UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

PROPUESTA DE UNA METODOLOGÍA PARA LA ENSEÑANZA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

OBED ISAAC SANTISTEBAN GARCÍA

ASESORADO POR: ING. RENÉ ORNELIS HOIL

GUATEMALA, MAYO DE 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

PROPUESTA DE UNA METODOLOGÍA PARA LA ENSEÑANZA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

TRABAJO DE GRADUACIÓN

PRESENTADO A JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

OBED ISAAC SANTISTEBAN GARCÍA
ASESORADO POR: ING. RENÉ ORNELIS HOIL

AL CONFERÍRSELE EL TÍTULO DE
INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MAYO DE 2005

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Sydney Alexander Samuels Milson
VOCAL I	Ing. Murphy Olympo Paiz Recinos
VOCAL II	Ing. Amahán Sánchez Álvarez
VOCAL III	Ing. Julio David Galicia Celada
VOCAL IV	Bachiller Kenneth Issur Estrada Ruiz
VOCAL V	Bachiller Elisa Yazminda Vides Leiva
SECRETARIO	Ing. Carlos Humberto Pérez Rodríguez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Herbert René Miranda Barrios
EXAMINADOR	Ing. Hugo Juárez
EXAMINADOR	Ing. Jorge Luis Álvarez
EXAMINADOR	Inga. Marlen Moralez
SECRETARIA	Inga. Gilda Marina Castellanos de Illescas

HONORABLE TRIBUNAL EXAMINADOR

Cumpliendo con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

PROPUESTA DE UNA METODOLOGÍA PARA LA ENSEÑANZA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha febrero de 2004.

Obed Isaac Santisteban García

AGRADECIMIENTOS

A Dios	No hay palabras que revelen completamente mi agradecimiento, por su perdón, por su misericordia, por la promesa, por todo, alabado seas, Padre Eterno y Dios Todopoderoso.
A mis padres	Por haberme dado con su apoyo la confianza, con su ejemplo el coraje y con su amor mi alimento.
A mis hermanos	Por ser parte de mi vida, firmes en mis flaquezas y fuertes en sus ternuras.
A mis compañeros de universidad	Por ser parte del esfuerzo en la lucha por aprender.
A mis amigos	Por ser el apoyo de este esfuerzo y la parte que mejor recuerdo de los años de estudio.
A mis abuelos	Por su amor y ser parte de mí.
A mis catedráticos	Por su instrucción y ayuda.
A la Universidad	Por su tradición.
A mi patria	Por la inversión realizada en mi educación.

DEDICATORIA

A mi madre, Amanda García
Con su abnegación, esfuerzo, lucha y amor, me ha llevado a donde estoy.

A mi padre, Jorge Santisteban
Con su ayuda, mi fuerza se hace más grande.

A mi abuela, Ovidia
Con su constancia, valor y fuerza, forjadora de mi devoción.

A mi abuelo, Pedro
Con su calidez y paciencia, que Dios lo tenga en su gloria.

A mi esposa, Cristy
Con su amor, el motivo de mi vida, mi dulce compañera.

A mi hija, Gimena
La fuente de mi felicidad, la gracia en mi corazón, mi niña.

INDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

Clase	Es el ámbito de definición de un conjunto de objetos.
Cohesión	Reunión o adherencia de las cosas entre sí o la materia de que están formadas.
Concurrente	Que coinciden en un mismo tiempo.
Constructor	Mecanismo usado en los lenguajes orientados a objetos que permite crear objetos y llevarlos a un estado válido.
Destructor	Mecanismo usado en los lenguajes orientados a objetos que permite liberar los recursos obtenidos por los objetos cuando fueron creados.
Encapsular	Colocar en una cápsula que aísla del ambiente externo.
Interfaz	Conexión entre dos sistemas independientes.
Lenguaje híbrido	Aquel lenguaje que permite la realización de programas en varios paradigmas a la vez.
Objeto	Unidad atómica que encapsula estado y comportamiento.
OOP	*Object oriented programming* (programación orientada a objetos)
Precedencia	Orden establecido de evaluación.
Recursiva	En la ejecución de un programa es volver al lugar de donde se originó.
Software	Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en la computadora.

RESUMEN

Este trabajo de graduación presenta una recopilación de las características de los paradigmas de programación más relevantes, mostrando el esquema conceptual de cada uno de ellos y profundizando más en el de la programación orientada a objetos. Se expone de forma sucinta a partir de la investigación realizada en los diferentes libros y documentos de la *Web*, con el afán de conformar una propuesta para la metodología de enseñanza de la programación en este paradigma.

Se propone el método de enseñanza constructivista, con el apoyo de lecturas activas y patrones pedagógicos, que intenten construir adecuadamente el modelo mental en el alumno y evitar la confusión que provoca el cambio de paradigma en la enseñanza de la programación, cuando los principios de ingeniería de software pueden inculcarse desde el inicio mediante la enseñanza del paradigma orientado a objetos.

Se propone enseñar los fundamentos de la programación y las cualidades de diseño, viendo la programación como un proceso de modelado, tomando en cuenta que se quiere construir un esquema adecuado para razonar en estos términos, de forma que se centre en la ingeniería más que en el algoritmo.

Como consecuencia, se da el contenido propuesto para los cursos de Introducción a la programación I y II, indicando el objetivo y el contenido de cada sesión, de forma que se dé mayor importancia la enseñanza con el ejemplo y promoción del descubrimiento y el sentido crítico en el estudiante.

OBJETIVOS

1. Proponer una metodología para la enseñanza de la programación orientada a objetos, que sea aplicable en la Universidad de San Carlos de Guatemala.

2. Plantear contenidos de los cursos de introducción a la programación I y II que apliquen la propuesta de la enseñanza de la programación orientada a objetos.

3. Fundamentar las causas que ameritan el cambio en la forma de enseñar la programación orientada a objetos.

INTRODUCCIÓN

En este trabajo se plantea una metodología para la enseñanza de la programación orientada a objetos, la cual concluye en los contenidos aplicables en los cursos de introducción a la programación I y II en la carrera de ingeniería de sistemas.

Este planteamiento surge como respuesta al problema encontrado por los estudiantes cuando se enfrentan al cambio de paradigma ocasionado por el paso del paradigma procedural al orientado a objetos y la aparente dificultad identificada por los catedráticos al enseñar el paradigma orientado a objetos.

En el capítulo I, se describe el marco teórico de cada uno de los paradigmas y su respectivo enfoque, identificando las bases sobre las cuales han sido planteados; luego se da un resumen de las teorías de aprendizaje y los métodos educativos, para finalizar con la descripción de patrones pedagógicos recientemente usados en la enseñanza de la programación.

En el capítulo II, se realiza un análisis de la enseñanza de la programación orientada a objetos y se comenta la evaluación de los estudiantes ya educados en el tema, de forma que sean la base para plantear soluciones a los problemas encontrados.

En el capítulo III, se plantean los fundamentos de la propuesta; entre ellos, la visión de la pintura completa y las técnicas de modelado. Luego se plantean los fundamentos del contenido; entre ellos, el modelo mental, las concepciones inválidas, las metáforas, el enfoque y la metodología de realización de cada tarea.

En el capítulo IV, se plantea el contenido de los cursos de introducción a la programación I y II, los cuales aplican la propuesta, indicando la participación del catedrático en cada sesión.

1 MARCO TEÓRICO

A continuación se detallará cada uno de los conceptos que serán usados como marco de referencia para la realización de la propuesta. De todos ellos se presentan, de forma más profunda, aquellos aspectos que son de utilidad para sustentar la propuesta.

1.1 Paradigma

Según el diccionario de la Real Academia Española proviene del latín paradigma, y este del griego παρά̓δειγμα. Se tienen 3 definiciones:

- Ejemplo o ejemplar.
- Cada uno de los esquemas formales en que se organizan las palabras nominales y verbales para sus respectivas flexiones.
- Conjunto cuyos elementos pueden aparecer alternativamente en algún contexto especificado; por ejemplo, niño, hombre, perro, pueden figurar en El -- se queja.

Según la primera definición, un paradigma se basa en un ejemplo. De acuerdo con la segunda definición, se resume que el paradigma es un esquema formal en que se organizan las palabras asociadas a nombres y las palabras asociadas a acción para sus respectivas alteraciones o cambios; y por la tercera definición, un paradigma es algo que bajo un contexto se da por sentado. Y es por eso que encontramos expresiones como “Hay que romper el paradigma”. Según la secuencia de definiciones, podemos decir que un ejemplo lo hacemos formal y luego lo damos por sentado.

1.2 Paradigma de la programación

Colección de conceptos que guían el proceso de construcción de un programa, determinando su estructura. Estos conceptos controlan la forma en que pensamos y formulamos los programas.

1.2.1 Imperativo

En la programación arriba-abajo se concibe el programa como un proceso por descubrir. Se piensa del problema como un proceso para ser descompuesto, se parte El problema en problemas más pequeños que cumplan las siguientes características:

- El problema más pequeño también es un problema.
- El problema más pequeño puede ser solucionable, aunque no se tenga una solución inmediata.
- Si todos los problemas más pequeños en la descomposición se pueden resolver, entonces se conoce una composición que al integrarse lleva tanto a la solución de los problemas pequeños juntos como a la solución del problema original.

El método busca aplicar la anterior técnica de forma recursiva a los problemas más pequeños, hasta que el paso 2 se degrada a "tengo una solución a este problema". Entonces se escribe un procedimiento o función para resolver el problema a ese nivel, y luego se aplica la composición indicada en el paso 3 para obtener un procedimiento o función que resuelve los problemas mayores.

Hay una jerarquía de funciones donde las soluciones a los problemas pequeños son jerarquizadas entre las soluciones para los problemas mayores; como consecuencia, la solución es como un árbol y en cada nodo del mismo se inicia un procedimiento o una función.

La naturaleza de la solución para un problema es un proceso o un conjunto de acciones, de forma que el paradigma se enfoca primeramente sobre los algoritmos y entonces hace que las estructuras de datos encajen en el proceso de desarrollo.

Un programa tradicional, que sigue el paradigma estructurado, se basa fundamentalmente en la ecuación de Wirth:

Algoritmos + Estructuras de datos = Programas

Esta ecuación trata por separado los algoritmos y los datos. De esta forma las funciones o procedimientos que tratan estos datos los van procesando y pasando de unos a otros hasta obtener el resultado deseado.

Se basa en el modelo de computador más extendido, el llamado modelo de Von Neumann que define una máquina capaz de ejecutar una serie de instrucciones en secuencia, una después de la otra. Estas instrucciones han de estar almacenadas en memoria principal para ser leídas y ejecutadas por la unidad de control.

1.2.2 Funcional

En la programación funcional, un programa se concibe como un conjunto de funciones que deben ser evaluadas para obtener un resultado. Se le denomina computación basada en cálculo. Consiste en definir funciones que buscan la solución del problema mediante la evaluación; tanto los argumentos como el resultado de una función pueden ser otra función o incluso la misma.

Está basada principalmente en el modelo de computación conocido como cálculo lambda (λ), el cual fue inventado por Alonzo Church en 1934, y que se basa en 2 principios básicos:

- Definir alguna(s) función(es) de un solo argumento y con un cuerpo específico, denotado por la terminología λx.B, en donde x determina el parámetro o argumento formal y B representa el cuerpo de la función, es decir f(x) = B.

- Aplicar alguna de las funciones creadas sobre un argumento real (A); lo que es conocido también con el nombre de reducción, y que no es otra cosa que sustituir las ocurrencias del argumento formal (x) que aparezcan en el cuerpo (B) de la función con el argumento real(A), es decir: (λx.B) A.

Ejemplo: si vemos el siguiente ejemplo: (λx. (x + 5)) 3, nos indica que en la expresión x + 5, debemos sustituir el valor de x por 3.

Cuando ya no es posible reducir una función, se dice que ésta se encuentra en su estado normal, o lo que es lo mismo, que se ha encontrado el valor de la función. Este último siempre dependerá únicamente de los argumentos y siempre tendrá la consistencia de regresar el mismo valor para los mismos argumentos.

Cuando una función puede servir como argumento de otra función o puede almacenarse como un valor en una estructura de datos, o bien cuando el resultado de una función es otra función, estamos hablando de funciones de orden superior.

Con esto el paradigma funcional permite realizar demostraciones matemáticas sobre las propiedades de una función. Mediante la transparencia referencial se asegura que el valor de una expresión no dependa del orden de evaluación de la misma, sino de sus argumentos; y mediante la reducción se reemplaza una expresión por otra equivalente más simple hasta llegar a una expresión canónica, a la cual no se le pueden aplicar más reglas.

Algunos lenguajes que utilizan este paradigma son: ASpecT, Caml, FP, J, Mercury, ML, OPAL, Scheme, Erlang, NESL, Oz, Pizza, Sisal, Gofer, Haskell, Hope, Hugs, Miranda, Clean e Id.

1.2.3 Lógico

En este paradigma el programa se ve como un formalismo de representación del conocimiento e inferencia mediante predicados.

Un predicado es un símbolo cuyo valor se encuentra en el dominio lógico (verdadero o falso) y representa una cualidad semántica en un cierto contexto acerca de las relaciones entre entidades o se puede definir como aserción lógica que representa el conocimiento que se tiene de un ambiente determinado.

Se especifica el problema a resolver mediante la creación de una teoría; en las relaciones definidas por la teoría no existe distinción de entrada o salida entre sus parámetros, y una consulta dada puede resultar en más de una respuesta; es decir, en la programación lógica se especifica un lenguaje y sus relaciones existentes, y su resultado no es determinístico.

Al plantear el conjunto de cláusulas en una estructura definida se tiene una representación del conocimiento. Este conocimiento se usa como base para inferencia de consultas referentes a este conocimiento.

Por eso un programa es una conjunción de cláusulas definidas con cuantificadores universales implícitos. Y la pregunta a plantear es una conjunción de literales positivas con cuantificación existencial implícita.

A fin de obtener un resultado, se evalúan las reglas que satisfacen los planteamientos, haciendo uso de varios mecanismos de control, tales como:

- El razonamiento hacia atrás, que consiste en que partiendo de un objetivo se buscan los hechos y reglas que nos lleven a demostrarlo.
- Búsqueda en profundidad, en el cual se analizan todas las posibles rutas que nos permitan demostrar los planteamientos, recorriendo las reglas y los hechos en el orden en que aparecen en el programa, es decir de arriba a abajo y de izquierda a derecha.
- Reevaluación, que es cuando se recorre un camino, cuyo inicio fue con base en la instanciación de una variable y éste conduzca a un punto en el que no se pude ir más allá; entonces, se retrocede hasta la regla que dio origen a la instanciación y se selecciona otra ruta alternativa.
- El operador de corte "!" y el operador "not" son operadores especiales que sirven para hacer más eficientes las búsquedas, limitando los recorridos.

1.2.4 Orientado a objetos

En este paradigma el programa se ve como un gráfico de elementos interactuando, cada uno de los cuales actúa como un servidor para alguna pieza de información u otro servicio; estos elementos son llamados objetos. Los objetos en el sistema son pensados como datos (datos de alto nivel) que encapsulan los algoritmos tan bien como los datos de bajo nivel; cada objeto puede también actuar como un cliente de otros objetos cuando éste necesita colaborar con aquellos otros para alcanzar el cumplimiento de su propio servicio.

En un sistema de un solo procesador, el flujo de control (el procesador) se mueve con los requerimientos de servicios (mensajes), así cuando un objeto hace un requerimiento

de un servicio de otro que requiere, espera a que el servicio sea completado y cualquier resultado sea retornado.

El procesador entonces se mueve al servidor para que éste pueda cumplir su requerimiento. Cuando el servidor completa su tarea, éste pasa cualquier resultado, junto con el procesador, de vuelta al cliente quien requiere el servicio.

La forma de construir un programa orientado a objetos es descubriendo primero los objetos mediante un proceso de simulación; los objetos de software en el sistema deben modelar los objetos del mundo real en el sistema modelado.

Una vez se tengan objetos candidatos, se les asigna responsabilidades, que resultan ser los servicios. Al conocer todo de una interacción en la cual el objeto actúa como servidor, se conoce el conjunto completo de responsabilidades de este objeto. Cuando varios objetos tienen el mismo comportamiento, entonces dan origen a una clase que los describirá, y se está en el punto de poder definir las responsabilidades y de modelar éstas como métodos.

Como primer paso, se necesita decidir, para cada clase, qué información necesita ser mantenida en ella, para que el servicio pueda ser realizado. Primero está la información que necesita ser mantenida mientras el servicio no se está realizando. Estos datos representan la información de la instancia de la clase (los datos de bajo nivel). Cuando los objetos de la clase compartirán información entre sí, entonces se definen estos datos a este nivel.

1.2.5 Paralelo

En este paradigma el programa se ve como un conjunto de elementos de procesamiento de información cooperando concurrentemente para cumplir un objetivo

común de forma eficiente. Cada elemento de procesamiento de información se denomina hilo (proceso). Bajo el modelo conceptual de una computadora paralela, se ve como una colección de procesadores típicamente del mismo tipo, interconectados de alguna manera para permitir la coordinación de sus actividades y el intercambio de datos.

Existen dos estilos fundamentales para la construcción de estos programas:

- El control de flujo: se crean procesos o hilos diferentes que operan sobre sus propios datos, con lo cual el programa se ve como un conjunto de procesos.
- El paralelismo en datos: se crean procesos o hilos iguales que aplican las mismas instrucciones sobre datos propios, y los datos del programa se dividen en procesos similares.
-

Con lo cual surgen varios pasos en la creación de programas paralelos:

- Descomposición: divide el computo en tareas que serán divididas en procesos. Las tareas pueden ser estáticas o dinámicas y pueden variar con el tiempo. Se busca tener suficientes tareas para mantener a los procesadores ocupados, pero no en exceso. Además, identifica la concurrencia y decide el nivel en el cual se explotará la misma.

- Asignamiento o agrupamiento: busca especificar la división del trabajo entre los procesos mediante la reducción de costos de comunicación y sincronización, así como el balance de carga.
- Orquestación: busca sincronizar adecuadamente las tareas para evitar tiempos perdidos, es decir, calendarizarlas adecuadamente. También busca la identificación de los datos, tanto su estructura como el protocolo de mensajes.

- Mapeo: es la decisión de asignación de procesos a determinados procesadores para maximizar la utilización. En algunos casos esta decisión queda a criterio del sistema operativo.
-

La ejecución de los programas entonces requiere coordinación de sincronización, la cual habilita la cooperación estableciendo que los eventos sucedan en algún orden, y permite la competencia de los procesos por recursos compartidos tales como exclusión mutua e inter bloqueos.

Algunos de los lenguajes que utilizan este paradigma son: Fortran 90, High Performance Fortran, C*, OCCAM y Linda.

1.3 La programación orientada a objetos

1.3.1 Breve historia

Inicia a finales de los 60 y comienza a cobrar importancia a lo largo de los 70 e incrementa su popularidad a mediados de los 80, se consolida en los 90 y hasta nuestros días. Aunque el interés inicial se centró en lenguajes de programación orientados a objetos, cuando comenzaron a madurar el interés se volvió hacia los métodos de análisis y diseño orientados a objetos.

En 1964 aparece SIMULA (por Dahl y Nygaard), que marca el comienzo de la orientación a objetos. Éste es un lenguaje de simulación, que ha influido en el desarrollo de otros lenguajes de programación orientados a objetos.

En 1972 surge SMALLTALK (por Kay, Goldberg e Ingalls), el cual es una ampliación de dos lenguajes, Simula y LISP (sin tipos). Origina el primer lenguaje de programación orientado a objetos puro.

Smalltalk surge con las características definidas por [Byte81] las cuales definen:

- Todo es un objeto.
- El programa es un conjunto de objetos que se comunican mediante mensajes.
- Todo objeto es instancia de una clase (tiene un tipo).
- La clase es el repositorio de comportamiento asociado con un objeto.
- Las clases se clasifican en jerarquías de herencia.
-

Los años 80 se caracterizaron por la proliferación de los lenguajes de programación orientada a objetos, con lo cual surgieron los lenguajes híbridos (es decir que soportaban tanto el concepto de objetos como el de programación estructurada), entre ellos están:

- Los que se basan en C:
- En 1983 (por Brad Cox y Tome Love) con Objective-C.
- En 1985 (por B. Stroustrup) con C++.
- Los basados en el lenguaje Pascal:
- En 1985 (por Apple y Wirth) con Object Pascal;
- En 1988 (Digital y Oliveti) con Modula-3;
- En 1988 surge CLOS el cual está basado en LISP.

En los años 90 surgen las aplicaciones centradas en la Web:

En 1995 Java-Sun; el cual se orientó en el comportamiento en páginas HTML con la tecnología de implementación código byte ejecutada por una máquina virtual.

Ahora en 2000 surge C# de Microsoft el cual está basado tanto en C++ como en Java y que mediante la tecnología de implementación código byte por una máquina virtual se ejecuta sobre la plataforma denominada .NET.

1.3.2 El origen de OOP, en búsqueda de un buen diseño

Una representación abstracta de un fenómeno que existe o sucede en el mundo real resulta más intuitiva a la percepción que el ser humano tiene de su entorno. Un sistema debe ser construido con un mínimo conjunto de partes incambiables; estas partes deben ser tan generales como sean posibles; y todas las partes del sistema deben mantenerse en un área de trabajo uniforme y permitir de manera sencilla el aumento de su funcionalidad.

1.3.3 Objeto

Es una unidad atómica que encapsula estado y comportamiento. Puede caracterizar una entidad física (bicicleta) o una abstracta (ecuación matemática). Este encapsulado tiene como ventaja mantener manejable la complejidad, un bajo impacto en los cambios e incrementar la reutilización.

El modelado de objetos permite representar el ciclo de vida de los objetos a través de sus interacciones. Por ejemplo, el objeto Banco de Guatemala está relacionado con el objeto cuenta corriente 21 del objeto Juan; estos objetos existen en un esquema de operaciones bancarias y en éste se relacionan.

1.3.4 Clase

Una clase es el ámbito de definición de un conjunto de objetos y describe la estructura de ellos. Sin embargo, solo define el esquema bajo el cual pueden ser representados. Esta definición tiene doble naturaleza programática:

- La de tipo sintáctico: es decir, es un módulo que encapsula componentes de software y es un mecanismo de organización.
- La de tipo semántico: es decir, un tipo que es un mecanismo para definir nuevos datos (de un dominio) que definen tanto su estructura como las operaciones que aplican a éste.

1.3.5 La ecuación del objeto

Un objeto puede estar definido por la siguiente ecuación.

Objeto = Identidad + Estado + Comportamiento.

De donde:

- Identidad: un identificador único y global para cada objeto dentro del sistema, el cual es determinado en el momento de la creación del objeto y es independiente de la localización física del mismo. Es independiente de las propiedades y no cambia durante toda la vida del objeto
- Estado: está representado por los valores de los atributos y evoluciona con el tiempo. Algunos atributos pueden ser constantes.
- Comportamiento: agrupa las competencias de un objeto y describe las acciones y reacciones de ese objeto.

Las operaciones de un objeto son consecuencia de un estímulo externo representado como mensaje enviado desde otro objeto. Los mensajes navegan por los enlaces, en principio en ambas direcciones. Por lo cual el estado y el comportamiento están relacionados, pues son dependientes.

1.3.6 La comunicación entre objetos

Un sistema informático puede verse como un conjunto de objetos autónomos y concurrentes que trabajan de manera coordinada en la consecución de un fin específico. El comportamiento global se basa pues en la comunicación entre los objetos que la componen.

Según la comunicación entre los objetos, los podemos clasificar así:

- Activos – Pasivos
- Clientes – Servidores, Agentes.
-

Objeto activo: es el que posee un hilo de ejecución propio y puede iniciar una actividad.

Objeto pasivo: no puede iniciar una actividad, pero puede enviar estímulos una vez que se le solicita un servicio.

Objeto cliente: es el que solicita un servicio.

Objeto servidor: es el objeto que provee el servicio solicitado.

Objeto agente: es el que reúne características de cliente y de servidor y son la base del mecanismo de delegación.

Mediante éste se introduce indirección, pues un objeto puede comunicarse con otro que ni conoce (como en el caso en que un agente hace de aislante).

1.3.7 El concepto de mensaje

Es la unidad de comunicación entre los objetos, el soporte de comunicación que vincula dinámicamente los objetos que fueron separados previamente en el proceso de descomposición. Éste adquiere su potencia cuando se asocia a la ejecución de responsabilidades independientes del tipo y a las llamadas cuya determinación de ejecución están basadas en el contexto del mismo.

1.3.8 Mensaje y estímulo

- Estímulo: causará la invocación de una operación, la creación o destrucción de un objeto o la aparición de una señal.
- Mensaje: es la especificación de un estímulo.

Con lo cual encontramos los tipos de flujo de control: llamada a procedimiento o flujo de control anidado, flujo de control plano, retorno de una llamada a procedimiento y otras variaciones como esperado y cronometrado.

1.3.9 Persistencia de un objeto

Designa la capacidad de un objeto de trascender en el espacio / tiempo, con lo cual surge la materialización del mismo (es decir, tomarlo de memoria secundaria para utilizarlo en la ejecución); de forma general, un objeto existe desde su creación hasta que se destruya.

Creación de objetos: existe un mecanismo explícito, o en ocasiones implícito, de creación de objetos denominados constructores, los cuales dejan el objeto en un estado válido; éste no tiene valores de retorno.

Destrucción de objetos: existe un mecanismo explícito o implícito de destrucción de objetos denominados destructores, los cuales liberan los recursos obtenidos por el objeto cuando fue creado; éste no tiene valores de retorno. Según la estrategia del lenguaje de compilación, un destructor libera los recursos o éstos son liberados posteriormente mediante la estrategia de limpieza de memoria denominada recolector de basura.

1.3.10 Ocultación de información

Representa dos ventajas básicas: se protegen los datos de accesos indebidos, el acoplamiento entre clases se disminuye y se favorece el concepto modular y el mantenimiento.

Un aspecto básico del encapsulamiento es que los atributos de una clase no deberían ser manipulables directamente por el resto de objetos.

A fin de soportar este principio, las especificaciones de acceso son definidas así:

- Pública: un cliente puede consultarlo y modificarlo; a fin de evitar violación al principio del ocultamiento, es necesario que ningún atributo de una clase sea público.
- Privada: sólo será accesible dentro de la clase, excepto en algunos lenguajes donde existe el concepto de clases amigas, las cuales podrán tener acceso a lo privado de una clase si son amigas de ésta.
- Protegida: visible dentro de la clase y sus derivadas; y además visible a las clases amigas.

1.3.11 Relaciones

Los enlaces entre objetos pueden representarse entre las respectivas clases, éstos son:

Asociación: la asociación expresa una conexión entre objetos en ambas direcciones. Una asociación es una abstracción de la relación existente en los enlaces entre objetos. Por ejemplo, el objeto Juan tiene un enlace con el objeto Universidad de San Carlos; por lo tanto, existe una asociación entre la clase estudiante y la clase universidad. Otro ejemplo de asociación sería: el objeto Juan es marido del objeto María y el objeto María es mujer del objeto Juan, por lo que existe un enlace denominado "casado con" el cual existe en ambas direcciones por lo cual existirá una relación de asociación entre la clase persona. Parte de la definición de una asociación es especificar el nivel de multiplicidad, mediante los límites inferiores y superiores, es decir, su mínimo y su máximo; con lo cual se da una restricción de existencia a la asociación. Por ejemplo en el caso del objeto aerolínea que tiene un enlace "boleto de avión" con el objeto viajero, su relación es cualificada al valor de 1, pues el viajero sólo puede viajar en una aerolínea en un momento dado.

Agregación: representa una relación que parte de entre los objetos, la cual puede ser caracterizada con precisión determinando las relaciones de comportamiento y estructura que existen entre el objeto agregado y cada uno de sus objetos componentes. De forma que la multiplicidad máxima hacia un objeto agregado puede ser disjunta (uno) o no disjunta (mayor que uno), y la mínima puede ser flexible (cero) o de estricta multiplicidad (mayor a cero); y hacia el objeto componente, la multiplicidad máxima puede ser de un solo valor (uno) o de muchos valores (más de uno), y la mínima puede ser con nulos permitidos (cero) o sin nulos permitidos (mayor a cero).

La agregación es caracterizada según si el objeto componente puede comunicarse directamente con objetos externos al objeto agregado, entonces será no inclusiva; si no se puede comunicar directamente, entonces será inclusiva. Si la composición del objeto agregado puede cambiar, entonces ésta será dinámica; pero si no cambia, entonces ésta será estática. Por ejemplo, el objeto ventana está compuesto por 2 barras de desplazamiento, un título y un cuerpo.

Generalización: permite gestionar la complejidad mediante una clasificación jerarquizada y sistemática de clases. Consiste en factorizar las propiedades comunes de un conjunto de clases en una clase más general. No es simétrica ni reflexiva, pero sí transitiva. Expresa una relación de inclusión entre conjuntos.

Las relaciones de agregación y generalización forman jerarquías de clases.

1.3.12 Herencia

Es el mecanismo que permite que una clase A herede propiedades de una clase B. Decimos "A hereda de B". Objetos de la clase A tienen así acceso a los atributos y métodos de la clase B sin necesidad de redefinirlos.

Superclase o clase padre y subclase o clase hija o derivada: si la clase A hereda de la clase B, entonces B es la superclase de A. A es subclase de B. Los objetos de una subclase pueden ser usados en las circunstancias donde son usados los objetos de la superclase correspondiente. Esto se debe al hecho de que los objetos de la subclase comparten el mismo comportamiento que los objetos de la superclase. Por supuesto, también se puede heredar de una subclase, haciendo que esta clase sea la superclase de la nueva subclase. Esto conduce a una jerarquía de relaciones superclase / subclase. Por ejemplo: la clase vehículo es superclase de las clases vehículo terrestre y vehículo aéreo; a su vez, la clase vehículo terrestre es superclase de las clases camión y carro.

Clase abstracta: una clase A se llama abstracta si es usada solamente como una superclase para otras clases. La clase A solamente especifica propiedades. No se usa para crear objetos. Las clases derivadas deben definir las propiedades de A. Las clases abstractas permiten estructurar una gráfica de herencia. Sin embargo, con su definición no se desea crear objetos a partir de ellas, sino solamente expresar características comunes de un conjunto de clases definidas por sus clases derivadas, forzándolas a que ofrezcan las mismas propiedades de su superclase.

Especialización: es una técnica eficaz para la extensión y reutilización. No es simétrica ni reflexiva pero sí transitiva. Expresa una relación de inclusión entre conjuntos; por ejemplo, las clases carro funcionando y carro descompuesto pueden ser especializaciones de la clase carro.

Clasificación estática: es cuando se hace una especialización o generalización basada en el espacio del objeto. Por ejemplo, un vehículo aéreo es una superclase de las subclases avión y helicóptero.

Clasificación dinámica: es cuando se hace una especialización o generalización basada en el espacio de estados de los objetos. Por ejemplo, un carro es una superclase de las subclases carro funcionando y carro descompuesto.

1.3.13 Herencia múltiple

Significa que una subclase puede tener más de una superclase. Esto permite que la subclase herede propiedades de más de una superclase y "mezcle" sus propiedades.

Si la clase A hereda de más de una clase, por ejemplo A hereda de B1, B2, hasta Bn, hablamos de herencia múltiple. Esto puede presentar conflictos de nomenclatura en A, si al menos dos de sus superclases definen propiedades con el mismo nombre. Este

mecanismo de clasificación debe ser usado de manera apropiada para evitar además errores de precedencia.

1.4 Teorías del aprendizaje

Cada teoría, cada autor, considera al aprendizaje de diferente forma y lo explica con diferentes conceptos. Para unos será un cambio de conducta o de comportamiento; para otro será una nueva forma de adaptarse; otros, en fin, lo explican como una vivencia personal, interna.

Sin embargo, cualquier teoría de aprendizaje está de acuerdo en que participan ineludiblemente en el proceso de aprendizaje los siguientes factores:

- **Estructura biológica.** participación de este componente personal con sus sistemas que contribuyen en los diferentes tipos de aprendizajes.
- **Inteligencia.** considerada como el grado necesario para comprender y procesar información, así como elaborar respuestas y acciones de pensamiento.
- **Contexto social.** las posibilidades de aprendizaje se desarrollan en vinculación con otros, en la relación con personas, tanto el círculo social inmediato y cercano como con aquél, más global, general y mediato.
- **Motivación.** focalización del individuo para satisfacer determinadas necesidades percibidas. Es un elemento dinámico, conativo, de impulso a la acción.
- **Operaciones mentales.** referidas al conocer y el pensar; desde lo percibido hasta los procesos cognitivos más complejos, como la reflexión, la imaginación, la extrapolación, etc.
- **Desarrollo histórico personal del individuo.** la experiencia preliminar y lo que actualmente es entendidos como producto de una evolución y desarrollo en el tiempo. El individuo actúa hoy con todo su pasado expresado en su realidad actual.

- **Componentes emocionales.** la experiencia del individuo con el mundo de las cosas y las personas se da en ambientes de tonalidades afectivas, generando tanto aprendizajes como sentimientos, coloridos que tiñen a cada sujeto en particular. Desde otra mirada, estos factores van integrándose y configurando una personalidad particular que caracteriza la forma como se enfrenta a los aprendizajes.

1.4.1 Las teorías conductistas

Se centran en la conducta observable intentando hacer un estudio totalmente empírico de la misma y queriendo controlar y predecir esta conducta. Su objetivo es conseguir una conducta determinada, para lo cual analiza el modo de conseguirla.

De esta teoría se plantean dos variantes: el condicionamiento clásico y el condicionamiento instrumental y operante. El primero de ellos describe una asociación entre estímulo y respuesta contigua, de forma que si sabemos plantear los estímulos adecuados, obtendremos la respuesta deseada. Esta variante explica tan solo comportamientos muy elementales. La segunda variante, el condicionamiento instrumental y operante, persigue la consolidación de la respuesta según el estímulo, buscando los reforzadores necesarios para implantar esta relación en el individuo.

Para las teorías conductistas, lo relevante en el aprendizaje es el cambio en la conducta observable de un sujeto, cómo éste actúa ante una situación particular. La conciencia, que no se ve, es considerada como "caja negra". En la relación de aprendizaje sujeto-objeto, centran la atención en la experiencia como objeto y en instancias puramente psicológicas, como la percepción, la asociación y el hábito como generadoras de respuestas del sujeto. No están interesados particularmente en los procesos internos del sujeto debido a que postulan la “objetividad”, en el sentido que sólo es posible hacer estudios de lo observable.

Tabla I. El poder explicativo de las teorías conductistas

<table>
<tr><td rowspan="2">Teorías conduc-tistas</td><td colspan="2">Poder explicativo</td></tr>
<tr><td colspan="2">Cómo describe el aprendizaje</td><td>Qué aprendizajes explica y/o predice</td></tr>
<tr><td>Watson</td><td rowspan="5">Comporta-mientos observables</td><td>Secuencia apropiada de estímulo-respuesta; comportamiento objetivo condicionado</td><td rowspan="5">Aprendizajes de automatismos como hábitos, habilidades y destrezas. Aprendizaje por ensayo y error</td></tr>
<tr><td>Guthrie</td><td>Conexión estímulo-respuesta por hábito</td></tr>
<tr><td>Thorndi ke</td><td>Formación de asociaciones estímulo-respuesta por refuerzo; comportamiento por conexiones neuronales</td></tr>
<tr><td>Hull-Spence</td><td>Refuerzo por asociación estímulo-organismo-respuesta</td></tr>
<tr><td>Skinner</td><td>Asociación respuesta-recompensa ante un estímulo (condicionamiento operante: la conducta está controlada por las consecuencias)</td></tr>
</table>

1.4.2 Teorías cognitivistas

Este modelo de teorías asume que el aprendizaje se produce a partir de la experiencia, pero, a diferencia del conductismo, lo concibe no como un simple traslado de la realidad, sino como una representación de dicha realidad.

Se pone el énfasis, por tanto, en el modo en que se adquieren tales representaciones del mundo, se almacenan y se recuperan de la memoria o estructura cognitiva; se realza el papel de la memoria con un valor constructivista. No se niega la existencia de otras formas de aprendizaje inferior, pero si su relevancia, atribuyendo el aprendizaje humano a procesos constructivos de asimilación y acomodación.

Concibe al sujeto como procesador activo de la información a través del registro y organización de dicha información para llegar a su reorganización y reestructuración en el aparato cognitivo del aprendiz.

A diferencia de las posiciones asociacionistas,

- No se trata de un cambio solo cuantitativo (en la probabilidad de la respuesta), sino cualitativo (en el significado de esa respuesta).
- No es un cambio originado en el mundo externo, sino en la propia necesidad interna de reestructurar nuestros conocimientos o de corregir sus desequilibrios.
- No cambian los elementos aislados (estímulos y respuestas), sino las estructuras de las que forman parte (teorías y modelos).
- En fin, no es un cambio mecánico, sino que requiere una implicación activa, basada en la reflexión y la toma de conciencia por parte del alumno.

1.4.3 El constructivismo

Está dado por la afirmación de que el conocimiento no es el resultado de una mera copia de la realidad preexistente, sino de un proceso dinámico e interactivo a través del cual la información externa es interpretada y re-interpretada por la mente, que va construyendo progresivamente modelos explicativos cada vez más complejos y potentes.

Esto significa que conocemos la realidad a través de los modelos que construimos para explicarla, y que estos modelos siempre son susceptibles de ser mejorados o cambiados.

Cómo el constructivismo impacta en el aprendizaje:

En el currículum (plan de estudios). El Constructivismo plantea la eliminación de un plan de estudios estandarizado. En su lugar, promueve el uso de programas personalizados de acuerdo con requisitos particulares del conocimiento anterior de los estudiantes. También pone énfasis en metodologías de solución de problemas prácticos.

Instrucción. Bajo la teoría del constructivismo, los educadores se centran en hacer conexiones entre diversos hechos y fomentar una nueva comprensión en los estudiantes. Los instructores adaptan sus estrategias de enseñanza a las respuestas del estudiante y animan a los estudiantes a que analicen, interpreten y predigan la información. Los profesores también confían realmente en preguntas de respuestas abiertas y promueven el diálogo extenso entre los propios estudiantes.

Evaluación. El constructivismo hace un llamado a la eliminación de grados y de las pruebas estandarizadas. En su lugar, la evaluación debe llegar a ser parte del proceso de aprendizaje de modo que los estudiantes desempeñen un papel más vital en el juicio de su propio progreso.

1.5 Métodos educativos

1.5.1 Inter actuación alumno-empresa

Se introduce al alumno en un entorno concreto donde pueda enfrentarse con los problemas que debe resolver. Los alumnos tienen un primer contacto con la empresa

contemplando la realidad actual e intuyendo las necesidades profesionales del sector. Se obtiene información durante un periodo corto de tiempo, luego ésta es usada como base para las prácticas de cada tarea. Esta información específica ha de ser obtenida por los estudiantes de una forma estructurada que permita formar los juicios adecuados sobre el entorno al cual es introducido, pues esto le servirá de base para aplicar los conceptos enseñados en el aula.

1.5.2 Comprensión colaborativa

Prácticas usadas para rápidamente compartir responsabilidades de un curso. Las actividades del estudiante son esencialmente enfocadas en el desarrollo de ejercicios de complejidad incremental, como sesiones de discusión, donde las soluciones diseñadas son criticadas y comparadas.

1.5.3 Orientado al aprendizaje

Todas las actividades del curso están orientadas en el alumno y no en el instructor. Y en los objetivos en lugar del contenido. Cada sesión se organiza para que los alumnos alcancen objetivos concretos. Después de la introducción con el contenido necesario para alcanzar los nuevos objetivos, los alumnos resuelven individualmente un problema.

1.5.4 Orientado a la enseñanza

Todas las actividades están orientadas hacia el contenido, cuyo objetivo es cumplir con los tiempos planeados en él.

1.5.5 Enseñanza constructivista

El aprendizaje es visto como un proceso personal, reflexivo y transformativo, en donde los estudiantes son motivados a analizar de forma crítica, a descubrir relaciones y patrones, comparar, contrastar y transformar la información en algo nuevo. Un elemento base es la especificación y uso de auténticas y complejas actividades durante el proceso de aprendizaje. Mientras el estudiante trabaja en el mundo real, se ve forzado a aprender, desarrollar y aplicar las habilidades necesarias para resolver problemas, adquiriéndolas, en parte, por ellos mismos.

1.5.6 Lectura activa

Esta es la filosofía: oí y olvidé, miré y recordé, lo hice y aprendí.

Se solicita a todos los estudiantes el estudio de un capítulo del libro de texto o un trabajo particular antes de la clase; además, deben responder las preguntas de revisión y entregarlas antes de iniciar la clase; la clase se inicia con anuncios administrativos como una hoja de un conjunto de ejercicios, y una prueba de preguntas se pasa a la clase; mientras los estudiantes trabajan, se regresa el trabajo calificado.

Durante la clase, se selecciona un ejercicio y se invita a consultar. Si las preguntas indican algún conocimiento relevante pero perdido, se provee de una conferencia corta del asunto. Típicamente los estudiantes hacen un ejercicio solos. Entonces se les pide que compartan su trabajo en pares o grupos de tres o cuatro. Las preguntas pueden ser presentadas a la clase por los estudiantes, pares o grupos.

En algunas ocasiones se agrega una técnica llamada la galería de arte. Se usa esto con diagramas o pequeños párrafos de texto. Cada grupo realiza su solución en pizarras.

Con grupos de 4 estudiantes y típicamente con 32 en clase, significa que serán dibujadas 8 soluciones.

Se trata de que todos los estudiantes piensen en la solución esencial, motivando comentarios de la clase. No cuentan los errores ni valdrán puntos: sin embargo, claramente se etiquetan los errores como tales y se precisan los riesgos al cometerlos. Una ayuda importante es el responder a una pregunta, esperando mientras los estudiantes piensan (actividad de cubrir) y después se realizan acciones visibles (actividades). Por ejemplo, se puede colocar un diagrama en la pizarra y decir: “silenciosamente piense que significa el diagrama, puede apuntar sus pensamientos; cuando tenga una respuesta, levante su mano. Seleccionaré a alguien para que responda en 30 segundos” .

Otra técnica es colocarse en círculo con los estudiantes, y que cada estudiante pregunte algo nuevo por turnos. Se ha estudiado una información mínima acerca del tema y también tienen la opción de responder la pregunta. Si la respuesta no está cubierta en sus lecturas y no ha sido cubierta en previas preguntas, es respondida por el maestro. Si a juicio del instructor un estudiante puede responder la pregunta, entonces ésta va de vuelta a los estudiantes para que la respondan.

1.6 Métodos de evaluación

1.6.1 La evaluación del proceso

Valora en que medida se está implementando con eficacia la enseñanza. Se centra en aspectos tales como quién esta participando, qué actividades se ofrecen, qué medidas se han adoptado, cuáles son las prácticas del educador a cargo y cuáles las reacciones de los estudiantes. Éste funciona como un sistema de alerta temprana; para ejecutarse requiere de diseños de evaluación y métodos de consulta menos formales, como autoevaluación y dictámenes de expertos.

1.6.2 La evaluación de resultados

Estudia lo que le ha sucedido a los individuos después de la aplicación de la enseñanza, se centra en los resultados de la intervención. El diseño de las evaluaciones de resultados puede variar en una serie niveles de complejidad. Se busca determinar si los participantes experimentaron un cambio en el grado de educación, usando pruebas de respuesta múltiple, pruebas verdadero o falso, pruebas de emparejamiento, preguntas cortas y problemas, preguntas amplias, mapas conceptuales y otras.

1.6.3 La evaluación basada en teorías

Analiza los vínculos entre factores causales, actividades y resultados; pretende determinar si se ha producido un cambio y, si es así, dónde y cómo. Presentan una teoría de cómo y por qué la aplicación de la enseñanza funciona como conjunto de pasos. Y los analiza para realizar un seguimiento del desarrollo de la hipótesis. Pretende con esto determinar, de todo el proceso, qué ha fallado y dónde.

1.7 Patrones pedagógicos

1.7.1 Enseña mejor quien enseña menos

Abstracciones concretas para mantener el interés de los estudiantes al presentar los primeros modelos.

1.7.2 Ejemplos de conocimiento

Escoger para ejemplos aquéllos que sean familiares a los estudiantes, pero que no estén dentro del área de especialización de los mismos.

1.7.3 Analogía vivida

Un concepto difícil puede resaltarse con una analogía que proporcione un lugar para remontarse a ésta y recordar los detalles.

1.7.4 Espiral

Organice el curso para presentarles temas a los estudiantes sin cubrirlos totalmente al principio para que puedan introducirse varios temas al inicio y entonces puedan ser usados. Esto puede conseguir que los estudiantes trabajen en problemas interesantes al inicio pues ellos tienen más herramientas para usar, aunque ellos no tienen, quizás, dominio completo de las herramientas. Entonces el instructor puede regresar cada vez a cada tema, quizás repetidamente, dando más de la información necesaria para que éstas sean dominadas.

1.7.5 Error

Se pide a los estudiantes que creen un artefacto como un programa o plan que contiene un error específico. El uso de este modelo enseña a los estudiantes explícitamente cómo reconocer y arreglar errores. Se le pide también al estudiante hacer ciertos errores explícitamente y entonces que explique las consecuencias.

1.7.6 Volar temprano

Problema: muchos tópicos interrelacionados.

Es difícil decidir cómo ordenar los tópicos de modo que los estudiantes aprecien la idea completa del curso. Si se espera hasta el final del curso, se gasta mucho tiempo en cuestiones preliminares, con lo cual los estudiantes pueden llegar a tener una idea errónea sobre la importancia relativa.

Obligación: ver la representación detallada en la etapa inicial de las ideas importantes sobre las que tratará el curso.

Se necesita saber cuál es la pequeña gran idea de cada curso. Tienen que estar en la capacidad de separar los conceptos importantes de los detalles que lo soportan. Los estudiantes por lo regular recuerdan lo que aprendieron primero.

Solución: primero identifique las ideas más importantes en el curso. Mantenga el curso para estas importantes ideas. Estas ideas llegarán a ser los principios organizacionales fundamentales del curso. Introduzca estas grandes ideas, y especialmente sus relaciones, al inicio del curso y retorne a estas repetidamente a través del curso.

Aquí ordenamos clases de tópicos en orden de importancia y encontramos vías para enseñar las ideas más importantes rápidamente.

Si el diseño es más importante que programar, entonces encuentre una vía para diseñar tan rápido como usted pueda. Si las funciones son más importantes que las sentencias en programación, entonces haga éstas primero. Si los objetos son más importantes que las funciones, entonces haga esto primero.

Discusión / consecuencia / implementación

Lo más importante en un curso o currículo recibe más atención del instructor y el estudiante, por lo que se tiene la oportunidad de regresar a ello cada vez que se quiera. La implementación es difícil, a menudo sólo aspectos simples de una importante idea pueden ser introducidos inicialmente. Algunas veces esto es suficiente para dar términos importantes e ideas generales. Algunas ideas "grandes" son muy avanzadas, y es difícil introducir algunos de estos conceptos rápidamente.

1.7.7 Caja de juguetes

El intento es dar a los estudiantes un conocimiento histórico y tecnológico bastante grande para dejarlos jugar con herramientas pedagógicas.

1.7.8 Caja de herramientas

El intento es permitir a los estudiantes construir un equipo de herramientas en cursos iniciales para que éste las use en cursos siguientes. Esto puede ser una guía muy buena para el desarrollo de software reutilizable. Este tipo de conocimiento era usado en la época medieval donde los aprendices construían sus herramientas, las cuales serían usadas por ellos al llegar a ser expertos en el área.

1.7.9 Disposición de la tierra

Se da a los estudiantes una experiencia al inicio examinando un software complejo, más allá de su habilidad actual, con el afán de mostrarles la complejidad del campo en el que están a punto de estudiar, y lo que serán capaces de producir una vez lleguen a concretar su estudio.

El artefacto debe incluir mucho de los temas a estudiar en el curso, y puede ser motivo de discusión para los siguientes puntos a enseñar.

1.7.10 Prueba y resultado

Los estudiantes pueden resolver preguntas "qué pasa si". Algunas son más efectivas y rápidas que la propia documentación. En el área de programación, al dar a los estudiantes varios ejercicios en los que se les pide que escriban pequeños programas y que resuelvan sus preguntas "qué pasa si", se forma el hábito de probar en la máquina.

1.7.11 Rellene los espacios en blanco

Los estudiantes pueden aprender tópicos complejos construyendo varias pequeñas partes de grandes artefactos. Esto persigue la capacidad de lectura y escritura de código para lo cual es necesario preparar un programa completo bien diseñado, o parte de él, y quitar una pequeña pieza de él. Se da esto a los estudiantes con instrucciones de llenar la parte perdida. La parte perdida necesita ser bien especificada.

2 ANÁLISIS DE LA ENSEÑANZA ACTUAL

2.1 Análisis de los métodos de enseñanza usados

2.1.1 El curso típico de programación

Un curso típico de programación en la universidad inicia con el aprendizaje de todas las construcciones programáticas; conocemos los ciclos, las decisiones, las operaciones de entrada y salida, variables simples, registros, punteros. Luego empezamos a aprender procedimientos, clases, constructores, plantillas, flujos de información, excepciones, etc. Finalmente usamos esas construcciones recientes en nuestra comprensión para escribir varios programas.

En muchos casos se usa el seudo código, el cual puede causar confusión en el alumno y puede llegar a mezclar aspectos de las gramáticas. Por eso es necesario poner excesivo cuidado en hacer una simetría con las estructuras sintácticas de los lenguajes usados en la industria y el seudo código, sin comentar más posibilidades en los formatos de las instrucciones de las que sean convenientes para poder hacer el paralelismo con estos lenguajes.

Además, algunos maestros facilitan al estudiante tablas de conversión que en cualquier momento (tanto en el desarrollo de sus prácticas como en los exámenes) pueda consultar.

El alumno es orientado a realizar diferentes tareas con los conceptos aprendidos, tales como ordenamiento de datos y el uso de construcciones progresivamente. Se evita la vinculación del desarrollo de programas con un lenguaje de programación concreto.

2.1.2 Por qué es malo iniciar con la enseñanza de la programación procedural, cuando el objetivo es la orientada a objetos.

En el currículo de estudios de la Universidad de San Carlos se prefiere enseñar programación procedural antes de la programación orientada a objetos, pues se tiene la opinión de que la programación orientada a objetos es un agregado que extiende el procedural.

Actualmente se enseña programación procedural y se hace bien; sin embargo, aún no se ha llegado a enseñar adecuadamente la programación orientada a objetos. Hacer esto requiere la comprensión de algunos principios subyacentes que afectan grandemente la forma de enseñar que se debe de usar.

Analizando el concepto de programa, en la programación procedural la naturaleza de la solución para un problema es un conjunto de acciones. Sin embargo un "programa" es más que procesos, consiste en ambos: procesos y datos. En las palabras de Niklaus Wirth, algoritmos + estructuras de datos = programas. Sin embargo el paradigma procedural se enfoca primero en los algoritmos y luego procura que las estructuras de datos encajen en ese proceso.

En el curso típico de programación bajo el paradigma procedural se discute la arquitectura de la máquina de Von Newman, y los estudiantes son animados a pensar en la memoria RAM como el almacén para procesos y datos. El programa es la transformación de los datos de un estado inicial a uno final; el énfasis es puesto en las sentencias que van cambiando el estado, y los datos son enviados al CPU para ser procesados. Este modelo encaja totalmente con el paradigma procedural pues tiene una simple relación entre el modelo físico y el virtual provisto por los lenguajes de programación; sin embargo, no lo hace con otras vías más abstractas de ver la computación, tal como el funcional y el orientado a objetos y mucho menos el paralelo. Por ejemplo, el paradigma funcional esconde totalmente la arquitectura física subyacente

y el orientado a objetos no, sin embargo lo cambia totalmente en el pensamiento, pues en lugar de que los datos sean movidos al CPU para su procesamiento, una metáfora muy común en programación orientada a objetos es que el CPU se mueve a los objetos para proveer energía de procesamiento.

Cuando se programa en un lenguaje de programación orientada a objetos puro, casi nunca se piensa en un programa como un proceso, y casi nunca se piensa en escribir una función (método, técnicamente) por descomponer esto en funciones simples. En lugar de esto, mientras el paradigma procedural se enfoca primero sobre los algoritmos, el paradigma de objetos se enfoca primero en los datos. Esto significa que los objetos en el sistema son pensados como datos (datos de alto nivel) que encapsulan algoritmos tan bien como los datos de bajo nivel.

En la programación orientada a objetos el programa en sí se ve como un gráfico de elementos que interactúan, cada uno de los cuales actúa como un servidor para alguna pieza de información u otro servicio. Cada elemento puede también actuar como un cliente de otros elementos cuando aquel elemento necesita colaborar con aquellos otros para alcanzar el cumplimiento de su propio servicio.

La metodología para construir un programa orientado a objetos es descubrir los objetos primero por medio de un proceso de simulación en el cual los objetos de software en el sistema deben modelar los objetos del mundo real del sistema modelado. De forma que como diseñador de programación orientada a objetos, se ven los elementos del modelo, no los procedimientos a estructurar. Estos elementos exhiben el comportamiento activo más bien que el actuado; éste comportamiento es el servicio que el objeto debe proveer. Si hay algo que no tiene comportamiento, probablemente no es un objeto.

Una vez se tienen objetos candidatos, se les asignan responsabilidades, que también son los servicios. Al conocer todo de una interacción en la cual un objeto actúa como servidor, se conoce el conjunto completo de responsabilidades de este objeto. Si hay varios objetos con el mismo comportamiento, entonces da origen a una clase que los describe. Entonces se está en posición de definir las responsabilidades y modelarlas mediante métodos. Como se puede notar, este paso no es el primero en el paradigma procedural, sino que es el segundo.

Luego, es necesario decidir, por cada objeto, qué información necesita ser mantenida en él para que el servicio pueda ser desarrollado. Esta es la información que necesita ser mantenida mientras el servicio no esté en proceso, no como los datos entre los métodos que solamente serán necesarios mientras el método esté ejecutándose. Estos datos representan la información de la instancia de la clase (los datos de bajo nivel). Si los objetos de la clase necesitan compartir información entre sí se definen estos datos a nivel de clase, llamándolos estáticos en algunos lenguajes como java.

Concluyendo, entonces, un programa procedural se ve como un árbol de funciones, y un programa orientado a objetos como una red de clientes y servidores. Vistas muy diferentes, por supuesto. Y para ser un buen programador ya sea de un tipo o de otro, es necesario estar cómodo con el trabajo que se trata de hacer. El esquema mental de la naturaleza de la computación tiene que ser natural para los procesos de pensamiento, de modo que los procesos típicos de pensamiento conduzcan a un conjunto de soluciones en lugar de al otro.

Esta es la naturaleza del cambio del paradigma que atraviesan los programadores del paradigma procedural al tratar de llegar a ser programadores de la orientación a objetos. No hay nada especialmente complejo sobre la programación orientada a objetos, más que cualquier cosa compleja sobre la programación procedural. La diferencia es que el mundo se mira completamente diferente en uno u otro paradigma.

Segun Stroustrup, un programador procedural experimentado llevará de un año a 18 meses para hacer el cambio a orientación a objetos. Ya que mientras se siga en este modo de aprendizaje, tratará de forma inconsciente de resolver problemas de estructuras de funciones y no por descubrimiento de objetos. Siempre que el estudiante vea que es difícil, caerá de regreso sobre lo que ya conoce mejor, la programación procedural. Esto demuestra que toma tiempo reenlazar la nueva forma de pensar. Y si ésta sucede, durante el año antes que suceda, los estudiantes construirán programas realmente malos, mezclando técnicas de forma torpe.

Debido a lo anterior, es concluyente entonces que es un error tratar de enseñar programación procedural primero, si el objetivo es la programación orientada a objetos. El mejor esfuerzo para los estudiantes principiantes debe hacerse al inicio, ya que será más difícil si se hace después, porque sus habilidades naturales para resolver los problemas serán enlazadas para buscar soluciones en los procesos y no en los objetos. Si se hace un excelente trabajo de enseñarles a pensar como un programador procedural, ellos harán frente a este cambio de paradigma durante 12 a 18 meses. Y es una pérdida de tiempo en un programa de estudios de cinco años.

Además es fácil moverse del paradigma orientado a objetos al procedural en lugar de al revés, ya que se aprende de funciones en programación orientada a objetos; buenos programas en el lenguaje procedural pueden ser construidos por programadores del paradigma orientado a objetos porque naturalmente parten sus programas en objetos como unidades de funcionalidad aunque no se tenga la herramienta de encapsulado que provee la programación orientada a objetos.

2.1.3 Análisis de la enseñanza del tema

La programación orientada a objetos es el paradigma de programación que más influencia ha tenido en la industria del software.

La programación orientada a objetos soporta elegantemente los conceptos que se han tratado de enseñar por muchos años, tales como programación estructurada, modularización y diseño de programas. Ésta también soporta las técnicas para resolver problemas que aún no han sido incluidas en el plan de estudios: programación en equipos, mantenimiento de grandes sistemas y reutilización del software. Por lo tanto, la programación orientada a objetos es la herramienta para enseñar las metodologías de programación que son consideradas importantes.

Según experiencias de maestros del área, han concordado que enseñar programación orientada a objetos parece ser más difícil que enseñar programación estructurada.

La causa de esta aparente dificultad al enseñar implica tomar en cuenta otro aspecto del problema, tal como lo refiere la siguiente cuestión: ¿cuando se debe enseñar la programación orientada a objetos?

Por mucho tiempo, la programación orientada a objetos ha sido considerada un punto avanzado lo que ha implicado incluir el tema al final del plan de estudios en el área de programación. Esto debe cambiar; la razón principal para hacer esto es el frecuente problema del cambio de paradigma. Aprender a programar en el estilo orientado a objetos parece ser más difícil después de haber utilizado un estilo procedural. La experiencia también muestra que los estudiantes no parecen tener mucha dificultad entendiendo los principios de la orientación a objetos si los conocen primero. Es el cambio lo difícil.

Para enseñar programación, la lección está clara: si queremos enseñar orientación a objetos, debe hacerse primero. El camino a orientación a objetos a través de programación prodedural es innecesariamente complicado. Los estudiantes primero aprenden un estilo de programación, luego tienen que dejar todo lo aprendido previamente, antes que se les muestre cómo deben hacer lo primero bien.

Desafortunadamente también varios libros de texto usan la programación procedural como un camino a los conceptos de objetos. Se ha dado el caso, especialmente por la influencia del lenguaje C++, que por su popularidad es usado bastante en la enseñanza y porque permite la programación procedural y la orientación a objetos, pues es hibrido en su definición y fue desarrollado como una extensión de C. Esto ha conducido a que mucha gente vea la orientación a objetos como otra construcción del lenguaje que puede ser enseñada después de las estructuras de control, punteros y recursión; lo cual es un malentendido. La orientación a objetos es un paradigma subyacente que forma nuestra manera de pensar sobre cómo resolver un problema en un modelo algorítmico. Éste no puede ser "agregado" a otras construcciones del lenguaje; más bien reemplaza la estructura de programación procedural.

Por causa de esto, los conceptos de orientación a objetos deben ser enseñados desde las etapas iniciales. Si se cree necesario que los estudiantes puedan también programar en el estilo procedural, entonces un lenguaje procedural puede ser introducido después. El cambio de paradigma al revés (de orientación a objetos a procedural) es mucho más fácil.

Pero ¿de donde entonces provienen las dificultades de enseñar orientación a objetos? Se puede decir que no es la orientación a objetos la causa del problema, sino la falta de un texto apropiado que no considere la orientación a objetos como un agregado del lenguaje de programación y la falta de una herramienta de programación adecuada para enseñar esta orientación en la práctica.

Esto entonces lleva a la siguiente pregunta: ¿qué herramienta debe ser usada? ¿Cómo debe ser un buen lenguaje y un buen ambiente para enseñar la programación orientada a objetos?

Según Bergin, el diseño de un lenguaje para la enseñanza de la orientación a objetos debe tener los siguientes requerimientos:

- **Conceptos claros:** debe reflejar con claridad el concepto, y evitar confusión entre lo que se explica y lo que se implementa en el lenguaje. Sin embargo, el lenguaje no debe dictar la clase, más bien el lenguaje debe reflejar el nivel de abstracción del modelo conceptual explicado en clase.
- **Orientación a objetos pura:** esto significa que sólo permita aplicar conceptos de la orientación a objetos y así evitar que los estudiantes piensen que están haciendo programas orientados a objetos al compilar en un lenguaje hibrido, cuando realmente están aplicando programación procedural, pues el lenguaje acepta tanto programación procedural como orientada a objetos; además usar un lenguaje puro ayudará a eliminar los malos hábitos de programación que los estudiantes traen de los niveles medios.
- **Seguridad:** significa que en el lenguaje debe ser fácil identificar los errores en tiempo de compilación y de corrida; además los mensajes que identifiquen los errores deben ser claros e identificar la causa, e indicar las construcciones propensas a error. Un ejemplo de este principio es un buen sistema de tipos, el lenguaje debe ser restringido y, como sea posible, escrito estáticamente, ya que en un lenguaje con tipos dinámicos la detección del error es difícil. Otro ejemplo de seguridad son los chequeos de límites de arreglos o el chequeo del uso de variables no inicializadas. Además, las construcciones que se conoce serán motivo de problemas deben ser evitadas lo más posible.
- **Alto nivel:** el nivel de abstracción de las construcciones debe ser alto. Se han de evitar construcciones que obliguen al estudiante a llevar control de tareas que fácilmente puede llevar el compilador. Tal es el caso del manejo de memoria dinámica, pues el control de esto es muy difícil, y más aún la detección de errores, principalmente si los estudiantes están iniciándose en el área.
- **Modelo de ejecución simple de objetos:** la forma de ejecución debe ser simple y fácil de entender, lo cual incluye el modelo de objetos y el modelo de memoria.
- **Sintaxis legible:** el lenguaje debe ser legible pues aumenta la comprensión, es decir, debe estar basado en palabras claves y no en símbolos, ya que éstos son

menos legibles y las palabras claves son más intuitivas. Además el lenguaje debe ser consistente, es decir que la misma construcción no debe usarse en distintos contextos con distinto significado; tal como en lenguajes humanos, es más importante el efecto sobre el lector que la rapidez de escritura, ya que la escritura se hace para el lector. Esta última frase ha de ser un principio a inculcar en el aprendizaje a fin de que se escriban programas pensando en el lector que los tendrá que entender para reusarlos o darles mantenimiento.

- **No redundante:** significa que para todo lo que se quiera hacer en el lenguaje, debe haber una, y solo una, forma de hacerlo. La redundancia está a menudo relacionada con flexibilidad y eficiencia: teniendo diferentes construcciones para alcanzar la misma tarea es a menudo útil para optimizar el código. Estas construcciones alternativas pueden diferir en detalles de bajo nivel tales como rendimiento, disposición de memoria, etc. Para escribir un código eficiente puede ser esencial la influencia de estos detalles de bajo nivel. Para los principiantes, la flexibilidad conlleva a menudo más confusión que eficiencia. Teniendo tres diferentes mecanismos para hacer la misma cosa, las cuales difieren en detalles sutiles, a menudo conlleva problemas para los estudiantes cuando ellos no pueden hacerse un completo juicio de los efectos de su elección.
- **Pequeño:** por efectos de enseñanza esto es fundamental, El catedrático evita tener que explicar por qué no usar ciertas construcciones del lenguaje cuando el estudiante las utiliza. Además, es adecuado tener un punto en el cual decir al estudiante: “esto es todo, ahora usted no aprenderá más construcciones, sino que pondrá todas a funcionar”. Así se le da seguridad al estudiante de que no hay otra construcción posterior que resuelve de forma más fácil y elegante lo que hasta el momento ha realizado.
- **De fácil transición a otros lenguajes:** esto significa que el lenguaje con el cual enseña debe ser de fácil transición a otros lenguajes de uso común en el mercado laboral. Dicho de otra forma, el lenguaje soporte para enseñar la orientación a objetos debe concentrarse en los conceptos básicos de este paradigma, pero no

necesariamente ser un lenguaje de uso común, pero si de fácil transición a otros lenguajes de uso común en el mercado laboral.

- **Soporte para aseguramiento de la corrección:** las técnicas de ingeniería de software enseñadas deben ser soportadas por el lenguaje de programación, a fin de que sean tomadas en cuenta seriamente por el estudiante y no solamente como guías de estilo.
- **Ambiente conveniente:** el ambiente de desarrollo debe esconder detalles del sistema operativo y además soportar la orientación a objetos, a fin de enfocar a los estudiantes en las tareas de programación más que en detalles del sistema operativo. De todos los puntos, éste es tan importante como los otros juntos.

De acuerdo con los anteriores principios, el lenguaje de programación C++ no está bien situado para soportar la enseñanza de la orientación a objetos, por el hecho inicial de ser un lenguaje híbrido, poco legible y bastante grande, que además es redundante y de bajo nivel y que lleva al estudiante a problemas innecesarios al inicio de su comprensión. Sin embargo, éste es el lenguaje que actualmente se utiliza para soportar la enseñanza de la programación orientada a objetos en la Universidad de San Carlos, lo cual es comprensible pues se considera la programación orientada a objetos como un agregado de la procedural, visión que es errónea.

2.1.4 El proceso de enseñanza

En la Universidad de San Carlos de Guatemala, se realiza un solo tipo de clase, la teórica, en la cual el catedrático expone los principales conceptos, correspondientes a los temas del programa definido.

Se realizan exámenes cortos con cierta periodicidad, con el afán de evaluar la comprensión de los estudiantes con respecto a un tema ya expuesto por el catedrático.

Éstos tienen la característica de ser sin aviso y pueden o no tener cierta validez en la nota final del curso.

Se realizan varios proyectos de programación que dan la capacidad de evaluar el aspecto práctico de la enseñanza, mediante programas que pretenden cubrir los conceptos que han sido expuestos en clase.

Mediante la ayuda de un auxiliar, se realizan las evaluaciones de los programas realizados, para lo cual éste toma en cuenta aquellos aspectos que sean sugeridos por el catedrático o aquéllos que en su caso considera importantes; con lo cual no existe una norma de evaluación definida. Esto conlleva la subjetividad de las evaluaciones así como la inconsistencia en la evaluación con respecto a la enseñanza, en algunos casos.

Continuamente el proceso de enseñanza está orientado a la resolución de problemas que conllevan cierto aumento de dificultad.

2.1.5 Desventajas del método de clase magistral

No induce al receptor al aprendizaje de habilidades y establece en general una barrera entre el conocimiento y la aplicación del mismo.

2.1.6 Evaluación de alumnos ya educados en el tema

Un programador de orientación a objetos, según la definición bajo la que nos regiremos, debe tener claros los conceptos para interactuar en este paradigma, de forma que su modelo mental le permita desenvolverse en la elaboración de sus programas, de acuerdo con los preceptos que definen esta orientación.

Nos basaremos en esta afirmación para poder evaluar a los alumnos ya educados en el tema; nuestras preguntas entonces estarán dirigidas a determinar los malos modelos mentales que el estudiante presenta debidos a la falta de una adecuada formación y el estrés que encierra el traslado de un paradigma a otro.

2.1.7 Evaluación

Con la evaluación se pretende determinar el grado de madurez al que los estudiantes han llegado en la comprensión de la orientación a objetos, para lo cual se incluyeron las siguientes preguntas:

- ¿Qué es un algoritmo? Esta pregunta intenta identificar el conocimiento básico del estudiante en cuanto a programación.
- Cuándo piensa en un programa, ¿en qué piensa? Esta pregunta busca determinar el esquema mental del estudiante con respecto al programa, y el enfoque que éste usa para pensar.
- ¿Cuál es el resultando de las siguientes asignaciones? A := 1; B := 2; B := A; B := B; PRINT A; PRINT B. Esta pregunta fue planteada con el objetivo de determinar el grado de concepción inválida del estudiante con respecto a las variables y asignaciones.
- ¿Cuál es el resultado de las siguiente operaciones, si la función leer espera tres dígitos del teclado? Leer(a, b, c); Leer(b, b, a); si el usuario presiona los dígitos en este orden: 5, 2, 7, 9, 1, 2 con las siguientes posibles respuestas :
- a será 5, b será 9 y c será 1. ( )
- a será 2, b será 1 y c será 7. ( )
- b será 1 y a será 5 y c será 7. ( )
- a, b y c tendrán el mismo valor ( )
- ninguno de los anteriores. ( )

- Esta pregunta fue presentada con el objeto de determinar el nivel de las concepciones erróneas que el estudiante posee en la utilización de parámetros.
- ¿Cuál es el resultado de las siguientes operaciones? Dim A as ObjetoString; A = new objetostring(); Dim B as object; B := A; A.valor = "HOLA"; PRINT A.valor, B.valor; Esta pregunta se incluyó para determinar el nivel de concepción errónea que se tiene sobre el manejo de objetos y asignaciones, así como el de punteros.
-
- A continuación el conjunto de preguntas que intentan determinar la concepción que el estudiante ha llegado a tener sobre la orientación a objetos. Cada una de estas están planteadas intentando captar la forma de pensar, y cada respuesta a ellas determina un nivel de comprensión que el estudiante ha logrado.
-
- Cuando piensa en clases, ¿en qué piensa? Concepto, Objeto, Variable, Categoría, Fenómeno, Tipo de dato abstracto, Registro, Otro.
- Cuando piensa en objeto, ¿en qué piensa? Instancia, Variable, Concepto, Categoría, Tipo de dato abstracto, Registro, Fenómeno, Comportamiento, Todas las anteriores, Otro.
- Cuando piensa en herencia, ¿en qué piensa? Jerarquía, Especialización y generalización, Relación de agrupación, Reutilización de código, División de tareas, Encapsulado de información, Todas las anteriores, Otro.
- Cuando piensa en método, ¿en qué piensa? Mensaje, Función, Procedimiento, Algoritmo, Contrato, Comunicación entre objetos, Todas las anteriores, Otro.
- Cuando piensa en atributo, ¿en qué piensa? Variable, Tipo, Objeto, Clase, Registro, Puntero, Estado, Todas las anteriores, Otro.
- Cuando piensa en colección, ¿en qué piensa? Arreglo, String, Objetos, Clases, Memoria, Punteros, Todas las anteriores, Otro.

- Cuando programa orientado a objetos, ¿qué es lo primero que realiza? Clases, Métodos, Modelo, Responsabilidades, Atributos, Comportamiento, Relaciones, Objetos, Algoritmo, Datos, Otro.
- ¿Cómo define un programa en objetos? La interacción de objetos, La colaboración de objetos hacia un objetivo, una secuencia de métodos de los objetos, La respuesta de los objetos a un estímulo, La integración de comportamientos, El cambio de estado de los objetos, El ciclo de vida de un objeto, Un proceso de tipos de datos abstractos, Otro.
- Cuando construye un programa orientado a objetos, ¿en qué piensa? Construir soluciones elegantes, Reutilizar la mayor cantidad de código posible, Resolver el requerimiento pedido por el usuario, Hacer un programa bastante legible y de fácil crecimiento, Definir adecuadamente el comportamiento de los objetos, Hacer eficiente el uso del procesador y la memoria, Modelar los fenómenos mediante objetos y los conceptos por clases, Utilizar la mayor cantidad de instrucciones que el lenguaje le permite, Otro.
- ¿En que casos aplica la herencia? Catalogar objetos de distintos ambientes, No tener que definir los mismos campos, Realizar comportamientos independientes de la referencia, Modelar los fenómenos mediante objetos y los conceptos por clases, Utilizar la mayor cantidad de instrucciones que el lenguaje le permite, Otro.
- Cuando piensa en polimorfismo, ¿en qué piensa? Reutilizar el código, Incrementar la legibilidad de su programa, Implementar distintos comportamientos con el mismo bosquejo, Evitar sentencias de opciones, case o switch, Modelar respuestas a estímulos con la misma estructura, Otro.
- Cuando programa orientado a objetos, ¿cómo determina los métodos de sus clases? En respuesta a requerimientos de otros objetos, Como comportamiento de un objeto, Como parte del algoritmo requerido, Basado en un modelo dinámico, Como propiedades de sus atributos, Otro.
-

2.1.8 Análisis de resultados

Se realizó la evaluación para un grupo de 30 estudiantes, que ya han cerrado cursos de la Universidad de San Carlos y que han realizado, en su trabajo, desarrollos con el paradigma orientado a objetos contando con experiencias de más de 1 año en su mayoría. Como resultado de esta muestra tenemos que:

- El 50% de los estudiantes conciben un programa como un sistema y un 40% lo considera procesos y datos, mientras que el restante 10% lo considera un modelo.
- El 90% tiene un adecuado modelo mental de las variables y parámetros, mientras que un 10% no.
- El 60% tiene clara la relación objeto y referencia.
- Solamente el 30% de los estudiantes asocian adecuadamente clase a un concepto, mientras que el 30% lo asocia con objeto, y un 40% con un tipo de dato abstracto.
- Un 40% asocia adecuadamente un objeto a instancia; sin embargo, otro 40% lo asocia a un tipo de dato abstracto, y el resto no tiene clara la definición.
- La pregunta con respecto a la herencia es la más dispersa en opiniones, pues solamente el 40% de los estudiantes asoció adecuadamente el concepto a una jerarquía o a la especialización y generalización. El grupo restante no tiene el concepto bien definido.
- En cuanto al método, sólo el 30% de los estudiantes tiene el concepto claro como una comunicación entre objetos o un mensaje; sin embargo, el restante 70% lo asocia a función, procedimiento o algoritmo.
- Sólo el 30% tiene claro el concepto de estado de un objeto, y el grupo restante lo asocia a estructuras del lenguaje más que al concepto como tal.
- El 30% de los estudiantes inicia con el modelado antes de la programación, el grupo restante está orientado a la definición formal de acuerdo con el lenguaje de programación.

- El 40% tiene claro el concepto de programa en la programación orientada a objetos; el grupo restante aún mantiene en su modelo mental la estructura definida por el paradigma estructurado.
- El caso de aplicación de herencia es bastante disperso y sólo el 30% conoce en qué momentos es apropiado utilizarla.
- La mayoría de los estudiantes, es decir un 70%, conoce cómo aplicar el polimorfismo; sin embargo, no lo aplican porque no realizan la adecuada modelación inicial al momento de desarrollar un programa.
- Debido a la falta de modelación, el 100% de los estudiantes definen métodos conforme los van requiriendo y no como una modelación dinámica.

3 PROPUESTA DE LA ENSEÑANZA DE PROGRAMACIÓN ORIENTADA A OBJETOS

3.1 Síntesis de los resultados

Se puede notar de acuerdo con los resultados obtenidos de las evaluaciones realizadas al grupo objetivo, que el mayor porcentaje de los alumnos ya educados en el tema, tienen un modelo mental bastante sesgado por el modelo estructurado más que por el modelo de orientación a objetos, e intentan incorporar los nuevos conceptos sin tener bien conformado el nuevo modelo mental.

Es de notar también que no se realiza la modelación al momento de programar y se ve como una realización estructurada de tipos de datos abstractos, debido a la secuencia en que han sido enseñados los distintos paradigmas.

Otro aspecto que resalta es que aunque se conocen superficialmente algunos conceptos, no se tiene la capacidad de implementarlos debido a la falta de integración de éste modelo conceptual con la adecuada definición de los pasos necesarios para llegar hacia su representación en un programa de computadora, y por lo tanto no se aplican, pues el estudiante al momento de implementarlos los ve confusos.

3.2 La enseñanza

Usaremos como base para la enseñanza las lecturas activas, ya que los estudiantes ganarán buen entendimiento rápidamente.

Estas técnicas activas que se describen más adelante deben atacarse juntas, formando un ciclo de vida para un sistema dado, como para cada clase y tarea; así se refuerza la visión de enseñar haciendo.

3.2.1 Mostrar la pintura completa

La visión ha de ser presentar la "pintura completa", es decir, definir la posición de los modelos de la orientación a objetos con respecto al mundo real y cómo éstos se acoplan. De esta forma el estudiante sabrá de dónde vienen los modelos y cómo serán usados.

En un típico curso de programación en la universidad, iniciamos con aprender todas las construcciones programáticas. Conocemos los ciclos, las decisiones, las operaciones de entrada y salida, las variables, registro y punteros. No hay momento para la "pintura completa", quizás el próximo curso.

Para mostrar la "pintura completa" será necesario iniciar con enseñar qué son los objetos y las clases; luego, cómo ellos se relacionan con los objetos en el mundo real; inmediatamente después de esto, cómo representarlos en un lenguaje (será la herramienta de modelado UML). Entonces se continúa estudiando las relaciones entre las clases, similares a las relaciones entre la gente o las cosas. Estas relaciones tienen directa transformación en el lenguaje de programación como clases, punteros o argumentos de funciones. Además, se deben dar a conocer las interacciones entre objetos que forman un diagrama de interacción, los cuales se trasladan a una estructura de llamadas a funciones, y finalmente se da a conocer cómo juntar todo.

Mostrar la "pintura completa" es necesario pues hace que la comprensión de los detalles sea dramática. Debido a esto, para cumplirlo, es básico que se tenga una herramienta CASE disponible para cada estudiante (o por lo menos una computadora conectada a un proyector).

Esto después de presentar la teoría (notación, semánticas, etc.), la cual se puede mostrar con un problema relacionado. El estudiante entonces armará su "pintura completa". Luego, el profesor arma una solución común en una clase interactiva donde todos dan su opinión; y por último, el profesor muestra una solución que enlaza los detalles, mostrando todas las construcciones específicas de la herramienta de modelado.

3.2.2 Mostrar trazos claros

La "pintura completa" no puede estar completa sin los trazos claros que enlacen todos los modelos. Tales trazos son una de las grandes ventajas de la orientación a objetos. El comunicar estos enlaces entre modelos orientados a objetos es una de las tareas más importantes al enseñar sobre la orientación a objetos. Un buen diseño no es solo diagramas de clases bien estructurados o diagramas de secuencia con apropiados patrones de comunicación. Un buen diseño significa que es estrictamente moldeable a los requerimientos del usuario, y permite producir código entendible y eficiente. La única vía para lograr esto es mantener una completa dependencia entre todos los modelos orientados a objetos producidos.

Para mostrar los trazos claros debe haber un "mapa" de todos los modelos. Este mapa puede ser tratado como un esqueleto, donde todos los modelos y sus relaciones son presentados en forma de resumen. Luego, en posteriores presentaciones y lecturas, es llenado el esqueleto con el contenido, así se mantendrá la "pintura completa" presente en todo momento, lo que permite que el estudiante se cree una figura de cada modelo en relación con otros.

Mostrar las relaciones entre modelos es simple si se usa una herramienta CASE, pero es necesario hacer hincapié en relacionar los modelos creados previamente, por ejemplo: la tarea puede ser crear un escenario para casos de uso creados antes, o hacer diagramas de secuencia que relacionen estos escenarios con modelos de clases.

Otro aspecto es mantener el código sincronizado. Si queremos relacionar nuestros modelos al código, podemos mostrar código generado de la herramienta CASE haciéndolo durante una lectura activa. Esto no mostrará a los estudiantes detalles de construcciones de programación. Pero sí mostrará una forma distinta de programar, programando por diseño.

3.2.3 Mostrar técnicas de modelado

La limitante de mostrar buenos modelos y las guías para construirlos es que éstos deben completarse en una sola clase, por lo que deben ser pequeños. Pero no sistemas triviales, ya que casi en todos los métodos orientados a objetos hay varias técnicas para crear modelos, y estas técnicas están documentadas y probadas en proyectos reales. Vamos a tomar algunas; estas técnicas son activas pues envuelven a todos los estudiantes en grupos en una clase. A continuación la descripción de éstas.

3.2.3.1 Clase de casos de uso

En ésta se trata de modelar los casos de uso para el sistema. Debemos tratar de desarrollar escenarios específicos para cada uno o alguno de los casos de uso. Los participantes escriben el escenario usando una simple gramática. Para obtener una mejor capacidad de análisis individual, se puede emular una entrevista, en la cual uno o dos de los participantes jugarán el rol de usuarios y otro, u otros dos, pueden jugar el rol de analistas. El resto del grupo revisará la entrevista.

Al contar con más tiempo, se pueden cambiar participantes por cada uno de los casos de uso. Cuando el objetivo sea de conocimiento general sobre escenarios, se conducirá una clase de escenario de grupos en la cual todos los participantes desarrollarán una solución común.

3.2.3.2 Clases CRC

La idea es emular las tarjetas de responsabilidad, para lo cual simplemente reunimos a todos los estudiantes y realizamos una tormenta de ideas. El profesor es el moderador para que así pueda retroalimentar a los estudiantes sobre sus pensamientos de orientación a objetos.

Otra forma de emularla es dividiendo al grupo en dos, donde el primer grupo es el de usuarios y el segundo es el de analistas y diseñadores. Se da un problema y, al iniciar, ambos grupos discuten sus soluciones. Los usuarios se pondrán de acuerdo en un vocabulario y en sus requerimientos; los analistas y diseñadores pensarán en las clases, responsabilidades, y colaboradores que existen en el dominio del problema; luego, los requerimientos serán verificados con el diseño; así los estudiantes conocen la importancia de la colaboración y la comunicación que elimina la brecha entre lo que se define y lo que se programa.

3.2.3.3 Clases de juego de roles

En ésta cada uno de los participantes juega un rol de una de las clases diseñadas. Así, se diseñan adecuadamente los patrones de comunicación en la aplicación, además de ser una forma perfecta para enseñar el diseño dinámico del sistema y mostrar los trazos entre modelos. La relación con las tarjetas CRC es obvia; sin embargo, con los casos de uso no, por lo que se tendrá a otro participante que mantenga la pista del escenario que se está jugando. Tendremos además otro participante que produzca el diagrama de interacción o secuencia durante el juego.

La primera parte de la visión de la pintura completa será una o varias clases de casos de uso; luego, se llevará con trazos claros a una o varias clases CRC y finalmente las anteriores clases serán la base para una o varias clases de juego de roles.

3.2.4 Institucionalizar el método de enseñanza

La reforma de prácticas de enseñanza en la educación universitaria debe darse con esfuerzos de todos los miembros de la escuela. Un excelente primer paso es seleccionar estrategias que promuevan lecturas activas con las cuales se pueda sentir cómodo tanto el estudiante como el profesor.

Las personas que deciden pueden estimular y ser el soporte para los esfuerzos de los miembros de la escuela por resaltar la importancia del aprendizaje activo en las noticias y publicaciones que distribuyen.

Los directores de la escuela pueden ayudar a estas iniciativas reconociendo y premiando la enseñanza excelente en general y la adopción de innovaciones particulares en la instrucción. Los programas comprensivos para demostrar este tipo de compromisos administrativos deben dirigir políticas y prácticas de empleo institucional, la obtención de recursos adecuados para desarrollo de la forma en que se instruye y el desarrollo de los planes de acción de estrategias administrativas.

3.3 El contenido

La base inicial de la instrucción será interactuar con un programa, la introducción de nuevo material ha de ser manejada por un problema concreto. La motivación para la introducción de un nuevo punto viene de la realización de una tarea concreta. Esto puede ser combinado con técnicas adicionales de aprendizaje basadas en problemas.

3.3.1 El modelo mental

Es necesario reforzar el modelo mental, modelando fenómenos por objetos y conceptos por categoría de objetos. Ejemplo fenómeno: en la fonda de don Juan, Juan le sirve el churrasco a Luis en la mesa cuatro; objetos: fonda de don Juan, Juan, churrasco, Luis, mesa. Y conceptos: restaurante, mesero, comida, servicio.

3.3.2 Cuidado con las metáforas

Un aspecto importante al introducir cada punto es ser cuidadoso con las metáforas. Por ejemplo: en el manejo de eventos, si se usa la metáfora de un juego de tenis, donde el oidor de la red espera el evento “red” y un oidor de la línea espera el evento “fuera”. En el caso de variables y asignaciones, procure usar cuidadosamente la metáfora de la caja y su contenido; a fin de evitar confundir la asignación, introduzca la copia de contenido y procure erradicar el concepto de contención de valores simultáneos y el traspaso de información. En el caso de polimorfismo y enlazado dinámico: el caso de una orquesta donde el trompetista piensa en soplar más duro, el del bombo en golpear más duro y el director de la orquesta en sonar más ruidosamente. Lo adecuado es enseñar el modelo explícitamente.

3.3.3 Concepciones inválidas

Otro aspecto bastante importante es identificar las concepciones inválidas sobre el objeto, como objeto igual a variable pues éste define más que una variable de instancia; objeto igual a registro, el objeto no es sólo almacenamiento; la confusión de objetos contra clases: en cada ejemplo use al menos dos instancias de una clase. La identificación de un objeto contra el identificador definido explícitamente por el usuario.

3.3.4 El enfoque

Procure enfocarse en el fundamento de la programación y las cualidades de diseño. Un ejemplo de estas cualidades puede ser la alta cohesión y el bajo acoplamiento. Vea la programación como un proceso de modelado, "programar es describir". Exprese cada modelo mental de forma explícita y procure evitar analogías que provoquen concepciones inválidas. Tome en cuenta que se quiere construir un esquema adecuado para razonar en términos de éste, por lo cual es necesario que se centre en la ingeniería, más bien que en el algoritmo.

3.3.5 Metodología de desarrollo de cada tarea

Utilice la siguiente metodología en cada uno de las tareas que requiera al estudiante, trabajándola primero conjuntamente:

- Identifique los servicios provistos por el sistema (casos de uso).
- Para cada caso de uso:
- Escriba un párrafo describiendo la interacción entre el usuario y el sistema para el servicio que va a ser logrado.
- Dibuje un diagrama de dominio para representar la interacción descrita.
- para cada caso de uso:
- Trate de identificar los objetos que el sistema debe contener para satisfacer la descripción del caso de uso específico. Si cualquier objeto no puede ser representado directamente usando los tipos existentes, defina una clase para especificar la estructura y comportamiento de estos objetos.
- Dibuje un diagrama de interacción de objetos detallado que muestre los objetos del sistema y la forma en que éstos colaboran para satisfacer la descripción de caso de uso específico.

- Para cada diagrama de interacción de objetos, del paso previo identificar la contribución de cada objeto y definir los métodos correspondientes.
- Dibujar un simple UML, tal como diagrama de clases que muestre las clases del sistema y sus relaciones (agregación y generalización/especialización).
- Para cada clase del diagrama de clases:
- Dé una implementación usando el lenguaje acordado.
- Pruebe el comportamiento del objeto implementado.
- Integre los objetos en un programa.

3.3.5.1 Un ejemplo

A continuación, un ejemplo de cómo puede aplicar la anterior metodología como base para explicarla al estudiante. Descripción de un usuario típico de un restaurante:

Llego, esperaré en línea por una mesa; salgo para ocupar una mesa; me muevo a mi mesa; ordeno y como una entrada; ordeno y como un plato principal; ordeno y como un postre; pregunto y pago la cuenta; me voy. En otro caso, sucede que no obtengo una mesa, me resigno y salgo sin realmente entrar.

Ejemplo de descripción de un mesero que sirve a un invitado:

Dejo el baño, elijo al invitado en espera con más tiempo; me muevo a su mesa: Pregunto al invitado por su decisión, espero por la orden del invitado, me muevo a la cocina, espero por la comida, me muevo de regreso a la mesa y doy la comida al invitado. Me muevo al baño y me uno a los meseros esperando para trabajar.

Los objetos del sistema pueden ser: línea de espera, mesa, entrada, plato principal, postre, cuenta; invitado, cocina, mesero, orden.

FIGURA 1. Diagrama de secuencia

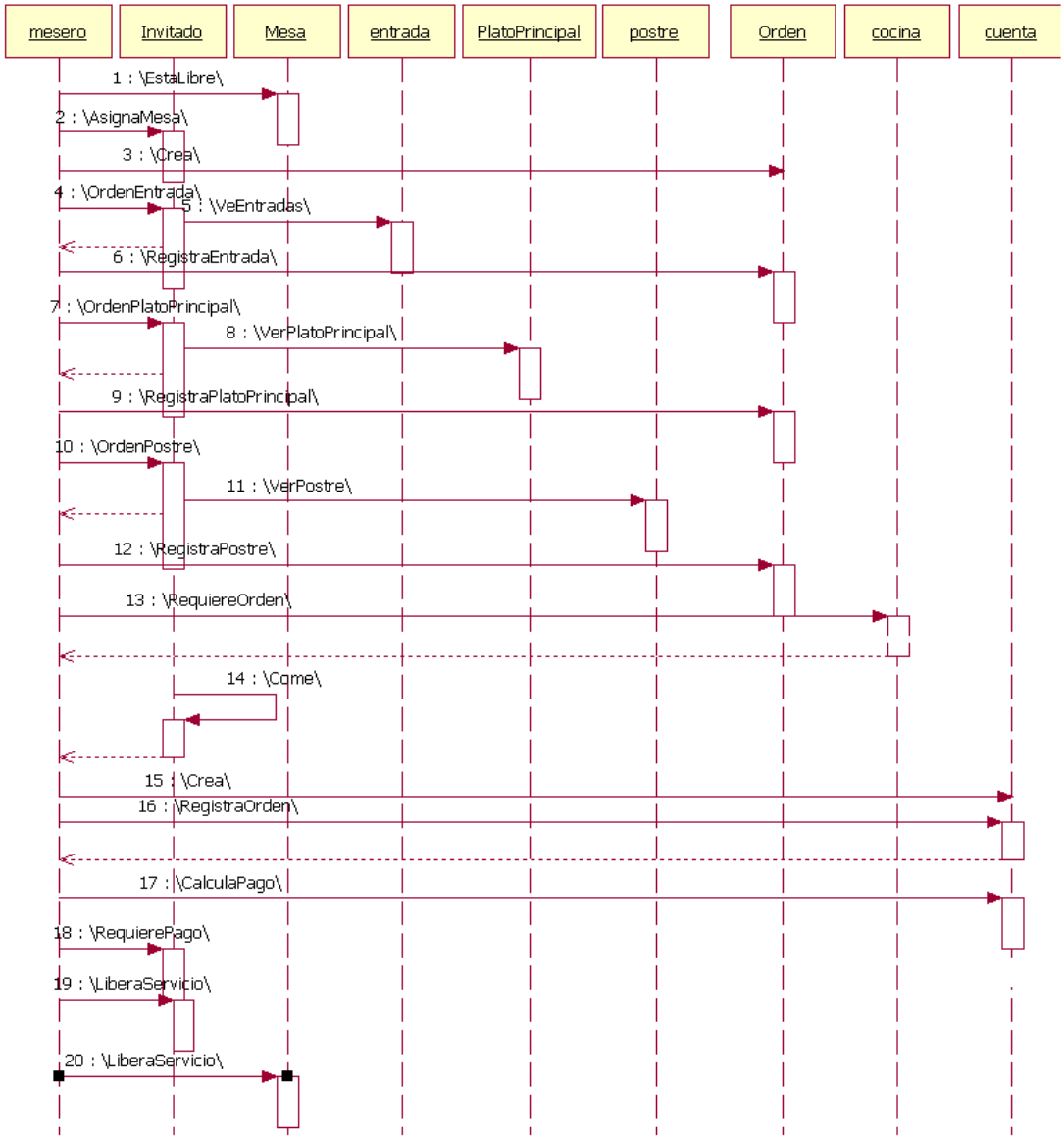
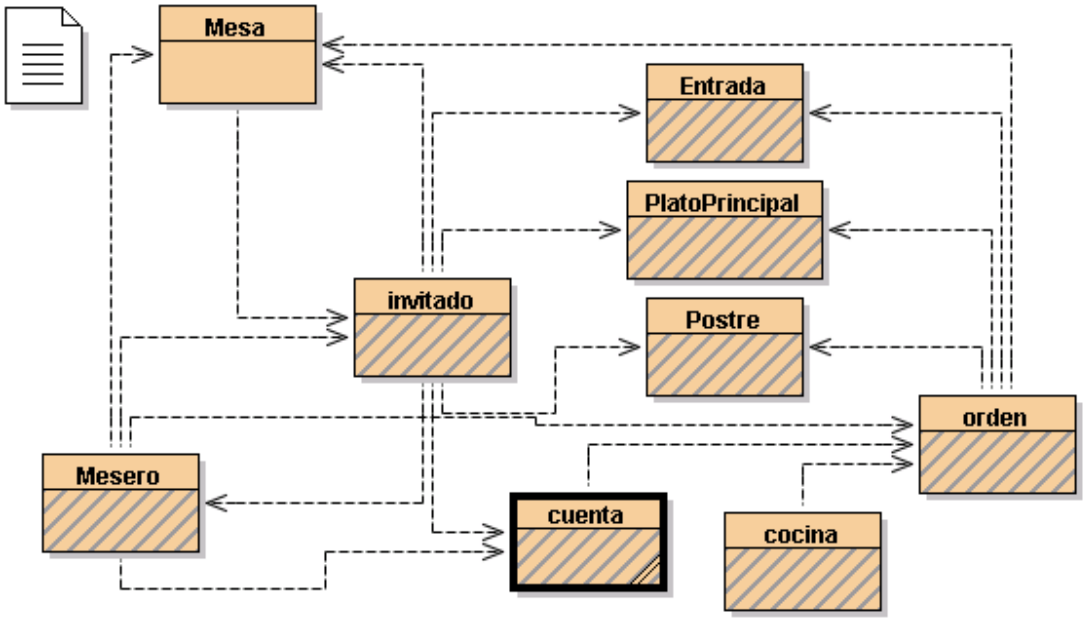


FIGURA 2. Diagrama de clases



Implementación en un lenguaje definido para el invitado:

FIGURA 3. Implementación en un lenguaje

```
public void vida() {
      llego();          // espero en línea
      if (Mesa.estaLibre()) {  // busco mesa
            AsignarMesa();
            OrdenEntrada();
            OrdenPlatoPrincipal();
            OrdenPostre();
            Come();
            RequierePago();
            LiberaServicio();
      } else                  // no obtuve mesa
            resigno();
   }
```

Para el mesero:

```
private void MeserodeInvitado() {
      salgo();                        // dejo el baño
      elInvitado=(Invitado) Object.InvitadosEnEspera.ObtenerPrimero();
      elInvitado.salgo();
      elInvitado.AsignarMesa();
      elInvitado.mesero=this;  // yo soy su mesero
      elInvitado.Activa();    // espero su orden
      LaOrden=(Orden)Object.Ordenes.Obtener();
      LaOrden.RegistraEntrada();
      LaOrden.RegistraPlatoPrincipal();
      LaOrden.RegistraPostre();
      Concina.RequiereOrden()
      elInvitado.Activa();
      Vacante();
}
```

3.3.6 Los ejemplos de objetos

Al iniciar la compresión OO, los ejemplos no deben ser difíciles ni inútiles.

En los cursos normales de programación orientado a objetos hay un primer grupo de objetos que son caracterizados por más de una representación, y éstos son usados con bastante frecuencia. Así se enfatiza el encapsulado de información, además de sus ventajas y aplicaciones. En esta categoría podemos incluir el ejemplo clásico de números complejos (con representaciones cartesianas y polares), el entero de múltiple precisión (implementado como un entero o un arreglo de dígitos), tiempo (expresado en diferentes unidades) y figuras geométricas como triángulos (representado por ángulos, lados o una combinación de ambos).

Un segundo grupo cubre los objetos tradicionales presentados en cursos de estructuras de datos, sólo que el acercamiento de orientación a objetos los dispone para ser introducidos a los estudiantes en una vía natural como parte inicial en la programación: éste es el caso de listas, pilas, colas y conjuntos; el análisis, evaluación y comparación de su representación alternativa.

Finalmente, hay objetos especialmente orientados a la herencia y derivación, independientemente de sus alternativas de representación. Algunos ejemplos son: polígonos (triángulos, cuadriláteros, rectángulos y cuadrados), persona (hombre, mujer), empleados (y administradores), roles universitarios (maestros, estudiantes, empleados), animales (perro, gato), etc.

3.3.7 El lenguaje que se debe usar

Siguiendo con la definición de Bergin sobre las características de un lenguaje apto para la enseñanza, su recomendación es usar el lenguaje BlueJ. Éste cumple con los requerimientos propuestos para la enseñanza, pues fue credo para eso; sin embargo, no es un lenguaje de alto uso en la industria, por lo que si se inicia enseñando la programación orientada a objetos mediante BlueJ será necesario encontrar el momento adecuado dónde realizar el cambio a un lenguaje de mayor uso en la industria; el lenguaje que más se acopla a las características definidas por Bergin, luego de BlueJ, es Java; por lo que será adecuada su utilización luego de haber realizado la introducción a la orientación a objetos con BlueJ.

4 PROPUESTA DE MODIFICACIÓN DE CONTENIDO PARA LOS CURSOS DE INTRODUCCIÓN A LA PROGRAMACIÓN I Y II

4.1 Contenido propuesto para el curso de Introducción a la programación I

4.1.1 (Sesiones 1 a 4) Armazón Conceptual

Mediante sesiones de caso de uso se describirá un sistema real. Éste puede ser un restaurante de comida rápida. La única restricción para esta asignación será que todos los estudiantes conozcan el sistema del mundo real, pues será a partir de este conocimiento que ellos construirán los casos de uso preliminares. Esta asignación permitirá conocer los conceptos básicos de sistemas, sus límites, las partes y cómo éstas interactúan, así como el ambiente externo. Se definirá el modelo mental concibiendo la realidad en términos de fenómenos y conceptos, modelando fenómenos por objetos y conceptos por categoría de objetos.

Mediante sesiones CRC, dividiendo a un grupo de alumnos como usuarios y otro grupo como desarrolladores y analistas, como se describió anteriormente, se identificarán las responsabilidades de los objetos encontrados en el sistema y las interacciones a la que responden, de forma que el modelo conceptual esté latente en ellos en lo que corresponde a comunicación en los sistemas y el manejo adecuado de responsabilidades, lo cual es la base para la orientación a objetos.

Además, usando sesiones de juegos de roles, los estudiantes podrán interactuar entre ellos realizando la acción correspondiente al sistema en cuestión, lo cual permitirá llevar el trazado de los casos de uso anteriores e identificar la interacción entre los componentes del sistema. De esta forma el estudiante es guiado en el descubrimiento del modelo mental de la orientación a objetos.

Esta armazón conceptual será armada con las sesiones tal como se muestran en la tabla II, donde la primera fila indica el número de la sesión, la segunda indica el tema, la tercera columna el subtema y la cuarta columna el ejemplo que será dirigido en la primera hora de clase, luego de la cual se le pedirá a los estudiantes que propongan un ejemplo diferente al que se les explicó y apliquen los conceptos:

Tabla II. (Sesiones 1 a 4) Explicación del armazón conceptual

Armazón conceptual				
	El restaurant macdonald's			
1		Sistema físico		El restaurante está conformado por el área de parqueo, el área de consumo, el área de caja, el área de entrega, el área de cocina, el área de menú y el área de recreación.
		Sistema		El sistema sirve comida de forma rápida
		Modelado		

Continuación…

			Fenómeno	Juan entra al restaurante, se dirige al área dónde ordenar y revisa el menú para identificar qué compra. Elige el combo 1 (big-mac, papas y coca cola), Juan le pide a José, quien registra su pedido que quiere el combo 1, José le pregunta si desea agrandar su combo por 2 quetzales más, Juan dice que sí, José entonces le pregunta si desea incluir algún postre a su orden, a lo cual Juan contesta que no. José entonces registra su pedido y le indica que son 30 quetzales. José entonces saca su billetera y revisa su efectivo, el cual no suma la cantidad a pagar, por lo que entonces decide utilizar su tarjeta de crédito y se la da a José. José entonces lleva su tarjeta de crédito y pide autorización al emisor de la tarjeta. Al pasar un momento, José obtiene la autorización y devuelve la tarjeta y el comprobante de su autorización a Juan. Entonces José entrega la orden de pedido a la cocina donde está María, quien toma de la fila de pedidos siempre el que lleva mayor tiempo en espera y realiza el pedido. Toma una rodaja de pan y la coloca sobre la mesa en la cual construye el pedido; toma una cucharada de mayonesa y la unta sobre la rodaja de pan;
				toma entonces una hoja de lechuga y la coloca sobre la rodaja de pan; luego, una rodaja de tomate que también coloca sobre la hoja de lechuga, luego un pedazo de carne y otra hoja de lechuga y dos rodajas de cebolla y dos de pepinillos. Vuelve entonces a tomar otra cucharada de mayonesa y la unta sobre la otra rodaja de pan y coloca esta parte del pan encima para construir el pedido. Luego toma papel aluminio y envuelve la hamburguesa colocándola en el área donde la puede tomar Carlos, quien basado en el pedido, si es para llevar toma una bolsa donde colocará las partes del pedido, pero si es para comer ahí tomará una bandeja para el mismo fin. Tomará entonces la hamburguesa, la colocará en la bolsa o la bandeja y tomará una medida de papas y llenará el vaso con gaseosa para completar la orden de Juan y se la entrega.

Continuación…

			Conceptos y diagramado de estos	cliente, cajero, caja, orden, menú, combo, ingredientes, precio, pago, emisor, empaque, contenedor de servicio, servidor, ración, vaso; esta será la base para representar los conceptos identificados mediante cuadros con su nombre en la parte superior (clases) e identificar la relación de uso y asociación o agregación existente entre ellas.
2		caso de uso	Actores	Determine la lista de objetivos de los actores propuestos: cajero O.1) cobrar el menú en el menor tiempo posible. O.2) ofrecer información sobre los menús y ofertas. O.3) llevar el control del dinero en la caja, cliente: O.1) comprar su alimento.
			Condiciones	
			Flujo básico	a) el cajero pregunta el combo que se va a consumir; b) el cliente pide el combo 1; c) el cajero anota el combo 1; d) el cajero pregunta si desea algún postre; e) el cliente dice que sí, siga en e.1; f) el cajero le indica el monto; g) el cliente paga con efectivo, siga en g.1; h) el cliente paga con tarjeta de crédito; i) el cajero pide autorización al emisor de la tarjeta; j) el emisor no devuelve autorización, siga en j.1; k) el emisor devuelve autorización; l) el cajero devuelve la tarjeta; m) el cajero entrega el comprobante al cliente; n) el cajero pide al cliente que espere.
			flujo alterno	g.1.a) el cliente entrega el monto al cajero g.1; b) el cajero determina el vuelto g.1; c) el cajero toma de los billetes de mayor valor en el vuelto para no quedarse sin sencillo g.1; d) el cajero da el vuelto al cliente, siga en m.
			Resultado	Realización de pedido

Continuación…

3		Tarjetas de responsabilidad		cajero R.1) anotar la comida que el cliente pida; R.2) ofrecer promociones y ofertas; R.3) informar de precios de los menús; R.4) calcular el monto a pagar; R.5) calcular el monto de vuelto; R.6) pedir autorización al emisor de la tarjeta; R.7) pedir a cocina el pedido; C.1) menú; C.2) emisor tarjeta; C.3) promoción; C.4) oferta; C.5) precio; C.6) efectivo; C.7) cliente.
4		Juego de roles		Un estudiante será el cajero, otro será el cliente, otro será la caja, otro será la orden, otro será el menú y actuarán el fenómeno descrito en la situación inicial; otro estudiante mantendrá la pista de los pasos que sus compañeros realizan mediante la construcción de un diagrama de interacción de objetos; esto servirá como base para la introducción de diagramas de colaboración.

4.1.2 (Sesión 5) Aplicación del armazón conceptual a un programa

Luego de haber completado el armazón conceptual del la orientación a objetos, se hará que los estudiantes interactúen con un programa que es implementado de acuerdo con este paradigma. Para esto se ha de usar el programa de calculadora de notación postfija inversa, con interfaz de usuario similar al del sistema operativo Windows. Ésta será dada a los estudiantes para que entiendan su comportamiento, entonces se les pide que usen la calculadora para evaluar el valor de la expresión 24-12*14-3 y describan en lenguaje natural, la interacción entre el usuario y el programa para evaluar la expresión antes mencionada.

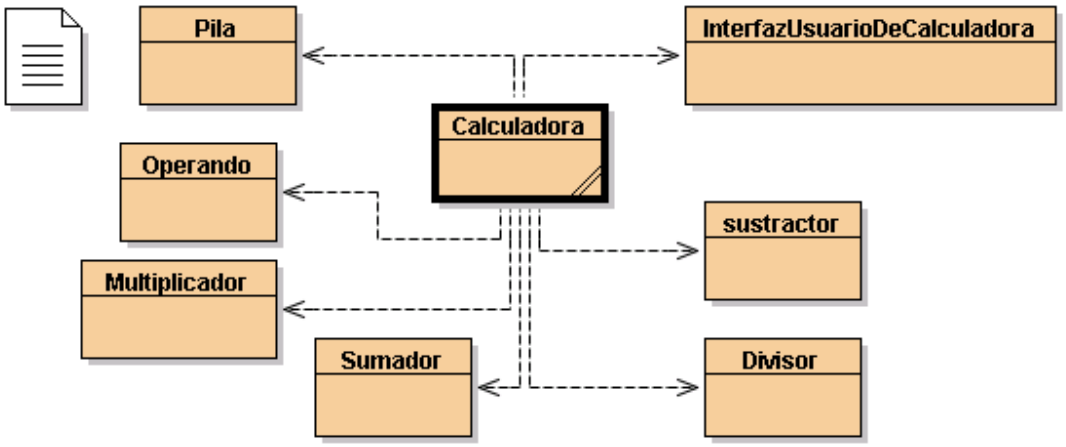
Luego se les pide que identifiquen los objetos que componen la calculadora y aumenten la descripción de la interacción con la referencia apropiada a estos objetos. Después deben representar gráficamente la interacción mediante un dibujo de un

diagrama de interacción de los objetos y el diagrama de clases preliminar usando la notación UML para clases y su agregación (tal como el que se muestra en la figura 4).

Tabla III. (Sesión 5) Aplicación del armazón conceptual

5		Interacción con un sistema de computadora		Mediante el sistema de la calculadora de notación postfija, se le pedirá al estudiante que construya un caso de uso correspondiente a la interacción del usuario con el sistema, por ejemplo, para la operación 24-12*14-3, luego de describir el fenómeno que identifique los conceptos o clases que componen el sistema. Además, para la interacción del usuario, también se debe realizar un diagrama de interacción para estas clases encontradas, sus responsabilidades y colaboradores.

FIGURA 4. Diagrama preliminar de clases de la calculadora NPI



4.1.3 (Sesión 6) Construcción de legos

Esto persigue enseñar la implementación, la idea es enfatizar en la construcción de legos e integrar los componentes predefinidos para formar en el estudiante la capacidad de hacer uso de las clases existentes desde el inicio y poder crear aplicaciones relativamente complicadas desde el inicio, pues sólo es requerido conocer los mensajes que han de ser enviados a las clases o instancias y la forma en que las clases o instancias responderán a éstos. Por eso será necesario haber construido un conjunto de clases específicamente desarrolladas para soportar esta idea.

Tabla IV. (Sesión 6) Construcción de legos

6		Construcción de legos		Se le muestra al estudiante un software ya realizado (la calculadora de notación postfija). Mediante la utilización de un ambiente de desarrollo (el cual puede ser BlueJ), se le muestra al estudiante cómo su construcción ha sido realizada mediante la inclusión de componentes. De igual forma, el estudiante puede operar los distintos objetos obteniendo los resultados de forma que experimenten con las clases en el ambiente BlueJ. En esta parte se introduce la instrucción inicial de un objeto, el punto inicial de ejecución de un programa mediante el enlace con el sistema operativo y el programa (principal correr), que da la vida al objeto en la computadora.

4.1.4 (Sesión 7 y 8) Responsabilidades de clases

En esta sesión se realizará el mapeo de la representación de clases al modelo formal de la computadora.

El estudiante es guiado a la representación de la clase y a la definición de los métodos asociados con ésta. El estudiante es guiado en el modelo de algoritmos y en los conceptos de variables, como los atributos y las sentencias que permiten realizar las responsabilidades de una clase. Se le pide al estudiante definir la clase Operando con la siguiente secuencia de teclas resultantes en la definición de la expresión 12 24 + =:

[1][3][BackSpace][2][Enter][3][4][CE][2][4][Enter][+][=]

Luego de haber definido la clase Operando, será necesario desarrollar un programa que pruebe este comportamiento, con lo cual será necesario desarrollar la prueba del comportamiento, tanto de la clase Operando, como de la clase Sumador y PresentadorDeResultados. Con lo cual se enfatizará la actividad correspondiente a la prueba de componentes.

Tabla V. (Sesión 7 y 8) Responsabilidades de clases

7		Responsabilidades de clases	Definición	A fin de practicar la definición en un esquema formal, se realiza la funcionalidad de evaluación para la secuencia de entrada del teclado [1] [BackSpace] [2] [Enter] [3] [CE] [2] [Enter] [+] [=]. En este caso se le debe dar al estudiante la clase que recibe los caracteres del teclado, a fin de evitar que el problema se convierta en cómo leer del teclado, y se le pide definir los conceptos en su representación de clase.
				conceptos: operando, sumador, resultado, teclado
			Identi-ficador	La representación única de un concepto.

Continuación…

			Métodos y estado	Mediante tarjetas CRC, se definirán las responsabilidades de los conceptos identificados en la funcionalidad anterior. Operando: R.1) guardar el valor, R.2) agregar digito al valor, R.3) borrar último dígito al valor, R.4) finalizar captación valor, R.5) devolver su valor C.1) teclado; en este punto se introduce la representación en un lenguaje formal del método de una clase y el ámbito del estado de un objeto para el caso de los atributos. Sumador: R.1) Sumar dos operandos, C.1) Operando, C.2) Resultado. Resultado: R.1) guardar el valor resultante, R.2) Mostrar el valor con formato.
			Variable	La caja de almacenamiento de datos, su copiado su asignación.
			Tipo básico	La representación de los valores.
			Parámetros	La copia de valores su representación y su referencia.
			Método	Todos los conceptos se usan en la representación formal en un lenguaje del método agregarDigitoAValor de la clase operando y también en la representación formal del estado de la clase operando, que en este caso es Valor.
8		Responsabilidades de clases 2	Definición	A fin de practicar la definición en un esquema formal, se realiza la funcionalidad de evaluación para la secuencia de entrada del teclado [1] [3] [BackSpace] [2] [Enter] [3] [4] [CE] [2] [4] [Enter] [+] [=] para este caso se le debe dar al estudiante la clase que recibe los caracteres del teclado, a fin de evitar que el problema se convierta en cómo leer del teclado, y se le pide definir los conceptos en su representación de clase.
			Colecciones y arreglos	Debido a la necesidad de llevar cada dígito por

Continuación…

				separado, se presenta al estudiante el concepto arreglo de caracteres en la clase String.
			Condi-ciones	A fin de realizar el cuerpo del método agregarDigitoAValor de la clase operando, se introduce la instrucción formal de representación de condiciones (if).
			Prece-dencia de opera-dores	Mediante ejemplos de evaluación de condiciones lógicas, se le explica al estudiante la forma de evaluación y agrupación adecuada en la evaluación de condiciones.

4.1.5 (Sesión 9) La interfaz y el bajo acoplamiento

La interfaz gráfica de usuario de la calculadora se le dará al estudiante (en código compilado) desde el inicio y se le requerirá escribir un programa demostrando la integración de la clase Operando, que se ha definido en la asignación anterior con la clase InterfazUsuarioDeCalculadora provista. Como el estudiante ya conoce el concepto de pasos de mensajes le será fácil integrar los objetos previamente definidos en un programa ya trabajando.

El programa dado a los estudiantes debe permitir definir el valor de una instancia Operando por medio de la interfaz de usuario provista. Además, la interfaz ha de soportar el uso de botones de dígitos así como el de backspace y reset. Entonces se le pide al estudiante dar la siguiente funcionalidad a la clase Operando: cuando el botón Enter sea presionado, el operando es insertado en la pila. El constructor de la clase InterfazUsuarioDeCalculadora ha de aceptar como un argumento una referencia al tipo Operando. Esta referencia es usada por la clase InterfazUsuarioDeCalculadora para comunicarse con la instancia Operando. El comportamiento de la instancia Operando con los mensajes iniciados por la clase InterfazUsuarioDeCalculadora debe estar

definido por los siguientes métodos: Void AgregarDigito(char ch); Void BorrarUltimoDigito(); Void Reset(); Void Completar().

Tabla VI. (Sesión 9) La Interfaz y el bajo acoplamiento

9		Interfaz de usuario gráfica	Mapa de bits	El concepto de cuadricula, y la representación por posiciones, será la base para la explicación de la interfaz gráfica (mapa de bits).
				Mediante la entrega al estudiante de la interfaz de usuario gráfica para la calculadora, se le pide integrar la clase que anteriormente desarrollo. Por lo cual el estudiante será motivado a la implementación de interfaz de usuario. Basada en gráficos; para este punto se le pedirá integrar su clase operando a esta interfaz de usuario; de esta forma comprenderá el concepto de bajo acoplamiento, pues ha realizado la clase operando y ésta puede ser usada tanto para la interfaz teclado, como para la interfaz gráfica.

4.1.6 (Sesión 10) Opciones de diseño

Con base en la anterior sesión, se evaluará la alternativa de diseño para la clase Operando. Esto provee una motivación excelente para los programadores novatos a fin de considerar la asociación de la representación de datos con la complejidad de los métodos producidos. Se le pide al estudiante considerar las siguientes alternativas para la presentación de valor del Operando: a) StringBuffer val; b) Double val;. Considerándolas mediante la escritura para cada representación de los métodos de la clase Operando. El objetivo es aclarar que el procesamiento algorítmico está bastante influenciado por la representación de la información. El estudiante verá que es adecuado usar un tiempo de investigación en la representación de los datos para simplificar los métodos que deben escribirse. Luego, a los estudiantes se les pide considerar las 2

alternativas de implementación del método Completar(): a la opción de convertir el miembro de datos val a Double y meterlo en la pila, o meter en la pila la instancia del Operando.

Tabla VII. (Sesión 10) Opciones de diseño

10		Alternativas de diseño		Se le plantea el efecto que tendría usar para el valor de la clase operando una variable tipo string o una variable tipo double. Y la alternativa de implementación del método finalizarCaptacion o Completar, entre guardar el valor del operando en la pila o guardar el operando en la pila.

4.1.7 (Sesión 11 y 12) Clases abstractas, herencia y polimorfismo

En este punto el estudiante ha alcanzado la etapa de conocer cómo programar basado en objetos. El momento es adecuado para el objetivo de aprendizaje; la programación orientada a objetos. Es el momento para introducir herencia y polimorfismo, y permitir al estudiante conocer estas asignaciones para practicar con construcciones del lenguaje que implementan estos conceptos. Se le pide al estudiante examinar la posibilidad de las siguientes clases de la calculador NPI: Sumador, Sustractor, Multiplicador, Divisor, y PresentadorDeResultados, a ser considerados como especializaciones de la clase Operador. Se le pide entonces el uso de herencia para representar en código la asociación indicada. Se discute qué pasa cuando el método que se hereda es redefinido por una subclase a fin de discutir el polimorfismo. En este punto es adecuado indicar que con polimorfismo pueden ser eliminadas las instrucciones if y switch. Se le pide al estudiante usar la notación UML para crear el diagrama de clases, como en la figura 2.

FIGURA 5. Diagrama reforzado de clases para la calculadora NPI

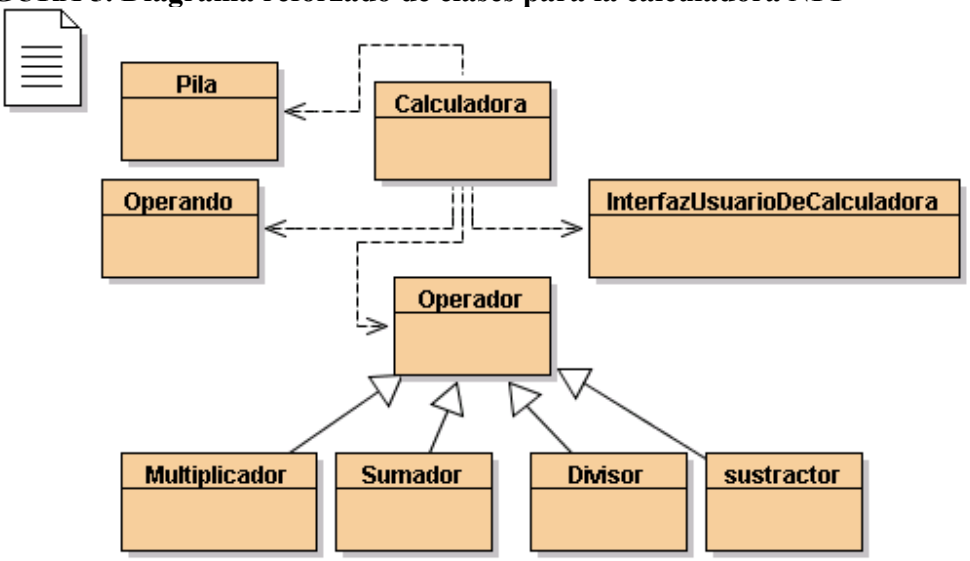


Tabla VIII. (Sesión 11 y 12) Clases abstractas, herencia y polimorfismo

11		Clases abstractas		Se definen las responsabilidades para las clases sumador, restador, multiplicador y divisor, así: sumador R.1) sumar operandos, C.1) Operando, C.2) Resultado; restador R.1) restar operandos, C.1) Operando, C.2) Resultado; Multiplicador: R.1) multiplicar operandos, C.1) Operando, C.2) Resultado; Divisor: R.1) dividir operandos, C.1) Operando, C.2) Resultado. Tendremos entonces los mismos colaboradores y la misma responsabilidad: operar operandos.
		Herencia		Tanto sumador, restador, divisor y multiplicador son especializaciones de operador y llevan consigo los mismos colaboradores. De igual forma, el operador generaliza la responsabilidad de sumador, restador, divisor y multiplicador, y siempre usará los colaboradores operando y resultado.
		Polimorfismo		Cada especialización opera de distinta forma, pero la generalización desconoce la forma en que cada uno lo hace, qué pasa al redefinir un método

Continuación…

				en la clase heredada, y cómo este concepto puede evitar el uso de condiciones.

4.1.8 (Sesión 13 y 14) Manejo de eventos

Esta asignación es usada para introducir lo básico del equipo de herramientas de ventana abstracta (awt). Se les pide a los estudiantes crear el desplegado de la clase InterfazUsuarioDeCalculadora. La ventana principal de la interfaz gráfica de la calculadora es una instancia de la clase Frame del paquete awt. En el primer paso sólo 3 objetos estándar de awt son usados: Frame, Button y TextField. Para ayudar al estudiante a manejar la complejidad de estas clases, es adecuado un resumen que describa los datos miembros y los métodos que son usados en el contexto de esta acción. Por ejemplo, los estudiantes son guiados para usar los siguientes métodos del Frame:

- setSize() // este método es heredado de Component para inicializar la dimensión de la ventana.
- setFont() // el método es heredado de Container para inicializar la fuente de la ventana.
- toFront() // el método es heredado de Window para traer la venta al frente.
-

Es muy importante no sobrecargar al estudiante en esta etapa con bastante detalle de awt, sino enfatizar sólo en los conceptos básicos. A través de este proceso, el estudiante es motivado a estudiar opcionalmente la definición de las clases arriba mencionadas y descubrir la utilidad de los métodos y miembro de datos. Esto es un buen ejercicio, examinar la forma en la cual está diseñada la librería básica de java y descubrir nuevos métodos para reforzar su programación.

El estudiante descubre el poder de la reutilización. A los estudiantes se les pide agregar comportamiento a la ventana de la calculadora que han realizado para que muestren un mensaje en el campo de despliegue de la calculadora cuando se presione el botón 0, entonces será el momento de introducir el manejo de eventos con el uso de diagramas de interacción de objetos.

Tabla IX. (Sesión 13 y 14) Manejo de eventos

13		Manejo de eventos		Mediante un diagrama de interacción de usuario e interfaz gráfica, se expone el manejo de eventos.
				Introducción a la librería de ventana abstracta, identificación de jerarquías en este diseño y construcción de propia interfaz de usuario; esto ayuda al estudiante a descubrir construcciones y aprender de código realizado por expertos (awt). Así como a identificar el esquema de eventos existente en la interacción de un usuario y la interfaz gráfica, que es la base de construcción de este diseño (diagrama de interacción de objetos).

4.1.9 (Sesión 15) Incrementando la reusabilidad

Se le da al estudiante el resto de las coordenadas de los botones de la ventana interfaz de la calculadora y se le pide completar su clase InterfazUsuarioDeCalculadora. Luego, se les da sugerencias para identificar la clase BotonDigito para evitar la repetición de código usado para crear e inicializar cada botón dígito como instancia de la clase Button. Él es guiado para evitar el uso de un manejador específico para cada botón digito y definir un manejador que puede actuar como un oidor de cada botón dígito. Se espera que definan ManejadorDeBotonDigito y BotonDigito.

Se introduce la construcción de interfaz. Se le pide al estudiante implementar la interacción entre los botones Operador y la correspondiente instancia Operador. El estudiante ahora notará que las subclases de operador tienen algo en común. Todas implementan la interfase ActionListener. Finalmente se les pide construir un diagrama de clases UML para representar la estructura de su InterfazUsuarioDeCalculadora.

Tabla X. (Sesión 15) Incrementando reusabilidad

15		incrementar reusabilidad		Cada botón de la interfaz gráfica de la calculadora es construido por medio de posiciones específicas en el cuadro de pantalla; a fin de evitar la inicialización de cada botón, se introduce una generalización para los botones que manejan los dígitos, de forma que sean construidos por una sola clase: botonDigito

4.1.10 (Sesión 16): Incrementando la flexibilidad

Un ejemplo de código para la clase InterfazUsuarioDeCalculadora es dado a los estudiantes. El código ha sido desarrollado sin aplicación de los principios de diseño básicos de la orientación a objetos. Se les pide a los estudiantes hacer lo siguiente: (a) dibujar el diagrama de clases del código dado, (b) identificar imperfecciones y rediseñar el diagrama de clases, y (c) modificar la estructura del código para estar de acuerdo con el nuevo diagrama de clases. Los estudiantes notan que un diseño orientado a objetos elimina el uso de la sentencia switch. Luego de esta asignación, los estudiantes obtienen un mejor entendimiento de la importancia del rol que un diagrama de clases juega en el desarrollo de un programa.

Tabla XI. (Sesión 16) Incrementando flexibilidad.

16		Incrementando flexibilidad		Se les da a los estudiantes el código desarrollado para la interfaz de usuario de la calculadora sin utilizar los principios de orientación a objetos; se construye con este código el diagrama de clases; se pide entonces identificar dónde no se aplicó el principio y se reconstruye el diagrama aplicando el principio. De esta forma, se intuye en el estudiante la necesidad de diseño, antes que construcción.

4.1.11 (Sesión 17) Encontrando errores

Un ejemplo de código para la calculadora NPI es dada y se le pide al estudiante identificar errores primero y luego agregar funcionalidad. Bastantes sugerencias son dadas para guiar a los estudiantes a identificar los errores y hacer las correcciones apropiadamente. Enfatizando en las actividades de lectura y trazado es útil para que las concepciones parciales lleguen a completarse.

Tabla XII. (Sesión 17) Encontrando errores

17		encontrando errores		Por medio del código realizado de la calculadora que debe tener ciertos errores, será necesario que los estudiantes los identifiquen, modifiquen el código y revisen el funcionamiento, a fin de inculcar en ellos la importancia del debug.

4.1.12 (Sesión 18) Excepciones y su verificación

Se le da al estudiante un conjunto de escenarios usando la calculadora NPI para forzar excepciones. Se le pide entonces modificar el código para que maneje las excepciones producidas.

Tabla XIII. (Sesión 18) Excepciones y su verificación

18		Verificando excepciones		Mediante casos de flujo alterno, se pide identificar el "qué pasa si" en los ejemplos anteriormente usados y realizar los pasos necesarios para capturar estas excepciones que pueden darse. De esta forma, se enseña la importancia del manejo de errores en un sistema.

A través del conjunto de sesiones, la idea de cada nuevo concepto ha sido construida. Por esto el estudiante, después de la misma, ha de recibir el código completo que ha sido requerido en la sesión. de forma que use el código: (a) como referencia de implementación para resolver algunas dificultades que tienen con asignaciones específicas y (b) como una base sobre la cual iniciar la nueva sesión.

El objetivo es concretar la idea en el estudiante de que la programación efectiva es raramente sobre la construcción de una pieza de código desde la nada, pero sí es una mezcla de reutilización y adaptación inteligente de código antes desarrollado.

4.2 Contenido propuesto del curso Introducción a la programación II:

4.2.1 (Sesión 1 y 2) Arquitectura

En esta sesión inicial, se introducirá el concepto de arquitectura con sus distintos aspectos que la definen, así como los distintos estilos de la misma.

Tabla XIV. (Sesión 1 y 2) Arquitectura

1		Arquitectura	Aspectos estructurales	Definición del sistema en un conjunto relacionado de componentes.
			Aspectos funcionales	Las responsabilidades y funcionalidad de cada componente y de cada conector.
			Aspectos de comporta-miento	La interacción de componentes en una situación o escenario.
2			Estilos de arquitectura	Distintos tipos de arquitecturas, sus principios básicos (C2, Unix Pipes and Filtres, Invocación implícita, basada en eventos, repositorios y sistemas en capas).
			OCL	Especificación de todos los tipos de restricciones, condiciones previas y posteriores sobre las operaciones y los métodos, garantías de los estados de transición y elementos del meta-modelo de clases de UML.

4.2.2 (Sesión 3) Definiciones arquitectónicas del sistema

La agilidad del estudiante ha de orientarse a conocer las peculiaridades de cada arquitectura planteada. Se les debe ahora inculcar el sentido crítico respecto al momento de utilizar determinada arquitectura, como lo muestra la siguiente tabla.

Tabla XV. (Sesión 3) Definiciones arquitectónicas del sistema

3		Definición arquitectónica del sistema		Cada arquitectura contiene características que será necesario tomar en cuenta al momento de plantear soluciones. Determinadas arquitecturas adoptan como estándar definiciones básicas para poder funcionar, con lo cual el estudiante debe conocer que tales aspectos han de ser tomados en cuenta al diseñar un sistema, y hay que cargar con estas premisas y elegir la más adecuada. Para lo cual se planteará al estudiante la necesidad de realizar un sistema de subastas. Sobre éste se plantearán determinados requisitos técnicos que han de cumplirse, y se determinará cual de las arquitecturas planteadas cubren de mejor forma los requisitos técnicos planteados.

4.2.3 (Sesión 4) La vista de usuario del sistema

Mediante esta sesión se describirá la funcionalidad de un simple sistema de subasta en línea, tal como lo muestra la siguiente tabla.

Tabla XVI. (Sesión 4) La vista de usuario del sistema.

4		Las distintas vistas del sistema	Vista de usuario	Esta fase se realizará con el afán de mostrar al estudiante la importancia de la adecuada definición de requerimientos, en la cual ha de ser experto para el cumplimiento de los mismos. Para eso se planteará un diagrama de casos de uso con los casos de uso significativos del sistema que representarán el corazón de la funcionalidad del sistema de forma que el estudiante aprenda a aplicar la regla de Pareto (80-20). Entre ellos estarán: c.u.1) **Crear Cuenta** (Con Ingreso): cómo las cuentas de usuarios son creadas y administradas, c.u.2) **Crear Subasta**: cómo las subastas son creadas y administradas, c.u.3) **Cerrar Subasta:** cómo las subastas llegan a su tiempo límite y son cerradas por el proceso interno de control de tiempo, y c.u.4) **Ofertar un producto** (con catálogos de subastas): cómo el sistema impone reglas de mínimo incremento de oferta y mantiene múltiples ofertas en un producto.

4.2.4 (Sesión 5) La vista lógica

Mediante esta sesión se mostrará el esquema que compone esta vista y cómo sus modelos concretan esa unidad lógica que define el sistema, de forma que el estudiante se forme el concepto de esta vista y vea cada parte de ella en el ámbito donde corresponde y cómo encaja en la pintura completa. El estudiante será guiado a llenar cada parte de esta vista, la cual incluye el modelo de análisis, de experiencia de usuario y de diseño.

4.2.5 (Sesión 6) Los Componentes de negocios

Se mostrará al estudiante la forma lógica de descomposición del sistema, de los artefactos de software necesarios para representar, implementar y publicar un concepto dado de negocios como un elemento autónomo y reutilizable de una aplicación grande.

Para esto se usa el sistema subasta en línea, el cual será descompuesto en tres componentes de negocios, un elemento común y componentes de servicios.

Cada uno de los componentes de negocios estará presente en tres capas: (1) Lógica de presentación (2) Lógica de negocios y (3) Lógica de integración. De esta forma, la arquitectura arma el sistema a lo largo de dos dimensiones: 1) dimensión a lo largo de la funcionalidad del sistema. y 2) a lo largo de las capas comúnmente reconocidas que se ocuparán de 2.a) presentación o cómo manejar la comunicación con el usuario y controlar su acceso a los servicios del sistema y a recursos, 2.b) negocios o cómo organizar los elementos del sistema que realizan los negocios y las funciones del sistema, y 2.c) integración o cómo conectar los elementos del sistema con los mecanismos de persistencia, otros sistemas, dispositivos físicos, etc.

Se tiene como resultado una matriz, como estructura del sistema donde cada elemento del diseño pertenece a un componente de negocios (o el elemento común y el componente de servicio) y una capa entre el componente/servicio. Estos componentes de negocios serán inducidos a comprensión en el estudiante de forma que elija los siguientes: C.1) Administración de cuentas de usuario, C.2) Administrador de Subasta y C.3) El catálogo de subastas.

4.2.6 Los patrones de diseño:

4.2.6.1 (Sesión 7) “Alta Cohesión y Controlador”

En esta sesión se introducirán los conceptos de patrones de diseño, aplicando el concepto basado en una situación planteada, tal como lo menciona la siguiente tabla.

Tabla XVII. (Sesión 7) Patrones "alta cohesión y controlador"

7		Patrones de diseño.	Alta cohesión	**¿Cómo mantener la complejidad manejable?** Es necesario mostrar a los estudiantes la forma adecuada de asignar responsabilidades, a fin de mantener una cohesión alta. Por lo tanto, debe organizar el sistema en capas discretas de forma que las capas inferiores sean los servicios en general y de bajo nivel, y las capas de alto nivel sean más aplicaciones específicas.
			Patrón controlador	**¿Quién debe ser el responsable de manejar un evento de entrada del sistema?** Muestre al estudiante que se debe asignar la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que represente todo el sistema o una que represente un escenario de casos de uso en el cual ocurre el evento. Sea claro en indicar que éste no es un manejador de la interfaz de usuario, más bien es quien define la operación del sistema y delega a otros objetos el trabajo que necesita ser hecho, es decir, controla la actividad sin hacer mucho él mismo. Sugiera que debería usarse el mismo controlador para todos los eventos del sistema de un caso de uso. Use como ejemplo la fachada o el controlador de caso de uso.

4.2.6.2 (Sesión 8) Patrones de diseño "experto y creador":

Los patrones de diseño serán fundamentales en la definición de programas; se continuará el descubrimiento de los mismos tal como lo menciona la siguiente tabla.

Tabla XVIII. (Sesión 8) Patrones "experto y creador".

8			Patrón experto	**¿Cuál es el principio general de asignación de responsabilidades a objetos?** Para enseñar esto al estudiante, defina el experto en la información, como aquella clase que tiene la información necesaria para completar la responsabilidad. Cuando para completar una responsabilidad es requerido llevar información en varias clases, implica que hay muchos expertos en la información que parcialmente colaboran en la tarea. La idea es que cada objeto use su propia información para completar su tarea y ése es el principio que debe quedar claro en el estudiante.
			Patrón creador	**¿Quién debe ser el responsable de crear una nueva instancia de alguna clase?** La regla a mostrar al estudiante será que una clase A podrá crear la instancia de una clase B, si A agrega objetos de B o si A contiene objetos de B, o si A guarda objetos de B o si A usa objetos de B y si A inicializa datos que serán pasados a A cuando se cree. La idea es soportar un bajo acoplamiento que permitirá menor dependencia de mantenimiento y mayor posibilidad de reutilización.

4.2.6.3 (Sesión 9) Patrones de diseño "bajo acoplamiento, polimorfismo"

Cada situación ha de intuir la adecuada solución de forma que sea descubierta por el estudiante, se continuará el descubrimiento de los patrones de diseño como lo menciona la siguiente tabla.

Tabla XIX. (Sesión 9) Patrones "bajo acoplamiento, polimorfismo y fabricación pura".

9			Bajo acopla-miento.	**¿Cómo soportar la baja dependencia, el bajo impacto en cambios e incrementar la reutilización?** Ya que el acoplamiento es una medida de a qué grado un elemento está conectado a, o tiene conocimiento de, o es tomado en cuenta en otros elementos, es necesario mostrar al estudiante que un elemento con bajo acoplamiento es aquél que no es dependiente de muchos otros elementos. La idea es mostrar que una clase con esta cualidad no es afectada por cambios en otros componentes del sistema.
			Patrón de polimor-fismo.	**¿Cómo manejar alternativas basadas en tipos? O ¿Cómo crear componentes de software armables?** La idea es mostrar al estudiante que no use el tipo de un objeto y la lógica condicional para desarrollar la variación de alternativas basadas en tipos; más bien lo que tiene que hacer es dar el mismo nombre al servicio en diferentes objetos, de esta forma una extensión requerida para una nueva variación será fácil de agregar al sistema sin afectar otras partes del mismo.

4.2.6.4 (Sesión 10) Patrones de diseño "fabricación puro, indirección y fábrica":

Cómo asignar la responsabilidad para evitar un alto impacto en las modificaciones.

Tabla XX. (Sesión 10) Patrones "fabricación puro, indirección y fábrica".

10			Patrón de fabricación puro	**¿Qué objeto debe tener la responsabilidad cuando no se quiere que viole el principio de alta cohesión y bajo acoplamiento?** En algunos casos será necesario crear clases que no representan un concepto del fenómeno que se está modelando; sin embargo, éstas serán creadas, pues manejarán un conjunto de responsabilidades que tienen la característica de alta cohesión.
			Patrón de indirección	**¿Dónde asignar responsabilidades para evitar acoplamiento directo entre dos o más cosas? ¿Cómo desacoplar objetos para que el bajo acoplamiento sea soportado y mantener alto el potencial de reutilización?** Para esto se debe enseñar al estudiante que debe asignar las responsabilidades a un objeto intermedio mediador entre otros componentes o servicios. Así ellos no estarán directamente acoplados. El intermediario crea una indirección entre los otros componentes.
			Patrón fábrica	**¿Quién debe ser el responsable de crear objetos cuando hay consideraciones especiales tales como lógica de creación compleja?** La idea es mostrar al estudiante que es deseable separar las responsabilidades de creación con lo cual se puede tener distintos métodos de creación para distintos tipos.

4.2.6.5 (Sesión 11) Patrones de diseño "adaptador, singlenton y estrategia"

Distintos problemas de diseño ha de conocer el estudiante para descubrir la mejor forma de resolverlos. Se continuará tal como lo muestra la siguiente tabla.

Tabla XXI. (Sesión 11) Patrones "adaptador, singlenton y estrategia".

11			Patrón adaptador	**¿Cómo resolver interfaces incompatibles o proveer una interfaz estable para componentes similares con interfaces diferentes?** Para esto debe mostrar al estudiante que debe convertir la interfaz del componente en otra interfaz a través de un objeto intermedio, este objeto intermedio será la solución. Este es un concepto sencillo, que ha de ser visto por el estudiante de forma elogiable para que comprenda que en un diseño simple resulta manejable.
			Patrón singlenton	**¿Qué hacer cuando exactamente una instancia de una clase es permitida ya que los objetos necesitan un solo punto de acceso global?** Explique al estudiante que aquí es aplicable el concepto de método estático de una clase el cual retorna la instancia única permitida. Use como ejemplo el manejo del acceso a la base de datos.
			Patrón estrategia	**¿Cómo diseñar para variados pero relacionados algoritmos? ¿Cómo diseñar para tener la habilidad de cambiar los algoritmos o políticas?** Muestre al estudiante que debe definir cada algoritmo en una clase separada con una interfaz común, usando el concepto de contexto y fuerte colaboración entre el contexto y la interfaz.

4.2.6.6 (Sesión 12) Patrones de diseño "compuesto y observador":

Cómo asignar la responsabilidad para evitar un alto impacto en las modificaciones.

Tabla XXII. (Sesión 12) Patrones "compuesto y observador".

12			Patrón compuesto	**¿Cómo tratar un grupo o composición de objetos en la misma vía como un objeto atómico?** Explique al estudiante que el uso de interfaz resulta adecuado en este tipo de problemas pues definiendo clases, una para el objeto compuesto y otra para el atómico sólo que implementado la misma interfaz, es la solución. Como ejemplo concreto puede usar algunas clases de la librería AWT para interfaces gráficos.
			Observador	**¿Qué hacer para que diferentes suscriptores dependan del cambio de estado o evento de un objeto y quieren reaccionar en su propia forma cuando el estado o evento suceda?** Para esto necesitará explicar al estudiante el concepto de publicador y suscriptor e interfaz de suscriptor de forma que el suscriptor implementa la interfaz y el publicador dinámicamente puede registrar suscriptores que están interesados en la notificación de un evento o estado cuando éste suceda. Como ejemplo concreto puede usar: la gestión de eventos de un interfaz gráfico o la representación de datos en una hoja de cálculo.

4.2.7 (Sesión 13) Vista de patrones

El concepto de diseño basado en modelos ya existentes y probados resulta en un diseño inicial con mayor posibilidad de éxito, ya que está basado en otras soluciones que han resultado exitosas en otros proyectos. De ahí que es adecuado inculcar en el estudiante la reutilización de soluciones exitosas mediante el diseño basado en éstas, tal como lo muestra la siguiente tabla.

Tabla XXIII. (Sesión 13) Vista de patrones

13		Vista de patrones.		De acuerdo con la idea de la pintura completa, se mostrará al estudiante cómo diseñar basado en los patrones que ya posee y cómo éstos unidos llegarán a influir en las decisiones de diseño del sistema.
			Diagrama a nivel de patrón	Es la creación de instancias de un patrón seleccionado y la identificación de las relaciones entre éstas. Para el sistema de subastas se definirán los patrones que lo componen.
			Diagrama a nivel de patrón con interfaces	Para el sistema de subastas se realizará el análisis de las relaciones entre los patrones, trasladando las relaciones entre patrones a relaciones entre interfaces y operaciones de los patrones.
			Diagrama a nivel de patrón detallado	Se incluirán las partes del patrón en el diagrama, de forma que muestre la relación de las interfaces u operaciones y las partes que componen el patrón.

4.2.8 (Sesión 14 y 15) Vista estática

Dentro de la vista estática, el estudiante debe conocer los elementos que la componen y cómo éstos se interrelacionan. Debe realizarse mediante el ejemplo del sistema de subastas.

Tabla XXIV. (Sesión 14 y 15) Vista estática

14			Modelo de dominio	Mediante el sistema de subastas, se definirá el modelo de negocio donde éste opera; se hará hincapié en que la utilidad de este modelo es la comunicación con el usuario.
			Diagrama de clases	Para el sistema de subastas, se realizará el diagrama de clases.
15			Reglas de Negocio.	Para el sistema de subastas, se dictaran las políticas, leyes físicas y leyes de gobierno que describen el enmarcado y el comportamiento sobre el cual el dominio opera. Ha de indicársele al estudiante que éstos no son requerimientos de la aplicación.
			Tarjetas CRC	Mediante el sistema de subasta, se construirá la tarjeta de responsabilidad de las clases que componen este sistema.

4.2.9 (Sesión 16 y 17) Vista dinámica

Esta vista ayudará al estudiante a llevar el diseño de sistemas de forma ordenada y con suficiente detalle, tal como lo muestra la siguiente tabla.

Tabla XXV. (Sesión 16 y 17) Vista dinámica

16			Diagrama de participan-tes	Se construirá, para cada caso de uso significante del sistema de subastas, las clases participantes y su dependencia.
			Diagrama de Colabora-ción	Para cada flujo básico de los casos de uso significantes, se construirá un diagrama de colaboración mostrando los mensajes y su orden de realización.
17			Diagrama de secuencia	Para cada flujo básico de los casos de uso significantes, se construirá un diagrama de secuencia mostrando el tiempo de ejecución y los mensajes realizados entre los objetos correspondientes.
			Diagrama de estado	Para los elementos correspondientes del sistema de subasta, se construirá un diagrama de estado que muestre el ciclo de cambio que está relacionado con el mismo.

4.2.10 (Sesión 18) Vista de proceso

Esta parte es importante ya que mostrará los requerimientos de ejecución del sistema y su ubicación en el modelo lógico.

Tabla XXVI. (Sesión 18) Vista de proceso

18			Vista de implementación	Para el sistema de subasta, se realizará la vista de proceso, indicando qué elemento del sistema corresponde a qué proceso, donde no todos los elementos de diseño serán mostrados pero algunos de ellos deben dar la idea al usuario.

4.3 Evaluación

A fin de hacer notables los avances a los estudiantes, y motivarlos a la investigación, se han de realizar preguntas sobre todos los temas que se esperan cubrir con la designación, a razón de unas 30 preguntas por tema, intentando tener un número bastante grande a fin de poder realizar tantos exámenes como se requieran, con un número considerable de preguntas. Para cada examen se elegirán al azar las preguntas de cada tema, no sólo las del tema que se ha finalizado de enseñar, sino que se dejarán al azar por cada tema provocando que al estudiante se le realicen incluso preguntas de las que aún no se les ha enseñado (es decir siempre se le preguntará algo de cada tema).

Algo que hay que tener en cuenta es no volver a utilizar las preguntas que ya fueron realizadas en anteriores exámenes. Así el número de preguntas a utilizar irá disminuyendo. Esto entonces permitirá que el estudiante se pueda comparar contra sí mismo respecto de su avance, pues siempre se le estará preguntando algo de cada tema en cada examen, es de esperarse que mejore a medida que va alcanzando el final. También ha de presentársele al alumno un indicador de su posición respecto al grupo, es decir un gráfico que le permita identificar su posición respecto del valor esperado en el grupo. Esto motivará al estudiante a mejorar su posición si está bajo la media.

CONCLUSIONES

1. Al enseñar programación orientada a objetos es necesario enfocarse en los fundamentos de la misma y las cualidades de diseño, viéndola como un proceso de modelado.

2. La programación orientada a objetos soporta elegantemente los conceptos de ingeniería de software, así como las técnicas para resolver problemas, por lo que ésta ha de ser la herramienta de formación adecuada para el estudiante.

3. Al iniciar la enseñanza de la programación mediante el paradigma orientado a objetos se evita el problema del cambio de paradigma que atraviesan los estudiantes del paradigma procedural al tratar de llegar a ser programadores de la orientada a objetos una vez que han aprendido la programación procedural.

4. Las sesiones de aprendizaje en la enseñanza de la programación orientada a objetos han de utilizar técnicas activas y patrones pedagógicos, ya que los estudiantes ganan entendimiento rápidamente.

5. La visión de presentar la pintura completa y los trazos claros en la enseñanza de la programación orientada a objetos facilita la comprensión de los detalles.

6. Las sesiones han de provocar el reconocimiento de concepciones inválidas e intentar reforzar el modelo mental adecuado a fin de formar en el estudiante el esquema mental que le permita pensar soluciones de diseño elegantes en el paradigma orientado a objetos.

7. La evaluación de la enseñanza de la programación orientada a objetos ha de ser más participativa, es decir, ser parte del proceso de aprendizaje del estudiante de forma que el juzgar su propio progreso lo motive a aprender.

RECOMENDACIONES

1. Institucionalizar el valor de una adecuada enseñanza de forma que se premie a los catedráticos que implementen técnicas de enseñanza que formen a los estudiantes, y no solo aquellas clases magistrales que provocan una barrera entre el conocimiento y la aplicación del mismo.

2. Evaluar el uso de los contenidos propuestos para los cursos de introducción a la programación I y II, para que formen al estudiante en los conceptos fundamentales de sistemas e ingeniería desde los inicios de la carrera y evitar los problemas causados con el cambio de paradigma.

BIBLIOGRAFÍA

1. Bergin, J. (2000, July, 2000). Fourteen Pedagogical Patterns for Teaching Computer Science. Paper presented at the Proceedings of the Fifth European Conference on Pattern Languages of Programs (EuroPLop 2000), Irsee, Germany.

2. Bertrand Meyer, Object-Oriented Software Construction, Prentice Hall, 1997

3. Doug Lea, Concurrent Programming in Java (tm) Second Editio n: Design Principles and Patterns, Addison-Wesley, 2000

4. IBM Rational Rose (2003). Visual Modeling With Rational Rose [Website]. Retrieved June 2003 en http://www.rational.com/products/rose/index.jsp

5. Kölling, M. (1999a). The Problem of Teaching Object-Oriented Programming, Part 2: Environments. Journal of Object-Oriented Programming.

6. Kölling, M. (1999b). Teaching Object Orientation with the Blue Environment. Journal of Object-Oriented Programming.

7. Kölling, M. (2001). BlueJ - Teaching Java [Website]. Retrieved September 2001 en : http://www.bluej.org

8. Kölling, M., & Rosenberg, J. (2001). Guidelines for Teaching Object Orientation with Java. Paper presented at the 6th conference on Innovation and Technology in Computer Science Education (ITiCSE 2001), Canterbury, UK.

9. Portal Educarchile : http:/ /www.educarchile.cl