Procedimientos y Funciones en Java

Funciones y procedimientos vs métodos

Las funciones y procedimientos son conjuntos de código que se invocan directamente, que pueden recibir parámetros en la llamada y que pueden retornar un valor (funciones) o no (procedimientos).

Estas funciones y procedimientos, que también pueden llamarse *métodos de clase*, se diferencian con los métodos (o *métodos de instancia*) en que utilizan la palabra reservada ***static.***

Los métodos, también llamados *métodos de instancia* necesitan una instanciación (o creación) de un objeto para poder ser utilizados.

Un ejemplo de utilización de un método de instancia sería los que utilizamos en Scanner

```
Scanner sc= new Scanner(System.in); //Instanciación de objeto sc
sc.next();                          //Llamada al método next()
```

Funciones

Una función puede recibir o no parámetros y retorna un valor al punto del código que realizó la llamada a la misma.

Para definirlas se utiliza la estructura siguiente:

```
[Visibilidad] static tipoDevuelto nombreFuncion([tipo arg1, tipo arg2,...]){
        // cuerpo de la funcion
        // return finaliza la función devolviendo el valor indicado.
}
```

Los valores entre corchetes son opcionales y de no ponerse visibilidad, esta toma el valor ***Public***

Funciones

Por ahora definiremos las funciones como **static**, que es lo que las identifica como metodos estáticos (de clase) y no de instancia, ya que es la forma en la que podremos usarlo desde el método main.

Será así hasta que nos pongamos con la orientación a objetos.

Funciones

tipodevuelto : Es el tipo de dato que devuelve la función, puede ser un tipo primitivo (int, boolean, etc.) o bien una clase Java (por ejemplo, String) o una clase creada por nosotros (Alumno, Teléfono, etc.)
nombreFunción: Es el nombre que le damos a la función: esPrimo, calcularFactura, etc... Por convenio suele usar camelCase, aunque no es obligatorio.
tipo arg1: entre paréntesis figurarán unos identificadores precedidos de su tipo, que representan los parámetros que le pasamos a la función para que realice los cálculos necesarios.
En el cuerpo de la función irían todos los cálculos, incluyendo uno o varios return que finalizan la función en ese momento, devolviendo un valor acorde a lo definido en la cabecera de la función.

```java
public class EjemploFunciones {

    public static void main(String[] args) {

        int i=3;

        if(esPar(i))
            System.out.printf("El valor %d es par",i);
        else
            System.out.printf("El valor %d es impar",i);


    }//main

    static boolean esPar(int valor)
    {
        boolean resultado;
        if(valor%2==0)
            resultado = true;
        else
            resultado = false;
        return resultado;
    }//esPar

}//class
```

```java
boolean ret=esPar(i);//Podemos realizar un paso intermedio y guardar el valor de retorno

if(ret)//que luego usaremos
{
    System.out.printf("El valor %d es par",i);
}


ret = esPar(3);//Como argumento no tiene por qué ir una variable. Podemos pasarle valores 'estáticos'
```

Procedimientos

La diferencia con las funciones radica en que los procedimientos no retornan un valor tras ejecutarse. En java esto se indica indicando que el tipo devuelto es ***void*** (tipo de retorno inexistente).

La sintaxis de definición es la siguiente:

[visibilidad] static void nombreProcedimiento([tipo arg1, tipo arg2,...])

Los valores entre corchetes son opcionales y de no ponerse visibilidad, esta toma el valor ***Public***

```java
public class EjemploProcedimientos {

    public static void main(String[] args) {

        int valor=6;
        calcularCuadradoEImprimir(valor);

    }//main


    static void calcularCuadradoEImprimir(int i)
    {
        int resultado=i*i;
        System.out.printf("El cuadrado de %d es %d",i,resultado);
    }//calcularCuadradoEImprimir

}//clase
```

Visibilidad de procedimientos y funciones

La visibilidad indica lo accesible que es el procedimiento o función fuera de la clase.

El valor por defecto, al no indicarla, es ***public***

Los posibles valores que veremos en el curso son:

- ***Public***: Visible desde dentro y desde fuera de la clase
- ***Private***: Visible sólo desde dentro de la clase
- ***Protected***: Visible desde la clase y clases hijas (la veremos más adelante con herencia)

```java
public class EjemploVisibilidad {

    public static String ejemploPublic()
    {
        return "Es Public";
    }//ejemploVisible

    private static String ejemploPrivate()
    {
        return "Es Private";
    }//ejemploPrivate

    public static String ejemploAccesoIndirectoPrivate()
    {
        return ejemploPrivate();//Acceso a un método privado de la misma clase
    }//ejemploAccesoIndirectoPrivate

    public static String ejemploAccesoIndirectoPublic()
    {
        return ejemploPublic();//Acceso a un método público de la misma clase
    }//ejemploAccesoIndirectoPublic

}//class
```

```java
public class EjemploVisibilidadMain {

    public static void main(String[] args) {
        EjemploVisibilidad.ejemploPublic();//Acceso a un método público de otra clases
        EjemploVisibilidad.ejemploPrivate();//No se puede acceder a un método privado de otra clase

    }//main

}//class
```

Variables en funciones (y procedimientos)

Dentro de las funciones se declara un ámbito (definido por sus { }) con lo que cualquier variable definida en una función sólo tendrá vigencia en esa función.

Si necesitamos una variable común a todas las funciones, una variable que se pueda usar a lo largo de todo el programa, debemos definirla después del nombre del programa. A este tipo de variables, visibles desde todo el programa, se le denominan **variables globales**.

Estas variables globales deben llevar el prefijo **static** antes de su tipo.

```java
import java.util.Scanner;
public class Ejemplo {
    static Scanner teclado;

    static void main(String[] args) {
        teclado = new Scanner (System.in);
        String nombre = pedirNombre();
        int edad = pedirEdad();
        System.out.printf("Hola %s. Tienes %d años%n", nombre, edad);
    } //fin main

    static String pedirNombre() {
        System.out.println("Introduce tu nombre:");
        return teclado.nextLine();
    }
    static int pedirEdad() {
        System.out.println("Introduce tu edad:");
        return teclado.nextInt();
   }
}
```

Paso de parámetros

En general, en los lenguajes de programación, se suele decir que hay dos posibilidades en cuanto a al paso de parámetros: por valor o por referencia.

Paso de parámetros **por valor** significa a que se pasa al método o función una copia del valor con el que se ha llamado al método, pero la variable “llamante” no es modificada, después de la ejecución del método, la variable sigue valiendo lo mismo.

Paso de parámetros **por referencia** significa que se lse pasa al método o función la referencia de memoria de la variable “llamante” de forma que además de tener el valor de esa variable, podemos modificar su valor en el método.
En Java, los parámetros se pasan por valor

Paso de parámetros

Lo comprobamos viendo la ejecución de este código:

```
public class Main {
    public static void main(String[] args) {
        int a = 10;
        doble (a);
        System.out.println(a);
    }
    static void doble (int x) { x = x *2;}
}
```

El programa mostaría “10”. Si los parámetros se pasasen por referencia, mostaría 20.

Más adelante veremos que **en el caso de los objetos** el comportamiento es ligeramente **diferente**.

Uso del depurador

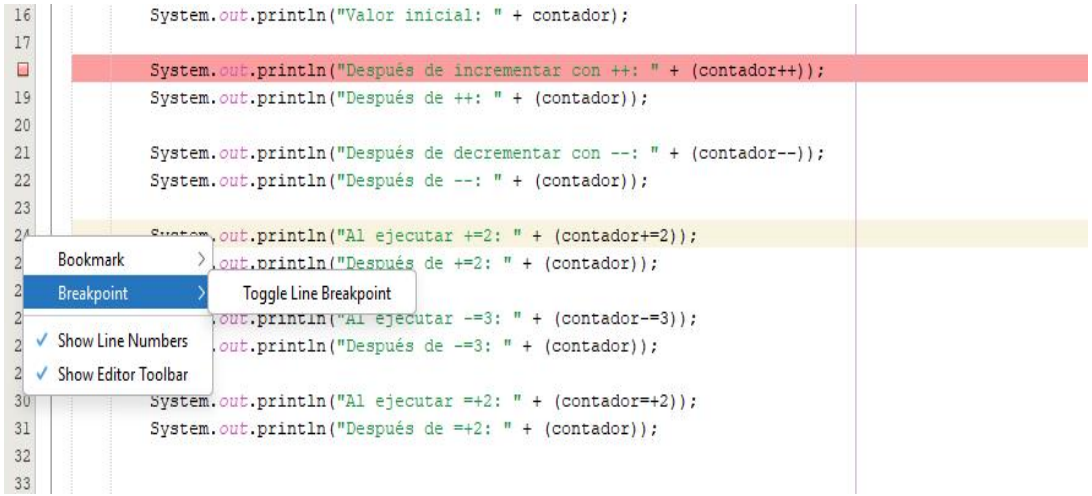
El depurador en los IDE es una herramienta muy útil a la hora de encontrar errores en el código.

Al contrario que la ejecución normal, mediante el depurador podemos interrumpir la ejecución del código y ejecutar paso por paso las líneas de código que queramos.

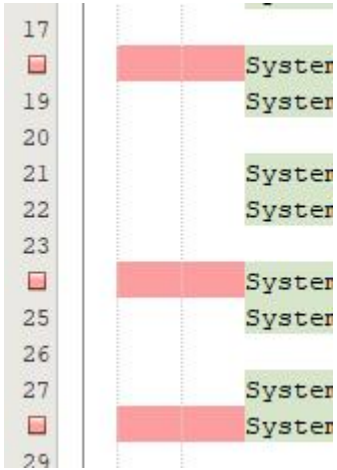
Uso del depurador en NetBeans

En las siguientes diapositivas se muestra el uso del depurador en NetBeans

Indicamos puntos de ruptura o ***breakpoints*** en la barra gris de código enumerado mediante doble click, pulsando “Toogle Line Breakpoint” en el menú del botón derecho o pulsando la combinación de letras Ctrl+F8 en la línea de código a interrumpir

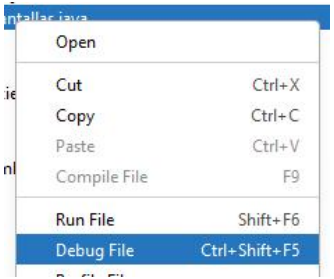


Podemos añadir tantos breakpoints como queramos y estos se nos marcarán en la barra de la izquierda y la línea con la interrupción en rojo.

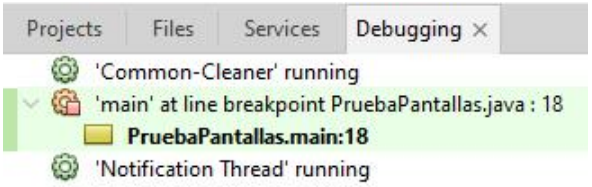


Para ejecutar en modo debug podemos usar el siguiente botón

O mediante el botón derecho encima del fichero a depurar:



Si el código tiene puntos de interrupción, al llegar a uno, se parará la ejecución del código y se nos mostrará la pantalla de Debugging

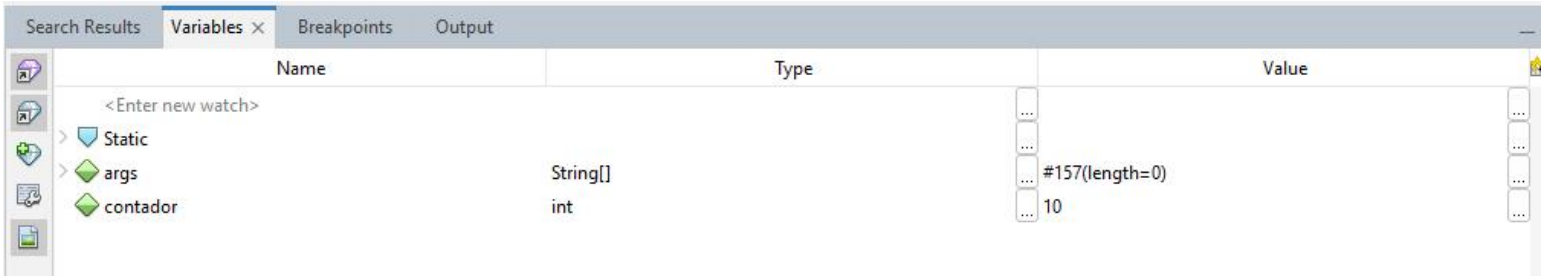


Para la ejecución paso por paso podemos utilizar tanto los botones de flechas del menú superior

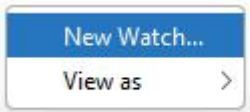
como las teclas equivalentes del teclado:

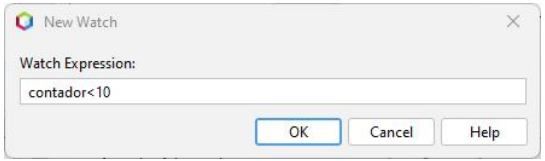
- Step Into: Tecla F7 → Si se pulsa sobre una línea de código con una llamada a función o procedimiento se entra en el código interno.
- Step Over: Tecla F8 → Ejecuta la siguiente instrucción sin entrar en procedimientos y funciones.
- Step Return: Ctrl + F7→ Se puede pulsar dentro de procedimientos y funciones y sale de ellos.
- Resume: Tecla F5 → Se continúa la ejecución de forma normal o hasta el próximo breakpoint

Ventana de variables. Muestra los valores de las variables definidas en el punto de ejecución actual además de su valor.

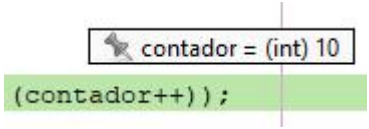


Podemos añadir expresiones a esta ventana para evaluarlas sin necesidad de parar el código completamente:

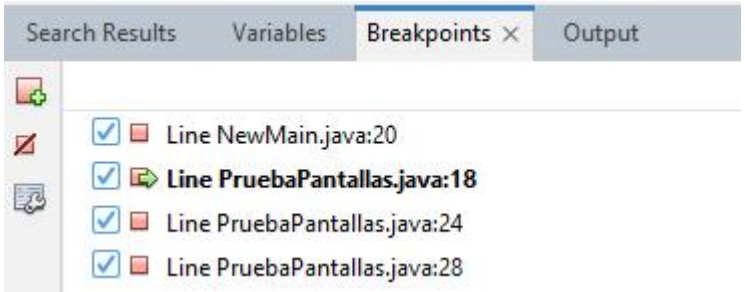




También se puede evaluar una variable diréctamente en el código situando el cursor encima



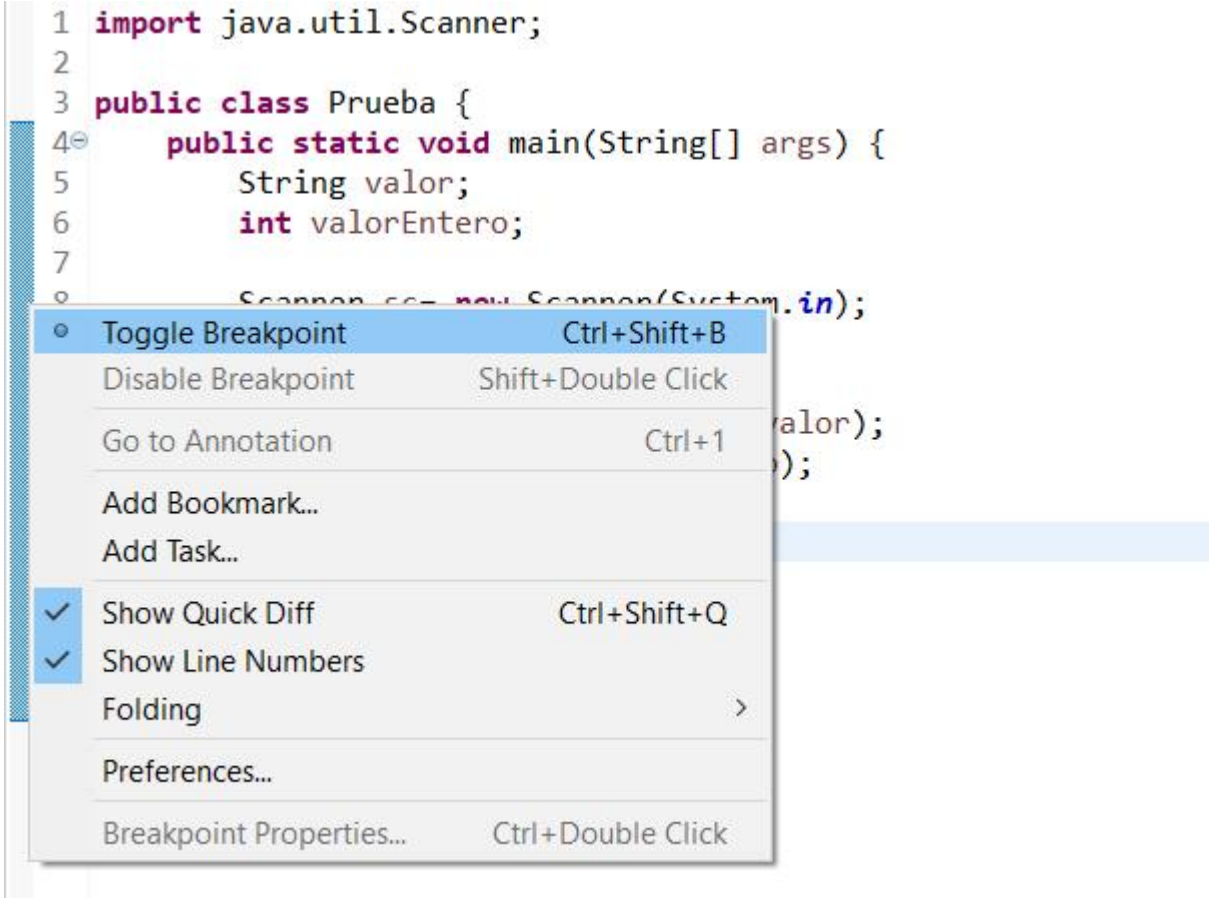
Ventana de breakpoints. Muestra los breakpoints del programa. En esta pantalla se pueden deshabilitar para que no interrumpan la ejecución del programa.



Uso del depurador en Eclipse

En las siguientes diapositivas se muestra el uso del depurador en Eclipse

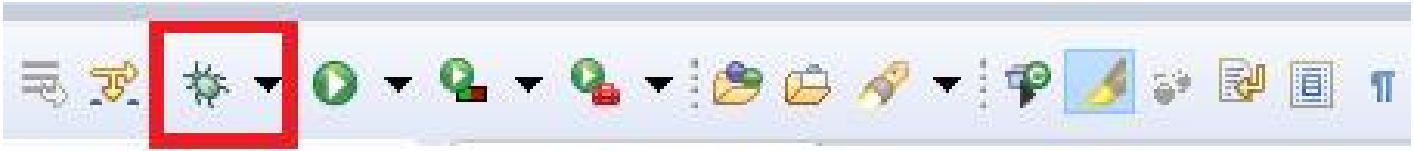
Indicamos puntos de ruptura o ***breakpoints*** en la barra azul mediante doble click, pulsando “Toogle Breakpoint” en el menú del botón derecho o pulsando la combinación de letras Ctrl+Shift+B en la línea de código a interrumpir



Podemos añadir tantos breakpoints como queramos y estos se nos marcarán en la barra azul de la izquierda

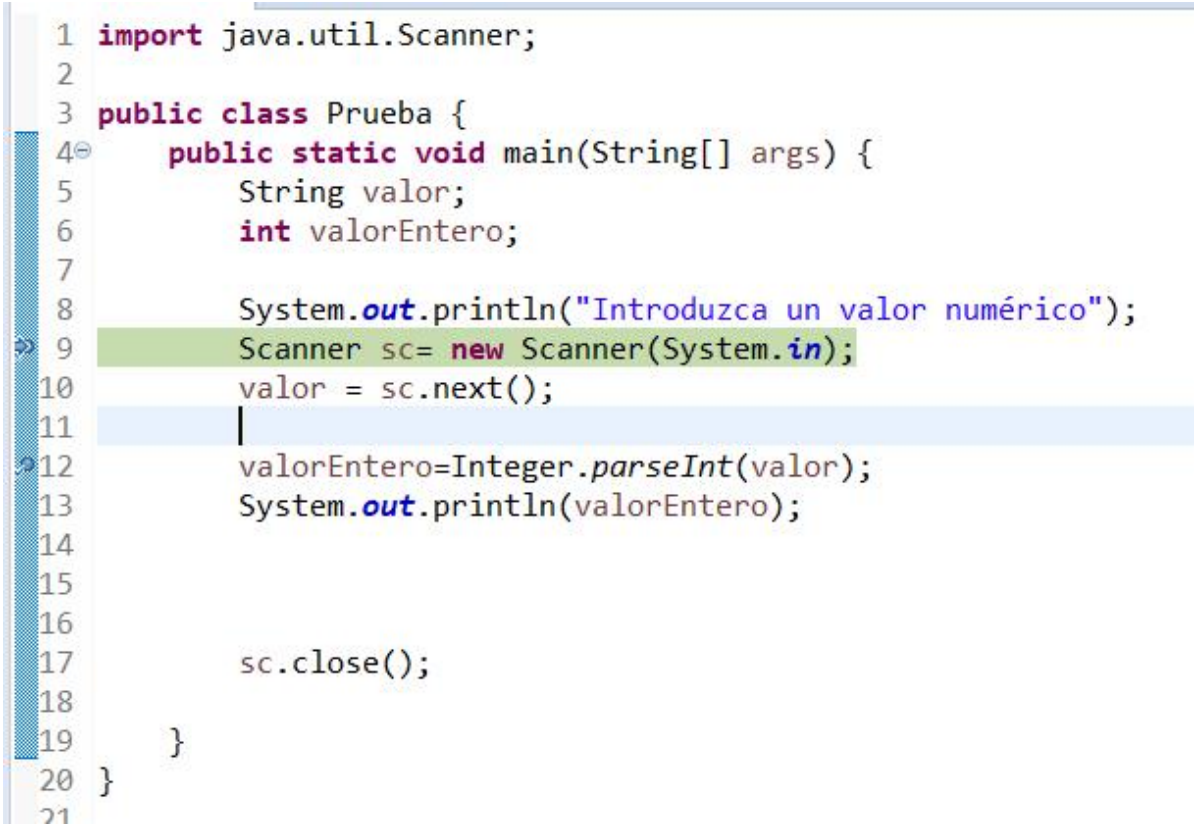
```java
import java.util.Scanner;

public class Prueba {
    public static void main(String[] args) {
        String valor;
        int valorEntero;

        Scanner sc= new Scanner(System.in);
        valor = sc.next();

        valorEntero=Integer.parseInt(valor);
        System.out.println(valorEntero);



        sc.close();

    }
}

```

Para ejecutar en modo debug utilizaremos el siguiente botón



Si el código tiene puntos de interrupción, al llegar a uno, Eclipse nos preguntará si queremos abrir la perspectiva de debug, que contiene ventanas específicas para la depuración de código

El punto que se está ejecutando actualmente en el programa se muestra con un color verde (si tenemos el tema por defecto de Eclipse)



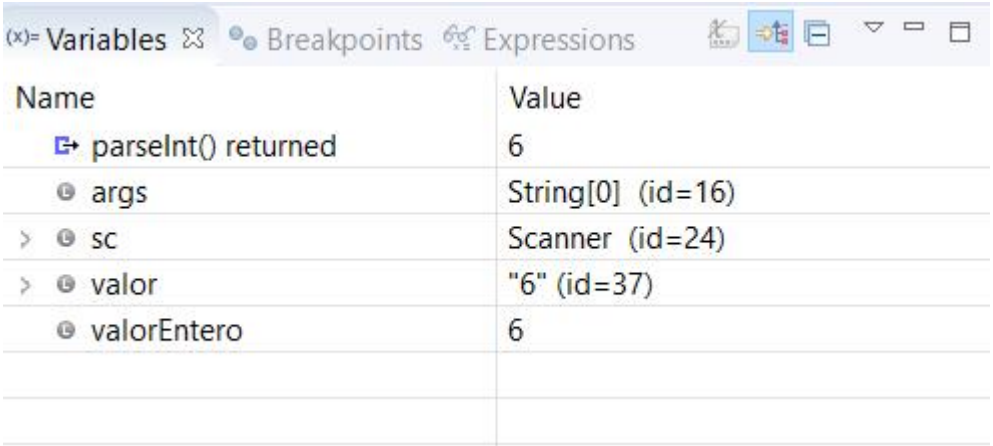
```
1  import java.util.Scanner;
2
3  public class Prueba {
4      public static void main(String[] args) {
5          String valor;
6          int valorEntero;
7
8          System.out.println("Introduzca un valor numérico");
9          Scanner sc= new Scanner(System.in);
10         valor = sc.next();
11
12         valorEntero=Integer.parseInt(valor);
13         System.out.println(valorEntero);
14
15
16
17         sc.close();
18
19     }
20 }
21
```

Para la ejecución paso por paso podemos utilizar tanto los botones de flechas del menú superior

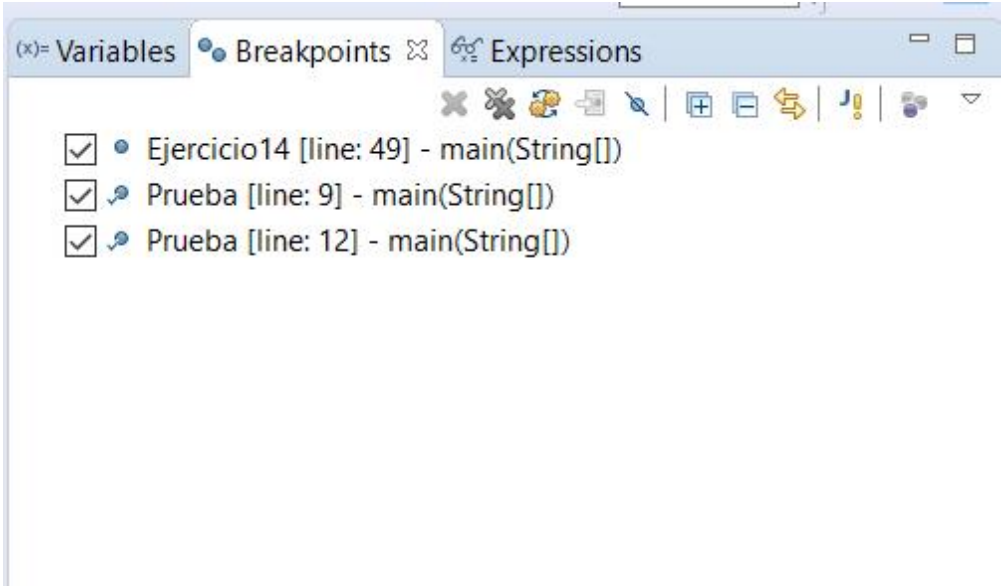
como las teclas equivalentes del teclado:

- Step Into: Tecla F5 → Si se pulsa sobre una línea de código con una llamada a función o procedimiento se entra en el código interno.
- Step Over: Tecla F6 → Ejecuta la siguiente instrucción sin entrar en procedimientos y funciones.
- Step Return: Tecla F7→ Se puede pulsar dentro de procedimientos y funciones y sale de ellos.
- Resume: Tecla F8 → Se continúa la ejecución de forma normal o hasta el próximo breakpoint

Ventana de variables. Muestra los valores de las variables definidas en el punto de ejecución actual además de su valor.



Ventana de breakpoints. Muestra los breakpoints del programa. En esta pantalla se pueden deshabilitar para que no interrumpan la ejecución del programa.



Ventana de expresiones. En esta ventana se pueden introducir expresiones de código para ver su valor en el punto de ejecución actual