Programación estructurada en Java

Santiago García Jimenez

Table of Contents

Introducción

Este texto pretende ser una guia para refrescar conocimientos que ya se tiene de la asignatura de programación de primer curso. En él se repasan conceptos que ya se conocen y se darán los primeros pasos en la programacion en Java.

La idea es que el alumno tenga un manual de consulta y repaso de programación básica en java con una serie de ejercicios que le ayudarán a asentar los conocimientos ya adquiridos o a refrescar aquellos que se han olvidado.

Se incluyen además en este manual, varios ejemplos de programas en Pascal y su equivalente en lenguaje Java para que el alumno pueda ver las similitudes de la programacion en ambos lenguajes.

Se describirá la forma de realizar los programas en Java sin entrar en los paradigmas de la orientación a objetos. Se describirán y utilizarán las clases mínimas para realizar las operaciónes que normalmente estan vinculadas en los niveles más bajos de la programación estructurada.

Los principales objetivos de este manual son que el lector entienda y aprenda como realizar un programa en lenguaje Java completo pero realizando una programacion de tipo estructurada. Que el lector sepa como afrontar la utilización de estructuras de control básicas, lectura y escritura desde teclado y pantalla así como a traves de archivos. Se explicará el formato para definir las variables y los arrays de una o más dimensiones además de hacer incapié en la forma de proceder a la hora de comparar los diferentes tipos de variables.

Compilacion y ejecución de un programa Java

Antes de empezar con la programación en Java veremos como se compila y ejecuta un programa en Java. El proceso de compilacion es un proceso que hay que realizar para pasar un programa de codigo que entienden las personas, que es el programa en si, a un codigo que puedan ejecutar las maquina. Antes de ejecutar cualquier programa de java, previamente hay que compilarlo para hacer que el ordenador lo pueda ejecutar. Para ello se utiliza un programa de la linea de comandos llamado ***javac***. La forma de ejecutarlo es muy sencilla, simplemente hay que ejecutar el comando seguido del codigo java que deseamos compilar. Los codigos en java tienen la extension ***.java***. Aqui tenemos un ejemplo de programa muy basico en java:

```
public class HolaMundo{

        public static void main(String[] args){

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

        }

}
```

Que se ha guardado en el fichero HolaMundo.java, luego el proceso de compilacion sera:

```
javac HolaMundo.java
```

Hay que tener en cuenta que el nombre del archivo ***.java*** debe de corresponderse siempre con el nombre que sucede a la parte del programa donde pone ***public class***. Como nosotros hemos puesto ***public class HolaMundo*** el archivo se debe de llamar ***HolaMundo.java***.

Una vez que se ha terminado de ejecutar el compilador, obtendremos los archivos ***.class*** que es el programa java compilado y listo para utilizar. La forma de ejecutar el programa es la siguiente:

```
java HolaMundo
```


Función main y ejecución de un programa en java

La estructura mínima para realizar un programa en java consta de la definición de una funcion que será la que se ejecute cuando llamemos al programa. Dicha función es a la que llamara el sistema operativo cuando ejecutemos el programa y tendrá que estar bien escrita para que el programa funcione correctamente.

Todas las aplicaciones que ejecutamos desde el sistema operativo tienen una serie de parametros que se le pueden pasar al programa. Estos parámetros son mas visibles cuando ejecutamos comandos a traves de consola:

```
animanegra@ChesseCake:~$ ls al
total 33180
drwxrxrx 50 animanegra animanegra     4096 sep  3 10:29 .
drwxrxrx  3 root       root           4096 may  4 20:50 ..
drwx  3 animanegra animanegra     4096 may  8 20:11 .adobe
animanegra@ChesseCake:
```

Este es un ejemplo de ejecucion del comando ***ls*** de linux. Como vemos tras la ejecución del comando le podemos pasar una serie de palabras que hacen que el programa se comporte de formas diferentes. Dichas palabras se denominan **parámetros**. Estamos ejecutando el mismo programa pero según lo que introduzcamos por el teclado a la hora de llamarlo realizará unas acciones u otras.

La función principal de cualquier lenguaje de programación permite recibir una serie de parámetros para permitir una comunicación básica entre el usuario y el programa. En la funcion principal que se define a la hora de programar en Java se puede observar claramente dicha funcionalidad. De manera que el programa más básico de Java que podemos realizar es el siguiente:

MyPrimerPrograma.java

```
public class MyPrimerPrograma{

        public static void main(String[] argv){


        }

}
```


Función main y ejecución de un programa en java

En este pequeño codigo se está definiendo el programa llamado ***MyPrimerPrograma***. Para definirlo hemos escrito ***public class*** antes del nombre que tendrá el programa. Todo el contenido de **nuestro programa deberá ir entre las llaves** en las que definiremos el código que se ejecutará en él. Además, el nombre del archivo en el que se aloja el codigo viene definido por el nombre del programa, de modo que el nombre del archivo java en el que debemos de guardar el codigo se deberá llamar **MyPrimerPrograma.java**.

A nuestro programa lo llama el sistema operativo. Como un programa podría llegar a tener muchas funcionalidades dentro del mismo, el sistema operativo siempre llamará a una funcionalidad específica que se tiene que llamar **main**. Debe de estar **exactamente definido tal y como se ve en el código de ejemplo**. Las piezas de codigo con funcionalidades especificas se denominan funciones, y el sistema operativo llamara a una funcion llamada main que tendra como **entrada** un **conjunto de palabras** que se le pueden introducir. Recordar el comando ***ls***, al que podiamos introducir diversas palabras para cambiar su funcionamiento. Es necesario que la funcion principal ofrezca al usuario una forma de obtener los datos que ha introducido el usuario a la hora de llamar al programa.

Del segmento de código que define la funcion main, lo único que podemos cambiar sin que deje de funcionar el programa es la palabra **argv**. Dicha palabra define la palabra que utilizaremos para acceder a lo que el usuario ha escrito en la linea de comandos. El estandar dicta que se use ***argv*** por su significado en ingles de **vector de argumentos,** pero **se le puede cambiar** el nombre por el que más cómodo nos parezca.

El resto del código de la función es inamovible, ya que si cambiamos cualquier cosa de ella el sistema operativo no sabrá a que función debe llamar para ejecutar el programa que deseamos lanzar.

Ocurre parecido con las palabras **public class MyPrimerPrograma**. Ademàs, deberemos de añadir un par de llaves que serán las que dictaminan donde empieza y termina nuestro código de programa para la funcion principal que estamos programando.

Ejecucion de un programa

Ya hemos visto lo que hace el sistema operativo cuando llama al programa para

ser ejecutado. Lo primero que hace es buscar la funcion main que esta definida en el código que ha hecho el programador y le pasa todo lo que adicionalmente ha escrito el usuario para que sirva de comunicación básica entre el programa y el usuario.

Una vez que el sistema operativo hace eso, empieza la ejecución del programa que irá leyendo lineas de una en una hasta terminar el codigo que debe de ejecutar. Es decir, todo lo que se ha puesto entre llaves.

De manera que si tenemos el siguiente programa:

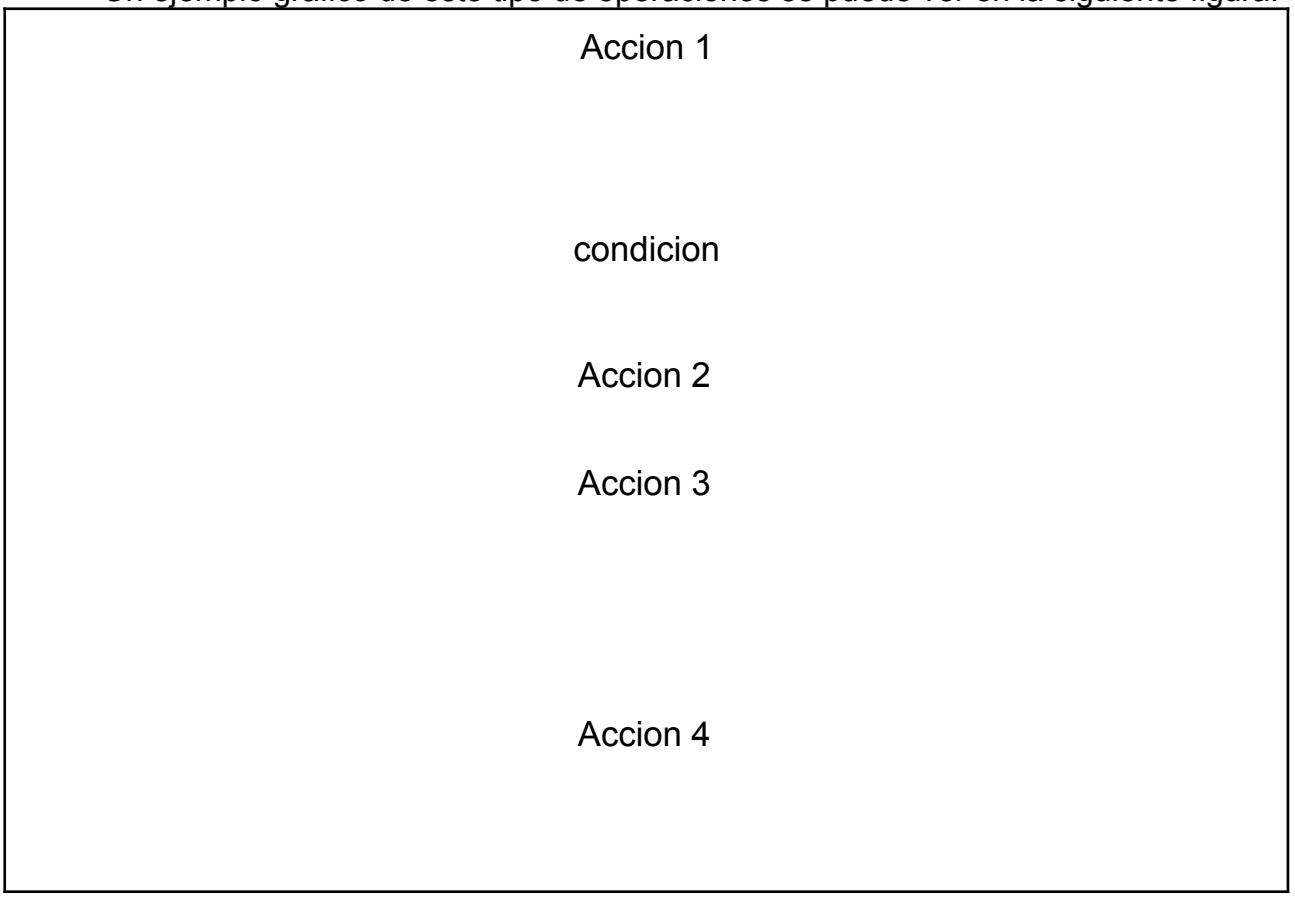
```
public class MyPrimerPrograma{

        public static void main(String[] argv){

                accion 1
                accion 2
                accion 3
                accion 4

        }

}
```

Las ordenes definidas por accion 1,2,3 y 4 se ejecutaran en orden de arriba a abajo. De esta forma, la accion 4 no se ejecutará hasta que no se hayan ejecutado todas las ordenes anteriores. Existen una serie de ordenes que permiten cambiar el flujo de programa de dos formas diferentes, una es haciendo que el programa se **bifurque** y ejecute determinadas operaciones cuando se cumpla una condicion y otras cuando se cumpla otra. Y otras directivas que permiten **repetir** ordenes hasta que una determinada condición se cumpla. **Todos los problemas** que se pueden resolver mediante la programación **se solucionan bifurcando o repitiendo acciones**. La complejidad viene de la forma de combinar esas dos sencillas operaciones para llegar a nuestro propósito.

Operaciones de bifurcación

En este tipo de operaciones se tomará una decisión de por donde debe continuar el código. Si se cumple una condición definida la ejecución realizará una serie de acciones, y si no realizará otras.

Un ejemplo gráfico de este tipo de operaciones se puede ver en la siguiente figura:



Se puede ver como tras la ejecución de la ***acción 1*** tenemos una condición. Si se cumple se ejecutará la ***accion 2*** y si no se ejecutará la ***acción 3***. Tras la ejecución de dichas acciones se ejecutará la ***accion 4***. De manera que la ***acción 1*** y ***4*** se ejecutarán siempre y la ***acción 2*** y ***3*** se ejecutarán según si se cumple o nó la condicion que hemos definido.

Este tipo de acciones en las que se bifurca la ejecución según el cumplimiento de

una condición, son las llamadas condicionales y se programan mediante la sentencia ***if*** que en ingles significa ***si***. Un ejemplo de codigo que utiliza estas sentencias es el siguiente:

```
public class MyPrimerPrograma{

        public static void main(String[] argv){

                accion 1
                if(condicion){
                        accion 2
                }else{
                        accion 3
                }
                accion 4

        }

}
```

Este es el código análogo al descrito por la figura de más arriba. En el código tenemos que el sistema operativo, nada más llamar a la función principal ***main***, ejecutará la ***acción 1***, después realizará una comprobación de si se cumple o no la condición que hemos definido. Si la condición es cierta, se ejecutará la ***acción 2*** y si es falsa se ejecutará la ***acción 3***. Tras la ejecución de la acción elegida, se ejecutará la ***acción 4***.

Se pueden introducir varias acciones a realizar dentro de un ***if***, de manera que podremos introducir el código que necesitemos ejecutar en caso de que se cumpla la acción o que no se cumpla. La única restricción es que el código que se ejecuta cuando se cumple la condición debe de ir entre las primeras llaves y el código que se ejecuta cuando no se cumple la condicion deberá ir entre las segundas llaves.

Un ejemplo de esto es el siguiente código:

```
public class MyPrimerPrograma{

        public static void main(String[] argv){

                accion 1
                accion 2
                if(condicion1){
                        accion 3
                        accion 4
                        if(condicion2){
                                accion 5
                                accion 6
                        }
                }else{
                        accion 7
                }
                accion 8
                accion 9
        }

}
```

Como se puede ver, se puede introducir el código que queramos dentro de las sentencias de tipo ***if***. En este caso tenemos dos condiciones que ejecutarán distintas acciones en funcion de si se cumplen o no. Se puede observar como tenemos un primer ***if*** con una condición a cumplir llamada ***condicion1*** y si se cumple tras ejecutar una serie de acciones realizaremos otra comprobacion de si se cumple la condicion marcada como ***condicion2***.

Preguntas:

1. ¿Para que se ejecute la ***acción 6*** que condiciones se deben de cumplir?
2. ¿Para que se ejecute la ***acción 7*** que condiciones se debe de cumplir?
3. ¿Para que se ejecute la ***acción3*** pero no se ejecute la accion 5 que condiciones se deben de cumplir?
4. ¿Para que se ejecute la ***acción 9*** que condiciones se deben de cumplir?
5. ¿Para que no se ejecute la ***acción 2*** que condiciones se deben de cumplir?

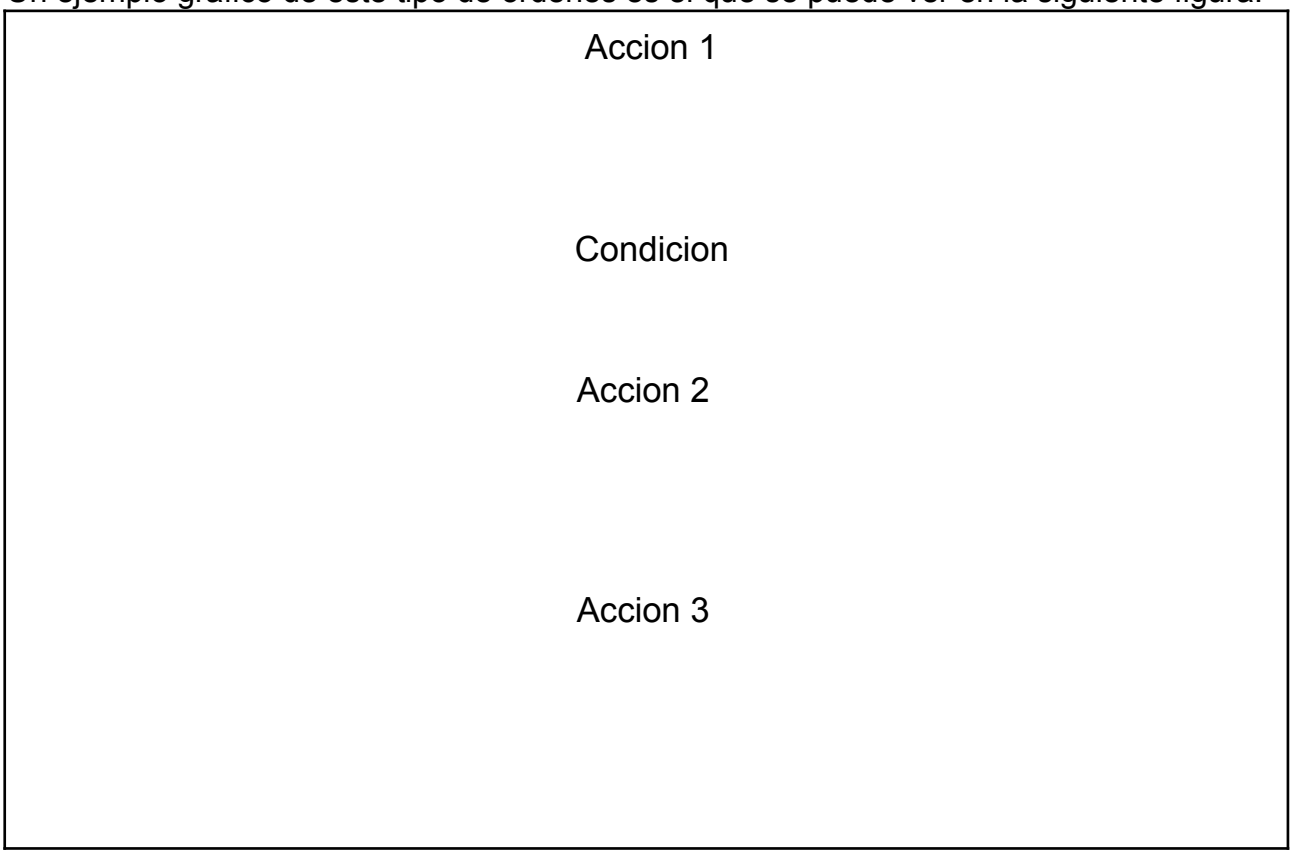
Programación estructurada en Java

Ejecucion de un programa

Operaciones de repetición

En este tipo de operaciones el flujo de ejecución de programa realizará las instrucciones que tenga dentro durante un número determinado de veces. Dependerá de si se cumple o no la llamada condición de salida.

Un ejemplo gráfico de este tipo de órdenes es el que se puede ver en la siguiente figura:



En este esquema se ejecutará la ***acción 1***, entonces se comprobará si se cumple o no la condición. En caso de que se cumpla se ejecutará la ***accion 2*** y se volverá a comprobar la condición de nuevo. En cuanto se compruebe la condición y no se cumpla, se pasará a ejecutar la ***acción 3***. De esta manera, lo que se defina dentro de la ***accion 2,*** se repetirá las veces necesarias para que se termine cumpliendo la condición.

Ejecucion de un programa

Un ejemplo de esta estructura en lenguaje java es el siguiente código:

```
public class MyPrimerPrograma{

        public static void main(String[] argv){

                accion 1
                accion 2
                while(condicion){
                        accion 3
                        accion 4
                }
                accion 8
                accion 9
        }

}
```

La ejecución del programa seguirá el siguiente curso, primero se ejecutarán las ***acciones 1*** y ***2***. Entonces se comprobará si se cumple o no la condición definida dentro del ***while***. Siempre que se cumpla la condición se entrará a ejecutar las acciones que estén definidas dentro de las llaves, en este caso la ***acción 3*** y ***4***. Tras la ejecución de la ***acción 4*** se volverá otra vez a la linea del ***while***, que volverá a comprobar si se cumple la condición. En cuanto la condición no se cumpla, el programa pasará a ejecutar las acciones que vienen despues del ***while***, que son las ***acciones 8*** y ***9***.

Hay que tener en cuenta que si tenemos una condición basada un valor que no se altera dentro del bucle, dicho bucle nunca encontrará una condición para salir, ocasionando que el programa nunca termine.

Existen dos sentencias en Java que permiten romper la ejecución dentro del bucle y hacer que salga, dichas directivas son ***break*** y ***continue***.

La directiva ***break***, permite que cuando se ejecute el programa salga del bucle en el que está, ejecutando la acción programada para ejecutarse justo después de la estructura de repetición. Si en lugar de ejecutar un ***break*** se ejecuta un ***continue***, el programa pasará directamente a evaluar la condición de la estructura de repetición que la contenga. Ocasionando que se vuelva a evaluar la condición y según se cumpla o no siga ejecutando, otra vez desde la primera acción, lo que contiene el bucle, o siga ejecutando las acciones de despues del bucle.

Estas dos directivas cobran sentido cuando vienen dentro de estructuras de tipo condicional y se quiere terminar el bucle de manera anticipada por algun motivo.

Ejecucion de un programa

Como en el caso de las estructuras de condición, en estas estructuras de repetición también se pueden anidar. De manera que podemos montar un programa como el que vemos a continuación:

```
public class MyPrimerPrograma{

        public static void main(String[] argv){

                accion 1
                accion 2
                while(condicion1){
                        accion 3
                        accion 4
                        if(condicion2){
                                while(condicion 3){
                                        accion 5
                                        accion 6
                                        if(condicion 4){
                                                accion 7
                                                break;
                                        }
                                        accion 8
                                }
                                accion 9
                        }else{
                                accion 10
                                continue;
                        }
                        accion 11
                }
                accion 12
                accion 13
        }

}
```

En este ejemplo se puede observar un programa en el que hemos anidado varias estructuras de repetición y varias estructuras de condición. Se puede observar como tenemos una serie de acciones que se ejecutan antes de empezar una repetición basada en la ***condición 1***. En caso de se

cumpla se ejecutarán repetidas veces, al menos hasta que se deje de cumplir la ***condición 1***, una serie de acciones que incluyen una estructura de bifurcación basada en si se cumple o no la ***condición 2***. Si se cumple esta ultima entraremos a una estructura de repetición que depende del cumplimiento de la ***condición 3***. Que repite varias acciones y una estructura de bifucarción basada en la ***condición 4***.

Se puede observar que se han incluido dos directivas que cambiarán el flujo normal de ejecucion de los bucles en los que estan incluidas, tenemos un ***break*** que está dentro de la estructura de repetición que depende de la ***condición 3*** y una directiva ***continue*** que está dentro de la estructura de repetición que contiene la ***condición 1***.

Preguntas

1. En caso de que no se cumpla la ***condicion 1*** que acciones debe de ejecutar el programa.
2. Hay alguna combinación posible de condiciones que permitan que se ejecute la ***acción 10*** y la ***11*** en la misma ejecución del programa. ¿Cual?
3. Hay alguna combinación de condiciones que permitan que se ejecute la ***acción 7*** y la ***acción 8*** en la misma ejecución de programa ¿Cual?
4. Depués del ***break*** ¿Que acción se ejecutará inmediatamente después?
5. Imagina una ejecución del programa en la que la ***condición 2*** nunca se cumple. Y la ***condición 1*** se va a cumplir durante ***5*** comprobaciones. Y siempre se cumplen las condiciones ***3*** y ***4*** ¿Que acciones ejecutará el programa y en que orden?
6. Imagine una ejecución similar a la anterior en la que nunca se cumpla la ***condición 3***, siempre se cumpla la ***condición 2*** y ***4***, y la ***condición 1*** se cumple durante ***5*** comprobaciones. ¿Que acciones ejecutará el programa y en que orden?

Variables

Declaracion de variables

Las variables son, junto con las estructuras de repetición y bifurcación, la base de la programación. Un programa deberá conseguir que unas variables dadas vayan tomando determinados valores gracias a la repetición y la no ejecución de determinadas sentencias para que al final consigamos que nos dé la respuesta al problema que se desea solucionar en las mismas u otras variables.

Las variables son un análogo a una caja en la que podemos almacenar cosas dentro, pero con la restricción de que, lo que introduzcamos se corresponda con lo que pone en la etiqueta de la caja. P.E: Si en la caja pone boligrafos solo se podrán meter boligrafos. De manera que si queremos utilizar una caja para almacenar cosas, primero etiquetaremos en ella que queremos meter y despues empezaremos a meter o coger contenidos de esta.

De igual forma, las variables deben de ser declaradas antes de empezar a utilizarse. La delcaración de una variable es el equivalente de etiquetar la caja con el nombre de los elementos que vamos a introducir. Una vez declaradas podremos darles valores y obtener valores de ellas.

Probablemente la opción mas sencilla para declarar variables es hacerlo al principio del programa, antes de realizar ninguna instrucción. Cuando se declara una variable el valor que esta tiene es impredecible, normalmente si es de tipo numérica suele ser 0 y si es de tipo carácter suele ser vacio, pero depende del sistema en el que se ejecute. Por este motivo la primera utilización de cualquier variable tras la declaración en el código deberá ser para darle valor.

La forma de declarar una variable siempre es la misma, primero se indica el tipo de

dato que puede almacenar una variable y después el nombre que le queramos dar a la variable. De esta forma podemos declarar por ejemplo las siguientes variables:

```
int variable1;
char variable2;
byte variable3;
```

En el trozo de código que vemos se han declarado tres variables de diferentes tipos.

Operaciones con variables

A partir de haber declarado las variables ya se pueden utilizar y se puden hacer 3 tipos de aciones con una variable. Para ello utilizaremos de ejemplo la variable ***variable1*** y supondremos que ha sido previamente declarada. Las acciones que se pueden realizar con las variables son operaciones de asignacion:

```
variable1 = 10;
```

En ellas aparece la variable en la parte izquierda de una igualdad. Y el valor que le deseemos dar en la parte derecha.

También se pueden realizar operaciones de obtención del valor. En este ejemplo utilizaremos la variable ***variable1*** y supondremos que tanto ***variable1*** como ***aux*** son variables ya declaradas y a ***variable1*** se le ha realizado previamente una operación de asignación:

```
aux = variable1;
```

En ellas utilizamos la variables para ceder su valor a otra variable. La variable que utilizamos aparecederá a la derecha y su valor se copiará a la variable que aparezca a la izquierda.

Siempre que aparezca un signo de igualdad se estara realizando una asignación de valor por lo que deberá aparecer al menos una variable que será la que reciba el valor que se asigna.

Se puede, en la parte de la derecha de la asignación, realizar operaciones con las variables para que el resultado final sea copiado a la variable que aparezca en la izquierda. En este caso realizamos operaciones con la ***variable 1*** y supondremos que las variables ***variable1***, ***aux*** y ***aux2*** estan declaradas y que a ***variable1*** y ***variable2*** se les ha realizado previamente una operación de asignación:

```
aux = variable1 + 48 * aux2;
```

En una asignación solo se cambia el valor de la variable que está a la izquierda del signo igual. Los demás valores permanecen sin alterar. Las operaciones típicas a realizar en este tipo de operaciones son:

```
res = a+b; // En res se almacenará la suma de a y b.
res = a*b; // En res se almacenará la multiplicacion de a por b.
res = a/b; // En res se almacenará la division de a entre b.
res = a%b; // En res se almacenara el valor de a modulo b, tambien llamado operador resto.
```


```
res = a+b*c; // Se pueden combinar todas las operaciones que queramos y el orden será // el orden de las operaciones
                matematicas, por lo que res valdra la multiplicacion // de b por c y a ello se le sumará el valor de a.
res = (a+b)*c; // Se puede cambiar el orden de operaciones mediante los parentesis.
```

Comprobaciones con variables

Por último, también se pueden realizar operaciones de comparación, que son las que van dentro de las condiciones a comprobar en las estructuras de bifurcación y de repetición, p.e: ***if*** y ***while***.

Las comparaciones típicas son la de igualdad (**==**), desigualdad (***!*=**), mayor (**>**), menor (**<**), mayor o igual (**>=**) y menor o igual (**<=**). Estas comparaciones devuelven verdadero (***true***) o falso (***false***) en caso de que se cumplan o no. Este tipo de respuesta es lo que espertan las comparaciones de las estructuras de bifurcación o repetición para ejecutar o no la parte del codigo que llevan dentro.

También se pueden concatenar las comparaciones mediante operadores and (**&&**) y operadores or (**||**), permitiendo realizar comprobaciones todo lo complejas que queramos para ejecutar ciertas partes del código.

En este ejemplo se mostrarán diversas variables que tomarán unos valores y ejecutarán algunas partes del código y otras no:

```
int aux;
int aux2;

aux=1;
aux2=10;

while(aux<=aux2){

        if(aux==6){

                aux=aux+1;

                while(aux2 > 8){

                        aux2=aux2-1;

                }

        }
```


```
        aux=aux+1;
}
```

Preguntas

1. Tras ejecutar todo el código ¿Que valor terminará teniendo ***aux***?
2. ¿Cuantas veces se ejecutan cada una de las dos asignaciones ***aux=aux+1***? 3. ¿Cuantas veces se ejecuta la asignación ***aux2=aux2-1***?
4. Tras ejecutar todo el código ¿Que valor termina teniendo la variable ***aux2***?
5. ¿Existe algun valor inicial de ***aux2*** que no sea 10 para el que, cuando termine de ejecutarse el código, ***aux*** valga tres veces menos que ***aux***? ¿Cual o cuales?
6. ¿Existe algun valor inicial de ***aux2*** que no sea 10 para el que, cuando termine de ejecutarse el código, ***aux*** valga una vez menos que ***aux***? ¿Cual o cuales?

Alcance de las variables

Las variables que declaramos tienen un alcance determinado en el código. En Java se pueden declarar variables a lo largo de todo el código, y esto genera una serie de problemas a la hora de utilizarse.

Las variables solo se podrán utilizar una vez que se han declarado y solo en el ámbito donde sean declaradas. De manera que si declaramos las variables al principio del código, dichas variables se podrán utilizar a lo largo de todo el código que generamos. Si se define una variable a mitad de código solo será a partir de ahí donde se podrá utilizar. Aparte, si declaramos una variable dentro de unas llaves, ya sea porque la estamos declarando dentro de un ***if*** o de un ***while***, esta variable no se podrá utilizar a partir de que salgamos de las llaves entre las que las hemos metido.

Un ejemplo de variables con distinto alcance lo podemos ver en el siguiente código:

```
int aux;

aux=2;

while(aux < 100){

        int aux2;

        aux2=1;
```


```
        while(aux2 < aux){

                int aux3;

                aux3=aux2;

                if(aux3 < aux){

                        aux3=aux3+1;

                }

                aux2=aux3+1;

        }

        aux=aux*aux2;

}
```

Preguntas

1. En que partes del codigo puedo utilizar la variable ***aux***.
2. En que partes del codigo puedo utilizar la variable ***aux2***.
3. En que partes del codigo puedo utilizar la variable ***aux3***.

Variables simples y objetos

En Java existen dos tipos de variables diferentes, unas son las variables simples y otras las variables de tipo objeto. Son facilmente reconocibles porque a la hora de declararlas, todas las variables simples tienen su primera letra en minúscula. Por otro lado, las variables de tipo objeto tienen su primera letra en mayúscula y son un tipo especial de variables que permiten la utilización de muchas funcionalidades.

Las variables de tipo básico en java son:

```
int a;
double b;
```


```
float c;
char d;
boolean e;
byte f;
short g;
long h;
```

Estas variables solo pueden utilizarse para las funcionalidades básicas de una variable que son, obtener su valor o ceder su valor a otra. Así como compararse para permitir verificar si se cumple o no una condición.

Las variables de tipo objeto se declaran de igual forma que las variables de tipo básico, definiendo el tipo de variable que es así como el nombre que van a tener. Utilizaremos para este apartado las variables de tipo objeto mas utilizadas para la programación estandar en Java. Estas variables son:

```
String a;
Integer b;
Float c;
Double d;
```

La forma de inicializar este tipo de variables es muy específica y se utiliza siempre la función ***new,*** de manera que para inicializar cada una de las variables anteriores se utilizará el siguiente formato:

```
a = new String(“una frase cualqueira”);
b = new Integer(56);
c = new Float(2.5);
d = new Double(4.4);
```

Una vez inicializadas, la forma de utilizar este tipo de variables especiales es mediante el uso de las funcionalidades que ellas mismas nos ofrecen a través del uso de puntos. Todo lo que pongamos despues de la variable y el punto serán funcionalidades que las variables de tipo objeto permiten utilizar. Una funcionalidad estandar de todas las variables de dicho tipo es la función

.equals(). Esta función nos devolverá si el valor que tenemos en una variable es igual al que tenemos en otra. La forma de utilizarlo es la siguiente:

```
a.equals("otra frase")
b.equals(3)
c.equals(a)
d.equals(4.4)
```

Al igual que para comparar una variable de este tipo no se puede hacer directamente a través de la comparación normal, la copia de este tipo de variables debe de realizarse mediante la funciónalidad ***.clone()*** que implementan todas las variables de tipo objeto.

En el siguiente ejemplo vemos como podemos clonar varias variables para poder utilizarlas mas tarde:

```
a2 = a.clone();
b2 = b.clone();
c2 = c.clone();
d2 = d.clone();
```

Al asignar directamente con el símbolo igual (**=**) lo que estaremos realmente haciendo es tener otro nombre para poder acceder a la misma variable. De manera que en lugar de copiar una variable estamos generando una forma de acceder a la misma variable con un nombre diferente.

```
a2=a.clone();
a3=a;
```

Si cambiamos algo de la variable interna en la variable ***a3*** en realidad lo estaremos cambiando tambien para ***a***. No ocurre lo mismo para ***a2*** ya que hemos realizado una copia.

Esto puede ver muy claramente cuando utilizamos arrays.

Variables

Arrays en Java

Los arrays en Java son variables de tipo objeto y por ello todos deben de ser declarados y una vez declarados, inicializados con la orden ***new.*** En el ejemplo se muestra como hacerlo:

```
int[] myarray;


myarray = new int[10];
```

En el ejemplo se ha declarado un array de variables básicas ***int*** que hemos creado con 10 huecos para dichas variables. Para acceder a cada posición del array se realizará mediante corchetes y un número, de manera similar a la que se utiliza en otros lenguajes:

```
myarray[0]=45;
myarray[1]=564;
myarray[2]=41;
...
myarray[9]=13;
```

Como se observa en el ejemplo los arrays deben de inicializarse desde la posición 0 y puede llegar hasta la posicion que corresponda con su tamaño menos 1. En caso de que intentamos acceder a una posicion del array que no exista, Java devolverá un error de acceso indebido.

En caso de declarar arrays de variables de tipo objeto, antes de asignar el valor deberemos de dar el valor en cada posicion mediante la función new.

Vamos a ver ahora el ejemplo de las diferencias entre ***.clone()*** y la asignación mediante el símbolo (**=**) en los objetos de tipo array:

```
int[] a,a2,a3;


a = new int[3];


a[0]=10;
a[1]=7;
a[2]=78;


a2=a.clone();
a3=a;
```

```
a3[1]=10;
a3[2]=10;


a2[0]=0;
```

Como se puede ver en el ejemplo se ha declarado tres variables que pueden almacenar un array de enteros básicos. Se ha creado un array y se ha dado valor a todos sus elementos. Tras hacerlo se ha realizado un clone que generá una clonación del array en la variable ***a2***. Por lo que ***a2*** tendrá una copia del array. Por otro lado en ***a3*** se ha realizado una asignación que, como se ha dicho antes, para las variables de tipo objeto, como en el caso de los arrays, lo único que se genera es un nombre adicional para acceder a los valores de la misma variable.

Preguntas:

1. Tras ejecutar todo el código ¿Que valor tendra la variable a para ***a[0]***,***a[1]*** y ***a[2]***? 2. Tras ejecutar todo el código ¿Que valor tendra la variable a para ***a2[0]***,***a2[1]*** y ***a2[2]***? 3. Tras ejecutar todo el código ¿Que valor tendra la variable a para ***a3[0]***,***a3[1]*** y ***a3[2]***?

Hay una forma típica de recorrer los arrays para dar valor a todas sus posiciones. Esta es mediante un bucle ***while***. En él, lo que haremos es utilizar una variable como marcador de la posición a la que vamos a dar el valor. Se puede ver un ejemplo en el siguiente código:

```
int[] myarray;
int i;


myarray = new int[10];


i=0;


while(i < 10){


        myarray[i]=10;
        i=i+1;


}
```

El objeto array en Java contiene una funcionalidad muy util que se puede utilizar para saber la longitud que tiene un array determinado. El nombre de esta funcionalidad es ***.length*** y se puede utilizar una vez que se ha creado el objeto poniendo un .length al nombre de la variable. Un ejemplo de esto es el siguiente:

Variables

```
int longitud;


longitud=myarray.length;
```

En este ejemplo se supone que se ha definido previamente el array llamado myarray. Si, este codigo se ejecuta a continuación del código anterior en longitud tendremos almacenado el número 10 ya que el array es de 10 posiciones.

Interacción con el usuario

Un programa que no permita interactuar con el usuario final sirve de muy poco. Al final los programas se desarrollan para ayudar al usuario a realizar distintas tareas y obtener una solución a un problema dado. El usuario deberá especificar los datos de entrada que permitirán al programa dar la solución al problema específico que está planteando el usuario. Si por ejemplo tenemos un programa que te dice si un número es o

no primo, deberemos de ofrecer al usuario una forma de dar el número que se desea verificar si es primo al programa y que el programa le diga al usuario si dicho número es o no primo. Si no se puede interactuar con el programa, de nada sirve que realice cálculos y resuelva problemas asombrosos. Si al final no podemos obtener la solución o no nos da la solución para los valores que nosotros queremos.

Java ofrece multitudes formas de interaccionar con el usuario y poder recoger valores que el usuario desee pasar a un programa, asi como ofrecer las respuestas al usuario.

Lectura de datos mediante argumentos

La forma más básica de comunicación con el usuario es el uso de los argumentos del programa. Los argumentos es la entrada de datos que teclea el usuario antes de pulsar el intro para que el programa se ejecute.

Este es el caso de utilización de la mayoria de programas de linea de comandos. Un ejemplo de esto es el comando ***cp*** que sirve para copiar un archivo a otro. La forma de utilizar cp es la siguiente:

```
cp archivo_origen archivo_destino
```

Cuando ejecutamos el comando le estamos pasando en el propio comando, el archivo origen y el archivo destino. El programa internamente tomará esos valores para realizar la copia del archivo. Una vez copiado el archivo correctamente el programa ***cp*** directamente no muestra nada por pantalla. Si ha habido algun error, el programa sacara el error por pantalla, como por ejemplo un error de nombre de fichero invalido.

Cuando programamos en Java, la forma de obtener los datos que ha pasado el usuario a la hora de ejecutar el programa es mediante una variable de entrada. Podemos ahora recordar como era la estructura del programa más básico de java en el que solo teniamos la función main, necesaria porque es la función a la que llamaba el sistema operativo para ejecutar lo que hemos programado. El programa era de esta forma:

```
public class MyPrimerPrograma{

        public static void main(String[] argv){
```


```
        }

}
```

Como se puede observar y ahora que sabemos lo que es una variable y un array, la función main tiene como entrada un array de Strings. Dicho array contendrá en sus posiciones las diferentes palabras que le ha pasado el usuario a la hora de ejecutar el programa. Como se puede ver el tipo de dato de dicho array es ***String*** por lo que cada posición del array contendrá la palabra escrita por el usuario en dicha posición empezando por la posición 0.

Se puede saber la cantidad de argumentos que ha introducido el usuario utilizando la funcionalidad length disponible en los arrays.

Para obtener un valor de una posición específica que se haya introducido por el usuario no hay mas que asignar dicha posición del array a una variable que creemos nosotros y después utilizarla como nos convenga. Se puede observar un ejemplo en el siguiente código:

```
public class MyPrimerPrograma{

	public static void main(String[] argv){

		String aux;

		if(argv.length > 2){

			aux = argv[2].clone();

		}
	}

}
```

Como se observa en el código, hemos realizado una copia de lo que el usuario ha introducido como tercer argumento y se ha introducido en la variable aux. Que es del mismo tipo que el array que contiene lo que el usuario ha escrito.

Salida de datos por pantalla

La salida de datos por pantalla en Java es muy sencilla. Para ello se utilizará una variable del sistema que se llama ***System.out*** y utilizaremos una funcionalidad de dicha variable que se llama .***println()***. Cada vez que se llame a esta funcionalidad se escribirá

por pantalla lo que le pasemos a dicha función como parámetros. El tipo de parámetro que acepta .***println()*** debe de ser ***String*** por lo que podremos escribir literales como ***"esto es un literal"*** o bien introducir variables de tipo ***String*** o que se hayan pasado a ***String***. Normalmente practicamente casi todas las variables de tipo objeto disponen de una funcionalidad llamada .***toString()*** que permiten convertir los valores contenidos en la variable en un ***String***. De manera que se puede imprimir su salida directamente mediante esta funcion. Un ejemplo de utilización de dicha funcionalidad es el conocido ejemplo del ***hola mundo*** en el que se escribe "hola mundo" en la pantalla del ordenador:

```
public class MyPrimerPrograma{

        public static void main(String[] argv){

                System.out.println("Hola mundo");

        }

}
```


Operaciones de cambio de tipo

Todos los lenguajes de programación disponen de ciertas funcionalidades que permiten asignar los valores de las variables de un tipo en otro. Esto permite por ejemplo componer textos para ofrecer por pantalla al usuario los datos que necesita que pueden estar internamente almacenados como enteros y no como ***Strings***, obtener del ***String*** que escribe el usuario el dato numerico que desea utilizar como entrada para el programa o obtener el valor de una variable en formato de entero para poder utilizar una función concreta que necesite la entrada de datos en dicho tipo de dato.

En este apartado se verán las conversiones más utilizadas en Java. Se empezará por la transformación a ***String***, que permite cambiar los datos a un formato que puede después escribirse en pantalla para que lo vea el usuario. En las variables que son de tipo objeto, se utiliza una funcionalidad llamada **.toString()** que devuelve una nueva ***String***. Dicha ***String*** se puede o bien almacenar en una variable de tipo ***String*** para seguir utilizando dicha ***String*** o bien utilizarla directamente para sacarla por pantalla. A continuación se muestra un ejemplo de la utilización de dicha funcionalidad para dar al usuario un resultado numérico:

```
public class Salida{

        public static void main(String[] args){

                Integer i;

                i = new Integer(args.length);

                System.out.println(i.toString());

        }

}
```

Este sencillo programa lo que hace es dar al usuario el número de argumentos que le esta pasando al programa. Como ***i*** es de tipo ***Integer***, hace falta realizar una conversión para que el ordenador sepa como escribir el tipo de dato ***Integer*** como si fuese un ***String***.

La mayoria de veces lo que se desea escribir es una frase entera en lugar de solo el dato que se desea saber para informar al usuario del resultado de una forma mas amigable. Lo bueno es que

el lenguaje Java permite concatenar ***Strings*** de una forma muy sencilla mediante el operador ***suma (+).*** Al igual que cuando sumamos números nos dá como resultado otro número que es la suma de las dos variables que introducimos, al utilizar la suma en ***Strings*** nos devolverá ua ***String*** compuesta por todo el texto seguido. De manera que se puede utilizar esa estrategia para dar al usuario los resultados componiendo textos y utilizando los valores de las variables que tienen los resultados:

```
public class Salida{

        public static void main(String[] args){

                Integer i;

                i = new Integer(args.length);

                System.out.println("El número de argumentos que has introducido es "+i.toString()); }

}
```

Para pasar las variables básicas a formato ***String*** se puede realizar creando una variable objeto del mismo tipo y utilizar su funcionalidad de ***.toString()***. De manera que si queremos pasar un ***int*** a ***String*** lo haremos a través de una variable nueva de tipo Integer de esta manera:

```
int i;
Integer ii;
String aux;

i = 10;
ii = new Integer(ii);
aux = ii.toString();
```

El código utilizado para pasar de una variable ***double***, ***float*** y otras básicas es análogo a este sistema.

Operaciones de cambio de tipo

Debido a cómo esta implementado Java, también se pueden utilizar directamente las variables de tipo basico, asi como las variables de tipo objeto numerico ***Integer***, ***Double*** y ***Float*** directamente para la composición de una cadena de texto. Por lo que una manera rapida de realizar las conversiones es concatenando las variables con la cadena vacia (***""***) lo que dará como resultado una ***String***. A continuacion se muestra un ejemplo de esto:

```
int i;
String aux;
i=10;
aux = “”+i;
```

En el caso de variables básicas de tipo numérico, entre ellas se pueden realizar asignaciones sin mayor problema siempre que añadamos un paréntesis y dentro se especifique el tipo de dato que queremos generar. Hay que tener en cuenta que perderemos la precision de las comas cuando por ejemplo pasemos de variables de tipo double a integer por ejemplo:

```
public class Prueba{

	public static void main(String[] args){

		int i;
		double aux;
		aux = 13.5;
		i = (int)aux;
		System.out.println(i+" "+aux);

	}

}
```

En el caso de tener variables de tipo objeto numericas ***Integer***, ***Double***, ***Float***... cuando queremos convertir a ***Integer*** tenemos una funcionalidad llamada ***.intValue()***, que se puede utilizar en las variables de tipo ***Double*** o ***Float***.

```
public class Prueba{
```


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

                Integer i,ii;
                Double aux;
                Float aux2;
                aux = new Double(13.6);
                aux2 = new Float(35.2);
                i=aux.intValue();
                ii=aux2.intValue();
                System.out.println(i+" "+aux+" "+ii+" "+aux2);

        }

}
```

Existen funciones analogas para convertir a ***Double*** y ***Float***. Para convertir a Double desde una variable de tipo ***Float*** o ***Integer*** se puede utilizar ***.doubleValue()***. En el caso de tener una variable de tipo ***Integer*** o de tipo ***Double*** se puede utilizar la funcionalidad ***.doubleValue()***.

Si lo que queremos es pasar de ***String*** a ***Integer***, ***Double*** o ***Float***, se puede generar una nueva variable de dichos tipos utilizando new y usando como entrada la variable de tipo ***String***:

```
public class Prueba{

        public static void main(String[] args){

                String palabra;

                Integer aux;
                Double aux2;
                Float aux3;

                palabra = new String("13");

                aux = new Integer(palabra);
                aux2 = new Double(palabra);
                aux3 = new Float(palabra);
```


```
                System.out.println(aux+" "+aux2+" "+aux3);


        }

}
```

Hay que darse cuenta que esto último es muy útil a la hora de obtener datos numéricos que ha introducido el usuario desde, por ejemplo, los argumentos del programa. Como ya se sabe, los argumentos se obtienen de un array de ***Strings*** por lo que no se pueden utilzar diréctamente para realizar cálculos matemáticos. Así que dichas variables deberán pasarse a un formato numérico, para que podamos después realizar operaciones con ellas. En el siguiente ejemplo se tomará el numero que nos de el usuario como argumento ***numero 1*** y se utilizara para imprimir los números desde el 0 hasta dicho número:

```
public class Prueba{


        public static void main(String[] args){


                Integer cont;
                Integer maximo;


                cont = new Integer(0);
                maximo = new Integer(args[0]);


                while(cont < maximo){


                        System.out.println("El contador va en "+cont);


                        cont = cont + 1;


                }


        }

}
```


Como apunte final cabe resaltar que en lo que respecta a variables numéricas, se pueden realizar asignaciones directas de variables básicas a sus respectivas variables de tipo objeto sin necesidad de inicializarlas mediante ***new***, de forma que se pueden hacer asignaciones de un ***int*** a un ***Integer*** de un ***double*** a un ***Double*** y de un ***float*** a un ***Float*** y vice versa.

```
public class Prueba{

	public static void main(String[] args){

		int i;
		Integer ii;

		double a;
		Double aa;

		float b;
		Float bb;

		i=3;
		a=4.4;
		b=(float)3.4;

		ii=i;
		aa=a;
		bb=b;

		System.out.println(" "+ii+" "+aa+" "+bb);

	}

}
```

En el caso de las ***Strings*** también se pueden generar sin necesidad del ***new*** utilizando simplemente asignaciones y escribiendo lo que se desee entre comillas o componiendo una nueva ***String*** via concatenación de varias ***Strings***. En este caso hay que tener especial cuidado a la hora de realizar las comparaciones y recordar que las comparaciones se hacen a través de la funcionalidad ***.equals()*** en las variables de tipo objeto.

Lectura y escritura de ficheros

La lectura y escritura de ficheros es otra de las formas de interacción entre el usuario y el programa. Muchas veces el usuario dispone de una serie de datos que quiere estudiar y desea que el programa los lea directamente sin tener que teclearlos uno por uno. Por otro lado el usuario puede querer obtener un fichero como salida del programa para poder imprimirlo o leer los datos en otro momento.

En Java existen variables de tipo objeto que nos ayudarán a realizar las tareas de escritura y lectura de ficheros de forma muy simple.

Escritura de ficheros

Para la escritura de ficheros se utilizaran siempre tres variables, una de tipo ***FileOutputStream***, otra de tipo ***OutputStreamWriter*** y otra de tipo ***BufferedWriter***. La razón por la que utilizamos tres variables diferentes es porque Java intenta juntar funcionalidades en cada variable de tipo objeto y usa un orden estratificado para crear las variables. El objetivo final es utilizar la funcionalidad que ofrecen las variable de tipo ***BufferedWriter*** ya que disponen una funcionalidad que permite escribir lineas en un fichero. El problema es que para poder leer lineas necesitamos antes saber escribir caracteres, por lo que para generar una variable de tipo ***BufferedWriter*** necesitamos darle como entrada una variable que tenga una funcionalidad que permita la escritura de caracteres, porque si no sabe como debe escribir un carácter no podra saber somo escribir una linea entera de texto. Las variables de tipo ***OutputStreamWriter*** tienen una funcionalidad que permite escribir caracteres, el problema es que para poder escribir caracteres necesitan saber primero como escribir bytes y en donde escribirlos. Por ello, le debemos pasar a la hora de generar la variable de este tipo una variable que le de dichas

funcionalidades. Ese tipo de variable son las variables de tipo ***FileOutputStream***.

Realmente se podría utilizar directamente la variable de tipo ***FileOutputStream*** para escribir byte a byte en los archivos, lo que pasa es que realmente tedioso el tener que pasar todas las variables a formato byte para despues realizar el proceso de escritura byte a byte.

Algo parecido pasa con el ***OutputStreamWritter***, pasar todo a caracteres para ir, de uno en uno, escribiéndolos en el fichero resulta muy frustrante. Probablemente la forma mas sencilla es componer cadenas de texto tal y como lo haciamos para sacar datos por pantalla, por este motivo se ha elegido el uso del ***BufferedWritter***.

La inicialización de las variables, permitirá abrir un fichero y que estemos en disposición de escribir en él. Eso implica que el puntero de escritura o el lugar de donde empezaremos a escribir se posiciona al principio del archivo. Lo que implica que si habia algo escrito en él, lo borraremos.

La forma de iniciar las variables es muy sencilla y se ejemplifica a continuación:

```
FileOutputStream fi;
```


```
OutputStreamWriter sw;
BufferedWriter bw;


fi = new FileOutputStream("ruta_al_fichero");
sw = new OutputStreamWriter(fi);
bw = new BufferedWriter(sw);
```

Como se puede observar simplemente metemos las variables una dentro de la otra y la primera necesita para ser creada un nombre del archivo que queremos escribir. Una vez se han inicializado las 3 variables usaremos la última para poder escribir. Y lo haremos mediante la funcionalidad ***.write()***. Esta funcionalidad espera recibir una String y dicha String la pasará al fichero especificado. De manera que si queremos escribir 3 lineas deberemos de ejecutar dicha orden 3 veces con los datos que queramos volcar al fichero, incluyendo un carácter de intro al final(***"\n"***). En el siguiente código utilizaremos el ***BufferedWriter*** para escribir 3 lineas de texto en el fichero:

```
bw.write("Esta es la linea de texto "+1+"\n");
bw.write("Esta es la linea "+2"de texto+"\n");
bw.write("Esta es la ultima linea de texto"+"\n");
```

Como se puede ver en el ejemplo la entrada de la funcionalidad ***.write()*** es una

String luego podemos componerla y formarla por distintos tipos de datos y concatenarlos utilzando el símbolo mas(+).

Una vez escritas las lineas de texto que se deseen siempre deberemos cerrar el archivo llamando a la funcionalidad ***.close()***. Esto se hace de la siguiente forma:

```
bw.close();
```

Para que el programa funcione se deben de añadir también una serie de instrucciones dentro de la función que contenga las variables que utilizan los archivos (tanto para lectura como para lectura), en este caso la funcion main. Esto es debido a que Java requiere que digamos que hacer en caso de que existan errores y como proceder ante ellos. Por ejemplo, que vamos a hacer si el archivo no existe o si no podemos escribir en el. Como estamos en un nivel básico de programación en java, la forma en que procederemos es directamente no preocuparnos de ello y hacer que se

notifique la excepción como error en caso de que ocurra. Esto se consigue añadiento las directivas ***throws*** en la funcion ***main***. A continuación, vemos un programa completo que escribe una serie de números del 0 al numero que especifique el usuario en el primer argumento en un fichero. Por supuesto utilizaremos una estructura de repeticion (***while***) para que las lineas se escriban de forma automática:

```
import java.io.*;

public class Escribe{

	public static void main(String[] argv) throws IOException{

		FileOutputStream os;
		OutputStreamWriter sw;
		BufferedWriter bw;

		int i;
		int max;

		os = new FileOutputStream("salida.txt");
		sw = new OutputStreamWriter(os);
		bw = new BufferedWriter(sw);

		max = new Integer(argv[0]);

		i = 0;

		while(i < max){

			bw.write("numero "+i+"\n");

			i=i+1;

		}

		bw.close();

	}

}
```


Lectura de ficheros

La lectura de ficheros en Java se realiza de forma muy similar a la escritura. Tambien se utilizarán tres variables que deberemos inicializar de la misma forma pero utilizaremos las variables ***FileInputStream***, ***InputStreamReader*** y ***BufferedReader***. Como se puede ver se parecen mucho a las anteriores solo que esta vez en vez de ***write*** y ***output*** llevan las palabras ***read*** e ***input***.

La forma de inicializarlas es la misma que la anterior, pero el archivo que se especifica en la inicializacion de la variable ***FileInputStream*** será el fichero a leer en lugar del fichero a escribir.

Un ejemplo de inicialización es el siguiente:

```
FileInputStream fi;
InputStreamReader is;
BufferedReader br;

fi = new FileInputStream("ruta al archivo");
is = new InputStreamReader(fi);
br = new BufferedReader(is);
```

A la hora de leer utilizaremos la funcionalidad ***.readLine()*** que es propia de las variables de tipo ***BufferedReader***. Esta funcionalidad devolverá una ***String*** que podremos salvar en una variable del mismo tipo. En este ejemplo salvaremos tres lineas leidas del archivo en tres variables diferentes:

```
linea1 = br.readLine();
linea2 = br.readLine();
linea3 = br.readLine();
```

Una vez terminada la lectura del archivo deberemos, al igual que cuando escribimos, cerrar el fichero usando la funcionalidad ***.close()*** de la variable de tipo ***BufferedReader***:

```

```


```
br.close();
```

Como ocurre en el caso de la escritura, cuando creamos las variables para iniciar la lectura, el puntero de lectura se situará al principio del archivo. De manera que si vamos leyendo lineas y queremos volver al principio del todo, simplemente deberemos cerrar el archivo y volver a inciar las tres variables iniciales. Al volverlas a iniciar y llamar otra vez a la funcionalidad ***.readLine()***, como el puntero de lectura a vuelto al principio volveremos a leer la primera linea de nuevo.

Codigos de ejemplo en java

Introducción

En este código se propondrán una serie de ejercicios muy básicos de lenguaje Java. Estos están preparados para tomar agilidad en el lenguaje, no para aprender a solucionar problemas. También se aporta una solución a la forma de programarlo. De esta manera puede utilizarse este capítulo como una vía de consulta básica a la hora de realizar los primeros programas en Java que es cuando mas dudas se tienen en lo que al lenguaje se refiere.

Escritura de texto por pantalla

Propuesta

Escriba un programa en Java que saque por pantalla la frase “Este es mi primer programa en java”

Solución

```
public class Ejemplo1{

        public static void main(String[] args){

                System.out.println("Este es mi primer programa en Java");

        }

}
```


Escritura de texto por pantalla de todos los parámetros

Propuesta

Escriba un programa que escriba por pantalla todos los parámetros que pasa un usuario por pantalla. En cada linea se debera escribir un parametro diferente. En caso de no pasar ningun parámetro el programa debera escribir por pantalla “No se han pasado parámetros al programa”.

Solución

```
public class Ejemplo2{

        public static void main(String[] argumentos){

                int i;

                i=0;

                while(i < argumentos.length){

                        System.out.println(argumentos[i]);

                        i=i+1;

                }

                if(i == 0){

                        System.out.println("No se han pasado parametros al programa"

);                  }

        }

 }
```


Rellenado de una tabla utilizando los parámetros

Propuesta

Utilizando como entrada los parámetros realiza el llenado de una tabla de tipo int. Después muestre por pantalla los valores de la tabla separados por espacios.

Solución

```
public class Ejemplo3{

	public static void main(String[] miargs){

		int[] tabla;
		Integer convierte;
		String salida;
		int i;

		if(miargs.length > 0){

			salida = new String();
			tabla = new int[miargs.length];

			i=0;

			while(miargs.length > i){

				tabla[i] = new Integer(miargs[i]);

				i=i+1;

			}

			i=0;

			while(miargs.length > i){
```


```
                                convierte = new Integer(tabla[i]);

                            salida = salida+convierte.toString()+" ";

                                i=i+1;

                          }

                        System.out.println(salida);

                }

        }

}
```


Entrada de texto desde teclado

Propuesta

Realice un programa que pida al usuario introducir una frase y imprima la primera palabra que haya introducido el usuario. El programa seguira pidiendo introducir frases hasta que el usuario deje una linea el blanco.

Solución

```
import java.io.*;

public class Ejemplo4{

        public static void main(String[] rguments) throws IOException{

                InputStreamReader sr;
                BufferedReader br;

                String lectura;
                String[] palabras;



                sr = new InputStreamReader(System.in);
                br = new BufferedReader(sr);

                System.out.println("Escriba una frase:");

                lectura = br.readLine();

                while(!lectura.equals("")){

                        palabras=lectura.split(" ");

                        System.out.println(palabras[0]);
```



```
                        System.out.println("Escriba una frase:");

                        lectura = br.readLine();

                }

                br.close();



        }

}
```

Programación estructurada en Java

Codigos de ejemplo en java

Salida de texto a un fichero

Propuesta

Codifique un programa que permita guardar en un fichero especificado por parametros una frase que le pida el programa introducir al usuario.

Solución

```
import java.io.*;

public class Ejemplo5{

	public static void main(String[] args) throws IOException{

		FileOutputStream fo;
		OutputStreamWriter sw;
		BufferedWriter bw;

		InputStreamReader ir;
		BufferedReader br;

		String linea;

		if(args.length == 1){

			fo = new FileOutputStream(args[0]);
			sw = new OutputStreamWriter(fo);
			bw = new BufferedWriter(sw);

			ir = new InputStreamReader(System.in);
			br = new BufferedReader(ir);

			System.out.println("Escriba una frase");

			linea = br.readLine();
```

Programación estructurada en Java

Codigos de ejemplo en java

```
                        bw.write(linea+"\n");


                        br.close();
                        bw.close();

                }

        }

}
```


Programación estructurada en Java

Codigos de ejemplo en java

Mostrar un archivo por pantalla

Propuesta

Realice un programa que permita mostrar por pantalla un archivo de texto que se le especifique por parametro.

Solución

```
import java.io.*;

public class Ejemplo6{

	public static void main(String[] arguments) throws IOException{

		FileInputStream fi;
		InputStreamReader sr;
		BufferedReader br;

		String lectura;
		String[] palabras;


		if(arguments.length == 1){

			fi = new FileInputStream(arguments[0]);
			sr = new InputStreamReader(fi);
			br = new BufferedReader(sr);

			lectura = br.readLine();

			while(lectura != null){

				System.out.println(lectura);

				lectura = br.readLine();
```


Programación estructurada en Java

Codigos de ejemplo en java

```
			}

			br.close();

		}


	}

}
```

Programación estructurada en Java

Codigos de ejemplo en java

Copia de un archivo a otro

Propuesta

Realice un programa que permita realizar una copia de un archivo de texto a otro. Los ficheros tanto de origen como de destino se especificaran como parámetro.

Solución

Suma de los números de un archivo

Propuesta

Realice un programa que permita calcular la suma de una serie de números que se le pasen en un archivo. El archivo siempre tendra un numero por linea y solo estará compuesto por números.

Solución