



Programación 1

Análisis de código y Debugging.

Análisis de código

Al resolver un problema es fundamental saber si estamos resolviendo lo pedido.

Para ello hay varias técnicas que se pueden usar de manera tal de seguir la ejecución del programa y ver que realmente cumpla con lo esperado.



```
//Load values from database store with channel select
NotificationClient NotificationClient = null; // = NotificationClient On
if (NotificationClient == null)
{
    NotificationClient = new bl.desktop NotificationClient() { Deny = fa
    //NotificationClient.Insert();
}
else
{
    NotificationClient.LastRequest = DateTime.Now,
    NotificationClient.RequestCount = NotificationClient.RequestCount +
    //NotificationClient.Update();
}
if (NotificationClient.Deny == false)
{
    NotificationRequest NotificationRequest = new bl.desktop Notification
    NotificationRequest.ClientId = NotificationClient.ClientId,
    NotificationRequest.CurrentRequestUserHostAddress =
    NotificationRequest.RequestCount = NotificationClient.RequestCount,
    NotificationRequest.Timestamp = DateTime.Now;
```

Donde quiero llegar 1/2



Una forma de saber si estoy haciendo lo correcto es ver dónde quiero llegar y cómo debería quedar el estado final de la ejecución del algoritmo y contenido de las variables.

Ejemplo: si quiero obtener el mayor, el menor y el promedio de una serie de números dada: 45, 58, 2, 9, 21 tal que el menor sería el 2, el mayor el 58 y el promedio 27.

Un **pseudocódigo** del algoritmo posible para resolver este problema podría ser:

recorrer la lista de nros, por cada nro verificar si es menor al menor, mayor al mayor y

sumar dicho valor en una variable acumuladora

incrementar la cantidad de nros que procese

cuando termina de recorrer la lista dividir la suma acumulada por la cantidad de nros

Donde quiero llegar 2/2



A simple vista parece fácil la resolución, pero siempre es bueno preguntarse algunas cosas como:

¿Qué pasaría si todos los valores fueran iguales? ¿Cuál es el menor? ¿Cuál es el mayor?

¿Qué pasa si tengo solo un elemento?

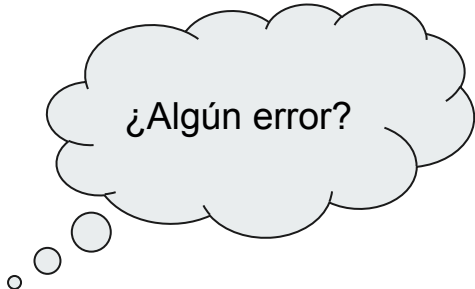
¿Conviene poner un valor extremo máximo al menor y un valor extremo mínimo al mayor? ¿Qué valor pongo?

¿Puedo suponer que el primer valor es el mayor y el menor que tengo hasta el momento?

Sentencia condicional if

```
public class If{  
    public static void main(String[] args) {  
        int entero = -10;  
        if (entero<0)  
            entero=-entero;  
        if (entero>=0)  
            System.out.println("Es positivo");  
    }  
}
```

¿Qué debería esperar que haga este código?
¿Puede que le falte algo?
Mucho depende de la lógica del negocio que se quiera implementar



¿Algún error?



¿Compila?

Sentencia condicional switch-case

```
public class Switch_Case{  
    public static void main(String[] args) {  
        char opcion = 'a';  
        int entero = 10;  
        switch (opcion) {  
            case 'a': System.out.println("Es a");  
                    entero=23;  
            case 'e': System.out.println("Es e");  
            case 'i': System.out.println("Es i");  
                    if (entero>15)  
                        entero=-34;  
            case 'o': System.out.println("Es o");  
            case 'u': System.out.println("Es u");  
                    if (entero<0)  
                        System.out.println("Es negativo");  
            default: System.out.println("No es vocal");  
        }  
    }  
}
```

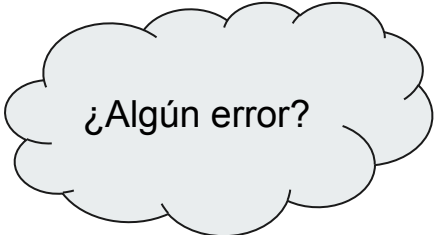
¿Algún error?

¿Compila?

Sentencia repetitiva for

Problema: Obtener el promedio de 6 valores ingresados por el usuario

```
public class For{  
    public static void main(String[] args) {  
        final int MAX = 10, MIN=4;  
        for (int i=MAX; i>MIN;i-)  
            int suma=suma+obtenerValor();  
        System.out.println("El promedio es: " + (suma/(MAX-MIN)));  
    }  
    public static int obtenerValor(){  
        BufferedReader entrada=new BufferedReader(new InputStreamReader(System.in));  
        int valor=0;  
        try{    System.out.println("Ingrese un numero entero: ");  
            valor = Integer.valueOf(entrada.readLine());  
        }catch (Exception exc){  
            System.out.println(exc);  
        }return valor;  
    }  
}
```

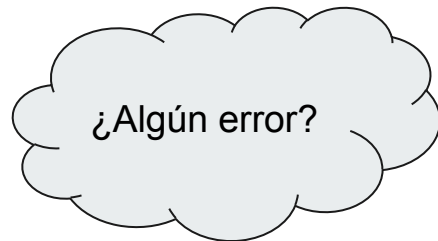


¿Algún error?

Sentencia repetitiva while

```
public class While{  
    public static void main(String[] args) {  
        BufferedReader entrada=new BufferedReader(new InputStreamReader(System.in));  
        int suma=0, cant=0, valor=0;  
        obtenerValor(valor, entrada);  
        while (valor!=0)  
            valor=obtenerValor(valor, entrada);  
        suma+=valor;  
        cant++;  
        System.out.println("El promedio es: " + (suma/cant));  
    }  
    public static int obtenerValor(int valor, BufferedReader entrada){  
        try{ System.out.println("Ingrese un numero entero: ");  
            valor = Integer.valueOf(entrada.readLine());  
        }catch (Exception exc){  
            System.out.println(exc);  
        }return valor;  
    }  
}
```

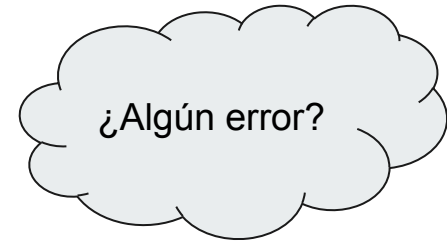
Problema: Obtener el promedio de los valores ingresados por el usuario mientras no ingrese un 0



Llamado a función

```
public class funcion{  
    public static void main(String[] args) {  
        BufferedReader entrada=new BufferedReader(new InputStreamReader(System.in));  
        int suma=0, cant=0, valor=0;  
        obtenerValor();  
        while (valor!=0){  
            valor=obtenerValor(valor, entrada);  
            suma+=valor;  
            cant++;  
        }  
        System.out.println("El promedio es: " + (suma/cant));  
    }  
    public static int obtenerValor(int valor, BufferedReader entrada){  
        try{ System.out.println("Ingrese un numero entero: ");  
            valor = Integer.valueOf(entrada.readLine());  
        }catch (Exception exc){  
            System.out.println(exc);  
        }return valor;  
    }  
}
```

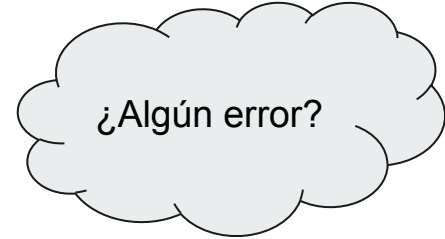
Problema: Obtener el promedio de los valores ingresados por el usuario mientras no ingrese un 0



Llamado a procedimiento

Problema: Obtener el promedio de los valores ingresados por el usuario mientras no ingrese un 0

```
public class procedimiento{
    public static void main(String[] args) {
        final String msj="El promedio es: ";
        int suma=0, cant=0, valor=0;
        obtenerValor();
        while (obtenerValor() !=0){
            suma+=valor;
            cant++;
        }
        mostrarMsj (suma, cant, msj);
    }
    public static void mostrarMsj(int cant, int suma, String msj){
        System.out.println(msj + (suma/cant));
    }
}
```





Alternativas de Debugging

En el papel:

Prueba de escritorio

En un entorno:

Impresiones por consola

Herramientas de debugging

Prueba de escritorio



Una técnica que no pierde vigencia es la conocida como **prueba de escritorio**. De mucha utilidad cuando quiero comprobar el funcionamiento de un algoritmo o programa usando papel y lápiz (**fundamental para cuando rindan el examen**).

Ejemplo:

Obtener el promedio de N valores cargados por teclado, verificar la correctitud de la solución mediante una prueba de escritorio.

Porción de código de interés

```
int suma=0, promedio=0, N=3;
for (int i=0; i<=N;i++){
    suma+=obtenerEntero();
}
promedio=suma/N;
```

Prueba de escritorio

Porción de código de interés

```
➡ 1  int suma=0, promedio=0, N=3;  
   2  for (int i=0; i<=N; i++){  
   3      suma+=obtenerEntero();  
   4  }  
   5  promedio=suma/N;
```


Resultado esperado:
Ingresan 8, 6 y 5. El promedio es 6.33

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3

Prueba de escritorio

Porción de código de interés



```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Al ser la primera ejecución del bucle FOR:


- Se ejecuta el bloque de inicialización

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3

Prueba de escritorio

Porción de código de interés



```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Al ser la primera ejecución del bucle FOR:

- Se ejecuta el bloque de inicialización

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3
i: 0

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Al ser la primera ejecución del bucle FOR:

- Se ejecuta el bloque de inicialización
- Se evalúa la condición de salida
 - $0 \leq 3$? true

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3
i: 0

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3
i: 0

Ingrese un número entero:

8



Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3
i: 0

Ingrese un número entero:

8

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

- Se ejecuta el bloque de actualización de la variable de control

Mapa de memoria RAM

suma: 0
promedio: 0
N: 3
i: 0

Ingrese un número entero:

8

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

- Se ejecuta el bloque de actualización de la variable de control
- Se evalúa la condición de salida
 - $1 \leq 3?$ true

Mapa de memoria RAM

suma: 0 8
promedio: 0
N: 3
i: 0 **1**

Ingrese un número entero:

8

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: 0 8
promedio: 0
N: 3
i: 0 1

Ingrese un número entero:

8

Ingrese un número entero:

6



Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
→ 3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: ~~0~~ 14
promedio: 0
N: 3
i: 0 1

Ingrese un número entero:

8

Ingrese un número entero:

6



Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++) {  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

- Se ejecuta el bloque de actualización de la variable de control

Mapa de memoria RAM

suma: 0 → 14
promedio: 0
N: 3
i: 0 → 1

Ingrese un número entero:

8

Ingrese un número entero:

6

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;
2 | for (int i=0; i<=N; i++){
3 |     suma+=obtenerEntero();
4 | }
5 | promedio=suma/N;
```

- Se ejecuta el bloque de actualización de la variable de control
- Se evalúa la condición de salida
 - $2 \leq 3?$ true

Mapa de memoria RAM

suma: ~~0~~ 14
promedio: 0
N: 3
i: ~~0~~ 2

Ingrese un número entero:

8

Ingrese un número entero:

6

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
→ 3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: ~~0~~ 14
promedio: 0
N: 3
i: ~~0~~ 2

Ingrese un número entero:

8

Ingrese un número entero:

6

Ingrese un número entero:

5



Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;
2 | for (int i=0; i<=N; i++){
3 |     suma+=obtenerEntero();
4 | }
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: ~~0+0+14~~ 19
promedio: 0
N: 3
i: ~~0+1~~ 2

Ingrese un número entero:

8

Ingrese un número entero:

6

Ingrese un número entero:

5



Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++) {  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

- Se ejecuta el bloque de actualización de la variable de control

Mapa de memoria RAM

suma: ~~0~~ 14 19
promedio: 0
N: 3
i: ~~0~~ 1 2

Ingrese un número entero:

8

Ingrese un número entero:

6

Ingrese un número entero:

5

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

- Se ejecuta el bloque de actualización de la variable de control
- Se evalúa la condición de salida
 - $3 \leq 3?$ **true**

Mapa de memoria RAM

suma: ~~0~~814 19
promedio: 0
N: 3
i: ~~0~~12 **3**

Ingrese un número entero:
8
Ingrese un número entero:
6
Ingrese un número entero:
5

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
→ 3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Se debe resolver en primer lugar la invocación a la función `obtenerEntero()`

Mapa de memoria RAM

suma: ~~0~~81419
promedio: 0
N: 3
i: ~~0~~123

Ingrese un número entero:
8
Ingrese un número entero:
6
Ingrese un número entero:
5
Ingrese un número entero:



Prueba de escritorio



Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<=N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

El error/bug está en la condición de salida del for

- Debe ser menor estricto ($i < N$)

Prueba de escritorio



Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

El error/bug está en la condición de salida del for

- Debe ser menor estricto ($i < N$)

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```

Volvemos a ejecutar la prueba...

- Se ejecuta el bloque de actualización de la variable de control
- Se evalúa la condición de salida
 - $3 < 3?$ **false**

Mapa de memoria RAM

suma: ~~0~~814 19

promedio: 0

N: 3

i: ~~0~~12 **3**

Ingrese un número entero:

8

Ingrese un número entero:

6

Ingrese un número entero:

5

Prueba de escritorio

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
→ 5 | promedio=suma/N;
```

Mapa de memoria RAM

suma: ~~0~~814 19
promedio: 0
N: 3
i: ~~0~~12 3

Volvemos a ejecutar la prueba...

- ¿Cuál es el resultado de la línea 5?

Ingrese un número entero:

8

Ingrese un número entero:

6

Ingrese un número entero:

5

Prueba de escritorio

Volvemos a ejecutar la prueba...

Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;  
2 | for (int i=0; i<N; i++){  
3 |     suma+=obtenerEntero();  
4 | }  
5 | promedio=suma/N;
```



Mapa de memoria RAM

suma: ~~0~~814 19

promedio: ~~0~~6

N: 3

i: ~~0~~12 3



Ingrese un número entero:

8

Ingrese un número entero:

6

Ingrese un número entero:

5

Prueba de escritorio



Porción de código de interés

```
1 | int suma=0, promedio=0, N=3;
2 | for (int i=0; i<N; i++){
3 |     suma+=obtenerEntero();
4 | }
5 | promedio=suma/N;
```

Errores/bugs detectados:

- promedio debe ser una variable de tipo *double*
- la división de enteros, da como resultado enteros

Prueba de escritorio



Porción de código de interés

```
1 | int suma=0, N=3;
2 | double promedio;
3 | for (int i=0; i<N; i++){
4 |     suma+=obtenerEntero();
5 | }
6 | promedio = (double)suma / N;
```

Errores/bugs detectados:

- promedio debe ser una variable de tipo *double*
- la división de enteros, da como resultado enteros



Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
public static void main(String[] args) {
    int minCant = Integer.MAX_VALUE;
    int minCaras = 1;
    int cant = 0; //auxiliar
    // Pueba con dados que tienen desde 3 hasta 10 caras
    for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
        cant = jugar(cantCaras);
        System.out.println("Con " + cantCaras + " se tardo " + cant + " rondas hacer generala.");
        if (minCant >= cant) {
            minCant = cant;
            minCaras = cantCaras;
        }
    }

    System.out.println("Se hizo generala en menos intentos con dados de " + minCaras + " caras.");
}
```

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.



Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a

```
public static void main(String[] args) {
    int minCant = Integer.MAX_VALUE;
    int minCaras = 1;
    int cant = 0; //auxiliar
    // Pueba con dados que tienen desde 3 hasta
    for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
        cant = jugar(cantCaras);
        System.out.println("Con " + cantCaras + " dados se hizo generala en " + minCant + " intentos");
        if (minCant >= cant) {
            minCant = cant;
            minCaras = cantCaras;
        }
    }

    System.out.println("Se hizo generala en menos intentos con dados de " + minCaras + " caras.");
}
```

```
public static int jugar(int cantCaras) {
    // Defino las variables
    int d1, d2, d3, d4; // Usamos 4 dados
    int cont = 0;
    boolean hiceGenerala = false;
    // Mientras no haga generala, sigo jugando y cuento
    while (!hiceGenerala) {
        // Tiro los dados
        d1 = tirarDado(cantCaras); d2 = tirarDado(cantCaras);
        d3 = tirarDado(cantCaras); d4 = tirarDado(cantCaras);
        // Compruebo si hice generala
        hiceGenerala = esGenerala(d1, d2, d3, d4);
        cont++;
    }
    // Retorno la cantidad de intentos hasta hacer generala
    return cont;
}



// Chequea que se haya hecho generala
public static boolean esGenerala(int d1, int d2, int d3, int d4) {
    return d1 == d2 && d3 == d4;
}

// Tira un dado con una determinada cantidad de caras
public static int tirarDado(int cantCaras) {
    double random = Math.random();
    return (int) random * cantCaras + 1;
}
```

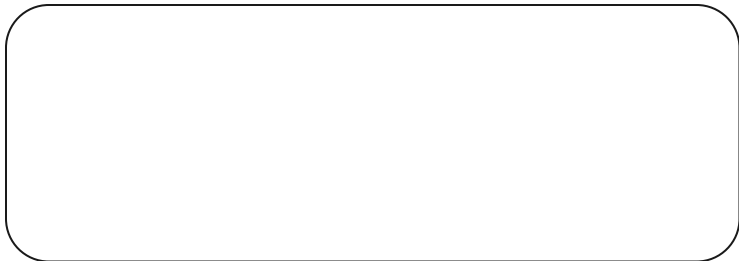
Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```



Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

minCant: 2147483647

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```



minCant: 2147483647

minCaras: 1

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

minCant: 2147483647
minCaras: 1
cant: 0

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Inicializa cantCaras y evalúa la condición cantCaras <= 10


Mapa de memoria RAM

minCant: 2147483647
minCaras: 1
cant: 0
cantCaras: 3

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 2147483647
minCaras: 1
cant: 0
cantCaras: 3

Se debe invocar a la función jugar con *cantCaras* como parámetro

Para llevar el control de las invocaciones a funciones se utiliza una pila de ejecución


Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3

El hilo de ejecución salta a la primera línea de *jugar*, se agrega la invocación a la pila de ejecución y se define el espacio propio de variables. donde *cantCaras* almacena una copia del valor pasado como parámetro


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3
d1:
d2:
d3:
d4:


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3
d1:
d2:
d3:
d4:
cont: 0


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1:
d2:
d3:
d4:
cont: 0

Se evalúa la condición


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1:
d2:
d3:
d4:
cont: 0

Se debe invocar a la función *tirarDado* con *cantCaras* como parámetro

Se debe incorporar el llamado a la pila de ejecución

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
39 // Tira un dado con una determinada cantidad de caras
40 public static int tirarDado(int cantCaras) {
41     double random = Math.random();
42     return (int) random * cantCaras + 1;
43 }
```

Se debe invocar a la función *tirarDado* con *cantCaras* como parámetro

Se debe incorporar el llamado a la pila de ejecución

Mapa de memoria RAM

cantCaras: 3

Pila de ejecución

```
Generala.tirarDado(int)
Generala.jugar(int)
Generala.main(String[])
```

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
39 // Tira un dado con una determinada cantidad de caras
40 public static int tirarDado(int cantCaras) {
41     double random = Math.random();
42     return (int) random * cantCaras + 1;
43 }
```

Se debe invocar a la función `Math.random()` sin parámetro

Se debe incorporar el llamado a la pila de ejecución

Mapa de memoria RAM

cantCaras: 3

Pila de ejecución

```
Generala.tirarDado(int)
Generala.jugar(int)
Generala.main(String[])
```

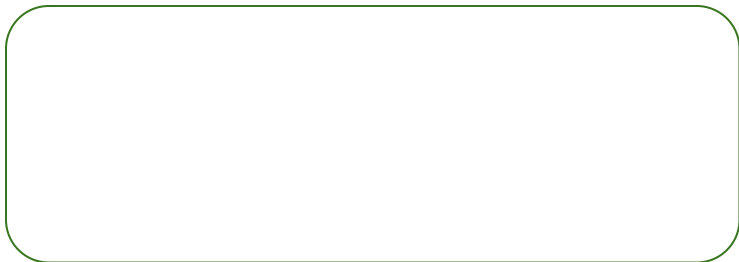
Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

En Math.class

```
185 public static double random() {  
186     // Nos alcanza con saber que retorna un valor double  
187     // mayor o igual a 0.0 y menor que 1.0  
188 }
```

Mapa de memoria RAM



Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

Se debe invocar a la función ***Math.random()*** sin parámetro

Se debe incorporar el llamado a la pila de ejecución

Pila de ejecución

```
Math.random()  
Generala.tirarDado(int)  
Generala.jugar(int)  
Generala.main(String[])
```

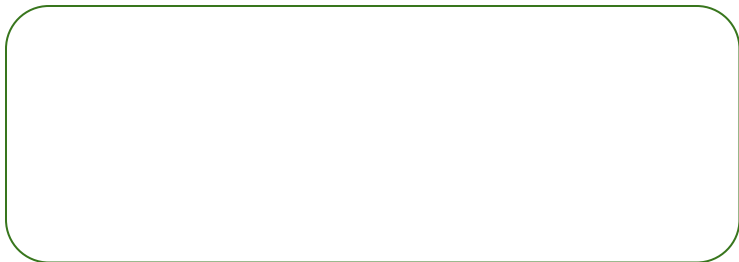
Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

En Math.class

```
185 public static double random() {  
186     // Nos alcanza con saber que retorna un valor double  
187     // mayor o igual a 0.0 y menor que 1.0  
188 }
```

Mapa de memoria RAM



Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

Cuando finaliza la ejecución de la función (return):

- se desapila la invocación
- se desechan las variables creadas en el ámbito de la función
- se retorna el valor, por ejemplo 0.358942014


Pila de ejecución

```
Math.random()  
Generala.tirarDado(int)  
Generala.jugar(int)  
Generala.main(String[])
```

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
39 // Tira un dado con una determinada cantidad de caras
40 public static int tirarDado(int cantCaras) {
41     double random = Math.random();
42     return (int) random * cantCaras + 1;
43 }
```

Se resuelve la expresión $(\text{int}) \text{ random} * \text{cantCaras} + 1$

Se retorna el valor, cual?

- $(\text{int}) 0.358942014 * 3 + 1$
- $0 * 3 + 1$

Mapa de memoria RAM

cantCaras: 3
random: 0.358942014

Pila de ejecución

Generala.tirarDado(int)
Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
39 // Tira un dado con una determinada cantidad de caras
40 public static int tirarDado(int cantCaras) {
41     double random = Math.random();
42     return (int) random * cantCaras + 1;
43 }
```



Mapa de memoria RAM

cantCaras: 3
random: 0.358942014

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

Se resuelve la expresión $(\text{int}) \text{ random} * \text{cantCaras} + 1$

Se retorna el valor, cual?

- $(\text{int}) 0.358942014 * 3 + 1$
- $0 * 3 + 1$




Pila de ejecución

Generala.tirarDado(int)
Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
39 // Tira un dado con una determinada cantidad de caras
40 public static int tirarDado(int cantCaras) {
41     double random = Math.random();
42     return (int) (random * cantCaras) + 1;
43 }
```

Mapa de memoria RAM

cantCaras: 3
random: 0.358942014

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

Se resuelve la expresión $(\text{int}) (\text{random} * \text{cantCaras}) + 1$

Se retorna el valor, cual?

- $(\text{int}) (0.358942014 * 3) + 1$
- $(\text{int}) (1.076826042) + 1$
- $1 + 1$
- 2


Pila de ejecución

Generala.tirarDado(int)
Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2:
d3:
d4:
cont: 0

Se debe invocar a la función *tirarDado* con *cantCaras* como parámetro

Se debe incorporar el llamado a la pila de ejecución... esto se repite en las líneas 29, 30 y 31


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2: 2
d3:
d4:
cont: 0

Se debe invocar a la función *tirarDado* con *cantCaras* como parámetro

Se debe incorporar el llamado a la pila de ejecución... esto se repite en las líneas 29, 30 y 31

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2: 2
d3: 3
d4:
cont: 0

Se debe invocar a la función *tirarDado* con *cantCaras* como parámetro

Se debe incorporar el llamado a la pila de ejecución... esto se repite en las líneas 29, 30 y 31

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2: 2
d3: 3
d4: 3
cont: 0

Se debe invocar a la función *esGenerala* con *d1*, *d2*, *d3* y *d4* como parámetro

Se debe incorporar el llamado a la pila de ejecución

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
44 public static boolean esGenerala(int d1, int d2, int d3, int d4) {  
45     return d1 == d2 && d3 == d4;  
46 }
```

Se resuelve la expresión `d1 == d2 && d3 == d4`

Se retorna el valor, cual?

- `2 == 2 && 3 == 3`
- `true && true`
- `true`



Mapa de memoria RAM

d1: 2
d2: 2
d3: 3
d4: 3

Pila de ejecución

```
Generala.esGenerala(int, int, int, int)  
Generala.jugar(int)  
Generala.main(String[])
```

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
44 public static boolean esGenerala(int d1, int d2, int d3, int d4) {  
45     return d1 == d2 && d2 == d3 && d3 == d4;  
46 }
```

Se resuelve la expresión `d1 == d2 && d2 == d3 && d3 == d4`

Se retorna el valor, cual?

- `2 == 2 && 2 == 3 && 3 == 3`
- `true && false && true`
- `false`

Mapa de memoria RAM

d1: 2
d2: 2
d3: 3
d4: 3

Pila de ejecución

Generala.esGenerala(int, int, int, int)
Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```



Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2: 2
d3: 3
d4: 3
cont: 0


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2: 2
d3: 3
d4: 3
cont: 1

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: 2
d2: 2
d3: 3
d4: 3
cont: 1

Varias iteraciones después.....



Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```



Mapa de memoria RAM

cantCaras: 3 hiceGenerala: false
d1: ~~2~~ ~~3~~ ~~1~~ ~~2~~ ~~3~~ ~~3~~ 2 1
d2: ~~2~~ ~~2~~ ~~1~~ ~~3~~ ~~2~~ ~~2~~ 2 1
d3: ~~3~~ ~~2~~ ~~1~~ ~~3~~ ~~2~~ ~~1~~ 1 1
d4: ~~3~~ ~~2~~ ~~3~~ ~~1~~ ~~1~~ ~~2~~ 3 1
cont: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 7

Varias iteraciones después.....



Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
44 public static boolean esGenerala(int d1, int d2, int d3, int d4) {  
45     return d1 == d2 && d2 == d3 && d3 == d4;  
46 }
```

Se resuelve la expresión `d1 == d2 && d2 == d3 && d3 == d4`

Se retorna el valor, cual?

- `1 == 1 && 1 == 1 && 1 == 1`
- `true && true && true`
- `true`

Mapa de memoria RAM

d1: 1
d2: 1
d3: 1
d4: 1

Pila de ejecución

Generala.esGenerala(int, int, int, int)
Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```



Mapa de memoria RAM

cantCaras: 3 hiceGenerala: ~~false~~ true
d1: ~~2~~ ~~3~~ ~~1~~ ~~2~~ ~~3~~ ~~3~~ 2 1
d2: ~~2~~ ~~2~~ ~~1~~ ~~3~~ ~~2~~ ~~2~~ 2 1
d3: ~~3~~ ~~2~~ ~~1~~ ~~3~~ ~~2~~ ~~1~~ 1 1
d4: ~~3~~ ~~2~~ ~~3~~ ~~1~~ ~~1~~ ~~2~~ 3 1
cont: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 6 7


Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?



```
20 public static int jugar(int cantCaras) {
21     // Defino las variables
22     int d1, d2, d3, d4; // Usamos 4 dados
23     int cont = 0;
24     boolean hiceGenerala = false;
25     // Mientras no haga generala, sigo jugando y cuento
26     while (!hiceGenerala) {
27         // Tiro los dados
28         d1 = tirarDado(cantCaras);
29         d2 = tirarDado(cantCaras);
30         d3 = tirarDado(cantCaras);
31         d4 = tirarDado(cantCaras);
32         // Compruebo si hice generala
33         hiceGenerala = esGenerala(d1, d2, d3, d4);
34         cont++;
35     }
36     // Retorno la cantidad de intentos hasta hacer generala
37     return cont;
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: ~~false~~ true
d1: ~~2 3 1 2 3 3 2~~ 1
d2: ~~2 2 1 3 2 2 2~~ 1
d3: ~~3 2 1 3 2 1 1~~ 1
d4: ~~3 2 3 1 1 2 3~~ 1
cont: ~~0 1 2 3 4 5 6 7~~ 8

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
20 public static int jugar(int cantCaras) {  
21     // Defino las variables  
22     int d1, d2, d3, d4; // Usamos 4 dados  
23     int cont = 0;  
24     boolean hiceGenerala = false;  
25     // Mientras no haga generala, sigo jugando y cuento  
26     while (!hiceGenerala) {  
27         // Tiro los dados  
28         d1 = tirarDado(cantCaras);  
29         d2 = tirarDado(cantCaras);  
30         d3 = tirarDado(cantCaras);  
31         d4 = tirarDado(cantCaras);  
32         // Compruebo si hice generala  
33         hiceGenerala = esGenerala(d1, d2, d3, d4);  
34         cont++;  
35     }  
36     // Retorno la cantidad de intentos hasta hacer generala  
37     return cont;  
38 }
```

Mapa de memoria RAM

cantCaras: 3 hiceGenerala: ~~false~~ true
d1: ~~2 3 1 2 3 3 2~~ 1
d2: ~~2 2 1 3 2 2 2~~ 1
d3: ~~3 2 1 3 2 1 1~~ 1
d4: ~~3 2 3 1 1 2 3~~ 1
cont: ~~0 1 2 3 4 5 6 7~~ 8

Retorna 8

Pila de ejecución

Generala.jugar(int)
Generala.main(String[])

Prueba de escritorio

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 2147483647
minCaras: 1
cant: 8
cantCaras: 3

Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 2147483647
minCaras: 1
cant: 8
cantCaras: 3

Con 3 se tardo 8 rondas en hacer generala.

Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 2147483647
minCaras: 1
cant: 8
cantCaras: 3

Con 3 se tardo 8 rondas en hacer generala.

Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 8
minCaras: 1
cant: 8
cantCaras: 3

Con 3 se tardo 8 rondas en hacer generala.

Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 8
minCaras: 3
cant: 8
cantCaras: 3



Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

minCant: 8 7
minCaras: 3 5
cant: 8 7
cantCaras: ~~3 4 5 6 7 8 9 10~~ 11

Con 3 se tardo 8 rondas en hacer generala.
Con 4 se tardo 82 rondas en hacer generala.
Con 5 se tardo 7 rondas en hacer generala.
Con 6 se tardo 105 rondas en hacer generala.
Con 7 se tardo 259 rondas en hacer generala.
Con 8 se tardo 1072 rondas en hacer generala.
Con 9 se tardo 88 rondas en hacer generala.
Con 10 se tardo 1695 rondas en hacer generala.

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Mapa de memoria RAM

minCant: 8 7
minCaras: 3 5
cant: 8 7
cantCaras: ~~3 4 5 6 7 8 9 10~~ 11

Con 3 se tardo 8 rondas en hacer generala.
Con 4 se tardo 82 rondas en hacer generala.
Con 5 se tardo 7 rondas en hacer generala.
Con 6 se tardo 105 rondas en hacer generala.
Con 7 se tardo 259 rondas en hacer generala.
Con 8 se tardo 1072 rondas en hacer generala.
Con 9 se tardo 88 rondas en hacer generala.
Con 10 se tardo 1695 rondas en hacer generala.

Pila de ejecución

Generala.main(String[])

Prueba de escritorio

¿Qué pasa cuando tenemos múltiples llamados a funciones?

```
1 public static void main(String[] args) {
2     int minCant = Integer.MAX_VALUE;
3     int minCaras = 1;
4     int cant = 0; //auxiliar
5     // Pueba con dados que tienen desde 3 hasta 10 caras
6     for (int cantCaras = 3; cantCaras <= 10; cantCaras++) {
7         cant = jugar(cantCaras);
8         System.out.println("Con " + cantCaras + " se tardo "
9             + cant + " rondas hacer generala.");
10        if (minCant >= cant) {
11            minCant = cant;
12            minCaras = cantCaras;
13        }
14    }
15
16    System.out.println("Se hizo generala en menos intentos con
17        dados de " + minCaras + " caras.");
18 }
```

Juega a hacer generala (tirar los dados todos con el mismo valor) con dados de distinta cantidad de caras. Informa con cuales dados lo logra en menos intentos.

Con 3 se tardo 8 rondas en hacer generala.
Con 4 se tardo 82 rondas en hacer generala.
Con 5 se tardo 7 rondas en hacer generala.
Con 6 se tardo 105 rondas en hacer generala.
Con 7 se tardo 259 rondas en hacer generala.
Con 8 se tardo 1072 rondas en hacer generala.
Con 9 se tardo 88 rondas en hacer generala.
Con 10 se tardo 1695 rondas en hacer generala.
Se hizo generala en menos intentos con dados de 5 caras.



Selección de valores de prueba

- Pruebas aleatorias
- Pruebas límite
- Pruebas basadas en especificaciones
- Pruebas basadas en escenarios
- Pruebas basadas en datos históricos

ACLARACIÓN IMPORTANTE:
Que el programa sea correcto para algunos casos, no indica que lo sea para TODOS.

Impresión por consola 1/2



Otra técnica muy usada cuando no tenemos posibilidades de usar una herramienta de debugging, es imprimir por consola valores de una variable para chequear si es correcto o no; o imprimir si entra en una condición o no.

Ejemplo: Tenemos un método que permite calcular el descuento aplicado cuando el importe es mayor a 230 y menor a 500.

Porción de código de interés

```
if (importe > 230 && importe < 500) {  
    System.out.println("Entró a la rama verdadera");  
    System.out.println("El valor de importe es: " + importe);  
    aplicarDescuento();  
}  
else {  
    System.out.println("No se cumplió la condición");  
}
```


Impresión por consola 1/2



Porción de código de interés

```
int suma=0, promedio=0, N=3, entero=0;
for (int i=0; i<=N;i++){
    entero = obtenerEntero();
    System.out.println("El valor que vino de la función es: " + entero);
    suma+=entero;
    System.out.println("La suma parcial es: " + suma);
}
System.out.println("La suma total fue: " + suma)
promedio=suma/N;
System.out.println("El promedio fue: " + promedio)
```

Primer Bug

El 9 de septiembre de 1947 la computadora Mark II, en la Universidad de Harvard (EE. UU.), sufrió una avería. Tras la inspección, los ingenieros diagnosticaron la causa: una polilla se había acercado a la máquina, tal vez atraída por la luz y el calor, y había cortocircuitado el relé número 70 del Panel F. Los técnicos dieron cuenta del incidente en su cuaderno con una entrada a las 15:45 en la que fijaron el insecto a la página con cinta adhesiva y anotaron: "Primer caso real de un bug [bicho] encontrado"

92

9/9

0800 Arctan started
 1000 " stopped - arctan ✓
 1300 (032) HP - MC 1.98264000
 (033) PRO 2 2.130476415
 convd 2.130676415

Relays 6-2 in 033 failed special speed test
 in relay 11.000 test.

1100 Started Cosine Tape (Sine check)
 1525 Started Multi-Adder Test.

1545



Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.
 1630 Arctan started.
 1700 closed down.

Relay
 2145
 Relay 8370

Debugging

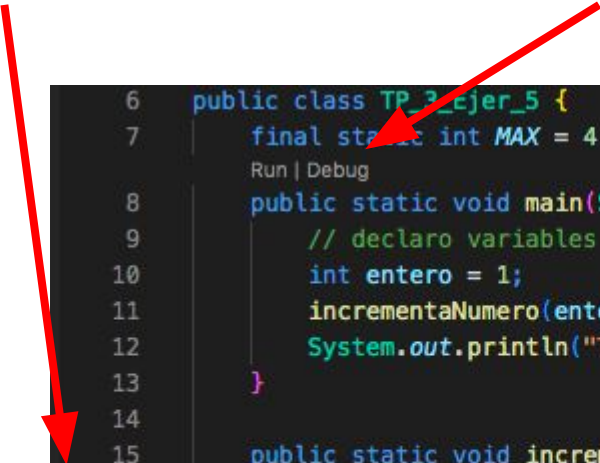
Los IDE's cuentan con una herramienta de debugging que permite ver la evolución del código paso a paso mostrando el estado de las variables en memoria RAM.

En general la mayoría lo hacen de formas similares, vamos a ver dos casos:



Debugging en VSCode 1/2

Para poder debuggear en VSCode deberán incluir al menos un **breakpoint** (se pueden poner n) al lado del número de línea como se ve a continuación (punto rojo). Luego presionar donde dice **Debug**



```
6 public class TP_2_Ejer_5 {
7     final static int MAX = 4;
8     public static void main(String[] args) {
9         // declaro variables y/o constantes
10        int entero = 1;
11        incrementaNumero(entero);
12        System.out.println("Terminó");
13    }
14
15    public static void incrementaNumero(int entero) {
16        for (int i = entero; i <= MAX; i++)
17            System.out.println("Numero entero: " + i);
18    }
19 }
```

The image shows a VS Code editor window with a Java file named `TP_2_Ejer_5.java`. The code contains a `main` method and an `incrementaNumero` method. A red dot, representing a breakpoint, is placed on the left margin next to line 16, which is the first line of the `for` loop in `incrementaNumero`. Two red arrows are present: one points from the top right towards the `Run | Debug` button in the editor's toolbar, and another points from the top left towards the red breakpoint dot on line 16.

Debugging en VSCode 2/2

Aparecerá un panel para interactuar con el debug en curso y la ejecución se detendrá en la línea que tiene el breakpoint

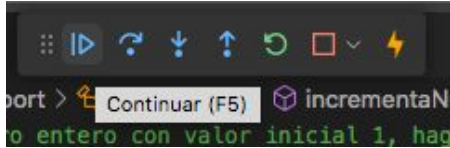
The screenshot shows the VS Code interface during a debug session. On the left, the **VARIABLES** panel is expanded, showing a local variable `entero` with the value `1`. A red arrow points from the text "Se muestran los valores de las variables o constantes locales involucradas" to this panel. In the center, the code editor displays a Java file `TP_3_Ejer_5.java`. A breakpoint is set on line 15, which is highlighted in green. A red arrow points from the text "Se muestran todos los valores de las variables o constantes involucradas (locales y globales)" to this line. The code in the editor is as follows:

```
1  /*Hacer un método que dado un número entero con valor inicial 1, haga una iteración incrementando el número
2  de a uno hasta un valor MAX = 4 (constante). Mientras itera deberá imprimir el número. Luego invocarlo desde
3  el programa principal y cuando termina imprimir por pantalla "terminó".
4
5  */
6
7  public class TP_3_Ejer_5 {
8      final static int MAX = 4;
9
10     public static void main(String[] args) {
11         // declaro variables y/o constantes
12         int entero = 1;
13         incrementaNumero(entero);
14         System.out.println("Terminó");
15     }
16
17     public static void incrementaNumero(int entero) {
18         entero = 1;
19         for (int i = entero; i <= MAX; i++) {
20             entero = 1, MAX = 4;
21             System.out.println("Numero entero: " + i);
22         }
23     }
24 }
```

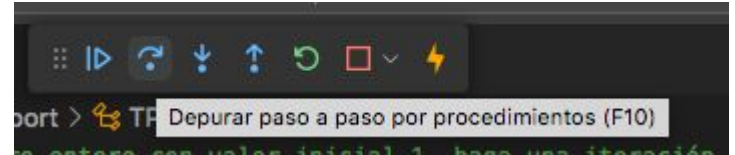
On the right, the **DEBUG CONSOLE** shows the current step in the debug process, indicating that the execution is stopped at the breakpoint on line 15.

Debugging en VSCode - Panel

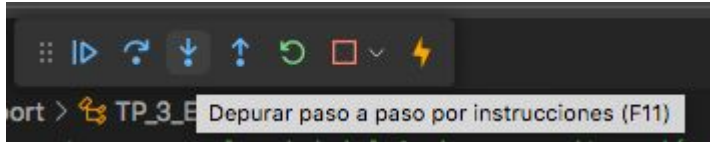
Continúa la ejecución normalmente, pero se detiene en el próximo breakpoint



Resuelve por método pero al salir para y sigue paso a paso



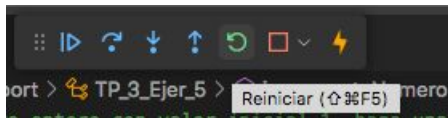
Entra al método para seguir paso a paso dentro



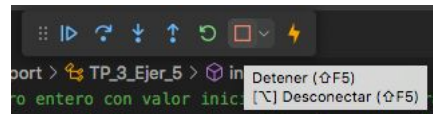
Sale de la depuración en el lugar que se encuentre y sigue luego del breakpoint



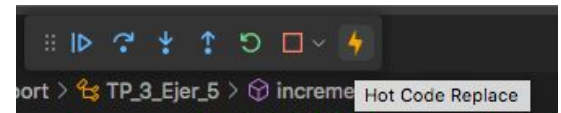
Reinicia la depuración



Detiene la depuración



Si se cambia código lo actualiza en el debug actual



Pila de ejecución en VSCode 1/3

Sirve para ver cómo se realiza el llamado a métodos en tiempo de ejecución

El main en la línea 14 llamó a obtenerNumero() que arranca en 22

Se van apilando los llamados a métodos, desde el main en adelante

The screenshot shows the VS Code interface during a debug session. The left sidebar displays the 'Ejecución y Depuración' (Execution and Debugging) view, which includes the 'Pila de Llamadas' (Call Stack) section. The call stack shows the following frames:

- Thread [Signal Dispatcher] (EN EJECUCIÓN)
- Thread [Finalizer] (EN EJECUCIÓN)
- Thread [Reference Handler] (EN EJECUCIÓN)
- Thread [main] (EN PAUSA EN STEP)
 - TP_3_Ejer_7.obtenerNumero() TP_3_Ejer_7.java 22:1
 - TP_3_Ejer_7.main(String[]) TP_3_Ejer_7.java 14:1

The main editor displays the source code of TP_3_Ejer_7.java. The code is as follows:

```
TP 3 > TP_3_Ejer_7.java > Java Language Support > TP_3_Ejer_7 > obtenerNumero
6 import java.io.BufferedReader;
7 import java.io.InputStreamReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     Run | Debug
13     public static void main(String[] args) {
14         // declaro variables y/o constantes
15         int numero = obtenerNumero();
16         while (numero != 0) {
17             sumatoria();
18             numero = obtenerNumero();
19         }
20     }
21
22     public static int obtenerNumero() {
23         int numero = 0;
24         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
25         try {
26             System.out.println("Ingrese un numero distinto de 0");
27             numero = Integer.valueOf(entrada.readLine());
28         } catch (Exception e) {
29             System.out.println(e);
30         }
31     }
32 }
```

A red arrow points from the text 'El main en la línea 14 llamó a obtenerNumero() que arranca en 22' to the 'TP_3_Ejer_7.main(String[])' frame in the call stack. Another red arrow points from the text 'Se van apilando los llamados a métodos, desde el main en adelante' to the 'TP_3_Ejer_7.obtenerNumero()' frame in the call stack.

Pila de ejecución en VSCode 2/3

Se puede ir siguiendo paso a paso y viendo como está la pila luego de cada sentencia

Puedo ver el estado de la variable número (local al main) que es 3

Cuando termina obtenerNumero() se desapila

The screenshot shows the VS Code interface during a Java program execution. The left sidebar displays the 'Ejecución y Depuración' (Execution and Debugging) view. Under 'VARIABLES', the 'Local' scope shows 'args: String[0]@6' and 'numero: 3'. Under 'INSPECCIÓN', the 'PILA DE LLAMADAS' (Call Stack) shows the current thread 'main' at 'TP_3_Ejer_7.main(String[])'. The main editor shows the source code of 'TP_3_Ejer_7.java'. The code includes a 'main' method and an 'obtenerNumero()' method. The 'while' loop in 'main' is currently executing, with 'numero' set to 3. The 'obtenerNumero()' method is shown below, which reads input from the console. The bottom status bar shows 'TERMINAL' with the output of the program, including the prompt 'Ingrese un numero distinto de 0' and the input '3'.

```
TP 3 > J TP_3_Ejer_7.java > Java Language Support > TP_3_Ejer_7 > main
3 naturales (son números enteros entre 1 y 200).
4
5
6 import java.io.BufferedReader;
7 import java.io.InputStreamReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     Run | Debug
13     public static void main(String[] args) {
14         // declaro variables y/o constantes
15         int numero = obtenerNumero(); numero = 3
16         while (numero != 0) {
17             sumatoria();
18             numero = obtenerNumero();
19         }
20
21     public static int obtenerNumero() {
22         int numero = 0;
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (IOException e) {
28             e.printStackTrace();
29         }
30     }
31 }
32
33 erver=n,suspend=y,address=localhost:52093 -cp /Users/juantoloz/Library/Application\ Support/Code/User/workspaceStorage/7ecb9f0c41aa4f0428fb171c5fe26ae7/redhat.java/jdt_ws/2023_ff9a72f9/bin TP_3_Ejer_7
(base) MacBook-Air-de-juan-35487:2023 juantoloz /usr/bin/env /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:52093 -cp /Users/juantoloz/Library/Application\ Support/Code/User/workspaceStorage/7ecb9f0c41aa4f0428fb171c5fe26ae7/redhat.java/jdt_ws/2023_ff9a72f9/bin TP_3_Ejer_7
Ingrese un numero distinto de 0
3
```


Pila de ejecución en VSCode 3/3

Llama a
sumatoria y
se apila

Ejecuta el procedimiento
sumatoria()

The screenshot shows the VS Code interface during a Java program execution. The left sidebar displays the 'Ejecución y Depuración' (Execution and Debugging) view with the 'Pila de Llamadas' (Call Stack) expanded. The call stack shows the following frames:

- Thread [Signal Dispatcher] (EN EJECUCIÓN)
- Thread [Finalizer] (EN EJECUCIÓN)
- Thread [Reference Handler] (EN EJECUCIÓN)
- Thread [main] (EN PAUSA EN STEP)
- TP_3_Ejer_7.sumatoria() TP_3_Ejer_7.java 34:1
- TP_3_Ejer_7.main(String[]) TP_3_Ejer_7.java 16:1

The main editor shows the source code of TP_3_Ejer_7.java. The current line of execution is line 34, which is the start of the `sumatoria()` method. The code is as follows:

```
14 int numero = obtenerNumero();
15 while (numero != 0) {
16     sumatoria();
17     numero = obtenerNumero();
18 }
19
20
21 public static int obtenerNumero() {
22     int numero = 0;
23     BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24     try {
25         System.out.println("Ingrese un numero distinto de 0");
26         numero = Integer.valueOf(entrada.readLine());
27     } catch (Exception e) {
28         System.out.println(e);
29     }
30     return numero;
31 }
32
33 public static void sumatoria() {
34     int suma = 0;
35     System.out.println("Sumatoria de los primeros 200 números naturales");
36     for (int i = 1; i <= MAX; i++)
37         suma += i;
```

At the bottom, the 'TERMINAL' tab shows the command used to run the program:

```
server=n,suspend=y,address=localhost:52093 -cp /Users/juantoloz/Library/Application\ Support/Code/User/workspaceStor
ge/7ecb9f0c41aa4f0428fb171c5fe26ae7/redhat.java/jdt_ws/2023_ff9a72f9/bin TP_3_Ejer_7
(base) MacBook-Air-de-juan-35487:2023 juantoloz$ /usr/bin/env /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents
Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:52093 -cp /Users/juantoloz/Li
```

Estado de las variables en VSCode 1/2

Veo el estado de las variables locales a obtenerNumero() que está en la pila de ejecución y su valor es 0 al inicio

VARIABLES

- Local
 - numero: 0

INSPECCIÓN

PILA DE LLAMADAS

Thread	Method	File	Line	Status
Thread [main]	TP_3_Ejer_7.obtenerNumero()	TP_3_Ejer_7.java	23:1	EN PAUSA EN STEP
Thread [Signal Dispatcher]				EN EJECUCIÓN
Thread [Finalizer]				EN EJECUCIÓN
Thread [Reference Handler]				EN EJECUCIÓN

```
TP 3 > J TP_3_Ejer_7.java > Java Language Support > TP_3_Ejer_7 > obtenerNumero
6 import java.io.BufferedReader;
7 import java.io.InputStreamReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     Run | Debug
13     public static void main(String[] args) {
14         // declaro variables y/o constantes
15         int numero = obtenerNumero();
16         while (numero != 0) {
17             sumatoria();
18             numero = obtenerNumero();
19         }
20
21     public static int obtenerNumero() {
22         int numero = 0; numero = 0
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (Exception e) {
28             System.out.println(e);
29         }
30     }
31 }
```

Ejecuta el procedimiento obtenerNumero()

PROBLEMAS 12 SALIDA CONSOLA DE DEPURACIÓN TERMINAL

(base) MacBook-Air-de-juan-35487:2023 juantoloza\$ /usr/bin/env /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:52093 -cp /Users/juantoloza/Library/Application\ Support/Code/User/workspaceStorage/7ecb9f0c41aa4f0428fb171c5fe26ae7/redhat.java/jdt ws/2023 ff9a72

Estado de las variables en VSCode 2/2



Si tuviera más variables o constantes puedo ver sus estados

EJECUCIÓN Y DEPURACIÓN TP_3_Ejer_5

VARIABLES

Local

- args: String[0]@6
- cantPacientes: 15
- cantFem: 10
- cantMasc: 10
- mayorA: 78
- cantPrestadores: 10

INSPECCIÓN

PILA DE LLAMADAS

- Thread [Signal Dispatcher] EN EJECUCIÓN
- Thread [Finalizer] EN EJECUCIÓN
- Thread [Reference Handler] EN EJECUCIÓN
- Thread [main] EN PAUSA EN STEP

TP_2_BonusDR_1.main(String[]) TP_2_BonusDR_1.ja...

TP 2 > J TP_2_BonusDR_1.java > Java Language Support > TP_2_BonusDR_1 > main

```
20
21 public class TP_2_BonusDR_1 {
22     public static void main(String[] args) { args = String[0]@6
23         // declaro variables y/o constantes
24         final int cantPacientes = 15, cantFem = 10, cantMasc = 10, mayorA = 78, cantPrestadores = 10, cantPacientesUltimos2Dias = 10;
25
26         char opcion = ' ';
27         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
28         try {
29             System.out.println("Ingrese una opción válida entre 1 y 4, con 0 termina");
30             opcion = entrada.readLine().charAt(0);
31             while (opcion != '0') {
32                 switch (opcion) {
33                     case '1':
34                         int cantPac = 0;
35                         System.out.println("Ingrese la cantidad de pacientes (no mayor a 15)");
36                         cantPac = Integer.valueOf(entrada.readLine());
37                         if (cantPac > 0 && cantPac <= cantPacientes) {
38                             int mayores78 = 0, anios = 0;
39                             while (cantPac > 0) {
40                                 System.out.println("Ingrese las edades (mayor a cero)");
41                                 anios = Integer.valueOf(entrada.readLine());
42                                 if (anios > mayorA)
```

PROBLEMAS 12 SALIDA CONSOLA DE DEPURACIÓN TERMINAL

/usr/bin/env /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java -agentlib:jdwp=transport=dt_socket,

Debugging en Eclipse 1/3

```
6 import java.io.BufferedReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     public static void main(String[] args) {
13         // declaro variables y/o constantes
14         int numero = obtenerNumero();
15         while (numero != 0) {
```

click derecho y selecciono
Toggle Breakpoint parado en
la línea de interés

• Toggle Breakpoint

Toggle Breakpoint Enablement

⇒ Run to Line

Go to Annotation

Switch to Theme...

☒ Validate

Add Bookmark...

Add Task...

☒ Show Quick Diff

☒ Show Line Numbers

Folding

Preferences...

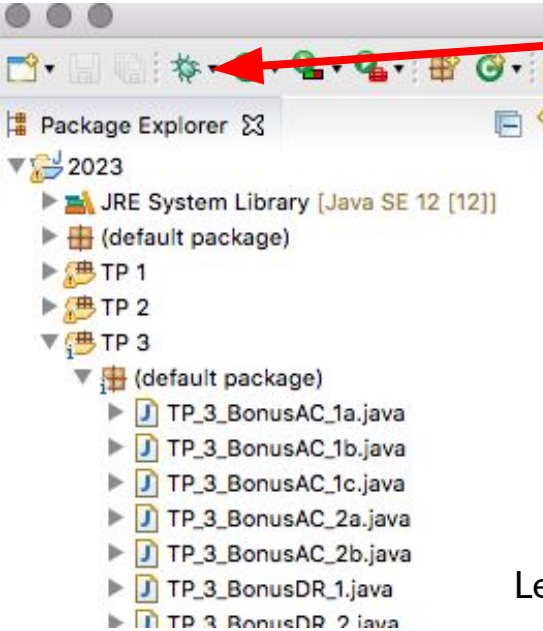
Breakpoint Properties...

```
39 }
40 }
```

```
5
6 import java.io.BufferedReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     public static void main(String[] args) {
13         // declaro variables y/o constantes
14         int numero = obtenerNumero();
15         while (numero != 0) {
16             sumatoria();
17             numero = obtenerNumero();
18         }
19     }
20
21     public static int obtenerNumero() {
22         int numero = 0;
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (Exception e) {
28             System.out.println(e);
29         }
30         return numero;
31     }
32
33     public static void sumatoria() {
34         int suma = 0;
35         System.out.println("Sumatoria de los primeros 200 números naturales");
36         for (int i = 1; i <= MAX; i++)
37             suma += i;
38         System.out.println("La suma es: " + suma);
39     }
40 }
```

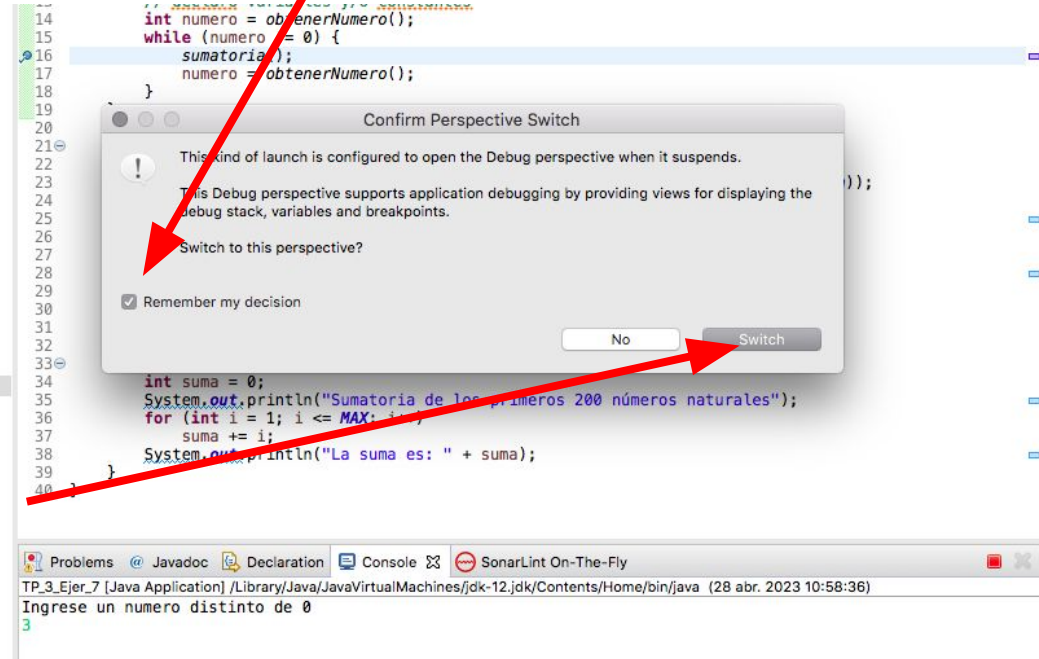
O puedo hacer doble click al
costado del número de línea
y se va a mostrar el
breakpoint en celeste/azul
puedo poner 1 o mas

Debugging en Eclipse 2/3



Luego hago click sobre “la polilla” (según cuenta la historia)

Le dan click en switch



Les advierte que va a pasar a la perspectiva de debugging y puede seleccionar que recuerde esta decisión así no les pregunta cada vez que debuggean

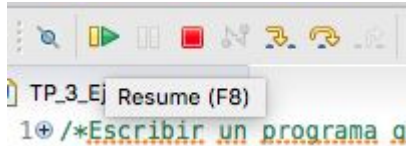

```
public class TP_3_Ejer_7 {
    final static int MAX = 200;

    public static void main(String[] args) {
        // declare variables y/o constantes
        int numero = obtenerNumero();
        while (numero != 0) {
            obtenerA()
        }
    }
}
```

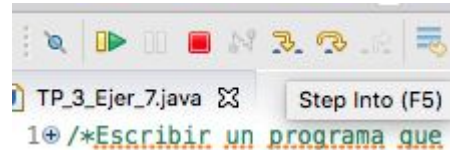
Debugging en Eclipse - Panel

Resuelve por método pero al salir para y sigue paso a paso

Continúa la ejecución normalmente como si no hubiese un breakpoint, si hubiese una iteración para en la próxima pasada



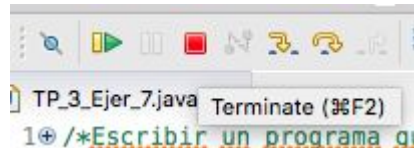
Entra al método para seguir paso a paso dentro



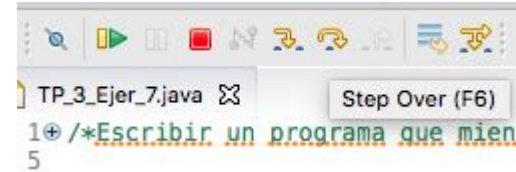
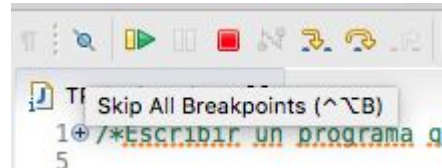
Reinicia la depuración



Detiene la depuración



Salta todos los breakpoints



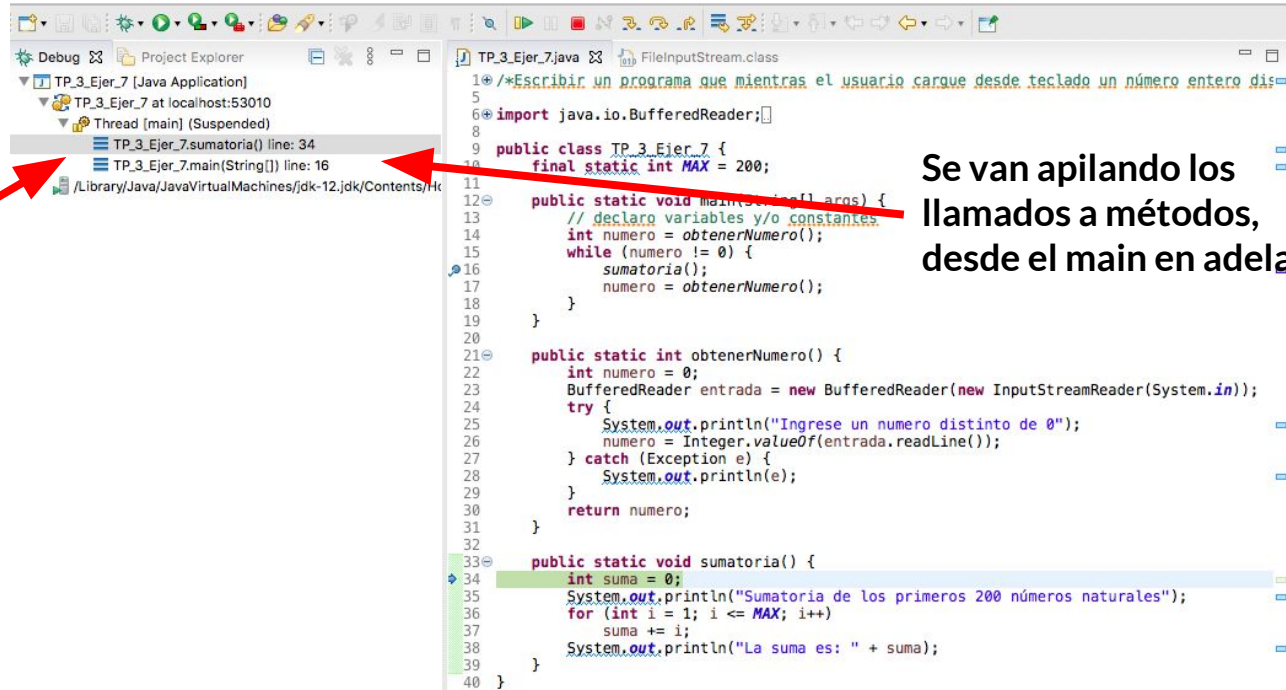
Sale de la depuración en el lugar que se encuentre y sigue luego del breakpoint

Si se cambia código lo actualiza en el debug actual



Pila de ejecución en Eclipse 1/3

Sirve para ver cómo se realiza el llamado a métodos en tiempo de ejecución



The screenshot shows the Eclipse IDE interface. On the left, the 'Debug' console displays the execution stack. The top entry is 'TP_3_Ejer_7.sumatoria() line: 34', which is highlighted with a red arrow. Below it is 'TP_3_Ejer_7.main(String[]) line: 16', also highlighted with a red arrow. The main editor on the right shows the source code of 'TP_3_Ejer_7.java'. The code includes a comment at line 1: '/*Escribir un programa que mientras el usuario cargue desde teclado un número entero dis...'. The code defines a class 'TP_3_Ejer_7' with a static final variable 'MAX' set to 200. The 'main' method (line 12) calls 'obtenerNumero()' and then 'sumatoria()'. The 'obtenerNumero()' method (line 21) reads input from the user and returns it. The 'sumatoria()' method (line 33) calculates the sum of the first 200 natural numbers. The code is as follows:

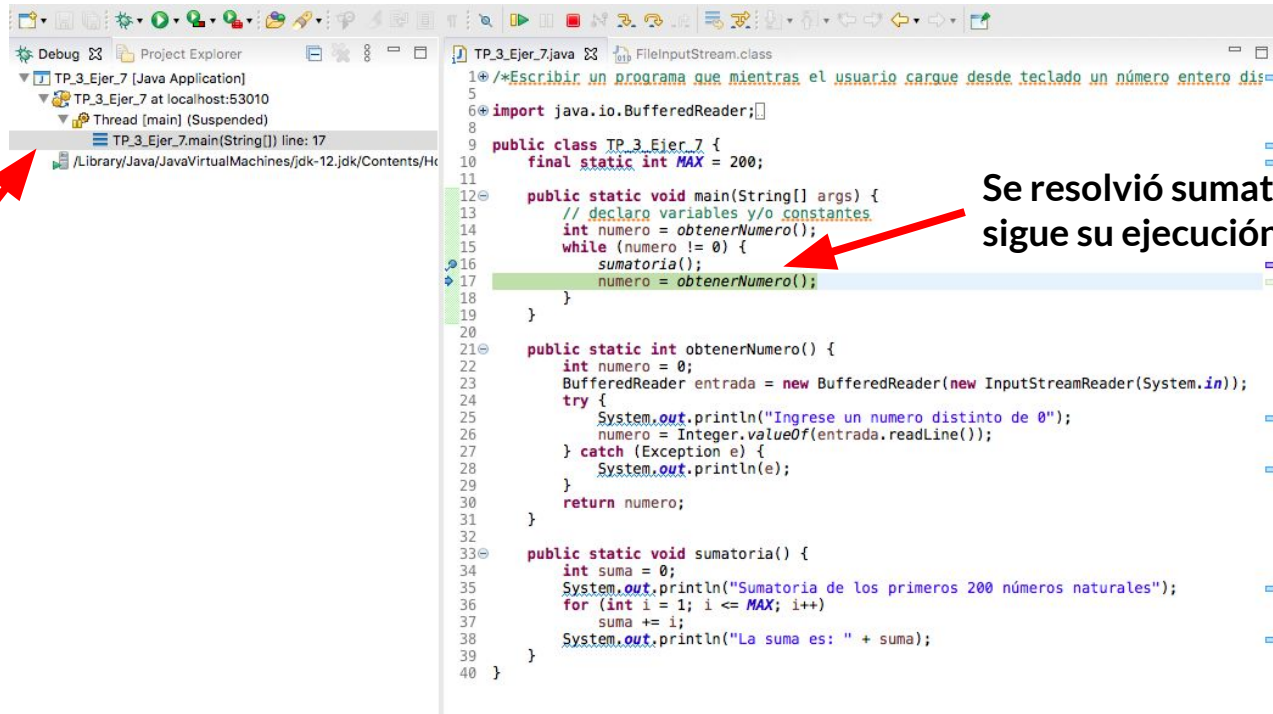
```
1  /*Escribir un programa que mientras el usuario cargue desde teclado un número entero dis...
2
3
4
5
6  import java.io.BufferedReader;
7
8
9  public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     public static void main(String[] args) {
13         // declare variables y/o constantes
14         int numero = obtenerNumero();
15         while (numero != 0) {
16             sumatoria();
17             numero = obtenerNumero();
18         }
19     }
20
21     public static int obtenerNumero() {
22         int numero = 0;
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (Exception e) {
28             System.out.println(e);
29         }
30         return numero;
31     }
32
33     public static void sumatoria() {
34         int suma = 0;
35         System.out.println("Sumatoria de los primeros 200 números naturales");
36         for (int i = 1; i <= MAX; i++)
37             suma += i;
38         System.out.println("La suma es: " + suma);
39     }
40 }
```

Se van apilando los llamados a métodos, desde el main en adelante

El main en la línea 16 llamó a sumatoria() que arranca en 34

Pila de ejecución en Eclipse 2/3

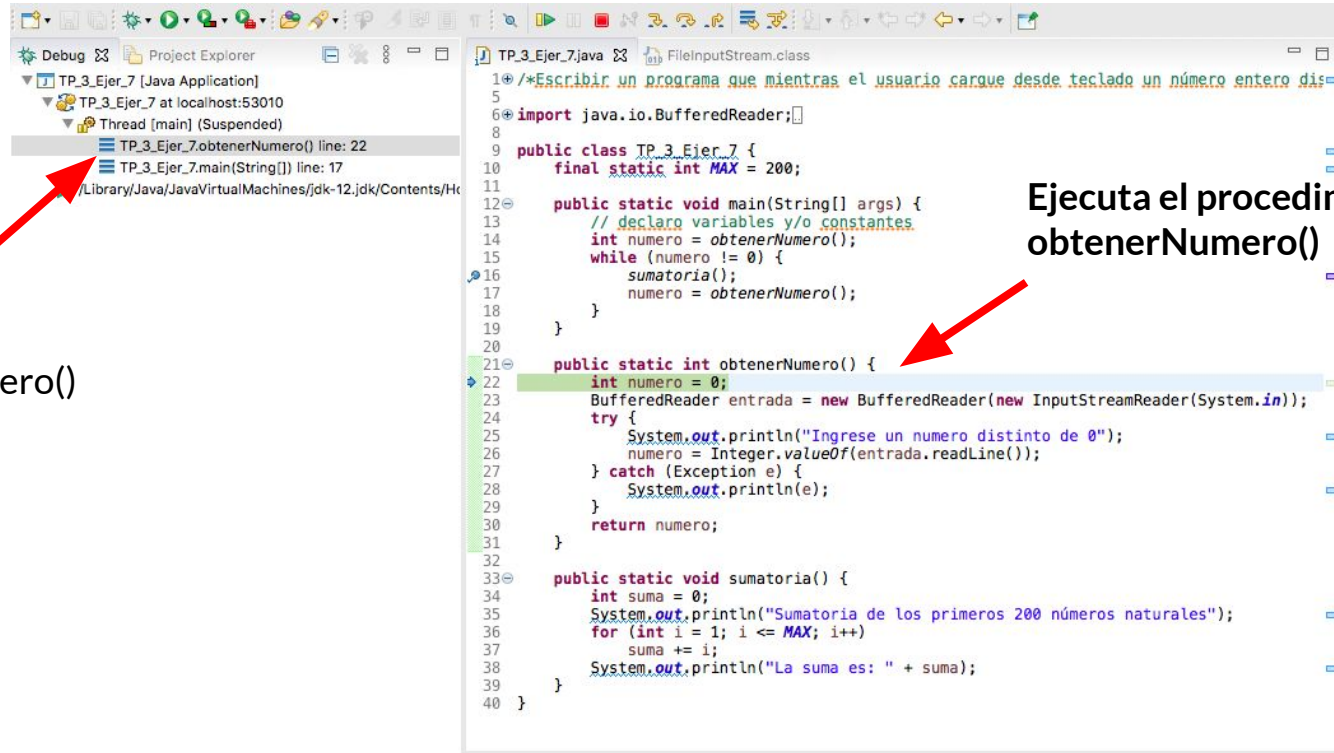
Se puede ir siguiendo paso a paso y viendo como está la pila luego de cada sentencia



Cuando termina
sumatoria() se
desapila

Se resolvió sumatoria() y
sigue su ejecución

Pila de ejecución en Eclipse 3/3



The screenshot shows the Eclipse IDE interface. On the left, the 'Debug' console displays the execution stack. The top entry is 'TP_3_Ejer_7 [Java Application]' at 'localhost:53010'. Below it, the stack is expanded to show 'Thread [main] (Suspended)' and 'TP_3_Ejer_7.obtenerNumero() line: 22'. A red arrow points from the text 'Llama a obtenerNumero() y se apila' to this entry. The main editor shows the source code of 'TP_3_Ejer_7.java'. The code includes a 'main' method that calls 'obtenerNumero()' in a 'while' loop. The 'obtenerNumero()' method is highlighted in blue, and a red arrow points from the text 'Ejecuta el procedimiento obtenerNumero()' to it. The 'sumatoria()' method is also visible below.

```
1  /*Escribir un programa que mientras el usuario cargue desde teclado un número entero dis=
2
3
4
5
6  import java.io.BufferedReader;
7
8
9  public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     public static void main(String[] args) {
13         // declaro variables y/o constantes
14         int numero = obtenerNumero();
15         while (numero != 0) {
16             sumatoria();
17             numero = obtenerNumero();
18         }
19     }
20
21     public static int obtenerNumero() {
22         int numero = 0;
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (Exception e) {
28             System.out.println(e);
29         }
30         return numero;
31     }
32
33     public static void sumatoria() {
34         int suma = 0;
35         System.out.println("Sumatoria de los primeros 200 números naturales");
36         for (int i = 1; i <= MAX; i++)
37             suma += i;
38         System.out.println("La suma es: " + suma);
39     }
40 }
```

Llama a
obtenerNumero()
y se apila

Ejecuta el procedimiento
obtenerNumero()

Estado de las variables en Eclipse 1/4

El estado de las variables es en el procedimiento `sumatoria()` según la pila de ejecución

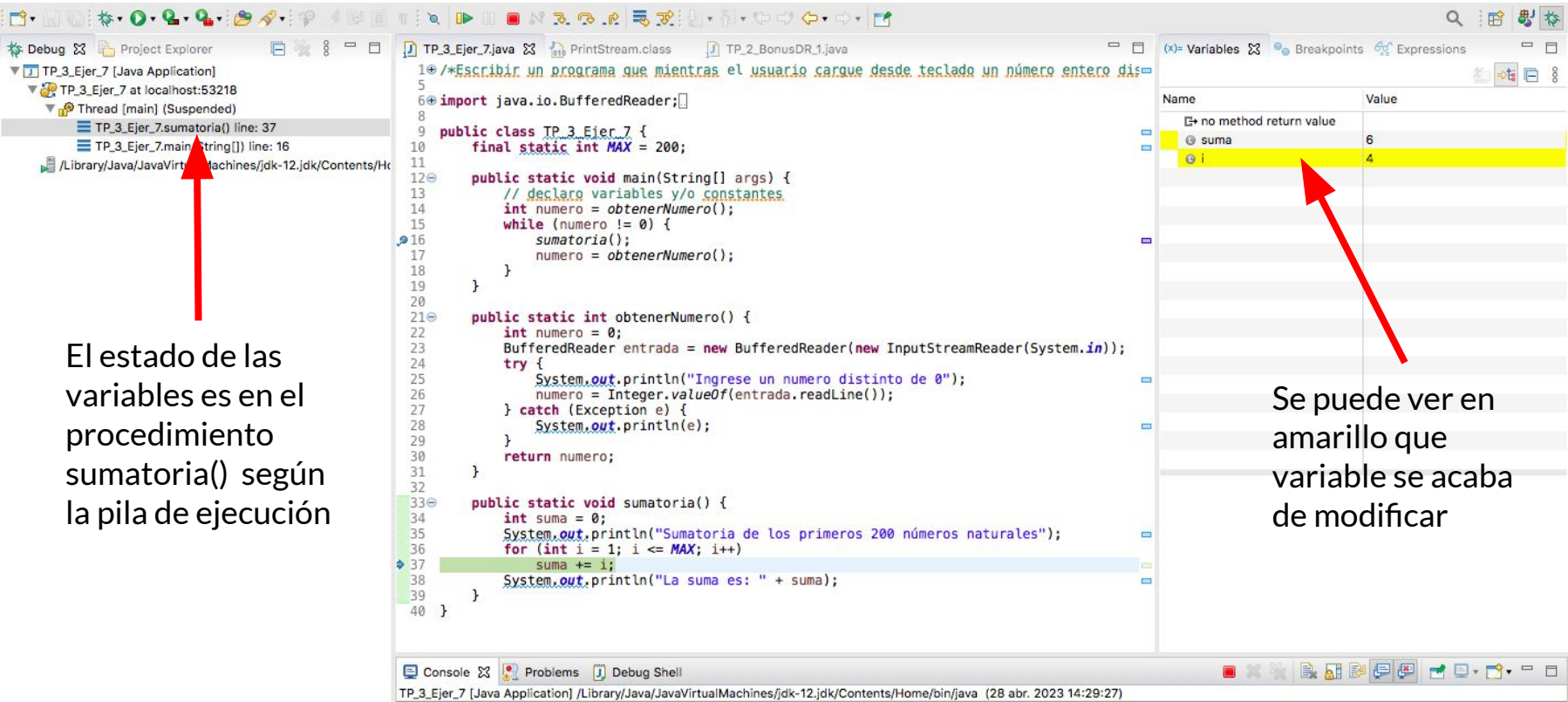
```
10 /*Escribir un programa que mientras el usuario cargue desde teclado un número entero dis=
5
6 import java.io.BufferedReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     public static void main(String[] args) {
13         // declaro variables y/o constantes
14         int numero = obtenerNumero();
15         while (numero != 0) {
16             sumatoria();
17             numero = obtenerNumero();
18         }
19     }
20
21     public static int obtenerNumero() {
22         int numero = 0;
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (Exception e) {
28             System.out.println(e);
29         }
30         return numero;
31     }
32
33     public static void sumatoria() {
34         int suma = 0;
35         System.out.println("Sumatoria de los primeros 200 números naturales");
36         for (int i = 1; i <= MAX; i++)
37             suma += i;
38         System.out.println("La suma es: " + suma);
39     }
40 }
```

Name	Value
no method return value	
suma	0

Se puede ver el valor de `suma` dentro del procedimiento

TP_3_Ejer_7 [Java Application] /Library/Java/JavaVirtualMachines/jdk-12.jdk/Contents/Home/bin/java (28 abr. 2023 14:29:27)

Estado de las variables en Eclipse 2/4



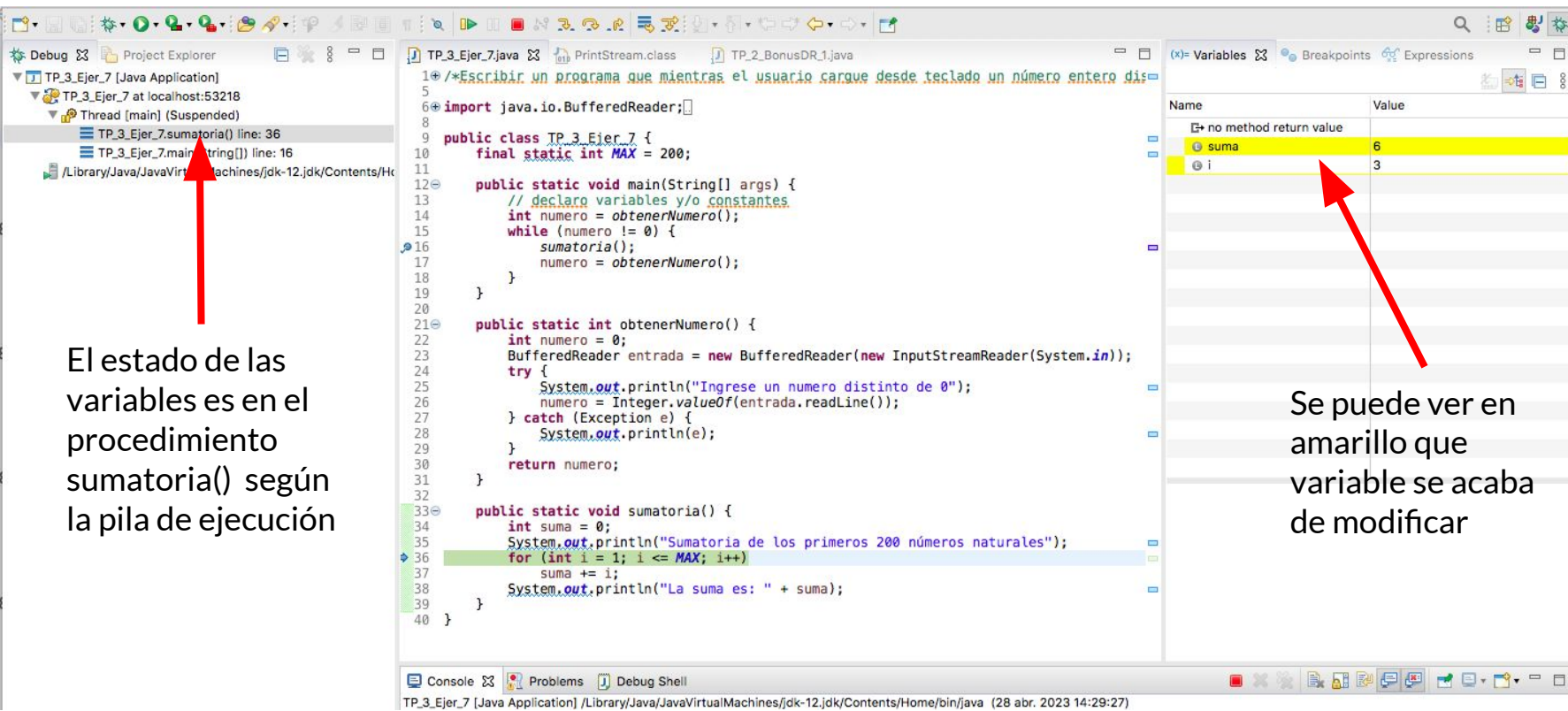
El estado de las variables es en el procedimiento `sumatoria()` según la pila de ejecución

```
1 /*Escribir un programa que mientras el usuario cargue desde teclado un número entero dis=
5
6 import java.io.BufferedReader;
8
9 public class TP_3_Ejer_7 {
10     final static int MAX = 200;
11
12     public static void main(String[] args) {
13         // declaro variables y/o constantes
14         int numero = obtenerNumero();
15         while (numero != 0) {
16             sumatoria();
17             numero = obtenerNumero();
18         }
19     }
20
21     public static int obtenerNumero() {
22         int numero = 0;
23         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24         try {
25             System.out.println("Ingrese un numero distinto de 0");
26             numero = Integer.valueOf(entrada.readLine());
27         } catch (Exception e) {
28             System.out.println(e);
29         }
30         return numero;
31     }
32
33     public static void sumatoria() {
34         int suma = 0;
35         System.out.println("Sumatoria de los primeros 200 números naturales");
36         for (int i = 1; i <= MAX; i++)
37             suma += i;
38         System.out.println("La suma es: " + suma);
39     }
40 }
```

Name	Value
no method return value	
suma	6
i	4

Se puede ver en amarillo que variable se acaba de modificar

Estado de las variables en Eclipse 3/4



The screenshot shows the Eclipse IDE with a Java application running in debug mode. The left sidebar shows the Project Explorer with the file `TP_3_Ejer_7.sumatoria()` selected. The main editor shows the source code of `TP_3_Ejer_7.java`. The right sidebar shows the Variables view with a table of current variables.

Code Snippet:

```
14 // declaro variables y/o constantes
15 int numero = obtenerNumero();
16 while (numero != 0) {
17     sumatoria();
18     numero = obtenerNumero();
19 }
21 public static int obtenerNumero() {
22     int numero = 0;
23     BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
24     try {
25         System.out.println("Ingrese un numero distinto de 0");
26         numero = Integer.valueOf(entrada.readLine());
27     } catch (Exception e) {
28         System.out.println(e);
29     }
30     return numero;
31 }
33 public static void sumatoria() {
34     int suma = 0;
35     System.out.println("Sumatoria de los primeros 200 números naturales");
36     for (int i = 1; i <= MAX; i++)
37         suma += i;
38     System.out.println("La suma es: " + suma);
39 }
40 }
```

Variables View:

Name	Value
↳ no method return value	
suma	6
i	3

El estado de las variables es en el procedimiento `sumatoria()` según la pila de ejecución

Se puede ver en amarillo que variable se acaba de modificar

Estado de las variables en Eclipse 4/4

The screenshot shows the Eclipse IDE with a Java application named "TP_2_BonusDR_1" running in debug mode. The main thread is suspended at line 25 of "TP_2_BonusDR_1.java". The "Variables" view on the right displays the current state of the variables in the scope of the main method.

Variables View:

Name	Value
no method return value	
args	String[0] (id=18)
cantPacientes	15
cantFem	10
cantMasc	10
mayorA	78
cantPrestadores	10

Code Snippet:

```
17
18 import java.io.BufferedReader;
19
20 public class TP_2_BonusDR_1 {
21     public static void main(String[] args) {
22         // declaro variables y/o constantes
23         final int cantPacientes = 15, cantFem = 10, cantMasc = 10, mayorA = 78, cantPre
24         cantPacientesUltimos2Dias = 10;
25         char opcion = ' ';
26         BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
27         try {
28             System.out.println("Ingrese una opción válida entre 1 y 4, con 0 termina");
29             opcion = entrada.readLine().charAt(0);
30             while (opcion != '0') {
31                 switch (opcion) {
32                     case '1':
33                         int cantPac = 0;
34                         System.out.println("Ingrese la cantidad de pacientes (no mayor
35                         cantPac = Integer.valueOf(entrada.readLine());
36                         if (cantPac > 0 && cantPac <= cantPacientes) {
37                             int mayores78 = 0, anios = 0;
38                             while (cantPac > 0) {
39                                 System.out.println("Ingrese las edades (mayor a cero)");
40                                 anios = Integer.valueOf(entrada.readLine());
41                                 if (anios > mayorA)
42                                     mayores78++;
43                                 cantPac--;
44                             }
45                             System.out.println("La cantidad de pacientes mayores a 78 e
46                         } else {
47                             System.out.println("La cantidad de pacientes no corresponde
48                         }
49                     }
50                     break;
51                 case '2':
52                     int cantPrest = 0;
53                     System.out.println("Ingrese la cantidad de prestadores (no may
54                     cantPrest = Integer.valueOf(entrada.readLine());
```

Annotations:

- A red arrow points from the text "El estado de las variables es en el main según la pila de ejecución" to the "Thread [main] (Suspended)" entry in the Debug Console.
- A red arrow points from the text "Si tuviera más variables o constantes puedo ver sus estados" to the "Variables" view.