Ciclos y métodos (Parte I)

Ciclos

Competencias

- Clasificar los ciclos y sus posibles aplicaciones en la programación.
- Construir programas Java con iteraciones a partir de diagramas de flujo.

Introducción

Los ciclos son estructuras de control que nos permiten repetir la ejecución de una o más instrucciones.

```
Mientras se cumpla una condición:
    Instrucción 1
    Instrucción 2
    Instrucción 3
```

El uso de estructuras de control, como los ciclos, es la clave para crear programas avanzados.

Uso de ciclos

Los posibles usos de ciclos en algoritmos son infinitos, nos permiten recorrer colecciones de datos o espacios de búsqueda.

- Si necesitamos buscar una palabra en un diccionario, debemos recorrer palabra por palabra hasta encontrar la que buscamos.
- Si necesitamos resolver un acertijo, debemos repasar cada una de las pistas hasta lograr obtener el resultado.
- Si necesitamos sumar el monto total de una factura, debemos sumar el costo de ítem a ítem para obtener el monto total.

En esta unidad estudiaremos distintos problemas que se resuelven con ciclos y que, si bien no siempre son utilizados en la industria, nos ayudarán a desarrollar las habilidades lógicas que necesitamos para ser buenos programadores y cuándo utilizarlos correctamente.

Estructuras Iterativas

En esta unidad veremos el uso de distintas estructuras de control que se pueden utilizar en el mundo de Java, las cuáles son las siguientes:

- While
- Do While
- For

Comenzaremos estudiando la sentencia while.

Bloque While

La instrucción while nos permite ejecutar una o más operaciones mientras se cumpla una condición. Su sintaxis es la siguiente:

```
while(condición){
    //Código que se ejecuta
}
```

While paso a paso

1. Se evalúa la condición; si es `true`, ingresa al ciclo.
2. Se ejecutan, secuencialmente, las instrucciones definidas dentro del ciclo.
3. Una vez ejecutadas todas las instrucciones se vuelve a evaluar la condición:

    a. Si se evalúa como `true` : vuelve a repetir.
    b. Si se evalúa como `false` : sale del ciclo.

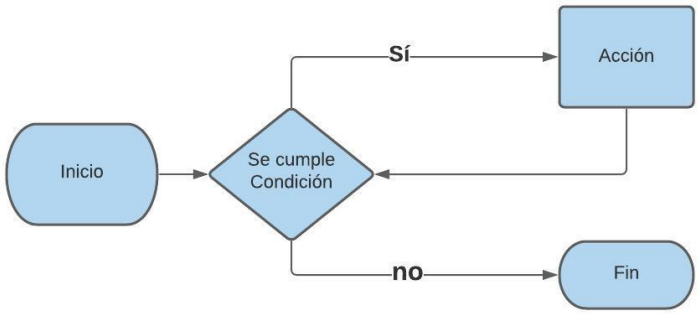

Imagen 1. Diagrama de flujo ciclo While.
Fuente: Desafío Latam.

Salida del ciclo

Anteriormente aprendimos que un algoritmo es una secuencia de pasos finita para resolver un problema. El ciclo while termina cuando alguna de las instrucciones no cumple la condición definida inicialmente.

Validación de entrada de datos utilizando while

Un ejemplo común para comenzar a estudiar los ciclos es validar la entrada de un dato, es decir, que este dato cumpla un criterio. Podemos, por ejemplo, validar que el usuario ingrese un número entre 1 y 10:

```
Scanner sc = new Scanner(System.in);
System.out.printf("Ingresa un número del 1 al 10: ");
int num = sc.nextInt();
while(num <1 || num > 10) {
    System.out.printf("El número no está entre 1 y 10\n");
    System.out.printf("Ingresa un número del 1 al 10: ");
    num = sc.nextInt();
}
System.out.printf("El número ingresado fue: %d \n",num);
```

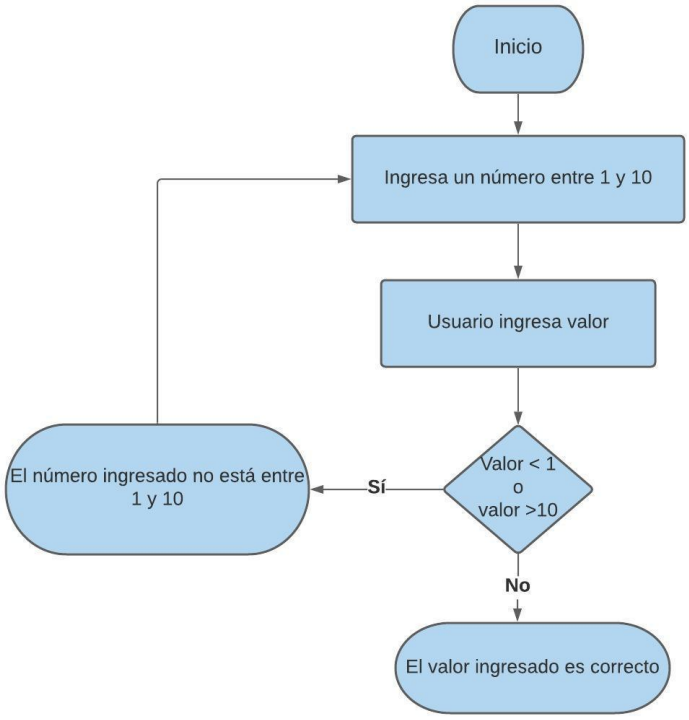

Imagen 2. Diagrama de flujos ingresando un valor entre 1 y 10.
Fuente: Desafío Latam.

Ejercicio de Integración

Podríamos utilizar la misma idea de validación para impedir al usuario entrar, hasta que ingrese la contraseña correcta, para este ejemplo la contraseña será "password".

```
Scanner sc = new Scanner(System.in);
System.out.printf("Ingrese la contraseña: ");
String password = sc.nextLine();
while(password.compareTo("password")!=0) {
    System.out.printf("La contraseña es incorrecta\n", password);
    System.out.printf("Ingrese la contraseña: ");
    password = sc.nextLine();
}
System.out.printf("La contraseña ingresada es correcta\n");
```

Bloque DO While

También existe la sentencia do while, que funciona al inverso del while. En este caso haremos una o más acciones, mientras se cumpla la condición. Las instrucciones se ejecutarán al menos una vez.

```
do{
    //acciones
} while (condición);
```

En el ejemplo de pedir un número del 1 al 10, quedaría de la siguiente manera:

```
int num;
do {
    System.out.printf("ingrese un numero entre 1 y 10:");
    num = sc.nextInt();
}while(num <1 || num > 10);
System.out.printf("El número ingresado es: %d\n",num);
```

Bloque FOR

Este bloque iterativo permite repetir una serie de instrucciones hasta que se cumpla un número determinado de repeticiones. Por lo tanto, el bloque `for` se compone por tres partes:

- Inicio o variable inicializadora.
- Condición, contar mientras la variable inicio sea “menor” o “menor igual a”…
- Cada cuantos pasos se avanza la variable inicializadora.

Lo sintaxis del ciclo `for` es la siguiente:

```
/*
   i: Variable inicializadora
   i<=x: Condición, el ciclo se repetira hasta que i sea mayor a x
   i++: Indica que i se incrementará de 1 en 1
*/
for(int i = 0;i<=x;i++) {
    System.out.println("Instrucciones");
}
```

Existen varias formas de aplicar el bucle `for`, algunas de estas son:

```
//Repite hasta 1 > 10
for(int i = 0;i<=10;i++) {
    System.out.println("Instrucciones 1 en 1");
}
//Repite hasta i > 10, pero salta de dos en dos
for(int i = 0;i<=10;i+=2) {
    System.out.println("Instrucciones 2 en 2");
}
//Repite hasta que i sea menor que cero
for(int i = 10;i>=0;i--) {
    System.out.println("Instrucciones 1 en 1 en reversa");
}
```

Ejercicio guiado: Menú

Podemos implementar de forma sencilla un menú de opciones para el usuario. La lógica es similar a la de la validación de entrada.

- Se muestra un texto con opciones.
- El usuario tiene que ingresar una opción válida -> validación de entrada.
- Si el usuario ingresa la opción `1`, mostramos un texto.
- Si el usuario ingresa la opción `2`, mostramos otro texto.
- Si el usuario ingresa la opción "`salir`" terminamos el programa.

Solución

1. *Diagrama de flujo*

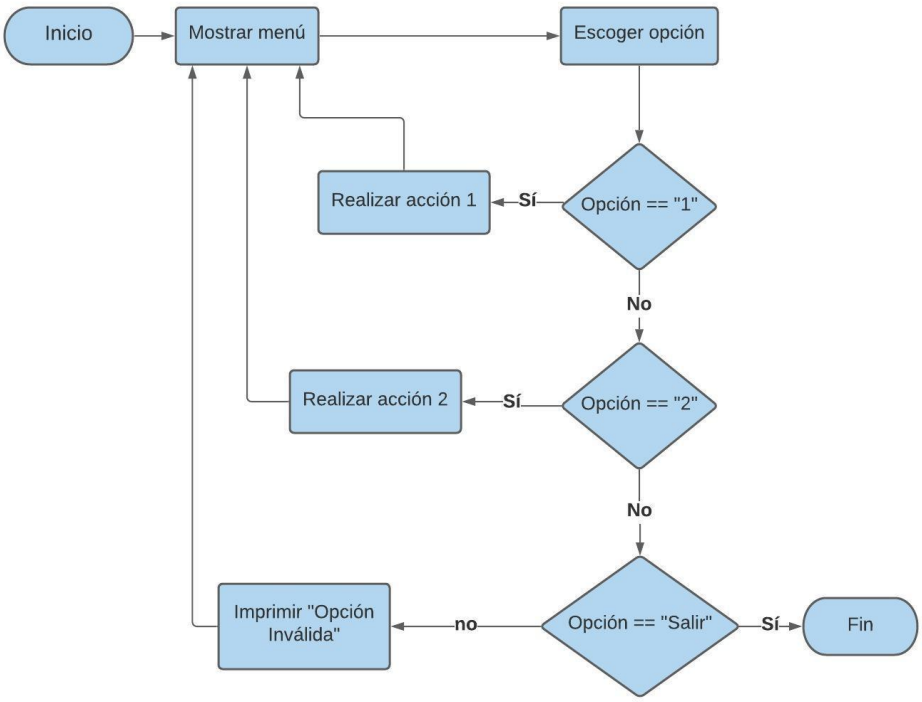

Imagen 3. Diagrama de flujo Solución Menú.
Fuente: Desafío Latam.

2. *Código*

```
Scanner sc = new Scanner(System.in);
String opcion = "";
//.equals devuelve verdadero (True) si son iguales, y falso (False) si los valores a comparar no son iguales
while(!opcion.equals("salir")) {
    System.out.printf("Escoge una opción\n");
    System.out.printf("1 -- Acción 1\n");
    System.out.printf("2 -- Acción 2\n");
    System.out.printf("Escribe 'salir' para terminar el programa\n\n");
    System.out.printf("Ingrese una opción:");
    opcion = sc.nextLine();
    if(opcion.equals("1")) {
        System.out.printf("Realizando acción 1\n");
    } else if(opcion.equals("2")) {
        System.out.printf("Realizando acción 2\n");
    } else if(opcion.equals("salir")) {
        System.out.printf("Saliendo...\n");
    } else {
        System.out.printf("Opción inválida\n");
    }
}
```

Ejercicio propuesto (1)

Escribir un programa que solicite la carga de dos números y nos realice una operación según la opción ingresada por el usuario.

Requerimientos:

1. Solicitar al usuario 2 números.
2. Mostrar un menú con las siguientes opciones:
    a. Sumar.
    b. Restar.
    c. Multiplicar.
    d. Dividir.
    e. Salir.

3. Realizar la operación según la opción ingresada por el usuario.
4. Mostrar en pantalla el resultado.

Ciclos y Contadores

Competencias

- Aplicar los distintos bloques repetitivos.
- Usar los bloques repetitivos para ejecutar operaciones con contadores.

Introducción

Previamente resolvimos ejercicios de ciclos, donde el fin del ciclo estaba determinado por el ingreso de un dato por parte del usuario. A continuación resolveremos problemas que requieren una cantidad determinada de ciclos o iteraciones. Por ejemplo, repetir un ciclo 10 veces, una cuenta regresiva o problemas de sumatorias.

Iterar

Iterar es dar una vuelta al ciclo. Hay muchos problemas que se pueden resolver de forma mucho más eficiente iterando, por ejemplo, contar desde 0 hasta 10.

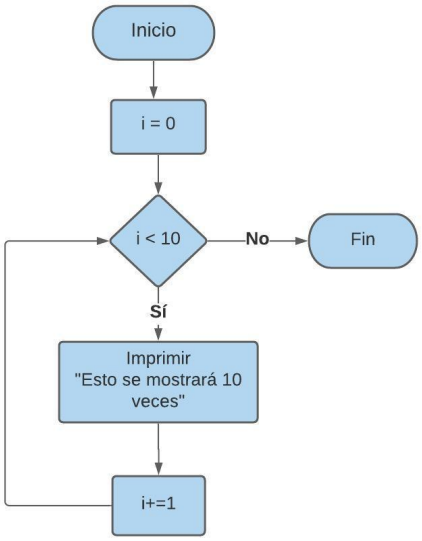

Imagen 4. Diagrama de flujo iterando 10 veces.
Fuente: Desafío Latam.

Contando con While

```
int i = 0;
while (i<10) {
    System.out.printf("Esto se mostrará 10 veces\n");
    i += 1 ; //IMPORTANTE Java
}
```

La instrucción `System.out.printf ("Esto se mostrará 10 veces\n")` se repetirá hasta que la variable `i` alcance el valor 10. Para entonces, la comparación de la instrucción `while` se evaluará como `false` y saldremos del ciclo.

En programación es una convención ocupar una variable llamada `i` como variable de iteración para operar en un ciclo.

Importante: Si no aumentamos el valor de la variable `i` entonces nunca llegará a ser igual o mayor a 10.

Que significa i+=1;

`+=` es un operador de asignación, muy similar a decir `a = 2` pero la diferencia es que con `+=` estamos diciendo "el valor anterior más 1".

```
a = 2; // aquí se asigna el valor 2 a la variable a
a += 2; /*aquí el valor de la variable aumentó en dos, y fue almacenada
nuevamente en a. Es decir ahora a = 4*/
a = a+2; //esto equivale a la sentencia anterior.
```

Transformando do a do while

Si bien con la sentencia while podemos mostrar 10 veces un mismo mensaje, también podemos usar el operador do while.

```
int i = 0;
do{
    System.out.printf("Esto se mostrará 10 veces\n");
    i += 1 ; //Importante
} while (i<10) ;
```

En este caso, se realizará al menos una vez, la operación de mostrar por pantalla y sumar 1 a la variable `i`, mientras `i` sea menor que 10. Una vez que `i` toma el valor 10, ya no se seguirá repitiendo el ciclo.

Importante: i += 1 es un incremento de 1 en 1. En este caso también se podría escribir `i++`; lo cual incrementará de 1 en 1 la variable `i`.

Operadores de asignación

La siguiente tabla muestra el comportamiento de los operadores de asignación.

Operador	Nombre	Ejemplo	Resultado
=	Asignación	a = 2	a toma el valor de 2
+=	Incremento y asignación	a += 2	a es incrementado en 2 y asignado el valor resultante
-=	Decremento y asignación	a -= 2	a es reducido en 2 y asignado el valor resultante
++	Incremento de 1 en 1	a++	Incrementa o suma 1 a la variable a
--	Decremento de 1 en 1	a--	Reduce o resta 1 a la variable a
*=	Multiplicación y asignación	a *= 3	a es multiplicado por 3 y asignado el valor resultante
/=	División y asignación	a /= 3	a es dividido por 3 y asignado el valor resultante

Tabla 1: Operadores de asignación.
Fuente: Desafío Latam.

Ejercicio guiado: La bomba de tiempo

Crearemos un algoritmo sencillo que realice una cuenta regresiva de 5 segundos.

Contar de forma regresiva es muy similar, solo debemos comenzar desde el valor correspondiente e ir disminuyendo su valor de uno en uno.

```
int i = 5;
while (i > 0) { //cuando lleguemos a cero terminamos.
    System.out.printf("%d\n",i);
    i--; //en cada iteración descontamos 1
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Para hacer que el programa espere de a 1 segundo antes de ejecutar otra sentencia, debemos agregar el paquete de `import java.util.concurrent.TimeUnit;` al inicio del programa y se utilizará el método `TimeUnit.SECONDS.sleep(1);` donde el parámetro 1 indica que la espera será de un segundo.

Incursionando en java: Try Catch

En Java los errores en tiempo de ejecución se denominan **excepciones**, y esto ocurre cuando se ejecuta un error en alguna de las instrucciones de nuestro programa. Por ejemplo, cuando se hace una división por cero o se abre mal un fichero.

Cuando se levanta una excepción, el programa se detiene en ese instante y no continua ejecutando las otras instrucciones. En este caso, se crea un objeto de una determinada clase (dependiendo del error), que nos proporcionará información sobre ese error. Todas estas clases tienen como clase padre **Throwable** (más adelante entraremos más en detalle sobre herencia, clases padres).

Java tiene un buen control de excepciones, evitando así que se detenga inesperadamente nuestro programa y, aunque se produzca una excepción, se siga ejecutando. Para ello tenemos la siguiente estructura:

```
try {
    // Instrucciones cuando no hay una excepción
} catch (TypeException ex) {
    // Instrucciones cuando se produce una excepción
} finally {
    // Instrucciones que se ejecutan, tanto si hay como si no hay
excepciones
}
```

En un bloque de código try-catch se puede omitir el catch o finally, pero no ambos.

En el ejercicio guiado: La bomba de tiempo

El método `TimeUnit.SECONDS.sleep(1);`

Si tratamos de ejecutar el método sin utilizar `try catch`, nos mostrará el siguiente error:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
Unhandled exception type InterruptedException
```

Ya que este método requiere control de excepciones, por ello se escribe de la siguiente manera.

```
try {
    TimeUnit.SECONDS.sleep(1);
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Ejercicio propuesto (2)

Crear un temporizador de acuerdo a un número ingresado por el usuario:

Requerimientos:

- Solicitar al usuario el ingreso de un número.
- Contar de forma regresiva desde el número ingresado.

Ciclos y sumatorias

Competencias

- Aplicar diagramas de flujos para resolver problemas con sumatorias.
- Distinguir la diferencia entre un contador y un acumulador.

Introducción

Aprenderemos a crear programas que, además de utilizar ciclos, operen sobre otra variable. Esta estructura nos permite resolver diversos tipos de problemas, la mayoría de ellos corresponde a problemas matemáticos. Sin embargo, el entendimiento de estos contenidos nos otorgará las habilidades necesarias para aplicar esta estructura a otros contextos.

Sumatorias

Para algunos -que no gustan de las matemáticas- el término sumatoria puede sonar algo intimidante. Sin embargo, para resolver una sumatoria solo necesitamos saber una cosa: sumar.

La sumatoria consiste en sumar todos los números de una secuencia. Por ejemplo: sumar todos los números entre 1 y 100. Esto no solo sirve para resolver ecuaciones matemáticas, sino también para que generemos las habilidades de abstracción necesarias para resolver diversos problemas.

Sumando de 1 a 100

$$1 + 2 + 3 + \ldots + 99 + 100 = ?$$

Resolver esto es muy similar a contar las cien veces, pero además de contar, vamos a ir guardando la suma **en cada iteración.**

```
int i = 0;
while (i < 100){
    i+=1;
}
```

En el código anterior, tenemos una variable `i` que aumenta de 1 en 1 en cada iteración. Ahora necesitamos ir guardando la suma de cada de uno de los números que vamos obteniendo de `i`.

Para ello crearemos una variable antes de entrar al ciclo para guardar esta suma, inicializando la variable en cero.

```
int i = 0;
int suma = 0; //Variable que guardará el resultado de la suma
while (i < 100){
    i+=1;
}
```

Si queremos sumar desde el valor 1 al 100, y como `i` parte en cero, debemos primero aumentar su valor en 1, y luego sumar.

```
int i = 0;
int suma = 0; //Variable que guardará el resultado de la suma
while (i < 100){
    i+=1;
    suma += i; //La variable suma, guarda el valor de suma + i
}
System.out.println(suma)
```

Por último, el diagrama de flujo de la solución, sería el siguiente:

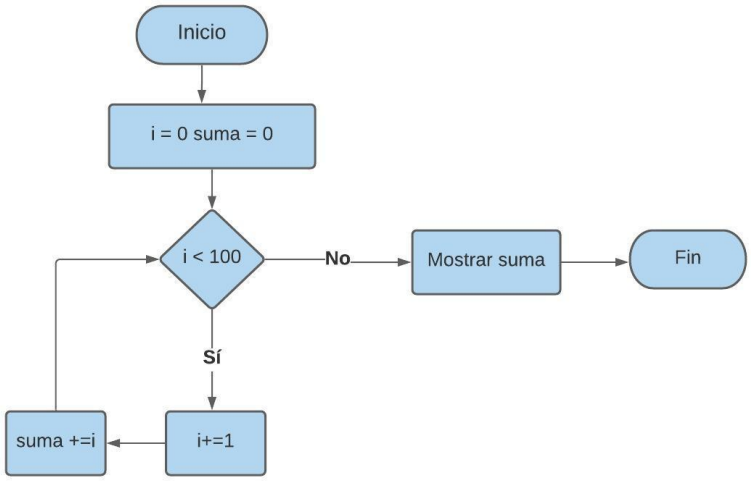

Imagen 5. Diagrama de flujo sumando hasta 100.

La instrucción `suma += i` es la encargada de aumentar el valor de la variable suma en cada iteración.

Este comportamiento, sumar y almacenar secuencialmente valores de variables, se conoce como la implementación de un **acumulador.** Un acumulador se diferencia de un contador, solo por el hecho que el acumulador se le suma un valor a la misma variable, y el contador aumenta su valor por un valor constante.

```
i = i + 1 // Contador
acum = acum + i //Acumulador
i++; //Cuenta de 1 en 1
i+=acum; //Acumula a i por el valor que tenga la variable acum
```

Ejercicio guiado: Suma de números

Contexto

Crear el programa `Suma.java`, donde el usuario ingresa un número, se muestra la suma de todos los números de 1 hasta ese número.

Ejemplo: si n = 100, el resultado es 5050.

1. Se debe crear el programa ejecutable main, donde se itere desde cero hasta el número ingresado por el usuario. En cada iteración se debe sumar cada número del conteo con el resultado anterior.

```
public class Ciclos {
    public static void main(String[] args) {
        //Se lee el valor ingresado por el usuario
        System.out.println("Ingrese un número");
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        //i es la variable que contará hasta el número ingresado
        int i = 0;
        //suma es la variable que irá guardando el resultado
        int suma = 0;
        while (i < n){
            i+=1;
            suma += i;
        }
        System.out.printf("%d\n",suma);
    }
}
```

Ejercicio propuesto (3):

Contexto

Crear el programa `SumaPar.java` donde se sumen únicamente los números pares dentro del ciclo entre 1 y un número ingresado por el usuario.

Ejemplo: si n = 100, el resultado debe ser 2550

1. Siguiendo con el programa del ejercicio anterior, se debe agregar la condición en base a la variable `i`, para que se realice la suma siempre y cuando `i` sea par.

Soluciones Ejercicios propuestos

Solución Ejercicio propuesto (1)

Escribir un programa que solicite la carga de dos números y nos realice una operación según la opción ingresada por el usuario.

Requerimientos:

1. Solicitar al usuario 2 números.

```
Scanner sc = new Scanner(System.in);
System.out.printf("Ingresa el primer número: \n");
int numero1 = sc.nextInt();
System.out.printf("Ingresa el segundo número: \n");
int numero2 = sc.nextInt();
```

2. Mostrar un menú con las siguientes opciones:
    a. Sumar.
    b. Restar.
    c. Multiplicar.
    d. Dividir.
    e. Salir.

3. Realizar la operación según la opción ingresada por el usuario.
4. Mostrar en pantalla el resultado.

```
// Declaramos la variable resultado en 0
int resultado = 0;
if(opcion.equals("1")) {
        resultado = numero1 + numero2;
        System.out.printf("El resultado de la suma es" + resultado);
    } else if(opcion.equals("2")) {
        resultado = numero1 - numero2;
        System.out.printf("El resultado de la resta es" + resultado);
    } else if(opcion.equals("3")) {
        resultado = numero1 * numero2;
        System.out.printf("El resultado de la multiplicación es" +
resultado);
    } else if(opcion.equals("4")) {
        resultado = numero1 / numero2;
        System.out.printf("El resultado de la división es" + resultado);
    }
      else if(opcion.equals("salir")) {
        System.out.printf("Saliendo...\n");
    } else {
        System.out.printf("Opción inválida\n");
    }
```

Finalmente el programa completo quedaría de la siguiente manera:

```
public class Calculadora {
    public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.printf("Ingresa el primer número: \n");
       int numero1 = sc.nextInt();
       System.out.printf("Ingresa el segundo número: \n");
       int numero2 = sc.nextInt();

       String opcion = "";
       while(!opcion.equals("salir")) {
          System.out.printf("Escoge una opción\n");
```

```
          System.out.printf("1 -- Sumar \n");
          System.out.printf("2 -- Restar \n");
          System.out.printf("3 -- Multiplicar \n");
          System.out.printf("3 -- Dividir \n");
          System.out.printf("Escribe 'salir' para terminar el
programa\n\n");
          System.out.printf("Ingrese una opción:");
          opcion = sc.nextLine();

          // Declaramos la variable resultado en 0
          int resultado = 0;
          if(opcion.equals("1")) {
             resultado = numero1 + numero2;
             System.out.printf("El resultado de la suma es" + resultado);
          } else if (opcion.equals("2")) {
               resultado = numero1 - numero2;
               System.out.printf("El resultado de la resta es" +
resultado);
          } else if (opcion.equals("3")) {
               resultado = numero1 * numero2;
               System.out.printf("El resultado de la multiplicación es" +
resultado);
          } else if (opcion.equals("4")) {
               resultado = numero1 / numero2;
               System.out.printf("El resultado de la división es" +
resultado);
         } else if(opcion.equals("salir")) {
              System.out.printf("Saliendo...\n");
         } else {
              System.out.printf("Opción inválida\n");
           }
       }
    }
}
```

Solución Ejercicio propuesto (2)

Crear un temporizador de acuerdo a un número ingresado por el usuario:

Requerimientos:

- Solicitar al usuario el ingreso de un número.

```
Scanner sc = new Scanner(System.in);
System.out.printf("Ingresa un número: \n");
int numero = sc.nextInt();
```

- Contar de forma regresiva desde el número ingresado.

```
while (numero > 0) { //cuando lleguemos a cero terminamos.
    System.out.printf("%d\n",numero);
    numero--; //en cada iteración descontamos 1
    try {
        TimeUnit.SECONDS.sleep(1);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

El programa completo quedaría así:

```
   import java.util.concurrent.TimeUnit;

public class Temporizador {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.printf("Ingresa un número: \n");
        int numero = sc.nextInt();

        while (numero > 0) { //cuando lleguemos a cero terminamos.
           System.out.printf("%d\n",numero);
           numero--; //en cada iteración descontamos 1
           try {
              TimeUnit.SECONDS.sleep(1);
           } catch (InterruptedException e) {
               e.printStackTrace();
           }
        }
    }
}
```

Solución Ejercicio propuesto (3)

Crear el programa `SumaPar.java` donde se sumen únicamente los números pares dentro del ciclo entre 1 y un número ingresado por el usuario.

Ejemplo: si n = 100, el resultado debe ser 2550.

2. Siguiendo con el programa del ejercicio anterior, se debe agregar la condición en base a la variable `i`, para que se realice la suma siempre y cuando `i` sea par.

```
public class Ciclos {
    public static void main(String[] args) {
        //Se lee el valor ingresado por el usuario
        System.out.println("Ingrese un número");
        Scanner sc = new Scanner(System.in);
        int numero = sc.nextInt();
        //i es la variable que contará hasta el número ingresado
        int i = 0;
        //suma es la variable que irá guardando el resultado
        int suma = 0;
        while (i < numero){
            i+=1;
            if(i%2 == 0) { // es par
                suma += i;
            }
        }
        System.out.printf("%d\n",suma);
    }
}
```