Primeros pasos

- Primeros pasos
- Elementos de un programa
- Separadores
- Operadores
- Comentarios
- Constantes y variables
- Ámbito
- Cadenas
- Arrays
- Control de flujo
- Funciones condicionales
- Bucles
- Funciones
- Estructuras de almacenamiento de datos
- Excepciones
- Convenios de escritura
- Algoritmos y estructuras de resolución de problemas sencillos
- Código utilizado en los ejemplos
- Tarea

Copyright 2026

Primeros pasos

Hasta el momento hemos realizado una introducción a Java indicando como instalarlo y configurarlo. En este segundo módulo se pretende profundizar en el lenguaje Java y empezar a realizar unos primeros algoritmos que nos permitan introducirnos de lleno en la programación orientada a objetos en el capítulo 3.

Copyright 2026

Elementos de un programa

En el módulo 1 ya hemos realizado un programa y hemos visto algunas palabras clave y hablado de clases y paquetes pero es ahora cuando vamos a profundizar en las posibilidades del lenguaje Java

Copyright 2026

Separadores

En Java utilizaremos los siguientes separadores:

- Corchetes []: Se utilizan en operaciones con vectores
- Llaves {}: Sirven para definir bloques de código dentro de las clases y métodos.
- Paréntesis (): Dentro de los paréntesis ubicaremos los parámetros de un método. También nos permiten realizar moldeados de objetos (lo trataremos en el tercer módulo).
- Punto y coma ;: Se utiliza para separar las distintas sentencias de código.
- Punto .: Se utiliza para referirnos a clases y subpaquetes. También nos da acceso a métodos o variables de variables.
- Coma ,: Sirve para separar parámetros dentro de un método. También nos permite separar variables en su declaración.

Con anterioridad ya hemos utilizado todos ellos y conforme avancemos en el curso vamos a ir viendo mas ejemplos de su uso. Vamos a ver un pequeño recordatorio de donde los hemos utilizado:

- Cuando definimos una variable llamada args que era un vector de Strings utilizamos los corchetes: *String**[]**__ args*.
- Cuando definimos la clase Modulo1HolaMundo ya utilizamos las llaves para definir el alcance de la clase: *public class Modulo1HolaMundo **{** (...) **}***.
- Creando creamos el método main hicimos uso para los paréntesis para indicar el comienzo y fin de los parámetros: *public static void main**(**String[] args**)** {*.
- Al finalizar la sentencia *System.out.println("Hola mundo")**;*** hicimos uso del punto y coma para delimitar el fin de la misma.
- En la misma sentencia que en el caso anterior *System**.**out**.**println("Hola mundo");* hicimos uso del punto para acceder a una clase y sus objetos. En este apartado entraremos en mayor profundidad mas adelante.
- El caracter coma aún no lo hemos utilizado, pero podríamos darle uso en el siguiente ejemplo: *void nombreFuncion(int param1**,** int param2);* Aquí la función de la coma no es otra mas que separar los parámetros param1 y para2 del método nombreFuncion.

Pregunta Verdadero-Falso

Copyright 2026

Para delimitar el contenido de una clase haré uso de los corchetes []

VerdaderoFalsoPara delimitar el contenido de una clase se hace uso de las llaves {} Ocurre lo mismo con los métodos.

En Java al final de cada línea debo colocar un punto y coma.

VerdaderoFalsoEl punto y coma solo debe colocarse al final de las sentencias. Cuando declaramos una clase o método no hay que hacerlo.

Para crear un vector haré uso de los corchetes []

VerdaderoFalsoPara definir un vector se usan los corchetes []

Copyright 2026

Operadores

Se hace necesario poder realizar cálculos o tomar decisiones con los datos que manejamos, es por ello que nos surge la necesidad de disponer de operadores que nos permitan hacerlo. A continuación procedo a enumerar los operadores mas habituales:

- = Operando de asignación. Nos permite asignar un valor a una variable.
- + Nos permite sumar números y concatenar cadenas.
- - Nos permite restar números.
- * Nos permite multiplicar números.
- / Nos permite dividir números.
- % Nos permite obtener el resto de una división.
- == Operando de comparación. Devuelve verdadero en caso de que las variables involucradas sean iguales.
- != Operando de comparación. Devuelve verdadero en caso de que las variables involucradas sean distintas.
- < Operando de comparación. Devuelve verdadero en caso de que la variable de la izquierda sea menor a la de la derecha y devuelve falso en cualquier otra situación.
- > Operando de comparación. Devuelve verdadero en caso de que la variable de la derecha sea menor a la de la izquierda y devuelve falso en cualquier otra situación.
- <=Operando de comparación. Devuelve verdadero en caso de que la variable de la izquierda sea menor o igual a la de la derecha y devuelve falso en cualquier otra situación.
- >=Operando de comparación. Devuelve verdadero en caso de que la variable de la derecha sea menor o igual a la de la izquierda y devuelve falso en cualquier otra situación.
- &&: Equivale al operador lógico AND. Devuelve true en caso de que las 2 variables involucradas valgan true, en otro caso devuelven false.
- ||: Equivale al operador lógico OR. Devuelve true en caso de que al menos 1 de las 2 variables involucradas valga true, en otro caso devuelve false.
- variable++ Evalúa el valor de la variable en cuestión y posteriormente le suma 1
- variable-- Evalúa el valor de la variable en cuestión y posteriormente le resta 1
- ++variable Suma 1 a la variable y posteriormente evalúa su valor
- --variable Resta 1 a la variable y posteriormente evalúa su valor

En Java existen mas operadores que los aquí indicados pero creo que con los aquí señalados es mas que suficiente para cubrir los objetivos de este curso. A lo largo del curso se procurará utilizar todos los operadores aquí indicados para así aclarar cualquier posible duda que pudiera existir con

Copyright 2026

ellos.

Copyright 2026

Comentarios

Los comentarios los introduciremos en nuestro código fuente para facilitar la lectura y comprensión posterior del código. La redacción de estos comentarios también van a facilitar en la posterior elaboración de la documentación del código.

¿Cómo se indica en un programa que algo es un comentario? Existen distintos modos de hacerlo:

- de línea: Todo lo que, en una línea, esté detrás del símbolo // se considerará un comentario y ni el compilador ni el intérprete lo tendrán en cuenta.
- de bloque: Todo lo que esté comprendido entre el símbolo / *y el símbolo* / se considerá un comentario y ni el compilador ni el intérprete lo tendrán en cuenta.
- de documentación: Todo lo que esté comprendido entre el símbolo /* *y el símbolo* / se considerará un comentario de documentación y en la generación de documentación y en la autoayuda del IDE netbeans nos proporcionará información extra.

Vamos a ver un extracto de código que contiene comentarios:

```
230     /**
231      * Método para activar usuarios
232      * @param usuarioEnSession usuario que realiza la acción
233      * @param pass contraseña
234      * @param passRepetido contraseña repetida
235      * @param mail email
236      * @param mailRepetido email repetido
237      * @param lenguajePreferido  identificador del lenguaje elegido por el usuario
238      * @param idAvatar identificador del avatar elegido por el usuario
239      * @param borrable S o N indica respectivamente si el usuario se puede borrar o no
240      * @return devuelve true en caso de actualizar el usuario y falso en caso contrario
241      * @throws TrivinetException
242      */
243     public boolean actualizarUsuario(Usuario usuarioEnSession, String pass,
244             String passRepetido, String mail, String mailRepetido,String lenguajePreferido,
245             Integer idAvatar, String borrable) throws TrivinetException {
246         /* LOG */
247         Util.log("actualizarUsuario("+usuarioEnSession+", pass: ***, passRepetido: ***, " +
248                 ""+mail+", "+mailRepetido+", "+
249                 lenguajePreferido+", " +
250                 "idAvatar: "+idAvatar+", borrable: "+borrable+")");
251         // Comprobaciones
252
253         // Variables
```

Entre las líneas 230 y 242 tenemos un comentario de documentación. En este tipo de comentarios, además de explicar para que sirve una función se puede detallar información adicional a través de

anotaciones como @param @return o @throws que sirven respectivamente para indicar los parámetros de la función, lo que devuelve la función y el tipo de excepción o excepciones que lanza la función.

En la línea 246 tenemos un ejemplo de comentario de bloque.

En las líneas 251 y 253 tenemos ejemplos de comentarios de línea.

Es muy importante concienciarnos a nosotros/as y a nuestro alumnado de la importancia de documentar el código para que quede reflejada la funcionalidad del código escrito y así facilitar su posterior mantenimiento.

Copyright 2026

Constantes y variables

Una **[variable](https://es.wikipedia.org/wiki/Variable_(programaci%C3%B3n)** es un lugar en memoria donde almacenaremos un dato que podrá cambiar o no a lo largo de la ejecución de nuestros programas. Todas las variables tienen que ser de un determinado tipo y tener un nombre. Por convenio las variables tienen al menos el primer caracter de su nombre en minúsculas. Una **[constante](https://es.wikipedia.org/wiki/Constante_(inform%C3%A1tica)** es un lugar en memoria en el cual almacenaremos un dato con el fin único de leerlo y sin posibilidad de modificación. Las constante deben tener un determinado tipo y deben tener un nombre. Por convenio el nombre de las constantes se escribe en mayúsculas. En java para definir una constante lo haremos exactamente igual que si definieramos una variable pero las constantes deben ser precedidas por la palabra reservada final.

Anteriormente hemos indicado que tanto las constantes como las variables deben ser de un determinado tipo, a continuación vamos a hablar de ellos.

En java **existen 2 tipos de variables: los tipos básicos** (también llamados tipos primitivos) **y los objetos**. De los objetos nos ocuparemos en el módulo 3 por lo que a continuación paso a enumerar los tipos primitivos mas habituales.

- Números enteros:
  - byte: utiliza 8 bits en memoria. Su rango va de -27 a 27-1
  - short: utiliza 16 bits en memoria. Su rango va de -215 a 215-1
  - int: utiliza 32 bits en memoria. Su rango va de -231 a 231-1
  - long: utiliza 64 bits en memoria. Su rango va de -263 a 263-1
- Números decimales:
  - float: utiliza 32 bits.
  - double: utiliza 64 bits.
- Booleanos:
  - boolean: utiliza 1 bit. Solo puede valer true (verdadero) o false (falso)
- Cadenas:
  - char: utiliza 16 bits.

Ejemplos con tipos primitivos:

```
int operando1 = 4; // Declaramos una variable llamada operando1 de tipo int y le asignamos el valor 4
```

```
final double PI = 3.1416; // Declaramos una constante de tipo double. El nombre de la constante es PI y le asignamos el valor 3,1416

boolean esMayor = false;//Declaramos una varible de tipo boolean, con nombre esMayor y le asignamos el valor falso

final char CARACTER = 'a';// Declaramos una costante de tipo char y nombre CARACTER. Le asignamos el valor 'a'. Fíjate en que son comillas simples.
```

Aunque lo veremos en el siguiente módulo del curso a continuación vamos a crear unas variables que harán referencias a objetos. Del código que aparece a continuación me interesa que nos fijemos en que **declaremos tipos primitivos u objetos la sintaxis es prácticamente la misma**:

```
Integer operando1  = new Integer(4); // Declaramos una variable llamada operando1 de tipo Integer y le asignamos el valor 4

final Double PI = new Double(3.1416); // Declaramos una constante de tipo Double. El nombre de la constante es PI y le asignamos el valor 3,1416

Boolean esMayor = new Boolean(false); // Declaramos una variable de tipo Boolean, con nombre esMayor y le asignamos el valor falso.

final String CADENA = "a";// Declaramos una constante de tipo String y nombre CADENA y le asignamos el valor "a". Fíjate en que son comillas dobles. Además aquí es el único lugar donde no hemos escrito new String("a") pero de esto ya hablaremos mas adelante aunque no es trivial
```

Como podemos intuir en base a estos ejemplos **cada tipo básico tiene asociado un objeto equivalente**. Pero como vemos a continuación (y profundizaremos en el módulo 3) también podemos crear variables del tipo de Objetos que hayamos creado.

```
Coche miCoche = new Coche(); //Declaramos una variable de tipo Coche con el nombre miCoche y llamamos al constructor de la clase. Mas adelante hablaremos de esto

Moto moto1 = null; //Declaramos una variable de tipo Moto con el nombre moto1 y le asignamos el valor null
```

Copyright 2026

Si eres observador/a te habrás dado cuenta de que los tipos básicos empiezan en minúscula mientras que los objetos comienzan en mayúsculas. Es probable que también te hayas percatado de que el nombre que asigno a las variables siempre comienza en minúsculas y el nombre que asigno a las constantes está completamente en mayúsculas, esto son convenios de escritura del lenguaje Java y tu puedes seguir tu propio estilo de código pero es interesante hacer uso del convenio porque facilita la lectura y comprensión del código yen caso de trabajar con otras personas se es mas productivo. En este módulo, un poco mas adelante, hablaremos en mayor profundidad de esta cuestión.

Copyright 2026

Ámbito

Las variables y constantes tienen asociado un ámbito. Simplificando, el ámbito indica donde existen y por tanto pueden utilizarse. A continuación voy a tratar de explicarlo con un extracto de código.

```
1   package modulo2ambito;
2   /**
3    * @author Pablo Ruiz Soria
4    */
5   public class Modulo2Ambito {
6       int variableClase = 2;
7       public static void main(String[] args) {
8           int variableMetodo = 2;
9           if(true){
10              int variableCondicion = 4;
11          }
12          for(int i = 0;i < 5; i++){
13              int variableBucle = 5;
14              for(int j = 5; j > 0; j--){
                    int variableOtroBucle = 6;
16              }
17          }
18      }
19  }
```

En el código que tenemos en la imagen anterior en la línea 6 declaramos la variable variableClase cuyo ámito es toda la clase, es decir, desde cualquier punto de la clase podríamos acceder a ella. En la línea 8 declaramos la variable variableMetodo, el ámbito de esta variable es el método main, es decir, puede ser accedida desde dentro del main pero no en otros métodos de la clase. En la línea 10 declaramos otra variable llamada variableCondicion cuyo ámbito es el bucle if, no es accesible desde ningún otro lugar. En la línea 12 declaramos la variable de nombre i que es accesible dentro de todo el bucle. En la línea 13 declaramos la variable variableBucle con la que ocurre lo mismo que con i. En la línea 14 declaramos la variable j cuyo ámbito es el bucle anidado y es únicamente accesible desde este bucle pero no del otro. Lo mismo ocurre en la línea 15 con la variable variableOtroBucle.

Pregunta Verdadero-Falso

En el ejemplo de código anterior, podría sustituir la línea 10 por System.out.println(variableClase); ya que la variable variableClase es accesible desde la línea 10

VerdaderoFalsovariableClase es accesible desde toda la clase Modulo2Ambito y la línea 10 está dentro de la clase por lo que es accesible.

En el ejemplo de código anterior, podría añadir al final de la línea 6 System.out.println(variableBucle); ya que la variable variableBucle es accesible desde la línea 6

VerdaderoFalsovariableBucle solo está accesible dentro del bucle que está entre las líneas 12 y 17 por lo que no es accesible en la línea 6 y obtendríamos un error de compilación.

Copyright 2026

Cadenas

Hasta el momento hemos trabajado con cadenas (**String**) en alguna ocasión. Hemos mostrado por pantalla una cadena:

```
System.out.println("Hello world!");
```

Hemos definido una constante

```
final String CADENA = "a";// Declaramos una constante de tipo String y nombre CADENA y le asignamos el valor "a". Fíjate en que son comillas dobles.
```

y del mismo modo podríamos haber creado una variable:

```
String texto = "Hola mundo";// Declaramos una variablede tipo String y nombre textoy le asignamos el valor "Hola mundo". Fíjate en que son comillas dobles.
```

String no es un tipo primitivo, String es una clase por lo que cada vez que hacemos uso de ella estamos creando un objeto.

Como ya vimos en este módulo en el apartado de constantes y variables y como veremos en el siguiente módulo de este curso que cada vez que creamos un objeto lo hacemos con la sintaxis *Clase nombreVariable = new Clase();*. Sin embargo según acabo de indicar si utilizamos String estamos creando un objeto ¿entonces porque aquí no usamos new y hacemos la asignación directamente? Vamos a ver un par de ejemplos:

Ejemplo 1, sin new:

Copyright 2026

Source History

```
1  package modulo2strings;
2  /**
3   * @author pablo
4   */
5  public class Modulo2Strings {
6      public static void main(String[] args) {
7          String cadena1 = "Hola";
8          String cadena2 = "Hola";
9          String cadena3 = "Adios";
           if(cadena1 == cadena2){
11             System.out.println("cadena1 y cadena2 son iguales");
12         }else{
13             System.out.println("cadena1 y cadena2 son distintas");
14         }
           if(cadena1 == cadena3){
16             System.out.println("cadena1 y cadena3 son iguales");
17         }else{
18             System.out.println("cadena1 y cadena3 son distintas");
19         }
20     }
21 }
22
```

modulo2strings.Modulo2Strings > main >

Output - Modulo2Strings (run)

```
run:
cadena1 y cadena2 son iguales
cadena1 y cadena3 son distintas
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ejemplo2, con new:

```java
package modulo2strings;
/**
 * @author pablo
 */
public class Modulo2Strings {
    public static void main(String[] args) {
        String cadena1 = new String("Hola");
        String cadena2 = new String("Hola");
        String cadena3 = new String("Adios");
        if(cadena1 == cadena2){
            System.out.println("cadena1 y cadena2 son iguales");
        }else{
            System.out.println("cadena1 y cadena2 son distintas");
        }
        if(cadena1 == cadena3){
            System.out.println("cadena1 y cadena3 son iguales");
        }else{
            System.out.println("cadena1 y cadena3 son distintas");
        }
    }
}
```

modulo2strings.Modulo2Strings > main > cadena3 >

Output - Modulo2Strings (run)

```
run:
cadena1 y cadena2 son distintas
cadena1 y cadena3 son distintas
BUILD SUCCESSFUL (total time: 0 seconds)
```

En el primer ejemplo, sin new, lo que ocurre es que cuando creamos la variable cadena1 se crea un espacio en memoria con el valor "Hola" y cuando creamos una nueva variable llamada cadena2 y decimos que valga "Hola" en vez de crearse un nuevo espacio en memoria se apunta al mismo, por eso, cuando se hace la comparación con == se comparan las zonas de memoria y al ser iguales nos dice que cadena1 es igual a cadena2. Sin embargo, en el 2º caso, al crear las variables cadena1 y cadena2 utilizando new estamos forzando a que cada variable (objeto) ocupe una zona de memoria distinta aunque tengan el mismo valor. Puede resultar confuso pero creo conveniente reseñar este caso especial ya que al tratarse de un objeto es mas que probable que nuestro alumnado nos pregunte por esta peculiaridad de este objeto. Esto no ocurre con ningún otro objeto.

En un entorno profesional no se usaría la clase String sino que se usarían las clases StringBuffer o StringBuilder (en función de las necesidades específicas). En el curso utilizaremos la clase String pero por curiosidad vamos a ver un ejemplo de código con estas clases:

Copyright 2026

```java
package modulo2strings;
/**
 * @author pablo
 */
public class Modulo2Strings {
    public static void main(String[] args) {
        StringBuffer cadena1 = new StringBuffer();
        cadena1.append("¡Hola ");
        cadena1.append("mundo!");
        StringBuilder cadena2 = new StringBuilder();
        cadena2.append("Hello ");
        cadena2.append("world!");
        System.out.println(cadena1);
        System.out.println(cadena2);
    }
}
```

modulo2strings.Modulo2Strings > main >

Output - Modulo2Strings (run)

```
run:
¡Hola mundo!
Hello world!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Copyright 2026

Arrays

En Java existen un tipo de variables llamadas * ***que nos permiten crear*** ***vectores** de un mismo elemento. Es importante indicar que un Array es un objeto.

Existen distintos modos de declarar un Array, vamos a verlos:

```
int[] array1 = new int[4]; // Creamos una variable llamada array1 que tendrá 4 posiciones
int array2[] = new int[4]; // Creamos una variable llamada array2 que tendrá 4 posiciones
int[] array3 = {1, 2, 3}; // Creamos una variable llamada array3 que en su posición 0 tendrá
un 1, en su posición 1 tendrá un 2 y en la posición 2 tendrá un 3
```

Cuando creamos un Array como es el caso de array1 y array2 y lo inicializamos pero no definimos el valor de sus posiciones estas toman el valor por defecto de la variable, en nuestro caso 0.

La primera posición de un Array es la posición 0. En caso de intentar acceder a una posición inexistente obtendremos una excepción (se tratan mas adelante)

De un modo análogo al anterior podemos crear de Arrayx (matrices). A continuación podemos ver un ejemplo:

```
int[][] matriz = new int[10][10];// Creamos una variable llamada matriz que es una matriz de
enteros de 10x10
```

Si queremos acceder al valor de un array debemos hacerlo con la sintaxis nombreVariable[posicion]. A continuación unos ejemplos:

```
boolean[] array4 = {false, true, false};
System.out.println(array4[0]);
System.out.println(array4[1]);
System.out.println(array4[2]);
```

Y lo que veríamos por pantalla:

```
false
true
```

Copyright 2026

```
false
```

Cuando lleguemos al apartado de control de flujo realizaremos mas ejemplos con .

Copyright 2026

Control de flujo

Además de poder operar con los datos tenemos la necesidad de que ocurran cosas distintas en función de los mismos. Para cubrir estas necesidades dispondremos de funciones condiciones y bucles.

Como funciones condicionales trabajaremos con:

- if/else if/else
- switch

Como bucles trabajaremos con:

- for
- while
- do while

En los nodo inferiores voy a tratar de explicarlos y veremos algunos ejemplos con código.

Copyright 2026

Funciones condicionales

Existen 2 funciones condicionales: **if/else if/else** y **switch**.

Vamos a comenzar por la estructura **if/else if/else**. En esta estructura la sintaxis es la siguiente:

```
if(condicion){
    sentencias;
}else if(condicion){
    sentencias;
}else{
    sentencias;
}
```

De la estructura anterior es obligatorio que exista siempre el if. else if puede haber 0 o varios. else puede haber 1 o 0. Y el funcionamiento es el siguiente. Cuando el compilador llega a la condición del if y la evalúa, si la condición es verdadera ejecutará las sentencias ubicadas dentro del if y habrá terminado con la estructura if/else if/else, ya no ejecutará sentencias de otros bloques aunque estas sean verdad. Si la condición del if es falsa se evalúa la condición del else if y si es verdadera se ejecutan las sentencias ubicadas dentro, en caso de ser falsa se evalúa el siguiente else if si lo hubiera. Si la condición de algún else if fuese verdadera se ejecutarían las sentencias de su interior y ya no se evaluarían ni ejecutarían mas condiciones de toda la estructura if/else if/else. En caso de que ninguna condición del if o los else if fuese verdadera y hubiese un bloque else se ejecutarían las sentencias contenidas dentro del bloque else.

Vamos a ver a continuación un ejemplo de 2 estructuras if/else if/else ya que es mas sencillo comprender esta estructura leyendo el código que tratando de explicarlo.

Copyright 2026

```java
/**
 * @author Pablo Ruiz Soria
 */
public class Principal {
    public static void main(String[] args) {
        int calificacion = 7;
        boolean avisarFamilia = false;
        if(calificacion >= 0 && calificacion < 5){
            System.out.println("Insuficiente");
            avisarFamilia = true;
        }else if(calificacion < 7){
            System.out.println("Suficiente");
        }else if(calificacion < 9){
            System.out.println("Notable");
        }else if(calificacion < 10){
            System.out.println("Sobresaliente");
        }else if(calificacion == 10){
            System.out.println("Matrícula");
        }else{
            System.out.println("Calificacion no valida");
        }
        if(avisarFamilia){
            System.out.println("Avisar a la familia");
        }
    }
}
```

Principal > main > if (avisarFamilia) >

```
Output - Modulo2Condicionales (run)
run:
Notable
```

Cuando el programa llega al if de la línea 8 se comprueba si la variable calificacion es mayor o igual a 0 (lo es) y que la variable calificacion sea menor a 5 (no lo es). Como el resultado de true && false es false no entramos dentro de sus llaves y se pasa a evaluar la condición de la línea 11. En la línea 11 se evalúa si la variable calificacion es menor a 7 (no lo es), al ser falsa se pasa a evaluar la condición de la línea 13. En la línea 13 se evalúa si la variable calificacion es menor a 9 (lo es), al ser verdad se ejecutan todas las sentencias ubicadas dentro de sus llaves, en ese caso solo hay 1 sentencia y se escribe Notable en consola. Como la condición de la línea 13 es verdad ya no se comprueban las condiciones de las líneas 15, 17 o 19 y se salta directamente a la línea 22.

En la línea 22 tenemos otra condición así que se evalúa si la variable avisarFamilia es true (no lo es), al ser falso no se ejecuta el código de su interior y al no haber ningún else if ni else se da por terminado este if.

Copyright 2026

Existe también la posibilidad de ejecutar esta estructura con la sintaxis:

```
condicion?valorSiCondicionVerdadera:valorSiCondicionFalsa;
```

Vamos a ver un ejemplo que sustituiría al código comprendido entre las líneas 22 a 24 de la imagen anterior

```
22          System.out.println(avisarFamilia?"Avisar a la familia":"");
```

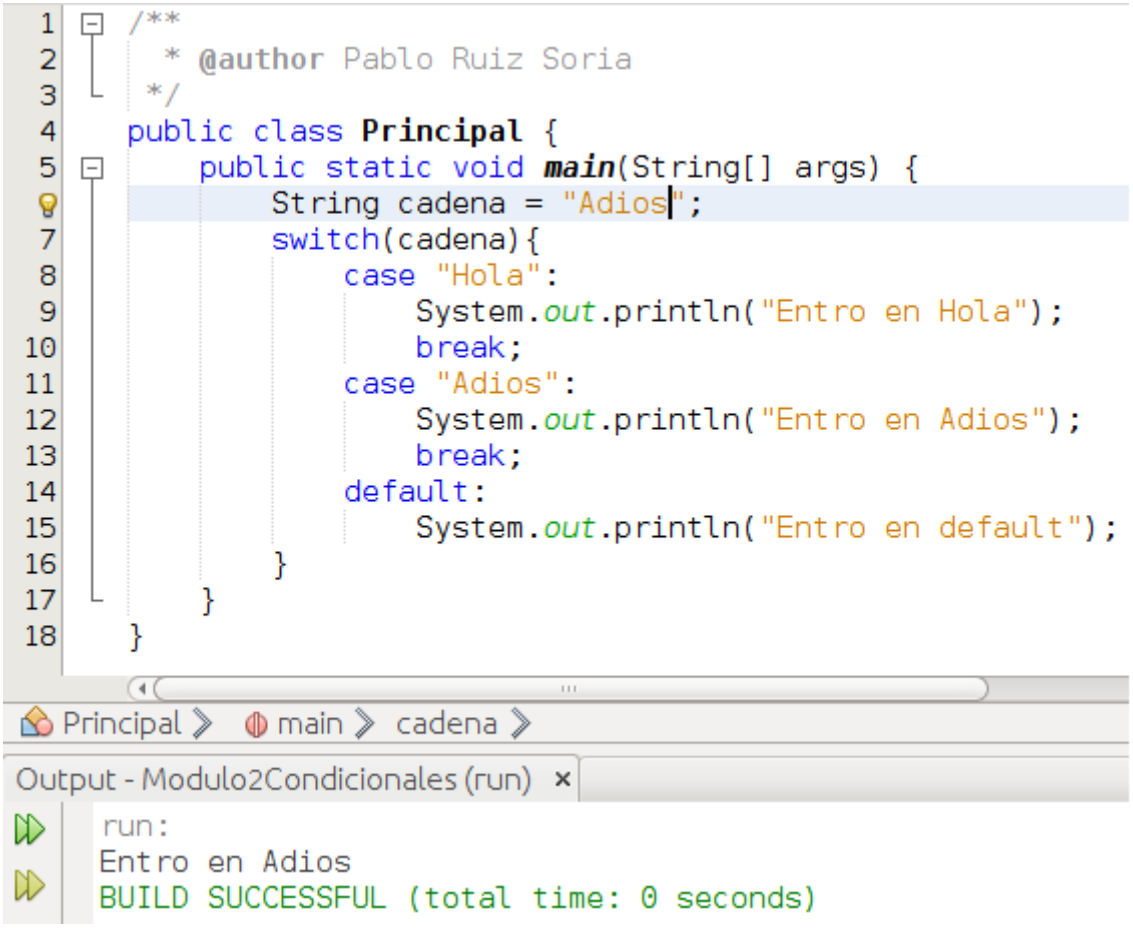
Se va a escribir algo por pantalla ¿el qué? dependerá del resultado que produzca la variable avisarFamilia, si la variable avisarFamilia es verdadero se devolverá "Avisar a la familia" y será eso lo que se escriba. Si la variable avisarFamilia es falso se devolverá lo que está tras los 2 puntos, es decir, "" y se escribirá una línea vacía.

Una vez vista la estructura if/else if/else vamos a continuar con la estructura **switch**. Su sintaxis es la siguiente:

```
switch(expresión){
    case valor1:
        Sentencias;
        break;
    case valor2:
        Sentencias;
        break;
    default:
        Sentencias;
}
```

De la sintaxis anterior es obligatorio utilizar la primera línea y la última y debe haber al menos 1 case con sus sentencias y un break tras las sentencias. El elemento default es opcional. Como quizás resulte algo complejo de explicar teóricamente vamos a ver un ejemplo:

Copyright 2026



```java
/**
 * @author Pablo Ruiz Soria
 */
public class Principal {
    public static void main(String[] args) {
        String cadena = "Adios";
        switch(cadena){
            case "Hola":
                System.out.println("Entro en Hola");
                break;
            case "Adios":
                System.out.println("Entro en Adios");
                break;
            default:
                System.out.println("Entro en default");
        }
    }
}
```

Principal > main > cadena

Output - Modulo2Condicionales (run)

```
run:
Entro en Adios
BUILD SUCCESSFUL (total time: 0 seconds)
```

En la línea 7 indicamos a nuestro switch que la variable de referencia es cadena. En la línea 8 comprobamos si "Hola" es igual a cadena y como no lo es pasamos al siguiente case, al de la línea 11. En la línea 11 se evalúa si la variable cadena es igual a "Adios" y como lo es entramos a ejecutar todas las sentencias ubicadas dentro de nuestro case, en este caso la de la línea 12. Luego llegamos a la línea 13, donde nos encontramos un break, este break nos lanza fuera del switch, a la línea 16. Si olvidásemos añadir el break de la línea 13 se ejecutaría la sentencia de la línea 12 y luego la sentencia de la línea 15 por ello hay que finalizar cada case con break.

Si hablamos de la **eficiencia interna** de cada una de estas estructuras condicionales la documentación de Java indica que es mas óptimo hacer uso de switch si bien es cierto que en los desarrollos que realicemos en el aula con nuestro alumnado no vamos a notar la diferencia de rendimiento.

Copyright 2026

Bucles

Ha llegado el momento de hablar de los bucles y de sus distintas formas en Java.

Lo primero es definir que es un bucle, en la wikipedia podemos encontrar la siguiente definición de bucle:

> Un bucle o ciclo, en programación , es una sentencia que se realiza repetidas veces a un trozo aislado de código, hasta que la condición asignada a dicho bucle deje de cumplirse.
>
> https://es.wikipedia.org/wiki/Bucle_(programaci%C3%B3n)

Ahora que sabemos lo que es vamos a ver los distintos bucles existentes en Java. Vamos a comenzar con el bucle **for**. Su sintaxis es la siguiente:

```
for(iniciacion; condicionFin; accionSobreIniciacion){
    sentencias;
}
```

En iniciacion deberemos establecer el valor inicial que queremos dar a nuestra variable de control. En condicionFin deberá haber alguna condición lógica y en accionSobreIniciacion deberemos establecer que queremos que pase tras cada iteracción. En sentencias estableceremos las sentencias de código que necesitemos. No se considera una buena práctica modificar la variable de iniciación en las sentencias. Una vez mas, vamos a reforzar la explicación con un ejemplo que incluya código con el fin de facilitar la compresión.

Copyright 2026

```java
/**
 * @author Pablo Ruiz Soria
 */
public class Principal {
    public static void main(String[] args) {
        int[] array1 = new int[4];
        for(int i = 0; i < array1.length ;i++){
            array1[i] = 10 * i;
        }
        System.out.print("[");
        for(int i = 0; i < array1.length ;i++){
            System.out.print(array1[i]);
            if(i < array1.length -1){
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```

Principal > main >

Output - Modulo2For (run) ×

```
run:
[0, 10, 20, 30]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Vamos a analizar esta pieza de código. En la línea 6 creamos un array de 4 posiciones (tal como vimos en el apartado Arrays). En la línea 7 tenemos un bucle for y en el nos encontramos int i = 0, esto hace que solo la primera vez que se accede al bucle se cree una variable llamada i que valga 0. Posteriormente se evalúa la condición array1.length (nos devuelve la longitud del array) y como es verdad (0 es menor que 4) se procede a ejecutar las sentencias del interior del for. En este caso se trata de una única sentencia que lo que hace es establecer un valor en la posición del array. Una vez se han ejecutado todas las sentencias de dentro de bucle for se ejecuta i++ y se reevalúa la condición, si esta es cierta se vuelven a ejecutar las sentencias del bucle y así hasta el momento en que la condición sea falsa, momento en el cual se sale del bucle. En la línea 10 escribimos por pantalla un corchete y posteriormente, en el bucle, recorremos las posiciones del array mostrando su valor y además en caso de que no se trate de la última posición del array añadimos una coma. Cuando termina el bucle se añade el cirre del corchete antes escrito. Voy a escribir lo que ocurre en el primer bucle for de un modo menos prosaico a continuación:

1º Se crea una variable llamada i y se le asigna el valor 0

2º Se comprueba que la variable i (que vale 0) sea menor a array1.length (que vale 4). Es verdad.

3º Ejecuto array1[i] = 10 * i; es decir, en la posición 0 del array1 pongo el resultado de multiplicar 10 * 0.

4º Hago i++. i pasa a valer 1.

5º Se comprueba que la variable i (que vale 1) sea menor a array1.length (que vale 4). Es verdad.

6º Ejecuto array1[i] = 10 * i; es decir, en la posición 1 del array1 pongo el resultado de multiplicar 10 * 1.

7º Hago i++. i pasa a valer 2.

8º Se comprueba que la variable i (que vale 2) sea menor a array1.length (que vale 4). Es verdad.

9º Ejecuto array1[i] = 10 * i; es decir, en la posición 2 del array1 pongo el resultado de multiplicar 10 * 2.

10º Hago i++. i pasa a valer 3.

11º Se comprueba que la variable i (que vale 3) sea menor a array1.length (que vale 4). Es verdad.

12º Ejecuto array1[i] = 10 * i; es decir, en la posición 3 del array1 pongo el resultado de multiplicar 10 * 3.

13º Hago i++. i pasa a valer 4.

14º Se comprueba que la variable i (que vale 4) sea menor a array1.length (que vale 4). Es falso, se sale del bucle.

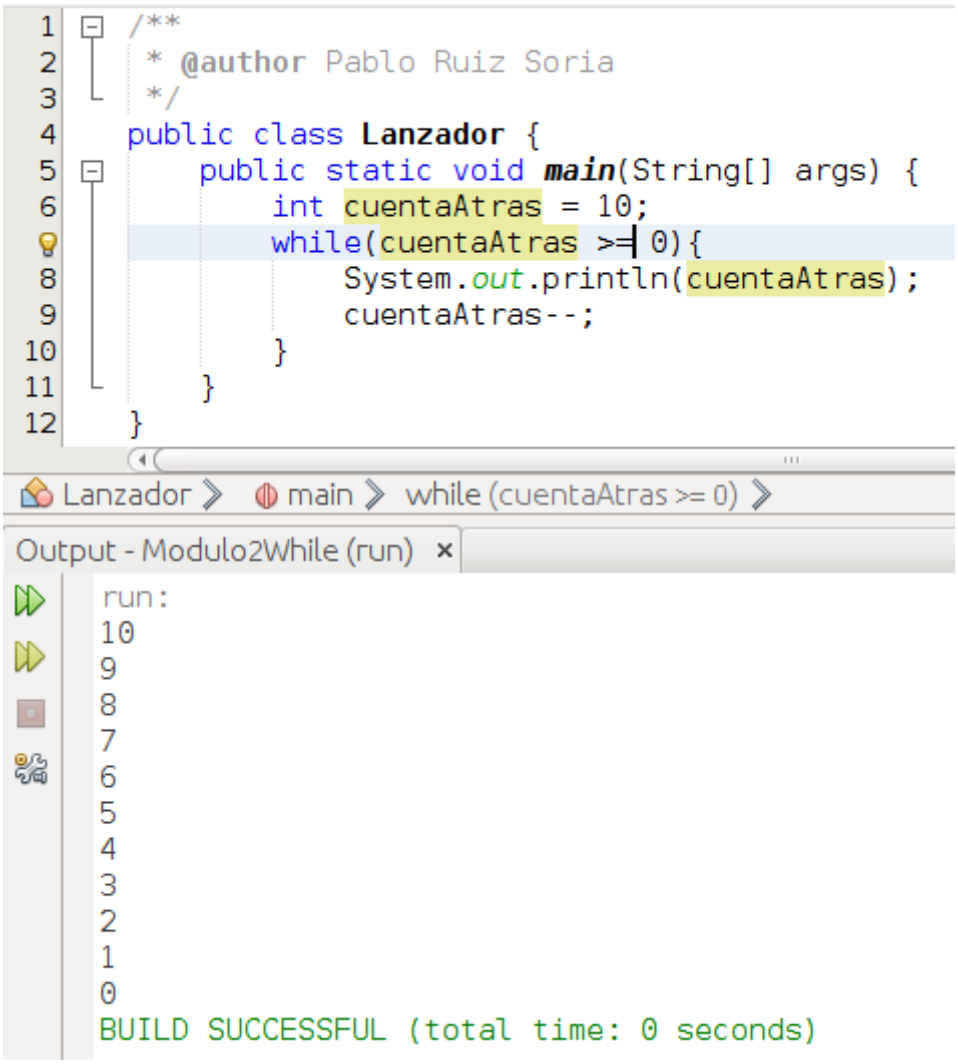
En Java existe otra manera de trabajar con bucles for, vamos a ver la sintaxis a continuación pero veremos ejemplos en el apartado de estructuras de almacenamiento de datos.

for(tipoVariable nombreVariable : listaQueContieneVariablesDeTipoVariable){ sentencias; }

Una vez hemos visto las posibilidades que nos ofrece el bucle for vamos a ver los **bucles while**. Los bucles while tienen la siguiente sintaxis:

```
while(condicion){
    sentencias;
}
```

En un bucle while las sentencias de su interior se van a estar ejecutando siempre mientras la condición del while sea verdadera. Por ello, salvo que queramos hacer un bucle infinito, en las sentencias actuaremos sobre la variable o variables que participen en la condición. Vamos a ver un ejemplo:

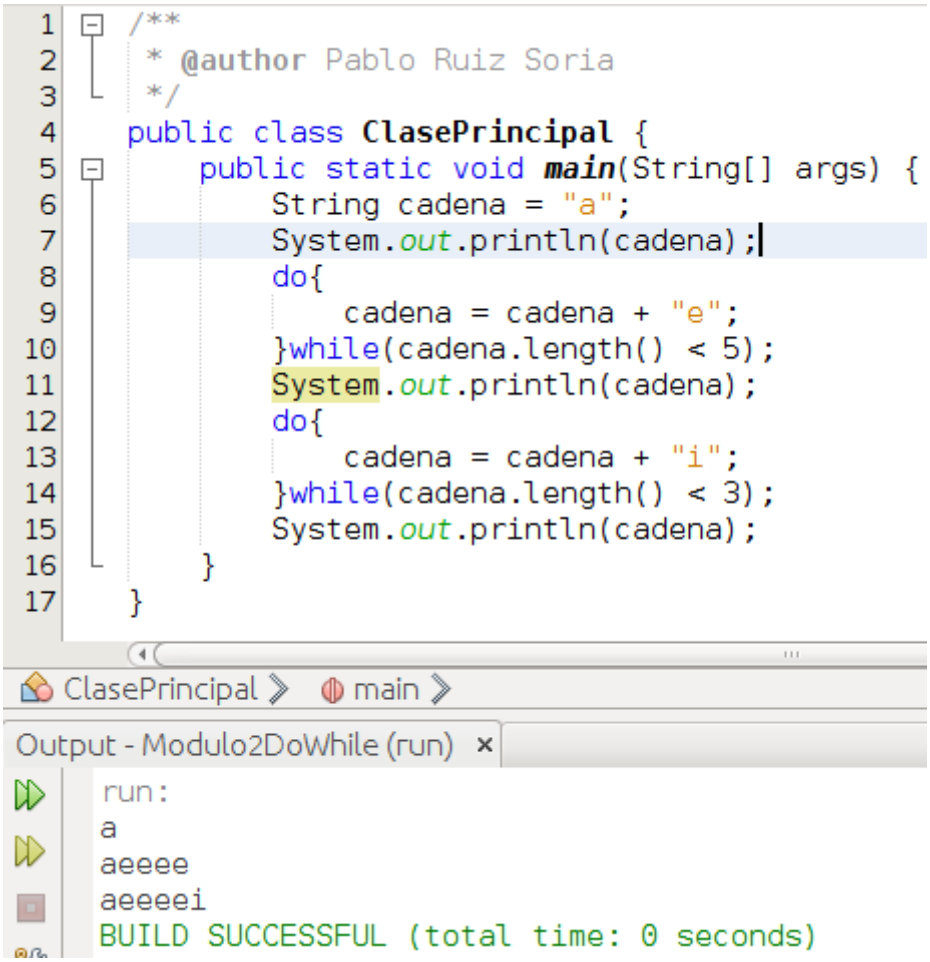


Lo que ocurre en el código de la imagen es lo siguiente. En la línea 6 creamos una variable de tipo entero con valor 10. En la línea 7 se evalúa que la variable antes creada sea mayor o igual a 0. Si es verdad, se ejecutan las sentencias de las líneas 8 y 9. Si es falso, se sale del bucle while.

En último lugar vamos a hablar de los bucles **do-while**. Estos bucles tienen la siguiente sintaxis:

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

Los bucles do-while son prácticamente iguales a los bucles while con la salvedad de que en un bucle do-while las sentencias contenidas dentro del mismo se van a ejecutar siempre 1 vez y luego se seguirán ejecutando mientras la condición sea verdadera. Al igual que en el caso anterior, salvo que queramos hacer un bucle infinito, en las sentencias deberemos actuar sobre la variable o variables que participen en la condición.



Vamos a analizar que sucede en este código. En la línea 6 creamos una variable de tipo String con el valor a. En la línea 7 mostramos en una línea el valor de nuestra variable cadena. Entre las líneas 8 y 10 nos encontramos un bucle do-while, procedemos ejecutando todo lo que está dentro del bucle por lo que a nuestra variable cadena le añadimos una e al final, ahora evaluamos la condición. Si la condición es cierta volvemos a ejecutar todas las sentencias contenidas dentro del do-while. Si la condición es falsa hemos terminado con el do-while. En la línea 11 mostramos en una línea el valor de nuestra variable cadena. Entre las líneas 12 y 14 nos encontramos otro bucle do-while, así que antes de nada ejecutamos las sentencias de su interior, en este caso a nuestra variable cadena le añadimos una i al final y una vez hemos ejecutado las sentencias procedemos a evaluar la condición.Si la condición es cierta volvemos a ejecutar todas las sentencias contenidas dentro del do-while. Si la condición es falsa hemos terminado con el do-while. En la línea 15 mostramos en una línea el valor de nuestra variable cadena.

Presta atención a este detalle: Para conocer el tamaño de un array utilizamos nombreVariableDeTipoArray.length mientras que para conocer la longitud de una cadena utilizamos nombreVariableTipoString.length(). La diferencia radica en el uso del paréntesis. Es algo que en nuestros primeros pasos con Java puede ocasionarnos algún pequeño trastorno.

Copyright 2026

Funciones

En este apartado vamos a hablar de las **funciones**, también llamadas **procedimientos**, y vamos a nombrar los **métodos** pero estos últimos los veremos en mayor profundidad en el módulo 3.

Una función es un trozo de código definido por un nombre a la cual se le pueden pasar parámetros o no y que puede o no devolver un valor.

Como primera aproximación a los métodos vamos a indicar que son algo muy parecido a una función pero ligado a un objeto. En el siguiente módulo profundizaremos en los métodos.

Vamos a ver unos trozos de códigos para comentarlos mas adelante:

Código con ejemplo de cálculo de medias

Ahora vamos a ver como hacer esto con una función:

Código con ejemplo de cálculo de medias con funciones

Y vamos a ver que hacemos y porqué. Lo primero es indicar que ambos códigos hacen lo mismo como podemos ver en la salida del programa sin embargo en la 2ª imagen entre las líneas 6 y 8 creamos una función que devuelve un float, tiene por nombre hazMedia y necesita 2 parámetros de tipo float. Esta función realiza la media de ambos datos y la retorna. En el segundo programa, en las líneas 13 y 17, vemos que para llamar a esta función es suficiente con indicar a que variable vamos a asignarle el valor devuelto y facilitar las variables con las que queremos que se haga el cálculo. Este ejemplo es trivial, pero si pensásemos en funciones mas complejas veríamos que al sacar factor común del código repetido este queda mucho mas simple y legible, además, si utilizamos una función estamos siguiendo el principio DRY (Don't Repeat Yourself, no te respitas) que nos va a permitir el que en caso de tener que realizar un cambio en el algoritmo únicamente haya que hacerlo en la función y no en todos los sitios donde aparezca el código no llevado a la función.

Vamos a ver ahora el código de una función que no devuelve nada:

Codigo con ejemplo de uso de funciones y procedimientos

En Java podemos hacer uso de la recursividad en las funciones, a continuación tenemos un ejemplo:

Ejemplo de código con función recursiva

Copyright 2026

Copyright 2026

Estructuras de almacenamiento de datos

Antes de comenzar vamos a conocer la definición que de **estructura de datos** ofrece la wikipedia:

> En programación, una estructura de datos es una forma particular de organizar datos en una computadora para que pueda ser utilizado de manera eficiente.
>
> https://es.wikipedia.org/wiki/Estructura_de_datos

En capítulos anteriores ya hemos trabajado con una estructura de datos, los **Arrays** (los cuales nos permitían almacenar datos en **vectores**).

Trabajar con Arrays puede ser suficiente para nuestras prácticas de aula pero conviene conocer las interfaces List, Map y Set. En el siguiente módulo del curso veremos que es una interface pero ahora nos interesan conocer algunas implementaciones de las interfaces antes mencionadas.

ArrayList, HashMap y HashSet son, respectivamente, algunas de las implementaciones de estas interfaces.

Vamos a ver para que usar cada una de estas estructuras de datos:

- **Arrays (vectores)**: Es la forma mas eficiente de almacenar objetos pero una vez defines el tamaño del vector no puedes ampliarlo o reducirlo. Además no puedes guardar variables de distinto tipo
- **List**: Almacena las variables en el orden en que se insertan. Nos permite tener valores duplicados en la lista. Nos permite tener variables de distinto tipo (al declararla no pondremos como en nuestro ejemplo)
- **Map**: No almacena el orden en que se insertan los datos (algunas de sus implementaciones si lo hacen). Para almacenar los datos se hacen usando el par clave-valor. No permite valores de clave repetidos pero si valores de valor repetidos. Nos permite tener variables de distinto tipo.

Copyright 2026

- **Set**: No almacena el orden en que se insertan los datos (algunas de sus implementaciones si lo hacen). No permite valores duplicados. Nos permite tener variables de distinto tipo. Es lo que debemos elegir si no queremos tener elementos repetidos en nuestra estructura de datos.

En este capítulos vamos a centrarnos en la clase **ArrayList**, vamos a ver un ejemplo:

```
1  import java.util.ArrayList;
2  /**
3   * @author Pablo Ruiz Soria
4   */
5  public class ClasePrincipal {
6      public static void main(String[] args) {
7          String[] vectorDeStrings = new String[2];
8          for(int i = 0; i < vectorDeStrings.length ;i++){
9              vectorDeStrings[i] = "Hola mundo";
10         }
           for(int i = 0; i < vectorDeStrings.length ;i++){
12             System.out.println(vectorDeStrings[i]);
13         }
14         ArrayList<String> listadeStrings = new ArrayList();
15         for(int i = 0; i < 2 ;i++){
16             listadeStrings.add("Hello world");
17         }
18         for(int i = 0; i < listadeStrings.size() ;i++){
19             System.out.println(listadeStrings.get(i));
20         }
21     }
22 }
```

```
Output - Modulo2EstructurasDeAlmacenamiento (run)
run:
Hola mundo
Hola mundo
Hello world
Hello world
```

Lo primero que vemos en la línea 1 es que para trabajar con la clase ArrayList hay que importarla. Entre las líneas 7 y 13 tenemos un ejemplo de como trabajar con Arrays (ya lo vimos con anterioridad). En la línea 14 nos encontramos con la creación de una variable llamada listadeStrings que es de tipo ArrayList y además le añadimos lo cual significa que en este ArrayList solo vamos a poder almacenar variables de tipo String. La primera diferencia con respecto a los Arrays es que aquí no definimos el tamaño del ArrayList. Esto es porque los ArrayList, a diferencia de los Arrays, son dinámicos. Podemos variar su tamaño en tiempo de ejecución según nuestras necesidades. En la línea 16 vemos como añadir un elemento a nuestro ArrayList. En la línea 18 vemos que para obtener el tamaño de un ArrayList utilizamos el método size. Y en la línea 19

vemos que para obtener una determinada posición de un ArrayList utilizamos el método get. Al igual que los en los Array, en los ArrayList se comienza a contar por 0. Hemos comentado anteriormente que los ArrayList son dinámicos por lo que en el ejemplo anterior podríamos añadir un tercer elemento a la lista sin necesidad de crear otra variable nueva, sin embargo, no podríamos hacerlo con el Array.

A continuación vamos a ver como quedaría el ejemplo anterior eliminando la parte relativa a los arrays y utilizando for mejorados para recorrer el ArrayList:

```
1  import java.util.ArrayList;
2  /**
3   * @author Pablo Ruiz Soria
4   */
5  public class ClasePrincipal {
6      public static void main(String[] args) {
7          ArrayList<String> listadeStrings = new ArrayList();
8          for(int i = 0; i < 2 ;i++){
9              listadeStrings.add("Hello world");
10         }
           listadeStrings.remove(0);//eliminamos el primer elemento
           for(String elemento:listadeStrings){
13             System.out.println(elemento);
14         }
15         listadeStrings.add("Nuevo elemento");
16         System.out.println("-----------");
           for(String elemento:listadeStrings){
18             System.out.println(elemento);
19         }
```

```
Output - Modulo2EstructurasDeAlmacenamiento (run)
run:
Hello world
-----------
Hello world
Nuevo elemento
BUILD SUCCESSFUL (total time: 0 seconds)
```

Lo relevante del código anterior lo encontramos en la línea 11 donde hacemos uso del método remove que nos permite borrar el elemento de la lista que nos interese. En la línea 12 y 17 nos encontramos con unos bucles for distintos a los que habíamos utilizado hasta la fecha. En estos bucles for lo que decimos es que extraiga cada vez el siguiente elemento de la lista y lo guarde en una variable llamada elemento de tipo String.

Excepciones

En Java, las excepciones son una clase (documentación) que se lanza cuando ocurre un error en tiempo de ejecución. Es decir, no son errores de compilación de algo que hemos escrito mal y el compilador no comprende sino que son errores que ocurren mientras nuestro programa está siendo usado. Java nos ofrece la posibilidad de controlar estos errores y recuperarnos de ellos sin abortar la ejecución de nuestro programa.

Vamos a ver un ejemplo:

Ejemplo de código con excepciones

Lo que vemos aquí es que el programa compila y se ejecuta sin mayor problema pero durante su ejecución se intenta realizar una división por 0 y se lanza una excepción (ArithmeticException) con el error, nos indica en que línea de nuestro código ocurre (línea 7) y se detiene la ejecución del programa. En un entorno real no sería tolerable que aunque una de las muchas cosas que pueden fallar el programa se detenga, por ello lo que hay que hacer es controlar la excepción y que el programa continúe su ejecución. Vamos a ver como contener el error:

Ejemplo de código con excepción controlada

Lo que hacemos aquí es crear un bloque try-catch-finally. Dentro del try le decimos lo que queremos que haga y si hay algún error de tipo Exception lo capturamos y hacemos lo que está dentro del catch. El bloque finally es opcional y es un trozo de código que se ejecutará siempre falle o no la ejecución en el bloque try. Todas las excepciones, como ArithmeticException, derivan de Exception por eso si en un bloque catch capturamos Exception capturaremos cualquier excepción. Es muy cómodo capturar cualquier excepción en un bloque catch pero lo idea es poner un bloque catch para cada excepción y así saber que falla concretamente. Vamos a ver un ejemplo:

Ejemplo de código con excepción controlada en varios catch

En el código anterior aparecen 2 bloques catch, en el primero de ellos (línea 11) controlo las excepciones que pudiesen ocurrir de tipo aritmético y en el segundo bloque (línea 14) capturo cualquier otro tipo de excepción.

En ocasiones puede resultarnos útil lanzar nosotros mismos un error bien de tiempo genérico (Exception) o bien extendiendo la clase Exception para crearnos nuestro error personalizado. Dado

que aún no hemos visto como extender una clase vamos a ver a continuación un trozo de código donde lancemos una excepción genérica:

Ejemplo de código lanzando excepciones

En la imagen anterior vemos que creamos una función en la línea 6 que realiza la división de 2 parámetros. En la línea 7 comprobamos que el dividendo no sea 0 ya que no se puede dividir por 0. En caso de que el dividendo sea 0 en la línea 8 lanzamos una excepción (realmente un objeto realmente que inicializamos con su constructor, veremos estos conceptos en el siguiente módulo) con un mensaje personalizado. Como dentro de nuestra función lanzamos una excepción tenemos que indicar de algún modo que este módulo lanza excepciones de tipo Excepcion, por ello en la línea 6 añadimos a nuestra función throws Exception. En el mail lo único que hago es mostrar por pantalla el resultado de llamar a esta función. En el primer caso vemos que entra al catch (línea 15) mientras que en el segundo caso al no lanzarse ninguna excepción no se ejecuta el código contenido en el catch. Ninguno de estos try-catch incluye el bloque finally que como dijimos anteriormente es opcional.

Copyright 2026

Convenios de escritura

Cuando os hablé de "Constanes y variables" ya comenté algo sobre los convenios de escritura en Java. Vamos a recordarlo:

> Si eres observador/a te habrás dado cuenta de que los tipos básicos empiezan en minúscula mientras que los objetos comienzan en mayúsculas. Es probable que también te hayas percatado de que el nombre que asigno a las variables siempre comienza en minúsculas y el nombre que asigno a las constantes está completamente en mayúsculas, esto son convenios de escritura del lenguaje Java y tu puedes seguir tu propio estilo de código pero es interesante hacer uso del convenio porque facilita la lectura y comprensión del código yen caso de trabajar con otras personas se es mas productivo. En este módulo, un poco mas adelante, hablaremos en mayor profundidad de esta cuestión.
>
> Pablo Ruiz Soria - Capítulo "Constantes y variables"

Hay que dejar claro que un convenio no es mas que eso, un convenio. No estamos obligados a seguirlo pero si va a facilitarnos la labor a la hora de entender código de otras personas. Además, si acostumbramos a nuestro alumnado a seguirlo nos facilitará enormemente la tarea de corrección y detección de errores en su código. Voy a elaborar una pequeña tabla donde recoger algunas generalidades al respecto:

|Elemento|Explicación|Ejemplos| |:--|:--|:--| |Clases|Siempre el primer carácter en mayúsculas. Si son varias palabras, cada una de ellas separada por la primera mayúscula. Buscaremos nombres descriptivos pero no largos. Usaremos nombres en singular.|PreguntaRespuestaClasificacionHistorica| |Variables|Siempre el primer carácter en minúsculas. Si son varias palabras, cada una de ellas separada por la primera mayúscula. Buscaremos nombres descriptivos pero no largos. Usaremos nombres en singular.|preguntarespuestaclasificacionHistorica| |Constantes|Todo en mayúsculas. Si son varias palabras, cada una de ellas separada por guión bajo (_). Buscaremos nombres descriptivos pero no largos. Usaremos nombres en singular.|PREGUNTARESPUESTACLASIFICACION_HISTORICA| |Paquetes|En minúsculas. No es habitual que tengan varias palabras. Si son paquetes para web generalmente usan el dominio invertido.|com.trivinet.modelocom.trivinet.util| |Funciones y métodos|Siempre el primer caracter en minúsculas. Si son varias palabras, cada una de ellas

separada por la primera mayúscula. Buscaremos nombres descriptivos pero no largos. Utilizaremos verbos.|obtenerPregunta(...)borrarRespuesta(...)establecerClasificacionHistorica(...)| |Parámetros|Estableceremos 1 espacio en blanco tras la coma que separa los parámetros en una función para facilitar la lectura.|obtenerPregunta(int id, boolean activa)| |Operadores|Estableceremos 1 espacio en blanco tras los mismos para facilitar la lectura.|a = b + c;a == b| |Paréntesis dentro de paréntesis|Cuando un paréntesis no sea el último procuraremos dejar un espacio en blanco para facilitar la lectura.|escribir(multiplicar(4, 5));| |Bloques de código|Todo el código dentro de un bloque (clase, método, condición, bucle, excepción,...) debe tener la misma identación| |Longitud de las líneas|Evitaremos hacer líneas de longitud mayor a 80 caracteres (en Netbeans viene delimitado por una línea roja). Recordad que una sentencia, en Java, termina con un punto y coma, no con el fin de la línea.|

No me cansaré de insistir en que el compilador no comprueba que sigamos ningún convenio (ni que la lógica del programa sea correcta). El compilador únicamente va a comprobar que sintácticamente el código está escrito del modo adecuado.

Copyright 2026

Algoritmos y estructuras de resolución de problemas sencillos

Para dar solución a este apartado he preferido crear un proyecto completo con Netbeans de modo que en él dé solución a distintas cuestiones como:

- Cálculo de áreas de cuadrados.
- Cálculo de áreas de rectángulos.
- Cálculo de áreas de circunferencias.
- Cálculo de longitud de circunferencias.
- Dibujo de cuadrados con el símbolo *.
- Dibujo de rectángulos con el símbolo *.
- Dibujo de triángulos con el símbolo *.
- Cálculo de medias aritméticas con vectores.
- Cálculo de medias aritméticas con listas.

Sería demasiado largo y farragoso comentar en este apartado todo el código de este proyecto no obstante he procurado que el código sea muy claro.

Para hacer más sencilla la lectura he dividido el código en 3 clases main. Para ejecutar una u otra basta con que en Netbeans, en las propiedades del proyecto, selecciones que clase main quieres ejecutar en cada momento. También puedes colocarte encima de la clase que contenga main que quieras ejecutar y pulsando sobre ella con el click derecho elegir run.

Puedes descargar el código desde aquí (zip - 0.02 MB)..

Código utilizado en los ejemplos

Módulo 2 Ámbito de las variables (zip - 0.01 MB).

Módulo 2 Cadenas (zip - 0.01 MB).

Módulo 2 Funciones condicionales (zip - 0.01 MB).

Módulo 2 Bucles for (zip - 0.01 MB).

Módulo 2 Bucles while (zip - 0.01 MB).

Módulo 2 Bucles do-while (zip - 0.01 MB).

Módulo 2 Funciones (zip - 0.01 MB).

Módulo 2 Estructuras de almacenamiento de datos (zip - 0.01 MB).

Módulo 2 For mejorado (zip - 0.01 MB).

Módulo 2 Excepciones (zip - 0.01 MB).

Módulo 2 Lanzamiento de excepciones (zip - 0.01 MB).

Módulo 2 Algoritmos y estructuras de resolución de problemas sencillos (zip - 0.02 MB).

Copyright 2026

Tarea

Tu tarea una vez acabado el segundo módulo consiste en:

- Crear un proyecto llamado Modulo2NombreApellido donde Nombre sea tu nombre y Apellido tu primer apellido. Ejemplo: Modulo2PabloRuiz
- En el proyecto deberás crear una paquete llamado tarea.
- Dentro del paquete tarea deberás crear una clase llamada Principal. En esta clase estará el método main.
- La clase Principal deberá tener las funciones sumarEnteros, restarEnteros, multiplicarEnteros, dividirDecimales, calcularFactorial, sumarArray y sumarLista.
- La función sumarEnteros deberá tener 2 parámetros que serán enteros y deberá mostrar una línea con el resultado de la operación.
- La función restarEnteros deberá tener 2 parámetros que serán enteros y deberá devolver la resta del 2º parámetro al 1º.
- La función multiplicarEnteros deberá tener 2 parámetros que serán enteros y deberá devolver el producto de ambos.
- La función dividirDecimales deberá tener 2 parámetros que serán decimales (double) y devolverá otro decimal (double). Si el divisor es igual a 0 lanzará una excepción genérica con el mensaje "No se puede dividir por cero".
- La función calcularFactorial deberá tener 1 parámetro entero y deberá devolver el factorial del parámetro dado. Puedes elegir entre hacerlo recursivo o no. En caso de que el parámetro sea un número negativo se lanzará una excepción genérica con el mensaje "No puedo calcular el factorial de un número negativo"
- La función sumarArray recibirá un parámetro, que será un vector de enteros. Deberá sumar todos los valores y devolver el resultado.
- La función sumarLista recibirá un parámetro, que será una lista de tipo ArrayList que solo contendrá enteros. Deberá sumar todos y mostrar por pantalla el resultado.
- En el main deberás controlar las excepciones que lancen los métodos para los que hemos establecido excepciones. La forma de controlarlos será mostrando por pantalla el mensaje de la excepción y continuando ejecutando el resto de sentencias.

Copyright 2026