Ediciones pensando el futuro
Editions thinking the future
Informática 2013
XV CONVENCIÓN Y FERIA INTERNACIONAL
15TH INTERNATIONAL CONVENTION AND FAIR
www.informaticahabana.com
www.informaticahabana.cu
ISBN: 978-959-7213-02-4

USO DE LA VISUALIZACIÓN JELIOT PARA APOYAR EL APRENDIZAJE ACELERADO DE LA PROGRAMACIÓN

USE OF JELIOT VISUALIZATION TO SUPPORT ACCELERATED PROGRAMMING LEARNING

Adolfo Di Mare

Escuela de Ciencias de la Computación e Informática, Universidad de Costa Rica

adolfo.dimare@ecci.ucr.ac.cr

RESUMEN: *Se argumenta que es muy buena idea usar Java como primer lenguaje de programación debido a que el ambiente integrado de programación Jelio junto con JUnitt apoyan el aprendizaje acelerado.*

Palabras Clave: Aprendizaje Acelerado, Java, Programación, Jeliot, JUnit.

ABSTRACT: *It is argued that it is a good idea to use Java as a first programming language because the Jeliot integrated development environment with JUnit support accelerated learning.*

KeyWords: Accelerated Learning, Java, Programming, Jeliot, JUnit.

1. INTRODUCCIÓN

Dice un viejo adagio que "con desconocidos, no se discute ni de política, ni de deportes ni de religión". En el contexto informático habría que agregar que tampoco se debe hablar de cuál es el primer lenguaje de programación que conviene usar en la carrera de computación. Sin embargo, aunque este sea un tema tabú, en la literatura se pueden encontrar personajes importantes que defienden el uso de Java como primer lenguaje [1], C++ como primer lenguaje [2], y otros lenguajes [3],[4]. La escogencia del lenguaje no es el único dilema que encara el docente que enseña el primer curso de programación, pues un grupo piensa que es mejor enseñar primero objetos y luego algoritmos [5], y otro piensa que los algoritmos son más importantes que los objetos.

La opinión de cada docente depende de su experiencia. El autor de este escrito ha trabajado programando, primero con Fortran y Pascal, para luego usar Modula y C++, lo que explica su sesgo a favor de C++ y también su tendencia a enseñar primero algoritmos y después POO [Programación Orientada a Objetos] en el primer curso de programación. Una buena justificación para este proceder son estas palabras de Alexander Stepanov, diseñador y constructor de la biblioteca estándar para C++ (traducción editada) [6]:

> *Me parece que la POO es técnicamente poco sólida. Trata de descomponer el mundo en términos de interfaces que varían desde un solo tipo. Para hacer frente a problemas reales se necesitan álgebras multivaluadas - familias de interfaces que abarcan múltiples tipos. Me parece POO filosóficamente poco sólida ya que afirma que todo es un objeto. Aunque esto fuera cierto, no es muy interesante puesto que decir que todo es un objeto es demasiado ambiguo. Me parece POO metodológicamente equivocada. Comienza con clases. Es como si los matemáticos comenzaran con axiomas cuando lo normal es comenzar con pruebas. Sólo cuando se han encontrado muchas pruebas relacionadas entre sí se puede llegar a, y terminar con, los axiomas. Lo mismo es cierto en la programación:* ***cualquier profesional tiene que comenzar con algoritmos*** *interesantes y solamente cuando se entienden bien se puede llegar a definir una interfaz eficaz y eficiente.*

Si hay que comenzar por los algoritmos también es prioritario para el programador aprender construcción de algoritmos primero, aunque eso no le quita su importancia a todos los demás temas de programación.

Es necesario definir qué significa "programar" para

luego determinar que es lo que hay que enseñar en el primer curso de programación. Por eso, es necesario que todo estudiante conozca las construcciones fundamentales que se requieren para implementar algoritmos.

Fundamentos de programación
• Secuenciación • Asignación y expresiones • Decisiones `if()` • Ciclos `for(;;)` y `while()` • Uso de vectores y matrices • Subrutinas y parámetros

Figura 1

Además, un experto programador conocerá recursividad, herencia, polimorfismo, programación genérica, programación concurrente y uso de excepciones pero, si se ha escogido enseñar primero algoritmos y luego objetos, la lista de tópicos a cubrir es la que aparece en la Figura 1. El objetivo primordial de este artículo es mostrar que la enseñanza de estos conceptos se puede acelerar si se usa el lenguaje Java junto con el entorno de programación Jeliot/JUnit.

2. JELIOT Y JAVA

Cada lenguaje tiene sus herramientas de programación que influyen en la decisión de usarlo. Por eso C++ tiene su biblioteca estándar, sin la que no tendría tanta popularidad. Algunos ambientes de programación se pueden utilizar con varios lenguajes, pero generalmente están especializados en alguno. Por eso, para algunos lenguajes existen herramientas que no están disponibles para otros, como ocurre con la la herramienta Jeliot [7] construida por Ben-Ari [8]:

`http://cs.joensuu.fi/jeliot/`

Ben-Ari construyó Jeliot para lograr que los estudiantes que todavía no saben programar puedan visualizar la ejecución de algoritmos, lo que les permite crear el modelo intelectual que necesitan para aprender [8].

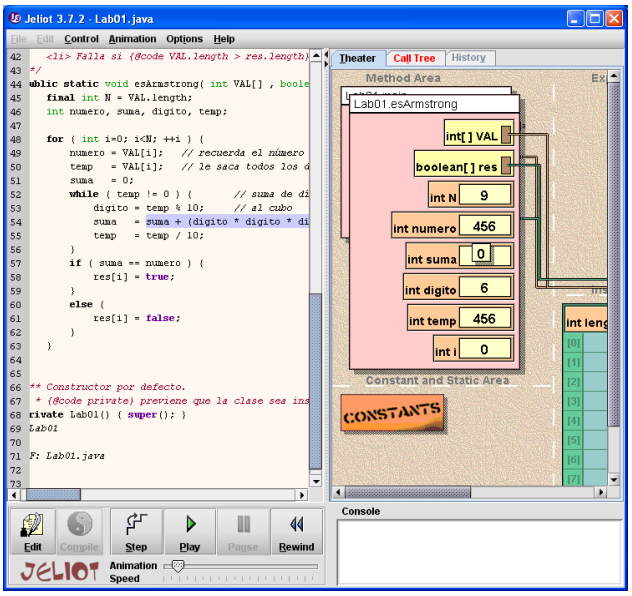

Figura 2

Como se ve en la Figura 2, el ambiente Jeliot es un depurador simbólico para Java que también muestra los registros de activación (en rosado) y los valores de las variables locales al método que se está visualizando. El significado de cada una de las construcciones sintácticas del lenguaje se puede apreciar muy bien al examinar cómo cambian los valores de las variables. También Jeliot muestra los valores de los campos de los objetos cuyo valor siempre está almacenado en la memoria dinámica.

Aunque existe la promesa implícita de construir una versión de Jeliot para otros lenguajes, es especial para C++, la realidad de los hechos es que Jeliot solo está disponible para Java: para usar Jeliot no queda más que programa en Java.

A primera vista parece que existen muchas herramientas similares a Jeliot para otros lenguajes, pero una concienzuda búsqueda permite constatar que, por lo menos al principio de la segunda década del siglo XXI, no existen herramientas similares a Jeliot para otros lenguajes. Pese a que existen muchas formas de visualizar algoritmos (en [9] hay bastantes ejemplos), no existen herramientas que permitan ver cómo varían las variables dinámicamente. Los depuradores simbólicos se pueden usar en sustitución de Jeliot, pero tienen la desventaja de que no muestran la traza de ejecución y los registros de activación. Por eso, en el momento de escritura de este artículo, la dupla Java←→Jeliot es única.

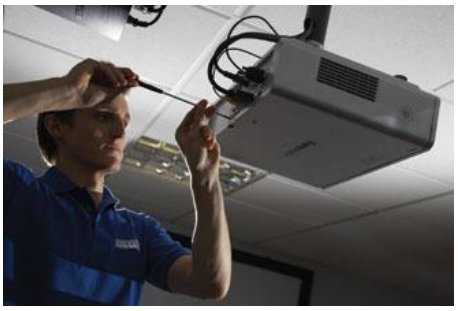

Figura 3

Para que los estudiantes puedan apreciar bien la ejecución Jeliot de un programa Java es necesario que el aula cuente con un proyector que muestre la pantalla del computador del profesor: así los estudiantes pueden ver los algoritmos en acción.

Definitivamente el primer curso de programación se debe impartir en un salón multimedios, en el que posiblemente el proyector estará colgando del techo del aula, como se muestra en la Figura 3 (por comodidad y para evitar el hurto). Las ventajas operativas y didácticas que se derivan de impartir lecciones con estos aparatos cubren con creces los inconvenientes que representa adquirirlos e instalarlos.

3. NÚMEROS DE ARMSTRONG

Se llaman números de Armstrong a los que tienen la propiedad de que son iguales a la suma del cubo de sus dígitos, como ocurre con el número $371=3^3+7^3+1^3$.

```
public static void esArmstrong(
    int VAL[] , boolean res[]
) {
final int N = VAL.length;
int numero, suma, digito, temp;

for ( int i=0; i<N; ++i ) {
    numero = VAL[i];   // recuerda el número
    temp   = VAL[i];   // le saca todos los dígitos
    suma   = 0;
    while ( temp != 0 ) {       // suma de dígitos
        digito = temp % 10;     // al cubo
        suma   = suma + (digito * digito * digito);
        temp   = temp / 10;
    }
    if ( suma == numero ) {
        res[i] = true;
    }
    else {
        res[i] = false;
    }
}
```

Figura 4

En la Figura 4 se muestra un bloque de código que sirve para determinar si un grupo de números son o no números de Armstrong. La gran cualidad que tiene este algoritmo Java es que incluye todas las construcciones necesarias para implementar algoritmos y es tan sucinta que cabe en muy poco espacio: como tiene menos de 25 renglones, se puede ver en cualquier pantalla o ventana. El programa completo se encuentra aquí:

http://di-mare.com/adolfo/cursos/2009-2/Lab01.java.txt

Vale la pena examinar este código para destacar su utilidad pedagógica. No solo usa asignaciones simples sino que tiene expresiones un poco más complejas. También usa tanto el ciclo **`for(;;)`** como el **`while()`** y el **`if()`** incluye la alternativa **`else{}`**. Además, se usa el vector **`VAL[]`** para obtener valores con los que se calcula el resultado booleano **`res[]`**. Realmente es sorprendente que, en tan poco código, estén representadas todas las construcciones sintácticas que se usan para construir algoritmos (ver Figura 1).

La gran ventaja de utilizar Java como primer lenguaje de programación es que, desde la primera lección en programación, se puede montar el programa de Armstrong en Jeliot para mostrarle a los estudiantes cómo cambian las variables: después de todo, un programa siempre es una secuencia de instrucciones que cambian valores almacenados en la memoria de la máquina. Es sorprendente lo rápido que el programa de Armstrong permite exponer los conceptos fundamentales de programación desde la primera lección. Jeliot es necesario porque sirve para explicar visualmente cómo funciona la computadora y cómo se ejecutan los algoritmos.

Una de las desventajas más importantes de Java como primer lenguaje es que tiene una sintaxis muy compleja. Pese a eso, la visualización Jeliot le permite a cada docente explicar cómo funcionan los algoritmos dejando de lado las complejidades de Java. Lo importantes es utilizar el programa de Armstrong para establecer en la mente de cada estudiante qué es lo que debe hacer para programar, de manera que luego, paulatinamente, adquiera el entrenamiento necesario para escribir sus propios programas de computación.

4. DRJAVA Y JUNIT

Una técnica simple que permite lograr pronto el aprendizaje de la compleja sintaxis de Java es aislar pequeños bloques de código, para que los estudiantes no tengan que lidiar en el resto del programa. Esto quiere decir que quien enseña puede entregar programas en los que solo falta una pequeña sección de código para que queden completos, de manera que los estudiantes solo deban trabajar en un reducido bloque de código: así interactúan con algoritmos relativamente pequeños. En [9] se describe una técnica que incorpora la biblioteca JUnit [10] para entrenar rápidamente a los estudiantes en el uso de ciclos y de ciclos anidados, como se muestra en estos ejemplos [11]:

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

```
public static int[] alVerres( int V[] ) {
    if ( V == null ) {
        return null;
    }
    int size = V.length;
    int R[] = new int[ size ];
    int i, j;
    {
        /*****************************\
        *                             *
        *   RELLENE CON SU ALGORITMO  *
        *                             *
        \*****************************/
    }
    return R;
}
```

```
for ( i=0,j=size-1; i<size; ++i,--j ) {
    R[j] = V[i];
}
```

Figura 5

En la parte de arriba de Figura 5 se muestra un primer ejercicio que debe resolver el estudiante usando Jeliot, que le permite una programación visual en la que inmediatamente nota cómo cambian las variables que use. Es necesario que cada alumno trate de encontrar la solución en una sesión de laboratorio, usando su propio computador, para que el profesor pueda guiarlo en el proceso de eliminar errores de sintaxis o de lógica. Por ejemplo, el uso de la coma `','` y del punto y coma `';'` es difícil de asimilar, pues para la mayor parte de los seres humanos estos 2 símbolos son equivalentes. En el enunciado de este ejercicio de laboratorio se aclara lo siguiente: "*Recuerde: agregue su algoritmo dentro del bloque marcado "**{ *** RELLENE CON SU ALGORITMO *** }**"... **¡No modifique ninguna otra parte del programa!***". Esto es muy importante pues, si los estudiantes cambian otras partes del programa, resultan errores de compilación que son muy difíciles de explicar o entender.

Una vez que el estudiante determina que su algoritmo Jeliot ya funciona, puede pasarlo a un programa en JUnit, para que su algoritmo sea ejecutado con muchos casos de prueba diferentes (que han sido previamente preparados por su profesor). Una manera simple y directa de ejecutar el programa JUnit es usar el entorno de programación DrJava [12] que muestra en color verde la ejecución exitosa de una prueba JUnit: si DrJava "se pone verde", el estudiante sabe que ya terminó su trabajo. El color rojo indica que el algoritmo todavía no funciona; en este caso DrJava muestra la prueba que falló, que es la que deberá usar el estudiante de vuelta en Jeliot para corregir su programa.

El entorno DrJava tiene dos cualidades que lo hacen idóneo: primero, incluye un botón **[Test]** que sirve para ejecutar directamente pruebas JUnit. Además, es un programa muy liviano y corre en casi cualquier computador, aún si se le ejecuta desde una memoria portátil USB (llave maya o "*USB stick*"). Otros ambientes también permiten lograr el mismo objetivo pero son más complejos y pesados.

El aprendizaje acelerado se logra primero usando el programa de Armstrong en Jeliot para mostrar cómo funcionan los algoritmos. Luego el estudiante complementa su aprendizaje escribiendo pequeños bloques de código en Jeliot para luego trasladarlos a DrJava en donde puede ejecutarlos con muchas pruebas JUnit. Estas herramientas juntas, DrJava, Jeliot y JUnit, son las que sirven para lograr el aprendizaje en una cantidad reducida de tiempo, que generalmente no pasa de 8 sesiones de clase. La disponibilidad y existencia de estas herramientas es la que, en opinión del autor, justifica el uso de Java como primer lenguaje de programación. Sí es importante que los estudiantes tengan la oportunidad de hacer muchas tareas o proyectos programados, pues así practican la teoría que reciben en las lecciones. Por eso, conviene usar muchas tareas cortas individuales, por lo menos al principio del primer curso de programación.

Para cualquier docente es muy laborioso crear las primeras tareas programadas junto con las pruebas JUnit, pues no es fácil prever todos los tipos de errores que los estudiantes van a cometer: los programas JUnit deben ser muy exhaustivos para que puedan detectar todos los posibles errores que los alumnos pueden cometer. Más aún, existen errores que no es posible detectar con JUnit (como el uso de `'&'` en lugar de `'&&'`). En la práctica ocurre que el profesor descubre nuevos errores cada vez que utiliza la misma tarea programada en un nuevo grupo de alumnos. Por eso conviene usar las mismas prácticas cada vez que se imparte el curso.

El uso de programas Jeliot/JUnit para aprender a programar tiene otro inconveniente (que no debiera serlo): algunos alumnos deciden copiar las tareas o prácticas, pues es relativamente sencillo encontrarlas resueltas en la red o pedirle a alguien que se las solucione. Debido a que es muy lento construir los casos de prueba, es natural que el profesor usará los mismos programas y tareas cada vez que imparta el curso. Por eso, si un alumno no hace su propia práctica porque copia o porque recibe demasiada asistencia de sus tutores o de sus amistades, pierde la oportunidad de aprender y luego es muy difícil que logre recuperarse: los demás habrán aprendido muy rápido y por eso es muy difícil que un estudiante pueda recuperarse solo. Al hacer los primeros exámenes ocurre que algunos obtienen notas suficientes para ganar el curso mientras otros fracasan rotundamente. Aún si el profesor les advierte que "copiar es morir" cuando uno aprende usando prácticas Jeliot/JUnit, muchos alumnos no escuchan o deciden ignorar la advertencia; es bueno tomar en cuenta esta realidad (por lo menos para disminuir la desilusión del docente cuando tiene que darle por perdido el curso a algunos estu-

diantes).

5. POPULARIDAD DE JAVA

Aunque las universidades deben concentrarse más en enseñar los conceptos teóricos que las técnicas prácticas, en computación importa mucho conocer la tecnología disponible. Por eso, conviene más usar un lenguaje popular que uno esotérico, aunque el esotérico tenga cualidades teóricas muy importantes.

1	C	19.9%
2	Java	17.2%
3	Objective-C	9.5%
4	C++	9.3%
5	C#	6.5%

Figura 6

El índice TIOBE es un indicador de la popularidad de los lenguajes de programación: en la Figura 6 se muestran los valores para el segundo semestre de 2012 [13]. Es fácil concluir que quien sepa programar en estos lenguajes podrá colaborar en la construcción más del 60% de los proyectos de programación que se realizan a nivel mundial. Es curioso, pero algunos lenguajes, como Pascal, ya no tienen ni un 1% de popularidad. Quien ya sabe Java, o C# que es un lenguaje similar a Java, puede trabajar en al menos la cuarta parte de los proyectos de programación que se realizan, y si luego llega a dominar C++ cubre también los otros lenguajes, pues C y Objective-C se pueden considerar como versiones reducidas de C++.

También es un hecho que algunas herramientas de aplicación usan lenguajes similares a Java para automatizar procesos, como ocurre en el "*Visual Basic for Applications*" que complementa el "*Office*" de Microsoft, o el "*OpenOffice Basic*" de Sun (estos lenguajes aumentan la potencia de las aplicaciones que usan ingenieros y tecnólogos).

6. CONTENIDO DEL CURSO

Al principio los estudiantes pueden utilizar Jeliot para construir sus programas, pero cuando ya conocen los fundamentos de programación mencionados en la Figura 1 queda suficiente tiempo para ahondar en otros temas. Por ejemplo, en un sitio pueden escoger tópicos de programación concurrente [14] para rellenar el curso de programación y en otros talvez les interese más que los muchachos puedan programar dispositivos móviles (como teléfonos celulares, por ejemplo).

Una estrategia adecuada para definir cuáles temas debe dominar quien terminó Programación 1 es usar la taxonomía del conocimiento publicada la *Association for Computing Machinery* [ACM] cuya versión preliminar 2013 ya está disponible [15].

Programación 1	
OS/OverviewOfOperatingSystems	2
SDF/FundamentalProgrammingConcepts	10
AL/FundamentalDataStructuresAlgorithms	9
PL/ObjectOrientedProgramming	4
Redacción de especificaciones	1
Créditos totales: 2 = ceil(26/15)	26

Figura 7

Al aplicar la metodología propuesta en [16], que se basa en la taxonomía curricular de la ACM [15], una propuesta para el primer curso de programación puede incluir las áreas temáticas que se muestran en la Figura 7, que comprenden un total de 26 horas de exposición en clase. Para calcular los créditos del curso, se puede hacer el siguiente ejercicio aritmético, en que curso de 4 créditos representa 60 horas de lección en un ciclo de 15 semanas, con un total de 4*15=60 horas lectivas. Si se necesitan 26 horas para los temas fundamentales de programación, en un curso con el temario de la Figura 7 quedarían 34 horas para cubrir otros temas en programación.

En la práctica, un estudiante debe invertir 2 horas de estudio individual, en casa o en el laboratorio, por cada hora de instrucción presencial, por lo que un curso de 4 créditos requiere 180 horas de estudio (60 de lecciones presenciales y 120 de estudio individual). La experiencia del autor es que los temas básicos definidos en la Figura 1 se pueden impartir en 16 horas lectivas (equivalentes a 8 sesiones de clase de 2 horas cada una), siempre y cuando los estudiantes inviertas las 32 horas adicionales de estudio y práctica individual para dominar la materia. Sin embargo, hace falta más de la mitad de las 60 horas lectivas para que los alumnos puedan hacer programas pequeños pero queda suficiente tiempo para abarcar otros temas.

Desafortunadamente, en la práctica es difícil lograr que todos los estudiantes ganen su curso de programación, pues alumnos de reciente ingreso muestran carencias importantes en su capacidad de estudio: esto obliga a los profesores a ser pacientes. En otras palabras, la promoción a muchas veces no pasa del 50% en los cursos de primer año de carrera universitaria debido a que los estudiantes todavía no han comprendido que solo memorizar materia no es suficiente para aprender, pese a que con solo memorizar generalmente cualquier alumno tiene un desempeño exitoso en la enseñanza secundaria.

También es posible cubrir los temas definidos en la Figura 7 en 2 o más cursos, es más director hacerlo en Programación 1 y posponer a otros cursos más avanzados el uso de bibliotecas junto con estructu-

ras de datos. De esta forma se puede dejar en un solo curso teórico/matemático el estudio de algoritmos y de estructuras de datos.

7. TEMAS ADICIONALES

Ya casi se cumple una década desde que está disponible permite la programación genérica en Java, pese a que solo provee verificación de tipos en tiempo de compilación y, por eso, elimina la necesidad de usar transferencias de tipos ("*type cast*") [17], en contraposición a las plantillas C++ que permiten una forma mucho más general de reutilización de algoritmos y clases.

Ya no se puede argumentar que Java carece potencia: es tan potente el compilador Java está escrito en Java, y se ha usado para construir programas complejos o muy complejos. Por ejemplo, Java es el lenguaje de programación escogido para crear aplicaciones en la plataforma Android, las que corren en dispositivos de computación que tienen limitaciones significativas porque usan procesadores de velocidad limitada y tienen restricciones severas de uso de energía, pues funcionan en dispositivos portátiles alimentados por baterías, como teléfonos celulares o tabletas electrónicas de lectura.

Después de que los estudiantes han comprendido los fundamentos de programación definido en la Figura 1, es relativamente simple ahondar en otros temas avanzados. En particular, el uso de contenedores sofisticados como el ArrayList<> de Java o del diccionario Map<> se puede hacer con naturalidad. Además, la frondosa biblioteca Java permite que los estudiantes novatos en programación puedan construir programas interesantes que usen una interfaz de ventanas o que tengan que lidiar con objetos complejos, como se requiere si el profesor asigna tareas en que hay que hacer muchos cálculos matemáticos.

A pesar de que muchos docentes argumentan que la programación orientada a objetos requiere de gran sofisticación y que no se puede usar pronto en el curso de programación, si los temas de algoritmos se cubren rápido es posible lidiar pronto con objetos e instancias. Quienes opinan que es mejor enseñar objetos primero puede que se vean inclinados a probar el enfoque expuesto aquí pues, aunque es contrario a su sentir, en realidad no atrasa mucho la introducción del tema de objetos, y talvez por eso les sea aceptable. A fin de cuentas, no se puede decir que un estudiante sabe programar si no usa una biblioteca de objetos para obtener soluciones a problemas concretos.

La limitación más importante de la técnica didáctica aquí descrita es que algunos estudiantes deciden copiar la solución a las prácticas, lo que los deja indefensos cuando hacen su examen porque no aprenden a programar. Es frustrante que los muchachos no escuchen las advertencias del profesor, pero están tan acostumbrados a que todo les salga bien que no reparan en asimilar consejos: después de todo, a la universidad solo ingresan los estudiantes de mejor promedio. Sin embargo, aún si pierden el curso la primera vez, los repitentes luego ganan el curso porque tienen tiempo de madurar el conocimiento para el siguiente ciclo lectivo aún si las tareas y práctica son las mismas del ciclo anterior.

8. CONCLUSIONES

La recomendación de usar de Java como primer lenguaje de programación se fundamenta en la disponibilidad de herramientas que facilitan la enseñanza, pues el dueto Jeliot/JUnit permite acelerar mucho el aprendizaje. Como Java es un lenguaje bastante completo, pues incorpora construcciones sintácticas avanzadas para programación orientada a los objetos, plantillas y uso de excepciones, su escogencia como primer lenguaje no se fundamenta en su popularidad o en la inclinación o sentir particular de un profesor. Sin embargo, otros lenguajes ofrecen oportunidades didácticas diferentes que pueden tornarlos en la mejor elección en algunos contextos académicos.

Es muy conveniente que los primeros trabajos del curso Programación 1 se hagan en Java, usando el dueto Jeliot/JUnit. Estos ejercicios deben ser realizados individualmente, preferiblemente en algunas sesiones de laboratorio asistidas por el profesor del curso. Para las lecciones, es necesario usar un proyector en el que el profesor muestre el funcionamiento de los algoritmos con Jeliot. Las herramientas pedagógicas disponibles hacen del lenguaje Java una excelente elección como primer lenguaje de programación,

9. AGRADECIMIENTOS

Alejandro Di Mare aportó varias sugerencias para exponer las ideas expresadas en este artículo. Además, varios docentes de la Escuela de Ciencias de la Computación e Informática también hicieron observaciones relevantes.

10. REFERENCIAS BIBLIOGRÁFICAS

[1] King, K. N.: The Case for Java as a First Language, Proceedings of the 35th Annual ACM Southeast Conference, pp. 124131, Abril 1997. `http://www2.gsu.edu/~matknk/java/reg97.htm`

[2] Stroustrup, B.: Learning Standard C++ as a New Language, C/C++ Users Journal. pp 43-

54. May 1999.
http://www.research.att.com/~bs/new_learning.pdf

[3] D. Blake, Jonathan: Language Considerations In The First Year CS Curriculum, Journal of Computing Sciences in Colleges, Volume 26 Issue 6, June 2011.

[4] Zelle, John M.: Python as a First Language.
http://mcsp.wartburg.edu/zelle/python/python-first.html

[5] 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

[6] Lo Russo, Graziano: An Interview with A. Stepanov, Edizioni Infomedia srl, 2008.
http://www.stlport.org/resources/StepanovUSA.html

[7] Jeliot 3: Visualization of Java Programs.
http://cs.joensuu.fi/jeliot/

[8] Ben-Ari, Mordechai: Constructivism in Computer Science Education, Journal of Computers in Mathematics and Science Teaching (2001) 20(1), 45-73.
http://portal.acm.org/citation.cfm?id=274308

[9] Ducasse, Stephane: Visualizing en Lectures and Presentations, M2 Reengineering, 2012.
http://stephane.ducasse.free.fr/Teaching.html

[10] JUnit JUnit is a community based testing framework,2012.
http://JUnit.org/

[11] Di Mare, Adolfo: Aprendizaje Java acelerado por casos de prueba JUnit, Artículo #22 del XVIII Congreso Iberoamericano de Educación Superior en Computación [CIESC 2010] realizado en la Universidad Nacional de Asunción, Asunción, Paraguay, octubre 2010.
http://www.di-mare.com/adolfo/p/JUnit6d.htm

[12] DrJava: DrJava is a lightweight development environment for writing Java programs, 2012.
http://drjava.org/

[13] TIOBE Programming Community Index, 2012.
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[14] Di Mare, Adolfo: Introducción de la programación concurrente en el primer curso de programación, Artículo #12 de la Octava Conferencia del Latin American And Caribbean Consortium Of Engineering Institutions LACCEI-2010 (Consorcio de Escuelas de Ingeniería de Latinoamérica y del Caribe), realizado en la Universidad Católica de Santa María de Arequipa, Perú, junio 2010.
http://www.di-mare.com/adolfo/p/cs1cp.htm

[15] Association for Computing Machinery: "Computer Science Curricula 2013 Strawman Draft (February 2012)", 2012.
http://ai.stanford.edu/users/sahami/CS2013/strawman-draft/cs2013-strawman.pdf

[16] Di Mare, Adolfo: Uso de la Taxonomía Curricular ACM para Mejorar la Carrera de Computación, Artículo 151 de la Décima Conferencia del Latin American And Caribbean Consortium Of Engineering Institutions (Consorsio de Escuelas de Ingeniería de Latinoamérica y del Caribe), (LACCEI-2012), realizada del 23 al 27 de julio de 2012 en Universidad Tecnológica de Panamá, Panamá, Panamá.
http://www.di-mare.com/adolfo/p/acmcompu.htm

[17] Mahmoud, Qusay H.: Using and Programming Generics in J2SE 5.0, 2004.
http://www.oracle.com/technetwork/articles/javase/generics-136597.html

ACERCA DEL AUTOR

Adolfo Di Mare: Investigador costarricense en la Escuela de Ciencias de la Computación e Informática [ECCI] de la Universidad de Costa Rica [UCR], en donde ostenta el rango de Profesor Catedrático. Trabaja en las tecnologías de Programación e Internet. También es Catedrático de la Universidad Autónoma de Centro América [UACA]. Obtuvo la Licenciatura en la Universidad de Costa Rica, la Maestría en Ciencias en la Universidad de California, Los Angeles [UCLA], y el Doctorado (Ph.D.) en la Universidad Autónoma de Centro América.