# ENTORNOS DE DESARROLLO - 1º DAM

# UD 3. SISTEMAS DE CONTROL DE VERSIONES

## 

## E01 Práctica guiada de depuración

|  |  |
| --- | --- |
| **Resultado de Aprendizaje 3**: Evalúa entornos integrados de desarrollo analizando sus características para editar y depurar código fuente y generar ejecutables. | |
| **Criterio de evaluación** | **Ejercicio/s** |
| RA03.b Se han identificado las herramientas de depuración y prueba de aplicaciones ofrecidas por el entorno de desarrollo, definiendo puntos de ruptura. | ÚNICO |
| RA03.c Se han utilizado las herramientas de depuración para examinar y modificar el comportamiento de un programa en tiempo de ejecución. |  |

|  |
| --- |
| Como entrega, se debe adjuntar un documento de **Google Docs** en el que se puedan ver las capturas de pantalla que evidencien que se han realizado los pasos indicados en este trabajo. |

Vamos a empezar a usar el debugger de eclipse, nada más algunas cosas básicas para los que empiezan con él.

1 Nuestro proyecto

[Creamos un proyecto java en eclipse](http://chuwiki.chuidiang.org/index.php?title=Crear_un_proyecto_con_Eclipse) y ponemos un par de clases para empezar a jugar con el debugger. Una con el main puede ser PruebaDebugger y la otra, por aquello de tener varias, puede ser ClaseParaDepurar. El código en ellas cualquiera, por ejemplo, el siguiente

package com.chuidiang.ejemplos;

public class PruebaDebugger {

public static void main(String[] args) {

ClaseParaDepurar clase = new ClaseParaDepurar();

double valor = clase.echaCuentas();

System.out.println("Sale "+valor);

}

}

package com.chuidiang.ejemplos;

import java.util.LinkedList;

import java.util.List;

public class ClaseParaDepurar {

private double suma;

private List<Double> lista;

public ClaseParaDepurar() {

System.out.println("construyendo clase");

}

public double echaCuentas() {

lista = new LinkedList<Double>();

for (int i = 0; i<100;i++){

lista.add((double)i);

}

suma = 0.0;

for (Double valor : lista){

suma +=valor;

}

return suma;

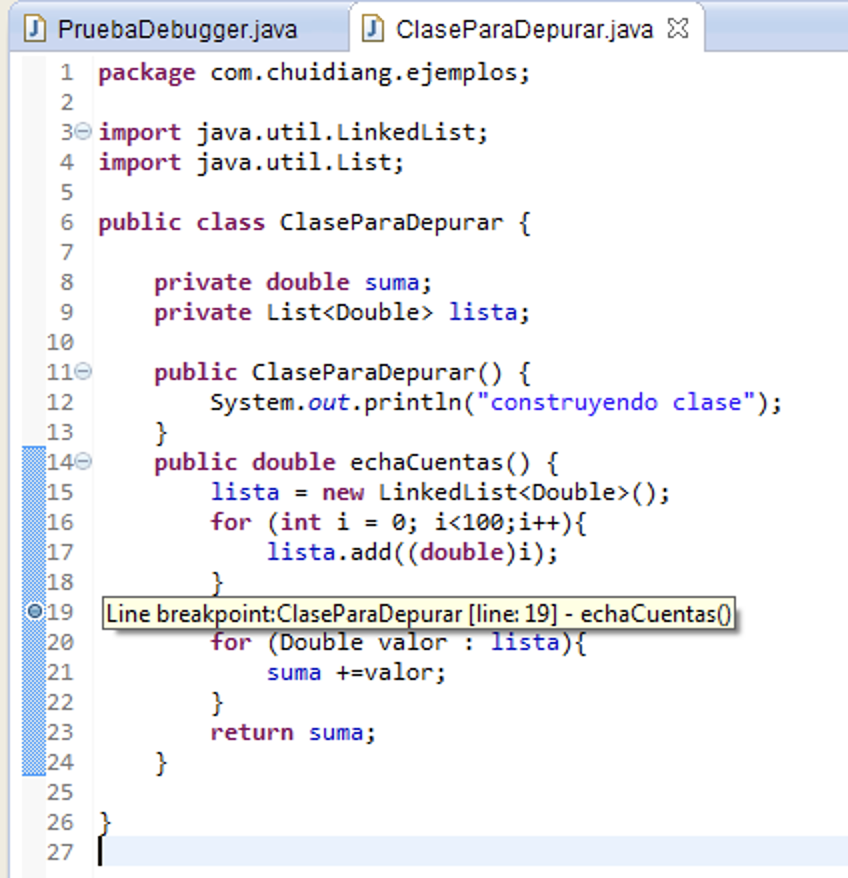
}

}

2 Poner un Breakpoint

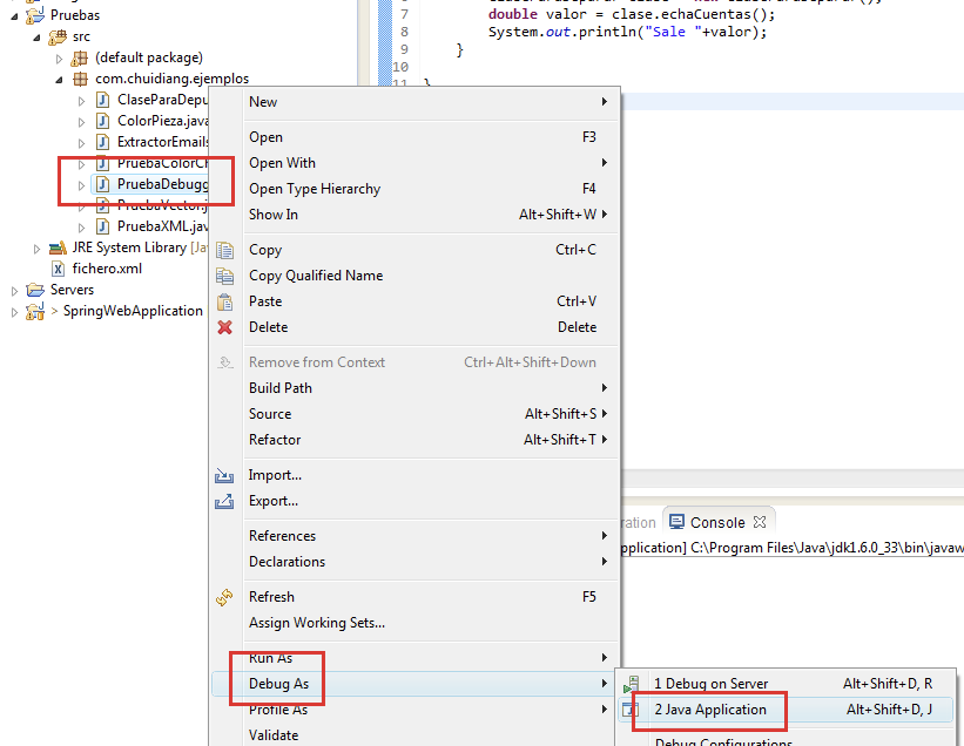
Un Breakpoint es un punto en nuestro código en el que al ejecutar con debugger, la ejecución se detendrá y podremos ver los valores que tienen las variables en ese momento, modificarlos, etc, etc. Poner un BreakPoint es sencillo, basta ir a nuestra clase y hacer doble click en el margen izquierdo del editor, justo en la línea donde queremos que la ejecución se detenga. En la figura vemos un circulito a la izquierda de la línea 19, ese es el breakpoink que acabamos de poner haciendo doble click justo ahí.

Si queremos quitarlo, nos basta con hacer doble click en él nuevamente.



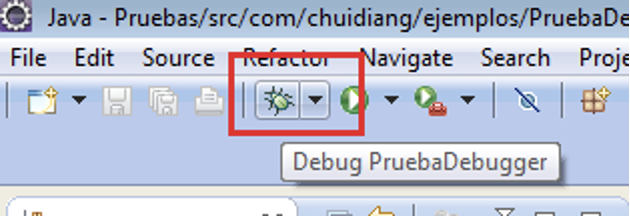
3 Arrancar el programa en modo debugger

Para arrancar el programa en modo debugger, sacamos el menú con el botón derecho del ratón sobre la clase que tiene el main. Buscamos "debug as" -> "java application"



Esto arrancará nuestro programa en modo debugger y se ejecutará de forma normal hasta que la ejecución pase por un Breakpoint, en ese momento se detendrá.

Este proceso del menú y "debug as" -> "Java application" sólo es necesario la primera vez. Las siguientes veces que queramos ejecutar, podemos usar el botón en la parte superior con el dibujo de una cucaracha (un "bug")



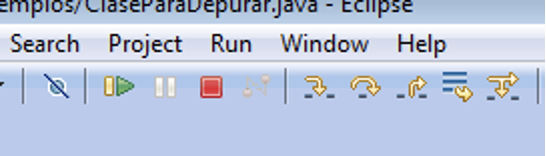
4 La vista de debugger

Cuando arranca el programa en modo debug, eclipse posiblemente nos pregunte si queremos pasar a la vista de debug. Le indicamos que sí, marcando el check de "Remember my decision" si no queremos que nos lo pregunte cada vez que arranquemos el debugger sin estar en esa vista.

El programa se ejecutará de forma normal hasta llegar a un breakpoint. Es ese momento se parará y tendremos una serie de "cosas" útiles en la vista de debugger para poder ver variables, hacer que el programa se ejecute paso a paso, ver quién llama a quién, etc.

**5** **Botones para ir paso a paso**

En la vista de debugger tenemos botones para ir ejecutando paso a paso a partir del breakpoint donde nos hemos parado. Están en la parte superior y son como unas flechitas variadas amarillas



El primer botón de la imagen, un breakpoint tachado, deshabilita temporalmente todos los breakpoints. Si hacemos que nuestro código corra, no se parará en ningún breakpoint.

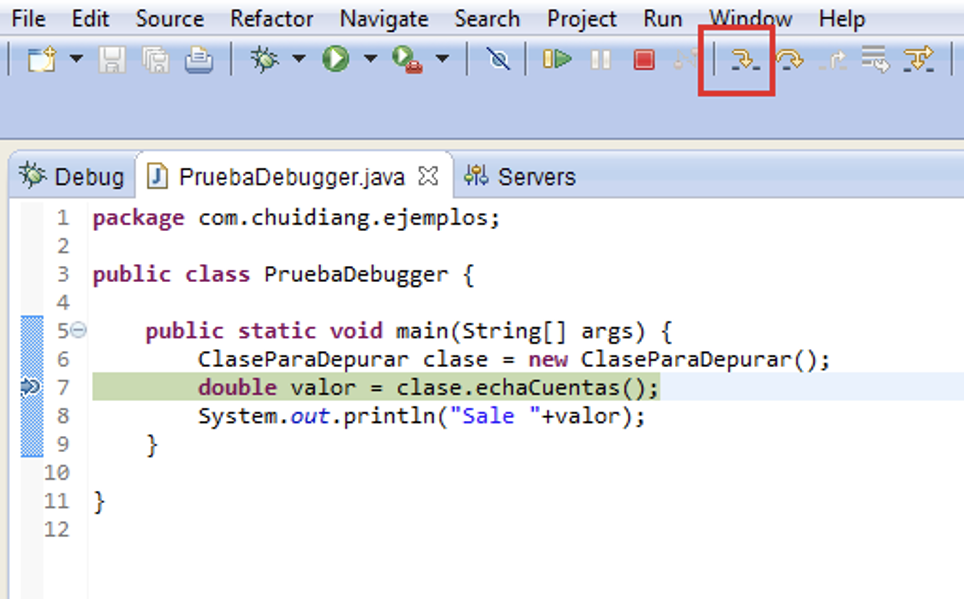
El siguiente botón, una flecha verde parecido al "play" de cualquier reproductor de música/video, hace que continue la ejecución de nuestro programa hasta el siguiente Breakpoint.

El siguiente botón, parecido al botón de pausa de cualquier reproductor de música/video, suspende la ejecución del programa que está corriendo. Es una forma de parar "al vuelo" el programa, esté donde esté en ese momento, haya o no breakpoint.

El cuadrado rojo es el botón de stop, aborta la ejecución de nuestro programa.

El siguiente botón que apenas se ve y cuyo tooltip es "disconnect" no tiene mucha utilidad en nuestro caso. Eclipse ofrece la posibilidad de [conectarse a un programa java que se está ejecutando fuera de eclipse para depurarlo](http://chuwiki.chuidiang.org/index.php?title=Debug_remoto_con_Eclipse). Este botón nos desconectaría de ese programa remoto que estamos depurando.

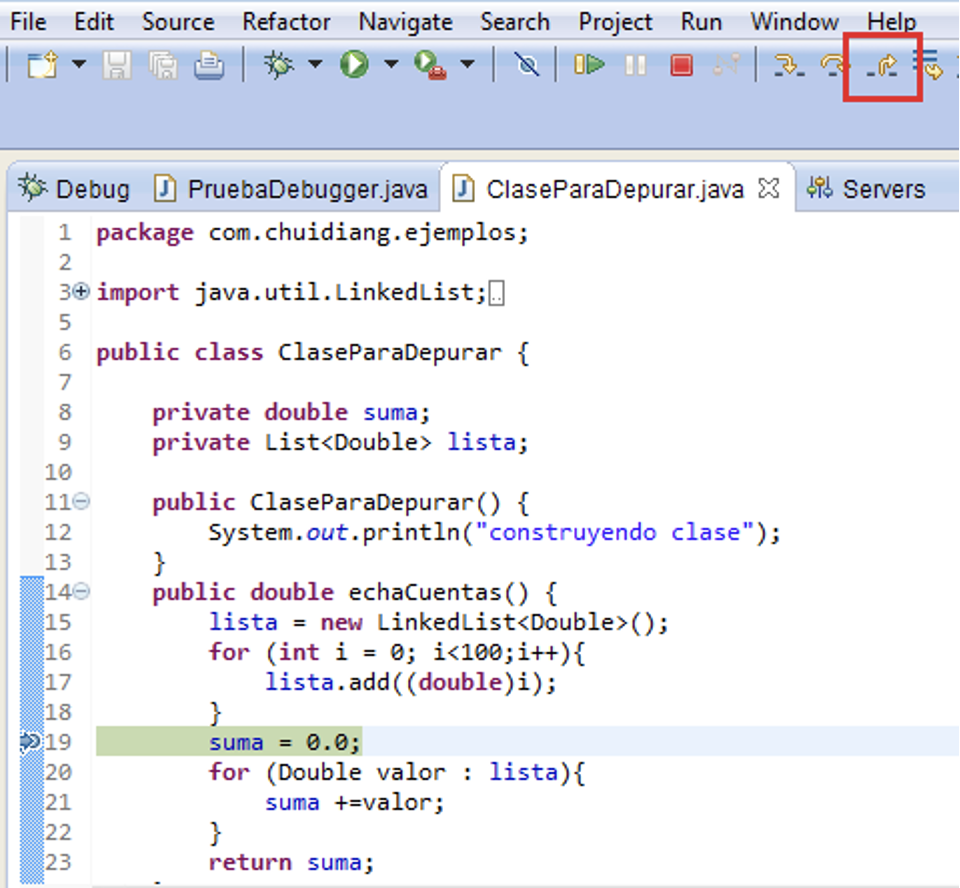
El siguiente botón, una flecha amarilla que se "mete" entre dos puntos negros, imagina que nuestro programa tiene el breakpoint en una línea en la que se hace una llamada a un método y se ha parado ahí. Pulsando este botón, el debugger se meterá dentro del método y se parará en la primera línea del mismo, de forma que podemos depurar dentro del método. Por ejemplo, si el breakpoint está donde se muestra en la siguiente figura



al pulsar el botón marcado en rojo, el debugger se meterá dentro del método echaCuentas() y se parará en la primera línea de ese método.

El siguiente botón, una flecha amarilla que salta sobre un punto negro, avanza un paso la ejecución del programa, pero sin meterse dentro del método que encuentre. Por ejemplo, en la figura anterior, si pulsáramos este botón, la ejecución pasaría a la línea System.out.println(), sin pararse dentro del método echaCuentas().

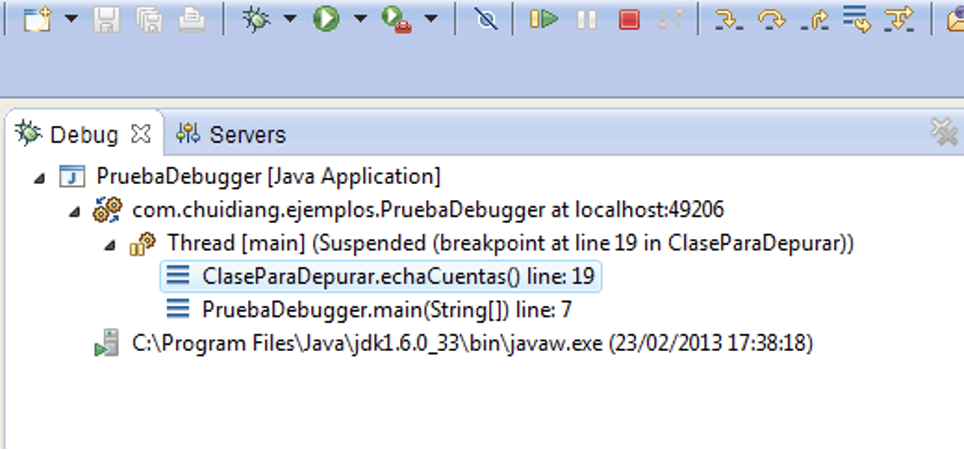
Finalmente, el último botón es una flecha saliendo de entre dos puntos negros, hace avanzar el programa hasta que nos salimos del método actual y vamos hasta el sitio donde ha sido llamado. Por ejemplo, si el debugger está parado donde la figura y pulsamos este botón



el debugger pasará justo al código dose se está llamando a este método, es decir, al echaCuentas() anterior.

**6** **La pila de llamadas**

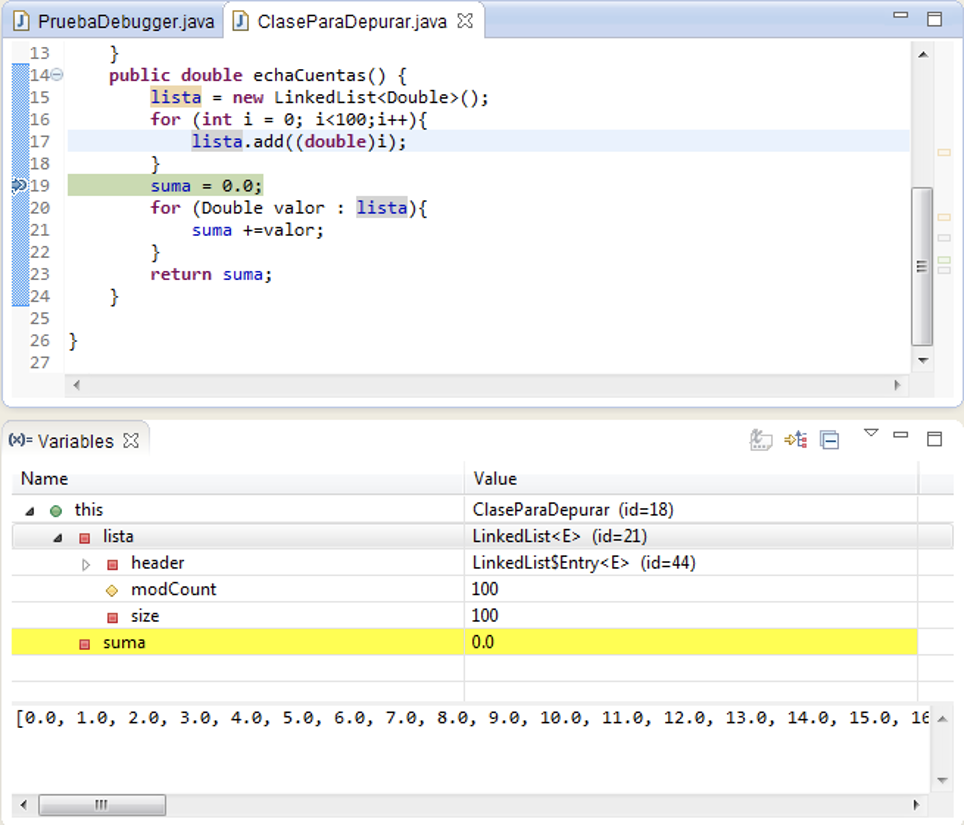
Tenemos una ventana, etiquetada "Debug" en la que aparece la pila de llamadas, como la de la siguiente figura



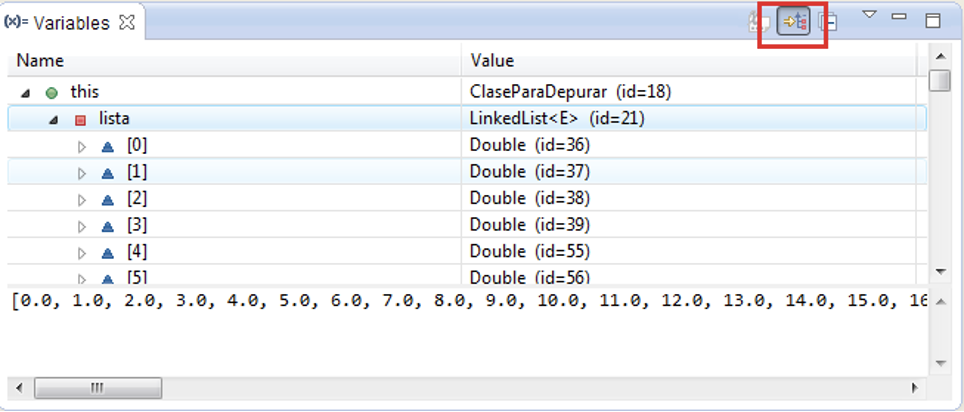
En ella podemos ver cada uno de los hilos en ejecución (Thread main en la figura, único hilo existente en esta mini aplicación) y debajo el fichero en el que ahora está la ejecución parada (marcada en azul en la figura), quién ha llamado a esa clase (línea justo debajo con el PruebaDebugger.main). Haciendo doble click en estas líneas, iremos al fichero justo al sitio donde se está ejecutando el código.

**7** **Visualizar y modificar variables**

Tenemos una ventana etiquetada Variables. En ella podemos ver todas las variables definidas en el punto en el que está parado nuestro código. Es la de la siguiente figura



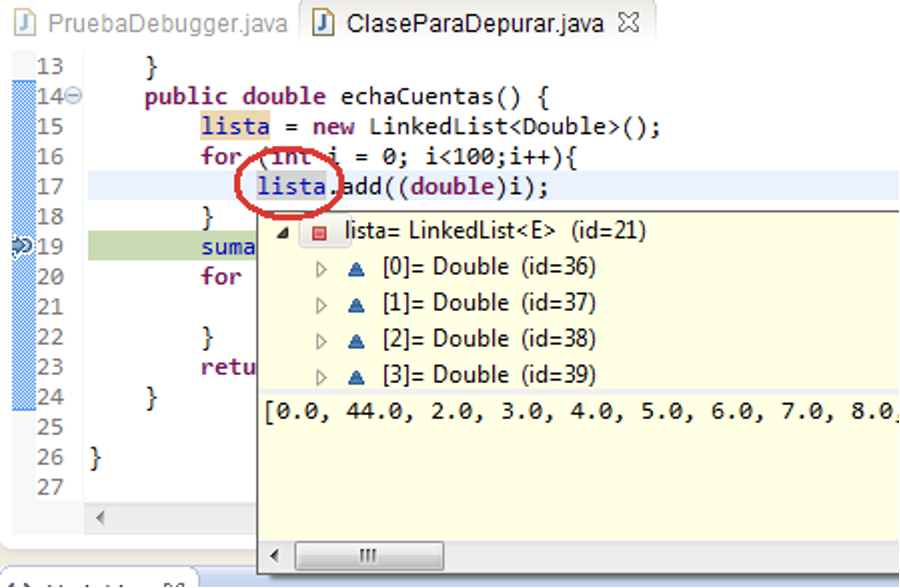
Vemos que en ese punto de ejecución nuestra lista ya tiene sus 100 elementos, pero suma todavía vale 0.0. La lista se ve un poco "rara" ya que estamos viendo todos los detalles internos de la clase LinkedList que hemos usado. Hay un botón que si lo pulsamos nos permite ver las colecciones de una forma más amigable, es el que se muestra en la siguiente figura



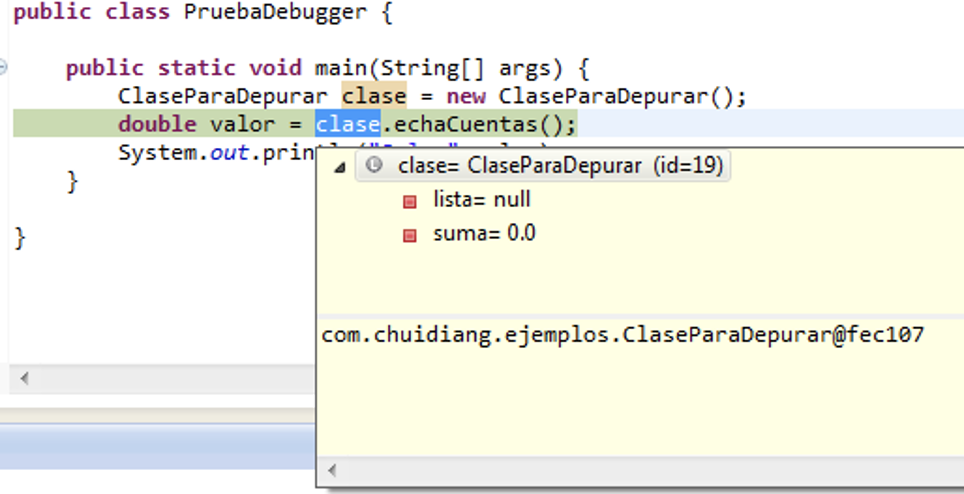
Vemos que ahora sólo se ven los elementos de la LinkedList, no sus "cosas" internas.

En esta ventana es fácil modificar el valor de una variable, basta con hacer click en el valor y cambiarlo. Por ejemplo, en la figura donde se ve suma=0.0, si hacemos click en el 0.0, podremos poner el valor que queramos. Esto valor se usará en los siguientes pasos de ejecución del debugger.

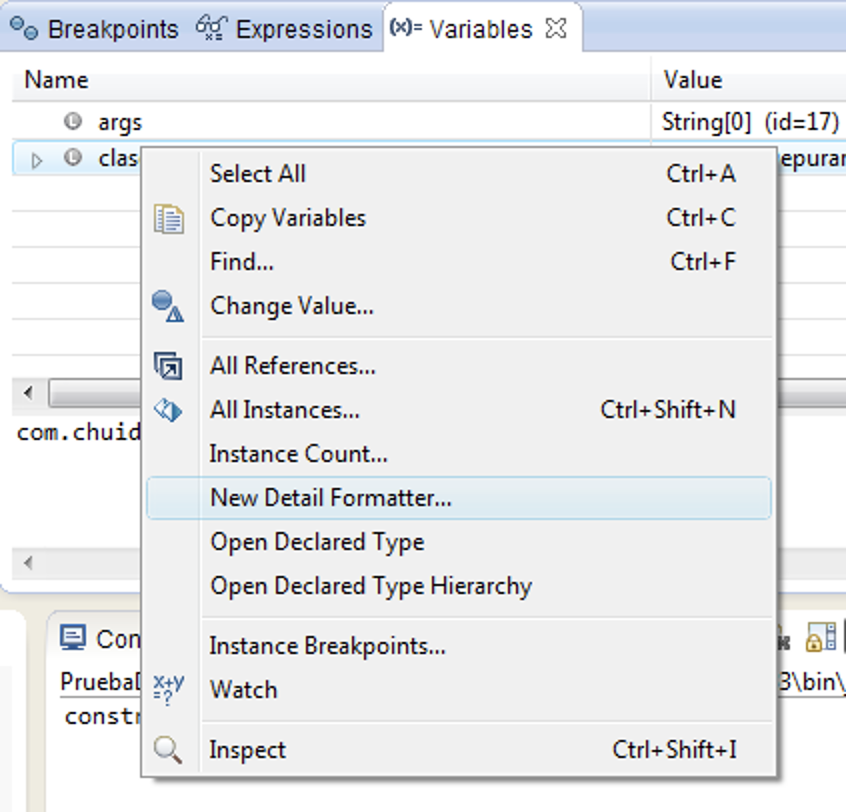
Otra forma de visualizar el valor de una variable consiste simplemente en poner el ratón sobre ella en el código. Aparecerá un tooltip con su contenido. En la siguiente figura hemos puesto el ratón sobre la variable lista señalada en rojo, aparece el tooltip con todos los valores.



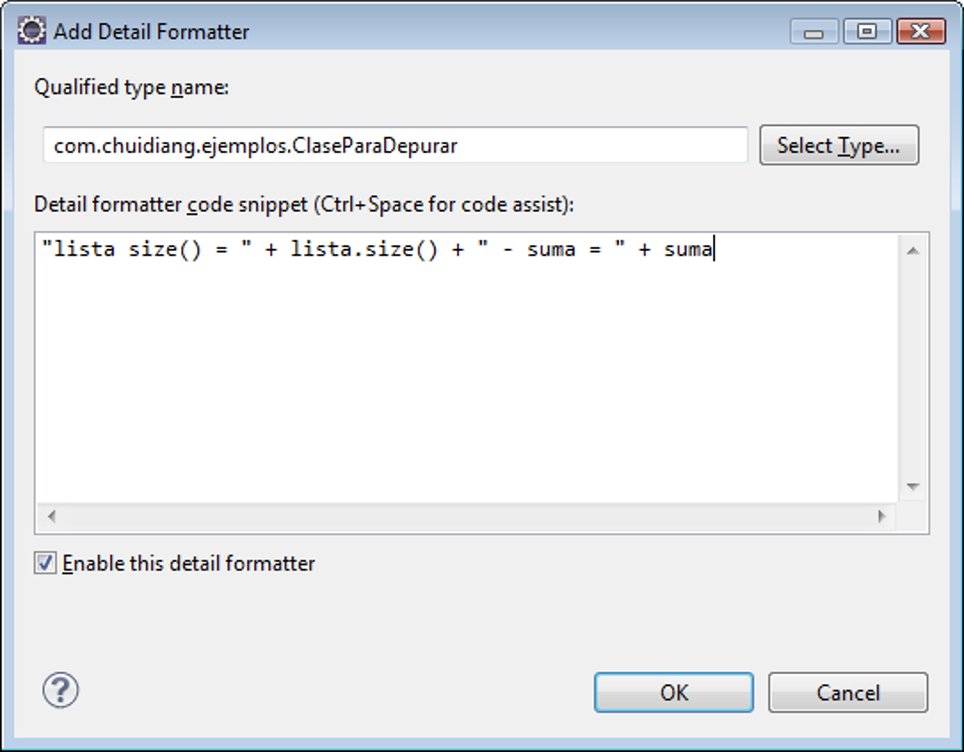
Cuando se muestra el valor de una variable, se llama a su método toString(), por lo que a veces la variable nos puede quedar algo fea y veremos algo como lo de la siguiente figura para la variable "clase". No es más que una llamada al método toString() de esa clase y lo que sale es lo que muestra



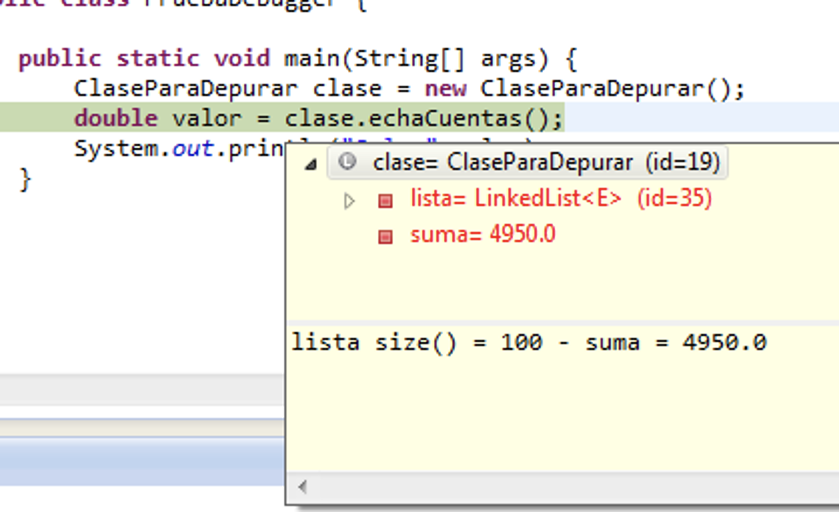
Vemos que sale el nombre de la clase, seguido de una @ y un número. Esto no nos dice nada. Afortunamente, podemos decirle a eclipse cómo queremos que se visualice una clase. Vamos a la ventana donde tenemos las variables, sacamos el menú con el botón derecho del ratón sobre la variable que nos interese y elegimos "New detail formatter"



Esto nos abrirá una ventana con una caja de texto donde podemos poner lo que queramos que se muestre para visualizar esa variable. Por ejemplo, en nuestra clase podemos querer que se vea el tamaño de la lista y el valor de la suma, así que nos basta con poner algo como lo de la siguiente figura

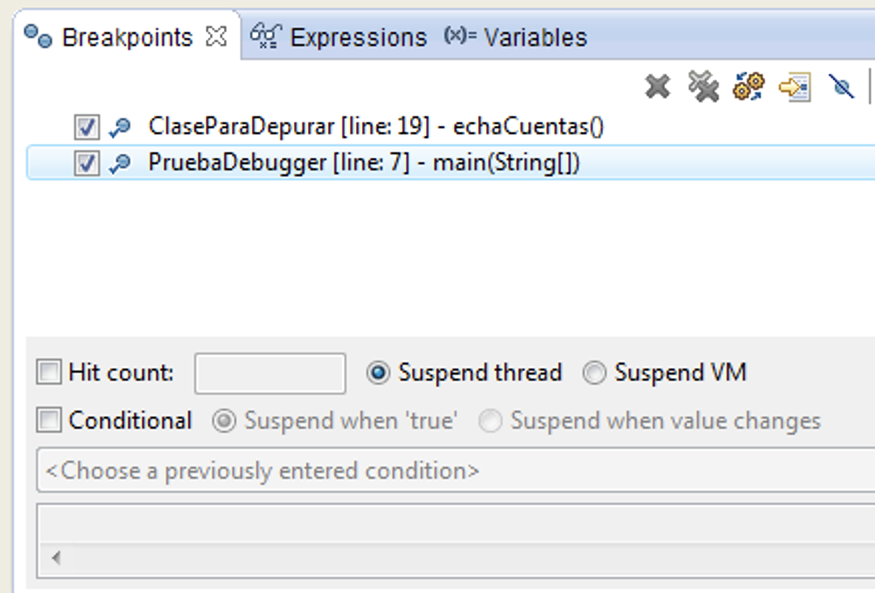


Obviamente, si en el ejemplo lista es null, la llamada a lista.getSize() dará un NullPointerException, y eso es lo que veremos al visualizar la variable. Si no es null, veremos algo como esto

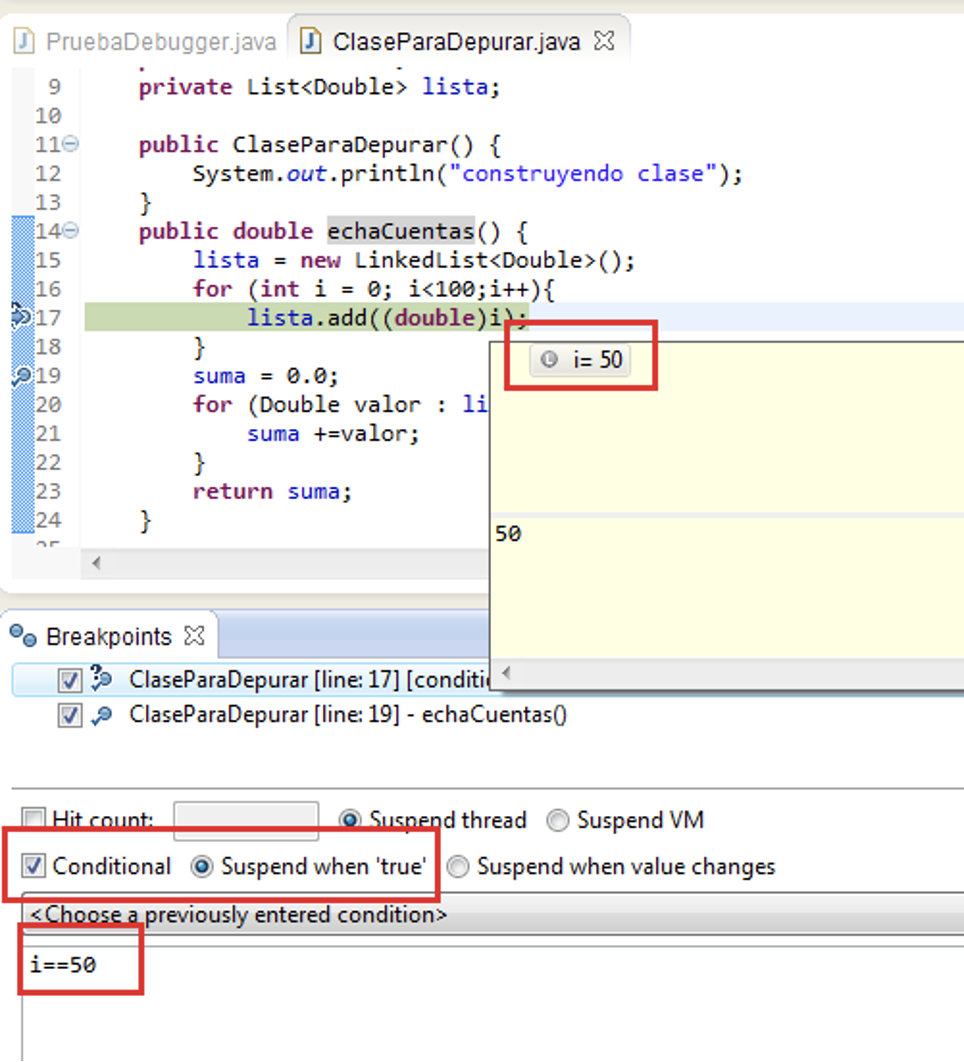


**8** **Breakpoints condicionales**

Tenemos otra ventana más con todos los breakpoints. Ahí podemos habilitarlos o deshabilitarlos sin más que marcar/desmarcar el check.



Lo que es más interesante es lo que se ve en la parte de debajo la ventana y es que podemos hacer que el código se pare o no en el breakpoint según alguna condición. Por ejemplo, ponemos el breakpoint en medio de un bucle, por ejemplo en el bucle donde estamos llenando la lista. Podemos hacer que el código se pare, por ejemplo, cuando estemos en la iteración 50 del bucle (cuando i==50), de esta forma



Hemos marcado el "check" etiquetado "conditional" y el etiquetado "Suspend when true". Debajo hemos puesto la condición "i==50" y hemos arrancado el debugger, cuando se ha detenido visualizamos el valor de la variable i, y efectivamente es 50.