aparecieron problemas como:

- Integrar el trabajo de todos los desarrolladores
- Conseguir que el resultado final sea eficiente
- Reutilización de código

Todo programa utiliza unos datos de entrada y produce unos resultados. Para esta labor se usan las variables de enlace o parámetros. Cada vez que se realiza una llamada a un subprograma, los datos de entrada son pasados por medio de determinadas variables y, análogamente, cuando termina la ejecución del subprograma, los resultado regresan mediante otras o las mismas variables.

Los parámetros pueden ser de dos tipos:

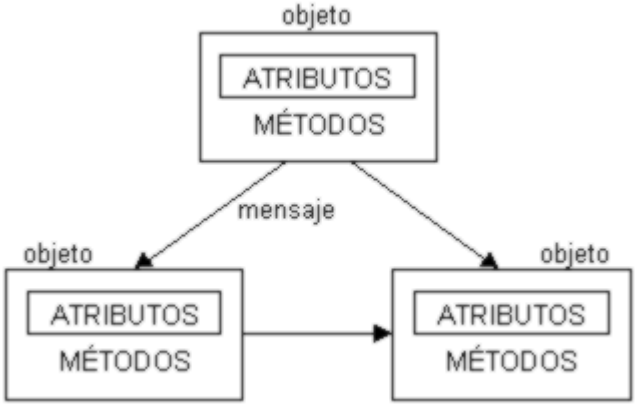
- Formales: Variables locales de un subprograma utilizadas para la recepción y el envío de los datos

- Actuales: Variables y datos enviados, en cada llamada de subprograma, por el programa o subprograma llamante.

Programación Orientada a Objetos

En los años 80 surgió la POO (Programación Orientada a Objetos) para dar una solución a estos problemas. Se define como un **paradigma de programación** que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Un paradigma es un conjunto de reglas, patrones o estilos de programación.

Un programa Orientado a Objetos es un conjunto de objetos, independientes entre sí, que dialogan entre ellos pasándose mensajes para llegar a resolver el problema en cuestión.



Hoy en día no se entiende la programación de apps para móviles o el desarrollo web sin el uso de un lenguaje POO.

Los elementos básicos de la programación orientada a objetos son los siguientes:

- Atributos o propiedades: en POO cada objeto dispone de una serie de atributos que definen sus características individuales y le permiten diferenciarse de otros (apariencia, estado, etc).
- Método: es una subrutina que puede pertenecer a una clase u objeto, y son una serie de sentencias para llevar a cabo una acción.
- Clase: las clases son un pilar fundamental de la POO y representan un conjunto de variables y métodos para operar con datos. Es una agrupación de:
    - Datos → Variables
    - Funciones → Métodos

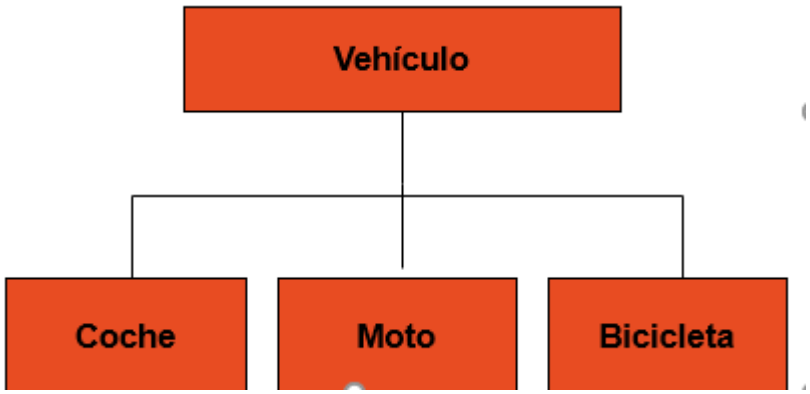
  Por ejemplo, se puede definir la clase **Coche**
    - Variables→ Numero de puertas, color, matrícula, marca, modelo
    - Métodos → Comprobar_velocidad()

- Objeto: Es una instancia de la clase, es decir, una vez creada una clase podemos declarar elementos de esa clase. Se pueden declarar dos objetos de la clase **Coche**

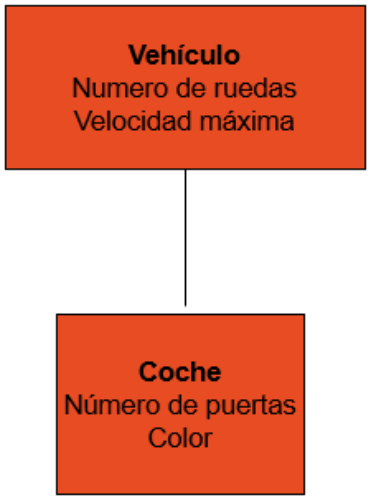
  C1→ 5 puertas, azul, 1234TTT,Opel, Corsa

  C2 → 3 puertas, rojo,5678QQQ, Seat, Ibiza

- Herencia: la herencia facilita la creación de objetos a partir de otros ya existentes o hace que una subclase obtenga el comportamiento de su clase principal o superclase.



Por ejemplo, un coche tendrá las siguientes características: Número de ruedas, velocidad máxima, número de puertas y color.



- Polimorfismo

4. Programación en Java

Cualquier programa básico Java estará compuesto de un fichero `.java`, que contendrá dentro una clase (con el mismo nombre que el fichero) y un método main:

```
public class HolaMundo {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hola Mundo");
    }

}
```

Comentarios

Nos van a permitir documentar nuestro código, o realizar ciertas aclaraciones sobre el mismo. Existen dos formas de hacerlo:

Para varias líneas: delimitado por `/* .... */`

Para una sola línea: delimitado por `// ...`

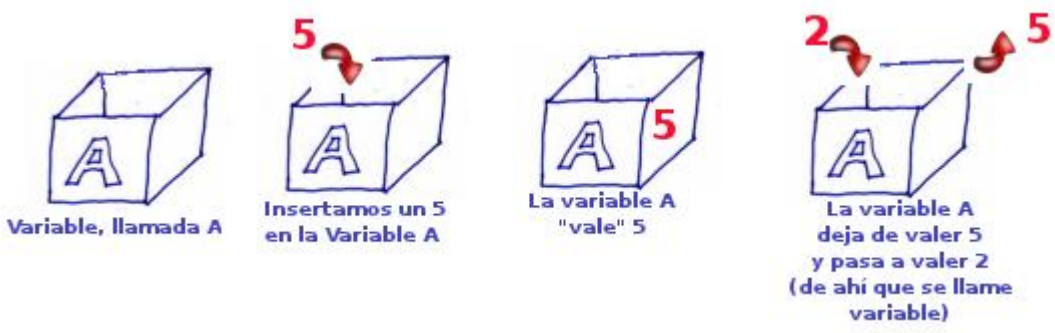
Variables

Es un nombre que contiene un valor que puede cambiar a lo largo del programa. Según su papel en el programa tenemos:

Variables miembro de la clase: Se definen en la clase fuera de cualquier método

Variables locales: Se definen dentro de un método o de un bloque { }

Los nombres de variables pueden contener cualquier conjunto de caracteres numéricos y alfabéticos que no sean palabras reservadas. Se utiliza la notación húngara. Los nombres de las clases son en mayúscula y los de los métodos en minúscula. Si está formado por una palabra compuesta se ponen en mayúscula las letras que correspondan con el inicio de las palabras.

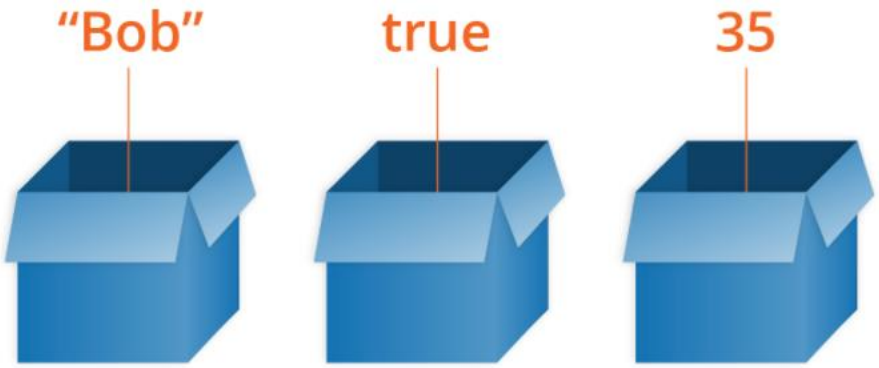


Tipos primitivos de variables:

- Boolean: No es un valor numérico, solo admite los valores true o false.
- Char: Usa el código UNICODE y ocupa cada carácter 16 bits.
- Enteros: Difieren en las precisiones y pueden ser positivos o negativos.

- Byte: 1 byte.
- Short: 2 bytes.
- Int: 4 bytes.
- Long: 8 bytes.

- Reales en punto flotante: igual que los enteros también difieren en las precisiones y pueden ser positivos o negativos.
  - Float: 4 bytes.
  - Double: 8 bytes.



Los tipos de datos compuestos están formados por tipos de datos simples. Se verán con mayor detalle en las siguientes unidades.

En la siguiente tabla se muestran los tipos de dato primitivos de Java con el intervalo de representación de valores que puede tomar y el tamaño en memoria correspondiente.

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
`boolean`	`true` o `false`	1	N.A.	N.A.	`false`
`char`	Carácter Unicode	16	\u0000	\uFFFF	\u0000
`byte`	Entero con signo	8	-128	128	0
`short`	Entero con signo	16	-32768	32767	0
`int`	Entero con signo	32	-2147483648	2147483647	0
`long`	Entero con signo	64	-9223372036854775808	9223372036854775807	0
`float`	Coma flotante de precisión simple Norma IEEE 754	32	±3.40282347E+38	±1.40239846E-45	0.0
`double`	Coma flotante de precisión doble Norma IEEE 754	64	±1.7976931348623157OE+308	±4.94065645841246544E-324	0.0

Para definir e inicializar una variable:

Tipo nombre_variable

P.e.
```
int x; //Declaración de la variable x de tipo entero
int y=5; //Declaración de x e inicialización a 5
```

Operadores

- Aritméticos: +,-, *, /, %

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 - 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2 7 / 2	0.25 3
%	Resto de la división entera	20 % 7	6

- Asignación: =

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
=	Operador asignación	n = 4	n vale 4

- Aritméticos combinados:+=, -=, *=, /*,%=

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
*=	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

- Unarios: + y – sirven para cambiar el signo a una expresión
- Incrementales: ++ y --

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

++	Incremento
	i++ primero se utiliza la variable y luego se incrementa su valor
	++i primero se incrementa el valor de la variable y luego se utiliza
		4++
		a=5;
		b=a++;
		a=5;
		b=++a;
			5
			
			a vale 6 y b vale 5
			
			a vale 6 y b vale 6

--	decremento	4--	3

- Relacionales: <,>,>=,<=,== y !=

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	`7 == 38`	`false`
!=	distinto que	`'a' != 'k'`	`true`
<	menor que	`'G' < 'B'`	`false`
>	mayor que	`'b' > 'a'`	`true`
<=	menor o igual que	`7.5 <= 7.38`	`false`
>=	mayor o igual que	`38 >= 7`	`true`

- Lógicos: &&, ||, !

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
!	Negación - NOT (unario)	`!false` `!(5==5)`	`true` `false`
|	Suma lógica – OR (binario)	`true | false` `(5==5)|(5<4)`	`true` `true`
^	Suma lógica exclusiva – XOR (binario)	`true ^ false` `(5==5)|(5<4)`	`true` `true`
&	Producto lógico – AND (binario)	`true & false` `(5==5)&(5<4)`	`false` `false`
||	Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true`	`true || false` `(5==5)||(5<4)`	`true` `true`
&&	Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false`	`false && true` `(5==5)&&(5<4)`	`false` `false`

- Concatenación de cadenas: +

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+	Operador concatenación	`"Hola" + "Juan"`	`"HolaJuan"`

- Nivel de bits:>>,<<,…

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
~	Negación ó complemento binario (unario)	`~12`	`-13`
|	Suma lógica binaria – OR (binario)	`12|10`	`8`
^	Suma lógica exclusiva – XOR (binario)	`12^10`	`6`
&	Producto lógico binario – AND (binario)	`12&10`	`14`
<<	Desplaza a la izquierda los bits del 1º operando tantas veces como indica el 2º operando (por la derecha siempre entra un cero)	`7<<2` `-7<<2`	`28` `-28`
>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando (por la izquierda entra siempre el mismo bit más significativo anterior)	`7>>2` `-7>>2`	`1` `-2`
>>>	Desplaza a la derecha los bits del 1º operando tantas veces como indica el 2º operando – sin signo (por la izquierda entra siempre un cero).	`7>>>2` `-7>>>2`	`1` `1073741822`

Separadores

Existen algunos caracteres que tienen un significado especial en el lenguaje Java. En la siguiente tabla se resumen los diferentes separadores que pueden encontrarse en el código fuente de un programa.

Separador	Descripción
`()`	Permiten modificar la **prioridad de una expresión**, **contener expresiones** para el control de flujo y realizar **conversiones de tipo**. Por otro lado pueden contener la **lista de parámetros** o argumentos, tanto en la definición de un método como en la llamada al mismo.
`{}`	Permiten definir **bloques de código** y ámbitos y contener los valores **iniciales** de las variables `array`
`[]`	Permiten declarar **variables de tipo `array`** (vectores o matrices) y referenciar sus elementos
`;`	Permite **separar sentencias**
`,`	Permite **separar identificadores** consecutivos en la declaración de variables y en las listas de parámetros. También se emplea para encadenar sentencias dentro de un **bucle `for`**
`.`	Permite **separar el nombre de un atributo o método** de su instancia de referencia. También separa el identificador de un **paquete** de los de los subpaquetes y clases

Expresiones

Una expresión es un conjunto de variables, operadores e invocaciones de métodos que se construyen para poder ser evaluadas retornando un resultado. Se evalúa generándose un único resultado de un tipo determinado.

Ejemplos de expresiones son:

```
int valor = 1;
if (valor 1 > valor2) { ... }
```

Cuando tengamos expresiones de evaluación complejas es recomendable que utilicemos paréntesis para saber cuál es el orden de ejecución de operaciones. Ya que si tenemos una expresión como

```
2 + 10 / 5
```

No será la misma si ponemos

```
(2 + 10) / 5
```

O bien

```
2 + (10 / 5)
```

En el caso de no utilizar paréntesis se ejecutará el orden de preferencia de operadores. En este caso la división tiene más preferencia que la suma.

Si dos operadores se encuentran en la misma expresión, el orden en el que se evalúan puede determinar el valor de la expresión. En la siguiente tabla se muestra el orden o prioridad en el que se ejecutan los operadores que se encuentren en la misma sentencia. Los operadores de la misma prioridad se evalúan de izquierda a derecha dentro de la expresión.

Prior.	Operador	Tipo de operador	Operación
1	++ -- +, - ~ ¡	Aritmético Aritmético Aritmético Integral Booleano	Incremento previo o posterior (unario) Incremento previo o posterior (unario) Suma unaria, Resta unaria Cambio de bits (unario) Negación (unario)
2	(tipo)	Cualquiera	
3	*, /, %	Aritmético	Multiplicación, división, resto
4	+, - +	Aritmético Cadena	Suma, resta Concatenación de cadenas
5	<< >> >>>	Integral Integral Integral	Desplazamiento de bits a izquierda Desplazamiento de bits a derecha con inclusión de signo Desplazamiento de bits a derecha con inclusión de cero
6	<, <= >, >= instanceof	Aritmético Aritmético Objeto, tipo	Menor que, Menor o igual que Mayor que, Mayor o igual que Comparación de tipos
7	== ¡= == ¡=	Primitivo Primitivo Objeto Objeto	Igual (valores idénticos) Desigual (valores diferentes) Igual (referencia al mismo objeto) Desigual (referencia a distintos objetos)
8	& &	Integral Booleano	Cambio de bits AND Producto booleano
9	^ ^	Integral Booleano	Cambio de bits XOR Suma exclusiva booleana
10	| |	Integral Booleano	Cambio de bits OR Suma booleana
11	&&	Booleano	AND condicional
12	||	Booleano	OR condicional
13	? :	Booleano, cualquiera, cualquiera	Operador condicional (ternario)
14	= *=, /=, %= +=, -= <<=, >>= >>>= &=, ^=, |=	Variable, cualquiera	Asignación Asignación con operación

Sentencia

Es una orden que indica al programa los pasos que debe dar. Hay que tener cuidado con el uso de mayúsculas y minúsculas, ya que Java es *case sensitive*. Obligatoriamente, todas las sentencias deben terminar con un punto y coma (;). Una sentencia es la unidad mínima de ejecución de un programa. Un programa se compone de conjunto de sentencias que acaban resolviendo un problema. Al final de cada una de las sentencias encontraremos un punto y coma (;).

Tenemos los siguientes tipos de sentencias:

1) Sentencias de declaración

```
int valor = 2;
```

2) Sentencias de asignación

```
valor = 2;
```

3) Sentencias de incremento o decremento

```
valor++;
```

4) Invocaciones a métodos

```
System.out.println("Hola Mundo");
```

5) Creaciones de objetos

```
Circulo miCirculo = new Circulo(2,3);
```

Bloque de código

Se trata de la agrupación de varias sentencias. En las próximas lecciones encontraremos muchos motivos para realizar estas agrupaciones. Siempre vienen delimitadas por una apertura y cierre de llaves { }.

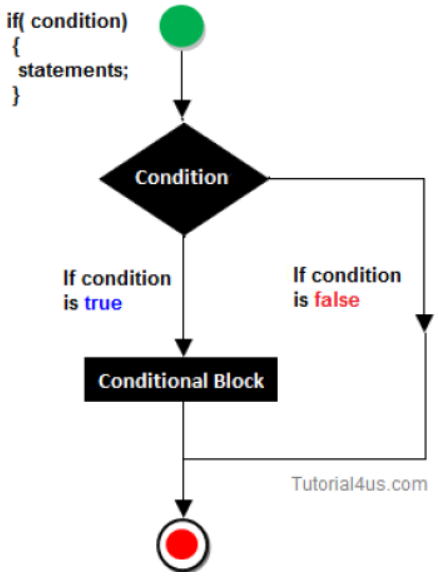
```
if (expresion) {
    // Bloque 1
} else {
    // Bloque 2
}
```

Estructuras de control

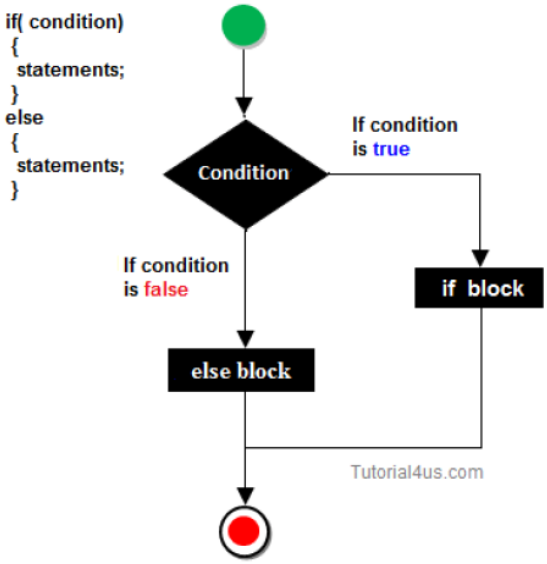
Hay dos tipos:

1) Estructuras de selección:

Simple



Doble



Múltiple

```
if(expresionbooleana){sentencias}
else if(expresionbooleana){sentencias}
else if(expresionbooleana){sentencias}
else if(expresionbooleana){sentencias}
else{ sentencias }
```

2) Estructuras de repetición o bucles

while

Nos permite repetir la ejecución de un bloque de sentencias. La repetición se realiza durante un número indeterminado de veces, mientras una expresión sea cierta. Una de las sentencias del cuerpo del bucle debe modificar alguna de las variables de la condición, para que, en alguna ocasión, la expresión sea falsa.

```
while (condicion) {
  ...
  ...
}
```

do-while

Nos permite repetir la ejecución de un bloque de sentencias. La condición, a diferencia de la estructura while, se evalúa al final del bucle. El cuerpo del bucle se ejecuta siempre, al menos, una vez.

```
do {
  ...
  ...
} while (condicion);
```

for

Se repite un bloque de código un número conocido a priori de veces.

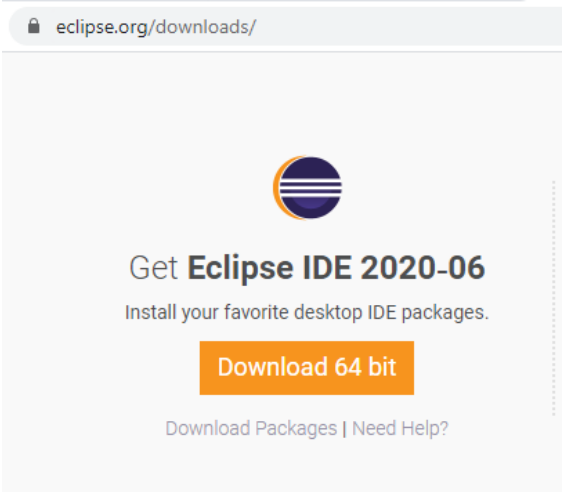
```
for(declaracion; condicion; incremento) {
 ...
  }
```

5. Entorno de Desarrollo: Eclipse

IDE (Integrated Development Environment) es una aplicación informática compuesta por:

- Editor de texto: Código fuente
- Compilador: Traducir código fuente a ejecutable
- Intérprete: Traducción a medida que se va ejecutando la instrucción
- Depurador: Detectar y corregir los errores en el código fuente
- Constructor de interfaz gráfica
- Control de versiones: Gestionar los cambios que se realizan en las aplicaciones

El IDE que se va a utilizar este curso es Eclipse



6. Bibliografía

- OpenWebinars
- Manualweb.net
- https://www.arkaitzgarro.com/