XV JENUI. Barcelona, 8-10 de julio de 2009
ISBN: 978-84-692-2758-9
http://jenui2009.fib.upc.edu/

iJava: un nuevo lenguaje para facilitar el paso del paradigma imperativo al orientado a objetos

Juan A. Sánchez
DIIC, Facultad de Informática, Universidad de Murcia
Campus de Espinardo, 30100 Espinardo, Murcia, España
jlaguna@um.es

Resumen

La aplicación del enfoque procedural permite una presentación paulatina y escalonada de los conceptos de programación imperativa. Sin embargo, algunos alumnos tienen dificultades al pasar a estudiar orientación a objetos (OO), especialmente si se utiliza un lenguaje distinto en cada paradigma. Por otro lado, el empleo de Java en la docencia de ambos modelos es criticado por lo inadecuado que resulta desde el punto de vista pedagógico y la confusión, y malos hábitos, que puede llegar a generar en los alumnos.

Para continuar aplicando el enfoque procedural facilitando la transición del paradigma imperativo al OO se ha desarrollado un nuevo lenguaje de programación imperativo denominado *iJava*. En este artículo se describe este lenguaje y su entorno de desarrollo específico, los aspectos pedagógicos que se han tenido en cuenta en su diseño y cómo favorece una transición paulatina hacia Java durante el aprendizaje de las bases de la programación imperativa. Finalmente se comenta la experiencia de su aplicación durante el curso 2008-09.

1. Motivación

En la Facultad de Matemáticas de la Universidad de Murcia (FMUM) se imparte la Licenciatura en Matemáticas, y el curso académico 2009-10 comenzará a impartirse el nuevo título de Graduado en Matemáticas. Ambos planes de estudio incluyen en su primer curso asignaturas de informática: una primera dedicada a la programación imperativa y la segunda al de la Programación Orientada a Objetos (POO) con Java.

En ésta, como en otras titulaciones, la elección del lenguaje Java viene determinada principalmente porque es muy demandado desde el mundo empresarial [1, 2]. Además, existen entornos de desarrollo profesionales, multiplataforma y gratuitos como NetBeans[1] o Eclipse[2] que pueden ser utilizados por los alumnos tanto en los laboratorios de la facultad como en sus propios ordenadores.

Por otro lado, la clara separación de contenidos (imperativo, POO) responde a la idea compartida por diversos autores [6] de que aprender a programar con los lenguajes de programación más demandados por el mundo empresarial, como lo es Java, no debe impedir el estudio de los conceptos básicos de programación imperativa.

En los últimos años han coexistido dos enfoques totalmente opuestos en cuanto al orden en que se deben enseñar los paradigmas de programación imperativo y OO: el conocido como primero procedural (procedural-first), y el conocido como primero objetos [3] (object-first). La discusión sobre qué enfoque es el más adecuado sigue haciendo correr ríos de tinta [5, 12], pero este debate excede el ámbito de este artículo. Personalmente creo que el enfoque más adecuado es el procedural-first, y a pesar del aparentemente generalizado escepticismo sobre la posibilidad de encontrar una solución [4], merece la pena seguir intentándolo.

Aunque el enfoque procedural-first es ade-

[1] http://www.netbeans.org
[2] http://www.eclipse.org/

cuado para presentar escalonada y paulatinamente los conceptos de programación, como muy bien argumentan Fernández y otros en [7], además de la dificultad para plantear el desarrollo de aplicaciones motivadoras, el mayor problema que tiene este enfoque es del "desplazamiento de paradigma".

El problema reside en la dificultad que encuentran algunos alumnos al verse expuestos a los conceptos de la POO tras haber estado trabajando con los de la programación imperativa durante cierto tiempo. Este efecto se puede ver acentuado si se utilizan lenguajes diferentes en cada parte. Por ejemplo, en [9] se justifica cuidadosamente lo adecuado que resulta desde el punto de vista pedagógico utilizar Pascal o C para la parte imperativa.

Tratando de evitar este problema, y como se propone en [11], en los últimos años se ha estado utilizando Java para impartir los conceptos de programación imperativa antes de pasar a estudiar los de OO también con Java.

En general, esta segunda alternativa presenta muchos más inconvenientes que ventajas como muy bien se explica en [7]. En concreto, los nuevos conceptos que deben aprender los alumnos aparecen entremezclados, y en la mayoría de los casos ocultos, entre otros elementos que aún no conocen, y que deben aprender a ignorar hasta que se les enseñen más adelante. Como ejemplo paradigmático, el primer programa que se le presenta a los alumnos (Listado 1) está repleto de conceptos de POO.

```
public class Prueba {
  public static void main(String []args){
    System.out.println("Hola mundo");
  }
}
```

Listado 1: El primer programa

Por lo tanto, si se desea mantener el enfoque procedural-first impartiendo en primer lugar programación imperativa, y a continuación, introducir a los alumnos a la POO con Java, es necesario contar con un lenguaje imperativo cuya sintaxis/semántica sea la misma que la de Java. De este modo se reduciría el desplazamiento de paradigma que se sufre en caso contrario.

En este artículo presentamos *iJava*, un nuevo lenguaje de programación imperativo basado en Java. *iJava* se ha desarrollado expresamente para poder mantener el enfoque procedural-first sin utilizar Java ni otros lenguajes imperativos completamente diferentes de Java en la parte imperativa. A lo largo del texto se describe el lenguaje, su entorno de desarrollo integrado (iJava Editor) y de qué manera prepara al alumno para una transición paulatina del paradigma imperativo al orientado a objetos impartido en Java reduciendo el efecto del desplazamiento de paradigma. Finalmente, se detalla cómo se ha utilizado *iJava* para impartir la primera parte de la asignatura obligatoria de Informática de la Licenciatura de Matemáticas de la FMUM.

La estructura del artículo es la siguiente: el apartado 2 describe *iJava*, sus características, las del IDE que lo acompaña y los aspectos pedagógicos tenidos en cuenta en su diseño. A continuación, el apartado 3 describe cómo se ha utilizado iJava en la asignatura de Informática en Matemáticas durante el curso 2008-09. En el apartado 4 se repasan algunos propuestas relacionadas con la presente y, por último, el apartado 5 cierra el artículo exponiendo algunas conclusiones y vías futuras.

2. iJava

iJava es un lenguaje de programación imperativo resultado de eliminar de Java todo lo relacionado con la orientación a objetos: clases, interfaces, modificadores de visibilidad y de ámbito, etc. De hecho, la sintaxis de iJava es un subconjunto de la de Java. El objetivo principal de iJava es servir como herramienta para llegar a aprender a programar en Java siguiendo el enfoque bottom-up, es decir, empezando por los conceptos de programación imperativa básicos antes de abordar los relacionados con la POO.

Si bien ya existen otros lenguajes imperativos muy adecuados para iniciarse en la programación, como por ejemplo Pascal, la gran ventaja de iJava respecto a los demás es que comparte sintaxis con Java. De este modo, se

produce una transición mucho más fácil al pasar del lenguaje imperativo al orientado a objetos y se reduce el efecto del desplazamiento de paradigma.

Al compartir sintaxis con Java, los estudiantes no sienten que están perdiendo el tiempo aprendiendo algo que a la larga no van a necesitar. Además, la gran mayoría de programas y funciones desarrollados en iJava por los alumnos pueden ser reutilizadas en las clases que más adelante escriban en Java. Por lo tanto, iJava fomenta la creación de software reutilizable.

Finalmente, iJava cuenta con un reducido conjunto de funciones de librería para el dibujo de primitivas gráficas. La posibilidad de hacer programas con resultados visuales motiva a los alumnos. Sin darse cuenta, los alumnos aplican los conceptos aprendidos con el objetivo de "asombrar" a sus compañeros y divertirse.

2.1. El editor de iJava

Los entornos de desarrollo de Java como Netbeans suelen incorporar cientos de funciones muy útiles para el programador experimentado, pero que pueden asustar o desmotivar al alumno principiante. Por esta razón, para escribir programas en iJava se utiliza un entorno de desarrollo integrado específico denominado *iJava Editor*. El editor de iJava es una aplicación Java disponible en la página `http://ants.inf.um.es/staff/jlaguna/ijava`.

El editor de iJava es extremadamente simple y está diseñado para evitar distracciones innecesarias. De este modo, el alumno puede concentrarse en lo realmente importante, es decir, el aprendizaje de la programación, la creación de algoritmos y su traducción a un lenguaje de programación. Además, al tratarse de una aplicación Java es multiplataforma por definición.

Como se puede ver en la figura 1, el editor de iJava consta exclusivamente de un panel para escribir el código fuente, un botón para compilar y ejecutar el programa y un segundo panel en el que aparecerán los mensajes de error del compilador.

El resultado de la ejecución de los programas se muestra en una segunda ventana (fi-

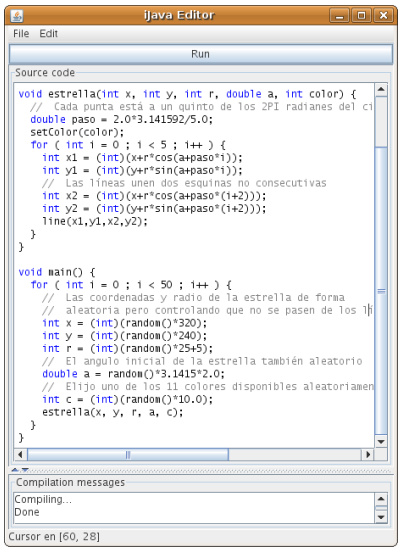

Figura 1: Apariencia del editor de iJava

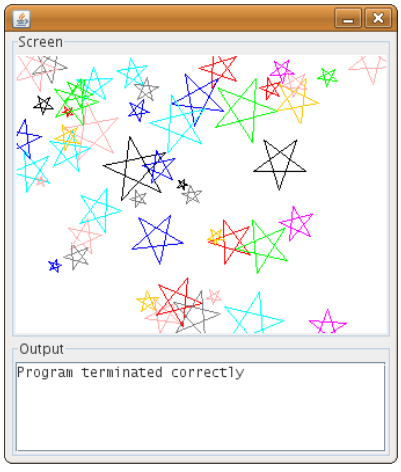

Figura 2: Ventana de resultados de la ejecución del programa

gura 2) en la que también hay dos paneles. El panel superior hace de dispositivo de salida gráfico y consiste en una matriz de puntos de 320x240 pixels. En esta "pantalla" es donde aparecen las primitivas gráficas generadas por las sencillas funciones de librería incluidas en iJava que veremos en el siguiente apartado. El panel inferior sirve de dispositivo de salida textual (o consola) a través del cual el programa puede mostrar mensajes.

2.2. Definición del lenguaje

Como se ha dicho antes la sintaxis de iJava es un subconjunto de la de Java. Los tipos de datos disponibles son los siguientes: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`. Los operadores disponibles son exactamente los mismos que en Java, así como la forma de declarar e inicializar variables y constantes. En concreto la palabra reservada `final` es la que indica la declaración de una constante. Del mismo modo, en iJava los bloques se delimitan con '{' y '}'. iJava también cuenta con las mismas sentencias de control de flujo e interación que Java: `if`, `if-else`, `switch`, `for`, `while` y `do-while`.

El lenguaje iJava incorpora un reducido conjunto de funciones de librería clasificadas en los tres grupos siguientes:

- **Entrada/salida**: `print`, `println`, `readChar`, `readInteger`, `readDouble` y `readString`.
- **Numéricas**: `sin`, `cos`, `sqrt`, `abs` y `random`.
- **Gráficas**: `setColor`, `point`, `line`, `rectangle`, `circle`, `clearScreen` y `sleep`.

Las funciones `print` y `println` aceptan cualquier tipo básico como argumento y muestran su valor por la consola. La familia de funciones `readxxx` muestra una ventana a través de la cual el usuario puede introducir un número entero, uno real, un carácter o una cadena de caracteres,[3] respectivamente.

[3]Por cadena de caracteres se entiende un array de caracteres cuyo último elemento es un carácter especial o marca de fin.

De las funciones numéricas destacar que las cuatro primeras trabajan con valores reales de tipo `double`. La función `random` devuelve un número real aleatorio entre 0 y 1 también de tipo `double`. Las funciones gráficas admiten múltiples parámetros que determinan las coordenadas, tamaños y propiedades de las primitivas gráficas que dibujan. La función `sleep` realiza una pausa del número de milisegundos que se le indique. Esta función, en combinación con `clearScreen`, permite la realización de pequeñas animaciones gráficas.

Además de usar funciones de librería, en iJava se pueden declarar funciones utilizando la misma sintaxis que los métodos de Java, pero sin utilizar los modificadores de visibilidad o ámbito de éste, es decir, usando una sintaxis muy similar a la del lenguaje C. Del mismo modo, los procedimientos son funciones que devuelven el tipo de dato especial `void`, pero este tipo no puede ser utilizado para declarar variables ni constantes.

Los arrays en iJava se comportan exactamente igual que en Java, con la única excepción de que no son objetos por lo que no tienen la propiedad `length`. Por lo tanto, se declaran, inicializan, crean y se accede a ellos del mismo modo que en Java. Concretamente, la creación de los arrays se realiza con la palabra reservada `new` y también pueden tener múltiples dimensiones.

Semánticamente hablando, la estructura de un programa válido en iJava puede tener dos formas. Puede consistir en una serie de declaraciones y sentencias o puede consistir en un conjunto de declaraciones de funciones. En este caso, una de ellas debe llamarse `main`, no tener parámetros y devolver `void`. De este modo, programas como el del listado 2 son válidos. Esto permite a los alumnos comenzar a familiarizarse con la escritura de programas desde el primer día de clase.

Por otro lado la semántica de paso de parámetros es idéntica a la de Java, es decir, sólo existe el paso por valor. Sin embargo, cuando un array se utiliza como argumento en la invocación de una función, desde el cuerpo de la misma sí que se puede modificar el contenido del mismo. Igualmente, es posible declarar

```
final int ADULTO = 18;
final double PI = 3.141592;
final double TWO_PI = 2.0*PI;
final double E = 2.718281;
int edad;
int grado;
boolean identidad;
double area;
```

Listado 2: Programa válido en iJava consistente en una serie de declaraciones de constantes y variables

funciones cuyo tipo devuelto sea un array.

2.3. Aspectos pedagógicos

Un programa válido en iJava puede consistir únicamente en una serie de sentencias y declaraciones de variables y constantes. Esto evita el desfase que se suele producir en la temporización de las clases teóricas y las de laboratorio, ya que los alumnos pueden escribir programas desde el primer día de clase. También facilita que el alumno se enfrente a la detección/corrección de errores sintácticos desde el comienzo con programas muy simples.

De no hacerse así se plantean tres alternativas: retrasar la creación del primer programa hasta que se hayan abordado el concepto de abstracción operacional mediante la construcción de funciones, adelantar dicha explicación al comienzo de la asignatura o recurrir al clásico "esto ya se verá más adelante". Las tres alternativas tienen puntos débiles. La primera implica el mencionado desfase temporal. La segunda impone un orden antinatural en la temporización de la asignatura o, lo que es peor, reduce a la nada el tiempo dedicado a conceptos que, desde mi punto de vista, deben estudiarse antes (variables, control de flujo, iteraciones, etc). La tercera opción es quizás la más utilizada y, posiblemente, la menos adecuada pedagógicamente hablando.

En general, al usar iJava, a medida que la asignatura avanza, los programas que realizan los alumnos van ganando en complejidad y extensión. De este modo, los propios alumnos ven claramente las ventajas del enfoque estructurado cuando, llegado el momento, se les presenta el concepto de abstracción operacional mediante la construcción de funciones y procedimientos. En ese punto, ellos mismos han sufrido los problemas típicos de un programa no estructurado, por lo que tienen un mejor criterio para valorar la mejora que supone el nuevo concepto.

Por otro lado, se decidió no incluir entre los tipos básicos de iJava uno que representara cadenas de caracteres. De este modo, para mostrar por consola la palabra "hola" es necesario utilizar cuatro invocaciones distintas a la función `print`. Esto fomenta el aprendizaje de la composición secuencial de comandos, la idea de flujo de programa y, además, permite usar el concepto de cadena de caracteres al explicar los arrays. Además, las funciones básicas de comparación, reemplazo de caracteres, o búsqueda de subcadenas, resultan ejemplos muy ilustrativos de las sentencias de control de flujo e iteración, y de los algoritmos de búsqueda básicos.

Como ya se ha mencionado, el uso de funciones de librería gráficas aumenta la motivación de los alumnos por desarrollar aplicaciones visualmente atractivas. Algunas de estas aplicaciones pueden consistir en pequeñas animaciones. Desde un punto de vista pedagógico éstos son ejemplos perfectos de aplicación de los bucles y del manejo de variables. Otras aplicaciones gráficas como el juego de las cuatro en raya o el del ahorcado, también ocultan buenos ejemplos de uso de arrays y bucles tras lo atractivo del resultado gráfico final.

Las funciones de librería de entrada/salida se incluyeron para aumentar la interactividad de las aplicaciones que se pueden desarrollar con iJava. Sin embargo, se decidió ocultar la complejidad inherente al manejo de estos dispositivos en Java (entrada síncrona/asíncrona, con o sin buffer, manejo de excepciones, etc). Además, las funciones (`print`, y `println`) tienen exactamente la misma sintaxis y semántica que los métodos de igual nombre disponibles en el objeto `System.out` de Java.

En relación con los arrays, en iJava hay dos aspectos a remarcar. Primero, el programador no puede conocer el tamaño de un array por lo que se ve forzado a arrastrar el tamaño uti-

lizado durante su creación por medio de parámetros adicionales en las funciones que manejan estas variables. Esto fomenta el manejo y aprendizaje de la parametrización de funciones. Segundo, la importancia de la introducción temprana del concepto del "aliasing". En particular el que se produce cuando se pasa como parámetro un array, o se asigna un array a otra variable del mismo tipo, momento a partir del cual dos nombres hacen referencia a un mismo conjunto de datos. Introducir en este momento este concepto facilita a los alumnos la comprensión del concepto de referencia que más adelante será ampliamente utilizado al trabajar con objetos en Java.

Finalmente, iJava tampoco cuenta con ningún mecanismo para la creación de tipos complejos como los registros de Pascal. Al no existir esta posibilidad, el alumno debe utilizar múltiples variables para representar los distintos atributos de una misma entidad. Si además se desea manejar varias entidades es necesario un array por atributo. Tras haber "peleado" con implementaciones de este tipo las ventajas de manejar objetos se hacen mucho más evidentes al comenzar con la POO.

3. iJava en la práctica

En este apartado se describe el desarrollo de la primera parte de la asignatura de Informática impartida en la Licenciatura de Matemáticas de la FMUM durante el primer cuatrimestre del curso 2008-09. Esta asignatura de carácter anual tiene 9 créditos de los cuales cada cuatrimestre dedica 3 a teoría (2h/sesión) y 1,5 a laboratorio (1h/sesión).

La evaluación de la asignatura consta de un examen teórico-práctico consistente en resolver varios problemas (30 %), la realización de dos conjuntos de problemas en grupos de dos alumnos (50 %), la realización de cinco pequeños trabajos individuales (10 %) y dos talleres en grupo (10 %). La calificación final es la suma ponderada de las calificaciones obtenidas en cada parte.

Tradicionalmente en el primer cuatrimestre de esta asignatura se introducía al alumno en la programación imperativa usando el propio Java, mientras que en el segundo se estudiaban los conceptos de POO también en Java. En ambos casos se utilizaba Netbeans como plataforma de desarrollo. Sin embargo, en los últimos años se detectó cierta dificultad entre los alumnos para asimilar bien los conceptos de la primera parte. Ésto hacía que los contenidos impartidos/asimilados en el segundo cuatrimestre se vieran reducidos debido a los necesarios repasos de conceptos anteriores.

Tema	**Sesiones**
Introducción	1
Manejo de información	1
Declaraciones y sentencias	1
Control de flujo	1
Iteraciones	1
Funciones y procedimientos	2
Arrays	2
Búsqueda y ordenación	1
Recursividad	1

Cuadro 1: Temporización de la asignatura

Esto motivó el desarrollo de iJava y su experimentación durante el presente curso académico. El temario seguido consta de los 9 temas mostrados en la tabla 1.

Los primeros dos temas ocuparon las dos primeras sesiones de teoría y una de laboratorio. Gracias al uso de iJava, a partir de la segunda semana de clase, comenzaron las sesiones de laboratorio en las que los alumnos escribían pequeños programas declarando variables, constantes, y planteando expresiones aritméticas o booleanas cuyos resultados mostraban por pantalla con la función de librería `print`.

Todos los temas ocuparon una sesión, excepto los temas 6 y 7 a los que se les dedicó dos sesiones a cada uno ya que, históricamente los conceptos de abstracción operacional y el manejo de arrays son los que habían costado más esfuerzo a los alumnos.

A partir de la cuarta semana, los alumnos ya manejaban con soltura el editor de iJava y se centraban únicamente en resolver los problemas propuestos. En este punto se les planteó el primero de los dos conjuntos de problemas

para realizar en pareja y, algunos alumnos, empezaban a escribir programas por su cuenta en los que utilizaban gráficos.

La semana 9 se dedicó a los algoritmos de búsqueda y ordenación, terminó el plazo de entrega de las primeras prácticas en pareja y se les plantearon las segundas compuestas mayoritariamente por ejercicios para manejar cadenas.

La semana 10, tras haber terminado el tema 8, se realizó el primer taller que tendría una duración de dos sesiones. En este caso se planteó la creación de un juego tipo cuatro en raya. Los objetivos principales eran aplicar la descomposición modular, crear funciones, utilizar arrays para representar el estado del juego, y aplicar los algoritmos de búsqueda lineal para detectar si había un ganador.

La semana 13 se dedicó al último tema y las dos últimas a realizar el segundo taller consistente en programar el juego del ahorcado.

En cuanto a los trabajos individuales, cada uno consistía en resolver un problema relacionado con uno de los tópicos de la asignatura. Destacar que el último, el relativo a la recursividad, consistía en implementar una función recursiva que hiciera un dibujo. Este ejercicio motivó mucho a los alumnos y desarrollaron muy buenos trabajos.

4. Trabajos relacionados

La idea de partir de un lenguaje simple e ir ampliando su sintaxis y semántica para terminar utilizando un lenguaje completo no es nueva. En particular, en [8] se describe un enfoque diacrónico de la docencia de la programación. En este enfoque se presentan los conceptos de la programación de forma escalonada y paulatina, se hace énfasis en la recurrencia de algunos de ellos y se trata de anular el efecto del desplazamiento de paradigma mediante la utilización de un lenguaje orientado a objetos expresamente diseñado a tal efecto. La sintaxis y semántica de este nuevo lenguaje coincide con la de Pascal, el lenguaje utilizado para la exposición de la programación imperativa en la asignatura anterior. El nuevo lenguaje incluye además los mínimos cambios necesarios para que se puedan utilizar los conceptos básicos del paradigma orientado a objetos.

Si bien estoy totalmente de acuerdo en el enfoque presentado por los autores del mencionado trabajo, creo que es importante que los alumnos utilicen y adquieran experiencia en el manejo de un lenguaje demandado por el mundo empresarial. Por esta razón, la idea de crear un lenguaje específico para la docencia de la POO creo que es menos positiva que la idea presentada en este trabajo consistente en crear un lenguaje específico para la docencia de la programación imperativa, estructurada y modular, pero que al mismo tiempo, permite el mismo transito fluido y paulatino hacia el lenguaje orientado a objetos usado en la docencia de POO, que en este caso es Java, un lenguaje ampliamente utilizado en la industria.

Otra aproximación similar a la anterior pero que sí permite el aprendizaje de Java es la propuesta en [10]. En ella se presenta el concepto de "nivel de lenguaje". Cada nivel de lenguaje es una reducción o simplificación del lenguaje de nivel superior considerando Java en su totalidad el de más alto nivel. De este modo los alumnos se enfrentan en primer lugar al nivel más bajo y, a medida que se avanza en el temario y se van presentado nuevos conceptos de POO, se va pasando a los niveles superiores. Los autores exponen también la extensión que de su propia entorno de desarrollo pedagógico, llamada Dr. Java[4], han hecho para que sea capaz de ofrecer estos distintos niveles de lenguaje.

A diferencia de la presente, esta propuesta está claramente enmarcada dentro del enfoque object-first. Sin embargo, podría ser interesante como metodología a utilizar en la segunda parte de la asignatura. Es decir, tras haber introducido a los alumnos en la programación imperativa, procedural y estructurada de forma progresiva y escalonada mediante iJava, parece adecuado continuar esta progresión escalonada dentro de Java, mediante la utilización de DrJava y los niveles de lenguaje que proponen sus autores.

[4] http://www.drjava.org

5. Conclusión y vías futuras

En este artículo se ha presentado el nuevo lenguaje de programación imperativo iJava como medio para reducir el problema del desplazamiento de paradigma que se produce en el enfoque procedural-first al utilizar dos lenguajes distintos siendo Java el segundo. Se han expuesto las razones pedagógicas por las cuales se considera una alternativa adecuada y se ha presentado un caso de aplicación práctica en la licenciatura de Matemáticas.

Es necesario esperar a que termine el curso académico para poder evaluar el grado de éxito del experimento. La posible mejora en la adaptación a Java durante el segundo cuatrimestre que vayan a experimentar los alumnos se verá reflejada en una encuesta que se tiene previsto realizar.

Actualmente se está estudiando la posibilidad de combinar iJava y DrJava en su modalidad de "niveles de lenguaje" para conseguir un curso de introducción a la programación con una curva de aprendizaje aún más suave.

Por último, cabe mencionar que iJava podría ser también un buen candidato a utilizar en otras carreras que, al igual que Matemáticas, requieren de ciertos conocimientos básicos de programación, y también en un primer curso de Ingeniería Informática.

Agradecimientos

Este trabajo se ha desarrollando dentro del proyecto "Experiencia Piloto de Implantación de la Metodología ECTS en la Licenciatura en Matemáticas, Primer Curso". Agradecimientos especiales para Antonio Ruiz.

Referencias

[1] *Informe de la Comisión de Evaluación del diseño del Título de Grado de Matemáticas. Libro blanco del Grado en Matemáticas.* ANECA, marzo 2004.

[2] *Salidas Profesionales de los Estudios de Matemáticas. Análisis de la Inserción Laboral y Ofertas de Empleo.* RSME-ANECA, julio 2007.

[3] Barnes, David J. and Kölling, M. *Objects First with Java A Practical Introduction using BlueJ.* Prentice Hal, 2008,

[4] Brooks, F. P. *No Silver Bullet - Essence and Accidents of Software Engineering.* IEEE Computer 20(4):10-19, April 1987.

[5] Bruce, Kim B. *Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list.* ACM SIGCSE Bulletin 36(4):29-39, Dec. 2004.

[6] Dingle, A., Zander C. *Assessing the ripple effect of CS1 language choice.* J. Comput. Small. Coll. 16(2):85-93, 2001.

[7] Fernández Muñoz L., Peña R., Nava F., Velázquez Iturbide A. *Análisis de las propuestas de la enseñanza de la programación orientada a objetos en los primeros cursos.* Actas de las VIII Jornadas de Enseñanza Universitaria de la Informática (JENUI'02). pp. 433-440, Cáceres, España, Octubre 2002.

[8] Fernández Muñoz L., Peña R., Nava F., Velázquez Iturbide A. *Enfoque Diacrónico para la Enseñanza de la Programación Imperativa.* Actas de las VIII Jornadas de Enseñanza Universitaria de la Informática (JENUI'02) pp. 407-414, Cáceres, España, Octubre 2002.

[9] García Molina, J. *¿Es conveniente la OO en primer curso de programación?.* VII Jornadas de Enseñanza Universitaria de la Informatica (JENUI'2001) pp. 293-298, Palma de Mallorca, 16-18 de julio, 2001.

[10] Hsia, James I., Simpson, E., Smith, D., and Cartwright, R. *Taming Java for the Classroom.*In Proc of the 36th ACM SIGCSE. pp. 327-331, St. Louis, USA, Feb. 2005.

[11] Reges, S. *Back to Basics in CS1 and CS2.* In Proc of the 37th ACM SIGCSE. p. 293-298, Houston, March 2006.

[12] Vilner, T., Zur, E., Gal-Ezer, J. *Fundamental concepts of CS1: procedural vs. object oriented paradigm - a case study.* ACM SIGCSE Bulletin 39(3):171-175, Sept. 2007.