

## Ciclos y métodos (Parte II)

### Dibujando patrones con ciclos

#### Competencias

- Reconocer patrones de repetición en un ciclo.
- Implementar ciclos para dibujar patrones.

#### Introducción

Una de las dificultades más frecuentes en principiantes al momento de resolver problemas de ciclos es la de identificar/entender el patrón. No entender un patrón de manera rápida e instantánea no nos hace menos inteligentes. A pesar de que hay personas que pueden resolver estos problemas de manera intuitiva, la mayoría de nosotros tuvo que aprender a resolverlos.

Resolveremos problemas desde los más sencillos a complejos identificando el patrón.

#### Recomendación

Antes de revisar los problemas a continuación, trata de resolverlos por tu cuenta para así entrenar el pensamiento lógico.

## Dibujando patrones

A continuación, dibujaremos por pantalla una serie de patrones utilizando principalmente ciclos e iteraciones en Java.

### Dibujando puntos

Crear un programa `SoloPuntos.java` que dibuje `n` puntos. Donde `n` es un valor ingresado por el usuario al ejecutar el programa.

Si por ejemplo `n = 5`, debe mostrar:

```
*****
```

### Solución 1

El primer intento para resolverlo podría ser utilizando instrucciones `if`:

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
if(n == 1) System.out.printf("*\n");
else if( n == 2) System.out.printf("**\n");
else if( n == 3) System.out.printf("***\n");
else if( n == 4) System.out.printf("****\n");
else if( n == 5) System.out.printf("*****\n");
```

Sin embargo, la solución es bastante limitada. ¿Qué sucedería si el usuario ingresa el valor 6?, ¿o 7?, ¿o 100?, ¿Vamos a programar todas las opciones hasta 100? Este tipo de problemas se resuelve mucho mejor con ciclos.

## Solución 2

Para resolver el problema con ciclos debemos, simplemente, identificar el patrón.

- Si el usuario ingresa 1, se dibuja un asterisco
- Si el usuario ingresa 2, se dibujan 2 asteriscos.
- Si el usuario ingresa  $n$ , se dibujan  $n$  asteriscos.

### *Solución 2: Análisis*

Para utilizar un ciclo **for**, debemos definir las 3 partes del ciclo que corresponden a la variable de iteración, la condición de término y el incremento.

- Variable de iteración:  $i$ , con valor de inicio 0.
- Condición de término:  $i < n$ , cuando  $i$  toma el valor de  $n-1$ , se ejecutará la última iteración.
- Incremento: Como se dibujarán de a 1 asterisco, el incremento debe ser de 1 en 1.

Resultando el siguiente código:

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
for(int i=0;i<n;i++) {
    System.out.printf("*");
}
System.out.printf("\n");
```

## Ejercicio guiado: Dibujando asteriscos y puntos

Crear el programa `AsteriscosYPuntos.java` que dibuje asteriscos y puntos intercalados, hasta  $n$ . Donde  $n$  es un valor ingresado por el usuario. Por ejemplo, si el usuario ingresa:

- 3

```
*.*
```

- 4

```
*.*.*
```

- 5

```
*.*.*.*
```

Es muy similar al planteamiento del problema anterior, ya que debemos dibujar  $n$  elementos, independiente de cual sea la figura que debemos dibujar.

- Si el usuario ingresa 1, dibujamos 1 elemento.
- Si el usuario ingresa 2, dibujamos 2 elementos.
- Si el usuario ingresa  $n$ , dibujamos  $n$  elementos.

Ahora debemos analizar el patrón:

- Las posiciones impares (1, 3, 5, ...,  $2N+1$ ) son puntos.
- Las posiciones pares (0, 2, 4, 6, ...,  $2N$ ) son asteriscos.

Recordemos que con el operador módulo `%` podemos obtener el resto al dividir 2 números.

Al aplicar el operador %2:

- Si resta 0 este número será par.
- Si resta 1, será impar.

```
3%2 // => 1 impar  
4%2 // => 0 par
```

Con el análisis previo ya podemos crear nuestro código.

**Paso 1:** Solicitamos al usuario el ingreso de un número:

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

**Paso 2:** Creamos el ciclo **for** junto a la variable de iteración: **i**, con valor de inicio 0.

```
for(i=0; i<n; i++) {  
    // Código a ejecutar  
}
```

La condición de término es  $i < n$ , cuando **i** toma el valor de **n-1**, se ejecutará la última iteración.

Como se dibujarán de a 1 elemento (asterisco o punto), el incremento debe ser de 1 en 1.

**Paso 3:** Utilizar el un **if** junto al operador módulo **%** dentro del ciclo, para decidir si se imprime un punto o un asterisco.

```
for(i=0; i<n; i++) {  
    /*  
        Condición que evalúa qué número es par o no. Al ser par es un  
        * e impar un . (punto). Nótese que las llaves de apertura y cierre no se  
        escriben en el código, ya que el if solo tiene una línea por bloque.  
    */  
    if(i%2==0)  
        System.out.printf("*");  
    else  
        System.out.printf(".");  
}
```

Resultando el código:

```
Scanner sc = new Scanner(System.in);  
System.out.printf("Ingresa un número: ");  
int n = sc.nextInt();  
int i;  
for(i=0;i<n;i++) {  
    /*  
        Condición que evalúa qué número es par o no. Al ser par es un  
        * e impar un . (punto). Nótese que las llaves de apertura y cierre no se  
        escriben en el código, ya que el if solo tiene una línea por bloque.  
    */  
    if(i%2==0)  
        System.out.printf("*");  
    else  
        System.out.printf(".");  
}  
System.out.printf("\n");
```

## Ciclos anidados

### Competencias

- Implementar ciclos anidados para resolver problemas.

### Introducción

Un ciclo anidado no es nada más que un ciclo dentro de otro ciclo. No existe ningún límite en torno a cuántos ciclos pueden haber anidados dentro de un código, aunque por cada uno aumentará la complejidad temporal del programa. Los ciclos anidados expanden el universo de los tipos de problemas que podemos resolver.

## Escribiendo tablas de multiplicar

A modo de ejemplo, para poner en práctica algunos ciclos anidados, vamos a utilizar un patrón conocido como lo son las tablas de multiplicar.

### Tabla de un número

Supongamos que queremos mostrar una tabla de multiplicar. Por ejemplo, la tabla del número 5.

```
for(int i=0;i<11;i++) {  
    System.out.printf("5 * %d = %d\n", i,5*i);  
}
```

¿Cómo podríamos hacer para mostrar todas las tablas de multiplicar del 0 al 10? ¡Fácil! Envoltiendo el código en otro ciclo que itere de 0 a 11.

### Tabla de todos los números

```
for(int j=0;j<11;j++) {  
    for(int i=0;i<11;i++) {  
        System.out.printf("%d * %d = %d\n",j, i,j*i);  
    }  
}
```

### Resultado

```
0 * 0 = 0  
0 * 1 = 0  
0 * 2 = 0  
0 * 3 = 0  
...  
...
```



```
...  
9 * 6 = 54  
9 * 7 = 63  
9 * 8 = 72  
9 * 9 = 81
```

## Dibujando con ciclos anidados

A continuación, nos centraremos en profundizar el uso de los ciclos y ciclos anidados, analizando distintos patrones para resolverlos de la manera adecuada. Los ejercicios que analizaremos son de ir dibujando patrones, ya que son ideales para practicar ciclos y ciclos anidados.

### Ejercicio 1 - Cuadrado

Veamos el siguiente patrón:

```
*****  
*****  
*****  
*****  
*****
```

Si el usuario ingresa por ejemplo el 5, se dibujará un cuadrado de 5\*5, donde cada elemento es un asterisco.

## Solución

De manera muy similar cuando debíamos dibujar una línea con los asteriscos, ahora debemos extender esta línea a un plano.

Recordemos:

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
for(int i=0;i<n;i++) {
    System.out.printf("*");
}
System.out.printf("\n");
```

Ahora, para extender `n` veces, debemos incorporar un nuevo ciclo for:

```
int n = sc.nextInt();
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        System.out.printf("*");
    }
    System.out.printf("\n");
}
```

## Ejercicio guiado: Escalera

Ahora veamos este patrón:

```
*
**
***
****
*****
```

### Solución

Para resolver el código tenemos que observar el patrón:

- Si el usuario ingresa 5, se dibujan 5 filas.
- En la fila 1, hay 1 asterisco.
- En la fila 2, hay 2 asteriscos.
- Por lo que podemos decir que en la fila  $n$  hay  $n$  asteriscos.

**Paso 1:** Primer acercamiento a la solución, dibujaremos las  $n$  filas

```
for(int i = 0; i < n; i++) {
    System.out.printf("*\n");
}
```

**Paso 2:** Dibujamos las columnas

```
for(int j = 0; j <= i; j++) {
    System.out.printf("*");
}
System.out.printf("\n");
/*Si estamos en la fila j = 3, correspondiente a la 4ta fila (las j parten
en cero), debemos dibujar 4 asteriscos. Por ende la condición de borde
queda dada
por i <=j*/
```

Ahora, juntando lo anterior quedaría:

```
Scanner sc = new Scanner(System.in);
System.out.printf("Ingresa un numero");
int n = sc.nextInt();
for(int i = 0; i < n; i++) {
    for(int j = 0; j <= i; j++) {
        System.out.printf("*");
    }
    System.out.printf("\n");
}
```

### Ejercicio Propuesto (1)

Siguiendo el ejercicio guiado, debemos utilizar y modificar el algoritmo anterior para imprimir el siguiente patrón, el límite será indicado por un valor ingresado por el usuario:

```
1
12
123
1234
12345
123456
```

## Métodos en Java

### Competencias

- Comprender qué es un método para implementarlos en el lenguaje Java.
- Hacer uso de métodos para dividir y reutilizar códigos en Java.

### Introducción

Hasta ahora hemos visto la implementación de algoritmos en Java para dar solución a ciertos problemas, no obstante esto lo hemos hecho de manera lineal y en un solo método: **El método main.**

Los métodos son fracciones de código que contienen las clases con instrucciones que se pueden utilizar más de una vez en todo el programa que estemos haciendo. Por ejemplo, si necesitamos realizar algún cálculo y este se repite varias veces dentro del programa. Para esto, debemos crear, por ejemplo, el método **cálculo**, y este podrá ser llamado varias veces en el código. Además, en programas complejos con muchas líneas de código, es necesario utilizar métodos ya que nos ayuda a organizar y entender mejor el programa completo.

## Métodos

Un método es una agrupación de instrucciones que realizan una determinada tarea, el cual permite reutilizar este código en distintas partes del programa.

Algunos métodos que ya hemos utilizado son:

- `String.CompareTo();`
- `System.out.printf();`
- `Integer.parseInt();`

Nosotros también podemos crear nuestros propios métodos, a esto llamaremos definir. Al utilizar un método ya creado por nosotros o por otras personas, le denominaremos llamar.

## Creación de métodos

Hasta el momento hemos utilizado métodos que no han sido creados por nosotros, a los cuales les hemos entregado valores y hemos obtenido otros de vuelta. Ahora aprenderemos a codificar dichos métodos desde cero para luego utilizarlos. Esto nos permitirá simplificar y reutilizar código.

## Definiendo un método

En Java un método se define usando la siguiente estructura.

```
[especificadores] tipoDevuelto nombreMetodo([lista parámetros]) [throws  
listaExcepciones]  
{  
    // instrucciones  
    [return valor;]  
}
```

- **especificadores (opcional):** Determina el acceso a este método.
  - **private:** Solo es accesible desde la misma clase donde reside el método.
  - **public:** Accesible por la misma clase o cualquier clase.
  - **protected:** Se puede acceder solo desde clases heredadas.
  - **Por defecto:** Si no se indica, el método es público dentro de su mismo paquete.
- **tipoDevuelto:** Determina el tipo del valor que va a retornar el método. Para retornar un valor se debe utilizar la instrucción return. En caso de que no se retorne ningún valor, el método debe indicar que este tipo será void. En este caso, si se devuelve algo podría ser cualquier tipo de dato, tales como String, int, float o bien una clase personalizada, por ejemplo, Cuadrado o Notas.
- **nombreMetodo:** Es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

- **lista parámetros (opcional):** Después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas.
  - Los parámetros son los datos de entrada que recibe el método para operar con ellos.
  - Un método puede recibir cero o más argumentos.
  - Se debe especificar para cada uno de ellos su tipo.
  - Los paréntesis son obligatorios aunque estén vacíos.
- **throws listaExcepciones (opcional):** Indica las excepciones que puede generar y manipular el método.
- **return:** Se utiliza para devolver un valor. En caso de ser void, no se utiliza.

Todas las instrucciones definidas dentro del método serán ejecutadas cuando hagamos un llamado a este.

### Ubicación del método dentro del código

Ya teniendo definido cuál es la sintaxis de cómo se escribe un método, nos entra la duda de ¿dónde creamos el o los métodos?

Veamos cómo es la estructura de la clase donde estamos trabajando.

```
package nombre.package;  
import java.util.Scanner;  
public class NombreClase{  
    public static void main(String[] args) {  
        //INSTRUCCIONES  
    }  
}
```



Podemos ver que al inicio tenemos los paquetes de código que estamos utilizando. Luego viene la definición de nuestra clase (en nuestro caso el programa), y a continuación, tenemos un método llamado `main()`, donde todo el código que hemos escrito hasta ahora está dentro de éste.

```
package nombre.package;
import java.util.Scanner;
public class NombreClase{
    public static void main(String[] args) {
        //INSTRUCCIONES
    }
    static void nombreMetodo(){ //Listado de métodos
        //Lista de instrucciones
    }
}
```

Luego del método `main`, podemos ir agregando todos los métodos que queramos para nuestro programa.

Cabe destacar que el modificador `static` se debe utilizar en estos métodos ya que la llamada al método será desde otro método `static` (`main`).

Creando nuestro primer método

```
import package metodos;
public class Metodos{
    public static void main(String[] args) {
        imprimirMenu(); //llamando al método imprimirMenu();
    }

    static void imprimirMenu() {
        System.out.printf("MENU:\n");
        System.out.printf("1) Opción 1\n");
        System.out.printf("2) Opción 2\n");
        System.out.printf("3) Opción 3\n");
        System.out.printf("4) Salir\n");
    }
}
```

En el código anterior, se puede apreciar que se ha creado el método imprimirMenú, el cual al ser llamado imprimirá el menú correspondiente.

Parámetros

Crear un método que recibe un parámetro es sencillo. En la definición del método utilizaremos paréntesis para especificar los parámetros que debe recibir.

```
static void incrementar(int numero) {
    int total = numero +1;
    System.out.printf("%d\n",total);
}
```

Para utilizar el método, basta con llamarlo y entregarle un valor del tipo de dato que requiere.

```
incrementar(4); //5
```

Decimos que los parámetros se exigen porque si no los especificamos, obtendremos un error.

```
incrementar(); //5
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
The method incrementar(int) in the type Metodos is not applicable for the  
arguments ()
```

Además, cabe destacar que el método puede recibir variables como parámetros:

```
int a = 4;  
incrementar(a); //5
```

Solamente debemos tener en consideración que la definición de los parámetros al momento de crear el método, define el tipo de datos de cada uno de sus parámetros, por lo que la variable que entre como parámetro debe ser del mismo tipo.

### Sobrecarga de un método

La sobrecarga de un método corresponde a escribir el mismo nombre del método con una definición distinta, por ejemplo, podemos tener el `metodo1()` y el `metodo1(10)`. En este ejemplo podemos llamar al mismo método, con la salvedad que la primera vez se llamó sin parámetros y la segunda se le pasó el parámetro 10.

```
public static String holaMundo(){  
    return "Hola Mundo!!";  
}  
  
public static String holaMundo(String nombre){  
    return "Hola Mundo " + nombre;  
}
```

En el ejemplo anterior, tenemos dos métodos con diferente definición de parámetros. Pues, si llamamos a los dos métodos tendremos un resultado distinto, no obstante ambos métodos se llaman de la misma manera. A esto llamamos sobrecarga de un método.

```
public class SobrecargaMetodo{

    public static void main(String[] args) {
        System.out.println(holaMundo());
        System.out.println(holaMundo("Pepito"));
    }

    public static String holaMundo(){
        return "Hola Mundo!!";
    }

    public static String holaMundo(String nombre){
        return "Hola Mundo " + nombre;
    }
}
```

## Retorno

Los métodos pueden recibir parámetros y pueden devolver un valor. A este valor se le conoce como retorno. En Java, al definir un método, antes del nombre del método se especifica el tipo de dato de retorno.

```
tipoRetorno nombreMetodo(){
    //Declaración de variables locales
    //Cuerpo del método
    return valor;
}
```

Donde valor es el valor retornado por el método y tipo es el tipo del valor de retorno.

Los métodos pueden o no devolver un valor, en el caso de que se devuelva un valor la sentencia return debe ser obligatoria.

Por ejemplo:

```
public class SobrecargaMetodo{

    public static void main(String[] args) {
        System.out.println(holaMundo());
        System.out.println(holaMundo("Pepito"));
    }

    public static String holaMundo(){
        return "Hola Mundo!!";
    }

    public static String holaMundo(String nombre){
        return "Hola Mundo " + nombre;
    }
}
```

En este caso, el método `holaMundo` por su definición, dice que devolverá un dato tipo `String`.

Pero también podrían haberse creado dichos métodos sin retorno de valor:

```
public class SobrecargaMetodo{

    public static void main(String[] args) {
        holaMundo();
        holaMundo("Pepito");
    }

    public static void holaMundo(){
        System.out.println("Hola Mundo!!");
    }

    public static void holaMundo(String nombre){
        System.out.println("Hola Mundo " + nombre);
    }
}
```

En este caso, al definir el retorno como void, estamos diciendo que el método no devolverá algún resultado, de lo contrario hará el proceso dentro de el mismo. En este caso, la sentencia return no es obligatoria.

En Java, si se define un método con retorno es necesario que siempre haya un valor de retorno.

```
public static String mayorOMenor(int edad) {  
    if (edad < 18) {  
        return "Menor de edad";  
    } else {  
        return "Mayor de edad";  
    }  
}
```

El return no necesariamente debe estar en la última línea, puede utilizarse antes de llegar al final del código, haciendo que lo siguiente no se ejecute, pero debe haber al menos un return que se ejecute sí o sí dentro del método.

Si escribiéramos solamente esto:

```
public static String mayorOMenor(int edad) {  
    if (edad < 18) {  
        return "Menor de edad";  
    }  
}
```

Nos dará el siguiente error, donde nos dice que debe haber obligatoriamente un return de tipo String.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
This method must return a result of type String
```

## Alcance de variables

### Competencias

- Aplicar el concepto de alcance de variables para así usarlas correctamente.
- Comprender el ámbito en que las variables actúan.

### Introducción

Los tipos de variable y el alcance de estas son conceptos muy importantes, nos permiten entender desde dónde podemos acceder a una variable. A veces podemos enfrentarnos a problemas donde queremos acceder a una variable, pero esta no lo permite. Es por eso que a continuación aprenderemos cuáles son los distintos tipos de variables dentro del mundo de Java y cuál es su alcance.

## Categorías de variables

Existen varias categorías de variables en Java que definen en parte el alcance de las mismas. Estas categorías son las siguientes:

- Variables locales
- Variables de instancia
- Variables de clase /static

## El alcance

El alcance o scope en inglés, define desde donde podemos acceder a una variable.

### Variables locales

Una variable definida dentro de un método puede ser accedida solamente dentro del mismo método. No puede ser accedida fuera de este.

```
public static void main(String[] args) {  
    System.out.println(aprobado(5,6));  
}  
static boolean aprobado(float nota1, float nota2) {  
    float promedio = (nota1+nota2)/2; //Promedio variable local del método  
    return promedio <=5 ? true:false;  
}
```

Si en el main tratamos de acceder a promedio.

```
System.out.println(promedio);
```



Obtendremos el siguiente error, donde nos dice que la variable promedio no se puede resolver debido a que no está en el mismo ámbito o scope.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
promedio cannot be resolved to a variable
```

### Los parámetros también cuentan como variables locales

Si en el main tratamos de acceder a `nota1`, obtendremos el mismo error que para promedio.

```
System.out.println(nota1);
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
nota1 cannot be resolved to a variable
```

### Variables de instancia

De momento no hemos utilizado ninguna variable de instancia. Hemos tenido variables locales en el método main y variables locales en los métodos que hemos creado.

Para crear una variable de instancia, debemos primero crear una nueva clase. Las variables de instancia hacen referencia a variables dentro de una clase, fuera de los métodos, las cuales pueden ser accedidas desde cualquiera de los métodos de dicha clase.

Veamos esto con un ejemplo.

Creamos una nueva clase a la cual denominaremos Notas, dentro del mismo paquete:

```
//Nueva clase
public class Notas {
    int nota1;
    int nota2;
    /*A los métodos que se llaman igual que la clase, los llamamos
constructores de la clase*/
    public Notas(int n1, int n2) {
        this.nota1 = n1;
        this.nota2 = n2;
    }
    //Otro método de la clase
    public float promedio() {
        return (float)(nota1+nota2)/2.0f;
    }
}
```

Por otro lado, tenemos el main de la siguiente forma.

```
package métodos;
import java.util.Scanner;
public class Metodos {
    public static void main(String[] args) {
        //Se crea la instancia de la clase Notas
        Notas n = new Notas(4,5);
        System.out.printf("%f\n",n.promedio());
    }
}
```

Así como hemos creado Strings (que es una clase), podemos utilizar las clases que nosotros necesitemos personalizar. Podemos ver que creamos una variable de tipo `Notas`, a la cual denominamos `n`.

Para crearla, utilizamos el constructor `Notas`, la cual recibe 2 parámetros y se inicializa el objeto con los valores `notas1 = 4`, y `notas2 = 5`.

En el ejemplo anterior, las variables `notas1` y `notas2` podemos accederlas o usarlas desde el método `promedio`, sin pasarlas como parámetros.

### Variables de clase/static

Para crear una variable de clase, basta con agregar al inicio de la variable la palabra `static`. Pero, ¿qué pasa al hacer dicha acción?

Al definir una variable de tipo `static`, hará que al modificar por ejemplo la variable en un objeto, se modifique para todos los objetos que estén creados.

Veamos el ejemplo anterior, definiéndolas como `static`

```
package metodos;
public class Notas {
    static int nota1;
    static int nota2;
    public Notas(int n1, int n2) {
        this.nota1 = n1;
        this.nota2 = n2;
    }
    public float promedio() {
        return (float)(nota1+nota2)/2.0f;
    }
}
```

Y al implementar la clase `Notas` en el método `Main`:

```
package metodos;
import java.util.Scanner;
public class Metodos {
    public static void main(String[] args) {
        Notas n = new Notas(4,5);
        Notas n2 = new Notas(4,6);
        System.out.printf("%f %f\n",n.promedio(),n2.promedio());
    }
}
```

El resultado para ambos promedios será,

```
5,000000 5,000000
```

Ya que al instanciar `n2`, asigna las variables `notas1` y `notas2` para ambos en 4 y 6.

## Ejercicio guiado: Calculadora Suma y Resta

### Contexto

Se necesita crear una calculadora que pida números al usuario de forma infinita hasta que el usuario ingrese el operador igual.

### Requerimientos

1. Los números a sumar o restar deben ser de tipo `float`.
2. Se debe indicar al usuario que debe ingresar, si un número, o un operador.
3. Como operador solo se debe admitir el ingreso del signo más (+) o menos (-).
4. Cuando el usuario ingrese el signo igual (=), se mostrará el resultado.

Solución:

Requerimiento 1:

- a. Se crea el proyecto, con una clase `Calculadora.java` dentro del package `cl.desafiolatam`

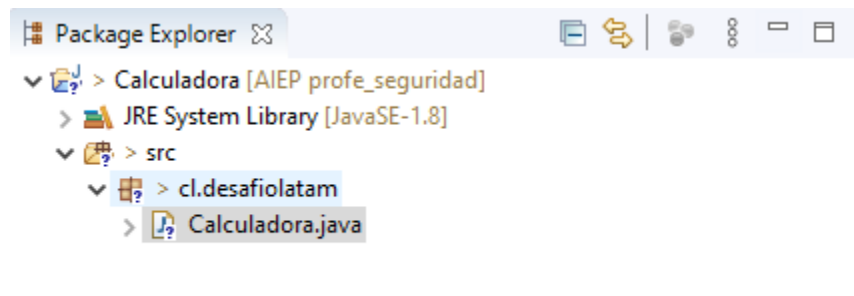


Imagen 1. Crear clase Calculadora.  
Fuente: Desafío Latam.

```
package cl.desafiolatam;  
  
public class Calculadora {  
  
    public static void main(String[] args) {  
  
    }  
}
```

b. Se declaran las variables

```
//Se declaran las variables a utilizar
//El número que ingrese el usuario debe ser float
float numero = 0f;
float resultado = 0;
//Variable tipo String, ya que puede ser un operador o un número
String ingreso = null;
//Variable que guardará el código ascii de cada ingreso del usuario
int ascii = 0;
/*
 * Se crea variable contador, para identificar si es el primer ingreso
 * de usuario, el cual debe ser un número, y luego un operador, número
 * así sucesivamente. Si contador es cero es inicio y debe ser un
 * número. Si es impar, debe ser un operador +, -, =
 */
int contador = 0;
Scanner sc = new Scanner(System.in); //Se crea objeto Scanner
```

Requerimiento 2:

- a. Utilizando el contador, se evalúa si el módulo es cero (par) o impar (distinto de cero) para saber si debe ingresar el usuario un número o bien un operador.

```
//Si contador es par el usuario debe ingresar un número, de lo contrario
debe ingresar un operador
if((contador % 2) == 0) {
    System.out.println("Ingrese un número: ");
}else {
    System.out.println("Ingrese un operador (+, -, =): ");
    contador++;
}
```

Requerimiento 3:

- a. Se agregan las condiciones para evaluar qué es lo que está ingresando el usuario, si un signo =, + o -, para así saber qué cálculo se debe realizar.

```
//Se lee por pantalla el ingreso del usuario, operador o número
ingreso = sc.next();
/*
 * Códigos ascii para operadores
 * + --> 43
 * - --> 45
 * = --> 61
 */
ascii = (int) ingreso.charAt(0);
/*
 * Si el programa inicio, es decir el contador esta en cero y el usuario
 * no ingreso un operador (+, -, =)
 */
if(contador == 0 && ascii != 43 && ascii != 45 && ascii != 61) {
    //Se hace un cast de ingreso (String) a float
    /*
     * Todas las clases tienen un parse, Integer.parseInt,
     * Double.parseDouble etc. SE hace de esta manera, porque un
     * String no se puede transformar en un número, sin embargo si
     * el String no es un número, el programa dará un error.
     */
    numero = Float.parseFloat(ingreso);
    //Se asigna a resultado el primer numero ingresado
    resultado = numero;
    contador++;
}else { // De lo contrario, es el segundo ingreso de usuario
    //Suma
    if(ascii == 43) {
        System.out.println("Ingrese un número: ");
        ingreso = sc.next(); //Debiese ser un número o si no error
        numero = Float.parseFloat(ingreso);
        resultado = resultado + numero;
        contador++;
    }
}
```

```
//Resta
if(ascii == 45) {
    System.out.println("Ingrese un número: ");
    ingreso = sc.next(); //Debiese ser un número o si no error
    numero = Float.parseFloat(ingreso);
    resultado = resultado - numero;
    contador++;
}
}
```

Requerimiento 4:

El algoritmo anterior del requerimiento 2 y 3 debe quedar dentro de un loop **do-while**, para que se repita el proceso hasta que el usuario ingrese el operador =.

```
do{
.
.
.
//Repita mientras el ingreso no sea =
}while((int)ingreso.charAt(0) != 61);
System.out.printf("El Resultado es: %f", resultado);
```



## Solución completa

```
package cl.desafiolatam;

import java.util.Scanner;

public class Calculadora {

    public static void main(String[] args) {
        //Se declaran las variables a utilizar
        //El número que ingrese el usuario debe ser float
        float numero = 0f;
        float resultado = 0;
        //Variable tipo String, ya que puede ser un operador o un
        número
        String ingreso = null;
        //Variable que guardará el código ascii de cada ingreso del
        usuario
        int ascii = 0;
        /*
        * Se crea variable contador, para identificar si es el primer
        ingreso
        * de usuario, el cual debe ser un número, y luego un operador,
        número
        * así sucesivamente. Si contador es cero es inicio y debe ser
        un
        * número. Si es impar, debe ser un operador +, -, =
        */
        int contador = 0;
        Scanner sc = new Scanner(System.in); //Se crea objeto Scanner
        do {
            //Si contador es par el usuario debe ingresar un número,
            de lo contrario debe ingresar un operador
            if((contador % 2) == 0) {
                System.out.println("Ingrese un número: ");
            }else {
                System.out.println("Ingrese un operador (+, -, =):");
            }
            contador++;
        } while (true);
    }
}
```

```
    }

    //Se lee por pantalla el ingreso del usuario, operador o
número

    ingreso = sc.next();
    /*
    * Códigos ascii para operadores
    * + --> 43
    * - --> 45
    * = --> 61
    */
    ascii = (int) ingreso.charAt(0);
    /*
    *Si el programa inicio, es decir el contador esta en
cero y el usuario
    * no ingreso un operador (+, -, =)
    */
    if(contador == 0 && ascii != 43 && ascii != 45 && ascii
!= 61) {

        //Se hace un cast de ingreso (String) a float
        /*
        * Todas las clases tienen un parse,
Integer.parseInt,
        * Double.parseDouble etc. SE hace de esta manera,
porque un
        * String no se puede transformar en un número, sin
embargo si
        * el String no es un número, el programa dará un
error.

        */
        numero = Float.parseFloat(ingreso);
        //Se asigna a resultado el primer numero ingresado
        resultado = numero;
        contador++;
    }else { // De lo contrario, es el segundo ingreso de
usuario

        //Suma
        if(ascii == 43) {
            System.out.println("Ingresa un número: ");
            ingreso = sc.next(); //Debiese ser un número
```

```
o si no error

        numero = Float.parseFloat(ingreso);
        resultado = resultado + numero;
        contador++;
    }
    //Resta
    if(ascii == 45) {
        System.out.println("Ingrese un número: ");
        ingreso = sc.next(); //Debiese ser un número
o si no error

        numero = Float.parseFloat(ingreso);
        resultado = resultado - numero;
        contador++;
    }
}
//Repite mientras el ingreso no sea =
}while((int)ingreso.charAt(0) != 61);
System.out.printf("El Resultado es: %f", resultado);
}

}
```

## Ejercicio propuesto (2)

### Contexto

Al ejercicio guiado anterior, agregar las operaciones de multiplicación y división, además modularizar el código en métodos. Estos métodos deben ser los siguientes:

- Sumar
- Restar
- Multiplicar
- Dividir

## Soluciones ejercicios propuestos

### Solución Ejercicio propuesto (1)

Siguiendo el ejercicio guiado, debemos modificar el algoritmo para imprimir el siguiente patrón, el límite será indicado por un valor ingresado por el usuario:

```
1
12
123
1234
12345
123456
```

**Paso 1:** Solicitamos un número al usuario el cual definirá el `limite` de la escalera.

**Paso 2:** Crear un ciclo que dibujará las filas.

**Paso 3:** Crear un ciclo anidado que dibujará las columnas.

Finalmente quedaría de la siguiente manera:

```
public class EscaleraNumeros {  
    public static void main(String args[]) {  
        Scanner sc = new Scanner(System.in);  
        System.out.printf("Ingresa un número");  
        int limite = sc.nextInt();  
        for(int i=1; i<=limite;i++)  
        {  
            for(int j=1;j<=i;j++)  
            {  
                System.out.print(j);  
            }  
            System.out.println("");  
        }  
    }  
}
```

## Solución Ejercicio Propuesto (2)

### Descripción Paso 1

Se agrega a la condición inicial, que además el `ascii` sea distinto de 42 (\*) multiplicación y 47 (/) división.

```
if(contador == 0 && ascii != 43 && ascii != 45 && ascii != 61 && ascii !=  
42 && ascii != 47)
```

### Descripción Paso 2

Se deben agregar las condiciones para que se ejecuten las operaciones de multiplicación y división.

```
//Multiplicación  
if(ascii == 42) {  
    System.out.println("Ingrese un número: ");  
    ingreso = sc.next(); //Debiese ser un número o si no error  
    numero = Float.parseFloat(ingreso);  
    resultado = resultado * numero;  
    contador++;  
}  
//División  
if(ascii == 47) {  
    System.out.println("Ingrese un número: ");  
    ingreso = sc.next(); //Debiese ser un número o si no error  
    numero = Float.parseFloat(ingreso);  
    resultado = resultado / numero;  
    contador++;  
}
```

Solución completa sin los métodos

```
package cl.desafiolatam;

import java.util.Scanner;

public class Calculadora {

    public static void main(String[] args) {
        //Se declaran las variables a utilizar
        //El número que ingrese el usuario debe ser float
        float numero = 0f;
        float resultado = 0;
        //Variable tipo String, ya que puede ser un operador o un
        número
        String ingreso = null;
        //Variable que guardará el código ascii de cada ingreso del
        usuario
        int ascii = 0;
        /*
        * Se crea variable contador, para identificar si es el primer
        ingreso
        * de usuario,el cual debe ser un número, y luego un operador,
        número
        * así sucesivamente. Si contador es cero es inicio y debe ser
        un
        * número. Si es impar, debe ser un operador +, -, =
        */
        int contador = 0;
        Scanner sc = new Scanner(System.in); //Se crea objeto Scanner
        do {
            //Si contador es par el usuario debe ingresar un número,
            de lo contrario debe ingresar un operador
            if((contador % 2) == 0) {
                System.out.println("Ingrese un número: ");
            }else {
                System.out.println("Ingrese un operador (+, -, *,
                /=): ");
                contador++;
            }
        }
```

```
número //Se lee por pantalla el ingreso del usuario, operador o

    ingreso = sc.next();
    /*
    * Códigos ascii para operadores
    * + --> 43
    * - --> 45
    * = --> 61
    * * --> 42
    * '/' --> 47
    */
    ascii = (int) ingreso.charAt(0);
    /*
    * Si el programa inicio, es decir el contador esta en
    cero y el usuario
    * no ingreso un operador (+, -, =)
    */
    if(contador == 0 && ascii != 43 && ascii != 45 && ascii
    != 61 && ascii != 42 && ascii != 47) {
        //Se hace un cast de ingreso (String) a float
        /*
        * Todas las clases tienen un parse,
        Integer.parseInt,
        * Double.parseDouble etc. SE hace de esta manera,
        porque un
        * String no se puede transformar en un número, sin
        embargo si
        * el String no es un número, el programa dará un
        error.
        */
        numero = Float.parseFloat(ingreso);
        //Se asigna a resultado el primer numero ingresado
        resultado = numero;
        contador++;
    }else { // De lo contrario, es el segundo ingreso de
    usuario

        //Suma
        if(ascii == 43) {
            System.out.println("Ingrese un número: ");
```



```

                                ingreso = sc.next(); //Debiese ser un número
o si no error

                                numero = Float.parseFloat(ingreso);
                                resultado = resultado + numero;
                                contador++;
                                }
                                //Resta
                                if(ascii == 45) {
                                    System.out.println("Ingrese un número: ");
                                    ingreso = sc.next(); //Debiese ser un número
o si no error

                                    numero = Float.parseFloat(ingreso);
                                    resultado = resultado - numero;
                                    contador++;
                                }

                                //Multiplicación
                                if(ascii == 42) {
                                    System.out.println("Ingrese un número: ");
                                    ingreso = sc.next(); //Debiese ser un número
o si no error

                                    numero = Float.parseFloat(ingreso);
                                    resultado = resultado * numero;
                                    contador++;
                                }
                                //División
                                if(ascii == 47) {
                                    System.out.println("Ingrese un número: ");
                                    ingreso = sc.next(); //Debiese ser un número
o si no error

                                    numero = Float.parseFloat(ingreso);
                                    resultado = resultado / numero;
                                    contador++;
                                }
                                }
                                //Repite mientras el ingreso no sea =
                                }while((int)ingreso.charAt(0) != 61);
                                System.out.printf("El Resultado es: %f", resultado);
                                }
                                }
```

### Descripción Paso 3

Del código completo, se extraerán las fracciones de código para las operaciones y se crearán los métodos:

Nótese que cada uno de los métodos recibe el resultado anterior y cada uno de los métodos se encarga de preguntar al usuario por el número a ingresar. Además, las variables `sc` (Scanner), `numero` (float) e `ingreso`(String), se declaran nuevamente para que actúen en el ámbito local de cada método.

```
static float sumar(float resultadoAnterior) {
    String ingreso = null;
    float numero = 0f;
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese un número: ");
    ingreso = sc.next(); //Debiese ser un número o si no error
    numero = Float.parseFloat(ingreso);
    return resultadoAnterior + numero;
}

static float restar(float resultadoAnterior) {
    String ingreso = null;
    float numero = 0f;
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese un número: ");
    ingreso = sc.next(); //Debiese ser un número o si no error
    numero = Float.parseFloat(ingreso);
    return resultadoAnterior - numero;
}

static float multiplicar(float resultadoAnterior) {
    String ingreso = null;
    float numero = 0f;
    Scanner sc = new Scanner(System.in);
    System.out.println("Ingrese un número: ");
    ingreso = sc.next(); //Debiese ser un número o si no error
    numero = Float.parseFloat(ingreso);
    return resultadoAnterior * numero;
}
```

```
static float dividir(float resultadoAnterior) {  
    String ingreso = null;  
    float numero = 0f;  
    Scanner sc = new Scanner(System.in);  
    System.out.println("Ingrese un número: ");  
    ingreso = sc.next(); //Debiese ser un número o si no error  
    numero = Float.parseFloat(ingreso);  
    return resultadoAnterior / numero;  
}
```

Solución completa con los métodos

```
package cl.desafiolatam;  
  
import java.util.Scanner;  
  
public class Calculadora {  
  
    public static void main(String[] args) {  
        //Se declaran las variables a utilizar  
        //El número que ingrese el usuario debe ser float  
        float numero = 0f;  
        float resultado = 0;  
        //Variable tipo String, ya que puede ser un operador o un  
        número  
        String ingreso = null;  
        //Variable que guardará el código ascii de cada ingreso del  
        usuario  
        int ascii = 0;  
        /*  
        * Se crea variable contador, para identificar si es el primer  
        ingreso  
        * de usuario,el cual debe ser un número, y luego un operador,  
        número  
        * así sucesivamente. Si contador es cero es inicio y debe ser  
        un
```

```
        * número. Si es impar, debe ser un operador +, -, =
    */
    int contador = 0;
    Scanner sc = new Scanner(System.in); //Se crea objeto Scanner
    do {
        //Si contador es par el usuario debe ingresar un número,
        de lo contrario debe ingresar un operador
        if((contador % 2) == 0) {
            System.out.println("Ingrese un número: ");
        }else {
            System.out.println("Ingrese un operador (+, -, *,
/=): ");
            contador++;
        }

        //Se lee por pantalla el ingreso del usuario, operador o
        número

        ingreso = sc.next();
        /*
        * Códigos ascii para operadores
        * + --> 43
        * - --> 45
        * = --> 61
        * * --> 42
        * '/' --> 47
        */
        ascii = (int) ingreso.charAt(0);
        /*
        *Si el programa inicio, es decir el contador esta en
        cero y el usuario
        * no ingreso un operador (+, -, =)
        */
        if(contador == 0 && ascii != 43 && ascii != 45 && ascii
!= 61 && ascii != 42 && ascii != 47) {
            //Se hace un cast de ingreso (String) a float
            /*
            * Todas las clases tienen un parse,
            Integer.parseInt,
            * Double.parseDouble etc. SE hace de esta manera,
            porque un
```

```
embargo si
error.

        * String no se puede transformar en un número, sin
        * el String no es un número, el programa dará un
        */
        numero = Float.parseFloat(ingreso);
        //Se asigna a resultado el primer numero ingresado
        resultado = numero;
        contador++;
    }else { // De lo contrario, es el segundo ingreso de
usuario
        //Suma
        if(ascii == 43) {
            resultado = sumar(resultado);
            contador++;
        }
        //Resta
        if(ascii == 45) {
            resultado = restar(resultado);
            contador++;
        }

        //Multiplicación
        if(ascii == 42) {
            resultado = multiplicar(resultado);
            contador++;
        }
        //División
        if(ascii == 47) {
            resultado = dividir(resultado);
            contador++;
        }
    }
    //Repite mientras el ingreso no sea =
}while((int)ingreso.charAt(0) != 61);
System.out.printf("El Resultado es: %f", resultado);
}

static float sumar(float resultadoAnterior) {
    String ingreso = null;
```

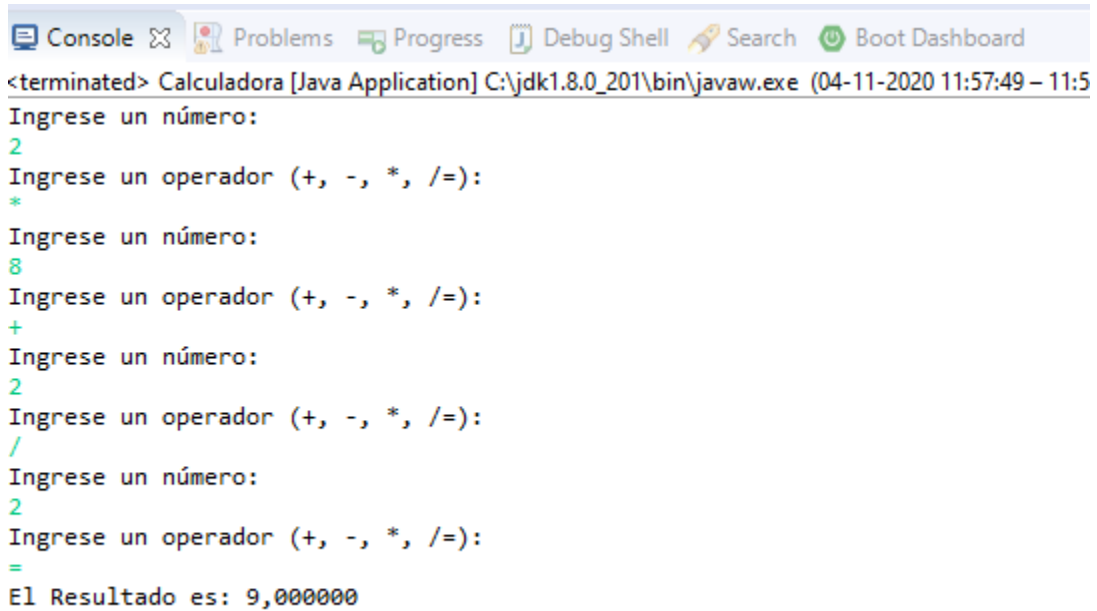
```
        float numero = 0f;
        Scanner sc = new Scanner(System.in);
        System.out.println("Ingrese un número: ");
        ingreso = sc.next(); //Debiese ser un número o si no error
        numero = Float.parseFloat(ingreso);
        return resultadoAnterior + numero;
    }

    static float restar(float resultadoAnterior) {
        String ingreso = null;
        float numero = 0f;
        Scanner sc = new Scanner(System.in);
        System.out.println("Ingrese un número: ");
        ingreso = sc.next(); //Debiese ser un número o si no error
        numero = Float.parseFloat(ingreso);
        return resultadoAnterior - numero;
    }

    static float multiplicar(float resultadoAnterior) {
        String ingreso = null;
        float numero = 0f;
        Scanner sc = new Scanner(System.in);
        System.out.println("Ingrese un número: ");
        ingreso = sc.next(); //Debiese ser un número o si no error
        numero = Float.parseFloat(ingreso);
        return resultadoAnterior * numero;
    }

    static float dividir(float resultadoAnterior) {
        String ingreso = null;
        float numero = 0f;
        Scanner sc = new Scanner(System.in);
        System.out.println("Ingrese un número: ");
        ingreso = sc.next(); //Debiese ser un número o si no error
        numero = Float.parseFloat(ingreso);
        return resultadoAnterior / numero;
    }
}
```

## Resultado



```
Console Problems Progress Debug Shell Search Boot Dashboard
<terminated> Calculadora [Java Application] C:\jdk1.8.0_201\bin\javaw.exe (04-11-2020 11:57:49 - 11:5
Ingrese un número:
2
Ingrese un operador (+, -, *, /):
*
Ingrese un número:
8
Ingrese un operador (+, -, *, /):
+
Ingrese un número:
2
Ingrese un operador (+, -, *, /):
/
Ingrese un número:
2
Ingrese un operador (+, -, *, /):
=
El Resultado es: 9,000000
```

Imagen 2. Solución ejercicio propuesto (2).  
Fuente: Desafío Latam.