¿Es conveniente la orientación a objetos en un primer curso de programación?

Jesús J. García Molina
Departamento de Informática y Sistemas
Universidad de Murcia
30071 Campus de Espinardo (Murcia)
e-mail: jmolina@um.es

Resumen

Este trabajo defiende la tesis de la inconveniencia de elegir el paradigma orientado a objetos para un curso de introducción a la programación, en el ámbito de las titulaciones universitarias de informática. Para justificar esta postura, primero se establece cuáles deberían ser los objetivos de un primer curso de programación y a continuación se analiza cómo la orientación a objetos dificulta la consecución de esos objetivos.

1. Motivación

Desde principios de la década pasada, la programación orientada a objetos (POO) es reconocida como el paradigma más adecuado para mejorar la calidad del software, ya que favorece dos aspectos importantes como son la reutilización y la extensibilidad del código. En los últimos años, la tecnología OO ha sido aceptada por la industria del software, como lo prueba el fenómeno Java. Así, el último informe curricular de ACM/IEEE [1] incluye la POO dentro del núcleo de conocimientos básicos que un estudiante de informática debe conocer.

Las directrices generales propias de las tres titulaciones de informática (B.O.E. de 20, Noviembre, 1990) no incluyen la POO como materia propia o parte de la materia *Metodología y Tecnología de la Programación*. Por tanto no debe extrañar que casi todos los planes de estudio de nuestras universidades, elaborados una vez publicadas las directrices, hayan ignorado la OO, e incluso entre los planes reformados recientemente son muy pocos los que incluyen una asignatura que enseñe los fundamentos de esta tecnología.

Se puede afirmar que en la actualidad existe un convencimiento en los departamentos encargados de la enseñanza de la programación de la necesidad de que un titulado de informática haya recibido una formación en OO a lo largo de sus estudios. La cuestión es qué, cuándo y cómo. Por razones de espacio, en este trabajo no se presenta una propuesta completa sobre cómo abordar la enseñanza de la tecnología OO en un plan de estudios, sino que nos limitaremos a estudiar la conveniencia o no de elegir el paradigma orientado a objetos para un primer curso de programación. No obstante, al final se incluirá un esquema de los contenidos que se consideran básicos.

Hasta el auge de la OO, el mundo académico estaba convencido de que la programación estructurada era el paradigma adecuado para una introducción a la programación, utilizando Pascal, Modula o Ada como lenguajes de programación. Sin embargo, conforme maduraba la OO, han ido apareciendo trabajos como [2,7] que defienden que el alumno debe iniciar el aprendizaje de la programación con la POO y se han publicado libros tales como [3, 9] que ilustran este enfoque. En nuestro país también ha surgido el debate, y actualmente muchos departamentos encargados de la docencia en programación están discutiendo si llevar o no a la práctica tales propuestas, siendo cada vez más los profesores que propugnan que los alumnos comiencen a programar con Java. De hecho, algunos centros ya han experimentado con la introducción en el primer curso de un lenguaje OO (por lo común Java, aunque también se ha probado con C++ y Delphi). En este trabajo defenderé mi convicción de que no es apropiado que los estudiantes de informática aterricen en el mundo de la programación a través de la perspectiva que proporciona la OO. Mis

argumentos son fruto de mi experiencia en la Facultad de Informatica de la Universidad de Murcia enseñando durante seis años programación desde un punto de vista tradicional (en concreto utilizando el enfoque presentado en [8]), diez años impartiendo una asignatura sobre los fundamentos de la OO y cuatro años enseñando métodos OO y técnicas de diseño OO.

El trabajo está organizado del siguiente modo. Primero se exponen los objetivos de un primer curso de programación a continuación se analiza cómo la POO dificulta la consecución de dichos objetivos y finalmente se presentan las conclusiones.

2. Un primer curso de programación

Todos los planes de estudio de informática incluyen en primer curso o bien dos asignaturas cuatrimestrales o bien una asignatura anual (lo menos común) destinadas a introducir al alumno en el universo de la programación de ordenadores. Aunque no tiene que ver con el problema que estamos tratando, hago un inciso para señalar mi convencimiento de la necesidad de que esta asignatura sea anual, ya que de este modo los alumnos tienen todo un curso para madurar una materia que para muchos de ellos supone entrar en una nueva forma de resolución de problemas. Además se evita un examen final en febrero, cuando apenas han tenido tiempo de asimilar y comprender los primeros conceptos y algoritmos, y también se permite que el profesor pueda organizar mejor las prácticas, ya que se elimina la interrupción entre el fin de una asignatura cuatrimestral y el comienzo de la siguiente.

Este primer curso (a partir de ahora supondré una asignatura anual) debe ser planificado con la suposición de que los alumnos no conocen absolutamente nada sobre programación, ya que en la enseñanza secundaria no existe ninguna materia obligatoria en la que se imparta una introducción a la programación. Desde mi punto de vista el objetivo del primer curso de programación es *dotar a los alumnos de los mecanismos necesarios para enfrentarse a la creación de programas para problemas pequeños y elementales en el marco de expresividad de un lenguaje imperativo estructurado, enseñándoles métodos, técnicas y conceptos que les permitan resolver los problemas, llegando de un modo riguroso desde la especificación al programa correcto, y siendo conscientes del problema de la eficiencia*. A continuación expondré una organización de un primer curso de programación y en base a ella justificaré la conveniencia del uso de un lenguaje imperativo estructurado (también se podría denominar lenguaje procedural) frente a un lenguaje OO.

El primer curso de programación debería estar organizado en dos partes: una introducción a la programación (IP) y unos fundamentos de la programación (FP), cada una con una carga mínima de 7.5 créditos, 4.5 teóricos y 3 prácticos.

La parte IP estaría destinada a que el alumno escriba sus primeros algoritmos utilizando: i) las composiciones secuencial, condicional e iterativa, ii) acciones y funciones, iii) tipos de datos básicos y iv) algoritmos básicos de recorrido y búsqueda en tablas (arrays) y secuencias. En los tres primeros meses debería manejar una notación algorítmica en vez de un lenguaje de programación, para prestar atención al diseño de los algoritmos, acostumbrando al alumno a pensar bien la solución antes de escribir el programa y haciéndole consciente de que se puede trabajar en el desarrollo de un algoritmo sin necesidad de un ordenador, todo ello para evitar crear "programadores compulsivos". Durante esta etapa puede seguirse el enfoque presentado en [8] (en nuestra facultad seguimos uno inspirado en dicho texto plasmado en [5]). En el último mes de IP, se introduce el concepto de lenguaje de programación, se establece la correspondencia entre la notación algorítmica y un lenguaje estructurado (Pascal o Modula), se enseña a los alumnos el manejo de un entorno de programación y entonces se les exige que traduzcan algunos de los algoritmos escritos en la notación al lenguaje de programación elegido, proceso que les resultará casi inmediato. El objetivo fundamental de IP es que el alumno comprenda bien la construcción de iteraciones. Para ello, es conveniente trabajar con el razonamiento inductivo que se esconde detrás del concepto de invariante (cuya formalización no es necesaria), resolviendo ejercicios clásicos como el de "la meseta más larga", "la secuencia con mayor peso", etc. Estos algoritmos iterativos implicarán el manejo de dos estructuras de datos básicas en programación: secuencias y arrays.

La parte de FP tendría como objetivo mostrar un conjunto de conceptos teóricos cuya

comprensión es fundamental para un buen programador, sobre los cuales se profundizará en cursos posteriores. Estos conceptos serían: i) abstracción operacional, ii) recursión, iii) eficiencia, iv) abstracción de datos, y v) estructuras de datos dinámicas. La introducción al problema de la eficiencia serviría para presentar los algoritmos básicos de ordenación y búsqueda. El otro objetivo importante de FP sería que los alumnos se enfrentasen al problema de construir programas pequeños, para lo cual utilizarían primero la técnica del refinamiento por pasos sucesivos y más adelante se les introduciría en la programación basada en módulos.

3. La OO en la introducción a la programación

Supuesto que aceptamos como razonable el planteamiento expuesto en el apartado anterior para un primer curso de programación, ahora analizaremos cómo influye la elección de la POO como primer paradigma en lugar del paradigma imperativo estructurado.

Antes de continuar conviene hacer algunas aclaraciones. Primera, la POO puede considerarse como un paradigma construido sobre la base del paradigma imperativo, en tanto en cuanto el modelo operacional está basado en la asignación, como mecanismo de cambio de estado, y en las estructuras de control secuencial, condicional e iterativa como mecanismos de control de la ejecución. Sin embargo, la POO cambia la invocación a rutinas por mensajes, lo cual combinado con la herencia introduce los conceptos de polimorfismo y ligadura dinámica, dando lugar a un paradigma claramente diferenciado.

En segundo lugar, existen diferentes visiones sobre cómo iniciar la enseñanza de la programación con la POO. Una posible visión, que podríamos denominar *enfoque débil*, sería impartir un curso tradicional de programación pero utilizando un lenguaje OO para escribir los programas (no se explican con detalle los fundamentos de la OO). En el otro extremo tendríamos un *enfoque fuerte* en el que se enseñan desde el principio los fundamentos de la OO y el alumno llega a manejar la herencia. Este segundo enfoque es defendido por B. Meyer en [7] y también se justifica en [2]. En medio, tendríamos un *enfoque híbrido* como el reflejado en [3], donde se plantea definir los conceptos de clase y de diseño dirigido por responsabilidades, combinado con el resto de temas propios de un primer año (estructuras de control, iteración, arrays, ordenación, recursividad, estructuras de datos dinámicas,…) pero no se maneja la herencia y el polimorfismo

La tercera aclaración es señalar que en la discusión que sigue supondremos que el lenguaje OO que se enseñaría en un primer curso es un lenguaje OO puro[1] (Eiffel, Smalltalk o Java) no un híbrido como C++, ya que existe un consenso sobre la conveniencia de iniciar la enseñanza de la OO con un lenguaje OO puro.

Si nos centramos en IP, nos encontramos que en esa etapa el alumno se debe preocupar principalmente de dominar algoritmos iterativos que manejan secuencias (simplemente obtenidas del buffer del teclado o generadas por recurrencia), strings y arrays (posiblemente multidimensionales) de tipos básicos. La introducción del concepto de objeto complica innecesariamente estos algoritmos, al ser necesario incluir el concepto de clase y mensaje, en vez de escribir una simple secuencia de instrucciones imperativas como se haría en Pascal, por ejemplo. En el caso de Java, se da una consideración especial a los tipos básicos y a los arrays, de modo que se podrían escribir los algoritmos de una forma similar a Pascal, pero todavía sería necesario incluir el concepto de clase (el programa sería la rutina *main*) y el de mensaje para escribir las operaciones de entrada/salida (aunque el alumno puede escribirlas por acto de fe, sin preocuparse del concepto de mensaje). Para manejar strings, ya sería necesario introducir la clase *String*, la creación de objetos y el empleo de mensajes (por ejemplo para comprobar la igualdad). Si se examina el texto [3], se puede ver cómo en el segundo capítulo, en el que sólo se analiza cómo mostrar por pantalla un mensaje (un string almacenado en una variable), se deben introducir los conceptos de clase, método, mensaje, objeto, referencia y asignación con semántica de referencia. Todo esto parece una complicación evitable.

---

[1] Aunque consciente de las discusiones sobre la pureza de Java, comparado con C++ se puede considerar un lenguaje OO puro.

Es claro que en una asignatura con el enfoque de IP la OO no produce ningún beneficio, al contrario. ¿Pero es posible dar otro enfoque para IP? En mi opinión, no. Al principio de sus estudios de programación, el alumno debe pelearse con el diseño de algoritmos que le van a permitir dominar la iteración y manejar los tipos y estructuras de datos básicos. Y el diseño de este tipo de algoritmos se ajusta al paradigma imperativo, ya que el concepto de mensaje, sobre el que se articula el modelo computacional OO, no es tenido en cuenta, aunque se use POO, por ser innecesario.

Por ejemplo, supongamos un ejercicio clásico de un curso de introducción a la programación: Sea una secuencia de 0's y 1's de longitud n (≥1) almacenada en un array *x*, debemos contar el número de pares de índices *(i,j)* con *i<j* para los cuales *x[i]=0* y *x[j]=1*. Una solución expresada en Pascal (sólo mostramos la iteración) sería:

```
i:=1;
np:=0;
nc:= 1- x[1];
while i<>n do begin
   i:= i + 1;
   if x[i] = 1 then np:= np + nc
   else nc:= nc + 1
end
writeln('Numero de pares=',np);
```

La comprensión del razonamiento que lleva a un algoritmo de este tipo es el principal objetivo de IP. Supuesto que en vez de una solución imperativa nos planteásemos una solución OO, ¿habría que cambiar la forma de razonar? Claramente no. No tiene sentido aplicar en este caso el modelo computacional OO caracterizado por el paso de mensajes entre objetos.

Los conceptos de la POO (clase, objeto, mensaje, herencia, polimorfismo y ligadura dinámica) y el modelo computacional subyacente son aplicables cuando es necesario estructurar programas en diferentes módulos. De hecho en un curso de introducción a la POO, para que el alumno ponga a prueba su compresión del significado de la OO, es preciso que se enfrente a la construcción de programas en los que deba identificar las clases, las relaciones entre clases, la interacción entre objetos. El típico problema del "cajero automático" ilustra bien el tipo de ejercicio adecuado para ese propósito.

Conviene también señalar que aunque el algoritmo anterior se expresaría prácticamente igual en Java, en cambio el alumno debería incluirlo en la rutina *main* de una clase y tratar las complicaciones propias de la entrada de datos.

En cuanto a la parte de FP, vamos a analizar por separado los cuatro principales aspectos que se tratan en ella: refinamiento por pasos sucesivos, recursividad, abstracción de datos y estructuras de datos dinámicas. El refinamiento por pasos sucesivos es una técnica propia de la programación estructurada, no aplicable en el contexto OO. He sido testigo de alumnos que debían escribir programas Java para un primer curso de programación, que aplicaban el refinamiento por pasos sucesivos dentro de la rutina *main* de una clase creada con el único objetivo de incluir dicha rutina, pero que no representaba una abstracción bien definida. Este tipo de clases ilustra cómo el uso de un lenguaje OO en IP o FP puede crear malos hábitos de programación.

La recursividad, al igual que pasaba con el estudio de la iteración, se estudia mediante esquemas algorítmicos en los que no tiene sentido aplicar el modelo computacional OO y es un concepto que todo programador debe conocer, pero para cuya comprensión la OO no aporta nada. Además, para explicar su funcionamiento interno es más natural hacerlo a través del concepto de rutina (y su invocación) que del concepto de método (y mensaje).

Existe un consenso en que el paradigma OO se fundamenta sobre el concepto de abstracción de datos. Una clase no es más que la implementación de un tipo abstracto de datos. Luego no admite discusión la conveniencia de estudiar este concepto y las estructuras de datos dinámicas introduciendo los conceptos de la OO. Pero entonces surge un grave problema como sería la necesidad del aprendizaje de un segundo lenguaje en un mismo curso: el alumno que comenzaba a defenderse con Pascal o Modula debería enfrentarse al aprendizaje de un lenguaje OO, como podría ser Java. ¿Es conveniente que un alumno de primer curso deba manejar dos lenguajes distintos, cada uno reflejando un paradigma diferente? Me parece excesivo ya que es necesaria una transición no trivial desde el

paradigma estructurado al paradigma OO. Además de la transición costosa de un lenguaje a otro, surgen otros problemas.

Si se utiliza un lenguaje OO en FP entonces los alumnos no conocerán el tipo puntero ya que trabajarían con semántica de referencia, pero esto no tendría excesiva importancia desde el punto de vista de la implementación de las estructuras de datos, ya que la forma de razonar es la misma se utilicen punteros explícitos o implícitos (referencias).

Si el lenguaje OO proporcionase genericidad, como es el caso de Eiffel, la definición de clases (módulos) que representasen estructuras de datos genéricas capaces de almacenar objetos de cualquier tipo (clases *contenedores*), no plantearía problemas. Sin embargo, si se utilizase un lenguaje sin genericidad como es el caso de Java, sería necesario usar la clase raíz *Object* como el tipo de objeto que se puede almacenar en el contenedor, siendo necesario manejar el concepto de polimorfismo. Este sería el mayor problema ya que el alumno tendría que manejar conceptos de la OO relacionados con la herencia: polimorfismo, clase abstracta, interface, regla de la asignación y necesidad de conversión de tipos (*narrowing* en Java), no siendo consciente de su significado.

En definitiva, puesto que no habría tiempo para impartir una buena introducción al paradigma OO al mismo tiempo que se enseñan los otros contenidos de FP, los alumnos deberían manejar los conceptos OO sin tener una clara comprensión de ellos. Además, insistir en el esfuerzo que supondría para el alumno aprender un nuevo paradigma y un nuevo lenguaje, lo que por otra parte dificultaría el estudio del resto de temas del curso. Por ello veo positivo que para la enseñanza de IP y FP se utilice Modula (también podría elegirse Pascal para IP y luego Modula para FP, ya que hay una continuidad entre ambos).

Nadie duda de la necesidad de que a lo largo de sus estudios universitarios, un estudiante de informática deberá conocer el paradigma imperativo y manejar lenguajes procedurales, como es el caso de C. Esto por dos motivos. Primero porque en algunas asignaturas, como es el caso de aquellas relacionadas con sistemas operativos y compiladores, es normalmente necesario el conocimiento del mencionado lenguaje. En segundo lugar porque todavía existen y se desarrollan numerosas aplicaciones con lenguajes procedurales, especialmente C, y por tanto el mercado demanda profesionales con esta formación.

Aceptando esta consideración, cabe plantearse la cuestión sobre si es más apropiado que un alumno conozca primero la POO y luego conozca la programación imperativa o es mejor hacerlo al revés. Por mi experiencia, no tengo dudas en afirmar que es mejor pasar de la programación imperativa a la POO, ya que el alumno comprende con facilidad las ventajas que aportan los conceptos OO.

Una vez el alumno ha recibido un curso de programación modular (por ejemplo con Modula) en el que ha visto que el concepto de módulo es diferente al de tipo abstracto de dato, es muy natural ver que una clase combina los conceptos de módulo y tipo. Del mismo modo se puede apreciar mejor el significado de la semántica de referencia, una vez se conoce la semántica de almacenamiento y el tipo puntero. El concepto de mensaje también se presenta de modo más elegante como un tipo de invocación de rutina en la que un parámetro juega un papel especial (objeto receptor). La herencia aparece como una nueva propiedad de esa combinación módulo-tipo que es la clase: un módulo-tipo que se define a partir de otro. El polimorfismo se puede presentar como una extensión del caso en el que una variable sólo se puede ligar a valores del tipo asociado en la declaración (monomorfismo). Y una vez se comprende el concepto de mensaje y polimorfismo surge de forma natural la necesidad de la ligadura dinámica, que se contrasta con la ligadura estática que conoce el alumno. En definitiva, parece que la POO se puede enseñar con más facilidad cuando se construye sobre los conocimientos y experiencia del paradigma imperativo.

Por otra parte, cuando el alumno ha diseñado/construido programas aplicando las técnicas estructuradas y modulares, comprende en toda su extensión los beneficios que aporta la OO en la mejora de la calidad del software (facilidad para el modelado, reutilización, extensibilidad,...).

En definitiva, creo que ninguno de los tres enfoques mencionados al principio de este apartado (*débil*, *fuerte* e *híbrido*) es adecuado para iniciar la enseñanza de la programación, sino que la POO debería ser incluida como una asignatura de tercer curso, una vez que el alumno ha recibido

una formación en programación procedural a través de una asignatura que combinase IP con FP, junto con otra que proporcionase una formación en algoritmia y estructuras de datos en el segundo curso. Tendría sentido que dicho curso de estructuras de datos utilizase un lenguaje OO para expresar los tipos abstractos de datos, de modo que el alumno al llegar al curso de OO del tercer año ya estuviese familiarizado con el lenguaje OO que se emplease en la parte práctica.

En el caso de la titulación de Ingeniero en Informática, debería existir en el plan de estudios una asignatura dedicada al estudio de un proceso software OO y a los patrones de diseño OO, en el sentido expuesto en [6], (éste puede ser un buen libro de texto, combinado con [4] para el estudio riguroso de los patrones de diseño básicos). Este es el enfoque que hemos seguido hasta el momento en nuestra facultad y que creo ha dado muy buenos resultados, así lo han entendido los alumnos en las evaluaciones realizadas sobre las asignaturas. Conviene señalar que en [1], el proceso OO y los patrones de diseño se consideran obligatorios dentro del curriculum.

En [7] se critica el enfoque que proponemos en este trabajo argumentando que los profesores de programación creen que lo más adecuado para sus alumnos es que aprendan los lenguajes en el mismo orden en que los aprendieron ellos: primero procedural, luego los lenguajes OO. Sin embargo, afirma que un buen día todo el mundo académico coincidió al considerar Pascal como el lenguaje adecuado para iniciar la enseñanza de la programación y nadie se echó las manos a la cabeza, así que si ahora todo el mundo considera la OO como el camino acertado, pues habrá que cambiar los esquemas. Además, entiende que los lenguajes procedurales se asimilan mejor una vez se conoce el paradigma OO. Mi experiencia dice lo contrario.

Creo que el enfoque expuesto en este trabajo refleja la idea de ir de lo más elemental a lo más complejo, como es habitual en la enseñanza. Primero se enseñan lenguajes monomórficos, luego polimórficos; tipos abstractos aislados antes que tipos que se definen a partir de otros estableciéndose relaciones de herencia; ligadura estática antes que la dinámica. Parece sensato ver la OO como una evolución del paradigma imperativo.

4. Conclusión

En este trabajo se ha intentado justificar la inconveniencia de utilizar un lenguaje OO en un primer curso de programación. Una vez se han establecido los objetivos de ese primer año se han planteado dos objeciones principales:

- los contenidos del primer curso se ajustan mejor a un lenguaje procedural, y si se utiliza un lenguaje OO el alumno usa los mecanismos OO sin una comprensión clara.
- Durante sus estudios, los alumnos deben conocer tanto el paradigma procedural como el OO, y es más apropiado y natural pasar del paradigma imperativo al OO que al revés.

Referencias

[1] ACM/IEEE, *Computing Curricula 2001*, Draft, March, 2000.

[2] Joel C. Adams, *Object-oriented Design: a five phase introduction to OO programming in CS1-2,* ACM SIGCSE'96. pp. 78-82, 1996.

[3] D. Arnow y G. Weiss, *Introduction to Programming using Java,* 2ª Edición, Addison-Wesley, 2000.

[4] E. Gamma et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[5] J. García Molina et al., *Introducción a la Programación*, ICE, Universidad de Murcia, Ed. Diego Marín, 1999.

[6] C. Larman, *Applying UML and Patterns: An introduction to object-oriented analysis and design*, Prentice-Hall, 1998.

[7] B. Meyer. *Towards an object-oriented curriculum.* Journal Object-Oriented Programming, pp 76-81, March, 1994.

[8] P. C. Scholl y J. P. Peyrin, *Esquemas Algorítmicos Fundamentales*, Masson, 1989.

[9] R. S. Wiener, *An Object-Oriented Introduction to Computer Science using Eiffel,* Prentice-Hall, 1996.