

ISSD

—Asc—
Analista de
Sistemas

ESTRUCTURAS
REPETITIVAS

AEDII

Algoritmos y Estructuras de Datos 1

Prof. Ing. Eduardo Mansilla
Módulo didáctico - 2019



Unidad 2



Clase 4



Estructuras repetitivas.
Estructura While.
Contadores. Acumuladores.



Al final de esta clase ya podrás:
| Comprender situaciones
problemáticas que para su
resolución impliquen el uso de
ciclos repetitivos con diferentes
características.
| Valorar la conveniencia
de aplicar contadores y
acumuladores para facilitar el
diseño de algoritmos eficientes.

Ciclos repetitivos



¡Bienvenidos a la segunda unidad de la materia que comienza con esta cuarta clase!

En esta oportunidad comenzaremos a aprender situaciones problemáticas que para su resolución impliquen el uso de ciclos repetitivos con diferentes características.

Además intentaremos valorar la conveniencia de aplicar contadores y acumuladores para facilitar el diseño de algoritmos eficientes.

Organizaremos la clase en cinco temas:

- | El primero de ellos nos va a introducir en las **estructuras repetitivas**.
- | En el segundo nos adentraremos en la **estructura While**.
- | En tercer lugar trabajaremos sobre los **contadores**,
- | y en cuarto lugar estudiaremos los **acumuladores**.

| Finalmente te proponemos dos **casos de aplicación**: en primer lugar, la **impresión de series matemáticas** y en segundo término, el **ingreso de valores hasta que se digite una clave**.

Al igual que la clase anterior, te proponemos ejercicios comentados y otros para que los resuevas vos sólo. Recordá que para aprender esta materia es necesario realizar todos y cada uno de los ejercicios.

¡Que la disfrutes!



Antes de comenzar con la clase, lee el siguiente artículo.

El 'detrás de escena' de los desarrolladores

La creación de videojuegos es un trabajo en equipo. Cómo se conforma el grupo para explotar el talento

¿Cómo se crean los videojuegos? ¿Qué trabajo y motivaciones están detrás de ellos? ¿Cómo son las personas y los equipos que los desarrollan? Infobae habló con un referente argentino del sector, Guillermo Averbuj, fundador y productor de contenidos de Gamester, con una larga trayectoria como diseñador de juegos y consultor para productos y empresas de Francia, Alemania, México e Israel. Guillermo divide su tiempo entre dos pasiones: crear videojuegos e incentivar a la industria nacional de manera proactiva e innovadora. Para lograrlo se comprometió en infinidad de actividades que van desde capacitaciones a la creación de IndustriaVG -el primer sitio bolsa de trabajo especializado en videojuegos en Latinoamérica, totalmente gratuito-. También fundó Game Work Jam, encuentros de trabajo cooperativo de desarrollo de videojuegos del que participan reconocidos trabajadores de la industria y cuyo objetivo es brindar apoyo y mentorías. Además, es el autor del primer manual sobre videojuegos disponible en español.

-¿Cómo son los videojuegos "made in Argentina"

Existen dos cosas, por un lado está el desarrollador que tiene ganas de explotar nuestra propia cultura y mostrar todo lo que nos conforma como argentinos, los aspectos regionales, la mitología local. Después está la gente que trata de hacer juegos que tienen que ver con los mercados más exitosos. Hay una mezcla, pero lo que sí es verdad es que tenemos una gran creatividad por explotar y hacer cosas nuevas e interesantes.

-¿Cuáles son los pasos para crear un videojuego?

Empezás enamorándote de la idea, eso no está recomendado porque siempre va a ir cambiando. Comenzás a armar tu proyecto, te encontrás con problemas que vas resolviendo y llega un punto en que odias lo que te está pasando (risas) por tener que resolver problemáticas que no eran pensadas en un principio. Luego, cuando lo lanzaste, es la sensación de tener un hijo al que creaste, casi artesanalmente, y se siente increíble. Es muy fuerte ver a la gente como se entretiene o, al menos, pasa una experiencia con lo que vos hiciste (porque no siempre es diversión, a veces pueden ser experiencias casi artísticas). Se están empezando a ver cada vez más videojuegos con temas educativos, es el nuevo medio de comunicación con los chicos.

-¿Cuánto dura el proceso de creación de un producto?

Depende mucho del tipo de juego que hagas. Si es para un mercado casual, si es para un proyecto propio que querés probar algún concepto, si es a largo plazo o querés que eso reviva en el tiempo. Lo recomendable es no invertir menos de 6 meses en un proyecto típico. También he participado en proyectos que duraron 4 años y que podrían no haber continuado, lo cerramos sólo para hacer una segunda versión, con más tecnología.

-¿Es un trabajo individual o en equipo?

Hay una pequeña parte de desarrolladores que se autodenominan los indies –término disputado entre varias “facciones”-. Hay gente que sí tiene las capacidades y la posibilidad de hacer un juego sólo; de hacer el diseño, la programación, el sonido, etc. Pero eso no es lo más frecuente. Lo común es estar dentro de un equipo y trabajar en un conjunto de 3 a 4 personas como mínimo, con un límite de unos 20 profesionales aproximadamente pero es una cuestión de presupuesto o de realidad.

-¿Los equipos son interdisciplinarios?

Absolutamente, hay artistas gráficos, ilustradores, diseñadores gráficos, diseñadores de usabilidad, diseñadores mecánicos (o game designer), además de la parte de management, productores que organizan el equipo, sonidistas, músicos, etc. Luego, a todo lo que es arte lo podés dividir en 2D y 3D. A veces se intercalan ambos para generar distintos efectos. Como si fuera poco, también tenés el equipo de testing y play testing, que chequea la calidad del producto, que vaya en el mercado y que sea entretenido.

-¿Está creciendo mucho el nicho del videojuego?

Si, muchísimo. Hace más de 12 años era sólo un grupo de locos haciendo “cositas”, luego se formó la cámara de desarrollo de videojuegos, ADVA, y está creciendo constantemente. Aparecieron empresas del exterior que se instalaron en nuestro país. Crecieron estos equipos indies, que la siguen luchando con un producto netamente nacional y cultural. Tenemos altibajos pero crecemos. No sé si en Argentina es un nicho como mercado, pero sí como industria. En nuestro país, como mercado, tenemos un grave problema que es la piratería. La gente está acostumbrada a que es un contenido que se bajan de una manera u otra, en vez de pagar como corresponde. Eso nos afecta mucho porque no podemos probar nuestros productos en el mercado del país en donde vivís que es el que mejor conoces.

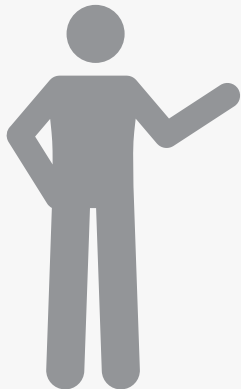
-Por último, me gustaría que nos cuentes sobre la actividad de la Fundación y cómo se apoya desde allí a los nuevos desarrolladores

Hace un par de meses, firmamos con varios interesados en la industria local y regional. FUNDAB, que es la Fundación Argentina de Videojuegos es sin fines de lucro y tiene como objetivos ayudar a que ciertos proyectos –que no son comunes de ser fondeados– consigan apoyo. La idea es que entren proyectos dentro de las 7 u 8 líneas que tenemos definidas y, cada referente va a comenzar a trabajar con la gente que fue aceptada para cada línea para poder hacerlos llegar a un puerto más formal. El objetivo es que puedan llegar a tener la idea bien armada, en lo formal, para luego poder buscar el financiamiento.

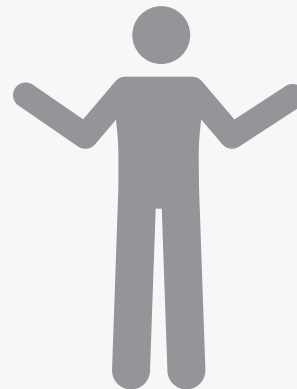
Infobae – Julio de 2016



La Programación tiene múltiples aristas: no se limita al desarrollo de aplicaciones administrativas, sino que abarca cada vez más áreas consideradas menos importantes, como los videojuegos.



Pasar de ser un usuario que juega con lo que otros pensaron, a ser capaces de imaginar y crear nuestros propios recorridos, personajes y desafíos, es cambiar el rol de consumidores al de productores de software de alto valor.



Con lo que vamos a aprender en la clase de hoy, estarás en condiciones de empezar a imaginar tus propias aplicaciones ¡Adelante!



Estructura Repetitiva

¿Conocés PacMan? Seguro que sí.
Supongamos ahora que te han pedido que realices la programación de este juego.

Ya sabés que el personaje se mueve dentro de un laberinto, y que cada vez que es tocado por uno de los fantasmas, pierde una vida. Podrá continuar el juego hasta que se quede sin vidas, en donde aparecerá el fatídico “Game Over”.

Si PacMan tiene 3 vidas cuando empieza el juego, con lo que sabés de programación hasta ahora, no te queda otra que repetir tres veces el algoritmo del juego.

Pero hay una solución mucho más práctica,

que consiste en usar una instrucción que se encarga de **repetir** automáticamente una secuencia de comandos determinada.

Si la condición que determina la ejecución de las instrucciones es siempre verdadera, estamos en presencia de un ciclo repetitivo infinito. Dicha situación es un error de programación, porque nunca finalizaría el programa. En el ejemplo de PacMan, la condición que dará por terminado el ciclo repetitivo, será que la variable **vidas** sea igual a cero, en cuyo caso el juego llegará a su fin.



Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces, dependiendo de una condición que puede ser falsa o verdadera.



Desempeños

Desempeño 33

En las clases anteriores hablamos de lo que significaba “inicializar” una variable. En el ejemplo del PacMan ¿cómo debería inicializarse la variable **vidas**? ¿Y la variable **puntos**?



La estructura While



Volvamos al ejemplo del PacMan.

Suponiendo, como dijimos, que tenemos una variable llamada **vidas** que empieza valiendo 3 y que disminuye en 1 cada vez que un fantasma toca al personaje, el juego continuará **mientras vidas** sea mayor a 0. Cuando **vidas** contenga el valor 0, significará que las vidas se acabaron, y también el juego. Por lo tanto, $\text{vidas} > 0$ será la **condición** que determinará que el ciclo repetitivo continúe. Cuando esta condición sea Falsa (es decir, que **vidas** sea igual o menor que 0), el ciclo dejará de funcionar.

Si te acordás algo de inglés, sabrás que la palabra “mientras” se traduce como *While*, y justamente así se llama la estructura repetitiva que vamos a analizar ahora.

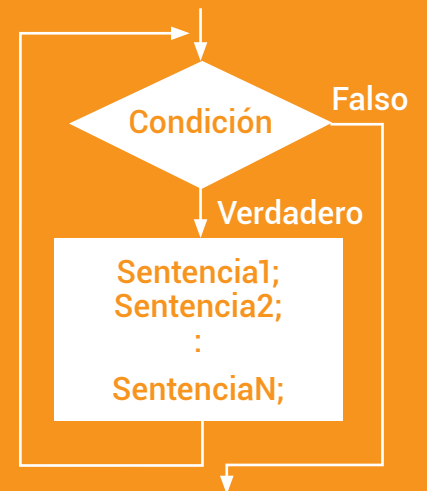
Su símbolo en diagramación es el que te mostramos en la imagen 1

En el diagrama de la estructura de la imagen 1 se puede observar que mientras la condición sea Verdadera, se procesarán las sentencias indicadas, de lo contrario, si la condición es Falsa, saldrá del ciclo repetitivo y continuará con la secuencia de instrucciones fuera del ciclo.

Cabe hacer notar, que si la condición es Falsa de entrada, no se procesarán las sentencias indicadas ni una sola vez.

While es un tipo de estructura repetitiva en donde la condición que repetirá las instrucciones, se encuentra al principio del ciclo.

1





Desempeño 34

| Pensá en alguna situación cotidiana en la que apliques una estructura de este tipo, por ejemplo: *Mientras el semáforo esté en verde, avanzo*, o *Mientras tenga batería en la notebook, pruebo los programas de Java*. Analiza en tu ejemplo, cuál es la condición. Grafícalo como diagrama de flujo.



Contadores

Volvamos una vez más al PacMan.

Suponiendo que tenemos una variable llamada **vidas** que almacena la cantidad de vidas que nos quedan, y otra llamada **puntos**, que guarda el puntaje total obtenido a cada momento: ¿cómo variará su contenido?

Primero, veamos el caso de **vidas**: empezará el juego valiendo 3, y cada vez que un fantasma toque al PacMan, su valor disminuirá en 1 (imagen 1).

Y **puntos**? En este caso, cuando empieza el juego contendrá el valor 0 y por cada pelotita que comamos, su contenido se incrementará en 10.

Los contadores muchas veces se utilizan con la finalidad de contar sucesos o acciones



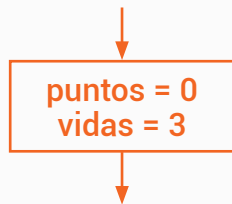
1



Las variables (tales como vidas y puntos en el caso del PacMan), cuyo valor se incrementa o decrementa en una cantidad constante cada vez que se produce un determinado suceso o acción, se denominan **contadores**.

dentro de un bucle. Debemos realizar una operación de **inicialización** (asignación de un valor de inicio) y posteriormente sucesivos **incrementos** o **decrementos** de ese valor inicial, de acuerdo a la ocurrencia de ciertos eventos.

La operación de inicialización **se situará antes y fuera del ciclo repetitivo**. Por ejemplo, para inicializar las variables que planteamos en el PacMan:



Muchas veces los contadores se utilizan para llevar un control del número de ocasiones en que se realiza una operación o se cumple una condición. Los incrementos son generalmente de uno en uno, pero puede haber contadores de dos en dos, tres en tres, etc. La simbología es la de asignación, o sea, un rectángulo, como por ejemplo:



En la primera operación del ejemplo, se establece que a la variable puntos se le debe asignar el valor que resulte de sumar el valor que tenía la misma variable anteriormente, más 10. Por lo tanto, la computadora tomará el valor que tiene la variable **puntos** en ese momento, le sumará 10, y volverá a almacenar el resultado en la misma variable **puntos**. Es decir, que independientemente del valor que tenga la variable **puntos**, cada vez que se procese esta instrucción, se incrementará en 10 su valor. El caso de la variable **vidas** es similar: cada vez que se ejecute la operación indicada, al valor que vidas tenga, se le restará 1 y se le asignará ese nuevo valor.

Cuando el programa recién empieza, **puntos** valdrá 0. Luego, cada vez que PacMan coma una pelotita, sumará 10. La secuencia entonces será 0, 10, 20, 30, etc. En cambio, inicialmente **vidas** contendrá 3, y por cada contacto con los fantasmas, disminuirá su valor en 1, siendo la secuencia: 3, 2, 1, 0.

Muchas veces los contadores se utilizan para llevar un control del número de ocasiones en que se realiza una operación o se cumple una condición.

Los incrementos son generalmente de uno en uno, pero puede haber contadores de dos en dos, tres en tres, etc.

La simbología es la de asignación, es decir, un rectángulo.

A photograph of several cyclists in a race, wearing white and red jerseys and helmets, riding on a road. The image is partially obscured by a red overlay on the left side.

→ Desempeño 35

| Algunos párrafos más arriba dice: "La operación de inicialización **se situará antes y fuera del ciclo repetitivo**. ¿Por qué? ¿Qué ocurriría si a la inicialización la ubicamos dentro del ciclo repetitivo? Si no se te ocurre, realizá el Ejemplo 17 que se encuentra a continuación y después volvé para completar este desempeño.



Continuamos con unos ejemplos...

Ejemplo 17

Listar los números del 1 al 100

Diseña un algoritmo que imprima todos los números del 1 al 100. Observá el diagrama y la codificación en las imágenes 1 y 2 respectivamente.

El diagrama de la imagen 1 comienza por colocar el **valor inicial** del contador en uno, que es el valor desde el cual se quiere comenzar a contar. Luego, se ingresa al ciclo repetitivo con la condición de verificar que el valor de la variable *i* se mantenga por debajo o igual a 100, que es el **valor final** al cual llegará la variable.

Si el valor de la variable *i* es menor o igual a 100, se ingresará al ciclo. La primera vez, la variable *i* tiene el valor 1, por lo tanto, la condición resulta Verdadera y se ingresa al ciclo imprimiendo el valor de la variable *i*, (1 en este caso).

Posteriormente se encuentra el contador, con lo cual se incrementa el valor de la variable en una unidad (pasará a valer 2 en este caso) y

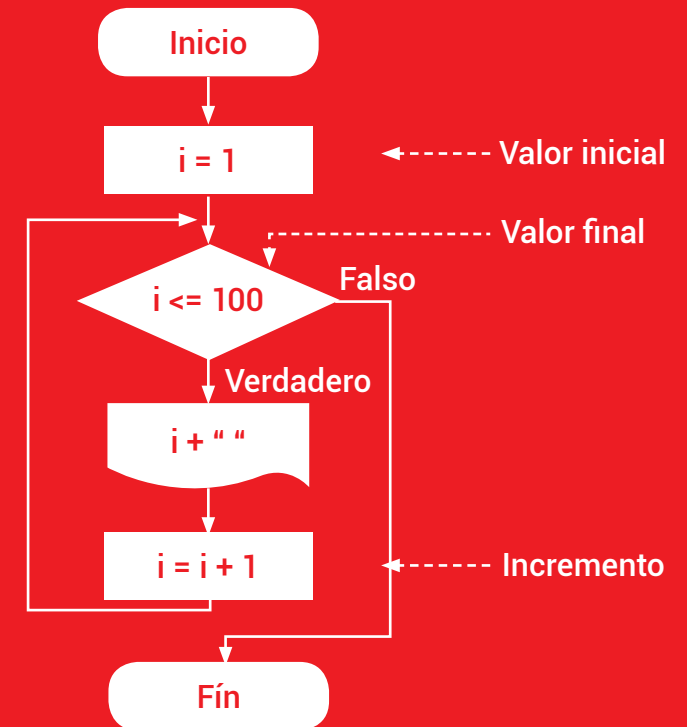
se vuelve a la condición. Ahora se preguntará si el valor de 2 es menor o igual 100, lo que resulta nuevamente Verdadero, imprimiéndose el número 2 en la pantalla.

Luego se incrementa nuevamente en una unidad, volviendo a la verificación de la condición, y así sucesivamente. Es muy común utilizar *i++* en lugar de la línea *i=i+1*; lo que indica que se debe incrementar en una unidad a la variable *i*.

Mientras la variable *i* tenga un valor entre 1 y 100, el programa se mantendrá dentro del ciclo. Cuando la variable *i* alcance el valor de 101, la condición resultará Falsa, y saldrá del ciclo finalizando la ejecución del programa.

En la impresión, se colocó la línea: *i + " "*, la que indica que se debe imprimir el valor que tiene la variable *i* seguido de un espacio en blanco. La única misión de este espacio en blanco, es que no salgan todos los números encimados.

La codificación será la de la imagen 2.



```
import hsa.Console;
class Ejemplo17
{
    static Console c;
    public static void main (String arg [])
    {
        int i;
        c = new Console ();
        i = 1;
        while (i <= 100)
        {
            c.print (i + " ");
            i = i + 1;
        }
    }
}
```

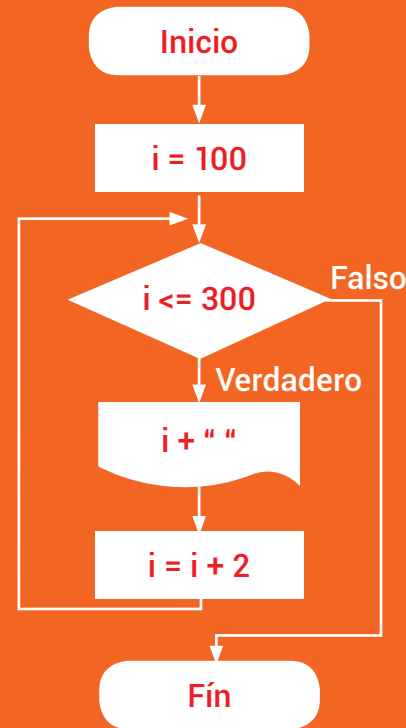
Como te habrás dado cuenta, el uso de ciclos repetitivos permite resolver problemas de una manera muy sencilla. Sin esta estructura, para obtener el mismo resultado hubiéramos tenido que escribir 100 veces las mismas instrucciones.

Ejemplo 18

Sólo los pares

Diseñar un algoritmo que imprima todos los números pares entre el 100 y el 300.

En este caso el **valor inicial** del contador será 100, el valor final será 300 y el **incremento** será de dos en dos. Veamos el diagrama y la codificación en la imagen 1 y 2 respectivamente.



```
import hsa.Console;
class Ejemplo18
{
    static Console c;
    public static void main (String arg [])
    {
        int i;
        c = new Console ();
        i = 100;
        while (i <= 300)
        {
            c.print (i + " ");
            i = i + 2;
        }
    }
}
```


Acumuladores



Supongamos ahora que estamos diseñando un juego en el que deberemos recoger distintos objetos para sumar puntos, pero considerando que hay objetos que otorgan diferentes puntajes.

En casos como éste, en lugar de usar un contador, emplearemos un **acumulador**.

Por ejemplo, habíamos visto que cada vez que PacMan comía una pelotita, sumaba 10 puntos. En ese caso, usamos un contador:

↓
puntos = puntos + 10
↓

Si en cambio hubiese objetos con diferentes valores, en lugar de sumar siempre 10, el incremento pasará a depender de otra variable, cuyo contenido será diferente según sea el objeto recogido:

↓
puntos = puntos + otraVariable;
↓

El principio de funcionamiento de un acumulador es similar al de un contador, sólo que en lugar de incrementarse (o decrementarse) en forma constante, lo hará de manera variable.

A photograph of several cyclists in a race, wearing helmets and jerseys, with a red tint overlay. The image is positioned at the top of the page.

→ Desempeño 36

| Pensá en alguna situación cotidiana en la que puedas aplicar el concepto de **acumulador**. Por ejemplo, *el carnet de conducir pierde una cantidad de puntos que es diferente según sea la gravedad de la infracción cometida*.



Algunos ejemplos de Acumuladores

Vayamos a un ejemplo para que el concepto quede más claro.

Ejemplo 19 Total a pagar

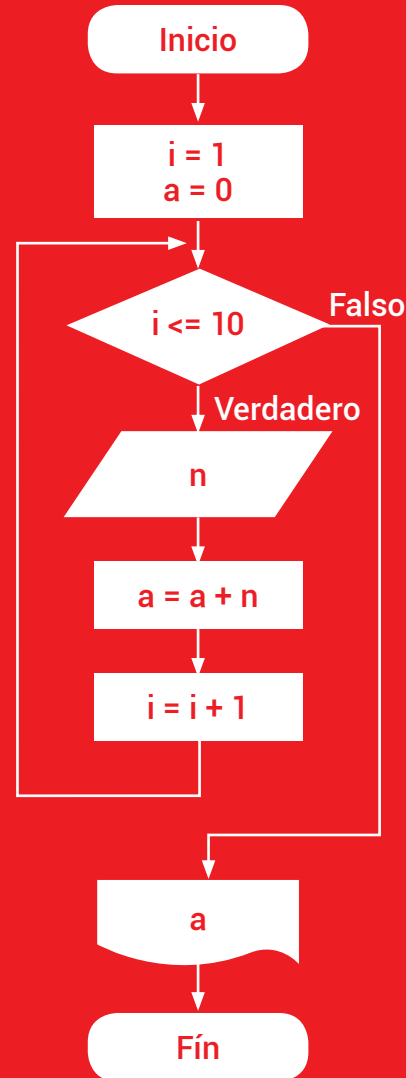
Diseña un algoritmo que permita Ingresar el precio de 10 productos por teclado y que luego muestre la suma de todos ellos.

Observá la imagen 1 y 2 que corresponden al diagrama de flujo y al código, respectivamente.

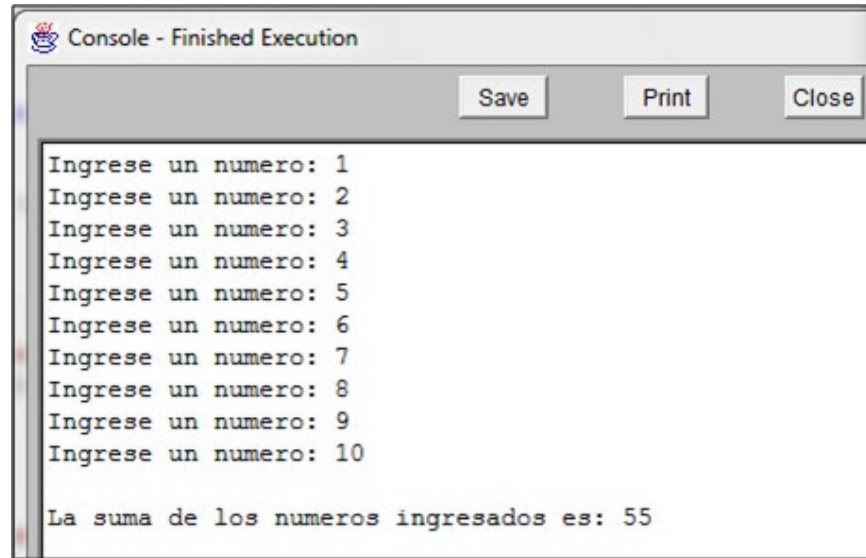
Para este ejemplo, se comienza inicializando el contador en 1 y el acumulador en 0. El contador será el encargado de permitir recorrer el ciclo 10 veces, mientras que el acumulador será el encargado de sumar los 10 números que se ingresen.

Se ingresa al ciclo, **mientras** el valor de la variable **i** sea menor o igual a 10. La primera vez **i** tendrá el valor 1, por lo tanto empezará el ciclo.

A continuación se deberá ingresar el primer



```
import hsa.Console;
class Ejemplo19
{
    static Console c;
    public static void main (String arg [])
    {
        int i, a, n;
        c = new Console ();
        i = 1;
        a = 0;
        while (i <= 10)
        {
            c.print("Ingrese un numero: ");
            n = c.readInt();
            a = a + n;
            i = i + 1;
        }
        c.println();
        c.println("La suma de los numeros
        ingresados es: " + a);
    }
}
```

valor. Este primer valor se almacenará en la variable **n**.

Luego el programa ejecuta la instrucción

a = a + n

Esta línea realiza lo siguiente: busca el valor de **a** (0 en este caso), lo suma al valor de **n** (recién ingresado por el teclado) y lo vuelve a almacenar en la variable **a**.

Luego el contador (la variable **i**) pasa a valer 2 y se ingresa otra vez a la condición, pidiendo otro valor para **n**, que será acumulado nuevamente en la variable **a**.

El proceso se repite hasta que la variable **i**, alcance el valor 11, momento en el cual saldrá del ciclo (ya que la condición **i** <= 10 pasará a ser Falsa).

Por último, se imprime el valor acumulado en **a** (es decir, el total a pagar, la suma de todos los importes parciales).

El programa, será el que se muestra en la imagen 2 de la página anterior.

Al ejecutar el programa y si se ingresan los números del 1 al 10, arrojará la pantalla que podés observar en la imagen 3.

Ejemplo 20

Múltiplos de 3 y 5

Ingresar 12 números por teclado y determinar: cuántos fueron múltiplos de 3, cuántos fueron múltiplos de 5 y cuántos fueron múltiplos de 3 y de 5 al mismo tiempo.

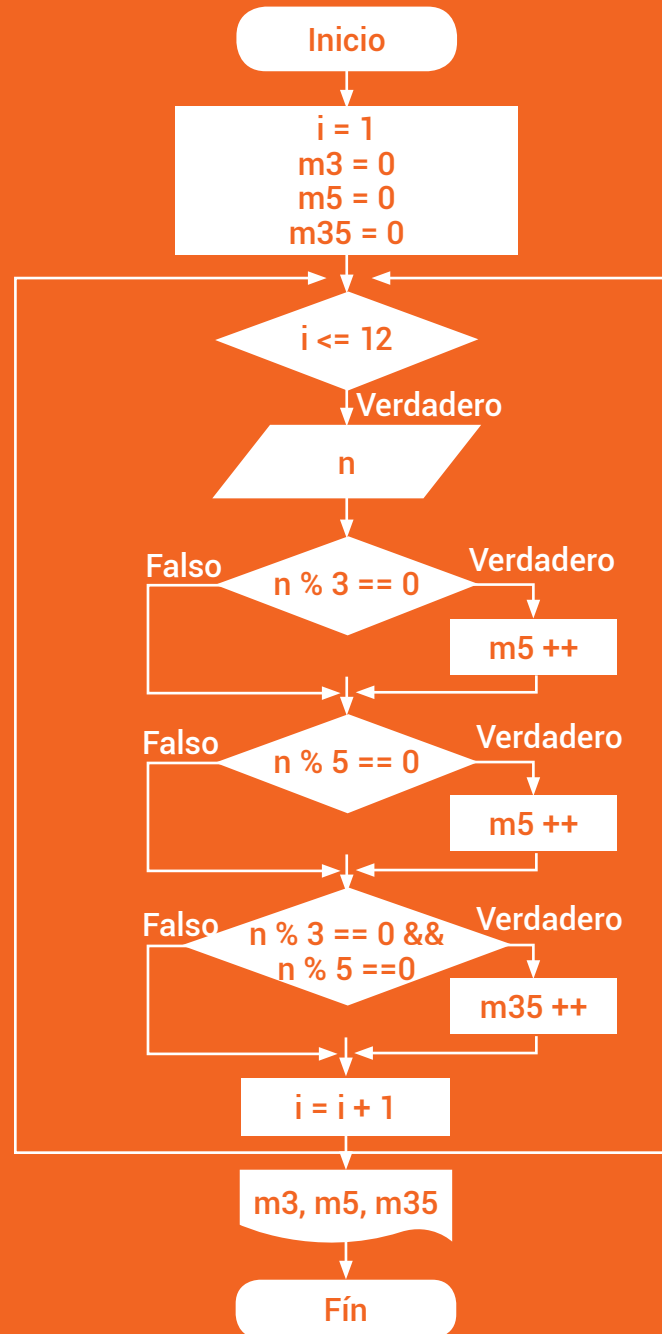
Observá en la imagen 1 y 2 de la siguiente página el diagrama y el código respectivamente.

Para saber si un número es múltiplo de 3, simplemente se toma el número y se lo divide por 3, si la división es exacta, o sea si el resto

es cero, el número en cuestión es múltiplo de 3. Por lo tanto, lo único que interesa saber, es si el número ingresado dividido por 3 da como resto cero. Para ello se utiliza el operador %, que como vimos en la clase anterior, devuelve precisamente el resto de la división de los operandos.

De la misma forma, se averigua si es múltiplo de 5 o de cualquier otro valor. Para saber si es múltiplo de 3 y 5 al mismo tiempo, solo se realiza una condición doble: $(n \% 3 \ \&\& \ n \% 5)$. En la imagen 2 te mostramos el programa correspondiente.

➔ En esta codificación, las instrucciones **if**, fueron utilizadas sin tener en cuenta las llaves limitadoras de principio y fin de bloque. Esto es así debido a que cuando en una estructura, (como por ejemplo en un **if**) se encuentra una sola sentencia, no es necesario colocar las llaves; ahora, si se debe procesar más de una instrucción, tanto por el Verdadero como por el Falso, sí es obligatorio colocar las llaves que limitan el bloque de instrucciones a ejecutar.



```
import hsa.Console;
class Ejemplo20
{
    static Console c;
    public static void main (String arg [])
    {
        int i, m3, m5, m35, n;
        c = new Console ();
        i = 1;
        m3 = 0;
        m5 = 0;
        m35 = 0;
        while (i <= 12)
        {
            c.print ("Ingrese un numero: ");
            n = c.readInt ();
            if (n % 3 == 0)
                m3++;
            if (n % 5 == 0)
                m5++;
            if (n % 3 == 0 && n % 5 == 0)
                m35++;
            i = i + 1;
        }
        c.println ();
        c.println ("La cantidad de multiplos de 3 es: " + m3);
        c.println ("La cantidad de multiplos de 5 es: " + m5);
        c.println ("La cantidad de multiplos de 3 y 5 es: " + m35);
    }
}
```

Casos de aplicación

Caso: Impresión de series matemáticas

Analicemos ahora un ejemplo de aplicación de ciclos, contadores y acumuladores con la intención de agilizar nuestra lógica jugando con los números: la impresión de una serie matemática.

Una serie matemática es una sucesión de términos vinculados por alguna relación, por ejemplo:

3, 5, 7, 9...

Tal como lo expresa la imagen 1, en esta primera serie se obtiene el siguiente término a partir de sumarle 2 al término anterior: $3+2=5$, $5+2=7$ y así sucesivamente, empezando por el

número 3. Los tres puntos finales, indican que la serie continúa.

Otro ejemplo:

1, 2, 4, 7, 11, 16, 22...

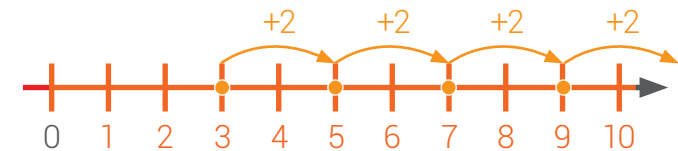
El caso de la imagen 2 es más complejo: cada término surge de sumarle un número de la secuencia de números naturales: $1+1=2$, $2+2=4$, $4+3=7$, etc.

Aprovechemos ahora la potencia de la programación, para crear en Java series matemáticas.

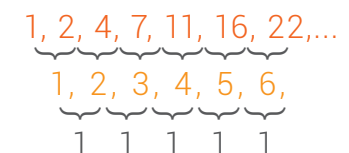
Ejemplo 21 Serie matemática

Imprimir 30 términos de la serie: 10 – 16 – 22 – 28...

1



2

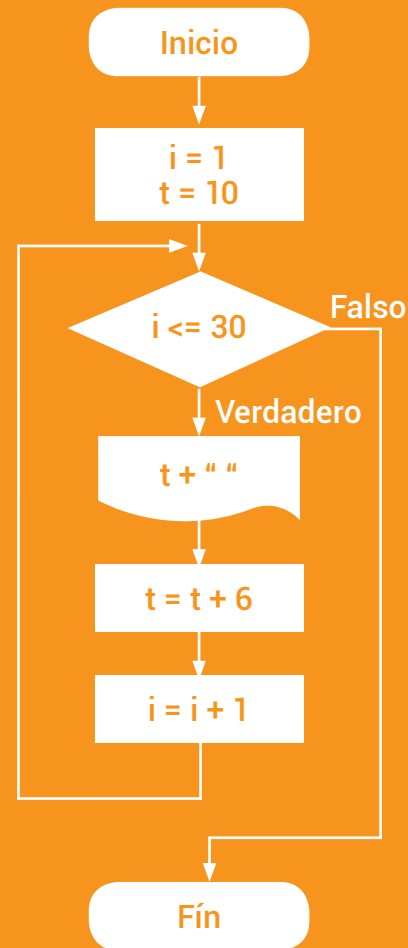


En este ejercicio **no se deben ingresar datos**, sino que el programa automáticamente calculará cada uno de los términos que se van a imprimir.

Cuando se deben imprimir los términos de una serie como esta, primero tenemos que encontrar cuál es la relación entre un elemento y otro. Razonando y observando los términos de la serie a imprimir, se puede deducir que se trata de un **contador** que va de seis en seis y comienza en 10, es decir, que se tiene una instrucción de la forma:
 $t = t + 6$.

Cada término estará formado entonces, como el anterior más seis: $10+6=16$, $16+6=22$, $22+6=28$, etc.

Teniendo ya la fórmula que rige a la serie, el resto simplemente consiste en encerrar a dicha fórmula en un ciclo que se repita 30 veces (que es lo que pide en este caso el enunciado), previa impresión de la variable t . En la imagen 1 te mostramos el desarrollo del algoritmo y en la imagen 2, el código.



```
import hsa.Console;
class Ejemplo20
{
    static Console c;
    public static void main (String arg [])
    {
        int i, t;
        c = new Console ();
        i = 1;
        t = 10;
        c.println("Impresion de la serie: ");
        c.println();
        while (i <= 30)
        {
            c.print(t + " ");
            t = t + 6;
            i = i + 1;
        }
    }
}
```



Desempeño 37

Obtené la suma de todos los números menores o iguales a 100. Es decir, que deberás imprimir la suma total de: $1+2+3+4+ \dots +98+99+100 =$

Desempeño 38

Ingresa 7 valores de tipo float y obtené su promedio indicando si el mismo es mayor o menor que 100.

Desempeño 39

Ingresa 15 valores, e imprimí:

- | Cuántos fueron pares.
- | Cuántos fueron impares.
- | Cuántos terminados en cero.



→ Desempeño 40

→ Desempeño 41

→ Desempeño 42

Ingresá 10 números y mostrá la suma de los cinco últimos.

Imprimí la suma final de 50 términos de la serie:

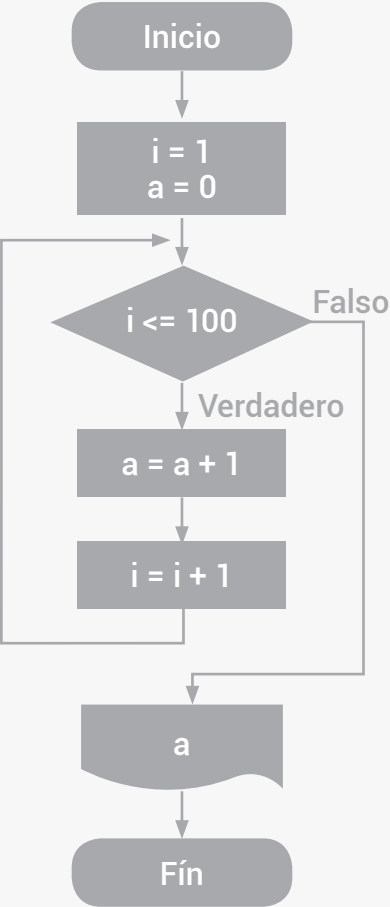
$$5 * 6 + 5 * 7 + 5 * 8 + 5 * 9 + \dots$$

Imprimí 15 términos de la serie:

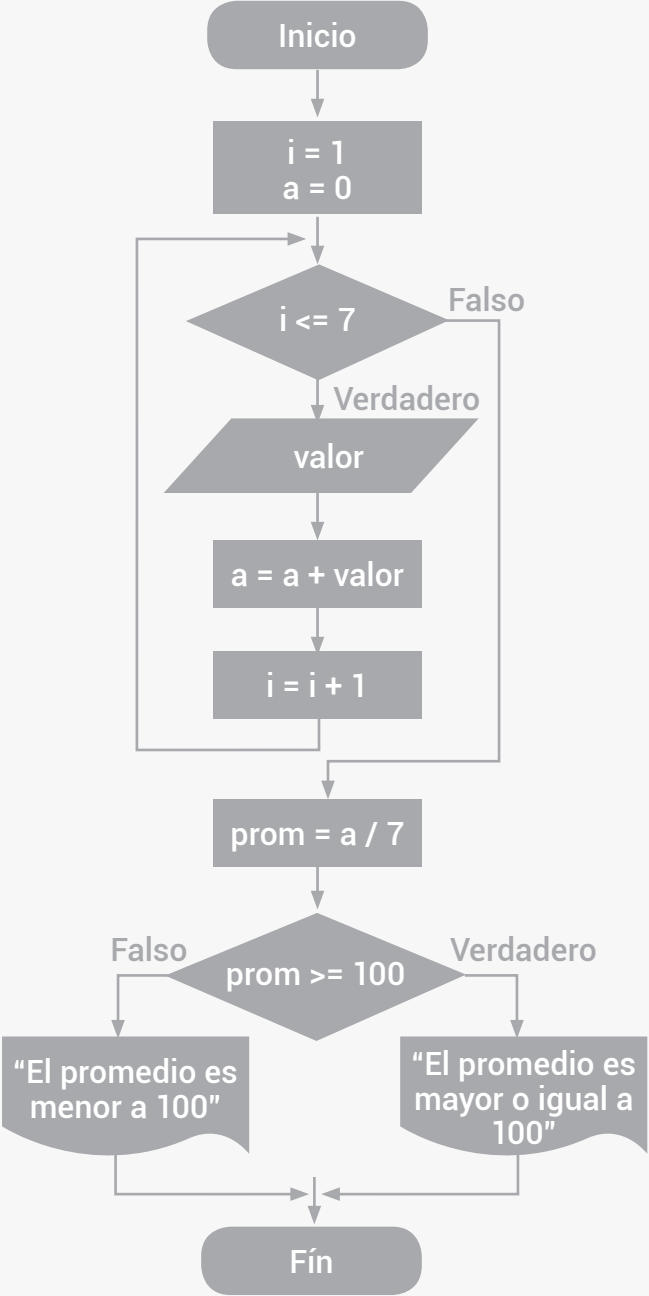
$$500 - 525 - 550 - 575 \dots$$

A continuación te facilitamos los diagramas de flujo de los desempeños anteriores.

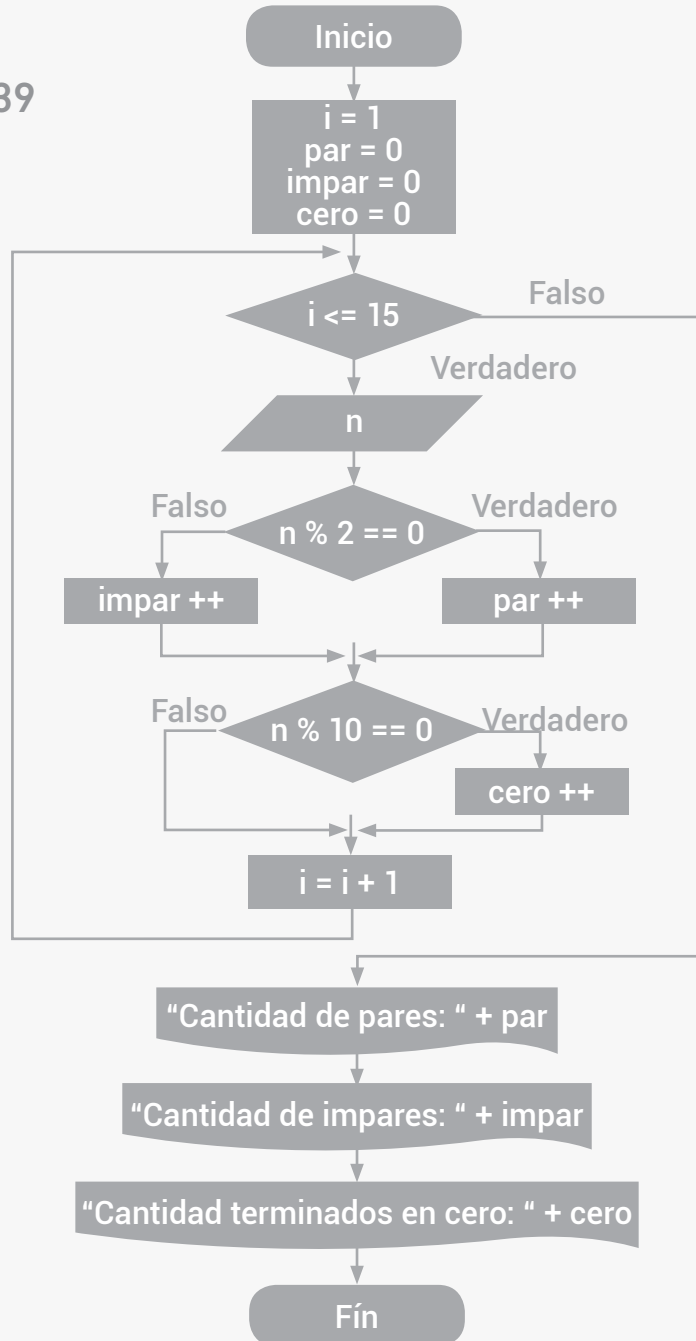
Desempeño 37



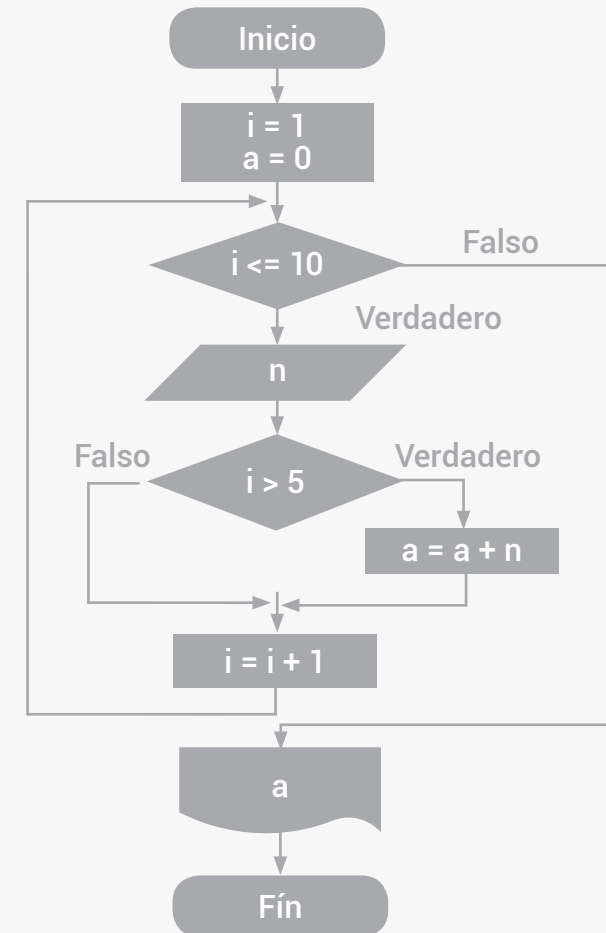
Desempeño 37



Desempeño 39



Desempeño 40



Caso: ingreso de valores hasta que se digite una clave

Supongamos la siguiente situación: en la Caja de una librería, la empleada carga en el sistema los precios de los artículos que el cliente ha elegido, para obtener la suma total a pagar.

Un ciclo repetitivo, como hemos visto, puede servir para este objetivo, acumulando el precio del siguiente artículo en cada vuelta del programa.

Pero un cliente no está obligado a comprar una cierta cantidad de productos, sino que el número puede variar: una persona compra 2 artículos mientras que otra adquiere 20. Entonces, tenemos un problema, ya que no sabemos previamente cuántas vueltas deberá dar el programa, y se nos complica definir la condición.

La solución, es prever un valor clave, que al ser ingresado, le avise al programa que ya están cargados todos los artículos de esa

compra para que muestre el total a pagar. Elegiremos ese valor clave de modo tal que no se confunda con los datos en sí. Veamos un ejemplo:

Ejemplo 22

Carga de una cantidad variable de sueldos

Ingresar sueldos de personas hasta que se digite el valor clave 99999. Determinar: cuántos fueron mayores o iguales a \$5000, cuántos menores de \$5000 y mostrar además, el sueldo promedio de todos los ingresados.



➡ El valor de 99999 es el que indica la finalización del ingreso de datos, por lo tanto este valor no debe ser tenido en cuenta en la cantidad de mayores a \$5000, ni tampoco para el cálculo del promedio.

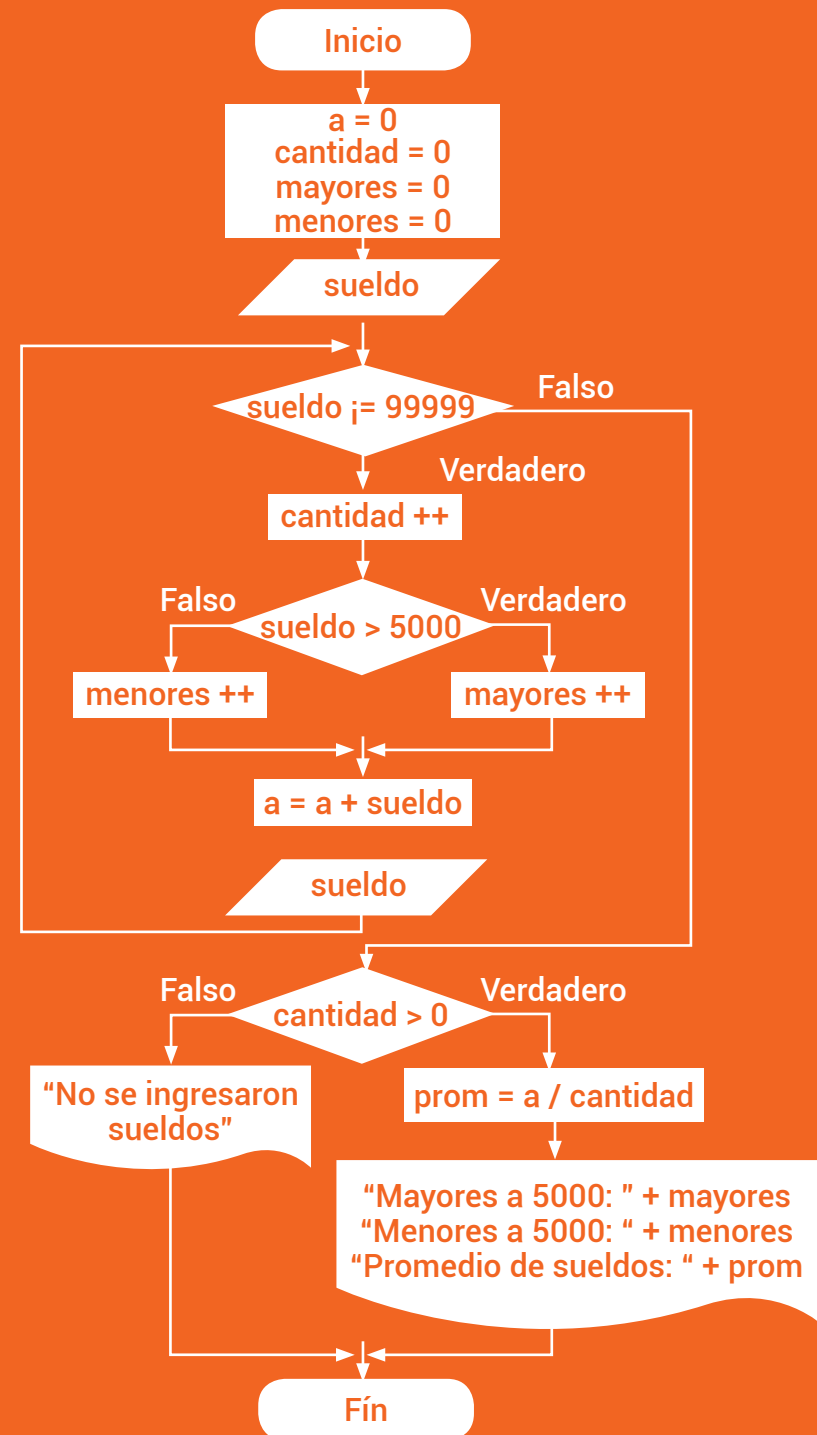
El diagrama que te presentamos en la imagen 1, comienza inicializando las variables en cero, tanto el acumulador (a), como los contadores (*cantidad*, *mayores* y *menores*).

Antes de entrar en el ciclo, se pide el primer sueldo, que en caso de ser distinto de la clave de finalización (99999) se ingresa al ciclo.

Inmediatamente se incrementa la cantidad, porque ya hay un sueldo válido, luego se procesa averiguando si es mayor o menor a \$5000, incrementando los contadores correspondientes. Sucesivamente, se acumulan los sueldos en la variable a, que es donde se guardará la suma total. Antes de cerrar el ciclo, se debe pedir nuevamente el sueldo y retornar a la condición del **while**.

La repetición de la carga de sueldos se realizará hasta que se ingrese el valor 99999, situación que hace que la condición resulta Falsa y se salga del ciclo **while**.

Para poder calcular el promedio de sueldos, es necesario que por lo menos se haya ingresado un sueldo, ya que si de entrada se digita el valor 99999, la variable *cantidad* quedará con el



valor cero y el promedio daría indeterminado. Por lo tanto, se debe verificar que *cantidad* sea mayor a cero, y en ese caso se procede a calcular el promedio y a la impresión de los contadores.

En caso que la cantidad sea cero, se imprime la leyenda correspondiente, indicando que no se han ingresado sueldos para calcular. En la imagen 2, se muestra la codificación respectiva.

```
import hsa.Console;
class Ejemplo22
{
    static Console c;
    public static void main (String arg [])
    {
        int cantidad, mayores, menores;
        float a, sueldo, prom;
        c = new Console ();
        a = 0;
        cantidad = 0;
        mayores = 0;
        menores = 0;

        c.print ("Ingrese un sueldo (99999
para terminar): ");
        sueldo = c.readFloat ();

        while (sueldo != 99999)
        {
            cantidad++;
            if (sueldo >= 5000)
                mayores++;
            else
                menores++;
        }
    }
}
```

```
        a = a + sueldo;
        c.print ("Ingrese un sueldo (99999 para
terminar): ");
        sueldo = c.readFloat ();
    }

    c.println ();
    if (cantidad > 0)
    {
        prom = a / cantidad;
        c.println ("Cantidad mayores o iguales a
$5000: "
            + mayores);
        c.println ("Cantidad menores a $5000: "
            +
            menores);
        c.println ("Promedio de sueldos: " +
prom);
    }
    else
        c.println ("No se ingresaron sueldos");
    }
}
```




→ Desempeño 43

Ingresá 12 precios de productos y determiná:
| Cuantos fueron menores a \$50.
| Cuantos fueron entre \$50 y \$100, ambos inclusive.
| Cuantos fueron mayores de \$100.

→ Desempeño 44

Imprimí todos los múltiplos de 7 menores a 100. (0 – 7 – 14 – 21 – 28...)

→ Desempeño 45

En un Banco se debe ingresar por cada cliente, el número de cuenta y el monto de depósito a realizar. El ingreso de datos se debe realizar hasta que se digite un valor de cuenta negativo. Se deberá imprimir lo siguiente:
| Cantidad de clientes procesados.
| Cantidad de clientes con depósitos superiores a \$2000.
| Suma acumulada de todos los depósitos.
| Promedio de todos los depósitos.



Desempeño 46

Mostrá la suma de los primeros 50 números impares

Desempeño 47

Ingresá dos valores numéricos y mostrá todos los números comprendidos entre ellos. No se deben imprimir los valores ingresados.

Desempeño 48

Dada la siguiente serie: $20 + 25 + 30 + 35 + \dots$ imprimí los 30 primeros términos de la misma y además, mostrá el resultado de la suma. (Tené en cuenta que la serie comienza con el valor 20)



Desempeño 49

Ingresá edades de personas hasta que se digite el valor clave 1111. Deberás imprimir lo siguiente:

| Cuántas personas fueron menores de edad (edad menor de 21 años).

| Cuántas personas fueron mayores de edad.

| Cuántos mayores de edad fueron adultos (edad mayor o igual a 35 años)

Desempeño 50

Realizá un programa que permita cargar el peso de 10 personas y calcule su promedio, imprimiendo las siguientes leyendas:

| Si el promedio es inferior a 55 kg, mostrar "Peso insuficiente"

| Si el promedio está entre 55 kg y 70 kg, mostrar "Peso Ideal"

| Si el promedio está entre 70 kg y 85 kg, mostrar "Peso Normal"

| Si el promedio es superior a 85 kg, mostrar "Peso Excedido"

Desempeño 51

Imprimí los números pares en forma decreciente desde el 800 hasta el 400.

Desempeño 52

Obtené el promedio de los múltiplos de 4 menores a 300.

Bibliografía




Imágenes

www.pexels.com

www.pixabay.com

www.flickr.com

A group of cyclists in white and red jerseys are racing on a track. They are wearing helmets and sunglasses, and their bikes are in motion. The background is a blurred green field.

Como te habrás dado cuenta, el uso de estructuras condicionales sumado a los ciclos repetitivos, despliegan la verdadera potencia y eficiencia de la programación.

En la clase que viene, seguimos agregando complejidad, así que no dejes ninguno de los ejercicios y ejemplos sin resolver.

¡Seguimos avanzando!