Pruebas de Unidad en BlueJ

Versión 1.0
para BlueJ versión 1.3.0

Michael Kölling
Mærsk Institute
University of Southern Denmark

Traducido al español por Matías Avallone
Instituto de Tecnología Ort Nº 2
Buenos Aires, Argentina
Diciembre de 2005

1 Introducción

Resumen: BlueJ proporciona funcionalidad de prueba de regresión integrando JUnit

1.1 Acerca de este tutorial – alcance y audiencia

Este tutorial introduce la funcionalidad de prueba de unidad en el entorno BlueJ. Asumimos que usted está familiarizado con la funcionalidad general de BlueJ. De no ser así, lea 'El Tutorial de BlueJ' primero (usted puede obtener este tutorial, y una versión electrónica de éste en http://www.bluej.org/doc/documentation.html).

También asumimos que usted está algo familiarizado con la idea de prueba de unidad (o al menos de prueba de software en general). Damos algunos indicadores en la siguiente sección.

1.2 ¿Qué es prueba de unidad?

El término *prueba de unidad* se refiere a la prueba individual de unidades separadas de un sistema de software. En sistemas orientados a objetos, estas unidades son, típicamente, clases y métodos. Así, en nuestro contexto, prueba de unidad se refiere a la prueba individual de métodos y clases en BlueJ.

Este tutorial discute herramientas de BlueJ para la prueba sistemática de unidad. Si usted está familiarizado con las características de la interacción de BlueJ, entonces usted sabe que es fácil en BlueJ probar métodos individuales interactivamente. Nos referimos a esto como *pruebas ad-hoc*.

Las *pruebas ad-hoc* son útiles, pero no lo suficientemente buenas para pruebas sistemáticas. Las características de pruebas de unidad en BlueJ le brinda las herramientas para grabar y reproducir pruebas, para fácilmente poder repetir las pruebas de unidad más adelante (típicamente, después de un cambio en el sistema), de modo que el desarrollador pueda ganar confianza de que los cambios recientes no han roto la funcionalidad existente. Esto se conoce como *pruebas de regresión*.

Los conceptos de prueba de unidad y prueba de regresión son viejos, pero aumentaron su popularidad recientemente con la publicación de la metodología de *programación eXtrema* (*eXtreme programming*)[1] y la herramienta de prueba de unidad para Java, *JUnit*.

JUnit es un framework de prueba de regresión escrito por Erich Gamma y Kent Beck. Se puede encontrar el software y mucha información en *http://www.junit.org* .

[1] Para descubrir qué es programación eXtrema, lea, por ejemplo, "*Extreme Programming Explained: Embrace Change*" ("*Programación eXtrema explicada: adopte el cambio*"), Kent Beck, Addison Wesley, 1999. Hay muchos otros libros disponibles. Hay un buen resumen en línea en http://www.xprogramming.com/xpmag/whatisxp.htm

1.3 Prueba de unidad en BlueJ

Las herramientas de prueba sistemática en BlueJ están basadas en JUnit, Así, un cierto conocimiento en el uso de JUnit ayuda a entender la prueba de unidad en BlueJ. Nosotros recomendamos leer algún artículo acerca de esto (quizás no ahora, pero sí otro momento). Existen muchos artículos, y el sitio web de JUnit es un buen punto de partida para encontrarlos.

La prueba de unidad en BlueJ combina la funcionalidad interactiva de prueba de BlueJ con la prueba de regresión de JUnit. Ambos métodos de prueba están completamente soportados. Adicionalmente, está disponible nueva funcionalidad resultante de la combinación de los dos sistemas: secuencias de prueba interactiva pueden ser grabadas, por ejemplo, para crear automáticamente métodos de prueba de JUnit para posteriores pruebas de regresión. Los ejemplos se brindan más adelante en este documento.

La funcionalidad de la unidad de prueba en BlueJ fue diseñada e implementada por Andrew Patterson (Monash University) como parte de este trabajo.

2 Permitir la funcionalidad de prueba de unidad

Resumen: Las herramientas de prueba se pueden visualizar habilitando la opción en preferencias.

El soporte explícito de pruebas en BlueJ está inicialmente deshabilitado. Para usar las herramientas de prueba seleccionar Tools – Preferencies... (Herramientas – Preferencias...) y seleccione la opción llamada Show unit testing tools (Mostrar herramientas de prueba).

Habilitando esta funcionalidad se agregan tres elementos a la interfaz: algunos botones y un indicador "recording" ("grabando") en la barra de herramientas de la ventana principal, un ítem Show Test Results (Mostrar Resultado de Pruebas) en el menú View (Ver), y un ítem Create Test Class (Crear Clase de Prueba) en el menú contextual de una clase compilada.

3 Creando clases de prueba

Resumen: Cree una clase de prueba seleccionando Create Test Class (Crear Clase de Prueba) *del menú contextual de la clase.*

El primer paso para la creación de la prueba de una clase o método en BlueJ es crear una clase de prueba.

Una clase de prueba es una clase asociada con una clase del proyecto (la cual llamaremos *clase de referencia*). La clase de prueba contiene pruebas para los métodos de la clase de referencia.

Para los ejemplos de este tutorial, utilizaremos el proyecto *people*, el cual es distribuido con BlueJ como uno de los ejemplos en el directorio *examples*. Usted puede abrirlo en su sistema para probar mientras lee esto.

Puede crear una clase de prueba haciendo click con el botón derecho (MacOS: control-click) en una clase compilada y seleccionando Create Test Class (Crear Clase de Prueba) del menú contextual. El nombre de la clase de prueba se crea automáticamente agregándole el sufijo "Test" al nombre de la clase de referencia. Por ejemplo, si el nombre de la clase de referencia es "Student", la clase de prueba se llamará "StudentTest".

Las clases de prueba se muestran en el diagrama marcadas con la etiqueta <<unit test>> (prueba de unidad) agregada a la clase de referencia. También tienen un color diferente (figura 1). Arrastrando la clase de referencia mantendrá la clase de prueba unida.

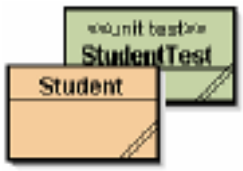

Figura 1: Una clase de referencia con su clase de prueba asociada

Las clases de prueba son tratadas de una forma especializada por el entorno. Tienen las funciones de clase usuales, como Open Editor (Abrir Editor), Compile (Compilar), Remove (Remover), pero también algunas funciones específicas de prueba (figura 2). La clase de prueba debe ser compilada para ver este menú.

Figura 2: El menú contextual de una clase de prueba

Crear una clase de prueba en sí mismo no crea ninguna prueba, pero nos da la opción de crear pruebas ahora. La clase de prueba es utilizada para mantener las pruebas que crearemos.

4 Creando métodos de prueba

Resumen: Cree un método de prueba seleccionando Create Test Method... (Crear Método de Prueba...) *del menú de la clase de prueba.*

Los objetos Student tienen dos métodos, `setName` y `getName` (heredados de Person), para asignar y recuperar el nombre del estudiante. Asumimos que queremos crear una prueba para chequear que esos métodos funcionan como esperamos.

Empezamos seleccionando Create Test Method... (Crear Método de Prueba...) desde la clase StudentTest. Un *método de prueba* implementa una única prueba (esto es: la prueba de una porción de funcionalidad).

Después de seleccionar esta función, será requerido un nombre para esta prueba. Los nombres de las pruebas siempre comienzan con el prefijo "test" – si el nombre elegido no comienza con "test", éste será automáticamente agregado. Así, ingresar "testName" o "name" creará en ambos casos un método de prueba llamado "testName".

Después de tipear el nombre y presionar Ok, toda la interacción será grabada como parte de esta prueba. El indicador 'recording' ('grabando') está encendido, y los botones para finalizar o cancelar esta grabación de la prueba están habilitados (figura 3).

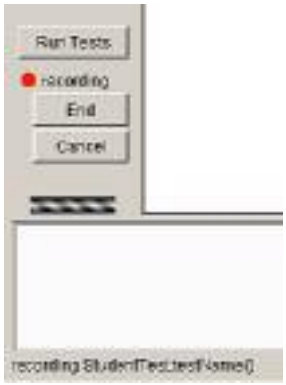

Figura 3: Botones de pruebas durante la grabación

Para grabar una prueba en este ejemplo, haga lo siguiente:

Cree un objeto Student, usando el constructor sin parámetros.

Llame al método `setName(newName)` (heredado de Person) y asigne el nombre "Fred"

Llame al método `getName()`

Después de llamar al método `getName` usted verá el diálogo de resultado. Mientras estamos grabando pruebas, los diálogos de resultados incluyen una parte que nos permite especificar aserciones en el resultado (figura 4). Podemos usar estas facilidad de aserciones para especificar el resultado previsto de la prueba. En nuestro caso, esperamos que el resultado de la llamada al método sea igual a la cadena "Fred", entonces podemos especificar esto como una aserción (figura 4).

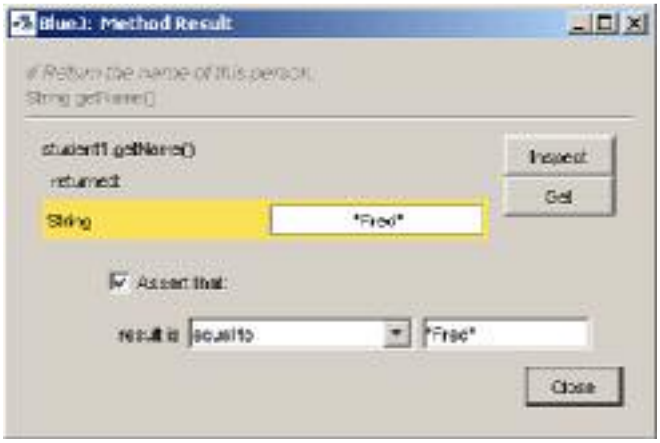

Figura 4: Un diálogo de resultado con opciones de aserción

Están disponibles diferentes clases de aserciones en el menú, incluyendo las pruebas de igualdad, nulos, y no nulos.

Esto concluye nuestro caso de prueba, así que podemos ahora hacer click en 'End' ('Fin') debajo del indicador de grabación de la prueba para finalizar.

Finalizando, los resultados de la ejecución del método de prueba se agregan a la clase de prueba. Este método de prueba está entonces disponible para ejecutar.

Podemos utilizar el botón de 'cancel' ('cancelar') para terminar la grabación y descartarla.

De una manera similar a este ejemplo, podemos registrar cualquier número de pruebas. Cada clase en el proyecto puede tener su propia clase de prueba, y cada clase de prueba puede tener cualquier número de pruebas.

Cada grabación de prueba puede consistir en un número arbitrario de acciones, incluyendo la creación arbitraria de casos y de cualquier número de aserciones.

5 Ejecutando pruebas

Resumen: Ejecute todas las pruebas haciendo click en el botón Run Tests (Ejecutar pruebas). *Ejecute pruebas individuales seleccionándolas del menú de la clase de prueba.*

Una vez que las pruebas fueron grabadas, éstas pueden ser ejecutadas. Las clases de prueba son clases Java, como las clases de referencia, por lo tanto deben ser compiladas antes de la ejecución.

BlueJ automáticamente intenta compilar las clases de prueba luego de cada grabación. Si la expresión de aserción contiene un error, o si la clase de prueba fue editada a mano, es necesario compilar la clase de prueba explícitamente antes de que pueda ser usada.

Podemos ahora hacer click con el botón derecho a la clase de prueba y ver la prueba que grabamos en el menú contextual de la clase. La figura 5 muestra un ejemplo de nuestro método de prueba `testName` de antes y una segunda prueba llamada `testStudentID`.

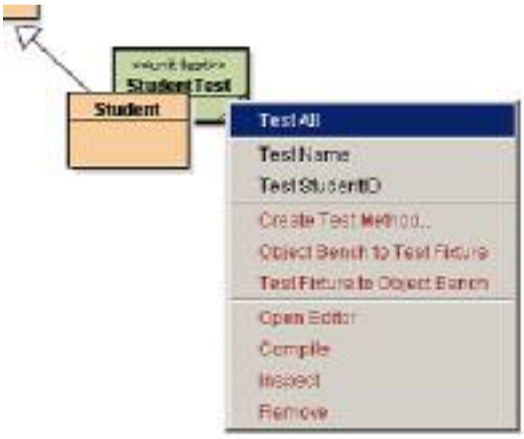

Figura 5: Menú de la clase de prueba con dos métodos de prueba definidos.

Seleccionando una prueba del menú ejecuta esa prueba individualmente. Seleccionando la opción Test All (Probar Todo) del menú de la clase de prueba, ejecuta todas las pruebas definidas en esa clase.

Cuando una prueba es corrida individualmente, una de dos cosas ocurren: si la prueba es satisfactoria (la condición establecida en la aserción se cumple) una pequeña nota indicando el éxito es mostrada en la barra de estado de la parte inferior de la ventana del proyecto. Si la prueba falla (una aserción falla o cualquier otro problema ocurre) se exhibe una ventana de resultado de la prueba presentando detalles acerca de la prueba ejecutada (figura 6).

Si son ejecutadas todas las pruebas, siempre se muestra la ventana de resultado de la prueba para mostrar el resultado de las pruebas.

Usted puede también utilizar el botón Run Tests (Ejecutar Pruebas) sobre el indicador de grabación de prueba en la ventana principal. Activando este botón correrá todas las pruebas en todas las clases de prueba en el paquete. Esta es la forma normal de ejecutar un juego completo de prueba para el paquete.

6 Interpretando resultados de pruebas

Resumen: Las ventanas de resultado de las pruebas muestran un resumen de las pruebas corridas y pueden mostrar detalles de fallas.

Figura 6: La ventana de resultado de la prueba

Cuando se han ejecutado las pruebas, la ventana de resultado muestra un resumen de los resultados (figura 6). El panel superior de la ventana muestra una lista de todas las pruebas ejecutadas, marcada con un icono indicando su éxito o falla. Un tilde verde indica una prueba satisfactoria, una cruz gris indica una falla en la prueba y una cruz roja marca un error.

El número de pruebas corridas, errores y fallas es también indicado en la sección media de la ventana.

Una prueba tiene una falla (cruz gris) si una de sus aserciones no se cumple. Por ejemplo, la aserción de la prueba puede especificar que el resultado de un método en particular no debería ser nulo, pero en este caso lo es.

Una prueba tiene un error si su ejecución dio lugar a cualquier otra clase de error, tal como una excepción inesperada que es lanzada.

Para cualquier prueba fracasada, los detalles de su falla pueden ser mostrados seleccionando la prueba en la lista. El panel inferior en la ventana exhibe la información del detalle sobre esta falla o error.

La barra en el medio de la ventana de prueba es el resumen principal de la ejecución de la prueba: si aparece verde, todo está bien – todas las pruebas han sido satisfactorias. Si es roja, hay un problema – al menos uno ha fallado.

Nótese que en MacOS esta barra no cambia de color.

7 ¿Qué es un fixture?

Resumen: Un fixture de prueba es un conjunto de objetos preparado utilizado como punto de partida para las pruebas

A veces, una prueba necesita algunos objetos creados antes de que la prueba real pueda comenzar. Por ejemplo, para varias de las pruebas que querríamos para la clase `Database` en el proyecto 'people', necesitamos un objeto `Database`, un objeto `Student` y un objeto `Staff`. Además, querríamos un estado específico (tal como un nombre y una edad) para el estudiante y miembro del staff.

Podríamos empezar cada prueba individual creando los objetos necesarios y poniéndolos en el estado apropiado para hacer esta prueba. Pero a medida que las pruebas se hacen más sofisticadas, esto puede volverse tedioso, y podemos usar un mecanismo mejor para evitar este trabajo extra.

Podemos crear un estado en el banco de objetos (un conjunto de objetos, cada uno en un cierto estado) los cuales queremos usar como punto de partida para todas las pruebas en una clase de prueba específica. Este conjunto inicial de objetos es llamado *fixture*.

Los fixtures pueden ser definidos, y serán automáticamente creados en el inicio de cada prueba de la misma clase de prueba, reduciendo así el trabajo extra de cada prueba individual.

8 Creando y usando fixtures de prueba

Resumen: Para crear un fixture para una clase de prueba, cree los objetos deseados en el banco de objetos y entonces seleccione Object Bench To Test Fixture (Banco de Objetos al Fixture de Prueba) *del menú de la clase de prueba.*

Empezamos creando un fixture de prueba simplemente creando los objetos que necesitamos, y haciendo llamadas a métodos en los objetos para llevarlos al estado deseado.

Por ejemplo, para probar nuestra clase base de datos en el proyecto personas, nosotros querríamos tener un objeto `Database`, un objeto `Student` con el nombre "Fred", el cual es insertado en la base de datos, y un objeto `Staff` con el nombre "Jane", quien no está en la base de datos. Podemos empezar simplemente creando los objetos y haciendo las llamadas necesarias para insertar a Fred en la base de datos.

Una vez que el estado en el banco de objetos es el que queremos para empezar nuestras pruebas, podemos seleccionar la función Object Bench To Test Fixture (Banco de Objetos al Fixture de Prueba) de la clase `DatabaseTest`.

Seleccionando esta función crearemos un fixture de prueba para la clase, mientras que quita todos los objetos del banco.

Cuando una clase tiene un fixture, éste es recreado al comienzo de cada prueba. Por ejemplo, si ahora creamos una nueva prueba para la clase `Database` (seleccionando Create Test Method (Crear Método de Prueba) de su clase de prueba) la situación definida en el fixture será automáticamente restaurada. Los objetos del fixture aparecerán un su estado definido en el banco al inicio de la grabación de la prueba.

El estado del fixture puede ser también recreado explícitamente en el banco seleccionando Test Fixture To Object Bench (Fixture de Prueba al Banco de Objetos) del menú de la clase de prueba. Esto puede ser útil en los casos donde queremos extender el fixture más tarde, porque nuevos métodos de prueba necesitan objetos adicionales del fixture.

En ese caso, podríamos usar Test Fixture To Object Bench (Fixture de Prueba al Banco de Objetos) para regenerar el estado del fixture, y entonces hacer manualmente las adiciones necesarias al estado del fixture. Una vez finalizada, podemos seleccionar nuevamente Object Bench To Test Fixture (Banco de Objetos al Fixture de Prueba) para almacenar el fixture. El fixture anterior será reemplazado.

9 Escribiendo métodos de prueba a mano

Resumen: Los métodos de prueba pueden escribirse directamente en el código fuente de la clase de prueba.

Generar métodos de prueba y fixtures grabando nuestra interacción y el estado del banco de objetos es sólo una opción de generación de pruebas de unidad. La otra opción es escribir estos métodos de prueba a mano.

Las clases de prueba son clases Java como cualquier otra clase en el proyecto, y pueden ser tratadas de la misma manera. En particular, podemos abrir el editor para ver el código fuente, editar el código, compilar y correrlo.

En el uso tradicional (no BlueJ) de JUnit, esta es la manera estándar de crear métodos de prueba, y BlueJ permite el mismo estilo de trabajo. Grabar pruebas interactivamente es un agregado a escribir pruebas a mano, no un reemplazo.

Para aquellos no familiarizados con JUnit, una buena manera de comenzar es generar un fixture de prueba y unos pocos métodos interactivamente, y entonces examinar el código fuente de la clase de prueba. Podemos observar que cada clase de prueba tiene un método llamado `setUp()` que es usado para preparar el fixture de prueba. También tiene un método adicional por cada prueba.

Está perfectamente bien editar a mano métodos de prueba existentes para modificar su comportamiento, o agregar métodos de prueba escritos a mano completamente nuevos. Recuerde que el nombre de un método de prueba debe comenzar con el prefijo “test” para que sea reconocido como método de prueba.

10 Escribiendo primero las pruebas

Resumen: Para crear pruebas antes de la implementación, éstas pueden ser escritas a mano, o pueden utilizarse métodos vacíos.

La metodología de Programación eXtrema [ref] sugiere que las pruebas deberían ser escritas *antes* de la implementación de cualquier método. Utilizando las pruebas de integración de BlueJ, esto puede ser hecho en dos formas diferentes.

En primer lugar, las pruebas pueden ser escritas a mano, como se explicó en la sección previa. La escritura de los métodos funciona entonces de la misma manera que en implementaciones no-BlueJ de JUnit.

En segundo lugar, podemos crear métodos vacíos en la clase de referencia, retornando valores simulados para métodos con tipo de retorno no-void. Después de hacer esto, podemos crear las pruebas utilizando la facilidad de grabado interactivo, y escribimos aserciones de acuerdo con nuestras expectativas de la implementación finalizada.

11 Pruebas multi-clase

Resumen: La función New Class... (Nueva clase...) *con el tipo de clase* Unit Test (Prueba de unidad) *puede ser usada para crear clases de prueba sueltas.*

Los ejemplos dados utilizan clases de prueba que están unidas a una clase de referencia. Esta unión no evita que las clases de prueba hagan uso de otros tipos de clase en sus pruebas, sino que sugiere una conexión lógica de la clase de prueba a su clase de referencia.

A veces las clases de prueba son escritas de forma que prueban varias clases en combinación. Estas clases no pertenecen lógicamente a una sola clase. Para documentar esto, deberían no estar directamente unidas a una única clase.

Podemos crear clases de prueba sueltas usando la función New Class... (Nueva clase...) normal, y luego seleccionar Unit Test (Prueba de unidad) como el tipo de clase en el diálogo de la nueva clase.

Las clases de prueba sueltas pueden ser usadas de la misma manera que otras clases de prueba. Podemos crear fixtures de prueba, hacer métodos de prueba y ejecutarlos.

12 Sólo los resúmenes

1. BlueJ proporciona funcionalidad de prueba de regresión integrando JUnit
2. Las herramientas de prueba se pueden visualizar habilitando la opción en preferencias.
3. Cree una clase de prueba seleccionando Create Test Class (Crear Clase de Prueba) del menú contextual de la clase.
4. Cree un método de prueba seleccionando Create Test Method... (Crear Método de Prueba...) del menú de la clase de prueba.
5. Ejecute todas las pruebas haciendo click en el botón Run Tests (Ejecutar pruebas). Ejecute pruebas individuales seleccionándolas del menú de la clase de prueba.
6. Las ventanas de resultado de las pruebas muestran un resumen de las pruebas corridas y pueden mostrar detalles de fallas.
7. Un fixture de prueba es un conjunto de objetos preparado utilizado como punto de partida para las pruebas
8. Para crear un fixture para una clase de prueba, cree los objetos deseados en el banco de objetos y entonces seleccione Object Bench To Test Fixture (Banco de Objetos al Fixture de Prueba) del menú de la clase de prueba.
9. Los métodos de prueba pueden escribirse directamente en el código fuente de la clase de prueba.
10. Para crear pruebas antes de la implementación, éstas pueden ser escritas a mano, o pueden utilizarse métodos vacíos.
11. La función New Class... (Nueva clase...) con el tipo de clase Unit Test (Prueba de unidad) puede ser usada para crear clases de prueba sueltas.

