Metodología para la enseñanza aprendizaje de la lógica de la programación orientada a objetos

Leobardo López Román
Dpto. de Ingeniería Industrial y de Sistemas
Universidad de Sonora
Hermosillo, Sonora, 83000, México
llopez@industrial.uson.mx

Resumen

A la comunidad académica le llevó muchos años consolidar la forma correcta como los estudiantes deben aprender a programar computadoras. Primero, deben aprender la metodología de la programación, usando técnicas de diseño algorítmicas o pseudolenguajes; y después, deben aprender como implementarla usando un lenguaje de programación.

Con el desarrollo del lenguaje Java y la penetración que ha tenido como el primer lenguaje que muchos estudiantes están aprendiendo; y debido a la falta de una metodología apropiada, se está cayendo en el error de enseñar a programar directamente con el lenguaje Java, dejando de lado el desarrollo de la lógica; y se están formando programadores sin lógica.

Este autor tiene la convicción de que en la actualidad los estudiantes deben aprender un lenguaje orientado a objetos como Java; pero antes, deben desarrollar la lógica básica de la programación orientada a objetos. Es por ello que he desarrollado una metodología de la programación, usando pseudocódigo en forma similar como se usaba con la programación estructurada; pero ahora integrando la lógica básica de la programación con los conceptos y estructuras de la programación orientada a objetos.

En esta ponencia se presenta un resumen de la metodología original, la cual esta escrita con todo detalle en el libro [12] *Metodología de la programación orientada a objetos*; publicado por la editorial Alfaomega (www.alfaomega.com.mx), México, 2006; ISBN 970-15-1173-5. Y es distribuido en los países de habla hispana.

1. Motivación

En la actualidad muchos estudiantes y profesionales de la programación de computadoras están aprendiendo Java, que es un lenguaje orientado a objetos; sin embargo, muchos de ellos, no están aprendiendo a programar orientado a objetos; porque se les está enseñando prácticamente en forma directa con el lenguaje Java; y no se les está enseñando a "pensar"; es decir, no están desarrollando la lógica de la programación orientada a objetos.

La idea de este autor es que lo fundamental al aprender a programar computadoras es desarrollar la lógica necesaria para solucionar problemas en forma algorítmica, independientemente de algún lenguaje de programación; esto es, aprender a diseñar programas (algoritmos) usando un pseudolenguaje; y no hacerlo directamente con un lenguaje.

Metodología de la programación orientada a objetos, es un desarrollo que viene a coadyuvar en la solución de una necesidad largamente experimentada por la comunidad académica de la programación de computadoras; contar con un método que permita conducir la enseñanza-aprendizaje de la programación, mediante el uso de un pseudolenguaje de diseño de programas (algoritmos) orientados a objetos.

La metodología contiene en forma natural los conceptos, estructuras y filosofía que se han generado hasta estos tiempos en que la programación orientada a objetos y el lenguaje Java marcan la pauta de la programación de computadoras.

Esta metodología es el resultado de la integración y adaptación de varias técnicas, como son; los conceptos y estructuras de la programación orientada a objetos: objetos, clases, encapsulación, herencia y polimorfismo; con el diagrama de clases de UML (Unified Modeling Language); con la arquitectura modelo-vista-controlador; con algunos conceptos introducidos por el lenguaje Java; y con los conceptos y bases lógicas de la programación estructurada en pseudocódigo.

Dicha metodología permite diseñar programas o algoritmos orientados a objetos; y prepara a los estudiantes para que puedan aprender y comprender cualquier lenguaje orientado a objetos como Java.

2. Estado del arte

La actividad de programar computadoras ha tenido varias crisis, provocadas por el permanente aumento en la complejidad de las aplicaciones que deben enfrentarse, lo que provoca que las técnicas y estructuras que resultan adecuadas en un momento, con el paso del tiempo se vuelvan inadecuadas. Esta problemática ha dado origen a dos revoluciones: La primera, a la que se le llamó Programación Estructurada; permitió evolucionar desde programar de una forma "tradicional", casi artesanal; a programar de una mejor forma, que aportó las bases para sustentar la segunda revolución en la evolución de los paradigmas de programación; a lo que hoy se conoce como programación orientada a objetos.

2.1. Programación tradicional

En la década de los 60´s y principios de los 70´s se programaba en forma "tradicional", en esos tiempos sólo existían las estructuras lógicas: Secuenciación, If y For (que se conocía como Do en Fortran); y se utilizaban los diagramas de flujo como técnica de diseño de programas (algoritmos). Al aumentar la complejidad de las aplicaciones que se enfrentaban, esa forma de programar tuvo una severa crisis.

2.2. Programación estructurada

A principios de la década de los 70´s, surge un movimiento llamado programación estructurada que vino a añadir nuevas estructuras, técnicas y conceptos a la programación: Se añadieron las estructuras lógicas DO-UNTIL, DOWHILE y se formalizaron el IF-THEN, IF-THEN-ELSE y CASE. Se inventó el módulo, la función y el concepto de parámetros. Se desarrollaron nuevas técnicas de diseño de programas (algoritmos): Pseudocódigo, diagramas Warnier, diagramas Chapin, Jackson, Diseño estructurado de Yourdon, Top Down Design (Diseño descendente), entre otras; que vinieron a desplazar a la tradicional técnica de diagramas de flujo.

Fueron apareciendo nuevos lenguajes: Pascal, C, Cobol estructurado, Basic estructurado. Se estableció que se debe aprender a programar utilizando un pseudolenguaje, es decir, no enseñar directamente con un lenguaje. Y se estableció que se debe usar un estilo de programación que haga más entendible el algoritmo y el programa.

Nuevamente al aumentar la complejidad de las aplicaciones que se enfrentaban, esa forma de programar tuvo una severa crisis. Esto llevó a que siguiera evolucionando y se generaron los conceptos de programación modular, y luego el concepto de abstracción de datos, para dar paso al desarrollo de la programación orientada a objetos.

2.3. Programación orientada a objetos

Aunque la programación orientada a objetos (POO) aparece muchos años antes, es a mediados de los 90´s cuando se generaliza su uso. La POO añade a la programación una nueva estructura: el Objeto, con sus conceptos; objetos, clases, encapsulación, herencia y polimorfismo. Aparecen nuevas técnicas de diseño: Booch, Rumbaugh, Jacobson, Yourdon, UML (Unified Modeling Language), etc. Se desarrollan nuevos lenguajes: C++, Java, C#, etc.

3. Problemática de la enseñanza aprendizaje de la programación orientada a objetos

En los últimos años se ha insistido y ejercido una gran presión para que Java sea el primer y único lenguaje que los estudiantes deben aprender. En consecuencia, muchas instituciones educativas, están enseñando Java desde la fase introductoria a la programación de computadoras; eliminando una formación previa que permita el desarrollo de la lógica básica de la programación. Alguna gente dice que al estudiar el lenguaje Java va implícito el desarrollo de la lógica; y que la programación es mucho más fácil, rápida, agradable y avanzada

en Java que lo que anteriormente era la programación.

Alguna gente dice que cualquier persona que no sepa nada de programación, puede entender fácilmente los conceptos de la programación orientada a objetos; y estoy de acuerdo en parte, porque en un nivel abstracto cualquiera puede comprenderlos. Pero en el momento en que se debe implementar los objetos en instrucciones en un lenguaje de programación como Java, es donde se dan cuanta que "algo" falta. Porque un programa orientado a objetos, se compone por un conjunto de objetos; y cada objeto, por un conjunto de métodos que implementan las funciones del objeto; a algunos de esos métodos hay que enviarles datos a través de parámetros, para que establezcan y accedan los datos; y otros métodos realizan cálculos. De manera que ese "algo" que falta es la lógica básica de la programación; que consiste en: Tipos de datos; entero, real, cadena, arreglos, etcétera; Estructuras de control; secuenciación, if-then, if-then-else, switch, do-while, for, while; métodos (módulos y funciones definidas por el usuario); parámetros por valor y por referencia. Es por ello que digo que esos elementos y estructuras son la base de la programación orientada a objetos; y que una persona que no desarrolle esas bases, jamás podrá comprender cómo implementar los métodos de objetos que procesan datos.

Por lo anterior, pienso que la programación orientada a objetos no se está enseñando adecuadamente en las instituciones de educación. ¿Cuál es la causa? Que en la bibliografía existente sobre programación orientada a objetos; esta ausente la metodología de la programación orientada a objetos enfocada a estudiantes principiantes. En otras palabras, existen muchos libros sobre POO y UML, pero no están enfocados para niveles básicos de aprendizaje. Los libros que están enfocados para niveles básicos de aprendizaje son los libros de Java, que son excelentes manuales del lenguaje Java; pero no conducen el aprendizaje de la lógica básica de la programación inmersa en la programación orientada a objetos. Llevando a que los estudiantes "aprendan a programar" sin desarrollar la lógica. Esto significa que los estudiantes realmente están aprendiendo a codificar usando el lenguaje Java, que es un lenguaje orientado a objetos; pero, no están aprendiendo a programar orientado a objetos usando el lenguaje Java, que sería lo correcto. Porque programar es un proceso que implica diseñar el programa antes de codificarlo. Además, sabemos que si un estudiante aprende a programar directamente con el lenguaje que esta de moda, su mente queda "casada" con ese lenguaje; y cuando el lenguaje que esta de moda cambie, la formación que se le dio con el anterior lenguaje se convierte en deformación [12].

4. Metodología de la programación orientada a objetos

Con el objetivo de llenar ese vacío que ha dejado la bibliografía y con el propósito de coadyuvar en el mejoramiento de la enseñanza-aprendizaje de la programación de computadoras, he desarrollado esta metodología de la programación orientada a objetos.

La metodología se divide en dos partes; en la *primera parte*, que abarca del capítulo uno al nueve, se estudia la técnica pseudocódigo y su uso en el diseño de algoritmos pequeños que tienen una sola tarea o función, por tanto, se establece el uso de una clase y dentro de la clase el método principal, donde se plasma la lógica que soluciona el problema. En esta primera parte se da énfasis al desarrollo de la lógica básica de la programación usando pseudocódigo. Se estudian los tipos de datos, identificadores, operaciones de entrada, cálculo y salida.

Las estructuras de control: La secuenciación; la selección simple (IF THEN), doble (IF THEN ELSE) y múltiple (SWITCH); la repetición DO…WHILE, la repetición FOR y la repetición WHILE. Los arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales. Y por último de esta primera parte, se estudia cómo usar más de un método en la clase, en problemas que involucran a más de una tarea o función, métodos que no regresan valor (equivalente a módulos en la programación estructurada), métodos que regresan valor (equivalentes a funciones definidas por el usuario en la programación estructurada), parámetros por valor y por referencia.

Este autor tiene la convicción de que el estudiante debe desarrollar las bases lógicas de la programación; es por ello, que esta primera parte es lo que se estudia o estudiaba en un primer curso de lógica de programación con técnicas estructuradas, pero enfocando la estructura del algoritmo en forma apropiada a la programación orientada a objetos; usando una clase y dentro de la clase el método principal, entrenando a los estudiantes para que desarrollen las bases lógicas de la programación de computadoras.

A continuación se presenta un ejemplo, para mostrar una idea general de cómo se usa la primera parte de la metodología en la solución de una aplicación.

Problema 1:

Elaborar un algoritmo que permita leer un vector de diez números en un arreglo de 10 elementos; que lo imprima e imprima la media. Leer el vector en un método; calcular la media en otro método; e imprimir el vector en otro método. Utilizando parámetros.

```
Algoritmo MEDIA CON METODOS
 Clase MediaConMetodos
  1. Método principal
     a. Declaraciones
        Variables
           vector: Arreglo[10] Real
           promedio: Real
     b. Llamar leerVector(vector)
     c. promedio = calcularMedia(vector)
     d. Llamar imprimirVector(vector)
     e. Imprimir promedio
     f. Fin Método principal

  2. Método leerVector(Ref vec: Arreglo[10] Real)
     a. Declaraciones
        Variables
           n: Entero
     b. FOR n=0; n<=9; n++
           1. Solicitar elemento vec[n]
           2. Leer vec[n]
     c. ENDFOR
     d. Fin Método leerVector

  3. Método calcularMedia(Ref v: Arreglo[10]
                                  Real): Real
     a. Declaraciones
        Variables
           sumatoria, prom: Real
           i: Entero
     b. sumatoria = 0
     c. FOR i=0; i<=9; i++
           1. sumatoria = sumatoria + v[i]
     d. ENDFOR
     e. prom = sumatoria / i
     f. return prom
     g. Fin Método calcularMedia

  4. Método imprimirVector(Ref vect: Arreglo[10]
                                     Real)
     a. Declaraciones
        Variables
           x: Entero
     b. FOR x=0; x<=9; x++
           1. Imprimir vect[x]
     c. ENDFOR
     d. Fin Método imprimirVector
 Fin Clase MediaConMetodos
Fin
```

Explicación:

El algoritmo tiene una clase; en la cual se tienen cuatro métodos. El método principal, en el que se define la variable vector, como un arreglo de 10 elementos; y la variable promedio. Enseguida llama al método leerVector(vector), enviando vector como parámetro y conectándolo con el parámetro por referencia vec; en el cual lee los 10 números.

Luego llama al método calcularMedia(vector) enviando vector como parámetro y conectándolo con el parámetro v; del cual calcula la media y la devuelve para colocarla en promedio.

A continuación llama imprimirVector(vector) enviando vector como parámetro y conectándolo con el parámetro vect; el cual imprime. Y finalmente imprime el promedio.

En la *segunda parte* de la metodología, que abarca del capítulo diez al dieciséis, es donde se estudian de lleno los conceptos de la programación orientada a objetos, integrándolos con el concepto de diagrama de clases de UML (Unified Modeling Language), con la arquitectura modelo-vista-

controlador, con las estructuras estudiadas en los primeros nueve capítulos y la incorporación de los conceptos de la programación orientada a objetos en la técnica pseudocódigo, logrando una metodología de la programación que permite diseñar algoritmos orientados a objetos.

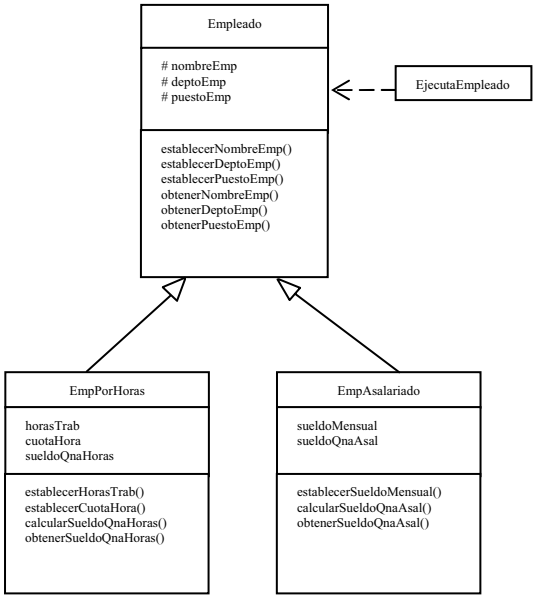
Lo relevante del método es que enseña a programar computadoras utilizando un pseudolenguaje (pseudocódigo), es decir, sin utilizar la computadora directamente. Esto permite desarrollar las capacidades mentales lógicas que una persona debe tener para programar computadoras y sienta las bases de disciplina y buena estructura. Este enfoque se le dificulta a mucha gente, sin embargo, hay que enfrentarlo, porque siendo la programación una actividad intelectual que requiere mucha creatividad, capacidades de abstracción, de análisis, y de síntesis; éstas no se pueden desarrollar operando un lenguaje en la computadora, sino ejercitando la mente con una metodología apropiada. A continuación se presenta un ejemplo, para mostrar una idea general de cómo se usa la segunda parte de la metodología en la solución de una aplicación.

Problema 2:

En cierta empresa se tienen empleados; los empleados se dividen en dos tipos: empleados por horas, a los que se les paga de acuerdo al número de horas trabajadas y a una cuota que se les paga por hora. El otro tipo son los empleados asalariados, a quienes se les paga de acuerdo a un sueldo fijo mensual. Por cada empleado se tienen los datos: Nombre, Departamento, Puesto; si es empleado por horas, el número de horas que trabajó y la cuota que se le paga por hora; si es empleado asalariado, el sueldo mensual que se le paga. Elaborar un algoritmo que permita leer los datos de los empleados e imprimir el nombre y el sueldo quincenal de cada empleado.

A continuación se tiene la solución en dos partes: En la primera, se diseña el diagrama de clases, que contiene la estructura general del programa (algoritmo); y después, en la segunda parte, se diseña el algoritmo que contiene la lógica que soluciona el problema usando pseudocódigo.

Diagrama de clases



Algoritmo CALCULA SUELDOS DE EMPLEADOS

Clase Empleado

1. Declaraciones
   Datos
   # nombreEmp: Cadena
   # deptoEmp: Cadena
   # puestoEmp: Cadena
2. Método establecerNombreEmp(nom: Cadena)
   a. nombreEmp = nom
   b. Fin Método establecerNombreEmp
3. Método establecerDeptoEmp(dep: Cadena)
   a. deptoEmp = dep
   b. Fin Método establecerDeptoEmp
4. Método establecerPuestoEmp(pue: Cadena)
   a. puestoEmp = pue
   b. Fin Método establecerPuestoEmp
5. Método obtenerNombreEmp(): Cadena
   a. return nombreEmp
   b. Fin Método obtenerNombreEmp
6. Método obtenerDeptoEmp(): Cadena
   a. return deptoEmp
   b. Fin Método obtenerDeptoEmp

```
   7. Método obtenerPuestoEmp(): Cadena
      a. return puestoEmp
      b. Fin Método obtenerPuestoEmp
Fin Clase Empleado

Clase EmpPorHoras hereda de Empleado
 1. Declaraciones
    Datos
        horasTrab: Entero
        cuotaHora: Real
        sueldoQnaHoras: Real
 2. Método establecerHorasTrab(horasTr: Entero)
    a. horasTrab = horasTr
    b. Fin Método establecerHorasTrab
 3. Método establecerCuotaHora(cuotaHr: Real)
    a. cuotaHora = cuotaHr
    b. Fin Método establecerCuotaHora
 4. Método calcularSueldoQnaHoras()
    a. sueldoQnaHoras = horasTrab * cuotaHora
    b. Fin Método calcularSueldoQnaHoras
 5. Método obtenerSueldoQnaHoras(): Real
    a. return sueldoQnaHoras
    b. Fin Método obtenerSueldoQnaHoras
Fin Clase EmpPorHoras

Clase EmpAsalariado hereda de Empleado
 1. Declaraciones
    Datos
        sueldoMensual: Real
        sueldoQnaAsal: Real
 2. Método establecerSueldoMensual(sdo: Real)
    a. sueldoMensual = sdo
    b. Fin Método establecerSueldoMensual
 3. Método calcularSueldoQnaAsal()
    a. sueldoQnaAsal = sueldoMensual / 2
    b. Fin Método calcularSueldoQnaAsal
 4. Método obtenerSueldoQnaAsal(): Real
    a. return sueldoQnaAsal
    b. Fin Método obtenerSueldoQnaAsal
Fin Clase EmpAsalariado

Clase EjecutaEmpleado
 1. Método principal
   a. Declaraciones
       Variables
          nomEmp, depto, puesto: Cadena
          hrsTra, tipoEmp: Entero
          cuoHr, sdoMen: Real
          desea: Carácter
   b. DO
      1. Imprimir Menu y solicitar tipo de
                  empleado
            Tipos de empleado
          1. Empleado por horas
          2. Empleado asalariado
            Teclee tipo:
      2. Leer tipoEmp
      3. Solicitar Nombre, departamento y puesto
      4. Leer nomEmp, depto, puesto
      5. IF tipoEmp = 1 THEN
         a. Crear objeto
            EmpPorHoras objEmp = new
                              EmpPorHoras()
         b. Solicitar número de horas trabajadas
                    y cuota por hora
         c. Leer hrsTra, cuoHr
         d. Establecer
       objEmp.establecerNombreEmp(nomEmp)
       objEmp.establecerDeptoEmp(depto)
       objEmp.establecerPuestoEmp(puesto)
       objEmp.establecerHorasTrab(hrsTra)
       objEmp.establecerCuotaHora(cuoHr)
         e. Calcular
            objEmp.calcularSueldoQnaHoras()
         f. Imprimir
               objEmp.obtenerNombreEmp()
               objEmp.obtenerDeptoEmp()
               objEmp.obtenerPuestoEmp()
               objEmp.obtenerSueldoQnaHoras()
      6. ELSE
         a. Crear objeto
            EmpAsalariado objEmp = new
                              EmpAsalariado()
         b. Solicitar sueldo mensual
         c. Leer sdoMen
         d. Establecer
      objEmp.establecerNombreEmp(nomEmp)
      objEmp.establecerDeptoEmp(depto)
      objEmp.establecerPuestoEmp(puesto)
      objEmp.establecerSueldoMensual(sdoMen)
         e. Calcular
            objEmp.calcularSueldoQnaAsal()
         f. Imprimir
               objEmp.obtenerNombreEmp()
               objEmp.obtenerDeptoEmp()
               objEmp.obtenerPuestoEmp()
               objEmp.obtenerSueldoQnaAsal()
      7. ENDIF
      8. Preguntar "¿Desea procesar otro
                              empleado(S/N)?"
      9. Leer desea
```

c. WHILE desea = “S”
d. Fin Método principal
Fin Clase EjecutaEmpleado
Fin

Explicación:

En el diagrama de clases, se esquematiza la estructura general de la solución. Se tiene la clase controlador EjecutaEmpleado, la cual utiliza al modelo, que está formado por tres clases jerarquizadas: La clase Empleado que es la superclase o clase principal; La clase EmpPorHoras que es una subclase que se deriva de la superclase Empleado; y La clase EmpAsalariado que es una subclase que se deriva de la superclase Empleado.

En el algoritmo se diseña la lógica de cada una de las clases usando pseudocódigo. En la clase Empleado se declaran los datos: nombreEmp, deptoEmp y puestoEmp; y los métodos establecerNombreEmp(), establecerDeptoEmp(), establecerPuestoEmp(), obtenerNombreEmp(), obtenerDeptoEmp() y obtenerPuestoEmp(); para establecer y obtener cada uno de los datos respectivamente. Empleado es la superclase que se usa para derivar subclases, a través del mecanismo de herencia; es por ello que a sus datos se le antepuso el símbolo # el cual indica que el dato es protegido (protected); los datos deben ser protegidos para que se puedan heredar.

En la clase EmpPorHoras que es una subclase que se deriva de Empleado, se declaran los datos: horasTrab, cuotaHora y suedoQnaHoras; y los métodos establecerHorasTrab(), establecerCuotaHora(), calcularSueldoQnaHoras() y obtenerSueldoQnaHoras(); para establecer las horas trabajadas, establecer la cuota por hora, calcular el sueldo quincenal del empleado por horas y obtener el valor del sueldo quincenal respectivamente. Al derivarse de Empleado, EmpPorHoras hereda los datos y métodos de Empleado a través del mecanismo de herencia.

En la clase EmpAsalariado que es una subclase que se deriva de Empleado, se declaran los datos: sueldoMensual y suedoQnaAsal; y los métodos establecerSueldoMensual(), calcularSueldoQnaAsal() y obtenerSueldoQnaAsal(); para establecer el sueldo mensual, calcular el sueldo quincenal del empleado asalariado y para obtener el sueldo quincenal respectivamente. Al derivarse de Empleado, EmpAsalariado hereda los datos y métodos de Empleado a través del mecanismo de herencia.

En la clase EjecutaEmpleado que es la clase controladora, es donde se establece la lógica que soluciona el problema utilizando las otras clases. Se declaran las variables necesarias para dar entrada a los datos. Se establece un ciclo DO…WHILE que permitirá procesar varios empleados. En el proceso de cada empleado, se debe indicar el tipo de empleado que es; y con el uso de un IF-THEN-ELSE se procesa de la forma que corresponda. Enseguida se genera el objeto correspondiente, luego se interactúa con el operador para que introduzca los datos, los cuales se leen y se llevan al objeto a través de los métodos setters; luego se calcula el sueldo llamando al método correspondiente; y finalmente se obtienen del objeto los datos que se van a imprimir, a través de los métodos getters.

Para conocer la metodología en detalle, se recomienda la lectura del libro [12].

5. Conclusión

Actualmente hay una tendencia a utilizar Java como primer lenguaje directamente con el concepto orientado a objetos, sin profundizar en el desarrollo de las bases lógicas de la programación, esto puede resultar muy dañino, porque vamos a generar programadores buenos para codificar usando lenguajes, pero sin bases lógicas; es decir, programadores que no saben programar.

Todo estudiante de sistemas, computación o informática debe aprender a programar orientado a objetos en lenguaje Java; pero para lograrlo, primero debe desarrollar las habilidades mentales lógicas necesarias; porque la programación es lógica y debe ser independiente de algún lenguaje de programación.

Ahora, la comunidad académica tiene a su disposición un libro [12] donde se presenta una metodología enfocada al desarrollo de las bases lógicas de la programación; esto es, para aprender a diseñar programas (algoritmos) orientados a objetos usando un pseudolenguaje (pseudocódigo). Dicha metodología, permite preparar a los estudiantes para que puedan aprender y comprender cualquier lenguaje orientado a objetos como Java, UML, etcétera.

Referencias

[1] Bell, D. y Parr, M., *Java para estudiantes Tercera edición*, México, Prentice Hall, 2003.

[2] Booch, G., *Análisis y diseño orientado a objetos con aplicaciones Segunda edición*, USA, México, Addison-Wesley/Díaz de Santos, 1996.

[3] Booch, G., Rumbaugh, J. y Jacobson, I., UML *El lenguaje unificado de modelado*, España, Addison Wesley, 1999.

[4] Ceballos, F.J., *Java 2 Curso de programación*, México, Alfaomega-Rama, 2000.

[5] Deitel, H.M. y Deitel, P.J., *Como programar en Java Quinta edición*, México, Pearson Prentice Hall, 2004.

[6] Horatman, C. S y Cornell, G,. *Core Java 2 Volume I Fundamentals 5th edition*. USA. The Sun Microsiystem Press Prentice Hall, 2000.

[7] Horton, I., *Beginning Java 2*, USA, Wrox Press Inc., 2000.

[8] Jacobson, I., Booch, G. y Rumbaugh, J., *UML El proceso unificado de desarrollo de software*, España, Addison Wesley, 2000.

[9] Joyanes, A.L., *Programación orientada a objetos Segunda edición*, España, Osborne Mc Graw Hill, 1998.

[10] Joyanes, A.L. y Fernández, A.M., *Java 2 Manual de programación*, España, Mc Graw Hill, 2001.

[11] Lemay, L. y Cadenhead, R., *Aprendiendo Java en 21 días*, México, Pearson Prentice Hall, 1999.

[12] López, R. L. *Metodología de la programación orientada a objetos,* Alfaomega (www.alfaomega.com.mx), México, 2006. ISBN 970-15-1173-5.

[13] Meyer, B., *Construcción de software orientado a objetos Segunda edición*, España, Prentice Hall, 1999.

[14] Rumbaugh, J., Jacobson, I. y Booch, G., *UML El lenguaje unificado de modelado. Manual de referencia*, España, Addison Wesley, 2000.

[15] Schildt, Herbert, *Fundamentos de programación en Java 2*, Colombia, Osborne Mc Graw Hill, 2002.

[16] Schildt, Herbert, *Java 2 Manual de referencia*, España, Osborne Mc Graw Hill, 2001.

[17] Wu, T.C*., Introducción a la programación orientada a objetos en Java*, España, Mc Graw Hill, 2001.

[18] http://java.sun.com

Nota de referencia:
Una ponencia similar se presentó como Artículo Invitado en la 5ta. Conferencia Iberoamericana en Sistemas, Cibernética e Informática CISCI 2006, organizada por el International Institute of Informatics and Systemics, en Orlando, Florida, EE.UU.; y se está presentando esta ponencia en JENUI 2007, porque es para la difusión del contenido del libro referenciado.