

Estructura general de un programa

por Pablo Ezequiel Jasinski

Todo programa tiene una estructura básica que se debe conocer, y en el caso de Java presenta elementos que tienen que ver con la programación orientada a objetos, tema que se irá desarrollando de a poco.

Para explicar esta estructura se utilizará el ejemplo “Hola mundo”, ya visto en el primer capítulo.

```
1  // HolaMundo.java
2  // Programa para imprimir texto por pantalla
3
4  public class Principal {
5
6      // El método main comienza la ejecución de la aplicación
7      public static void main(String[] args) {
8
9          System.out.println("Hola mundo");
10
11      } // Fin del método main
12
13 } // Fin de la clase Principal
```

En Java un programa está conformado por clases, por lo tanto, siempre que se programe, se estará programando sobre una clase. Ésta hay que definirla, y para hacerlo se debe escribir la palabra clave **class**, seguida del **nombre de la clase**. Luego se debe escribir una llave de apertura { y una llave de cierre }, y todos el código de la clase irá dentro de estas llaves.

En el ejemplo se observa la clase Principal definida, y dentro de ella un método con código.

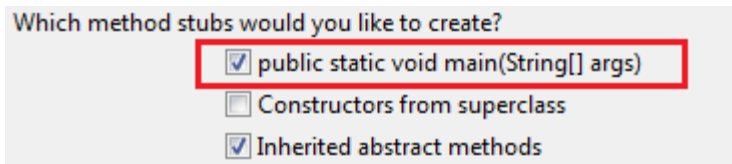
```
public class Principal {
    ...
}
```

En primera instancia se trabajará anteponiendo la palabra clave **public** en la definición de una clase, para no crear restricciones que puedan interferir con el funcionamiento del programa, más adelante se desarrollara el tema pertinente a esta palabra clave.

Luego de definir la clase, se debe escribir el código con el cual se trabajará. En la clase principal de todo programa que se cree, siempre debe existir el método **main**. Éste método es el primero que Java invocará al iniciar la ejecución del programa, luego de esto, podrán ser invocados otros métodos, según el programador lo defina. Debe tenerse en cuenta que el método main se debe definir solo una vez en un programa. Para definir el método main se debe poner siempre exactamente esto:

```
public static void main(String[] args) {
    ...
}
```

Con eclipse no hace falta estrictamente escribir todo esto para definir al método main, ya que cuando se crea una nueva clase, existe la opción de marcar una casilla con la leyenda “*public static void main (String[] args)*” lo cual hará que Eclipse escriba el método automáticamente.



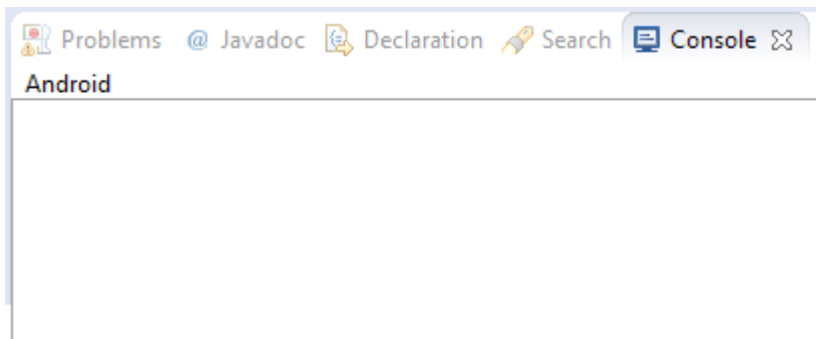
Al igual que en la definición de una clase, se deben colocar las llaves que indicarán el alcance del método, y en donde se deben poner las expresiones que se consideren para darle funcionalidad al programa.

println y scanner

Dos elementos fundamentales para comenzar a programar

La mayoría de los programas en la actualidad, tienen la característica de interactuar con el usuario, por lo cual el usuario puede ingresar valores o datos al programa, crear archivos para que el mismo los lea, o interactuar con elementos gráficos que signifiquen algo para la aplicación, a su vez que el programa puede mostrar información relevante para el usuario, en forma de texto, archivos o gráficos y así interactuar con el mismo.

En primera instancia, la forma más primitiva que utiliza Java para la interacción con el usuario es la consola, la cual está ubicada generalmente en la parte inferior del IDE.



Mediante la consola, se podrá mostrar texto, así como también pedir algún valor a un usuario. Para mostrar texto en la consola hay varias formas, dentro de las más utilizadas existen dos:

```
System.out.print(""); // Sirve para mostrar texto en una sola línea
System.out.println(""); // Sirve para mostrar texto seguido de un salto de línea
```

Ejemplo del uso de estos métodos:

```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          System.out.print("Este es un mensaje ");
6          System.out.print("que se muestra en una sola línea");
7          System.out.println("Mientras este mensaje");
8          System.out.println("se muestra en");
9          System.out.println("varias líneas");
10
11     } // fin del método main
12 } // fin de la clase Principal

```

Dentro de los paréntesis se pueden utilizar el operador de concatenación de cadenas, y mostrar valores de otros tipos como enteros, decimales y booleanos.

La forma de interaccionar con el usuario, es pidiéndole que ingrese algún valor. Para esto existe la clase Scanner. Para poder utilizar una clase, por lo general, se la debe instanciar generando un objeto de ésta misma, concepto que se explicará más adelante. Para crear un objeto de una clase se debe:

1. Definir el atributo de instancia
2. Crear el objeto

Definir un atributo de instancia es parecido a definir un atributo de tipo primitivo, solo que en vez de utilizar los tipos primitivos, se utilizará la clase como tipo, y luego de esto se asignará un nombre que describa al atributo.

Para definir el atributo de instancia de la clase Scanner se procede de la siguiente manera:

Tipo o clase

```

Scanner entrada;

```

Nombre del
atributo de
instancia

Una vez definido el atributo, se debe crear el objeto, para crear un objeto se escribe la palabra clave **new** seguido del nombre de la clase la cual se debe escribir poniendo paréntesis “()” al final de este, como si se tratara de un método, y dentro de los paréntesis se deberán pasar argumentos, si así la clase los requiera. Esto se debe a que, al crear un objeto, se llama al método principal o **constructor** del mismo, el cual se ejecutará y podrá recibir o no parámetros, según como este programado.

Para crear un objeto de Scanner se procede de la siguiente manera:

```

entrada = new Scanner(System.in);

```

atributo de
instancia

clase de la cual
se creará el
objeto

palabra clave
para crear
objeto

parámetro de
entrada que
exige el
constructor de
la clase Scanner

Como se puede ver se crea el objeto de Scanner nombrado entrada, y luego entre paréntesis se pasa un parámetro a la función constructor de la clase Scanner, porque ésta así lo exige. Sin detallar mucho, este parámetro sirve para indicar que el objeto creado servirá como medio para el ingreso de datos.



Para usar algunas clases, como por ejemplo la clase Scanner, se deben importar las librerías que la contengan. Las librerías se importan antes de la definición de la clase, y para hacerlo se debe poner el siguiente código: `import` rutaDeLaLibrería;

Para poder utilizar la clase Scanner se debe importar mediante el siguiente código:

```
import java.util.Scanner;
```



Tip: Cuando se escribe Scanner entrada; eclipse acusará error, ya que no se ha importado la librería que contiene la clase Scanner, y se observa que la palabra Scanner estará subrayada en color rojo. Al hacer clic sobre la palabra que esta subrayada y presionar la combinación de teclas **ctrl + o**, eclipse importa automáticamente la librería necesaria para trabajar con la clase, ahorrando al programador el trabajo de escribir toda la ruta. En el caso de que se encuentren clases con el mismo nombre en diferentes librerías, el IDE desplegará una lista con las opciones existentes de rutas a importar, y el programador deberá escoger la adecuada.

Una vez hecho todo esto el objeto entrada, se encontrará listo para usar, y mediante este se podrán invocar métodos y atributos que pueda contener la clase Scanner. Para invocar métodos o atributos de la clase mediante el objeto se debe escribir el nombre del atributo de instancia, y luego un punto "." que actúa de comunicador, y de esta manera poder llamar métodos y atributos.

Los métodos que más pueden servir para el cometido de este tema son los siguientes:

```
next();           // Se espera que el usuario ingrese por consola un valor de tipo String
nextInt();        // Se espera que el usuario ingrese por consola un valor de tipo int
nextBoolean();    // Se espera que el usuario ingrese por consola un valor de tipo boolean
nextFloat();      // Se espera que el usuario ingrese por consola un valor de tipo float
nextDouble();     // Se espera que el usuario ingrese por consola un valor de tipo double
close();          // Se cierra el scanner para que no consuma más recursos
```

Los primeros cinco métodos esperan que el usuario ingrese un valor mediante la consola, y ese valor debe ser guardado en alguna variable para que el mismo no se pierda. Cuando no se use más el scanner se debe cerrar para que éste no consuma recursos en memoria mediante el método close(). Ejemplos de utilización de estos métodos serían:

```
String nombre = entrada.next();
int edad = entrada.nextInt();
float impuesto = entrada.nextFloat();
```

Ejemplo:

```
1  import java.util.Scanner;
2
3  public class Principal {
4
5      public static void main(String[] args) {
6
7          Scanner entrada = new Scanner(System.in);
8          String nombre, apellido;
9          byte edad;
10
11          System.out.println("Ingrese su nombre");
12          nombre = entrada.next();
13
14          System.out.println("Ingrese su apellido");
15          apellido = entrada.next();
16
17          System.out.println("Ingrese su edad");
18          edad = entrada.nextByte();
19
20          entrada.close();
21
22          System.out.println("Usted ha ingresado a la matrix");
23          System.out.println("Ahora se lo conocerá como " + nombre + " " +
24                              apellido + " y comenzará una nueva vida con
25                              la edad de " + edad + " años. ");
26
27
28      } // fin del método main
29  } // fin de la clase Principal
```



Buenas prácticas: Cuando se utilizan objetos que pueden tener acceso a muchos métodos de una clase, es buena práctica dirigirse a la documentación oficial de Oracle para conocer todos los métodos y atributos de la clase a los que se puedan tener acceso.

La documentación de todas las clases de Java se puede encontrar en:

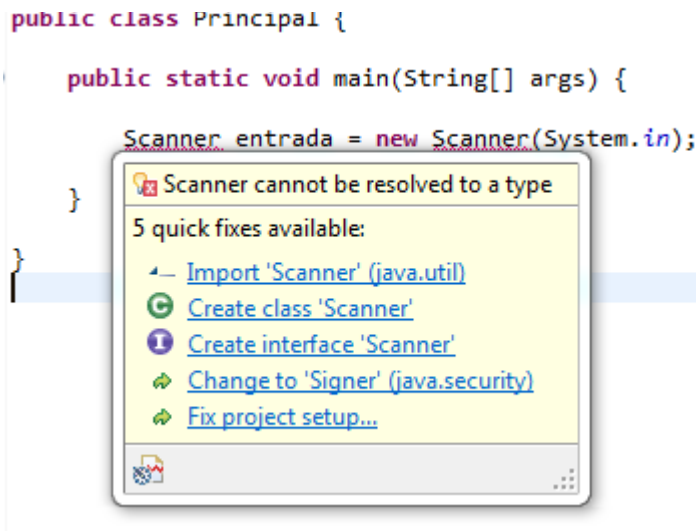
<http://docs.oracle.com/javase/7/docs/api/>

La documentación de la clase Scanner puede encontrarse en:

<http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>

Resolución de errores

Cuando se programa con algún tipo de error, eclipse lo detecta en tiempo de programación y subraya en rojo las palabras en donde se contiene el error. Una forma fácil es resolver errores utilizando la ayuda que proporciona el ide. Para eso se debe posicionar el puntero del mouse sobre la palabra que se encuentra subrayada y si existen sugerencias para resolver el error, eclipse listará las posibles soluciones. Para resolver el error marcado entonces, el programador debe elegir de la lista propuesta por eclipse, la opción que resulte más conveniente.



Estructuras de control

Una estructura de control es una serie de palabras reservadas con una estructura predefinida por el lenguaje, la cual sirve para diferentes motivos, en los que se encuentran, la toma de decisiones, para hacer que mediante una condición se ejecute una porción de código o bien otra, repetición de código, para generar un bucle que repita el código que se encuentre dentro de este, y de ésta manera generar algoritmos eficientes para distintos casos, etc.

Estructuras de selección

Las estructuras de selección sirven para tomar decisiones, según una evaluación o condición, mediante la cual tornará el flujo de ejecución para cierto camino según sea el resultado de la evaluación.

Ejemplo de una estructura de selección en la vida real puede ser la situación en la que un estudiante se levanta a la mañana para ir al colegio, sale al patio de su casa para ver si llueve y respecto a esta evaluación llevará paraguas o no.

Si llueve entonces **llevar paraguas** sino **no llevar paraguas**

Estructura if de selección simple

La estructura if de selección simple sirve para evaluar una condición, y en el caso de que la condición se cumpla o dicho de otra manera que la condición sea verdadera, se ejecutará el código de la siguiente instrucción.

Para usar esta estructura primero se debe escribir la palabra clave `if` luego de esto se colocan paréntesis de apertura y cierre, y dentro de los paréntesis se coloca la condición a ser evaluada. Luego en la misma línea o bien en la línea siguiente se escribe la instrucción que se quiera ejecutar en caso de que el resultado de la condición sea verdadera. Si la instrucción a ejecutar en el caso de que la condición se cumpla es una sola, pueden obviarse las llaves "{}", en otro caso se debe escribir, luego de la condición una llave de apertura "{", y luego de las instrucciones a ejecutar una llave de cierre "}".

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int nro = 3;
6
7          if (nro > 0) System.out.println("El número es positivo");
8
9      } // fin del método main
10 } // fin de la clase Principal
```

Estructura if de selección doble

La estructura if de selección doble, se diferencia de la simple en que, ésta última ejecuta solo las instrucciones siguientes si la condición a evaluar es verdadera, de otra manera ignora estas instrucciones y no hace más nada, en cambio en la estructura if de selección doble, existe otra porción de código con instrucciones que se ejecutarán en el caso de que la condición sea falsa.

Para usar esta estructura se procede de la misma manera que con la estructura if de selección simple, solo que al final de las instrucciones que se ejecutarían si la condición es verdadera, se escribe la palabra clave `else` y seguido de ésta las instrucciones a ejecutarse si la condición es falsa. De la misma manera se utilizan las llaves para delimitar y organizar que instrucciones pertenecerán al bloque que se ejecutará si la condición es verdadera y lo mismo para el caso de que la condición sea falsa.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int nro = 3;
6
7          if (nro > 0) {
8              System.out.println("El número es positivo");
9          } else {
10             System.out.println("El número no es positivo");
11         }
12
13     } // fin del método main
14 } // fin de la clase Principal
```

Estructuras if ... else anidadas

Cuando se necesitan contemplar más situaciones que en los casos anteriores, se puede recurrir al anidamiento de estas instrucciones. En el ejemplo anterior, se podía informar si un era positivo o no, pero si se quisiera informar si un número es positivo, negativo o cero, de la forma mostrada anteriormente, no se podría hacer, por lo cual se recurre a anidar estas instrucciones de selección.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int nro = 3;
6
7          if (nro > 0) {
8              System.out.println("El número es positivo");
9          } else if (nro < 0) {
10             System.out.println("El número es negativo");
11          } else {
12             System.out.println("El número es cero");
13          }
14
15     } // fin del método main
16 } // fin de la clase Principal
```

De esta manera pueden anidarse todas las instrucciones if ... else que se desea para generar cuantos casos sean necesarios.

Operador condicional ternario (?:)

El operador condicional “?:” es el único operador ternario que existe en Java, esto quiere decir que es el único operador que utiliza tres operandos. El mismo puede servir para reemplazar al if ... else y la estructura de este operando es:

Condición ? valor si la condición es verdadera : valor si la condición es falsa

Por lo tanto, como todo operador, este devuelve un resultado, del tipo que el programador le indique, y que podrá ser almacenado en algún atributo, o bien podrá ser mostrado por consola.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          byte nota = 5, nro = 0;
6          String res;
7
8          res = nota >= 6 ? "El alumno aprueba" : "El alumno no aprueba";
9
10         System.out.println(res);
11
12         res = nro > 0 ? "El número es positivo" : nro < 0 ? "El número es
13             negativo" : "El número es 0";
14
15         System.out.println(res);
16
17     } // fin del método main
18 } // fin de la clase Principal
```


Como se puede observar, también es posible anidar estos operadores para generar instrucciones if ... else anidadas.

Estructura switch case

La estructura switch case, sirve para realizar selecciones, determinando diferentes casos en los valores que pudiese tomar un atributo. Esta estructura se utiliza a menudo para reemplazar a las instrucciones if ... else anidadas, ya que con la misma se pueden presentar las condiciones y las decisiones de una manera más prolija y ordenada. Para usar esta estructura se debe escribir la palabra clave **switch** y entre paréntesis el atributo a evaluar. Luego se abren llaves, y se deben ir detallando los casos mediante la palabra reservada **case**, luego el valor a evaluar seguido por dos puntos ":" debajo de esto se deberá escribir el código correspondiente a la acción a realizar si ese fuera el valor del atributo, y para finalizar el caso se pone la palabra clave **break**; Se pueden poner cuantos casos necesarios y existe una opción, la cual ejecutará un código determinado, en el caso de que el atributo no corresponda a ningún caso anterior. Para esto se debe escribir la palabra clave **default**: luego el código y al igual que en cada caso se debe finalizar la instrucción con la palabra break. Para finalizar el switch se debe cerrar la llave que se abrió previamente.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          Scanner entrada = new Scanner(System.in);
6          System.out.println("Ingrese un número entre 1 y 3");
7          byte nro = entrada.nextByte();
8          entrada.close();
9          switch(nro) {
10
11              case 1:
12                  System.out.println("Ingreso el número 1");
13                  break;
14
15              case 2:
16                  System.out.println("Ingreso el número 2");
17                  break;
18
19              case 3:
20                  System.out.println("Ingreso el número 3");
21                  break;
22
23              default:
24                  System.out.println("No ingreso un número válido");
25                  break;
26
27          } // fin del switch
28      } // fin del método main
29 } // fin de la clase Principal
```

El ejemplo anterior equivale a hacer lo siguiente:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          Scanner entrada = new Scanner(System.in);
6          System.out.println("Ingrese un número entre 1 y 3");
7          byte nro = entrada.nextByte();
```

```

8      entrada.close();
9
10     if(nro == 1){
11         System.out.println("Ingreso el número 1");
12     } else if (nro == 2){
13         System.out.println("Ingreso el número 2");
14     } else if (nro == 3){
15         System.out.println("Ingreso el número 3");
16     } else {
17         System.out.println("No ingreso un número valido");
18     } // fin del if
19
20 } // fin del método main
21 } // fin de la clase Principal

```

Como se puede observar, el resultado del programa será el mismo, pero la forma en que se presentan los datos difiere, y en el caso del uso del switch case, estos se presentan de una manera más ordenada.

Estructuras de repetición

A menudo es necesario utilizar estructuras de repetición, a modo de simplificar el código, y no llenar el programa con líneas innecesarias. Por ejemplo si en un programa, se necesitaría mostrar un mensaje 100 veces se debería proceder de la siguiente manera:

```

1      System.out.println("Mensaje");
2      System.out.println("Mensaje");
3      System.out.println("Mensaje");
4      System.out.println("Mensaje");
5      System.out.println("Mensaje");
...
96     System.out.println("Mensaje");
97     System.out.println("Mensaje");
98     System.out.println("Mensaje");
99     System.out.println("Mensaje");
100    System.out.println("Mensaje");

```

En vez de utilizar 100 líneas de código para mostrar 100 veces un mensaje se podría hacer algo como:

```

1      Repetir ( 100 veces)
2      {
3          System.out.println("Mensaje");
4      }

```

De ésta manera con solo 4 líneas de código se puede realizar lo mismo que lo que se hacía como 100 líneas de código.

Estructura while

La estructura de repetición while sirve para repetir un segmento de código siempre y cuando se cumpla una condición predefinida por el programador. Para utilizar la misma se debe escribir la palabra reservada **while** y entre llaves el código que se desee repetir.

Ejemplo:

```

1      public class Principal {
2
3          public static void main(String[] args) {

```

```

4
5     int contador = 0;
6
7     while (contador++ < 5)
8     {
9         System.out.println("El contador vale " + contador );
10    }
11
12    } // fin del método main
13 } // fin de la clase Principal

```

El ejemplo anterior, repite el código entre llaves, mientras que el atributo contador sea menor que 5. Por lo tanto se muestra un mensaje 5 veces, el cual irá informando el valor del contador, a medida que este vaya incrementando.



Observación: Si el atributo contador hubiese sido inicializado con el valor 5, la estructura while hubiese verificado que la condición (`contador++ < 5`) sería falsa, por lo que no habría entrado al ciclo, y no se hubiese mostrado ningún mensaje.

Estructura do while

Esta estructura es muy similar a la estructura while, pero como se ha marcado en la observación de la estructura anterior, lo primero que hace el while, es verificar si la condición es verdadera o falsa, y según el resultado de la expresión entrara o no en la ejecución del ciclo. En el caso de la estructura do while, se entra al ciclo sin verificar ninguna condición, y al final del ciclo se realiza la verificación de la condición, por lo que si esta resulta verdadera, volverá a ejecutar el ciclo, y si resulta falsa seguirá adelante en la ejecución del programa. Para utilizarla se debe escribir la palabra reservada **do**, entre llaves se escribe el código que se desea repetir, y luego de la llave de cierre se debe escribir la palabra reservada **while**, indicando la condición de repetición entre paréntesis, finalizando la estructura con un “;”.

Ejemplo:

```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int contador = 5;
6
7          do
8          {
9              System.out.println("El contador vale " + contador );
10         }while(contador++ < 5);
11
12     } // fin del método main
13 } // fin de la clase Principal

```

Este ejemplo mostrará una sola vez el mensaje “El contador vale 5”, y luego, al verificar que la condición de repetición no se cumple, no volverá a repetir el código. Si en vez de usar la estructura do while, se hubiese usado while, no se hubiera entrado al ciclo de repetición y no se hubiese mostrado nada por pantalla ya que al no entrar al ciclo no se ejecuta el código dentro de él.

Estructura for

La estructura o bucle for posee la misma finalidad que la estructura while, solo que en un bucle for se presentan los datos y las condiciones de manera diferente, posibilitando escribir la condición de repetición y la operación que sufre el atributo incluido en la condición, en una sola línea.

Una particularidad que ofrece Java, es declarar la variable utilizada en la condición dentro de la misma línea, por lo que de ésta manera, se pueden evitar problemas, y se evita también que ésta variable no siga cargada en memoria innecesariamente luego de terminar el ciclo for.

Para utilizar este tipo de estructura se debe escribir la palabra reservada **for** y luego entre paréntesis deben indicarse:

- a) El atributo incluido en la condición inicializado
- b) La condición
- c) La operación que sufre el atributo luego de finalizar un ciclo

Estos tres parámetros deben estar separados con “,”.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          for(int contador = 0; contador < 5; contador++){
6              System.out.println("El contador vale " + contador );
7          }
8
9      } // fin del método main
10 } // fin de la clase Principal
```

Este ejemplo es equivalente al ejemplo de la estructura de repetición while, solo que ésta estructura se arma de una manera diferente. Primero se declara el atributo contador y se lo inicializa con un valor de 0, luego se escribe la condición, indicando que repita el código siempre y cuando el valor del atributo contador sea menor a 5, y finalmente se indica que al finalizar cada ciclo, el valor del atributo contador se incremente en 1 (uno).

De esta manera, el programa mostrará un mensaje indicando el valor del contador en cada ciclo 5 (cinco) veces.

Estructuras de repetición anidadas

Al igual que, como se ha visto, las estructuras de control pueden ser anidadas, también se podrán anidar las estructuras de repetición. Eso es muy útil cuando se trabaja con vectores unidimensionales y multidimensionales, también cuando se trabaja con archivos, entre otras utilidades.

Ejemplo

```
1  public class Principal {
2
3      public static void main(String[] args) {
```

```
4
5     for(int i = 0; i < 10; i++){
6         for (int j = 0; j < 10; j++){
7             System.out.println(i + " " + j);
8         } // fin del for j
9     } // fin del for i
10
11 } // fin del método main
12 } // fin de la clase Principal
```

Este algoritmo mostrará los números del 00 al 99.

Instrucciones break y continue

Existen dos instrucciones que sirven para controlar o alterar el flujo de ejecución de un programa. Estas instrucciones son utilizadas en las estructuras de repetición, aunque como ya se ha visto, la palabra clave **break** aparece en la estructura de control switch, y en ese caso sirve para terminar la ejecución del switch.

Instrucción break

La instrucción break sirve para terminar la ejecución de una estructura de repetición cualquiera sea, o bien de la estructura de control switch. La misma sirve para escapar anticipadamente de un bucle o bien para ignorar el resto de una instrucción de control switch.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          for (int i = 0; i < 10; i++){
6
7              if(i == 5) break;
8              System.out.println("Repetición nro " + i);
9
10         } // fin del ciclo for
11     } // fin del método main
12 } // fin de la clase Principal
```

Este programa mostrará en la consola: 0, 1, 2, 3, 4 y luego saldrá del ciclo for

Instrucción continue

La instrucción continue se utiliza en estructuras de repetición, y al ejecutarse se omite el código restante dentro del cuerpo del ciclo para proceder a la siguiente iteración del mismo.

Ejemplo:

```
1  public class Principal {
```

```

2
3     public static void main(String[] args) {
4
5         for (int i = 0; i < 10; i++){
6
7             if(i == 5) continue;
8             System.out.println("Repetición nro " + i);
9
10        } // fin del ciclo for
11    } // fin del método main
12 } // fin de la clase Principal

```

Este programa mostrará en la consola: 0, 1, 2, 3, 4, 6, 7, 8, 9, y luego saldrá del ciclo for

Arreglos

Un arreglo es una estructura que sirve para almacenar datos de un mismo tipo, y mantenerlos organizados y ordenados dentro de esta estructura. Una estructura podría servir para guardar los días de la semana, para guardar un conjunto de números primos, u organizar datos referentes a la aplicación, como por ejemplo el sueldo de 10 empleados.

En casos como estos resulta mucho más cómodo trabajar con estructuras ya que sería más fácil guardar estos datos en una estructura que crear una variable por cada elemento.

Si se declara un atributo por cada sueldo correspondiente a cada empleado se podría ver algo así:

```

float sueldo_empleado_1 = 8000f;
float sueldo_empleado_2 = 9500f;
float sueldo_empleado_3 = 7000f;
float sueldo_empleado_4 = 6500f;
float sueldo_empleado_5 = 8500f;
float sueldo_empleado_6 = 10000f;
float sueldo_empleado_7 = 5500f;
float sueldo_empleado_8 = 9000f;
float sueldo_empleado_9 = 12000f;
float sueldo_empleado_10 = 7300f;

```

En cambio, si estos datos se guardan en una estructura, la forma sería, algo parecido a esto:

float sueldos	8000f	9500f	7000f	6500f	8500f	10000f	5500f	9000f	12000f	7300f
	emp1	emp2	emp3	emp4	emp5	emp6	emp7	emp8	emp9	emp10

Como se puede observar, de esta manera se tienen los datos de una manera más organizada, en donde cada posición de esta estructura, corresponderá al sueldo de un empleado, y en el caso de que se quisieran agregar más datos a la tabla, bastará con agregar más campos a la misma, sin necesidad de seguir definiendo atributos. En el ejemplo anterior, facilita un poco las cosas, pero sin embargo la diferencia no es tan grande, pero si en vez de tener 10 empleados, se tuvieran 100 empleados, se tendrían que crear 100 atributos, y cada vez que se quiera utilizar un atributo se deberá llamarlo, por lo tanto tener en cuenta los 100 nombre asignados a estos atributos, y haría que el programa se sobrecargue con tanto código, y sea poco legible. En cambio con una estructura que contenga 100 campos, y que cada campo corresponda al sueldo de un empleado, simplificaría mucho las cosas.

Declaración y creación de un arreglo

Para declarar un arreglo se debe indicar el tipo que contendrán los datos del mismo, luego escribir corchetes [] para indicar que el atributo que se creará será una estructura de datos y seguido de eso el nombre que se le quiera dar a la estructura.

```
tipo[] nombre;
```

Para declarar el arreglo correspondiente al ejemplo de los sueldos de los empleados, se debe hacer:

```
float[] sueldos;
```

Hecho esto, el arreglo estará declarado, pero hasta el momento no se ha indicado cuantos campos tendrá el mismo, y tampoco se podrá utilizar por el momento, ya que en Java un arreglo es tratado como un objeto, y como todo objeto debe ser creado para su utilización.

Como ya se ha visto, para crear un objeto se debe anteponer la palabra clave **new**, y luego el tipo o clase de la cual se creará este objeto. Por lo tanto, una vez declarado el arreglo de sueldos, para crear el objeto que va a permitir interactuar con ese arreglo se debe hacer:

```
sueldos = new float[100];
```

De esta manera, se crea el objeto para la estructura sueldos, indicándole entre corchetes el tamaño del arreglo, o la cantidad de campos que contendrá el mismo, y luego de esto, se podrá trabajar con el sin problemas.

Uso de un arreglo

Para trabajar con el arreglo se debe escribir el nombre del mismo, y entre corchetes indicar la posición del arreglo a la cual se quiera acceder. Una vez hecho esto, se podrá guardar algún valor en dicha posición o bien leer el valor que tiene, si éste existiera.

```
sueldos[50] = 5000f;
```

```
System.out.println("El sueldo del empleado 51 es " + sueldos[50]);
```



Observación: La primer posición de un arreglo es la posición 0, y en el ejemplo de los sueldos, ésta posición equivaldrá al sueldo del empleado número 1, por lo tanto, la posición número 50 en el arreglo de sueldos, corresponde al sueldo del empleado número 51. Y por lo tanto, si el arreglo tiene 100 campos, o mejor dicho tiene una longitud de 100, el primer valor corresponde a la posición 0, y el último valor corresponde a la posición 99.

Inicialización de un arreglo

Existe una forma para declarar e inicializar los valores de un arreglo siguiendo un método diferente al visto anteriormente. Para esto se debe declarar de igual manera en que se ha explicado, pero en vez de crear el objeto mediante la palabra **new**, se debe poner un signo "=" luego de la declaración y entre llaves indicar los datos que contendrá el mismo, separados por comas.

```
String[] diasLaborales =  
{ "Lunes", "Martes", "Miércoles", "Jueves", "Viernes" };
```

Al declarar el arreglo e inicializarlo de esta forma, se podrá trabajar con él sin necesidad de crear al objeto de manera convencional, ya que Java lo hará automáticamente, y tampoco se necesitará indicarle el tamaño, pues el mismo será, en el caso del ejemplo anterior, cinco.

Atributo length

Como un arreglo es un objeto, es de esperar que puedan existir métodos y/o atributos a los cuales poder acceder. Un atributo muy útil es el atributo **length**, mediante él, se puede saber el tamaño de un arreglo, y esto es muy útil a la hora de recorrer un arreglo, ya que de ésta manera se generaliza en código.


Recorrer los elementos de un arreglo

Para recorrer todos los elementos que pueda contener un array, se pueden utilizar estructuras de repetición, lo cual simplificará el código a escribir, y lo hará mucho más general.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          float[] sueldos = {4500f, 7000f, 6200f, 8100f, 7500f};
6
7          for(int i = 0; i < sueldos.length; i++){
8              System.out.println("Sueldo " + (i+1) + ": $ " + sueldos[i]);
9          } // fin del for
10
11      } // fin del método main
12 } // fin de la clase Principal
```

Al ejecutar el programa, se podrá ver por consola lo siguiente:



The screenshot shows a Java IDE window with tabs for Problems, Javadoc, Declaration, and Console. The Console tab is active, displaying the output of the program. The output is as follows:

```
<terminated> Principal [Java Application] C:\Program Files\Java\
Sueldo 1: $ 4500.0
Sueldo 2: $ 7000.0
Sueldo 3: $ 6200.0
Sueldo 4: $ 8100.0
Sueldo 5: $ 7500.0
```

En éste ejemplo se puede visualizar la utilidad del atributo **length**, ya que si se necesitaran agregar 5 sueldos más al arreglo, no habría necesidad de modificar la condición del **for**.

Estructura for mejorada

Existe otra manera de realizar un ciclo **for**, la cual puede ser útil en ciertos casos que requieran recorrer un array. Esta forma es más simplificada y permite recorrer una estructura de datos o vector, sin necesidad de usar un contador. La sintaxis de esta estructura es:

```
for (tipo atributo : arreglo) {
    instrucción;
}
```


El primer parámetro del ciclo for, será en donde se irán guardando los valores de la posición del arreglo en la que se encuentre, y ésta servirá para mostrarlos, o realizar operaciones, y el segundo parámetro corresponderá al arreglo que se quiere recorrer.

Ejemplo:

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          float[] sueldos = {4500f, 7000f, 6200f, 8100f, 7500f};
6
7          for(float sueldo : sueldos){
8              System.out.println("Sueldo: $ " + sueldo);
9          } // fin del for especial
10
11     } // fin del método main
12 } // fin de la clase Principal
```

Como ya se ha dicho, esta estructura de for mejorada, tendrá sus ventajas y sus desventajas, y se deberá utilizar según las necesidades del problema a resolver.

Las ventajas que puede presentar son, por una parte la simplicidad de código, por otra parte ésta estructura evitará salirse de los límites del array, esto quiere decir, que si un arreglo tiene una longitud de 10, y en un for convencional se pone erróneamente como condición $i < 11$, suponiendo que en el cuerpo del ciclo se encontrara una instrucción como `System.out.println(array[i]);` cuando i sea igual a 10, Java intentará acceder a la posición 10 del vector, pero al ser la longitud 10, esto quiere decir que sus posiciones irán del 0 al 9, por lo tanto la posición 10 no existirá, y Java terminará la ejecución del programa informando un error en el que indicará que se han excedido los límites del arreglo.

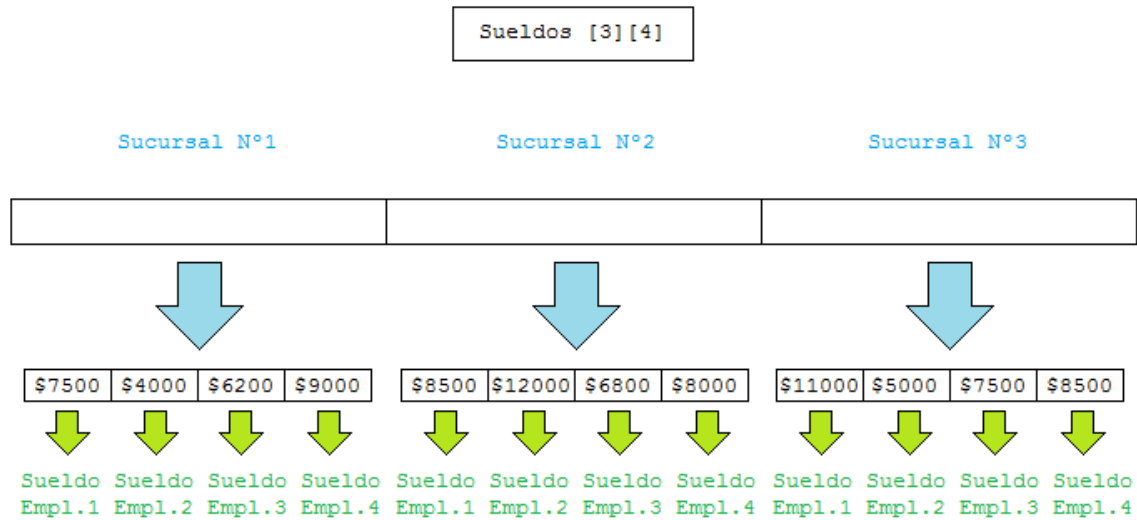
Una desventaja de éste tipo de estructura es que, como se puede observar, la misma puede servir para recorrer un array y mostrar valores, o extraer los mismos para algún tipo de operación, pero al no existir un contador, no se podrán modificar los valores dentro de la estructura.

Por último se puede marcar, que al no tener un contador, no se podrá controlar el flujo de ejecución mediante instrucciones `break` o `continue`, ya que no se le puede indicar cuando realizar estas acciones, esto puede resultar ventajoso o no según el caso en que se presente.

Arreglos multidimensionales

Hasta el momento se han visto arreglos que son de una sola dimensión, ya que como se puede apreciar, asemeja a una lista, o a una fila con cierta cantidad de columnas. Pero se pueden trabajar con múltiples dimensiones, lo más normal es trabajar con arreglos de dos dimensiones conocidos como matrices. Básicamente el funcionamiento de estos es el mismo que el de un arreglo unidimensional. Para definir, crear, y usar un arreglo multidimensional, se procede de la misma manera que con los unidimensionales, solo que se le agregan los corchetes correspondientes a la cantidad de dimensiones que tenga el mismo.

Una matriz podría verse como un arreglo unidimensional que en cada campo contendrá otro arreglo multidimensional, y un ejemplo para utilizar matrices sería administrar los sueldos de los empleados de muchas sucursales.

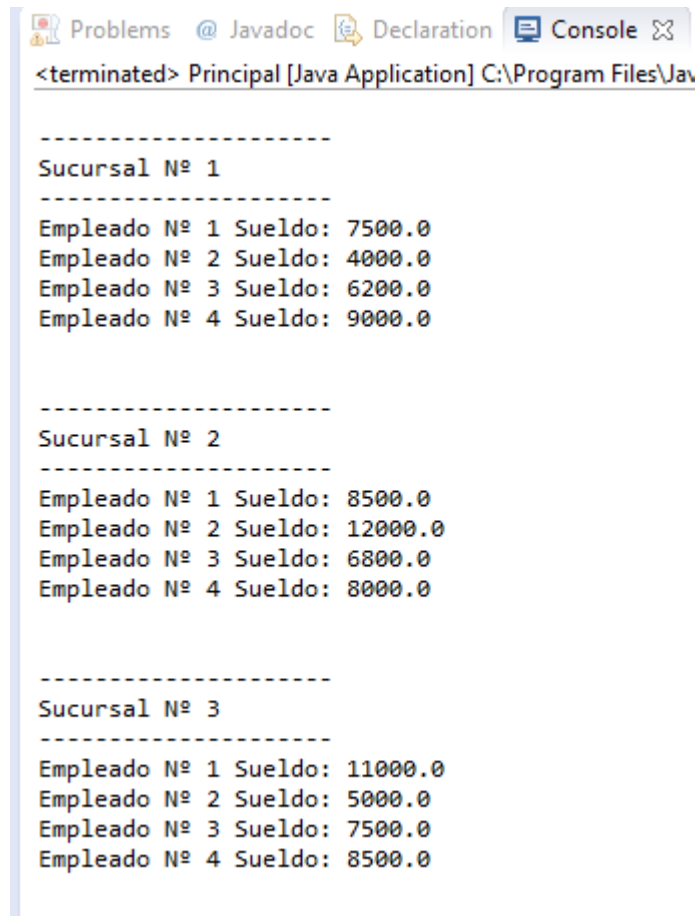


```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          float[][] sueldos = { {7500,4000,6200,9000},
6                                {8500,12000,6800,8000},
7                                {11000,5000,7500,8500}};
8
9          for(int i = 0; i < sueldos.length; i++){
10
11              System.out.println();
12              System.out.println("-----");
13              System.out.println("Sucursal N° " + (i+1));
14              System.out.println("-----");
15
16              for(int j = 0; j < sueldos[0].length; j++){
17
18                  System.out.println("Empleado N° " + (j+1) +
19                                     " Sueldo: " + sueldos[i][j]);
20              } // Fin del for j
21
22              System.out.println();
23
24          } // Fin del for i
25
26      } // Fin del método main
27  } // Fin de la clase Principal

```

Este programa mostrará lo siguiente en la consola:



```
<terminated> Principal [Java Application] C:\Program Files\Jav  
  
-----  
Sucursal N° 1  
-----  
Empleado N° 1 Sueldo: 7500.0  
Empleado N° 2 Sueldo: 4000.0  
Empleado N° 3 Sueldo: 6200.0  
Empleado N° 4 Sueldo: 9000.0  
  
-----  
Sucursal N° 2  
-----  
Empleado N° 1 Sueldo: 8500.0  
Empleado N° 2 Sueldo: 12000.0  
Empleado N° 3 Sueldo: 6800.0  
Empleado N° 4 Sueldo: 8000.0  
  
-----  
Sucursal N° 3  
-----  
Empleado N° 1 Sueldo: 11000.0  
Empleado N° 2 Sueldo: 5000.0  
Empleado N° 3 Sueldo: 7500.0  
Empleado N° 4 Sueldo: 8500.0
```

Ámbito de las variables

Cada atributo o variable dentro del programa tiene un ámbito en donde tendrá validez y podrá ser utilizada. El ámbito de cada variable estará definida entre llaves, por lo cual cuando se crea un atributo, éste tendrá validez dentro de la estructura en donde se lo haya declarado. Luego de una llave de cierre, no solo no tendrá validez, sino que el atributo o variable será destruido de la memoria. Si un atributo se encuentra en dos estructuras diferentes, las cuales no se relacionan entre sí, se podrán declarar atributos con el mismo nombre.

Ejemplo:

```

1 public class Principal {
2
3     int atributo0 = 55;
4
5     public static void main(String[] args) {
6
7         int atributo1 = 1; // válido dentro de todo el método main
8
9         System.out.println(atributo0); // Correcto
10
11         if(atributo1 > 0) { // Correcto
12             int atributo2 = 30; // Se crea atributo2
13             System.out.println(atributo1); // Muestra 1
14             System.out.println(atributo2); // Muestra 30
15         } // Se elimina atributo2
16
17         if(atributo2 > 0) { // Error: atributo2 no existe
18             System.out.println(atributo2); // Error: atributo2 no existe
19         }
20
21         if(atributo0 > 0){ // Correcto
22             int atributo2 = -50; // Se crea atributo2
23             int atributo3 = 10; // Se crea atributo3
24             if(atributo3 < 0){ // Correcto
25                 int atributo3 = 20; // Error: redefinición de atributo3
26                 int atributo4 = 10; // Se crea atributo4
27
28                 System.out.println(atributo1); // Muestra 1
29                 System.out.println(atributo2); // Muestra -50
30                 System.out.println(atributo3); // Muestra 10
31                 System.out.println(atributo4); // Muestra 10
32                 System.out.println(atributo0); // Muestra 55
33             } // Se elimina atributo4
34         } // Se elimina atributo2 y atributo3
35
36         for(int i = 0; i < atributo1; i++){ // Se crea i
37             int atributo0 = 155; // Error: redefinición de atributo0
38             int atributo3 = 10; // Se crea atributo3
39             System.out.println(atributo3); // Correcto
40
41             for(int j = 0; j < atributo0; j++){ // Se crea j
42                 int atributo1 = 5; // Error: redefinición de atributo1
43                 int atributo2 = -15; // Correcto
44                 System.out.println(atributo1); // Muestra 1
45                 System.out.println(atributo2); // Muestra -15
46                 System.out.println(atributo3); // Muestra 10
47                 System.out.println(atributo4); // Error: atributo4 no existe
48                 System.out.println(i); // Muestra i
49             } // Se elimina j y atributo2
50
51             System.out.println(j); // Error: j no existe
52         } // Se elimina atributo3
53     } // Se elimina atributo1
54 } // Se elimina atributo0
55
56
57
58 }

```

Manejo de excepciones

Una excepción en Java se produce cuando ocurre un problema durante la ejecución de un programa, se llaman así porque no son habituales y presentan excepciones a una regla. El manejo de excepciones permite que, cuando ocurra un error, se lo pueda capturar, y resolver el problema, sin parar la ejecución de la aplicación. Estos errores pueden surgir, por ejemplo, al dividir un número entero por cero, o al sobrepasar los límites de un array.

Cuando se produce un error, la máquina virtual de Java lanza una excepción, que contendrá datos acerca del error, y mediante esos datos, se podrá armar un algoritmo para poder solucionar el error ocurrido, y de esta manera evitar que el programa termine su ejecución.

Para controlar estos errores se dispone de varios bloques, los cuales se detallan a continuación:

Bloque try:

El bloque try, sirve para encerrar la porción de código que pueda generar un error. La palabra try, en inglés, quiere decir intentar, por lo que, al encerrar una porción de código entre llaves mediante el bloque try, se está indicando que se debe intentar ejecutar esa porción de código, y en el caso de que contenga un error, poder capturarlo y controlarlo.

Ejemplo:

```
1  try {  
2  
3      // Código que pueda generar errores  
4  
5  }
```

Bloque catch:

El bloque anterior, intenta ejecutar una porción de código que pueda llegar a generar algún error, y en el caso que éste surgiera, el bloque try no serviría para nada si no se le añade algún tipo de código que se encargue de capturar el error generado. Para esto existe el bloque catch, el cual, se ejecutará en el caso de que se produzca un error, y dentro de éste se podrá escribir el código correspondiente para resolver el error o bien informar del mismo.

Ejemplo:

```
1  catch (Exception e) {  
2  
3      // Código donde se trata el problema  
4  
5  }
```

Bloque finally:

Este bloque sirve para indicar a Java, que ejecute una porción de código, indiferentemente si produce o no una excepción.

Ejemplo:

```
1  finally {  
2  
3      // Código que se ejecutará se produzca o no una excepción  
4  
5  }
```

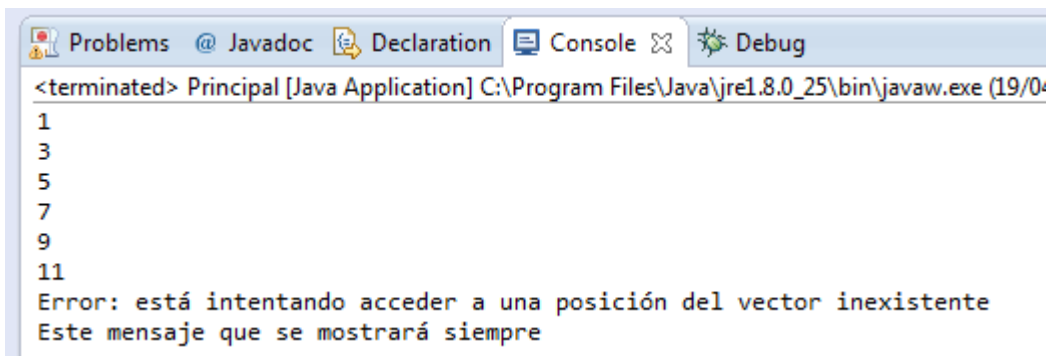
Ejemplo:

```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int[] vector = {1,3,5,7,9,11};
6
7          try {
8
9              for(int i = 0; i <= vector.length; i++){
10                 System.out.println(vector[i]);
11             }
12
13         } catch(Exception e){
14
15             System.out.println("Error: está intentando acceder a una "
16                               + "posición del vector inexistente");
17
18         } finally {
19
20             System.out.println("Este mensaje se mostrará siempre");
21
22         }
23     } // Fin del método main
24 } // Fin de la clase Principal

```

Este programa arrojará como resultado lo siguiente:



```

<terminated> Principal [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (19/04/2016 10:10:10)
1
3
5
7
9
11
Error: está intentando acceder a una posición del vector inexistente
Este mensaje que se mostrará siempre

```

Para el caso expuesto el ejemplo sirve perfectamente, pero si se tuviera un código como el siguiente:

```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int[] vector = {1,3,5,7,9,11};
6          int numero = 10;
7
8          try {
9
10             for(int i = 0; i <= vector.length; i++){
11                 System.out.println(vector[i]);
12             }
13
14             numero /= 0;
15

```

```

16     } catch(Exception e){
17
18         System.out.println("Error: está intentando acceder a una "
19                             + "posición del vector inexistente");
20
21     } finally {
22
23         System.out.println("Este mensaje se mostrará siempre");
24
25     }
26
27 } // Fin del método main
28 } // Fin de la clase Principal

```

En este caso, el resultado será el mismo que en el primer ejemplo, pero ésta vez el problema no es que se está intentando acceder a una posición inexistente del vector, sino que se está intentando dividir un número entero sobre cero, lo cual produce error.

Por lo tanto aquí no sirve la generalización al momento de capturar un error, para esto se pueden implementar varias soluciones.

La solución más simple es mostrar un mensaje de error general, e informar algo como, "**Se ha producido un error**", pero esta forma es poco conveniente ya que, se pueden producir muchos errores, y con un mensaje de este tipo no se sabe que error se generó o que es lo que lo pudo haber producido.

Otra solución un poco más conveniente que la anterior es mostrar un mensaje de error mediante el objeto de tipo Exception, para mostrar un mensaje de error mediante éste objeto se debe escribir el nombre del mismo, y llamar al método toString. De ésta manera Java indicará cual fue el error o la excepción que se haya capturado.

La tercera solución posible, es mediante varios bloques catch, capturar cada error que se pueda producir individualmente, y mostrar un mensaje según sea el error capturado.

Ejemplo:

```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int[] vector = {1,3,5,7,9,11};
6          int numero = 10;
7
8          try {
9
10             for(int i = 0; i <= vector.length; i++){
11                 System.out.println(vector[i]);
12             }
13
14             numero /= 0;
15
16         } catch(ArrayIndexOutOfBoundsException e){
17
18             System.out.println("Error: " + e.toString());
19
20             System.out.println("Se está intentando acceder a una "
21                                 + "posición del vector inexistente");
22
23         } catch(ArithmeticException e){
24
25             System.out.println("Error: " + e.toString());
26             System.out.println("Se está intentando dividir un entero por"
27                                 + " cero");
28
29         }
30     }
31 }

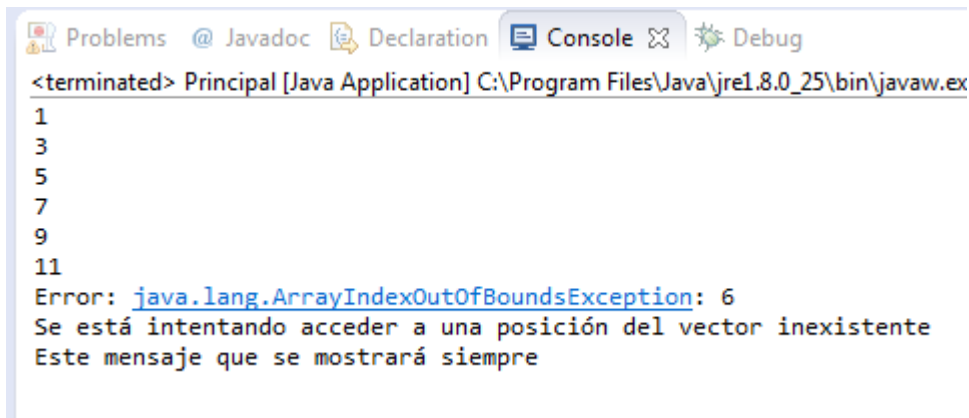
```

```

21     } finally {
22
23         System.out.println("Este mensaje se mostrará siempre");
24
25     }
26
27 } // Fin del método main
28 } // Fin de la clase Principal

```

Este programa mostrará como resultado lo siguiente:



```

<terminated> Principal [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe
1
3
5
7
9
11
Error: java.lang.ArrayIndexOutOfBoundsException: 6
Se está intentando acceder a una posición del vector inexistente
Este mensaje que se mostrará siempre

```

Como se puede observar, solo muestra un error, y este será el primer error que se genere, ya que cuando pasa esto, Java no seguirá ejecutando código del bloque try, y entrará directamente al catch correspondiente de la excepción lanzada.

También puede verse que, a diferencia de los ejemplos anteriores, dentro de los parámetros del bloque catch, no se indica `Exception` e, sino que el tipo será la excepción que pudiera generarse como, `ArrayIndexOutOfBoundsException` y `ArithmeticException`.

Formas para determinar el tipo de excepción lanzada

Entonces, ¿Cómo saber cuál será el tipo de la excepción lanzada?

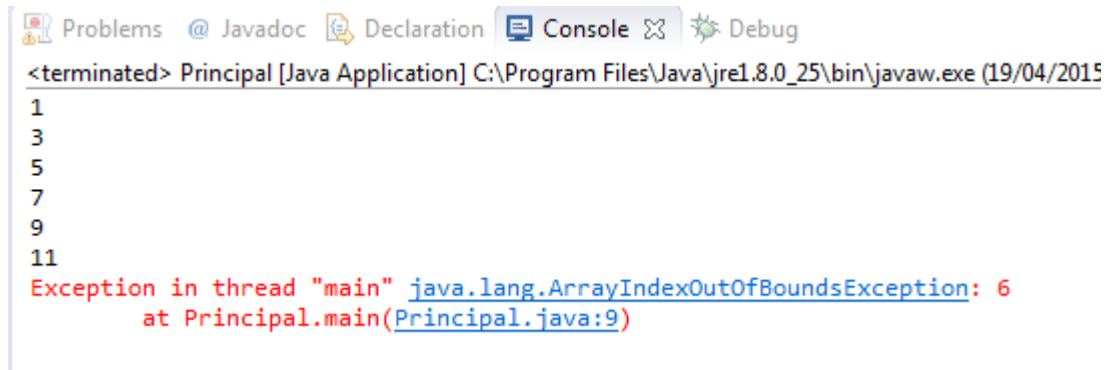
Para esto se puede proceder de diferentes maneras, pero básicamente las dos formas consisten en forzar que se produzca el error y que de ésta manera Java muestre por consola cual es el tipo de excepción. Por ejemplo, en el caso de que se sobrepasen los límites de un vector, en el catch se deberá indicar `ArrayIndexOutOfBoundsException` e, para saber que la excepción es de tipo `ArrayIndexOutOfBoundsException` se puede proceder de la siguiente manera:

```

1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int[] vector = {1,3,5,7,9,11};
6
7
8          for(int i = 0; i <= vector.length; i++){
9              System.out.println(vector[i]);
10         }
11
12     } // Fin del método main
13 } // Fin de la clase Principal

```


Al hacer esto, Java terminará la ejecución del programa y mostrará por consola lo siguiente:



The screenshot shows the IDE's console window with the following text:

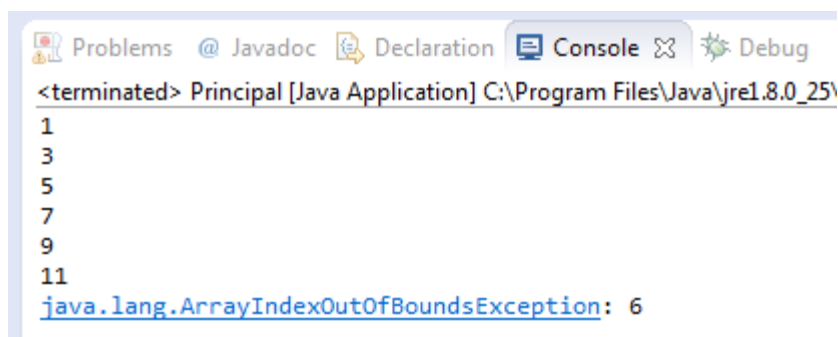
```
<terminated> Principal [Java Application] C:\Program Files\Java\jre1.8.0_25\bin\javaw.exe (19/04/2015
1
3
5
7
9
11
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 6
    at Principal.main(Principal.java:9)
```

Como se puede apreciar, Java indica el nombre del tipo de la excepción, que efectivamente es `ArrayIndexOutOfBoundsException`. De esta manera se sabrá qué tipo poner al armar el bloque `catch`.

La otra manera es hacer un `try catch`, indicando como parámetro del `catch`, que la excepción será de tipo `Exception`, y mediante el objeto llamar al método `toString`, para mostrar la excepción generada por consola.

```
1  public class Principal {
2
3      public static void main(String[] args) {
4
5          int[] vector = {1,3,5,7,9,11};
6
7          try {
8
9              for(int i = 0; i <= vector.length; i++){
10                 System.out.println(vector[i]);
11             }
12
13             catch(Exception e){
14
15                 System.out.println(e);
16             }
17
18         } // Fin del método main
19     } // Fin de la clase Principal
```

El resultado será el siguiente:



The screenshot shows the IDE's console window with the following text:

```
<terminated> Principal [Java Application] C:\Program Files\Java\jre1.8.0_25\
1
3
5
7
9
11
java.lang.ArrayIndexOutOfBoundsException: 6
```

Nuevamente se observa como Java indica el tipo de excepción que se genera.