Aprendizaje Java acelerado por casos de prueba JUnit

Adolfo Di Mare

Escuela de Ciencias de la Computación e Informática
Universidad de Costa Rica
adolfo.dimare@ecci.ucr.ac.cr

Resumen La herramienta JUnit para prueba unitaria de módulos se puede usar para lograr que el aprendiz de programación Java no necesite conocer con gran detalle la sintaxis del lenguaje, permitiéndole que desde el principio se concentre en construir algoritmos, lo que facilita y acelera la primera etapa del aprendizaje de la programación.
Palabras clave Prueba unitaria de programas, programación por ejemplos, introducción de técnicas de programación, software.

1. Introducción

Java ha ganado popularidad como el primer lenguaje de programación por muchas razones [1]: es un lenguaje completo que además es muy utilizado en la industria [2]. Sin embargo, para programar en Java es necesario conocer antes su sintaxis con buen detalle y además es necesario utilizar un entorno de desarrollo adecuado para la implementación de programas (IDE: "*Integrated Development Environment*"). Muchos docentes han tratado de evadir la complejidad sintáctica de Java introducciendo objetos antes de hablar de algoritmos [3]. El entorno de desarrollo BlueJ para Java es un es ejemplo de cómo la automatización se usa para producir el esqueleto de las clases a partir de objetos que se visualizan en un ambiente gráfico [4].

Discutir si se debe enseñar algoritmos primero vs. objetos primero no es el objetivo de este trabajo, pues se parte de la suposición de que el docente cuenta con herramientas que le permitan mostrarle a sus estudiantes hacia adónde va el aprendizaje, de manera que ellos puedan construir su propio modelo intelectual para lograr aprender, como lo expone BenAri [5] cuando justifica la herramienta Jeliot para visualizar la ejecución de algoritmos Java [6]. Más bien aquí se muestra que JUnit sirve para que el docente pueda saltar sobre una buena parte de la sintaxis Java para sumergir a sus estudiantes directamente en la construcción de algoritmos sencillos muy rápidamente. El entorno de desarrollo DrJava es el prefirido del autor, pues es liviano y se puede ejecutar desde la llave maya, lo que les permite a los estudiantes usar cualquier equipo para hacer sus prácticas [7].

Si el profesor puede mostrar rápido los componentes más importantes de la programación Java tendrá más tiempo para examinar conceptos avanzados,

como parametrización o concurrencia; de lo contrario debe invertir una buena parte de su curso lidiando con la sintaxis de Java.

Aquí proponemos el uso de datos de prueba JUnit para alcanzar el siguiente objetivo del primer curso de programación: "Proveer al estudiante la formación básica en programación y construcción de algoritmos para su adecuado desempeño en los cursos subsiguientes de la carrera, fomentándole sus habilidades generales para la resolución de problemas".

2. JUnit en 3 párrafos

Los adeptos a la llamada "Programación Extrema" han construido muchas herramientas que les permiten probar primero, luego codificar y por último depurar [8]. No hay que discutir si este paradigma es apropiado o no para utilizar sus herramientas para la enseñanza. JUnit es un conjunto de clases Java orientado a ejercitar todos los componentes de un programa para determinar si funcionan correctamente [9]. JUnit permite hacer la prueba unitaria de módulos ya sean rutinas simples, métodos complejos o programas completos.

En el corazón de JUnit está el verbo "assertTrue(cond)" que evalúa su argumento y acumula las ocasiones en que resulta falso; cada condición "(cond)" representa un dato de prueba o un caso de prueba. Por ejemplo, si el sumador "Bib.sume(3,5)" no es 8, esta invocación graba el hecho que puede ser reportado luego por JUnit:

```
assertTrue( 8 == Bib.sume(3,5) ); // JUnit registra si falla
```

La arquitectura de JUnit es particular para el lenguaje Java, pero independientemente de cómo está hecha funciona bien y es muy utilizada tanto en la academia como en al industria. La prueba unitaria se puede utilizar para complementar la especificación de módulos, como se muestra en [10] en el contexto de C++.

3. Un ejemplo JUnit sencillo

Para que el estudiante aplique ciclos y sencuenciación el profesor muchas veces le pide que sume los valores almacenados en un vector. Así surge el método entero "sumador(int[])" que retorna la suma de los valores del vector.

```
public static int sumador( int VEC[] ) {
    if ( VEC.equals(null) ) { // VEC[] no existe todavia
        return 0;
    }
    else if ( VEC.length==0 ) {
        return 0;
    }
    int suma = 0; // acumulador
    {
         /*****************************\
         *                             *
         *   RELLENE CON SU ALGORITMO  *
         *                             *
         \*****************************/
    }
    return suma;
}


int suma = 0; // acumulador
{   // Solucion
    /*****************************\
    *                             *
    *   RELLENE CON SU ALGORITMO  *
    *                             *
    \*****************************/
    final int N = VEC.length;
    for ( int i=0; i<N; ++i ) {
        suma = suma + VEC[i];
    }
}
```

Figura 1: Rellene con su algoritmo

En la Figura 1 se muestra la implementación del método “sumador()”: falta un bloque de código que el alumno debe rellenar con la siguiente implementación que se muestra en la parte inferior (o con una equivalente). Este ejercicio le permite al profesor concentrar la atención del estudiante en el algoritmo de suma, sin examinar el resto del código. Por ejemplo, pueden pasar varias sesiones de trabajo antes de que el profesor explique por qué sumador() es un método “estático” o que muestre que la comparación con el valor “null” usando el método “equal()” mejora la implementación porque generaliza los valores a los que se puede aplicar sumador().

```
/** test -> {@code sumador()}. */
public void test_sumador() {
    { int V[] = { 1,2,3,4,5 }; assertTrue( sumador(V) == 15 ); }
    { int V[] = { 2,2,2,2,2 }; assertTrue( sumador(V) == 10 ); }
    { int V[] = { 8,0,2,1,9 }; assertTrue( sumador(V) == 20 ); }
    { int V[] = { 4,3,2,1,0 }; assertTrue( sumador(V) == 10 ); }
    { int V[] = { 0,1,2,3,4 }; assertTrue( sumador(V) == 10 ); }
}
```

Figura 2: Pruebas para sumador()

En la Figura 2 está la implementación del método JUnit que hace la prueba. Esta prueba fue preparada de antemano por el profesor y es el trabajo del alumno lograr que su programa funcione. Es sencillo saber si ya el alumno terminó su práctica, pues la ejecución que se produce al pulsar el botón [Test] en DrJava resulta en una barra verde, de lo contrario el IDE retorna un error marcando el renglón con el color amarillo [7].

```
import junit.framework.*;

/** Datos de prueba para {@code sumador(int[])}. */
public class TestSumador extends TestCase {
    /** Suma de los valores de {@code VEC(int[])}. */
    public static int sumador( int VEC[] ) {
        // ...        {
        /*****************************\
        *                             *
        *   RELLENE CON SU ALGORITMO  *
        *                             *
        \*****************************/
        }
    }
    /** test -> {@code sumador()}. */
    public void test_sumador() {
        // ...
    }
}
```

Figura 3: Estructura del programa JUnit completo

En la Figura 3 se muestra la estructura del programa JUnit completo. El método "sumador()" es el que debe completar el alumno, y "test_sumador()" es el método que contiene los datos de prueba; es importante que el nombre del método de prueba comience con "test" porque así JUnit lo puede encontrar y ejecutar dinámicamente, en tiempo de ejecución. La parte marcada "** RELLENE CON SU ALGORITMO **" es la que debe completar el estudiante.

Muchas veces conviene que el profesor resuelva varios problemas similares a los que debe enfrentar el alumno. Pese a que el profesor debe trabajar un

poquito más al impartir lecciones y prácticas, pues debe implementar tanto los ejercicios como sus soluciones antes de presentarlas a sus estudiantes, a fin de cuentas es posible lograr que los estudiantes trabajen pronto en programas que no se limitan a una docena de líneas de código, como por ejemplo el juego de "Toques y Famas" de esta tarea programada:

```
http://www.di-mare.com/adolfo/cursos/2009-2/pi-ta-3.htm
% http://www.di-mare.com/adolfo/cursos/2009-2/pi-ta-4.htm
```

Si el algoritmo escrito por el estudiante es incorrecto, la ejecución JUnit del programa de prueba produce un error que aparece en amarillo en DrJava. Lo mismo ocurre si el programa tiene un error de sintaxis.

```
{    for ( int gM=1; gM<=5; ++gM ) {
suma = suma + VEC[i-1];}}
```

Figura 4: Mala indentación

La dificultad de aprender Java disminuye porque el IDE DrJava le presenta al estudiante los errores de lógica y de sintaxis de la misma manera: el renglón en donde está el error queda pintado de color amarillo. Sin embargo, los estudiantes pueden cometer errores muy creativos, como los que se muestran en Figura 4, en donde el espaciado, la indentación y la elección de lo identificadores es incorrecta. Además, también puede ocurrir que el profesor olvide cubrir todos los casos pertinentes, como ocurre en este ejemplo en que el alumno ha evitado usar la propiedad VEC.length sustituyéndolo por un valor fijo que no produce errores, pues todos los datos de prueba usan un vector que almacena 5 valores. Por eso es importante revisar el algoritmo escrito por cada estudiante, para determinar si es una solución muy particular y para verificar que el formato de la codificación es correcto [11].

Si por error un alumno borra algún corchete "{ }" o modifica el código que está fuera del ámbito marcado "** RELLENE CON SU ALGORITMO **" deberá comenzar desde el principio, cargando una copia limpia del código, pues de lo contrario le será prácticamente lograr que el programa compile correctamente. También ocurre muchas veces que los novatos usan lógica estrambótica para su solución (pues todavía no saben programar).

4. Conexión JUnit <==> DrJava

Cada profesor enseña Java usando el IDE que más le place. Aquí recomendamos DrJava porque incorpora la biblioteca JUnit, lo que permite usarla directamente sin necesidad de siquiera mencionarla. Por eso basta indicarle al estudiante que ejecute su programa pulsando el botón [Test] de DrJava, en lugar del botón [Run] que se usa para ejecutar los demás programas. El novato no conoce las diferencia entre casos de prueba, clases y algoritmos, pero cuando pulsa [Test] lo que obtiene es un "error amarillo" que le indica que su trabajo todavía no está completo. Este sincretismo tecnológico evita que el estudiante tenga dudas

sobre el comportamiento de la computadora, pues cuando ya terminó su trabajo la barrita del [Test] se pone verde y antes de eso todos los errores, sean estos errores de sintaxis o de lógica, se muestran como un "error amarillo".

Por supuesto, otros entornos de desarrollo permiten el uso de pruebas unitarias JUnit. En opinión del autor, además de que incorpora el JUnit como uno de sus módulos internos, la ventaja relativa del DrJava es que se puede ejecutar directamente, sin necesidad de instalación (siempre y cuando el computador ya tenga instalado el ambiente Java JDK). Esta facilidad de uso le permite a cada estudiante llevar prácticamente todo el software en su llave maya, para trabajar en cualquier computador que encuentre disponible. El dueto JUnit ˜ DrJava es tan sencillo de utilizar que al profesor le bastan minutos para explicarle a sus estudiantes qué tienen que hacer, en lugar de usar horas o días enteros. Por eso se acelera el aprendizaje del lenguaje.

5. Resultados pedagógicos

Como un estudio empírico para validar la propuesta pedagógica presentada en este artículo, al finalizar el curso en que se introdujo esta técnica, mediante un pequeño cuestionario se le pidió a los estudiantes opinar sobre el uso de ejemplos y prácticas JUnit. Más de la mitad de los estudiantes no recordaban ya la diferencia entre un programa [Test] y uno [Run], que es la forma en que distinguían los programas JUnit de los otros, pero más de dos tercios comentaron que les pareció extraño que el libro de texto no hablara de esos ejemplos [12], [13]. Parece que, como JUnit se usó al principio del curso, una vez que la sintaxis Java dejó de ser una asunto relevante porque el problema principal pasó a ser implementar el algoritmo adecuado para la solución de un problema, la mayoría de los estudiantes simplemente dejaron de lado el botón [Test] para concentrarse en terminar sus proyectos programados.

En estos días de acceso Internet prácticamente ilimitado, algunos alumnos buscan soluciones a sus tareas y proyectos en los sitios pirata de la red. Esto puede limitar la reutilización de ejemplos porque los malos estudiantes podrían encontrar las soluciones a las prácticas en la red, lo que efectivamente les impediría aprovechar el aprendizaje de estos pequeños ejercicios de algoritmos apoyados por JUnit. Es importante que cada docente le recuerde a sus alumnos que quien copie los ejercicios y quede rezagado porque no aprendió, posiblemente fracasará en el curso pues luego no tendrá la oportunidad de recuperar el tiempo perdido: mientras los demás estarán concentrados en implementar programas ellos todavía deberán lidiar con las complejidades de la sintaxis Java.

El uso tempranero de JUnit facilita abordar rápidamente la construcción de programas usando pruebas unitarias (no en todas en todas las carreras de ingeniería existe un segundo curso de programación).

6. Conclusión

El uso de pruebas unitarias JUnit le permite al profesor limitar el contexto de trabajo Java que debe enfrentar el estudiante. Al reducir la complejidad sintáctica de los ejemplos el aprendizaje del lenguaje se acelera. El autor ha usado JUnit porque el primer curso de programacin es Java, pero quienes utilicen C++ pueden usar otras bibliotecas de prueba unitaria como BUnit [10].

Referencias

1. King, K. N.: "The Case for Java as a First Language", Proceedings of the 35th Annual ACM Southeast Conference, pp. 124131, Abril 1997.
2. TIOBE Programming Community Index, 2010. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
3. Griffin, Jean & Fickett, Mark & Powell, Rita: "Objects-First, Algorithms-Early with BotWorld (white paper)", University of Pennsylvania, 2005. http://www.seas.upenn.edu/~botworld/files/botworld-012009.pdf
4. Barnes, David J. & Klling Michael, "Programación orientada a objetos con Java", ISBN: 978-84-8322-350-5, Pearson Educación, 2007. http://www.bluej.org/objects-first/ http://www.bluej.org/objects-first/resources/projects.zip
5. Ben-Ari , Mordechai: "Constructivism in Computer Science Education 1", Journal of Computers in Mathematics and Science Teaching (2001) 20(1), 45-73. http://portal.acm.org/citation.cfm?id=274308
6. Moreno, A. & Myller, N. & Sutinen, E. & Ben-Ari, M.: "Visualizing programs with Jeliot 3", AVI '04: Proceedings of the working conference on Advanced visual interfaces, New York, ACM Press, pp373-376, 2004. http://cs.joensuu.fi/jeliot/
7. Allen, Eric & Cartwright, Robert & Stoler, Brian: "Dr Java: A lightweight pedagogic environment for Java", Rice University, ACM SIGCSE'02, February 27-March 3, 2002, Covington, Kentucky, USA. http://www.DrJava.org
8. Beck, Kent: eXtreme Programming Explained, Addison Wesley, Reading, MA, USA, 1999.
9. JUnit http://JUnit.org
10. Di Mare, Adolfo: BUnit.h: Un módulo simple para aprender prueba unitaria de programas en C++ , X Simposio Internacional de Informática Educativa (SIIE'08) realizado del 1 al 3 de octubre 2008, Salamanca, España, I.S.B.N.: 978-84-7800-312-9, pp425-430, octubre 2008. http://www.di-mare.com/adolfo/p/BUnit.htm
11. Di Mare, Adolfo: "Reglas de Indentación", Universidad de Costa Rica, 2008. http://www.di-mare.com/adolfo/p/Indentacion.htm
12. Ceballos, Francisco Javier, "Java 2 - Curso de Programación 3 ed.", ISBN 970-15-1164-6, Alfaomega Ra-Ma, 2006. http://www.fjceballos.es/publicaciones_java.html http://www.ra-ma.es/down/Java2-IGyApIn3ed-Ceballos.zip
13. Deitel, H.M. & Deitel, P.J. "Java Cómo programar 5ta edición", ISBN 970-26-0518-0, Prentice-Hall, 2004. http://www.deitel.com/ ftp://ftp.prenhall.com/pub/deitel/J_HTP/java-htp5e/examples/java5-examples.zip