

# Depuración de código

## 1.1 Introducción

Nesta parte da unidade didáctica que nos ocupa preténdense os seguintes obxectivos:

- Utilizar o contorno de desenvolvemento libre para depurar código e utilización de puntos de ruptura.
- Examinar e modificar o comportamento dun programa en tempo de execución utilizando o contorno de desenvolvemento libre.

## 1.2 A depuración de código

A operación de depuración serve para examinar o código da aplicación en tempo de execución e buscar solucións a erros detectados. Permite executar liñas de código ata un momento no que se pode examinar o estado das súas variables para descubrir problemas.

Cando un programa ten certa complexidade, a depuración é imprescindible para detectar posibles erros.

Os exemplos de depuración desta actividade utilizarán a clase **Factorial**. A clase factorial é un programa que calcula e imprime o factorial de  $n$  ( $n!=1*2*3*...*n$ ). O programa, non obstante, ten un erro lóxico e da unha resposta incorrecta para  $n=20$  ("O factorial de 20 é - 2102132736") posto que o resultado é un número negativo.

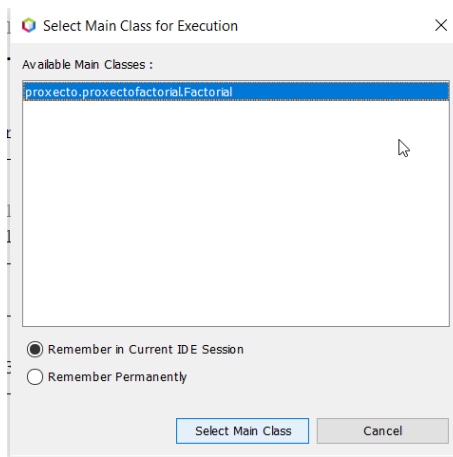
O programa para calcular o factorial ten o seguinte código:

```
package proxecto.proxecto factorial;
/**
 * Calcula o factorial de n
 */
public class Factorial {
    public static void main(String[] args) {
        int n=20;
        int factorial=1;

        //n!=1*2*3*...*n
        for (int i=1;i<=n;i++) {
            factorial*=i;
        }
        System.out.println("O factorial de "+n+" é "+factorial);
    }
}
```

### Sesión de depuración

Para iniciar unha sesión de depuración é indispensable que o arquivo ou proxecto a depurar teña método **main**, é dicir, ten que ser posible executalo. Ao tentar depurar o proxecto, aparece unha ventá que pide que se elixa a clase Main para poder executalo.

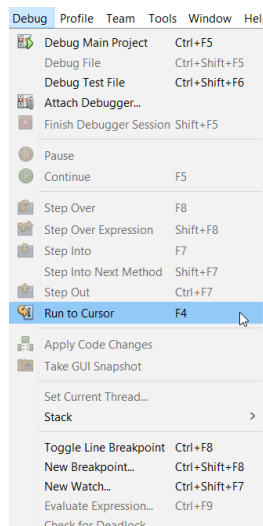


NetBeans permite iniciar sesión de depuración dun proxecto ou dun arquivo e elixir como empezar a depuración de diferentes maneiras. Por exemplo permite depurar:


- Un proxecto establecido como proxecto principal, premendo Ctrl-F5 ou indo ao menú principal e elixindo *Debug->Debug Project*. Para establecer un proxecto como principal imos no menú principal a *Run* e eliximos *Set Main Project*.
- Un proxecto calquera dende a ventá *Projects*, seleccionando o proxecto, facendo clic dereito, e elixindo *Debug*.
- Un arquivo fonte que se estea editando nese momento, indo ao menú principal e elixindo *Debug->Debug File*.
- Un arquivo calquera dende a ventá *Projects*, seleccionando o arquivo, facendo clic dereito, e elixindo *Debug File*.

En todos os casos anteriores, a sesión de depuración comeza no arquivo seleccionado ou na clase principal do proxecto seleccionado e sigue a execución ata que finalice normalmente o programa, encontre algún erro ou algún **punto de interrupción**.

No menú **Debug**, pódese ver que a sesión de depuración tamén pode iniciarse para que a **execución** se faga ata a liña na que estea o **cursor** (*Run to Cursor*).

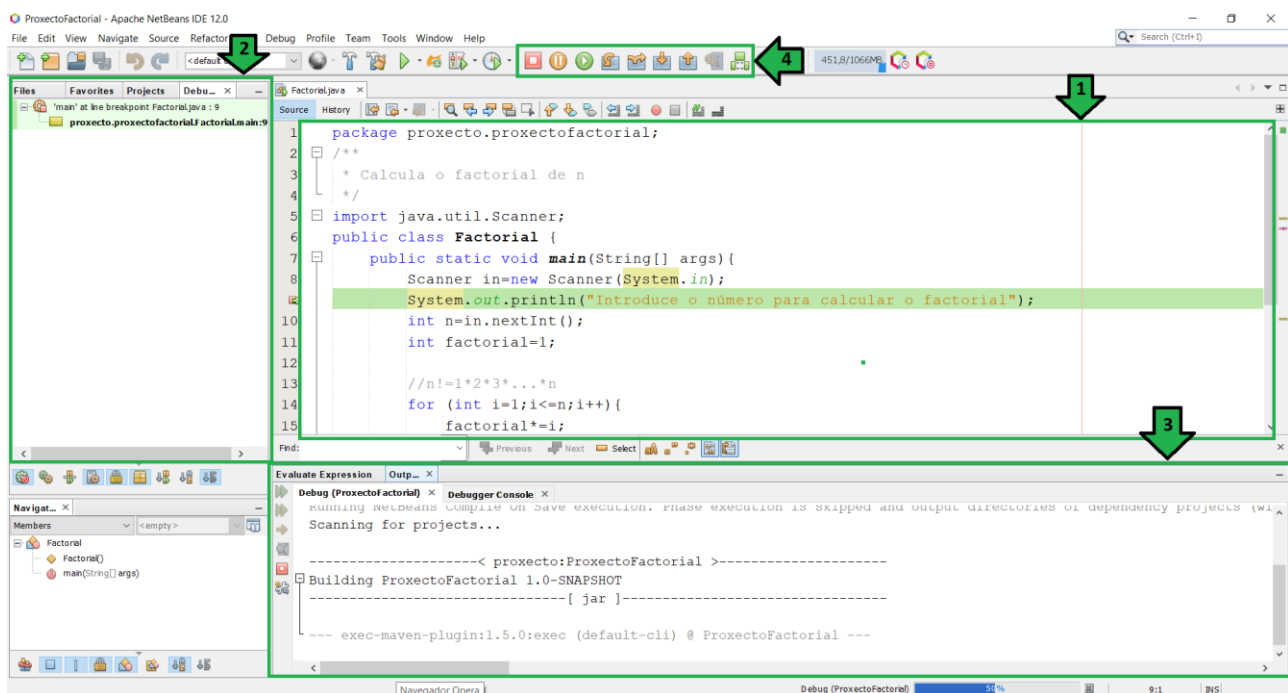


## Execución ata o cursor

A opción *Run to Cursor* permite executar o programa ata a localización do cursor no arquivo que se está editando e pausa o programa ata que se lle indica a seguinte operación a realizar na depuración. O arquivo editado debe ser chamado dende a clase principal do proxecto principal. Para realizar esta depuración débese situar o cursor no código fonte, premer F4 ou seleccionar no menú principal *Debug > Run to Cursor* ou seleccionar  na barra de ferramentas da depuración unha vez se indicou que se quería depurar o proxecto.

## Pantalla de depuración

A pantalla de depuración aparece despois de iniciada a depuración e ten varias zonas. Por exemplo, despois de iniciada unha sesión de depuración paso a paso aparecen as zonas que se ven na imaxe seguinte.



**Zona 1:** Zona co código fonte en depuración. A seguinte liña a executar no proceso de depuración aparece marcada con cor de fondo verde e unha frecha verde na marxe esquerda.

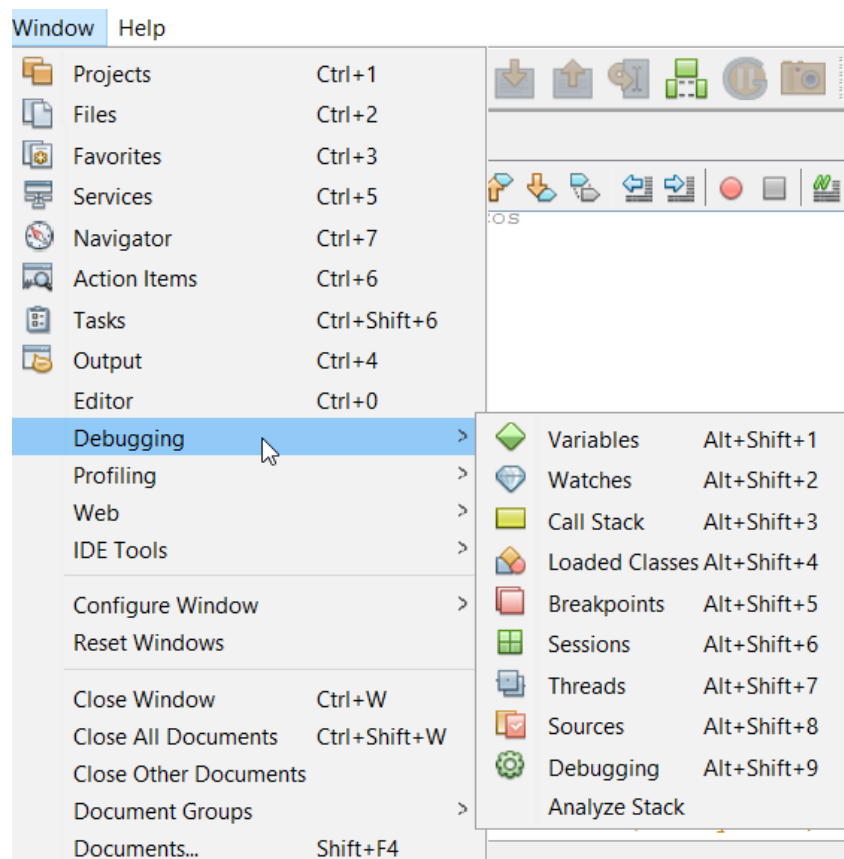
**Zona 2:** Ventá **Debugging** con información sobre os procesos que se están executando. Na parte inferior desa ventá aparece un menú de iconas para poder cambiar a vista da información:



### Zona 3: Zona con diversas ventás:

- Ventás de saída ou **output** que se subdivide en:
  - **Debugger Console** (Consola do depurador) con información sobre o proceso de depuración.
  - **Debug (ProxectoFactorial)**. Se o proxecto en concreto ten entradas e saídas dende a consola realizaranse aquí as entradas de datos e verase a saída de resultados.
- Ventá **Variables** na que se pode ver e cambiar información sobre as variables locais e expresións.
- Ventá Puntos de interrupción ou **breakpoints** na que se pode ver e cambiar información sobre os puntos de interrupción.

**Zona 4:** Barra de ferramentas de depuración para indicar o seguinte paso a realizar na depuración. Algunha das mesmas aparecen tamén activadas no menú principal Debug. Se non se ve algunha das ventás anteriores, pode accederse á opción *Window* → Debugging do menú principal e elixir a ventá que se desexa ver.



Durante o proceso de depuración pode ocorrer que a execución dunha liña de código precise dunha entrada por teclado e entón:

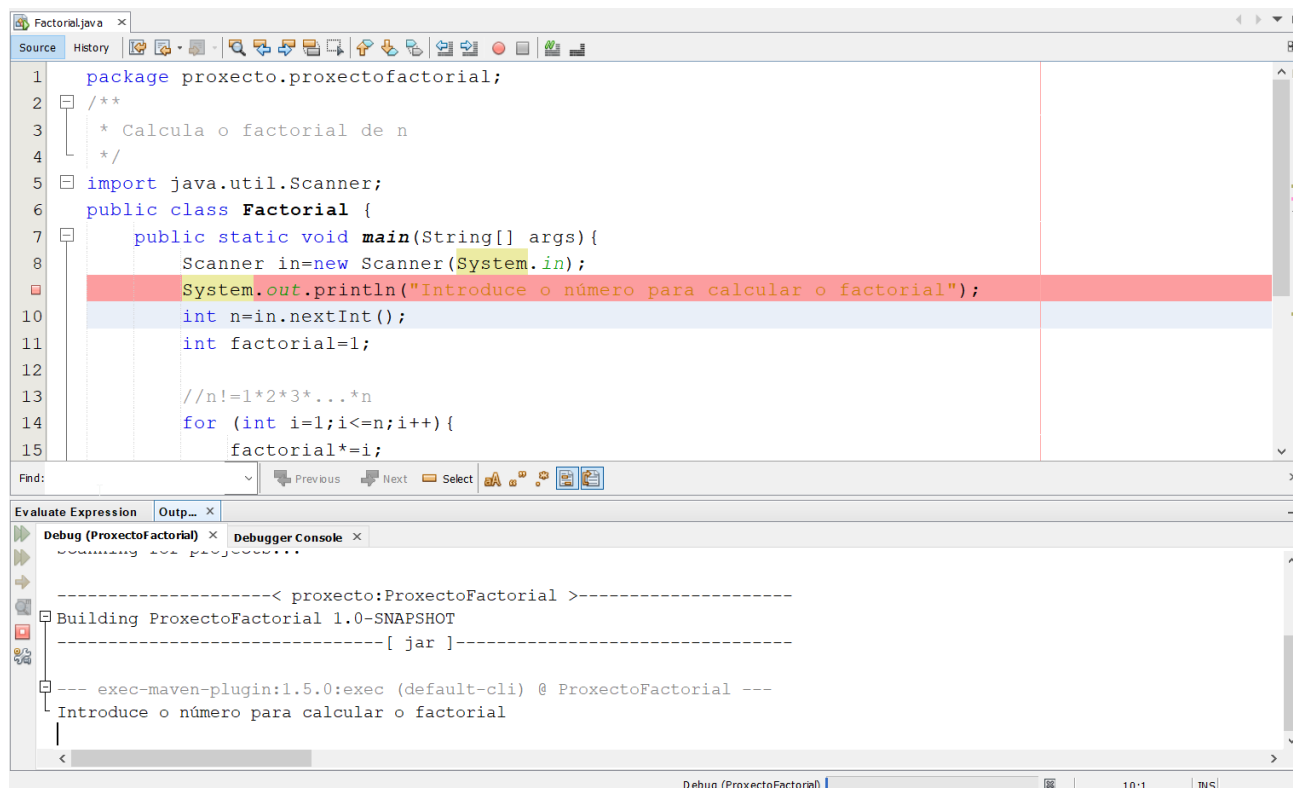
A liña de código que ten a entrada pasa de ter fondo verde a ter fondo azul na ventá de edición.

Para ver o que ocorre ao ter que introducir datos por teclado, cambiamos o código que estamos depurando polo seguinte:











```
package proxecto.proxecto factorial;
/**
 * Calcula o factorial de n
 */
import java.util.Scanner;
public class Factorial {
    public static void main(String[] args){
        Scanner in=new Scanner(System.in);
        System.out.println("Introduce o número para calcular o factorial");
        int n=in.nextInt();
        int factorial=1;

        //n!=1*2*3*...*n
        for (int i=1;i<=n;i++){
            factorial*=i;
        }
        System.out.println("O factorial de "+n+" é "+factorial);
    }
}
```

O proceso de depuración queda detido ata que se teclee o dato na ventá de saída Debug (proxectoFactorial).



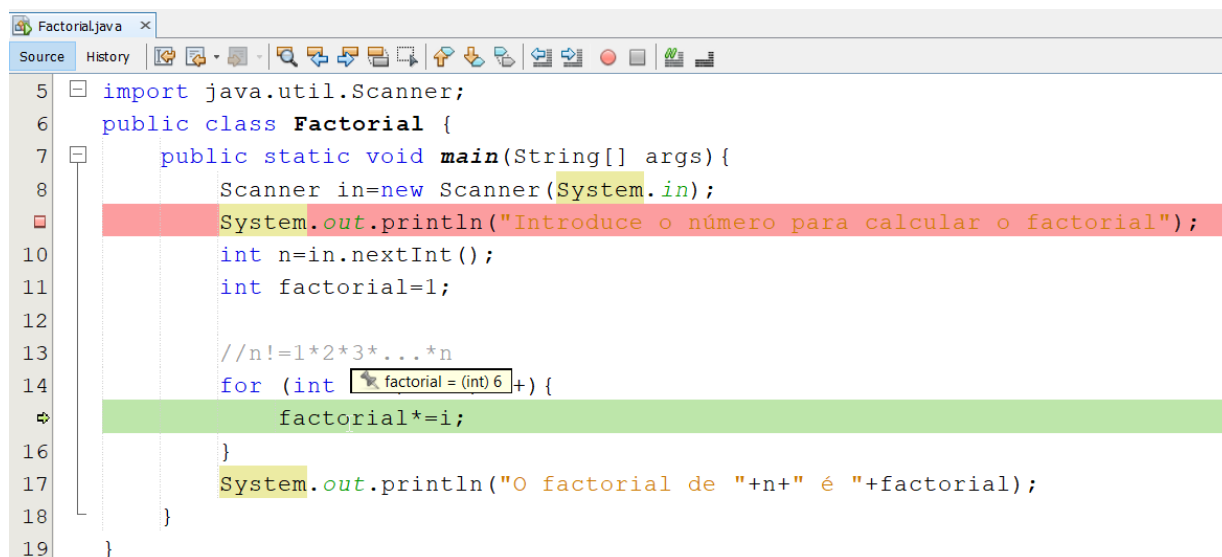
Explicación rápida do significado de cada icona da barra de ferramentas de depuración (zona 4):

| Icona   | Descrición                                    | Detalle  |
|---|---|--|
|    | Finalizar sesión do depurador (Maiúsculas+F5) | Finalizar instantaneamente a depuración.   |
|    | Pausa   | Facer unha pausa no proceso de depuración.   |
|    | Continuar (F5)                                | Continuar a execución normal do programa despois de facer unha pausa.  |
|    | Continuar execución (F8)                      | Executa unha instrución. Se contén unha chamada a un método, este executase completo sen parar en cada unha das instrucións.   |
|    | Continuar sobre a execución (Maiúsculas+F8)   | Continuar sobre a execución.<br>É un refinamento de F8 para expresións con chamadas a métodos. Cada vez que nunha sesión de depuración paso a paso se utiliza este comando sobre unha expresión con chamada a métodos, párase a depuración antes de executar a chamada ao método actual podendo ver o historial dos valores de retorno dos métodos inmediatamente previos e o valor dos argumentos do método actual na ventá de Variables locais. Pode ser útil cando os valores de retorno dun método non se gardan nunha variable e por tanto non poden ser inspeccionados en tempo de depuración. |
|  | Paso a paso (F7)                              | Executar Paso a paso.<br>Permite entrar na execución paso a paso do método da liña actual. De haber máis dun método na liña, poderase: escoller o método que se vai depurar paso a paso utilizando as teclas de movemento do cursor ou a tecla tab e confirmando con F7 ou executar normalmente (F7)   |
|  | Executar e saír (Ctrl+F7)                     | No caso de estar depurando dentro dun método, Ctrl+F7 finaliza a sesión de depuración do método e volve ao método que chamou ao actual.<br>No caso de utilizarse no método principal, finaliza a sesión de depuración.   |
|  | Executar ata o cursor F4                      | Executar ata o cursor e espera instrucións para continuar coa depuración.  |
|  | Aplicar cambios no código                     | Aplicar cambios no código.   |
|  |   | Premer para activar a recollida de lixo.   |

## Inspección e modificación de variables, expresións e métodos

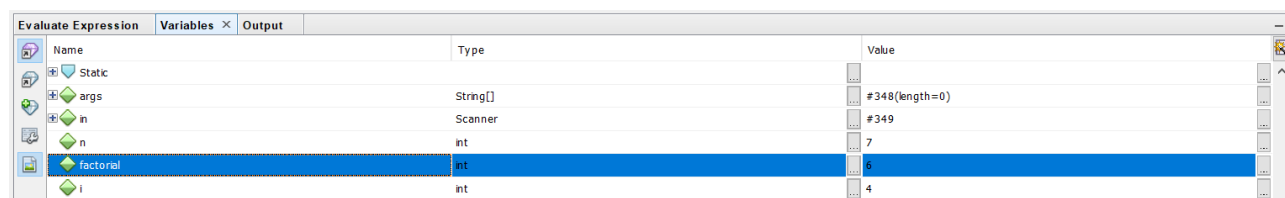
### No código fonte


Durante unha sesión de depuración pódense ver o tipo e valor dunha variable situando o cursor sobre ela no código fonte:

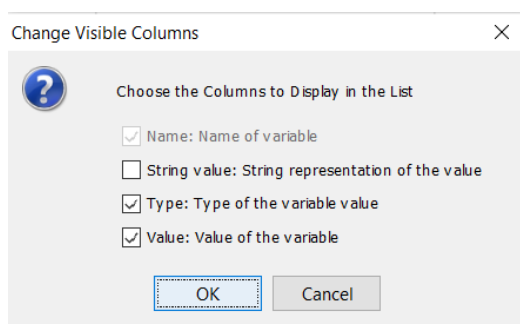


### Na ventá Variables


Durante unha sesión de depuración pódese ver información sobre as variables locais na ventá **Variables** para variables locais e atributos de clase se existen.



A icona situada á dereita  permite modificar a información que se ve.

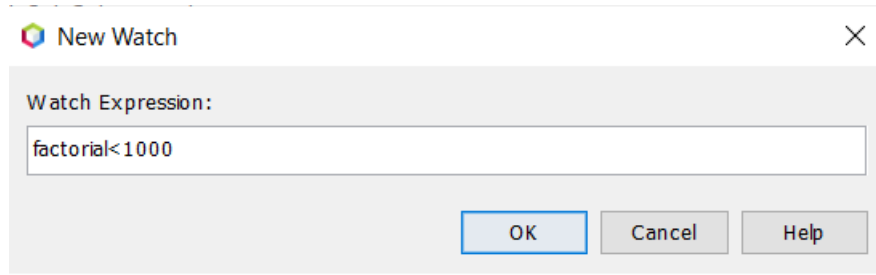


Ademais das variables locais, pódense engadir expresións para observar (*watches*). Isto pódese facer de varias maneiras:

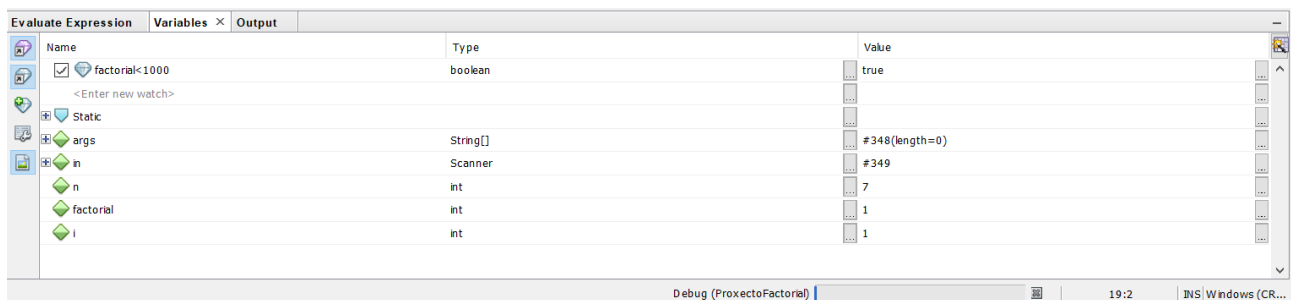
- Seleccionar a expresión no arquivo fonte editado, premer clic dereito e elixir New Watch (*Novo elemento observado*) ou premer Ctrl+Maiúsculas+F7.
- Seleccionar no menú principal *Debug* → *New Watch*.
- Premer na icona  da ventá *Variables*.

- Teclear o novo elemento na liña da ventá *Variables* que pon *<Enter new watch>*.

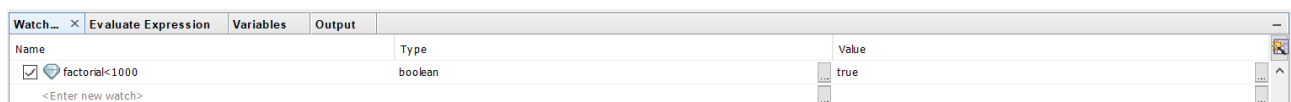
En calquera dos tres primeiros casos, aparece unha ventá na que ten que quedar definida a expresión que se quere observar.



A expresión aparece engadida na ventá *Variables* como un *watch*.



Os elementos observados poden moverse entre a ventá *Variables* e a ventá *Watches View (Elementos observados)* utilizando o botón

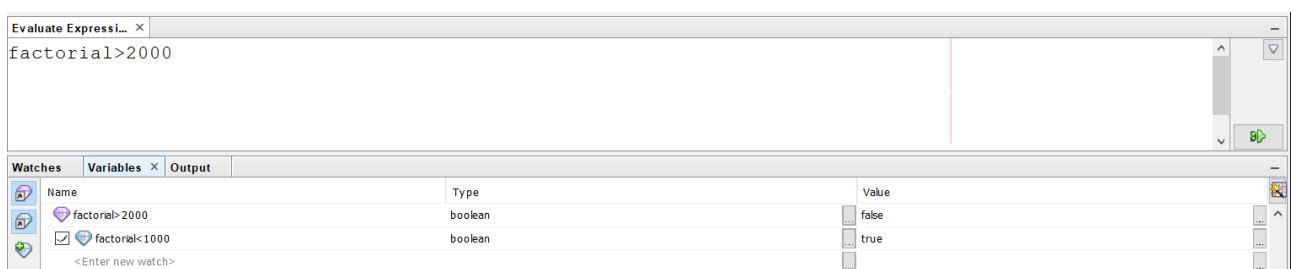


Na ventá *Variables* tamén se pode facer clic dereito sobre o nome dunha variable ou expresión e facer cambios como por exemplo: eliminar desa ventá unha variable ou expresión, eliminar todas, ver o valor noutro formato ou editar.

[Co menú Depuración → Avaliar expresión](#)


Durante unha sesión de depuración pódese ver o resultado dunha expresión dende o menú principal elixindo *Debug → Evaluate expression* (Avaliar expresión) ou premendo **Ctrl+F9**; ábrese a ventá *Evaluate Expressions or code snippets* na que se pode teclear a expresión e avaliar o resultado nese momento premendo sobre o botón

Se a expresión non é posible no contexto actual non se podería ver o resultado. O resultado será similar ao seguinte:





### Coa opción de depuración *Continuar sobre a execución*

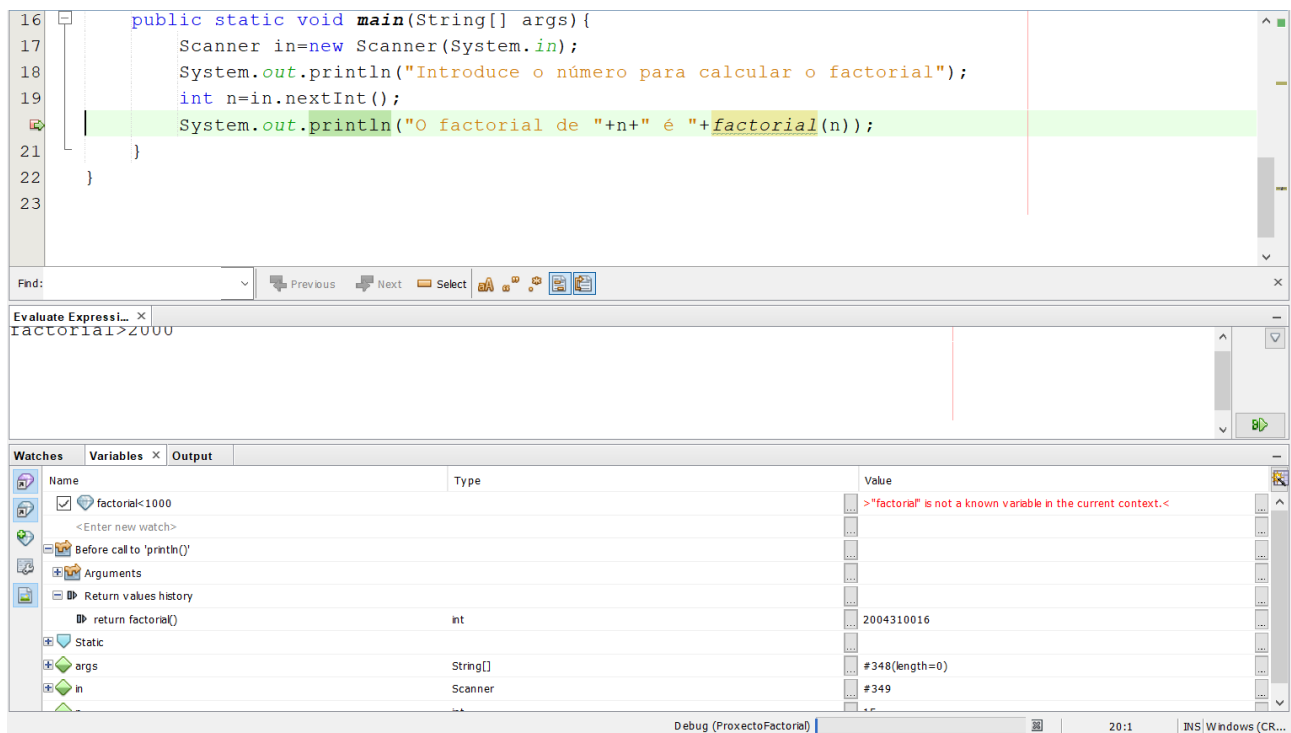
Pódense ver os valores de retorno dos métodos previos (*Return values history*) e o valor dos parámetros do método seguinte (*Before to call to...*) , na ventá *Variables* cando nunha sesión de depuración se utiliza Maiúsculas+F8 (*Continuar sobre a execución*) ou se preme na icona  da barra de depuración sobre unha expresión con chamada a un método.

Para ver este funcionamento, imos modificar o código para que teña un método:

```
package proxecto.proxecto factorial;
/**
 * Calcula o factorial de n
 */
import java.util.Scanner;
public class Factorial {
    public static int factorial(int num){
        int factorial=1;

        //n!=1*2*3*...*n
        for (int i=1;i<=num;i++){
            factorial*=i;
        }
        return factorial;
    }
    public static void main(String[] args){
        Scanner in=new Scanner(System.in);
        System.out.println("Introduce o número para calcular o factorial");
        int n=in.nextInt();
        System.out.println("O factorial de "+n+" é "+factorial(n));
    }
}
```


Podemos observar o que se visualiza ao chamar a `println()`:



The screenshot shows an IDE with a code editor and a debug console. The code editor displays the `main` method of the `Factorial` class. The line `System.out.println("O factorial de "+n+" é "+factorial(n));` is highlighted in green. The debug console shows the output of the program: `Factorial>2004310016`. Below the console, the **Watches** tab is active, showing a table of variables and their values.

| Name                       | Type     | Value   |
|----------------------------|----------|---|
| factorial<1000             |          | >"factorial" is not a known variable in the current context.< |
| <Enter new watch>          |          | ...   |
| Before call to 'println()' |          | ...   |
| Arguments                  |          | ...   |
| Return values history      |          | ...   |
| return factorial()         | int      | 2004310016  |
| Static                     |          | ...   |
| args                       | String[] | #348(length=0)  |
| in                         | Scanner  | #349  |

## Modificación

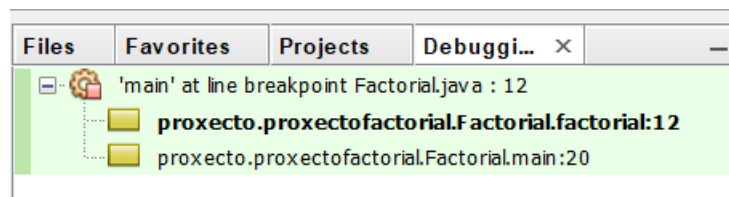
Na ventá de avaliar expresións, na de variables e na de elementos observados pódese modificar o valor dunha variable e continuar o proceso de depuración con ese valor. Para iso faise clic ao carón do valor actual da variable, ou utilízase a icona  cando exista, para teclear o novo valor, prémese en *Aceptar* e continúaase coa depuración.

## Pila (call stack)

A utilización da pila de chamadas é especialmente útil cando se utilizan varios fíos ou subprocesos durante a execución. Cando se inicia unha sesión de depuración, ábrese automaticamente a ventá *Debugging* con información sobre os subprocesos existentes e a pila de chamadas de cada un dos subprocesos suspendidos ou pausados; só unha delas chamadas é a chamada actual.

## Ventá *Debugging*









A ventá *Debugging* ten o seguinte aspecto:












Esta imaxe indica unha sesión de depuración na que a execución de `main()` queda interrompida na liña 20 por unha chamada ao método `factorial()`. O método `factorial()` é o método actual e está pausado na liña 12.

A última chamada realizada é a chamada que se considera actual e indícase na pila en letra grosa. De consultar as variables locais veríanse as da chamada actual. Se os arquivos fontes están dispoñibles, pódese facer clic dereito na chamada e elixir *Go to source* para ver o código fonte da chamada.

A carón de cada proceso aparece unha icona con información sobre o proceso:

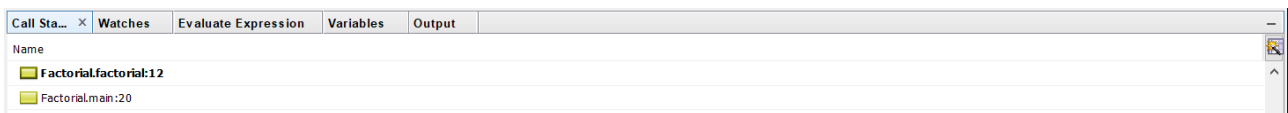
| Icons   | Description   |
|---|---|
|  | Indicates a thread that is running                          |
|  | Indicates a thread suspended by hitting a breakpoint        |
|  | Indicates a suspended thread                                |
|  | Indicates a thread group where all threads are running      |
|  | Indicates a thread group where all threads are suspended    |
|  | Indicates a thread group with running and suspended threads |
|  | Indicates a call stack frame                                |
|  | Indicates a call stack frame group                          |

Pódese cambiar a vista desta información utilizando o menú de iconas da parte inferior:



| Button  | Description  |
|---|--|
|  | Show thread groups   |
|  | Shows or hides the controls for suspending and resuming threads. |
|  | Show system threads  |
|  | Show suspended and current threads only                          |
|  | Show monitors  |
|  | Show qualified names   |
|  | Sort by suspended/resumed state                                  |
|  | Sort by name   |
|  | Sort by default  |

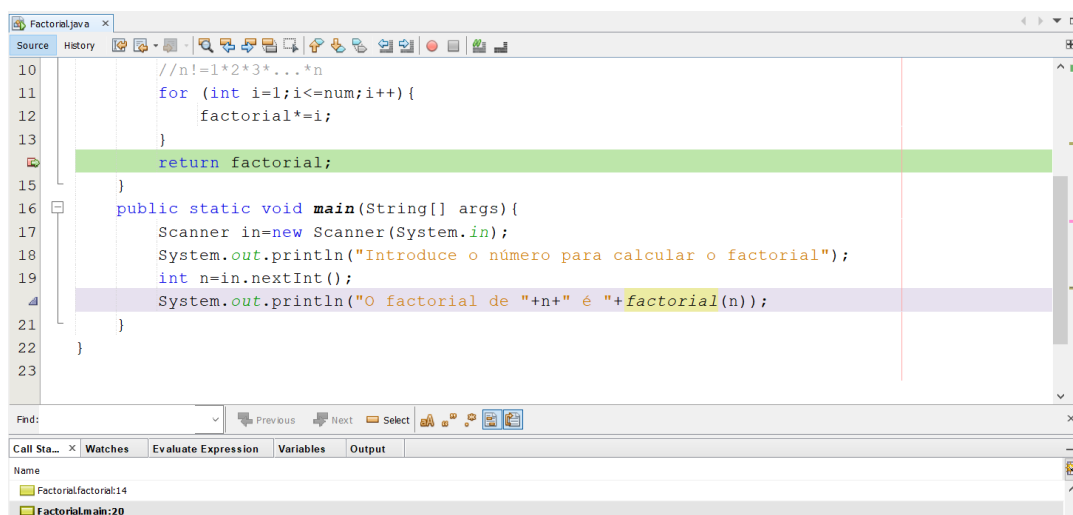
### Ver a pila

A información sobre a pila de chamadas tamén se pode ver indo ao menú principal e elixindo *Window* → *Debugging* → *Call Stack* ou premer **Alt+Maiúsculas+3** e abrírase a ventá *Call Stack View* na zona de ventás. A información para cada chamada está marcada por unha icona e a descrición da chamada.



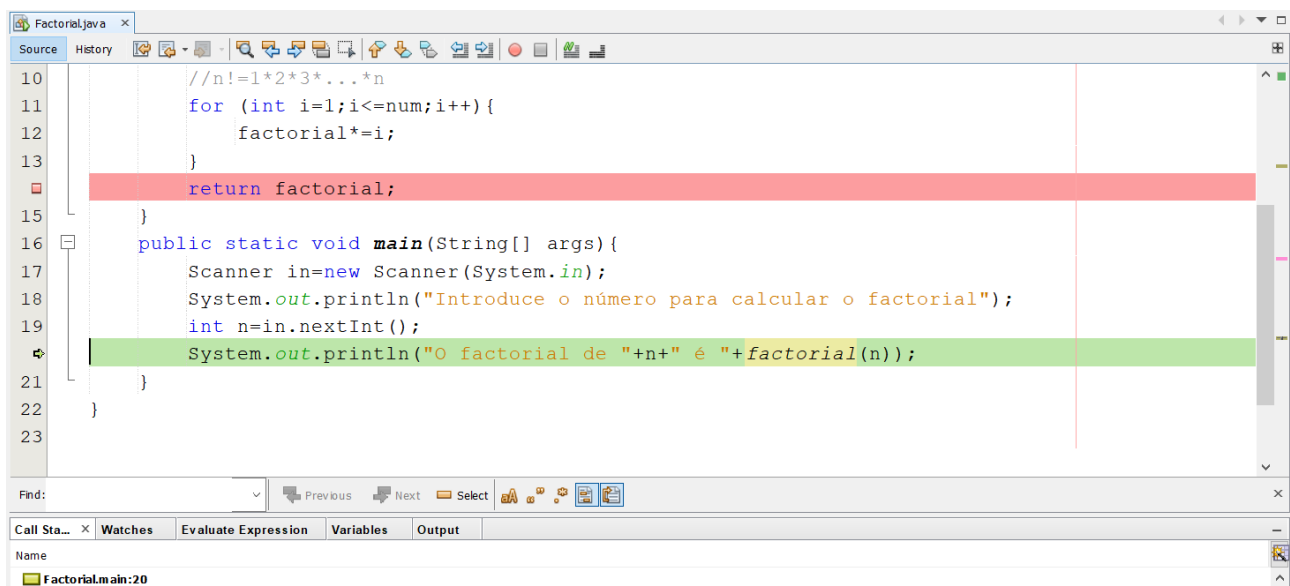
### Cambios e movementos na pila

Utilizando *Debug* → *Stack* do menú principal, pódese elixir *Make Caller Current* (Asignar actual ao emisor da chamada) ou *Make Callee Current* (Asignar actual ao receptor da chamada) para moverse pola pila poñendo como chamada actual a chamada anterior á chamada actual ou a seguinte. Nese caso, na ventá de edición visualízase a liña de código corresponde a esa chamada marcada como , e na ventá de variables, veranse as variables correspondentes a esa chamada, pero non se cambia a cabeceira da pila, nin varían as chamadas da pila nin o marcador de depuración .



Para designar como actual calquera chamada da pila, débese seleccionar a chamada na pila, facer clic dereito e elixir *Make Current*. Visualízase na ventá de edición a liña de código que corresponde a esa chamada e na ventá de variables as variables que se poidan ver nesa chamada, pero non cambia a cabeceira da pila, as chamadas da pila nin o marcador de depuración.

Utilizando dende o menú principal *Debug* → *Stack* → *Pop Topmost Call*, elimínase a chamada actual na cabeceira da pila, a seguinte chamada da pila pasa á cabeceira e por tanto a ser a chamada actual e o marcador da depuración no código fonte móvese á instrución que chamou á chamada á borrada. De seguir coa depuración, a chamada será repetida.




Seleccionando unha chamada na ventá *Call Stack View*, facendo clic dereito nela e elixindo *Pop To Here*, pódese poñer esa chamada como cabeceira da pila, elimínanse as chamadas posteriores da pila e actualízase o marcador de depuración. A eliminación dunha chamada da pila non implica que se vaian eliminar os efectos causados por esas chamadas. Por exemplo, se unha chamada abre a conexión cunha base de datos e esa chamada se borra da pila, a base de datos permanecerá aberta.

#### Copiar pila en formato texto

Dende a ventá *Call Stack View*, pódese facer clic dereito e elixir *Copy Stack* para pasar a lista de elementos da pila ao portapapeles en formato texto. Por exemplo para a pila anterior:

```
at proxecto.proxecto.factorial.Factorial.factorial(Factorial.java:14)
at proxecto.proxecto.factorial.Factorial.main(Factorial.java:20)
```

## Aplicar cambios no código

Pódense facer certas modificacións no código en tempo de depuración sen ter que reiniciar o programa. Para corrixir o código, débese corrixir na ventá de edición, ir ao menú principal e elixir *Debug* → *Apply Code Changes* ou premer en  na barra de ferramentas de depuración, para recompilar e facer a reparación do código fonte.

Consideracións xerais:

- Se hai erros durante a compilación, non se realizan os cambios e hai que arranxar os erros.
- Se non hai erros, o código obxecto resultante cambiarase polo que se estaba executando na depuración e:
  - Se os cambios do código se fan dentro do método actual que se está a depurar, a pila de chamadas modifícase eliminando a chamada a ese método para permitir volver chamar a ese módulo.
  - Se os cambios se fan despois de ter chamado ao método, non se modificará a pila e para utilizar de novo o código modificado, deben eliminarse as chamadas da pila que conteñan ese código.
- Están excluídas as seguintes modificacións:
  - Cambiar un modificador dun campo, un método ou unha clase.
  - Agregar ou quitar métodos ou campos.
  - Cambiar a xerarquía de clases.
  - Cambiar clases que non foron cargadas na máquina virtual.

## Punto de interrupción

### Definición

Un **punto de interrupción** ou ruptura ou **breakpoint** é unha marca no código fonte que indica ao depurador que se deteña nese punto e espere instrucións para continuar, podendo nese tempo facer operacións de inspección de variables, expresións ou código. Os puntos de interrupción de Java defínense a nivel global e afectan a todos os proxectos que inclúan o código fonte que ten o punto de interrupción. Por exemplo, se Factorial.java tivera un punto de interrupción no método factorial(), cada vez que se depure un proxecto que inclúa esa clase, a sesión de depuración deteríase nese método. NetBeans permite varios tipos de puntos de interrupción:

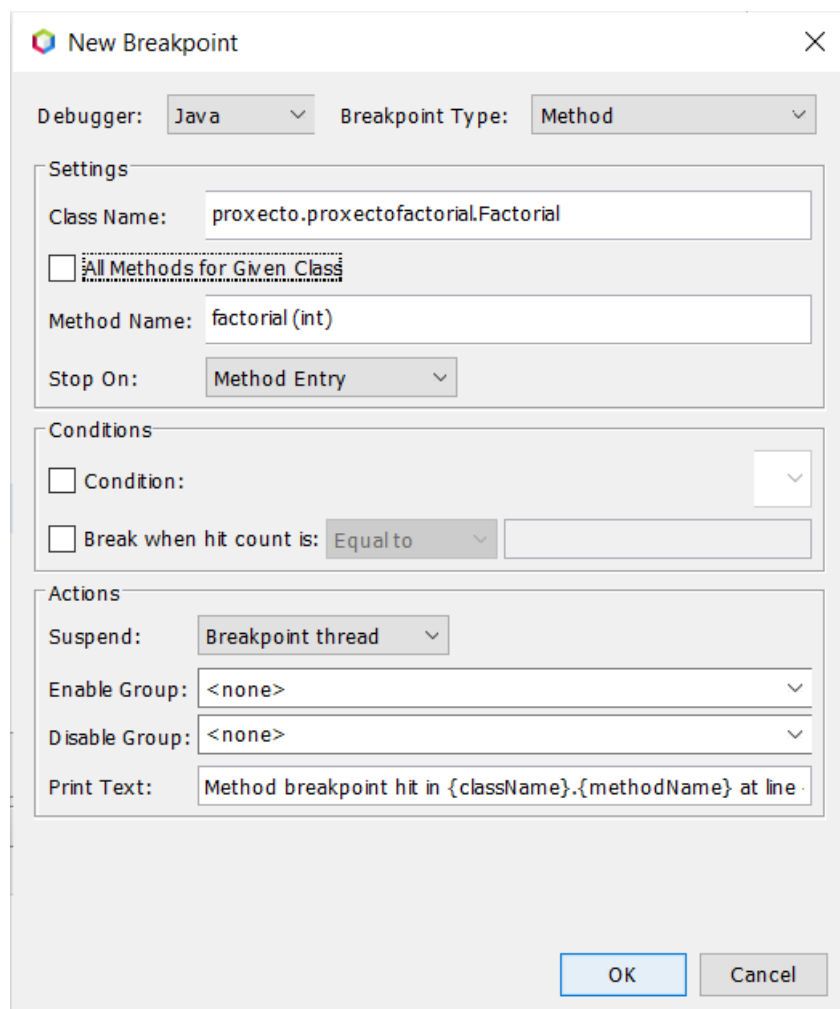
- **Liña:** para que se pare ao chegar a esa liña.
- **Clase:** para que se pare no momento de cargar a clase.

- **Excepción:** para que se pare cando se detecte unha excepción independentemente de se o programa controla esa excepción.
- **Campo:** para que se pare cando se accede e/ou modifica o campo dunha clase.
- **Método:** para que se pare cando se entra e/ou se sae dun método.
- **Subproceso:** para que se pare cando se empeza e/ou finaliza un subproceso ou fío (*thread*).

#### Establecer un punto de interrupción

Para establecer un punto de interrupción, haberá que seleccionar o elemento do código no que se desexa establecer o punto de interrupción e elixir no menú *Debug->New Breakpoint* ou premer **Ctrl+Maiúsculas+F8**. Aparece a caixa de diálogo *New Breakpoint* con información por defecto relacionada co elemento seleccionado e na que terán que facerse os axustes convenientes. O IDE indica o punto de interrupción establecido mediante unha icona na marxe esquerda do código fonte e os puntos de interrupción de liña indícaos ademais poñendo a liña con fondo roxo.

Na seguinte imaxe móstrase un exemplo de punto de interrupción de método para que a depuración se interrompa cando se entre no método factorial da clase Factorial:



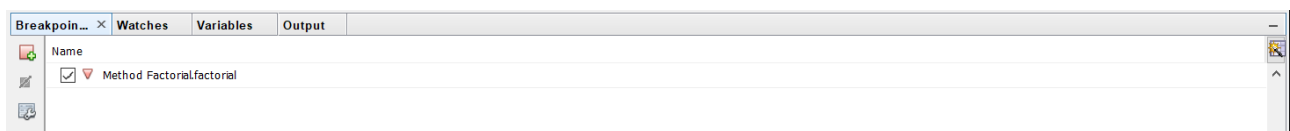
No código fonte aparecerá o punto de interrupción de método marcado cunha icona como aparece na seguinte imaxe:

```

8      public static int factorial(int num) {
9          int factorial=1;
10         //n!=1*2*3*...*n
11         for (int i=1;i<=num;i++) {
12             factorial*=i;
13         }
14         return factorial;
15     }

```

Durante a sesión de depuración, NetBeans comproba a validez dos puntos de interrupción e se non os encontra válidos indícao mediante a icona "rota" no código fonte e mostra unha mensaxe de erro na consola do depurador. Ademais é posible ver a ventá Breakpoint View con información sobre os puntos de interrupción. Se a ventá non está aberta pódese abrir dende o menú principal *Window* → *Debugging* → *Breakpoints* ou premendo Alt+Maiúsculas+5.



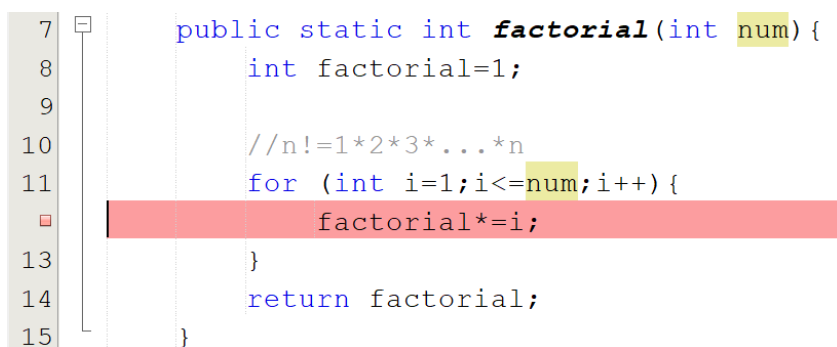
As iconas posibles para marcar os puntos de interrupción son:

| Annotation | Description  |
|------------|--|
|            | Breakpoint   |
|            | Disabled breakpoint  |
|            | Invalid breakpoint   |
|            | Multiple breakpoints   |
|            | Method or field breakpoint   |
|            | Disabled method or field breakpoint  |
|            | Invalid method or field breakpoint   |
|            | Conditional breakpoint   |
|            | Disabled conditional breakpoint  |
|            | Invalid conditional breakpoint   |
|            | Program counter  |
|            | Program counter and one breakpoint   |
|            | Program counter and multiple breakpoints   |
|            | The call site or place in the source code from which the current call on the call stack was made |
|            | Suspended threads  |
|            | Thread suspended by hitting a breakpoint   |

Un punto de interrupción de liña é un os máis utilizados e por iso hai varias maneiras de establecelos:

- O máis sinxelo é facer clic sobre a marxe esquerda da ventá de edición á altura da liña na que se desexa colocar o punto de interrupción.
- Colocar o cursor sobre a liña na que se encontre a instrución onde queremos poñer o punto de interrupción, facer clic co botón dereito e seleccionar a opción *Toggle Line Breakpoint* (Ocultar/Mostrar liña de punto de interrupción) ou premer as teclas Ctrl + F8.
- Colocar o cursor sobre a liña na que se encontre a instrución onde queremos poñer dito punto, e no menú principal elixir *Debug* → *Toggle Line Breakpoint* (Ocultar/Mostrar liña de punto de interrupción) ou premer as teclas Ctrl + F8.

No código fonte aparecerá o punto de interrupción marcado como se indica na seguinte imaxe.



```
7  public static int factorial(int num) {  
8      int factorial=1;  
9  
10     //n!=1*2*3*...*n  
11     for (int i=1;i<=num;i++){  
12         factorial*=i;  
13     }  
14     return factorial;  
15 }
```

The image shows a code editor with a line breakpoint set on line 11. The breakpoint is represented by a small red square in the left margin. The code is a Java function to calculate the factorial of a number. The variable 'num' is highlighted in yellow in the original image.

#### Executar ata o punto de interrupción

Para poder executar un proxecto principal ata un punto de interrupción, hai que ter definido o punto de interrupción e depurar o proxecto elixindo por exemplo no menú principal a opción *Debug* → *Debug project (nome de proxecto)*, que executará o programa principal ata o primeiro punto de interrupción, ou excepción ou ata o final se non existen puntos de interrupción. A partir do punto de interrupción, pódese seguir depurando coas opcións xa vistas.

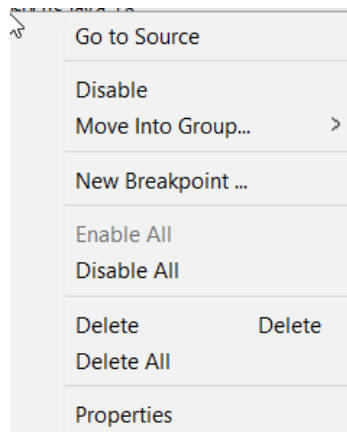
#### Modificar un punto de interrupción

Dende que se establece un punto de interrupción, pódese desactivar (queda rexistrado pero non en uso) se é que está activado, activar se é que estaba desactivado, eliminar (desaparece), ou modificar. Todas estas operacións empézanse a realizar dende a ventá *Breakpoints View* e son:

- Unha forma rápida de activar ou desactivar un punto é marcar ou desmarcar o textbox correspondente na ventá *Breakpoints View*.



- Unha forma rápida de eliminar un punto é colocar o cursor sobre o nome do punto de interrupción na ventá anterior e premer Supr.
- Todas as operacións de modificación dun punto de interrupción pódense facer colocando o rato sobre o nome do punto de interrupción na ventá anterior e facendo clic co botón dereito. Aparece unha lista de operacións posibles:



- No caso de estar sobre un punto de interrupción activo, na ventá de opcións aparece dispoñible a opción *Disable* (Desactivar); se estivera desactivado, aparecería no seu lugar a opción *Enable* (Activar).
- As opcións de desactivar todo, activar todo ou eliminar todo terán efecto sobre todos os puntos de interrupción da ventá.
- A opción *Properties* visualiza a ventá *Breakpoint Properties* (Propiedades de punto de interrupción), na que se pode axustar a configuración do punto de interrupción.
- Tamén se pode crear un novo punto de interrupción.

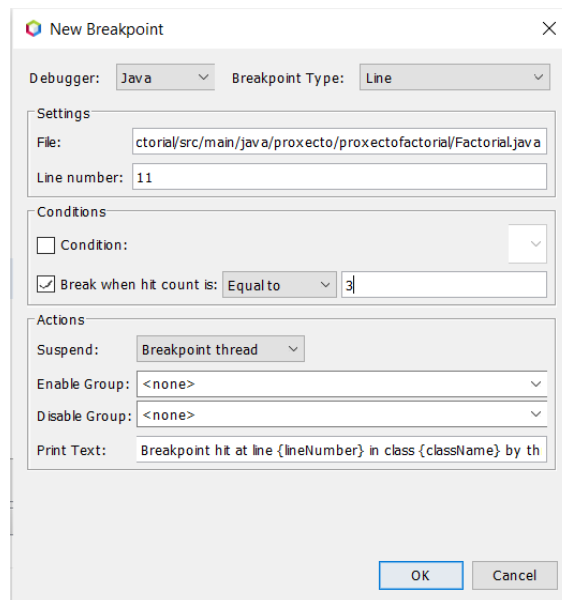
#### Poñer condicións ao punto de interrupción

Pódense poñer condicións para que un punto de interrupción pare unha depuración; algunhas son comúns para todos os puntos de interrupción e outras dependen de se son de fío, se son de clase ou se son de excepción.

#### Condicións válidas para todos os puntos de interrupción

Todos os puntos de interrupción teñen a posibilidade de parar unha depuración en función dunha frecuencia establecida, marcando o checkbox Break when hit count is: - *Pausa cando se contabilice un alcance*, seleccionando un criterio da lista despregable (Equal to - *Igual a*, Greater than - *é maior que*, Multiple of - *é múltiplo de*) e establecendo un valor numérico para ese criterio na ventá de propiedades do punto.

A seguinte imaxe mostra a condición de que o punto de interrupción de liña situado na liña 11 de Factorial.java se active a terceira vez que se pase por ela durante unha sesión de depuración.

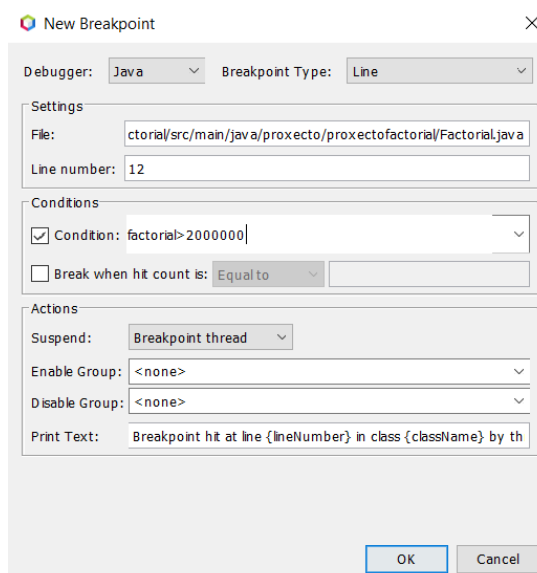


### Condicións válidas para todos os puntos agás os de tipo *thread*

Os puntos de interrupción que non son tipo *fío*, teñen a posibilidade de parar unha depuración cando unha determinada condición é certa. Esta condición establécese na ventá de propiedades do punto de interrupción, seleccionando *Condition* - *Condición* e tecleando a condición. A condición debe seguir as normas de sintaxe de Java e pode incluír variables e métodos utilizados no contexto actual coas seguintes excepcións:

- As importacións son ignoradas. Débense usar nome completos como `obj instanceof java.lang.String`
- Non se pode acceder directamente a métodos e variables de clases externas. Débese utilizar `this.nome` ou `this.$1`.

A seguinte imaxe mostra a condición de que o punto de interrupción de liña situado na liña 12 de `Factorial.java` se active cando a variable `factorial` teña un valor maior a 2000000 durante unha sesión de depuración.



### Condições específicas para os pontos de classe e exceção

Pódense dar as seguintes condicións específicas:

- Os puntos de interrupción de clases tamén permiten poñer como condición a exclusión dalgunha clase.
- Os puntos de interrupción de excepcións permiten poñer como condición un filtrado de clase a incluír ou excluír.