Programación orientada a objetos

Capítulo 6

Objetos con buen comportamiento

Tema 6. Objetos con buen comportamiento. Semana 6	
1- Prueba y depuración 2- Pruebas de unidad en BlueJ 3- Pruebas automatizadas 4- Modularización e interfaces 5- Comentarios y estilo 6- Seguimiento manual 7- Elegir una estrategia de prueba	1- Estudiar el capítulo 6 del libro base para la Unidad Didáctica I 2- Leer los Apéndices G y H del libro base para la Unidad Didáctica I 3- Definir la estrategia de prueba a utilizar en la práctica

Prueba y depuración

La **prueba** es la actividad cuyo objetivo es determinar si una pieza de código (un método, una clase o un programa) produce el comportamiento pretendido.

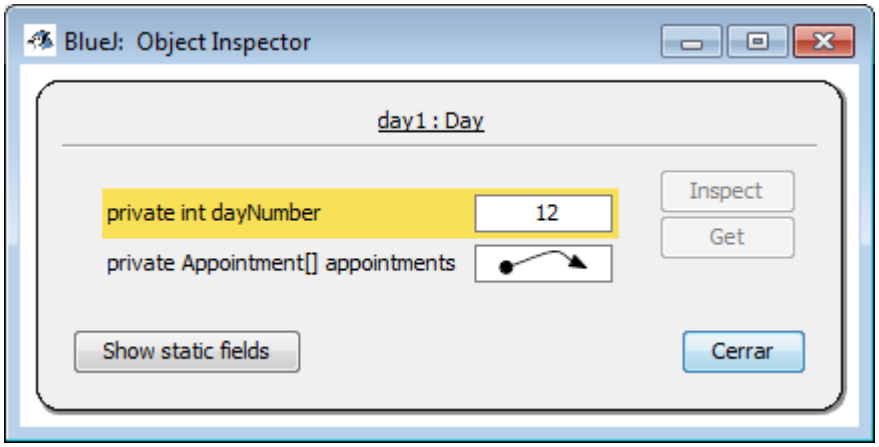
La *prueba* es una actividad dedicada a determinar si un segmento de código contiene errores. No es fácil construir una buena prueba, hay mucho para pensar cuando se prueba un programa.

La **depuración** es el intento de apuntar con precisión y corregir un error en el código.

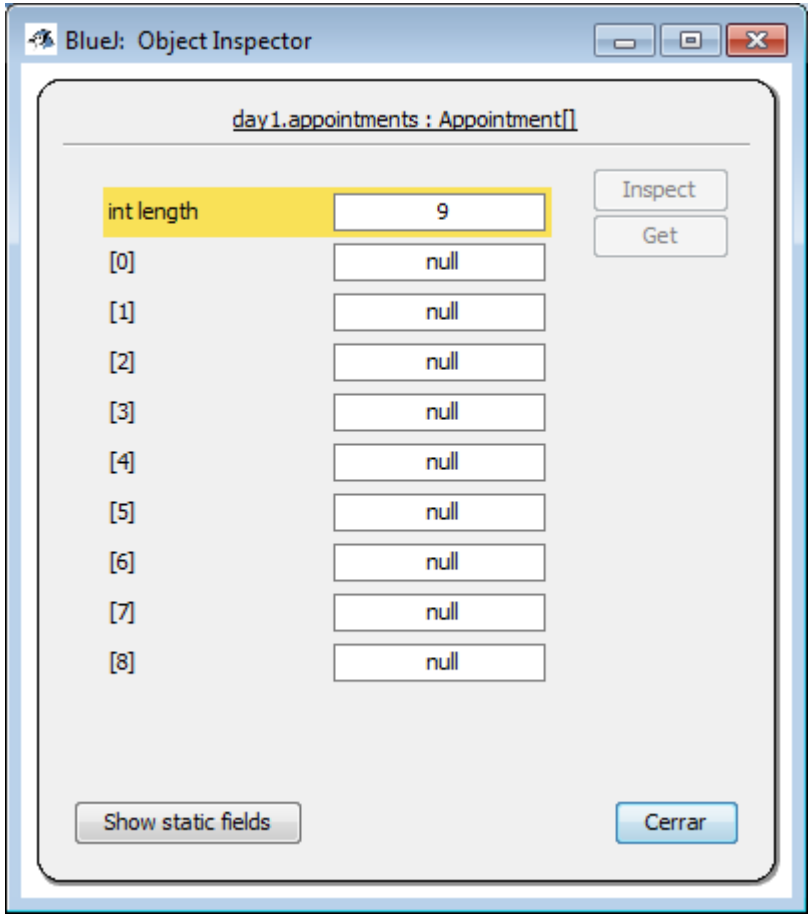
La *depuración* viene a continuación de la prueba. Si las pruebas demostraron que se presentó un error, usamos técnicas de depuración para encontrar exactamente dónde está ese error y corregirlo. Puede haber una cantidad significativa de trabajo entre saber que existe un error y encontrar su causa y solucionarlo.

Pruebas de unidad en BlueJ

- Pruebas de partes individuales de una aplicación
  - Un método
  - Una clase
- Nunca es demasiado pronto para hacer pruebas
- Realizar pruebas sobre el proyecto Dairy-prototype
- Usar Inspectores
  - Comprobar los límites (máximo y mínimos)



Inspect de la tabla



Pruebas positivas y pruebas negativas

Una prueba positiva es la prueba de aquellos casos que esperamos que resulten exitosos.

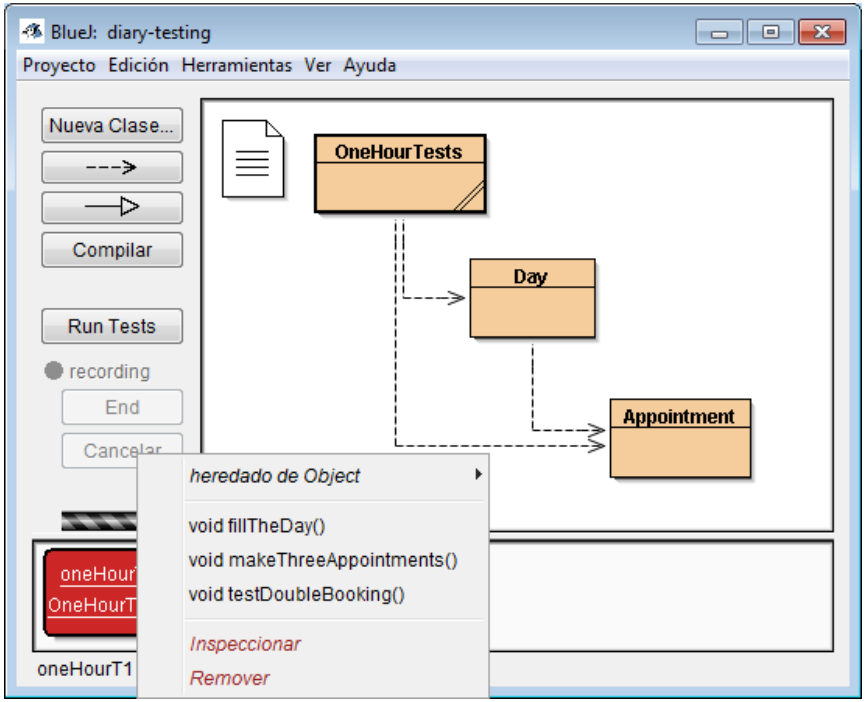
ejemplo, anotar una cita de una hora de duración en el medio de un día que aún está vacío es una prueba positiva. Cuando probamos con casos positivos nos tenemos que convencer de que el código realmente funciona como esperábamos.

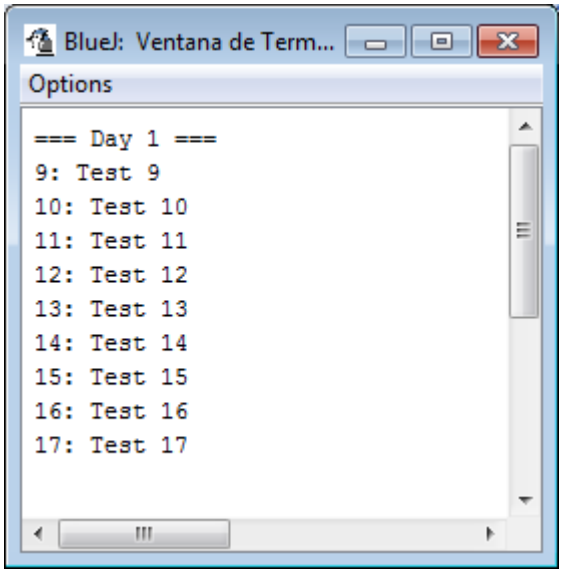
Una prueba negativa es la prueba de aquellos casos que esperamos que fallen.

Una *prueba negativa* es la prueba de aquellos casos que esperamos que fallen. Anotar dos citas en una misma hora o registrar una cita fuera de los límites válidos del día son ambos ejemplos de pruebas negativas. Cuando probamos con casos negativos esperamos que el programa maneje este error de cierta manera especificada y controlada.

Pruebas automatizadas

- Permite repetir pruebas de modo automático
- Prueba de regresión:
  - Consisten en ejecutar nuevamente las pruebas pasadas previamente para asegurarse de que las nueva versión aún las pasa
  - Escribir un programa que actúa como equipo de prueba o batería de prueba

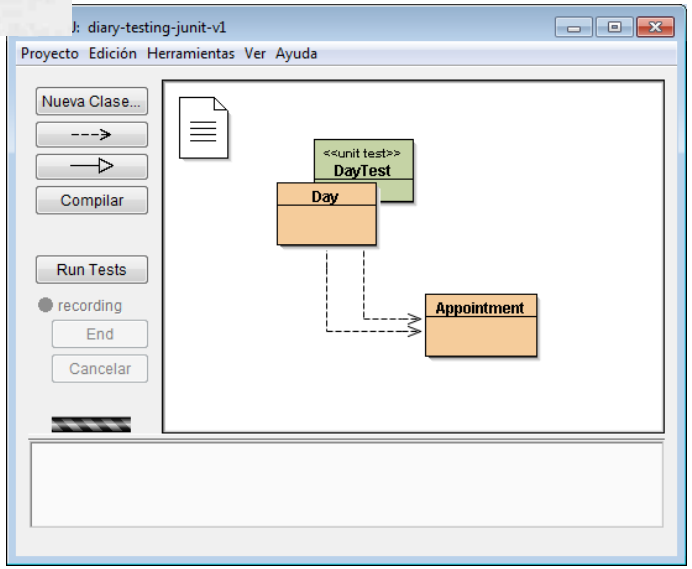




Control automático de las pruebas (JUnit)

- Unidad de prueba (unit test)
  - Son una característica del BlueJ
  - Diseñadas para pruebas de regresión
  - Se crean con “Create test class” en el menú contextual de botón derecho sobre una clase
  - En el CD:
    - Testing-tutorial.pdf

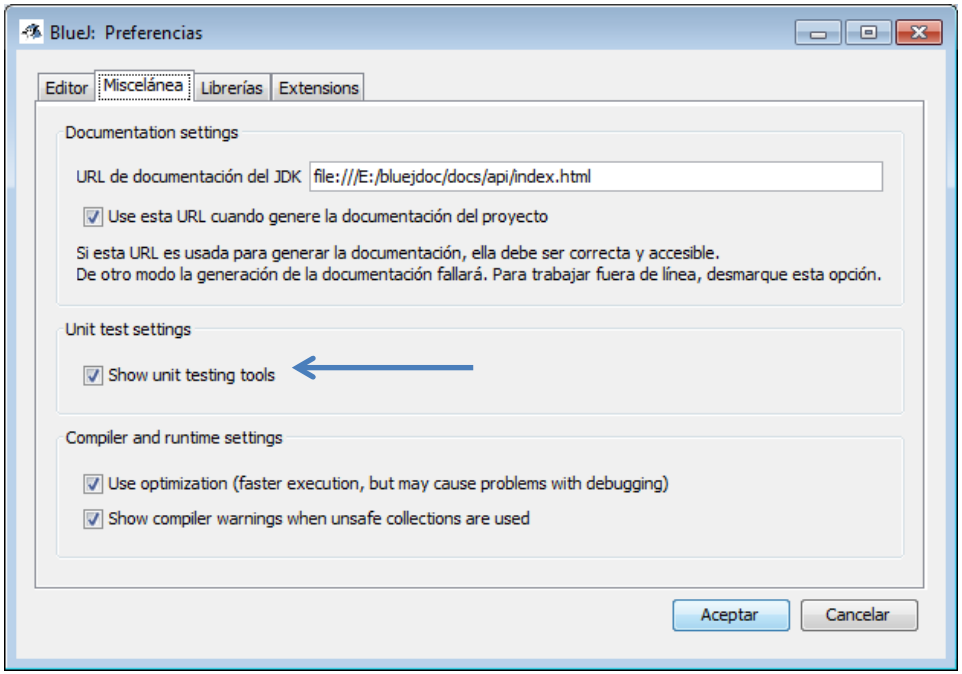
JUnit, www.junit.org JUnit es un popular marco de trabajo (*framework*) para implementar en Java pruebas de unidad organizadas y pruebas de regresión. Está disponible independientemente del entorno específico de desarrollo que se use, así como también está integrado a muchos entornos. JUnit fue desarrollado por Erich Gamma y Kent Beck. Puede encontrar el software y gran cantidad de información sobre él en `http://www.junit.org`.



Herramientas JUnit de pruebas unitarias

Habilitar la funcionalidad de pruebas unitarias

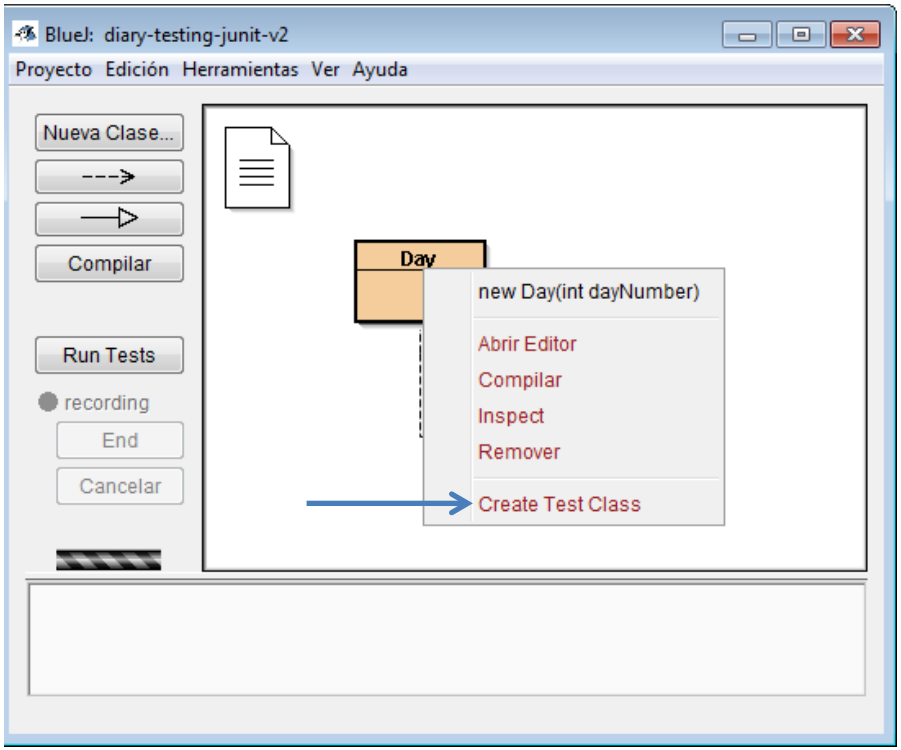
Para habilitar la funcionalidad de pruebas unitarias de BlueJ es necesario asegurarse de que esté marcada la opción *Show unit testing tools* en el menú *Tools-Preferences-Miscellaneous*. Una vez marcada, la ventana principal de BlueJ contendrá algunos botones adicionales que se activan cuando se abre un proyecto.



Crear una clase de prueba

Se crea una clase de prueba haciendo clic con el botón derecho del *ratón* sobre una clase en el diagrama de clases y seleccionando la opción *Create Test Class*. El nombre de una clase de prueba se determina automáticamente agregando la palabra «Test» a modo de sufijo al nombre de la clase asociada. Alternativamente, puede crearse una clase de prueba seleccionando el botón *New Class...* y eligiendo *Unit Test* como el tipo de la clase. En este caso, la elección del nombre es totalmente libre.

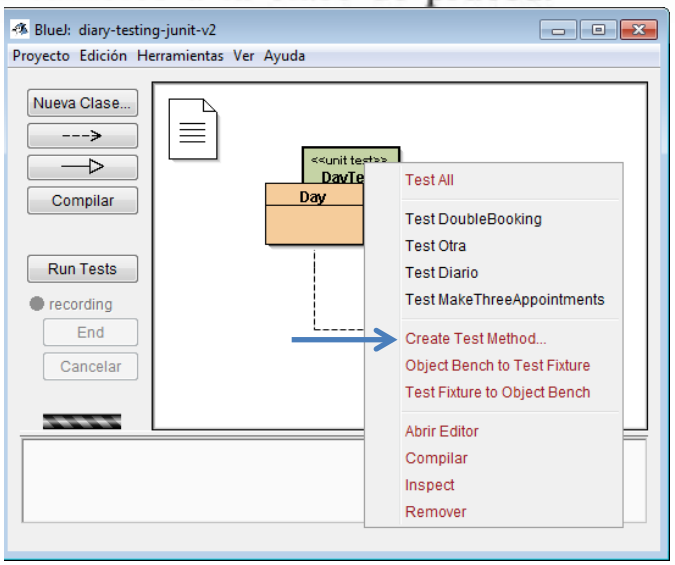
Las clases de prueba se denotan con `<<unit test>>` en el diagrama de clases y tienen un color diferente del color de las clases ordinarias.

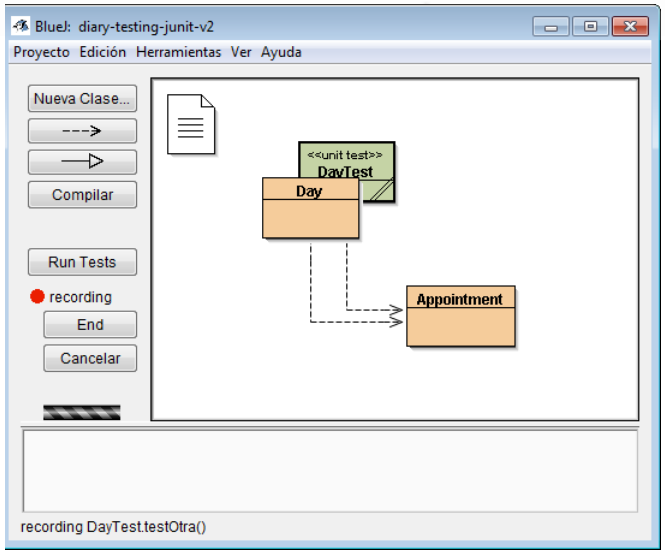


Crear un método de prueba

Los métodos de prueba se pueden crear interactivamente. Se puede grabar la secuencia de interacciones del usuario con el diagrama de clases y con el banco de objetos y luego capturarla como una secuencia de sentencias y de declaraciones Java en un método de la clase de prueba. Se comienza la grabación seleccionando la opción *Create Test Method* del menú contextual asociado con una clase de prueba. BlueJ solicitará el nombre del nuevo método. Si el nombre no comienza con la palabra `test` entonces se agregará como un prefijo del nombre del método. El símbolo de grabación a la izquierda del diagrama de clase se pondrá de color rojo y se vuelven disponibles los botones *End* y *Cancel*.

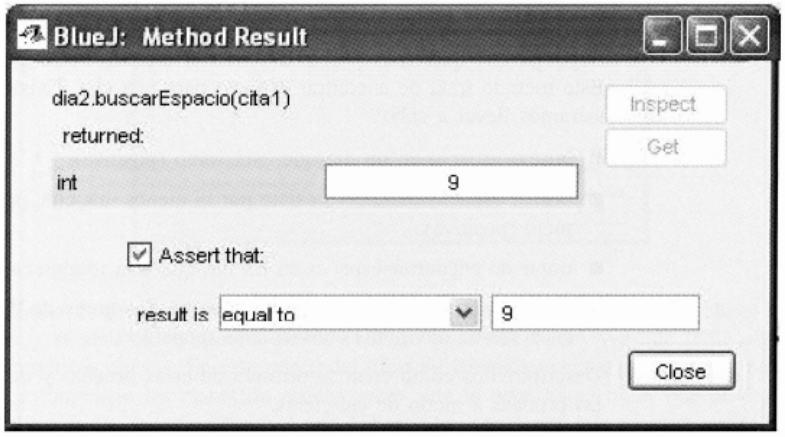
Una vez que comenzó la grabación, cualquier creación de objeto o llamadas a métodos formarán parte del código del método que se está creando. Seleccione *End* para completar la grabación y capturar la prueba o *Cancel* para descartar la grabación y dejar sin cambios a la clase de prueba.





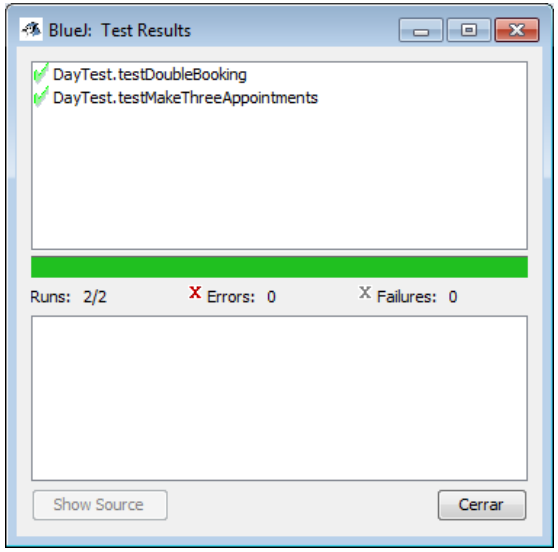
Pruebas con aserciones

Mientras se graba un método de prueba, cualquier llamada a método que retorne un resultado abrirá una ventana del tipo *Method Result* que ofrece la oportunidad de evaluar el valor del resultado marcando la opción *Assert that*. El menú desplegable que aparece contiene un conjunto de aserciones posibles para el valor del resultado. Si se estableció una aserción, será codificada como una llamada a método en el método de prueba que dará un `AssertionError` en el caso en que la prueba falle.



Ejecutar pruebas

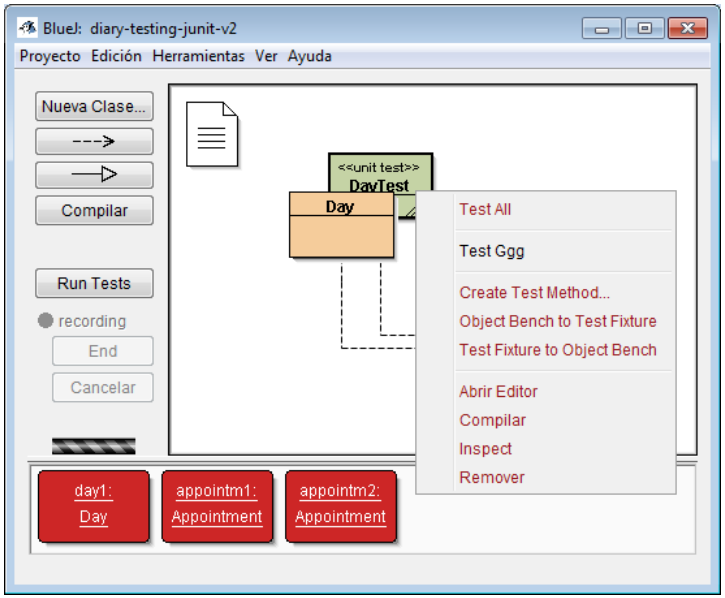
Los métodos se pueden ejecutar individualmente seleccionándolos del menú contextual asociado a la clase de prueba. Si una prueba resulta exitosa, se indicará mediante un mensaje en la línea de estado de la ventana principal. Si una prueba fracasa aparecerá la ventana *Test Results*. Al seleccionar *Test All* del menú contextual de la clase de prueba se ejecutarán todas las pruebas de una sola clase. En la ventana *Test Results* se detallará el éxito o el fracaso de cada método.



Conjunto de Objeto de prueba (Fixture: juego de pruebas)

- Crear objetos para las pruebas
- Opciones del menú contextual de “unit test”
  - Object Bench to Test Fixture
    - Los objetos que creamos en el “banco de objetos” se incorporan al código en el método “Setup”
    - El método “setup” se invoca automáticamente antes de llamar a cada método de prueba, por lo que todos los objetos del juego de pruebas estarán disponibles para todas las pruebas
  - Test Fixture to Object Bench
    - Para agregar objetos al Fixture, primero pasamos los objetos que ya están en Fixture al banco de objetos (“Test Fixture to Object Bench”)
    - Añadimos el objeto que deseemos
    - Y pulsamos “Object Bench to Test Fixture”. Y atenemos el nuevo Fixture, con los objetos añadidos

A *fixture* es un conjunto de objetos con un estado definido que sirve como base para las pruebas de unidades.



Seguimientos

- Seguimiento manual
- Seguimiento verbal
- Sentencias de impresión
- Depuradores

Método llamado	valorEnVisor	operandoIzquierdo	operadorAnterior
estado inicial	0	0	' '
limpiar	0	0	' '
numeroPresionado(3)	3	0	' '

Términos introducidos en este capítulo

error de sintaxis, error de lógica, prueba, depuración, prueba de unidad, prueba positiva, prueba negativa, prueba de regresión, seguimiento manual, secuencia de llamadas

Resumen de conceptos

- **prueba** La prueba es la actividad de descubrir si una pieza de código (un método, una clase o un programa) produce el comportamiento pretendido.
- **depuración** La depuración es el intento de apuntar con precisión y corregir el código de un error.
- **prueba positiva** Una prueba positiva es la prueba de los casos que se espera que resulten exitosos.
- **prueba negativa** Una prueba negativa es la prueba de los casos en que se espera que falle.
- **aserción** Una aserción es una expresión que establece una condición que esperamos que resulte verdadera. Si la condición es falsa, decimos que falló la aserción. Esto indica un error en nuestro programa.
- **fixture** Un fixture es un conjunto de objetos en un estado definido que sirven como una base para las pruebas de unidad.
- **seguimiento** Un seguimiento es la actividad de trabajar a través de un segmento de código línea por línea, mientras se observan cambios de estado y otros comportamientos de la aplicación.