4. Uso de estructuras de control.

Autor	Ⓧ Xerach Casanova
Clase	Programación
Fecha	@Dec 27, 2020 11:04 PM

1. Introducción

La gran mayoría de lenguajes de programación poseen estructuras que permiten controlar el flujo de la información de los programas. Estas estructuras suelen ser comunes en todos y solo cambia su sintaxis.

Los tipos de estructura de programación que se emplean para controlar el flujo de datos son:

- **Secuencia:** compuestas por 0, 1 o N sentencias ejecutadas en el orden que se han escrito. Es la estructura más sencilla, sobre la que se construye el resto de estructuras.
- **Selección:** es un tipo de sentencia especial de decisión y de secuencias de instrucciones asociadas a ella. Según la evaluación de la sentencia (suele ser verdadero o falso), se ejecuta una secuencia de instrucciones u otras. Pueden ser simples, compuestas o múltiples.

- **Iteración:** sentencia especial de decisión y secuencia de instrucciones que se repiten según el resultado de la evaluación de la sentencia de decisión. La secuencia de instrucciones alojada dentro se ejecuta repetidamente si la sentencia de decisión arroja un valor correcto.
- **Sentencias de salto:** no son recomendables pero es necesario conocerlas.
- **Manejo de excepciones en Java:** para gestionar errores y situaciones especiales.

2. Sentencias y bloques

- Es necesario colocar las instrucciones o sentencias una detrás de otras en el orden exacto que deben ejecutarse.
- Cada sentencia debe estar escrita en una sola línea para que el código sea más legible y la localización de errores sea sencilla y rápida. Los errores en herramientas de programación se asocian a número de línea.
- Las sentencias pueden ocupar varias líneas del programa si son muy largas, pero siempre terminan en punto y coma.
- En algunas ocasiones, sobre todo en estructuras de control, no se utiliza punto y coma al final de la cabecera de la estructura.
- Existen sentencias nulas, solo contienen punto y coma y no hacen nada.
- Un bloque de sentencias es un conjunto de sentencias que se encierra en llaves y se ejecutan como una única orden. Agrupan sentencias y clarifican el código. Los bloques de sentencias se utilizan en casi todas las estructuras de control de flujo, clases, métodos, etc...
- En un bloque de sentencias, las instrucciones se deben colocar en un orden exacto o no, dependiendo de la situación.

Debemos tener en cuenta, a la hora de programar las siguientes normas en java:

- Declarar cada variable antes de utilizarla,
- Las variables declaradas dentro de un método, no se inicializan con un valor. Así que antes de trabajar con su valor, hay que dárselo. En el caso de las variables declaradas fuera de métodos se les da un valor inicial si el programador no se las da:
  - Numéricas: 0
  - Objetos: Null.
  - Booleanas: false
  - Char: ''.

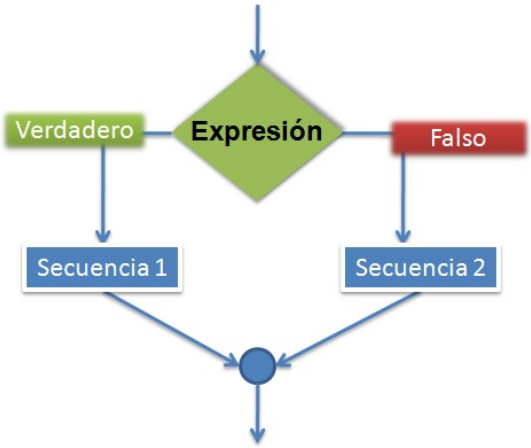
3. Estructuras de selección

Las estructuras de selección constan de una sentencia especial de decisión y un conjunto de secuencias de instrucciones. La sentencia de decisión es evaluada y devuelve valor verdadero o falso. En función de ese valor se ejecuta una secuencia de instrucciones u otras.

En C, verdadero o falso se representa mediante literal entero (0 para falso, 1 para true). En java se toman valores de true o false. Ojo: existe la clase Boolean y el tipo primitivo boolean.

La estructura de selección se divide en:

- simple: if.
- compuesta: if - else.
- basadas en el operador condicional. (condición) ? : valor verdadero : valor falso
- múltiples: switch.



3.1. Estructura if / if - else

Es una estructura condicional. En función del resultado se ejecuta un a sentencia o bloque de estas.

Estructura if simple. Si la evaluación de la expresión lógica ofrece un resultado verdadero se ejecuta la sentencia o sentencias. Si es falso, no se ejecuta ninguna instrucción.

```
if (expresión-lógica) //al ser una sola instrucción, no son necesarias las llaves

sentencia1;

if (expresión-lógica)
{

   sentencia1;

   sentencia2;

   …;

   sentenciaN;

}
```

Estructura if de doble alternativa. Si la evaluación de la expresión lógica ofrece resultado verdadero se ejecuta la sentencia o sentencias del bloque if. Si es falso se ejecuta la sentencia del bloque else

```
if (expresión-lógica) //al ser una sola instrucción, no son necesarias las llaves

    sentencia1;

else

    sentencia2;
if (expresión-lógica)
```

```
{
   sentencia1;
   …;
   sentenciaN;
}
else
{
   sentencia1;
   …;
   sentenciaN;
}
```

La sentencia else no es obligatoria, pero sí es necesaria cuando hay que llevar a cabo alguna acción en caso de que la expresión lógica no se cumpla.

Las condiciones if e if - else, pueden anidarse, dentro de un bloque de sentencias if/if-else se puede incluirse otro if/if-else. Si el nivel de anidamiento es demasiado profundo puede provocar problemas de eficiencia y legibilidad y habría que plantearse elegir otro tipo de estructuras más adecuadas.

Cuando usamos anidamiento debemos poner especial atención en saber que estructura else está asociada a un cada if.

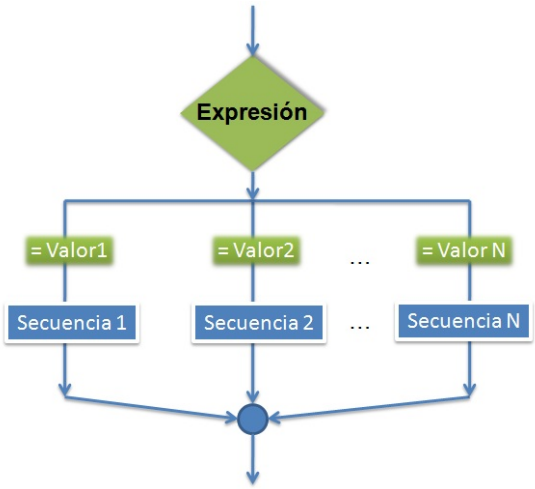
3.2. Estructura Switch

Es un tipo de selección múltiple, ideal si nuestro programa debe elegir entre más de dos alternativas.

```
switch (expresion) {
 case valor1:
    sentencia1_1;
    sentencia1_2;
    ….
    break;
     ….
    ….
case valorN:
    sentenciaN_1;
    sentenciaN_2;
     ….
```

```
    break;
default:
    sentencias-default;
}
```

- La expresión debe ser char, byte, short o int. Las constantes de cada case debe ser del mismo tipo o compatible.
- La expresión va entre paréntesis.
- Cada case lleva asociado un valor y finaliza con dos puntos.
- El bloque de sentencias asociado a la cláusula default puede finalizar con la sentencia de ruptura break o no.



4. Estructuras de repetición

La función de estas estructuras es repetir la ejecución de una serie de instrucciones teniendo en cuenta una condición.

En Java existen cuatro tipos de estructuras de repetición, bucles o estructuras iterativas:

- Bucle for (repite para) - Controlado por contador.
- Bucle for/in (repite para cada) - Controlado por contador.
- Bucle While (repite mientras) - Controlado por sucesos.
- Bucle Do While (repite hasta) - Controlado por sucesos.

El uso de estos cuatro bucles depende de:

- Si sabemos cuantas veces necesitamos repetir un conjunto de instrucciones.
- Si hemos de repetir un conjunto de instrucciones si una condición satisface un conjunto de valores.
- Si sabemos hasta cuando debemos repetir un conjunto de instrucciones.
- Si sabemos si hemos de estar repitiendo un conjunto de instrucciones mientras se cumpla una condición.

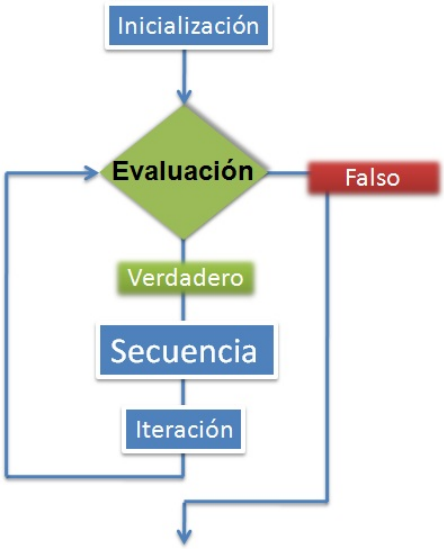
Algunos de estos bucles son equivalentes entre sí y se puede solucionar el mismo problema utilizando distintos tipos de bucles.

4.1. Estructura for

Bucle controlado por contador. Tiene las siguientes características:

- Se ejecuta un número determinado de veces.
- Utiliza una variable contadora que controla las iteraciones.

Existen tres operaciones que se llevan a cabo en los bucles for, que se ven en el código siguiente...




```
for (inicialización; condición; iteración)

   sentencia;  //Con una sola instrucción no es necesario utilizar llaves
for (inicialización; condición; iteración)
{

   sentencia1;

   sentencia2;

   ...

   sentenciaN;

}
```

- **Inicialización** de la variable que se encarga de controlar el final del bucle
- **Condición.** Una expresión que evaluará la variable de control, mientras sea falsa, se repetirá el cuerpo del bucle. Cuando se cumpla, terminará la ejecución.
- **Iteración.** Indica la manera en que la variable de control va cambiado en cada iteración (incremento o decremento y no solo de uno en uno).

4.2. Estructura for/in

Es una mejora incorporada en la versión 5.0 de java.

Permite realizar recorridos sobre arrays y colecciones de objetos. Se les llama también bucle for mejorado o bucle foreach.

```
for (declaración: expresión) {

   sentencia1;

   …
```

```
    sentenciaN;

}
```

- Donde expresión es un array o colección de objetos.
- Donde declaración es una declaración de una variable cuyo tipo sea compatible con la expresión. Normalmente será el tipo y el nombre de la variable a declarar.

Para cada elemento de la expresión, se guarda el elemento en la variable declarada y realiza las instrucciones del bucle. Después, en cada una de las iteraciones del bucle tendremos la variable declarada en el elemento actual de la expresión. El bucle acabará cuando se termine de recorrer el array o la colección de objetos.

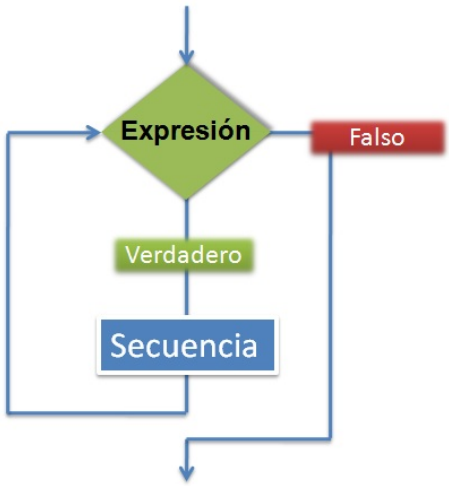
Permiten al programador despreocuparse del número de veces que se ha de iterar, pero no sabremos en que iteración estamos, salvo que añadamos una variable contadora extra.

4.2. Estructura while

Es una estructura de repetición controlada por sucesos.

Su característica principal es que las instrucciones contenidas dentro de él se ejecutan al menos una vez, si la condición inicial a evaluar es verdadera. Si la evaluación inicial es falsa el cuerpo del bucle no se ejecuta.

Dentro del bucle debe realizarse alguna acción que modifique la condición que controla su ejecución, para no ejecutar un bucle infinito.




```
while (condición)

sentencia;
while (condición) {

   sentencia1;

   …

   sentenciaN;

}
```

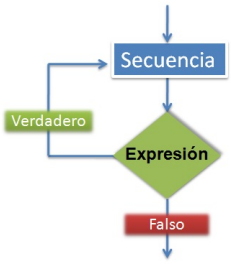
La condición del bucle siempre se evalúa al principio, por tanto las instrucciones contenidas pueden no llegarse a ejecutar nunca en caso de que sea falsa.

4.4. Estructura do-while

Funciona exactamente igual que el while, pero la característica fundamental de esta estructura es que las instrucciones contenidas se ejecutarán al menos una vez y, a partir de ahí, se repetirá su ejecución hasta que la condición sea verdadera. Sigue siendo imprescindible realizar alguna acción que modifique la condición que controla su ejecución para no entrar en un bucle infinito.

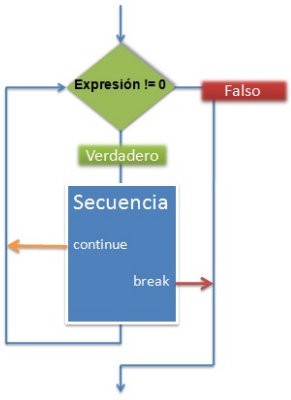
```
do
   sentencia; //Con una sola sentencia no son necesarias las llaves

while (condición);

do
{

   sentencia1;

   …

   sentenciaN;

}
while (condición);
```

La condición siempre se evaluará después de una primera ejecución del bucle, por lo que siempre se ejecutarán las instrucciones contenidas en él, al menos una vez.



5. Estructuras de salto

Se desaconseja su uso porque pueden provocar mala estructuración de código e incremento de dificultad de mantenimiento. Java incorpora ciertas sentencias que es necesario conocer por ser útiles en algunas partes del programa: break, continue y return.



5.1. Sentencias break y continue

Instrucciones que permiten modificar el comportamiento de otras estructuras o sentencias de control al estar incluidas en algún punto de la secuencia.

La sentencia break incide en estructuras switch, while, for y do-while:

- Apareciendo dentro de una secuencia de instrucciones , la estructura se termina inmediatamente.
- Si aparece dentro de un bucle anidado finaliza solo la sentencia de la iteración más interna.

```
public class sentencia_break {
    public static void main(String[] args) {
        // Declaración de variables
        int contador;


        //Procesamiento y salida de información

        for (contador=1;contador<=10;contador++)
        {
            if (contador==7)
                break;
            System.out.println ("Valor: " + contador);
        }
         System.out.println ("Fin del programa");
        /* El bucle sólo se ejecutará en 6 ocasiones, ya que cuando
         * la variable contador sea igual a 7 encontraremos un break que
         * romperá el flujo del bucle, transfiriéndonos a la sentencia que
         * imprime el mensaje de Fin del programa.
         */
    }
```

La sentencia continue incide sobre while, for y do-while.

- Si aparece una sentencia continue dentro de las sentencias, dicha sentencia dará por terminada la iteración actual y ejecutará una nueva.
- Si aparece en el interior de un bucle anidado solo afectará a la iteración más interna.

```
 * Uso de la sentencia continue
 */
public class sentencia_continue {
    public static void main(String[] args) {
        // Declaración de variables
        int contador;

        System.out.println ("Imprimiendo los números pares que hay del 1 al 10... ");
        //Procesamiento y salida de información

        for (contador=1;contador<=10;contador++)
        {
            if (contador % 2 != 0) continue;
            System.out.print(contador + " ");
        }
         System.out.println ("\nFin del programa");
        /* Las iteraciones del bucle que generarán la impresión de cada uno
         * de los números pares, serán aquellas en las que el resultado de
         * calcular el resto de la división entre 2 de cada valor de la variable
         * contador, sea igual a 0.
         */
    }

}

```

5.2. Sentencia return

Modifican la ejecución de un método y se puede utilizar de dos formas:

- Para terminar la ejecución del método donde esté escrita, transfiriendo el control al punto desde donde se hizo la llamada.
- Para devolver o retornar un valor que se incluye junto a return.

Es posible utilizar return en cualquier punto del método, pero suelen aparecer al final de un método, además es lo más recomendable.

6. Excepciones

Las excepciones son errores no sintácticos, que se generan en tiempo de ejecución. Se deben manejar este tipo de excepciones adecuadamente. El manejo de ellos radica en:

- Que el código que se encarga de manejar errores es perfectamente identificable. Además puede estar separado del código de la aplicación.
- Que Java tiene una gran cantidad de errores estándar asociados a multitud de fallos comunes: divisiones por cero, fallo de entrada de datos, etc. y podemos gestionarlas de manera específica.

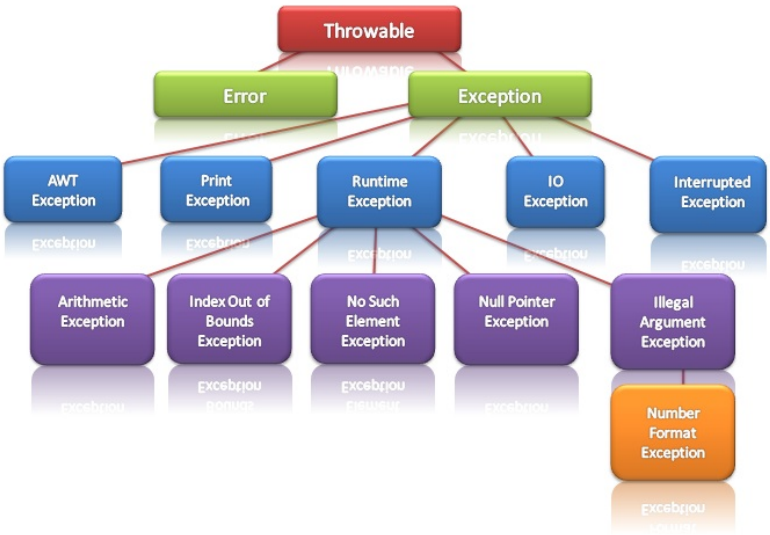
En Java se pueden preparar fragmentos de código susceptibles de provocar errores de ejecución, para ser lanzados hacia otras zonas creadas previamente para tratar dichas excepciones (throw). Si estas excepciones no se tratan, el programa probablemente se detendrá.

Las excepciones están representadas por clases. El paquete java.lang.Exception y sus subpaquetes contienen todos los tipos de excepciones. Todas derivan de la clase Throwable, como por ejemplo Error y Exception.

Cuando se produce un Error, Java genera un objeto asociado a esa excepción, este objeto es de la clase Exception o alguna de sus herederas, el cual se pasa al código que lo maneja. Este código puede manipular las propiedades del objeto Exception.

También podemos lanzar nuestras propias excepciones, derivadas de la clase Exception. Estas clases derivadas se ubican en dos grupos:

- Excepciones en tiempo de ejecución. Cuando el programador no ha tenido cuidado al escribir su código.
- Excepciones que indican que ha sucedido algo inesperado o fuera de control.



6.2. Capturar una excepción

Para ello se emplea la estructura de captura de excepciones try - catch - finally.

Se declaran bloques de código donde es posible que ocurra una excepción con try. Con catch capturamos las excepciones y las manejamos.

```
try {

   código que puede generar excepciones;

} catch (Tipo_excepcion_1 objeto_excepcion) {

   Manejo de excepción de Tipo_excepcion_1;

} catch (Tipo_excepcion_2 objeto_excepcion) {

   Manejo de excepción de Tipo_excepcion_2;

}

...

finally {

   instrucciones que se ejecutan siempre

}
```

La instrucción finally es opcional y solo puede aparecer una vez.

Cada catch maneja un tipo de excepción. Cuando se produce una excepción, se busca el catch que posea el manejador adecuado y será el que utilice el mismo tipo de excepción producida. Al ser la clase Exception superclase de todas, se pueden dar problemas.

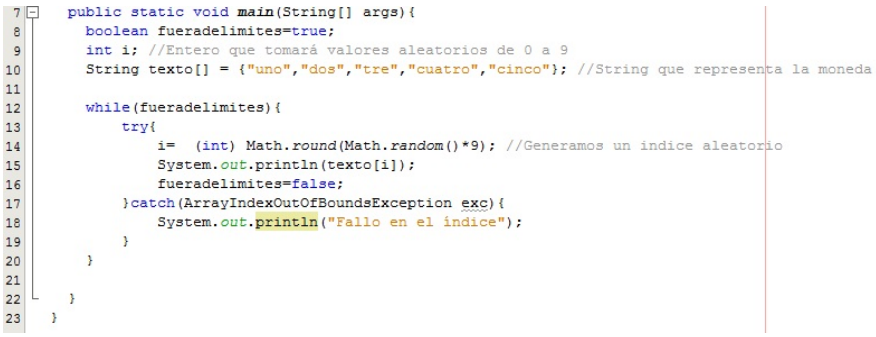
El último catch debe ser el que capture excepciones genéricas y los primeros deben ser más específicos. Si se van a tratar todas las excepciones, entonces basta con un solo catch que capture objetos.

6.2. El manejo de excepciones

Se pueden tratar de dos formas:

- **Interrupción.** Se asume que el programa ha encontrado un error irrecuperable. La operación que dio lugar a la excepción se anula y no hay manera de regresar al código que la provocó, por tanto esa operación se anula.
- **Reanudación.** Se puede manejar el error y regresar de nuevo al código que lo provocó.

Java emplea la primera forma, pero se puede simular una segunda mediante un bloque try con un while en su interior. Repetirá hasta que el error deje de existir.



```java
public static void main(String[] args){
  boolean fueradelimites=true;
  int i; //Entero que tomará valores aleatorios de 0 a 9
  String texto[] = {"uno","dos","tre","cuatro","cinco"}; //String que representa la moneda

  while(fueradelimites){
      try{
          i=  (int) Math.round(Math.random()*9); //Generamos un indice aleatorio
          System.out.println(texto[i]);
          fueradelimites=false;
      }catch(ArrayIndexOutOfBoundsException exc){
          System.out.println("Fallo en el indice");
      }
  }

}
}
```

En este código se generan números aleatorios almacenados en la variable i. Seguidamente ese valor se utiliza en el índice del array y si el valor es mayor al tamaño del mismo, se genera un error del tipo ArrayIndexOutofBoundsException, el cual gestionamos con un catch. El código se repetirá gracias al while.

6.3. Delegación de excepciones con throws.

Cuando se produce una excepción es necesario saber quien es el encargado de solucionarla. Puede ser que sea el mismo método llamado o el código que hizo la llamada.

Cuando un método utiliza una sentencia que puede generar una excepción, pero esa excepción no es capturada y tratada por él sino que se encarga de su gestión quien llamó al método, se trata de una delegación de excepciones.

```
public class delegacion_excepciones {

     ...

     public int leeaño(BufferedReader lector) throws IOException, NumberFormatException{

          String linea = teclado.readLine();

          Return Integer.parseInt(linea);

     }

     ...

}
```

IOException y NumberFormatException son dos posibles excepciones del método leeaño, pero no las gestiona, teniendo en su cabecera un listado de excepciones que puede lancar, separadas por comas.

7. Depuración de programas

Durante la etapa de implementación, los programadores deben realizar otro tipo de pruebas que verifiquen el código que escriben, a parte de los de Caja Blanca y Caja Negra.

La depuración de programas es el proceso en el cual se identifican y corrigen errores (debugging). Hay tres etapas por las que pasa un programa cuando se desarrolla y que puede generar errores:

- **Compilación.** Una vez hemos terminado un programa, solemos pasar cierto tiempo eliminando errores de compilación. El compilador de java puede sacar a la luz errores que nos e aprecian a simple vista.
- **Enlazado.** Se trata de errpres qie son lanzados en la etapa de compilación si se encuentran librerías que han sido mal enlazadas, por ejemplo por utilizar parámetros incorrectos en tipo y número, o si esos métodos no existen.
- **Ejecución.** Es frecuente que un programa en ejecución no funcione como esperaba. Algunos errores serán detectados automáticamente y otros simplemente darán comportamientos no esperados (bugs). Al ser errores poco claros hay que recurrir a labores de investigación para encontrar la causa.

El proceso de depuración se hace a través del entorno de desarrollo que estemos utilizando. Es una herramienta que nos ayudar a eliminar posibles errores. Podemos utilizar depuradores simples como el jdb de Java o bien utilizar un depurador existente en nuestro IDE.

Los elementos que utiliza un depurador son:

- **Breakpoints o puntos de ruptura.** Son determinados por el programador a lo largo del código fuente. La ejecución del programa se detiene en estos puntos y el depurador muestra los valores de las variables tal y como están en ese momento. Se analizan los valores que tienen y el valor que deberían tener para recoger información importante sobre el proceso de depuración.
- **Ejecución paso a paso.** Podemos ejecutar el programa línea por línea, siguiendo el proceso de ejecución y supervisar el funcionamiento del mismo. El debugger ofrece la posibilidad de no entrar en métodos que no queramos en una ejecución paso a paso.

8. Documentación del código

Documentar nuestro código es necesario para facilitar su mantenimiento y reutilización, debemos documentar obligatoriamente: clases, paquetes, constructores, métodos y atributos. Opcionalmente: bucles, partes de algoritmos,...

Se puede generar documentación de nuestro código a través de la herramienta Javadoc.

La documentación se realiza a través de unos comentarios especiales llamados comentarios de documentación, que serán tomados por Javadoc para generar archivos html que permiten posteriormente navegar por cualquier documentación con cualquier navegador web.

Estos comentarios comienzan por /** y terminan en */. En su interior podemos encontrar dos partes diferenciadas:

```
/**
* Descripción principal (texto/HTML)
*
* Etiquetas (texto/HTML)
*/
```

- Zona de descripción. En ella se escribe un comentario sobre la clase, atributo, constructor o método, se puede incluir cualquier cantidad de texto, e incluso etiquetas html que le den formato.

- Zona de etiquetas: Se coloca un conjunto de etiquetas a las que se asocian textos, cada etiqueta tiene un significado especial y aparece en lugares determinados de la documentación una vez generada.

```
/**
 * Returns the index of the first occurrence of the specified element in
 * this vector, searching forwards from <code>index</code>, or returns -1 if
 * the element is not found.
 *
 * @param o element to search for
 * @param index index to start searching from
 * @return the index of the first occurrence of the element in
 * this vector at position <code>index</code> or later in the vector;
 * <code>-1</code> if the element is not found
 * @throws IndexOutOfBoundsException if the specified index is negative
 * @see Object#equals(Object)
 */
public int indexOf(Object o, int index) ...
```

8.1. Etiquetas y posición

Existen dos tipos de etiqueta:

- **Etiquetas de bloque:** son etiquetas que solo pueden ser incluidas en el bloque de documentación, después de la descripción principal y comienzan con @.
- **Etiquetas de texto:** son etiquetas que se ponen en cualquier punto de la descripción o en cualquier parte de la documentación asociada a una etiqueta de bloque. Se definen entre llaves.

	@autor	{@code}	{@docRoot}	@deprecated	@exception	{@inheritDoc}	{@link}	{@literal}
Descripción	✓		✓	✓			✓	✓
Paquete	✓		✓	✓			✓	✓
Clases e Interfaces	✓		✓	✓			✓	✓
Atributos			✓	✓			✓	✓
Constructores y métodos			✓	✓	✓	✓	✓	

8.2. Uso de las etiquetas

Las más habituales son:

- @autor texto con el nombre: Se admite en clases e interfaces. No necesita formato especial, podemos incluir tantas como queramos.
- @version texto de la versión: El texto de la versión no necesita formato especial. Es conveniente incluir versión y la fecha. Podemos incluir varias etiquetas una detrás de otra.
- @deprecated texto: Indica que no debería utilizarse, indicando en el texto las causas. Se puede utilizar en todos los apartados de la documentación y se puede añadir lo que se puede utilizar en su lugar.

```
@deprecated El método no funciona correctamente. Se recomienda el uso de {@ling metodoCorrecto}
```

- @exception nombre-excepción texto: Esta etiqueta es equivalente a @throws
- @param nombre-atributo texto: es aplicable a parámetros de constructores y métodos, describe los parámetros del constructor o método. El nombre del atributo es igual al del parámetro y seguidamente lleva una descripción.

```
@param fromIndex: El índice del primer elemento que debe ser eliminado.
```

- @return nombre texto: se puede omitir en métodos void, se describe explícitamente que tipo o clase de valor devuelve y sus posibles rangos de valores.

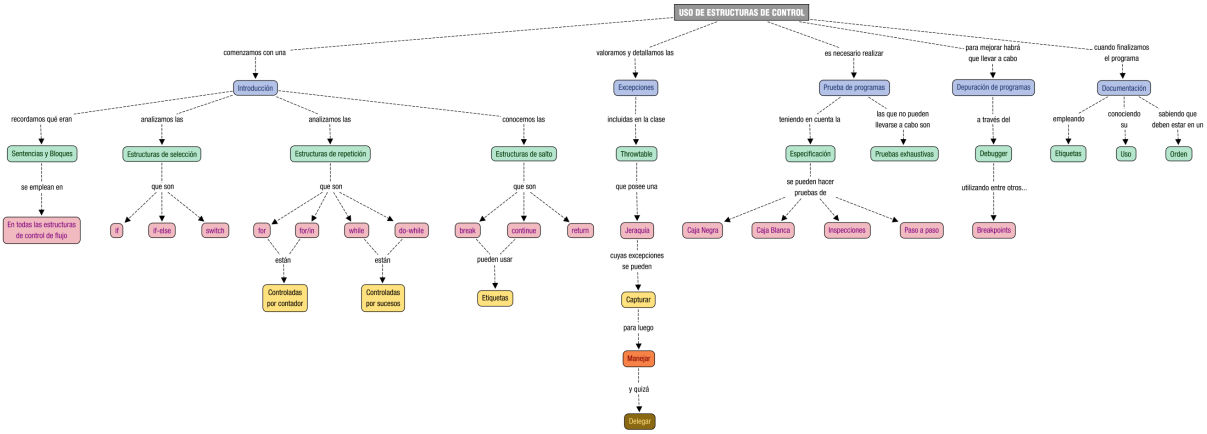
```
/**

* Chequea si un vector no contiene elementos.
*
* @return <code>verdadero</code>si solo si este vector no contiene componentes, esto es, su tamaño es cero;
* <code>falso</code> en cualquier otro caso.
*/
public boolean VectorVacio() {
return elementCount == 0;
}
```

- @see referencia: se aplica a clases, interfaces, constructores, métodos, atributos y paquetes. Añade enlaces de referencia a otras partes de la documentación.

```
* @see "Diseño de patrones: La reusabilidad de los elementos de la programación orientada a objetos"

* @see <a href="http://www.w3.org/WAI/">Web Accessibility Initiative</a>

* @see String#equals(Object) equals
```

- @throws nombre-excepción texto: se indica el nombre completo de la excepción, se puede añadir una etiqueta por cada excepción que se lance por la cláusula throws, siguiendo orden alfabético. Aplicable en constructores y métodos, describiendo las posibles excepciones.

Mapa conceptual


USO DE ESTRUCTURAS DE CONTROL
comenzamos con una
Introducción
recordamos qué eran
Sentencias y Bloques
se emplean en
En todas las estructuras de control de flujo
analizamos las
Estructuras de selección
que son
if
if-else
switch
analizamos las
Estructuras de repetición
que son
for
for/in
while
do-while
están
Controladas por contador
están
Controladas por sucesos
conocemos las
Estructuras de salto
que son
break
continue
return
pueden usar
Etiquetas
valoramos y detallamos las
Excepciones
incluidas en la clase
Throwtable
que posee una
Jerarquía
cuyas excepciones se pueden
Capturar
para luego
Manejar
y quizá
Delegar
es necesario realizar
Prueba de programas
teniendo en cuenta la
Especificación
se pueden hacer pruebas de
Caja Negra
Caja Blanca
Inspecciones
Paso a paso
las que no pueden llevarse a cabo son
Pruebas exhaustivas
para mejorar habrá que llevar a cabo
Depuración de programas
a través del
Debugger
utilizando entre otros...
Breakpoints
cuando finalizamos el programa
Documentación
empleando
Etiquetas
conociendo su
Uso
sabiendo que deben estar en un
Orden