7

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

GENERACIÓN DE CÓDIGO JAVA A PARTIR DE DISEÑOS EN UML

TESIS PROFESIONAL
QUE PARA OBTENER EL TÍTULO DE :

LICENCIADO EN INFORMÁTICA

P R E S E N T A :

JESÚS CUAUHTÉMOC GARCÍA FLORES

ASESOR:
DR. RICARDO RIVERA SOLER

MÉXICO, D.F. 2002

Paginación

Discontinua

Dedicatorias

A mi padre:

A. Jesús García López
Por la comprensión y paciencia que ha depositado en mí
y por la motivación para seguir adelante en todo momento de mi vida;
quién además, con sus consejos me ha ayudado a guiarme por el camino correcto.

A mi madre:

Guadalupe Flores Granillo
Quien me ha brindado día a día todo su amor y cariño
y ha apoyado en los momentos más difíciles.

Jesús Cuauhtémoc García Flores

Agradecimientos

A la UNAM:

Por darme la oportunidad de ser su alumno y por formarme tanto como persona como en lo profesional.

A mis maestros:

Por compartir sus conocimientos durante mi estancia de formación universitaria, y en especial a Santiago Suárez Castañón quién me apoyó y aconsejó en momentos importantes en mi vida de estudiante y al Dr. Ricardo Rivera Soler por apoyarme a culminar este gran paso en mi vida.

A la DGAA:

Por formar parte de su plan de formación de recursos humanos, en el cuál encontré personas con gran espíritu y adquirí conocimientos de todo tipo.

A TV-UNAM:

Por ser parte de su equipo de trabajo y en dónde conocí personas quienes me apoyan y brindan su amistad.

A mis amigos:

Andrés, Carlos, Claudia, Cynthia, Edgar, Elvira, Fabiola, Ivan, Jorge, Mario, Melissa, Norma y Salvador
Por brindarme su amistad desinteresada en todo momento y por compartir conmigo gratos momentos.

Jesús Cuauhtémoc García Flores

ADVERTENCIAS

A lo largo de la presente tesis se presentan temas con un nivel técnico, por tal motivo para una mejor comprensión, se recomienda que los lectores se encuentren involucrados con conocimientos generales de la tecnología de objetos, experiencias en UML y fundamentos de programación en Java, debido a que el contenido de la tesis expone cómo los modelos en UML pueden ser trasladados a código Java.

Introducción

INTRODUCCIÓN

Durante el desarrollo de software la utilización de modelos permite representar diferentes vistas de los sistemas. La claridad de los modelos es indispensable para construir sistemas robustos. Así surge UML, el cual es un lenguaje de modelado que esta compuesto de una notación basada en objetos. Los elementos de la notación aparecen en diferentes diagramas y son utilizados por diferentes métodos para expresar su diseño.

Los diagramas se crean para modelar el problema y para garantizar que al momento de entregar el software satisfaga los requerimientos de los usuarios o de la empresa. Así vemos que el modelado es importante, pero el producto principal de un equipo de desarrollo es el software; por tal motivo es importante que los modelos y la implementación correspondan entre sí.

Algunos de los elementos modelados con UML corresponden con código en un lenguaje orientado a objetos; un lenguaje en particular es Java, el cual es un lenguaje de programación orientado a objetos, el cual ha crecido notablemente para el diseño de aplicaciones y programas.

El presente trabajo de investigación esta compuesto de cinco capítulos.

El capítulo primero es el Marco Problemático, en el cual se identificó la necesidad e importancia de tratar el tema "Generación de código Java a partir de diseños en UML". Se aplicaron cuestionarios a personas calificadas en el tema, entre ellos personas con conocimientos teóricos, prácticos y profesionales; en base a sus opiniones se planteó la hipótesis propuesta.

El capítulo segundo esta formado por el Marco Teórico, en él se realizó la recopilación de la información y se integró lo relativo al conocimiento existente del tema. La base de la investigación y la profundidad en el conocimiento se adquirió al elaborar este capítulo.

Al capítulo tercero lo forma el Marco Conceptual, en éste capítulo se detalla y se desarrollan los temas de la investigación. Este capítulo se compone de conceptos y principios de la orientación a objetos, descripción de elementos, diagramas y vistas en UML y cómo los diagramas en UML se implementan a código Java; siendo este último el punto fundamental de la investigación.

El capítulo cuarto es el Marco Metodológico, aquí es en donde se presenta la comprobación de la hipótesis que se propuso en el Marco problemático. Se aplicaron otros cuestionarios para probar la hipótesis preliminar.

El quinto capítulo consiste del Marco Instrumental, en el cual se describen las actividades que se realizaron con motivo de la investigación realizada.

INDICE

Marco Problemático

1. MARCO PROBLEMÁTICO

1.1 Antecedentes

El gran desarrollo de nuevas técnicas de ingeniería de software para la elaboración de sistemas de información ha crecido notablemente en los últimos años, debido a las grandes necesidades de contar con otras alternativas que se adapten a los cambios que sufre la informática y el que cubran las exigencias organizacionales. Por tal motivo, actualmente el desarrollo de software se puede elaborar con otra forma diferente a la tradicional, dicha opción se basa en el paradigma de orientación a objetos.

Al desarrollar proyectos de software con técnicas de orientación a objetos conlleva múltiples ventajas durante su desarrollo, desde el análisis del sistema hasta realizar su construcción; en nuestros días esto facilita a los analistas, líderes de proyecto, programadores y diseñadores de software el desarrollo de aplicaciones para cualquier tipo de empresa o compañía, todo ello como resultado de que en la actualidad las empresas tienen la necesidad de automatizar su información.

Los métodos orientados a objetos permiten diseñar y construir sistemas confiables, robustos y flexibles, aún para aplicaciones complejas. Al ser una metodología ofrece su propio proceso de desarrollo, notación y herramientas, con los cuales se da solución a diferentes situaciones y problemas. Un proceso de software orientado a objetos esta formado por un conjunto de fases, técnicas, métodos y prácticas que la gente emplea para desarrollar y mantener el software, a este proceso se le asocia una serie de artefactos, como planos, documentos, modelos, código, pruebas y manuales; todo ello para producir un producto de software. Las etapas fundamentales de un proceso orientado a objetos son el análisis, diseño y construcción, todas ellas enfocadas con el paradigma orientado a objetos.

El análisis y el diseño orientado a objetos sitúa el estado actual de un problema y proporciona su solución lógica dentro de la perspectiva de los objetos, con la finalidad de tener el dominio sobre ese problema. El resultado de estas etapas consiste en generar una arquitectura del sistema a desarrollar, dicha arquitectura es representada en modelos que van a formar parte de la documentación; actualmente el UML se ha adoptado como la notación estándar en la mayoría de los procesos de software orientados a objetos. Una vez concluidos los diagramas durante el análisis y diseño, se dispone de suficientes detalles para comenzar a generar código y servirán de entrada para la construcción en un lenguaje orientado a objetos como Java; es decir, que sea útil para que los programadores puedan representarlos en código sin sufrir alguna forma de conversión.

Una ventaja del análisis, del diseño y de la programación orientada a objetos es ofrecer una guía completa de principio a fin para realizar la codificación a partir de los requerimientos. Con ello no se quiere decir que sea fácil ni que se pueda seguir mecánicamente, pues existen demasiadas variables, pero dan una aproximación real para resolver el problema.

1.2Identificación del problema

La parte esencial de un sistema de software es la investigación formal del problema y el proceso de diseño y no el apresurarse a codificar, ya que esto puede originar sistemas difíciles de entender, ampliar y de darles mantenimiento.

En general, el pasar de la etapa de diseño a la construcción durante el desarrollo de cualquier sistema, no es un paso fácil para la generación de código, sino todo lo contrario. En realidad los resultados obtenidos durante el diseño son un primer paso incompleto. Por la mala interpretación que se le dé a los modelos generados en las etapas de análisis y diseño para comenzar a codificar el sistema, existe gran probabilidad de desarrollar una aplicación deficiente, que constantemente sufra cambios y modificaciones, y con un mayor riesgo de que fracase.

Los artefactos producidos en las etapas de análisis y diseño en UML muchas veces no son del todo útiles para la construcción, debido a que no se conoce su correspondencia en código Java o es complicado traducirlo con facilidad. Por tal motivo la construcción del sistema termina mucho después del tiempo programado y por consiguiente su liberación será pospuesta.

1.3Demarcación del fenómeno

Esta investigación se realiza con el fin de que todas aquellas personas con conocimientos generales de la tecnología de objetos, experiencias en UML y fundamentos de programación en Java; que tengan interés en adquirir un entendimiento teórico - práctico de cómo implementar en Java diseños en UML, puedan ponerlo en práctica durante el desarrollo de software. Aún para que analistas, programadores, líderes de proyecto y todos los involucrados en el desarrollo de software, tengan una visión general de la transición entre la arquitectura y la codificación del sistema, sin ser expertos en UML o en un lenguaje orientado a objetos.

1.4 Conocimiento empírico en el medio

Para obtener información acerca del trabajo de investigación se realizó un cuestionario, el cuál se le aplicó a diversas personas involucradas y con conocimientos del tema.

1.4.1 Definición de las preguntas

Pregunta	Justificación	Respuesta esperada
1. En orden del 1 al 3, ¿Cuál es la etapa a la que se le debe dedicar mayor tiempo durante el proceso de software?	Conocer la prioridad en tiempo que se le da al diseño y a la construcción.	*Análisis: 1* *Diseño: 2* *Construcción: 3*
2. ¿Que notación o notaciones conoce para el análisis y diseño de sistemas orientados a objetos?	Como existen una serie de notaciones para modelar sistemas orientados a objetos, ver cual es la más conocida y utilizada.	*UML*
3. ¿Cree que los modelos en UML son útiles para representar y comunicar elementos que componen al sistema?	Remarcar la importancia del UML en el desarrollo del sistema.	*Si*
4. ¿Cree usted que sería fácil hacer la codificación del sistema sin apoyarse de modelos?	Ver que tan importante es disponer de modelos y diagramas para programar el sistema.	*No*

5. ¿Qué conceptos de orientación a objetos conoce, que puedan ser aplicados al desarrollo de modelos UML?	Saber cuales son los conceptos del paradigma orientado a objetos que pueden ser representados en diagramas en UML.	*Clases, objetos, herencia, relaciones, polimorfismo, encapsulación, métodos, interfaces, etc.*
6. ¿Qué lenguajes orientados a objetos conoce?	Saber cual es el lenguaje más utilizado.	*Java* *C++*
7. ¿Qué ventajas encuentra al pasar de un diseño con UML a un lenguaje orientado a objetos?	Conocer cuál es la solución principal que aportan los modelos en UML al pasar a la construcción.	*Facilita el paso a la codificación.*
8. ¿Cree usted que si un programador utiliza modelos en UML para comenzar a codificar el sistema, sea necesario que tenga cierto conocimiento de la notación?	Ver si la transición de UML a Java requiere de conocimientos de la notación.	*Sí*
9. ¿Cómo ha visto que ha sido aceptado el desarrollo orientado a objetos en empresas encargadas de producir software?	Conocer la difusión y la aceptación del paradigma orientado a objetos.	*Positivamente*

1.4.2 Formato del cuestionario

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

CUESTIONARIO

Nombre: _____

OBJETIVO: Obtener una opinión empírica para el tema de tesis "Generación de código Java a partir de diseños en UML" aplicado por Jesús Cuauhtémoc García Flores.

INSTRUCCIONES: *Responda las siguientes preguntas.*

1. ¿En orden del 1 al 3 cual es la etapa a la que se le debe dedicar mayor tiempo durante el proceso de software?

   Análisis ( )
   Diseño ( )
   Construcción ( )

2. ¿Que notación o notaciones conoce para el análisis y diseño de sistemas orientados a objetos?

   _____
   _____

3. ¿Cree que los modelos en UML son útiles para representar y comunicar elementos que componen al sistema?

   ( ) SI ( ) NO

4. ¿Cree usted que sería fácil hacer la codificación del sistema sin apoyarse de modelos?

   ( ) SI ( ) NO

5. ¿Qué conceptos de orientación a objetos conoce, que puedan ser aplicados al desarrollo de modelos en UML?

   _____
   _____
   _____

6. ¿Qué lenguajes orientados a objetos conoce?

7. ¿Qué ventajas encuentra al pasar de un diseño con UML a un lenguaje orientado a objetos?

8. ¿Cree usted que si un programador utiliza modelos en UML para comenzar a codificar el sistema, sea necesario que tenga cierto conocimiento de la notación?

( ) SI ( ) NO

9. ¿Cómo ha visto que ha sido aceptado el desarrollo orientado a objetos en las empresas encargadas de producir software?

( ) Positivamente ( ) Regular ( ) Negativamente

1.4.3 Aplicación del cuestionario

Las personas a las que se les aplicó el cuestionario fueron las siguientes:

Mario Alberto Cortés Ulloa
IQuatro

Andrés Garibay Tierradentro
El Universal

Edgar Hernández Rojas
e-Siglo

Jorge Ivan Pérez Ramírez
Infotec

Ma. Teresa Ventura Miranda
Subdirección de Sistemas, UNAM

1.4.4 Recopilación

1. En orden del 1 al 3 ¿Cuál es la etapa a la que se le debe dedicar mayor tiempo durante el proceso de software?

	Mario Cortés	***Andrés Garibay***	***Edgar Hernández***	***Ivan Pérez***	***Ma. Teresa Ventura***
Análisis	1	1	1	1	1
Diseño	2	2	2	2	2
Construcción	3	3	3	3	3

Conclusión: Las personas entrevistadas coincidieron en que a la construcción del sistema se le debe de dedicar menor tiempo, esto demuestra que el análisis y diseño del sistema debe de tomar el tiempo necesario, debido a que es la parte medular durante el desarrollo del sistema.

2. ¿Qué notación o notaciones conoce para el análisis y diseño de sistemas orientados a objetos?

Mario Cortés	***Andrés Garibay***	***Edgar Hernández***	***Ivan Pérez***	***Ma. Teresa Ventura***
UML	UML	UML	UML	UML Booch

Conclusión: Todos coincidieron en que UML es la notación más conocida de entre todas las existentes para desarrollar sistemas orientados a objetos.

3. ¿Cree que los modelos en UML son útiles para representar y comunicar elementos que componen al sistema?

Mario Cortés	***Andrés Garibay***	***Edgar Hernández***	***Ivan Pérez***	***Ma. Teresa Ventura***
Sí	Sí	Sí	Sí	Sí

Conclusión: Es necesario contar con modelos cuando se está desarrollando un sistema. Y más que nada como una herramienta para representar tanto el comportamiento del sistema como su arquitectura. Son una parte esencial dentro del desarrollo de software, para comunicarse con el equipo de trabajo.

4. ¿Cree usted que sería fácil hacer la codificación del sistema sin apoyarse de modelos?

Mario Cortés	***Andrés Garibay***	***Edgar Hernández***	***Ivan Pérez***	***Ma. Teresa Ventura***
No	No	No	No	No

Conclusión: Los modelos de diseño son necesarios para comenzar a construir el sistema, van a comunicarle a los programadores cómo debe de funcionar el sistema y les va a facilitar el empezar a generar código.

5. ¿Qué conceptos de orientación a objetos conoce, que puedan ser aplicados al desarrollo de modelos en UML?

	Mario Cortés	Andrés Garibay	Edgar Hernández	Ivan Pérez	Ma. Teresa Ventura
Clase	X	X	X	X	X
Objeto	X	X	X	X	X
Herencia	X	X	X	X	X
Polimorfismo	X				
Encapsulación					X
Mensaje		X	X		
Modularidad		X			X
Reutilización		X			
Jerarquía	X				
Abstracción					X

Conclusión: Objeto, clase y herencia son los conceptos más conocidos y aplicados por las personas entrevistadas que utilizan modelos en UML. Otros como mensaje, jerarquía, polimorfismo y encapsulación son menos representados.

6. ¿Qué lenguajes orientados a objetos conoce?

Mario Cortés	Andrés Garibay	Edgar Hernández	Ivan Pérez	Ma. Teresa Ventura
Java C++	Java	Java C++	Java	Java C++

Conclusión: Java es el lenguaje orientado a objetos más conocido.

7. ¿Qué ventajas encuentra al pasar de un diseño con UML a un lenguaje orientado a objetos?

Mario Cortés	Transparencia para desarrollar los objetos al iniciar la construcción del programa.
Andrés Garibay	Que precisamente UML fue creado como un modelador de desarrollos en lenguajes orientados a objetos, de hecho sirve también para cualquier sistema, pero para aprovechar todas las ventajas que nos ofrece el UML es mejor que el desarrollo sea orientado a objetos.
Edgar Hernández	Reutilización de código y portabilidad.
Ivan Pérez	Que al haber realizado un análisis y diseño con orientación a objetos desde un principio, la implementación es más transparente.

Ma. Teresa Ventura	Es más transparente cuando un modelo esta lo suficientemente detallado, por lo que se ahorra tiempo y se fomenta la reutilización.

Conclusión: Se encuentran diversas ventajas desde la perspectiva de los entrevistados como: la correspondencia entre modelos y código y la reducción de tiempos.

8. ¿Cree usted que si un programador utiliza modelos en UML para comenzar a codificar el sistema, sea necesario que tenga cierto conocimiento de la notación?

Mario Cortés	*Andrés Garibay*	*Edgar Hernández*	*Ivan Pérez*	*Ma. Teresa Ventura*
Sí	Sí	No	Sí	Sí

Conclusión: La mayoría considera necesario conocer la notación UML para poder implementar los modelos en un lenguaje de programación orientado a objetos.

9. ¿Cómo ha visto que ha sido aceptado el desarrollo orientado a objetos en las empresas encargadas de producir software?

	Mario Cortés	*Andrés Garibay*	*Edgar Hernández*	*Ivan Pérez*	*Ma. Teresa Ventura*
Positivamente	X		X		
Regular		X		X	X
Negativamente					

Conclusión: Algunos coinciden en que ha tenido auge la orientación a objetos y otros creen que no ha sido totalmente aceptada por la comunidad dedicada al desarrollo de software.

1.5 Opiniones profesionales

Para obtener más información acerca del trabajo de investigación, se aplicó un cuestionario a personas profesionales en el tema. De acuerdo a la información que se esperaba obtener, el contenido del cuestionario para personas expertas en el tema tiene algunas variantes que el de personas con conocimiento empírico, debido a que se busca recabar otro tipo de información.

1.5.1 Definición de las preguntas

Pregunta	Justificación	Respuesta esperada
1. **¿En orden del 1 al 3 cual es la etapa a la que se le debe dedicar mayor tiempo durante el proceso de software?**	Conocer la prioridad en tiempo que se le da al diseño y a la construcción.	*Análisis 1* *Diseño 2* *Construcción 3*
2. **¿Cómo definiría el paradigma orientado a objetos?**	Conocer el concepto de orientación a objetos del entrevistado.	*Es una forma de solucionar un problema a través de objetos, tal como se ve en el mundo real.*
3. **¿Que notación o notaciones conoce para el análisis y diseño de sistemas orientados a objetos?**	Como existen una serie de notaciones orientadas a objetos, ver cual es la más utilizada.	*UML*
4. **¿Cree usted que sería fácil hacer la codificación del sistema sin apoyarse de modelos?**	Ver que tan importante es disponer de modelos y diagramas para programar.	*No*

5. ¿En qué proporción los modelos en UML generados durante las etapas de análisis y diseño le son útiles al codificar en un lenguaje orientado a objetos?	Ver si en realidad es útil la documentación generada durante el diseño del sistema al programarlo.	*Mucha*
6. ¿Cuáles conceptos de orientación a objetos ha utilizado en modelos UML y en código Java?	Saber cuales son los conceptos del paradigma orientado a objetos al utilizarse diagramas en UML para codificar.	*Clases, objetos, herencia, relaciones, polimorfismo, variables, métodos, interfaces, etc.*
7. ¿Qué utilidad le ha demostrado que tengan los modelos en UML al codificar?	Notar las ventajas y soluciones que aportan los modelos al aplicarlos a situaciones reales.	*Son útiles para representar y entender el sistema en desarrollo.*
8. ¿Cree que los modelos en UML son útiles para representar y comunicar elementos que componen al sistema?	Remarcar la importancia del UML en el desarrollo del sistema.	*Sí*
9. ¿Cuál es la ventaja que encuentra al pasar de un diseño orientado a objetos a un lenguaje orientado a objetos?	Conocer cual es la ventaja principal que aportan los modelos en UML al pasar a la construcción.	*Facilita el paso a la codificación.*

10. ¿Cree usted que se requiere tener conocimientos de UML para traducir los modelos a código orientado a objetos?	Ver si la transición de UML a Java es fácil de realizar.	*Sí*
11. ¿Qué lenguajes orientados a objetos ha utilizado para codificar sistemas?	Saber cual es el lenguaje más utilizado	*Java* *C++*
12. ¿Cómo ha visto que ha sido aceptado el desarrollo orientado a objetos en empresas encargadas de producir software?	Conocer la difusión y la aceptación del paradigma orientado a objetos.	*Positivamente*

1.5.2 Formato del cuestionario

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

CUESTIONARIO

Nombre: _____

OBJETIVO: Obtener una opinión profesional para el tema de tesis "Generación de código Java a partir de diseños en UML" aplicado por Jesús Cuauhtémoc García Flores.

INSTRUCCIONES: *Responda las siguientes preguntas.*

1. ¿En orden del 1 al 3 cual es la etapa a la que se le debe dedicar mayor tiempo durante el proceso de software?

   Análisis ( )
   Diseño ( )
   Construcción ( )

2. ¿Cómo definiría el paradigma de la orientación a objetos?

   _____
   _____

3. ¿Que notación o notaciones conoce para el análisis y diseño de sistemas orientados a objetos?

   _____
   _____

4. ¿Cree usted que sería fácil hacer la codificación del sistema sin apoyarse de modelos?

   ( ) SI ( ) NO

5. ¿En qué proporción los modelos en UML generados durante las etapas de análisis y diseño le han sido útiles al codificar en un lenguaje orientado a objetos?

( ) Toda ( ) Mucha ( )Medianamente ( ) Poca ( ) Nada

6. ¿Cuáles conceptos de orientación a objetos ha utilizado en modelos UML y en código Java?

7. ¿Qué utilidad le ha demostrado que tengan los modelos en UML al codificar?

8. ¿Cree que los modelos en UML son útiles para representar y comunicar elementos que componen al sistema?

   ( ) SI ( ) NO

9. ¿Cuál es la ventaja que encuentra al pasar de un diseño orientado a objetos a un lenguaje orientado a objetos?

   ( ) Facilita el paso a la codificación
   ( ) Es más fácil entender la arquitectura propuesta
   ( ) Se reduce el tiempo de codificación
   ( ) Otra:
   Especifique:

10. ¿Cree usted que se requiere de tener conocimientos esenciales de UML para traducir los modelos a un lenguaje orientado a objetos?

    ( ) SI ( ) NO

11. ¿Qué lenguajes orientados a objetos ha utilizado para codificar sistemas?

12. ¿Cómo ha visto que ha sido aceptado el desarrollo orientado a objetos en las empresas encargadas de producir software?

    ( ) Positivamente ( ) Regular ( ) Negativamente

1.5.3 Aplicación del cuestionario

Las personas a las que se les aplicó el cuestionario fueron las siguientes:

Renato Barahona Neri
Becario DS-DGSCA

Juan Carlos Chávez García
AAIDA Organismo Integrado de Distribución S.A. de C.V.

Gerardo León Lastra
Televisión Universitaria, TV-UNAM

Abraham Omar Lugo León
Televisión Universitaria, TV-UNAM

Edgar Mendoza Arreola
Circulo Interware de México S.C.

1.5.4 Recopilación

1. En orden del 1 al 3 ¿Cuál es la etapa a la que se le debe dedicar mayor tiempo durante el proceso de software?

	Renato Barahona	*Juan Carlos Chávez*	*Gerardo León*	*Omar Lugo*	*Edgar Mendoza*
Análisis	1	1	1	1	2
Diseño	2	2	2	2	1
Construcción	3	3	3	3	3

Conclusión: La mayoría de las personas entrevistadas coincide en que a la etapa de análisis es a la que hay que dedicarle más tiempo en el desarrollo de un sistema, posteriormente la etapa de diseño y finalmente la construcción del sistema. Con esto se puede ver que en la implementación del sistema deben de reducirse los tiempos de desarrollo.

2. ¿Cómo definiría el paradigma de la orientación a objetos?

Renato Barahona	Es un paradigma que descompone los elementos (tanto físicos como lógicos) que conforman un sistema en objetos, así mismo, se presentan las relaciones entre estos y entre estos y el ambiente.
Juan Carlos Chávez	Es una metodología para abstraer y resolver los problemas de automatización de información, desde un enfoque del mundo real, a sistemas informáticos.
Gerardo León	Descomposición que agrupa datos y procedimientos en bloques relacionados.
Omar Lugo	Permite definir y agrupar todas las partes que integran la solución de nuestro problema en clases que funcionan como tipos de datos, que, una vez instanciadas, dan forma a objetos concretos interrelacionables, e incluyen métodos aplicables a distintos problemas, permitiendo la reutilización de componentes.
Edgar Mendoza	Como una forma de abstraer la realidad basado en responsabilidades, conocimiento e interacciones de entidades relevantes en el contexto del problema que se desea resolver.

Conclusión: Todos consideran que la orientación a objetos es una forma de solucionar problemas a través de objetos, los cuales van a colaborar para la realización de una tarea o función dentro de un sistema.

3. ¿Qué notación o notaciones conoce para el análisis y diseño de sistemas orientados a objetos?

Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
UML	UML	UML	UML	UML

Conclusión: Todos coincidieron en conocer como notación para lenguajes orientados a objetos el Lenguaje de Modelado Unificado (UML).

4. ¿Cree usted que sería fácil hacer la codificación del sistema sin apoyarse de modelos?

Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
No	No	No	No	No

Conclusión: Todos consideran necesario el contar con modelos para pasar a la construcción del sistema.

5. ¿En que proporción los modelos en UML generados durante las etapas de análisis y diseño le son útiles al codificar en un lenguaje orientado a objetos?

	Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
Toda	X	X			
Mucha			X	X	X
Medianamente					
Poca					
Nada					

Conclusión: La mayoría considera que son de mucha utilidad los modelos en UML para comenzar a codificar el sistema, los demás consideran que es total la utilidad de esos diseños. Con esto se puede ver que los modelos con UML son útiles al pasar del diseño a la codificación.

6. ¿Cuáles conceptos de orientación a objetos ha utilizado en modelos UML y en código Java?

	Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
Clase	X	X	X	X	X
Objeto	X	X	X	X	X
Herencia	X	X	X	X	X
Polimorfismo	X	X	X	X	X
Encapsulación		X	X	X	X
Mensaje		X	X	X	

Jerarquía	X	X	X	X	X
Visibilidad					X

Conclusión: Todos han trabajado con clases, objetos, herencia, polimorfismo y jerarquía de clases durante el diseño y codificación de un sistema; por otro lado algunos también han manejado la encapsulación, mensajes y la visibilidad. Esto depende del nivel de profundidad de los diagramas.

7. ¿Qué utilidad le ha demostrado que tengan los modelos en UML al codificar?

Renato Barahona	Debido a la flexibilidad existente en los modelos UML (representan diferentes vistas del sistema) se facilita tanto la codificación como la implantación y unificación de conceptos, ya que se representa lo que el sistema debe hacer y es más sencillo para el programador traducir estos modelos a un lenguaje de programación OO que si tuviera los mismos modelos pero relativos a otro paradigma.
Juan Carlos Chávez	Considero que constituyen una guía indispensable para la buena estructuración, organización y elaboración de código, ya que se conocen de antemano los objetos a crear y la interacción entre éstos para resolver el problema.
Gerardo León	Notación suscinta.
Omar Lugo	Como metodología de diseño, nos permite visualizar el problema de forma muy cercana a la codificación, además nos muestra el flujo de información y la relación que existe entre las clases definidas. El hecho de ver de forma escrita las clases que vamos a utilizar y los atributos que las describen nos da la visión de lo que vamos a construir con el lenguaje.
Edgar Mendoza	Claridad en requerimientos y alcance del sistema. Comprensión de la arquitectura propuesta.

Conclusión: Todos coinciden en que los modelos en UML dan solución a aspectos como la organización y arquitectura de sistema, y facilita su implementación debido a que muestra aspectos que corresponden en código.

8. ¿Cree que los modelos en UML son útiles para representar y comunicar elementos que componen al sistema?

Renato Barahona	***Juan Carlos Chávez***	***Gerardo León***	***Omar Lugo***	***Edgar Mendoza***
Sí	Sí	Sí	Sí	Sí

Conclusión: Se coincidió en que los modelos con notación UML que dan como resultado en las etapas de análisis y diseño, facilitan el entendimiento y solución del problema, por el cual se desarrollará el sistema.

9. ¿Cuál es la ventaja que encuentra al pasar de un diseño orientado a objetos a un lenguaje orientado a objetos?

	Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
Facilita el paso a la codificación			X	X	
Fácil entendimiento de la arquitectura	X			X	X
Reducción en tiempos de implementación		X	X	X	

Conclusión: En general, tanto los modelos son útiles para reducir tiempo al codificar, así como para el fácil entendimiento de la arquitectura. Pero también para unos fue importante el considerar que son de suma importancia para facilitar el paso a la codificación.

10. ¿Cree usted que se requiere de tener conocimientos esenciales de UML para traducir los modelos a un lenguaje orientado a objetos?

Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
Sí	No	No	Sí	Sí

Conclusión: De las personas entrevistadas, tres coinciden en que es recomendable conocer elementos esenciales del UML para su codificación; las otras dos personas no lo creen necesario.

11. ¿Qué lenguajes orientados a objetos ha utilizado para codificar sistemas?

Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
Java	Java	Java y C++	Java	Java y C++

Conclusión: Java es uno de los lenguajes orientados a objetos más conocido y utilizado dentro de la industria del desarrollo de sistemas.

12. ¿Cómo ha visto que ha sido aceptado el desarrollo orientado a objetos en las empresas encargadas de producir software?

	Renato Barahona	Juan Carlos Chávez	Gerardo León	Omar Lugo	Edgar Mendoza
Positivamente	X	X	X	X	
Regular					X
Negativamente					

Conclusión: La mayoría coincide en que el desarrollo de software orientado a objetos a sido aceptado positivamente, por tal motivo, esta forma de elaborar sistemas cada vez ha sido más aceptada por los analistas, desarrolladores y líderes de proyecto para la solución de problemas.

1.6Hipótesis preeliminar

Variable independiente

Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la interpretación de ellos en código Java al comenzar la codificación del sistema...

Variable dependiente

- Facilitará la transición entre el diseño y la construcción del sistema.
- Permitirá la claridad y concordancia de los modelos con el código.
- Comunicará transparentemente la arquitectura del sistema en base a objetos entre diseñadores y programadores.
- Permitirá reducir los tiempos de elaboración del software en la etapa de construcción.

Proposición positiva

- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la interpretación de ellos en código Java al comenzar la codificación del sistema;* facilitará la transición entre el diseño y la construcción del sistema.
- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la interpretación de ellos en código Java al comenzar la codificación del sistema;* permitirá claridad y concordancia de los modelos con el código.
- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la interpretación de ellos en código Java al comenzar la codificación del sistema;* comunicará transparentemente la arquitectura del sistema en base a objetos entre diseñadores y programadores.
- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la interpretación de ellos en código Java al comenzar la codificación del sistema;* permitirá reducir los tiempos de elaboración del software en la etapa de construcción.

Proposición negativa

- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la no interpretación de ellos en código Java al comenzar la codificación del sistema;* no facilitará la transición entre el diseño y la construcción del sistema.

- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la no interpretación de ellos en código Java al comenzar la codificación del sistema;* no permitirá claridad y concordancia de los modelos con el código.

- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la no interpretación de ellos en código Java al comenzar la codificación del sistema;* no comunicará transparentemente la arquitectura del sistema en base a objetos entre diseñadores y programadores.

- *Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos de diseño en UML, la no interpretación de ellos en código Java al comenzar la codificación del sistema;* no permitirá reducir los tiempos de elaboración del software en la etapa de construcción.

1.7 Objetivos

1.7.1 Personales

1. Obtener el título profesional como Licenciado en Informática por medio del desarrollo de un trabajo de investigación llamado Tesis, de acuerdo al Reglamento General de Exámenes, Capítulo IV - Exámenes Profesionales y de Grado, Artículos del 44 al 56.

2. Ampliar mis conocimientos sobre el paradigma orientado a objetos, la notación UML y Java; como elementos de suma importancia para el desarrollo de software orientado a objetos.

3. Obtener el conocimiento para entender que varios modelos de análisis y diseño en UML tienen significado y correspondencia en términos de lenguaje Java.

4. Interpretar los modelos en UML para codificarlos con facilidad en un lenguaje orientado a objetos como Java, todo ello con el fin de tener un dominio del tema y aplicarlo a problemas reales dentro del contexto laboral.

1.7.2 Particulares - Generales

1. Aportar una guía que de a conocer cómo los aspectos esenciales de la orientación a objetos pueden ser representados en modelos UML y en un lenguaje orientado a objetos como Java.

2. Conocer la utilidad de los modelos desarrollados en UML durante las etapas de análisis y diseño orientado a objetos, y la relación directa que tienen con un lenguaje orientado a objetos.

3. Probar que mediante las descripciones en UML es posible construir componentes Java, por lo que no se requiere de alguna forma de conversión al pasar de una etapa a otra.

Marco Teórico

2. MARCO TEÓRICO

2.1 Acopio de libros

2.1.1 Libros de estudio

Nombre: ***The Essence of Object-Oriented Programming with Java and UML***
Autor: *Bruce E. Wampler, Ph.D*
Editorial: *Addison-Wesley*
Edición: *Primera*
ISBN: *0-201-73410-9*

Capítulo 1
Objetos, UML y Java

El desarrollo de software a través del paradigma de orientación a objetos permite diseñar sistemas robustos y de fácil mantenimiento, todo ello a través de representaciones gráficas en UML que permiten el entendimiento del sistema. Java es un lenguaje de programación orientado a objetos con el que se pueden realizar distintas aplicaciones.

Capítulo 2
La esencia de los objetos

La orientación a objetos es una forma diferente para resolver problemas y desarrollar su solución. Por tal motivo los sistemas han sido diseñados a través de la abstracción de objetos, clases, mensajes, jerarquías y polimorfismo. Un objeto representa una instancia de una cosa real o abstracta. Una clase es la definición de atributos y métodos necesarios que hacen referencia a un grupo de objetos; así mismo las clases pueden ser organizadas en jerarquías y pueden asociarse. La herencia es cuando una subclase deriva de una superclase. Y el polimorfismo consiste en la utilización de métodos por parte de objetos que pertenecen a una subclase.

Capítulo 3
Objetos en Java

Este capítulo trata acerca de Java como un verdadero lenguaje orientado a objetos, se definen métodos y atributos de una clase y se dan convenciones del lenguaje. La visibilidad para atributos y métodos, y los tipos de asociaciones entre clases. Java soporta la herencia múltiple a través de mecanismos como las interfaces. Los constructores sirven para inicializar objetos al construirlos. Los mensajes son implementados como llamadas a métodos de otras clases.

Capítulo 4

Análisis y diseño orientado a objetos

En la actualidad existen una serie de metodologías orientadas a objetos. Todas incluyen de alguna forma las fases de planeación, construcción y liberación. La parte inicial del proyecto debe ser la colaboración cliente-desarrollador. En el AOO se deben de encontrar los objetos del sistema. En el DOO se refinan los resultados del AOO; posteriormente continúa la codificación, pruebas e integración del sistema.

Capítulo 5
Interfaces gráficas de usuario orientadas a objetos con Swing

En este capítulo se realiza un ejemplo de una aplicación basada en GUI, donde se utilizan contenedores y componentes que responden a ciertos eventos. Básicamente se muestra una construcción en Java con elementos Swing.

Capítulo 6
Un caso de estudio en Java

Se construye una pequeña aplicación, se hace el análisis del problema y se representan en diagramas de clases en UML, para posteriormente diseñar varias clases que se implementarán, todo en ello en Java y Swing.

Capítulo 7
Diseño de patrones

El diseño de patrones se ha convertido en una parte elemental del software orientado a objetos. Los patrones son definidos usando una plantilla que describa su uso. Y existe una clasificación de ellos.

Nombre: ***Object-Oriented Software Development in Java: Principles, Patterns and Frameworks***
Autor: *Xiaoping Jia*
Editorial: *Addison-Wesley*
Edición: *Primera*
ISBN: *0-201-35084-X*

Capítulo 1
Desarrollo de software orientado a objetos

El desarrollo de software orientado a objetos consiste del análisis, diseño e implementación. En el análisis se modelan los aspectos esenciales referentes al problema, el cual se soluciona a través de objetos, clases y sus relaciones. Existen principios fundamentales del desarrollo orientado a objetos, tales como modularidad, abstracción, encapsulación y niveles de abstracción. La relación entre clases e interfaces permiten conocer su herencia, implementación, asociaciones, agregaciones y composiciones.

Capítulo 2
Introducción a Java

Se dan a conocer las características de Java como un lenguaje orientado a objetos, distribuido e independiente de la plataforma. Se compara con otros lenguajes de programación. Java tiene dos tipos de programas, las aplicaciones y los applets. Se ilustra la estructura básica de Java y sus herramientas de desarrollo.

Capítulo 3
Elementos de Java

Java soporta dos tipos de datos: primitivos y de referencia. Dentro de los datos primitivos están el boolean, byte, short, int, long, char, float y double; y los de referencia son los objetos y los arreglos. Se definen elementos tales como clases, interfaces, paquetes y excepciones como elementos primordiales del lenguaje.

Capítulo 4
Clases y herencia

Existen conceptos en Java como métodos y constructores que pueden ser sobrecargados, solo si tienen diferente firma. La herencia es un mecanismo por el cual se puede reutilizar la implementación de una superclase a una subclase. La sobreescritura es cuando un método de una clase tiene la misma firma y tipo de retorno. Java no soporta la herencia múltiple, para realizarla se deben de usar interfaces.

Capítulo 5
Diseño por abstracción

Los patrones son una descripción esquemática de soluciones a problemas comunes dentro del diseño de software; es decir, cada patrón representa una solución genérica a un problema. Los componentes genéricos son programas que representan los componentes, tales como clases y paquetes.

Capítulo 6
Frameworks en aplicaciones orientadas a objetos

Un framework es un conjunto de clases que representan diseños reutilizables de sistemas de software en una aplicación en particular. Un objeto "*collection*" contiene un conjunto de otros objetos, los cuales son llamados elementos de la colección. Se dan a conocer elementos de AWT, como componentes de la interfaz del usuario.

Nombre: ***Object Oriented Software Design & Construction with Java***
Autor: *Dennis Kafura*
Editorial: *Prentice Hall*
Edición: *Primera*
ISBN: *0-13-011264-X*

Capítulo 1
Conceptos básicos

La abstracción es una técnica que toma las características principales de un elemento; dentro del desarrollo de software permite conocer los atributos y comportamiento de una entidad. La separación va a permitir el clasificar entidades de acuerdo a su comportamiento. Otros conceptos son las clases y objetos. La composición consiste en organizar varios sistemas en uno general. La generalización identifica características y comportamiento comunes entre entidades.

Capítulo 2
Utilización de objetos de una misma clase

Se definen clases y se crean objetos en Java. En este capítulo se toma la notación UML para representar clases y objetos. La estructura de las clases en Java consisten de variables y métodos. Se explica la sobrecarga de métodos.

Capítulo 3
Utilización de objetos de diferentes clases

Dentro de un sistema la comunicación entre objetos representa la parte esencial del software, por ello se relacionan las clases. Los diagramas de secuencia muestran las interacciones entre los objetos participantes. Las interfaces consisten de un número de métodos que los va a implementar una clase.

Capítulo 4
Implementación de una clase

Se proporcionan las propiedades para implementar una clase en términos correctos. La relación de agregación consiste en describir los componentes que forman a un todo, la agregación tiene algunas ventajas como la simplicidad, seguridad, especialización, estructura y substitución. Existen dos tipos de agregación: la estática y la dinámica.

Capítulo 5
Elaboración de un sistema orientado a objetos

Un sistema de software debe de contemplar algunas consideraciones con respecto a las clases antes de implementarlas: el diseño, las pruebas, la agrupación y la documentación. El diseño consiste en poner en practica la abstracción para descubrir las clases y evaluarlas; las pruebas deben de detectar errores y fallas de las clases; y la documentación debe ser generada para describir las clases en cuanto a su entendimiento y uso dentro del sistema.

Capítulo 6
Herencia

La generalización permite mostrar la herencia de clases, esto con la finalidad de que las clases compartan su implementación. La herencia múltiple no es permitida en Java, por tal motivo la forma de solucionar este problema, es a través de la implementación de interfaces. La jerarquía de clases va de la mano con la herencia.

Capítulo 7
Construcción de interfaces de usuario en Java

La interfaces en Java se realizan a través de dos librerías: AWT y Swing, cada una contiene sus propias clases para construir las GUI´s. En este capítulo se muestran la estructura de las interfaces y sus principales características como color, componentes, fuentes, bordes, etc.

Nombre: ***El Lenguaje de Modelado Unificado***
Autor: *Ivar Jacobson, Grady Booch y James Rumbaugh*
Editorial: *Addison-Wesley*
Edición: *Primera*
ISBN: *84-7829-028-1*

Capítulo 1
Por qué modelamos

El modelado de un sistema es parte importante durante el desarrollo de software, debido a sirven para abstraer el problema y facilitan la comunicación entre las personas que construirán el producto. Así el propósito principal de utilizar modelos en UML es ayudar a especificar, visualizar, construir y documentar un sistema.

Capítulo 2
Presentación de UML

UML es un lenguaje de modelado utilizado para visualizar, documentar, construir y documentar los artefactos de un sistema de software. Los sistemas orientados a objetos pueden ser representados con la notación UML. Existen elementos para construir modelos en UML: entidades, relaciones y diagramas. La arquitectura de un sistema esta formada por cinco vistas: casos de uso, diseño, procesos, implementación y despliegue.

Capítulo 4
Clases

Una clase describe un conjunto de objetos que comparten atributos, operaciones y relaciones. Un atributo es una característica de la clase; una operación es una tarea que puede realizar un objeto que pertenece a la clase. Las clases cumplen con una obligación que se conoce como responsabilidad.

Capítulo 5
Relaciones

Las relaciones son conexiones entre elementos. Las relaciones que forman parte del modelado orientado a objetos son de: dependencia, generalización y asociación. La dependencia representa cuando un elemento utiliza a otro; la generalización es cuando un elemento general tiene relación con un elemento específico; y las asociaciones conectan a objetos de un elemento con los objetos de otro elemento.

Capítulo 6
Mecanismos comunes

Existen una serie de especificaciones que forman parte del UML, como: estereotipos, valores etiquetados y restricciones. Un estereotipo permite agrupar elementos; un valor etiquetado consiste en añadir propiedades a un elemento; y una restricción es una regla que forma parte de un elemento.

Capítulo 7
Diagramas

Todo sistema esta formado por modelos que sirven para representar y comprender con claridad los subsistemas que forman al sistema. Una vista consiste en organizar y estructurar un sistema. El diagrama es una representación gráfica de una serie de elementos. Dentro de los diagramas estructurales tenemos a los de clases, objetos, componentes y despliegue; y como diagramas de comportamiento están los de casos de uso, secuencia, colaboración, estados y actividades.

Capítulo 8
Diagramas de clases

En el desarrollo de un sistema orientado a objetos los diagramas de clases son los más utilizados, estos están formado por un conjunto de clases, interfaces y relaciones. Las colaboraciones entre clases se representan a través de las relaciones. Con los diagramas de clases se puede realizar el diseño lógico de la base de datos, debido a que son un subconjunto de los diagramas entidad-relación; y por otro lado se puede generar código a partir de un modelo o viceversa.

Capítulo 9
Características avanzadas de las clases

Las clases son elementos de construcción que a su vez están formadas por una serie de propiedades o mecanismos, como los clasificadores. Los clasificadores describen características estructurales y de comportamiento. La visibilidad especifica el nivel de acceso que tienen los atributos y operaciones.

Capítulo 10
Características avanzadas de las relaciones

Existen tipos de relaciones avanzadas como: dependencia, generalización, asociación, realización y refinamiento. La relación de dependencia especifica un cambio en la especificación de un elemento. La generalización es una relación entre una superclase y una subclase. Una asociación es la conexión entre elementos. La realización es la especificación de un contrato entre clasificadores.

Capítulo 11
Interfaces, tipos y roles

Una interfaz es una serie de operaciones que son utilizadas por una clase o componente. Un tipo es un estereotipo de una clase. Un rol es el comportamiento de una entidad participante en un contexto en particular.

Capítulo 12
Paquetes

Un paquete es un mecanismo para agrupar clases. Un paquete puede tener visibilidad para controlar el acceso de sus clases. Existen dos formas en las que un paquete puede relacionarse con otro paquete, una de ellas es por medio de la generalización y la otra a través de la dependencia.

Capítulo 13
Instancias

Una instancia es un objeto de una clase en particular. Las instancias que se modelen en UML serán instancias de clases. Cada instancia tiene un nombre que la distingue de las demás instancias.

Capítulo 14
Diagramas de objetos

Un diagrama de objetos representa una colección de objetos y sus relaciones. Los objetos y enlaces forman a este tipo de diagrama; a diferencia del diagrama de clases el diagrama de objetos da la perspectiva de instancias reales.

Capítulo 15
Interacciones

Una interacción es el comportamiento de un objeto al intercambiar mensajes con otros objetos. Por otro lado un mensaje es la comunicación entre objetos, el cual se realiza especificando la información que se transmitirá. Los objetos participan en las interacciones. Otro tipo de interacciones son los enlaces, los cuales son conexiones entre objetos.

Capítulo 16
Casos de uso

Los casos de uso especifican el comportamiento del sistema, cada caso de uso esta formado por una serie de acciones que describen el flujo de los eventos. Los actores interactúan con los casos de uso, un actor puede ser una persona, un sistema o un dispositivo. Un caso de uso se puede relacionar con otro caso de uso o con un actor.

Capítulo 17
Diagramas de casos de uso

Los diagramas de casos de uso representan el comportamiento de un sistema y se emplean para modelar la vista de casos de uso. Están formados por casos de uso, actores

y relaciones. Este tipo de diagramas se utiliza para modelar tanto el contexto de un sistema, como los requisitos de un sistema.

Capítulo 18
Diagramas de interacción

Un diagrama de interacción muestra las relaciones entre objetos, los elementos que lo forman son: objetos, enlaces y mensajes. Los diagramas de secuencia y los diagramas de colaboración forman parte de los diagramas de interacción. Un diagrama de secuencia muestra las interacciones entre objetos ordenados a través del tiempo; y los diagramas de colaboración representan las interacciones entre objetos organizados.

Capítulo 19
Diagramas de actividades

Cada diagrama de actividad muestra el flujo de control entre actividades, es decir modela los pasos de un proceso. Una actividad es la ejecución de una acción. Un diagrama de actividad esta compuesto por estados, transiciones y objetos.

Capítulo 20
Eventos y señales

Un evento es un acontecimiento en el tiempo que puede permitir la transición de un estado a otro. Existen eventos externos y eventos internos; los externos ocurren entre el sistema y sus actores, los internos se realizan entre los objetos del sistema. Una señal es un evento entre objetos que representa un estímulo.

Capítulo 21
Máquinas de estado

Una máquina de estados consiste de una secuencia de estados que representan el comportamiento de un objeto. Los objetos pueden responder a ciertos eventos; tras ocurrir un evento se ejecutará una actividad. Una actividad es una operación que toma un momento en realizarse dentro del sistema. El estado de un objeto es una posible condición en la que un objetos puede existir. El cambio de un estado a otro estado se realiza a través de transiciones. Pueden darse la existencia de subestados dentro de un estado, a esto se le conoce como anidación de estados.

Nombre: ***Visual Modeling with Rational Rose 2000 and UML***
Autor: *Terry Quatrani*
Editorial: *Addison-Wesley*
Edición: *Primera*
ISBN: *0-201-69961-3*

Capítulo 1
Introducción

Los modelos visuales son útiles para solucionar problemas reales; a través de ellos se pueden representar ideas y soluciones. Una notación es un componente importante para el desarrollo de software y es utilizada durante el proceso de desarrollo de software. UML permite comunicar elementos de análisis y diseño de un sistema.

Capítulo 3
Creación de casos de uso

El comportamiento del sistema es documentado en modelos de casos de uso; los cuales, representan las funciones, el entorno y su relación. Un actor es cualquier cosa o persona que interactúa con el sistema. Un caso de uso es una función proporcionada por el sistema, y será documentado con su descripción, flujo principal, subflujos y flujos alternativos. Los diagramas de actividad representan el flujo para una operación del sistema.

Capítulo 4
Encontrando clases

Los objetos representan una entidad real o abstracta del sistema y debe de tener tres características: estado, comportamiento e identidad. Una clase es una descripción de un conjunto de objetos con propiedades y comportamiento en común; una clase puede tener un estereotipo; es decir, un tipo de clase. Así mismo, las clases pueden ser agrupadas en paquetes. Las clases y paquetes participantes en un caso de uso se representan en un diagrama de clases.

Capítulo 5
Descubriendo interacción de objetos

Una vez que los escenarios son documentados posteriormente se representan en diagramas de interacción, los cuales son de dos tipos: diagramas de secuencia y diagramas de colaboración. Ambos tipos de diagramas muestran la relación entre objetos, pero los de secuencia son de acuerdo al tiempo.

Capítulo 6
Especificación de relaciones

Para lograr la colaboración entre objetos es necesario hacerlo mediante relaciones, los tipos de relaciones son: asociaciones y agregaciones. Una asociación es la conexión bidireccional entre dos objetos y las agregaciones es un caso especial de asociaciones, una clase es el todo y otra clase la parte(s). Las relaciones pueden integrar otros elementos como: una frase que comunique su significado e indicadores de multiplicidad que definen el número de objetos que participan. De igual modo existen relaciones entre paquetes.

Capítulo 7
Agregando estructura y comportamiento

Cada clase tiene un conjunto de responsabilidades que definen el comportamiento de los objetos de esa clase, los atributos de una clase definen sus características. Los mensajes entre objetos en los diagramas de interacción, corresponden a las responsabilidades que desempeñarán los objetos que se creen de una clase.

Capítulo 8
Descubriendo herencia

La herencia es la relación entre clases donde una clase adquiere la estructura y comportamiento de una o más clases. Existen conceptos como generalización y especialización relacionados con la herencia. La generalización radica en crear superclases que tienen estructura y comportamiento de varias clases; la especialización consiste en crear subclases que especialicen a una superclase.

Capítulo 9
Analizando comportamiento de objetos

Los diagramas de estados muestran el comportamiento de un objeto; los cuales están formados por sus estados, mensajes y acciones. Un estado es una posible condición en la que se encuentra un objeto. Los cambios entre un estado y otro son conocidos como transiciones. Cada diagrama de estados tiene un estado inicial, pero puede tener varios estados finales.

Capítulo 10
Revisando el modelo

Al avanzar con el desarrollo de software, es necesario realizar algunos ajustes, para que los elementos que los componen sean homogéneos. Combinar o dividir las clases es común al revisar los diagramas; incluso eliminar algunas clases cuyo comportamiento y estructura son inexistentes.

Capítulo 11
Diseñando la arquitectura del sistema

La arquitectura del sistema consiste en que va a realizar el sistema. Existen cinco vistas de la arquitectura: de casos de uso, de diseño, de implementación, de procesos y de despliegue. La vista de casos de uso consiste en conocer el entono y entendimiento del sistema, se representa con los modelos de casos de uso; la vista de diseño representa la funcionalidad del sistema, esto es a través del diagrama de clases; la vista de implementación se basa en la organización modular del software dentro del ambiente de desarrollo y es a través del diagrama de componentes; la vista de procesos muestra la estructura del sistema al momento de ejecutarse; la vista de despliegue muestra la configuración del hardware del sistema, la representan los diagramas de desarrollo.

2.1.2 Libros de lectura ligera

Nombre: ***Análisis y Diseño Orientado a Objetos***
Autor: *Grady Booch*
Editorial: *Addison-Wesley*
Edición: *Segunda*
ISBN: *9-684-44352-8*

Capítulo 2
El modelo de objetos

Actualmente el análisis, diseño y programación se pueden realizar a través del paradigma orientado a objetos, para la solución de problemas de software. Algunos conceptos primordiales son la abstracción, la cual denota las características esenciales de un objeto que lo distinguen de los demás. El encapsulamiento es el proceso de ocultar la estructura y comportamiento de un objeto.

Capítulo 3
Clases y objetos

Un objeto tiene estado, comportamiento e identidad. La estructura y comportamiento de objetos similares están definidos en una clase en común. Los dos tipos de jerarquías entre objetos son la inclusión y las agregaciones.

Nombre: ***El Proceso Unificado de Desarrollo de Software***
Autor: *Ivar Jacobson, Grady Booch y James Rumbaugh*
Editorial: *Addison-Wesley*
Edición: *Primera*
ISBN: *0-201-57169-2*

Capítulo 2
Implementación

En la implementación se comienza con el resultado del diseño y codificar el sistema en términos de componentes, es decir, código fuente. Posteriormente se integran y se enlazan en uno o más ejecutables, por lo que es necesario realizar comprobaciones del sistema. En esta fase es necesario el modelo de diseño, para reflejar las clases en términos de componentes.

2.2 Tesis

Título: ***Documentación del análisis y diseño de sistemas orientados a objetos con UML***
Autor: *Jiménez Herrada, Jenny*
Fecha: *2001*
Carrera: *Informática*
Facultad: *Facultad de Contaduría y Administración, UNAM*
Ubicación: *001-00623-J1-2001*

Comentario:

Trata acerca de la importancia y utilidad de documentar sistemas orientados a objetos con notación UML durante las etapas de análisis y diseño. En esta tesis se definen conceptos orientados a objetos, todos ellos provenientes de autores o empresas dedicadas a la industria del software. Se presentan los métodos de desarrollo orientados a objetos, entre ellos los de Booch, Jacobson y Rumbaugh.

Título: ***El software orientado a objetos***
Autor: *Preciado López, José Guillermo*
Fecha: *1996*
Carrera: *Ciencias de la Computación*
Facultad: *Facultad de Ciencias, UNAM*
Ubicación: *001-00321-P4-1996*

Título: ***Una aplicación de la metodología OMT para diseño orientado a objetos***
Autor: *Oliver Suárez, Isabel Laura*
Fecha: *1998*
Carrera: *Ciencias de la Computación*
Facultad: *Facultad de Ciencias, UNAM*
Ubicación: *001-00321-O2-1998*

Título: ***RUP y UML como elementos clave en el desarrollo de sistemas orientados a objetos***
Autor:
Fecha: *2002*
Carrera: *Informática*
Facultad: *Facultad de Contaduría y Administración, UNAM*
Ubicación:

Comentario:

La tesis comienza tratando la historia de la orientación a objetos y los orígenes de las metodologías orientadas a objetos, enfatizandose en las etapas de análisis y diseño. Los métodos de Grady Booch, Ivar Jacobson y James Rumbaugh son de los que han aportado más dentro del desarrollo de software orientado a objetos. Se definen conceptos y principios de la orientación a objetos y de UML. Se explican otros métodos orientados a objetos, de los cuales se describe cada una de sus fases, entre ellos el RUP. Se explican los diagramas que forman parte de la notación UML.

2.3 Revistas

Nombre: ***JavaPro - Guide to Middleware***
Domicilio: *209 Hamilton Ave., Palo Alto, CA 94301*
Periodo: *Octubre*
Volumen: *3*
Número: *13*

Título: ***Why UML and Java?***
Autor: *Steve Downey*
Fecha: *15 de Octubre de 1999*

Comentario:

Se define al UML como una notación que representa al análisis y diseño orientado a objetos; el cual es independiente del lenguaje o lenguajes en el que se vaya a codificar el sistema. Es utilizado para comunicar las especificaciones del sistema a los programadores en Java.

Se definen el análisis, diseño y programación orientados a objetos, para conocer la diferencia entre cada uno de ellos. El análisis orientado a objetos dentro del desarrollo de un proyecto es de suma importancia, describe lo que el sistema va a hacer. Es representado a través de objetos, por lo que debe de ser lo más cercano a la realidad. En el diseño orientado a objetos se especifica como va a resolver el sistema el problema. Y la programación orientada a objetos consiste en la implementación en código del sistema.

Título: ***Java and UML***
Autor: *Jacques Surveyer*
Fecha: *15 de Octubre de 1999*

Comentario:

Habla sobre lo que es UML y como ha evolucionado a través del tiempo. Algunas metodologías han adoptado a UML para modelar el análisis y diseño orientado a objetos. En la actualidad existen herramientas CASE que soportan a UML y Java, las cuales han sido adoptadas por varios proveedores. También trata la contribución de UML dentro del desarrollo de software, al igual que los obstáculos existentes a los que se enfrenta.

Básicamente se dan a conocer los pros y contras de las herramientas CASE orientadas a objetos; se evalúa la generación de código Java a través de los modelos en UML, debido a que no todas lo soportan.

Nombre: ***JavaPro***
Domicilio: *209 Hamilton Ave., Palo Alto, CA 94301*
Periodo: *Junio/Julio*
Volumen: *2*
Número: *3*

Título: ***Design Java Apps with UML***
Autor: *Hans-Erik Erikkson y Magnus Penker*
Fecha: *1 de Junio de 1998*

Comentario:

Trata sobre el desarrollo de un sistema orientado a objetos; dicho proyecto consiste en el préstamo y reservación de libros; el cual, se elabora a través de las etapas del ciclo de vida del software.

Se trabajan las fases de análisis, diseño, implementación, pruebas y liberación. En el análisis se describen los requerimientos del sistema y los casos de uso, se definen los conceptos del sistema utilizados por los usuarios y expertos. En el diseño se realiza la arquitectura del sistema, se definen las interfaces de usuario, los objetos y la base de datos. La implementación comienza con la programación del sistema, por medio de los diagramas. Las pruebas se hacen durante y al terminar la codificación. Todos los modelos y diagramas generados sirven como parte de la documentación al liberar el sistema.

Nombre: ***EETimes (Electronic Engineering Times)***
Domicilio: *600 Community Drive Manhasset, NY 11030*

Artículo: ***Using UML to drive Java can alleviate chaos***
Autor: *Bill Graham*
Fecha: *1 de Abril de 2002*

Comentario:

En este artículo se describe la importancia que ha adquirido el UML como lenguaje de modelado para aplicaciones que serán codificadas en Java. Se menciona la utilidad de los diagramas de clases para representar la estructura del sistema. La generación de código Java a partir de modelos en UML ha sido adoptada por las empresas que se dedican al desarrollo de software. Para que sean de utilidad los modelos en UML al implementarlos tienen que representar el problema en forma correcta, por que de otra forma la aplicación será deficiente.

2.4 Artículos

Título: ***Mapping UML desings to Java***
Autor: *William Harrison, Charles Barton, Mukund Raghavachari*
Compañía: *IBM*
Fecha:

Comentario:

Se describe en el artículo un método para generar código Java a través de diagramas de UML, basándose en los diagramas de clases y de los elementos que los forman, tales como: clases y relaciones. Uno de los objetivos principales al realizar esta técnica es el de minimizar esfuerzos para los diseñadores y programadores.

Título: ***A Quick Introduction to UML and Java***
Autor: *Rob McGovern*
Fecha: *21 de Febrero de 2002*

Comentario:

El artículo habla de la claridad de los modelos en UML al realizar el desarrollo de un sistema. Esto es una ventaja para los diseñadores y programadores, debido a que UML es un lenguaje de modelado que puede ser interpretado en Java. Elementos como clases y relaciones, que son parte de los diagramas de clases son representados en código en Java.

Título: ***An overwiew of object relationships / The basics of UML and Java associations***
Autor: *Scott W. Ambler*
Compañía: *IBM - DeveloperWorks*
Fecha: *8 de Febrero de 2001*

Comentario:

Este artículo proporciona una breve introducción a los tipos de relaciones en UML. Dentro de los tipos de relaciones tenemos a las asociaciones, agregaciones y composiciones, las cuales son la más utilizadas en los diagramas de clases. Se presenta como se modelan y cuales son los elementos que las conforman.

Título: ***Implementing object relationship with a singular multiplicity***
Autor: *Scott W. Ambler*
Compañía: *IBM - DeveloperWorks*
Fecha: *2 de Marzo de 2001*

Comentario:

Este artículo trata sobre la implementación de relaciones uno a uno y uno a muchos, las cuales son consideradas como relaciones con multiplicidad simple. Se representa este tipo de relación con modelos en UML y se ejemplifica como se codificaría en lenguaje Java. Forma parte de una serie de artículos que hablan acerca de las relaciones entre clases en UML.

Título: ***The fundamentals of one-to-many bidireccional associations***
Autor: *Scott W. Ambler*
Compañía: *IBM - DeveloperWorks*
Fecha: *8 de Marzo de 2001*

Comentario:

Es un breve artículo, en el que se tratan temas acerca de las relaciones bidireccionales en UML, como los elementos que la conforman y su implementación en Java, Se da a conocer una serie de practicas que pueden ser útiles al pasar este tipo de relación a código.

Da a conocer la utilidad de las clases contenedoras, para almacenar objetos que pueden ser guardados durante la relación.

Título: ***Using collections to manage many-to-many object relationship***
Autor: *Scott W. Ambler*
Compañía: *IBM - DeveloperWorks*
Fecha: *15 de Marzo de 2001*

Comentario:

Habla acerca de que en Java, una relación muchos a muchos es implementada utilizando clases contenedoras y operaciones que manipulen las colecciones de los objetos. Las relaciones de muchos a muchos son difíciles de implementar, debido a que un objeto de la clase contenedora debe de mantener referencias de otros objetos.

2.5 Internet

Título: ***Mapping between UML and Java***
Autor: *Jan Stelovsky*
Compañía: *University of Hawaii*
URL: *http://ami.ics.hawaii.edu/courses/665/notes/mapping.html*
http://ami.ics.hawaii.edu/courses/665/notes/uml_class.html
http://ami.ics.hawaii.edu/courses/665/notes/subclassing_composition.html

Comentario:

Está página está basada en un curso impartido por el profesor Jan Stelovsky de la Universidad de Hawai en 1998, el cual habla de cómo pasar de diagramas de clases en UML a Java. Explica de forma clara y concreta la representación de los elementos que componen al diagrama de clases en el lenguaje de programación orientado a objetos Java. Utilizan como herramienta CASE para desarrollar los diagramas a Rational Rose.

Título: ***Modeling Java Applications using UML***
Autor: *Kirk Knoernschild*
Compañía:
URL: *http://www.kirkk.com/ReferenceCard.html*
Versión: *(2000)*

Comentario:

Trata la página sobre el modelado en UML para aplicaciones que serán codificadas en Java. Se definen los diagramas de UML con los elementos que los componen, su equivalente en lenguaje Java y ejemplos de diagramas y código. Básicamente es un manual de referencia para conocer aspectos esenciales de diseño e implementación.

Título: ***Modeling Java Applications using UML***
Autor: *Allen I. Holub*
Compañía: *Allen I. Holub & Associates*
URL: *http://www.holub.com/class/uml/uml.html*
Versión: *Ver. 1.07 (2/4/2002)*

Comentario:

Ofrece una panorámica de los diseños orientados a objetos y Java. El sitio describe los modelos estáticos como el diagrama de clases, los modelos dinámicos tal como el diagrama de secuencia, y el diagrama de estados; definiendo sus características y elementos que los conforman, así como representarlos en código Java. Contiene ejemplos prácticos de los tres tipos de diagramas antes mencionados.

Título: ***Ingeniería de Software Orientada a Objetos***
Autor: *Dr. Alfredo Weitzenfeld*
Compañía: *ITAM*
URL: *http://cannes.rhon.itam.mx/Alfredo/Espaniol/Publicaciones/IngSW.htm*
Versión: *(Enero 2001)*

Comentario:

Da a conocer la publicación de su libro referente a ingeniería de software orientada a objetos. Contiene la explicación de cada una de las etapas del ciclo de vida de desarrollo, aspectos importantes de UML y Java. Se ejemplifica a través de un ejemplo a lo largo de cada una de las etapas.

Marco Conceptual

3 MARCO CONCEPTUAL

3.1 Orientación a objetos

La orientación a objetos es una método para resolver problemas, es utilizado para el desarrollo de sistemas. Surge por la necesidad de crear software más complejo y como complemento para reflejar la topología de lenguajes de alto nivel como SmallTalk, Ada, Object Pascal, C++, Java y Eiffel.

Es un paradigma basado en objetos, el cual es una entidad o concepto del mundo real que forma parte de un modelo de software. Esto en la actualidad representa una estructura metodológica para la ingeniería de software y proporciona las bases de una disciplina de ingeniería que se realiza sistemáticamente.

3.1.1 Historia

La orientación a objetos surge en 1967 con un lenguaje de programación llamado Simula67, el cual fue desarrollado en Noruega por Krinsten Nygaard y Ole-Johan Dahl; introduciendo así los primeros conceptos de la orientación a objetos que se conocen en la actualidad, tales como clases y subclases.

En los años 70's los científicos del centro de investigación de Palo Alto Xerox, desarrollan a SmallTalk, siendo el primer lenguaje orientado a objetos puro, el cual utiliza conceptos como clases y objetos.

Pero aún existía un problema dentro de la programación, el cual consistía en que las variables eran visibles desde cualquier parte del código, así que D. Parnas propuso ocultar la información. Esta idea consistía básicamente en encapsular las variables de un objeto y que se accese a ellas a través de sus operaciones asociadas.

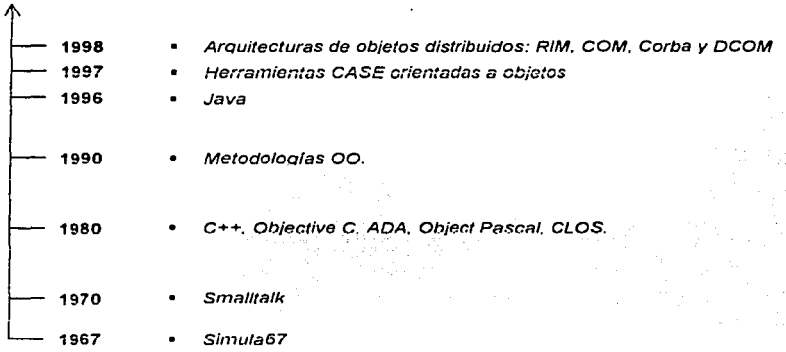
En los años 80´s los Laboratorios de AT&T a través de Bjarne Stroustrup, realizan una extensión de C con un enfoque orientado a objetos conocido como C++. Varios fabricantes lo dan a conocer y existe una mayor atención hacia la orientación a objetos por la comunidad dedicada al desarrollo de software. Por tal motivo surgen otros lenguajes orientados a objetos como: CLOS, Object Pascal, Objective C y Ada.

A inicios de los 90´s la orientación a objetos se consolida como una forma para solucionar problemas de ingeniería de software y comienzan a surgir las primeras metodologías orientadas a objetos.

En 1996 surge Java, como un lenguaje orientado a objetos con la filosofía de aprovechar el software existente, adaptarlo a otros problemas sin modificar el código existente.

En 1997 y 1998 surgen herramientas CASE orientadas a objetos y se desarrollan arquitecturas de objetos distribuidos.

La orientación a objetos es la culminación de años de experiencia en encontrar un camino de desarrollo de software, y actualmente es un método utilizado para el desarrollo de sistemas de software.



Evolución de la orientación a objetos.

3.1.2 Principios y conceptos de la orientación a objetos

La tecnología orientada a objetos requiere dar a conocer algunos conceptos que son elementales para su entendimiento y de los cuales se va a recurrir, ya que son la base para el desarrollo de software orientado a objetos. Los elementos primordiales que conforman este enfoque son los siguientes:

Objeto {
- *estado*
- *comportamiento*
- *identidad*
- *instancia*
- *ciclo de vida de un objeto*

Encapsulación

Método

Mensaje

Clase { *instancia de clase*

Herencia

Polimorfismo

3.1.2.1 Objeto

Un *objeto* es un elemento o cosa independiente que forma parte de nuestro mundo, que tiene un conjunto de características propias y realiza una serie de tareas específicas. Un objeto puede ser real o abstracto.

Objeto empleado

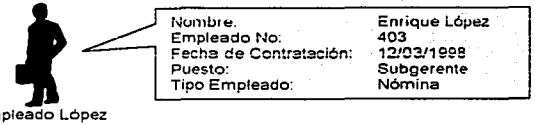
En objeto es el empleado de una empresa

Otros ejemplos de objetos son los siguientes:

- una factura
- un software
- una organización
- un empleado
- una automóvil
- una impresora

Un objeto tiene su propio *estado, comportamiento e identidad.*

- El estado de un objeto es una posible condición en la un objeto puede existir, dicho estado puede cambiar a través del tiempo y es definido por un conjunto de propiedades o características conocidas como *atributos*, los cuales tienen un valor.

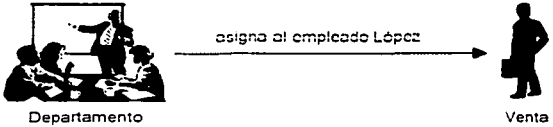


El Empleado López tiene definidas sus características tales como su nombre, no de empleado, fecha de contratación, puesto y tipo de empleado, lo cual establece que pertenece al conjunto de empleados. En el caso en que se relacione con otros objetos, tales como "Departamento", "Cargo"; el objeto puede asumir varios estados tales como "ascendido", "despedido", "con aumento", "por honorarios", etc.

También el estado depende de la relación que el objeto pueda tener con otros objetos.

Tales estados pueden estar asociados a un ciclo de vida del objeto en particular; por ejemplo el objeto Empleado primero estuvo en el objeto "bolsa de trabajo", posteriormente paso al objeto "empresa de contratación" y así sucesivamente hasta que pase al objeto "pensión".

- El comportamiento de un objeto determina como un objeto actúa y cual es su reacción hacia otros objetos.



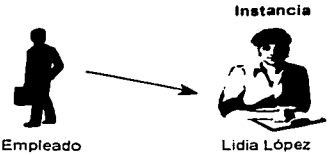
El objeto Departamento, comunica al objeto Empleado que realice una Venta a una empresa, por lo que el empleado López debe de aceptarla

- La identidad de un objeto es única, se podrá darse el caso contrario, sólo si el estado de un objeto es igual al de otro objeto, pero aún así son distintos entre sí.

El conjunto de empleado tiene dos trabajadores E. López, los cuales son del mismo tipo, son similares, pero hay características diferentes entre sí que los hacen distintos

El enfoque de orientación a objetos se basa en que un objeto puede estar formado por otros objetos, así como esos objetos a su vez pueden estar formados por otros objetos. Esta forma de composición permite definir objetos muy complejos.

En la orientación a objetos el término de instancia se usa para expresar una subordinación en cuanto a los elementos usados; en este caso, los objetos. Dicho de otra forma, una instancia identifica y define a un objeto (en este caso empleado) como un objeto en particular, (en este ejemplo, el empleado Lidia López). Podemos afirmar entonces que "el empleado Enrique López" es una instancia de "empleado".



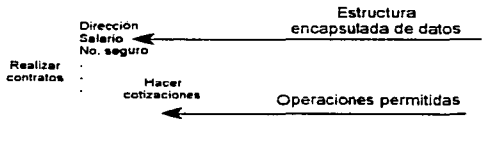
Por ejemplo, un objeto puede ser Empleado, pero una instancia de empleado sería "el empleado Lidia López".

El ciclo de vida de un objeto, es el tiempo de existencia de un objeto y consta de:

1. Cuando se crea el objeto.
2. Un objeto envía mensajes a otro, ya sea de objeto a objeto o de objeto a usuario.
3. Eliminación del objeto.

3.1.2.2 Encapsulación

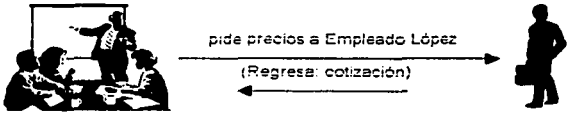
Al empaque conjunto de datos y métodos, se le llama encapsulado. La encapsulación se logra a través del *ocultamiento de la información*, que consiste en esconder sus datos de los demás objetos y permite el acceso a ellos mediante sus propios métodos y operaciones; esto con el fin de que no puedan ver su contenido y estructura.



El objeto Empleado tiene ocultos sus datos personales, pero se puede comunicar con otros objetos a través de las operaciones que realiza, tales como: Realizar Contratos, hacer cotizaciones, etc.

3.1.2.3 Método

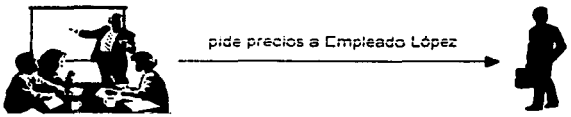
Durante el análisis y desarrollo orientado a objetos nos interesa el comportamiento que tenga un objeto, dicho comportamiento recibe el nombre de método el cual va a depender del valor que presenten sus datos y de las operaciones que se realicen sobre esos datos. Los métodos especifican la forma en que se controlan los datos de un objeto y sólo hacen referencia a los datos de ese objeto, por lo que no deben de tener acceso a los datos de otro objeto. Para utilizar los datos de otro objeto, le debe de enviar un mensaje a ese objeto, así se va a ejecutar un método.



Se ilustra en el ejemplo que el empleado de ventas va a responder a la petición de precios, mediante el envió de una cotización (acción), la cual presenta los precios del equipo de cómputo (respuesta)

3.1.2.4 Mensaje

Para que un objeto lleve a cabo una acción, se le envía una solicitud. Esta hace que realice una operación, la cual ejecuta el método apropiado y puede regresar una respuesta. Dicho mensaje contiene el nombre del objeto, el nombre de una operación y, a veces el grupo de parámetros.



Como se muestra en el ejemplo, el Depto de Compras de una empresa se puede comunicar con un agente de ventas al enviarle una solicitud de precios de equipo de cómputo por medio de un teléfono o fax. Otros métodos podrían ser que el agente de ventas envíe factura, demostración de un producto, realizar contrato, etc.

Un objeto enviará una petición, para que se realice una acción, y para conocer si ejecutará o no se va a emitir una respuesta.

3.1.2.5 Clase

Una *clase* es la descripción de un grupo de objetos con propiedades, comportamiento, relaciones con otros objetos y semántica en común. Dicha clase es una abstracción, en donde se enfatizan sus características relevantes. Por tal motivo, existen muchos objetos identificados dentro de una clase y cada uno de ellos es una instancia de clase.

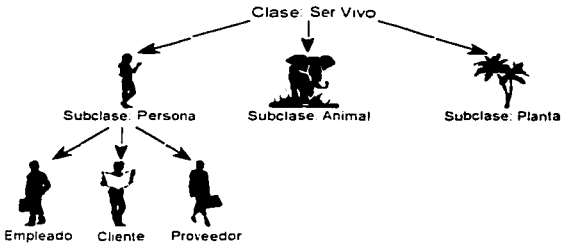
La relación entre una clase y un objeto consiste en que una clase es una definición abstracta de un objeto, del cual se va a definir la estructura y comportamiento de cada objeto en la clase, esto sirve como una plantilla para crear objetos. Por tal motivo muchos objetos pueden ser agrupados dentro de una clase.

De la Clase "Persona" se van a crear los objetos: "Empleado, "Cliente" y "Proveedor", los cuales van a contar con características y comportamientos en común, tales como, nombre, dirección, trabaja, paga impuestos, etc.

Al definir una clase se crea una serie de características reutilizables en lugar de que sus atributos se repitan una y otra vez.

3.1.2.6 Herencia

La *herencia* es cuando una clase comparte la estructura y comportamiento definidos a una o a más clases.

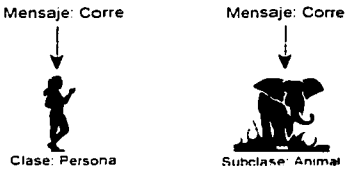


La clase que recibe la herencia se le llama *subclase* o *clase derivada*, y la clase de la cual se hereda se le llama *superclase* o *clase base*.

La herencia sirve para definir una jerarquía de tipo es-un entre clases en donde una clase hereda de una o más superclases generalizadas. Típicamente, una clase subclase especializa su(s) superclase(s) incrementando o sobreponiendo el comportamiento y estructura existente.

3.1.2.7 Polimorfismo

El *polimorfismo* es cuando objetos de clases distintas, pero con una superclase en común, pueden responder de manera diferente a un mismo mensaje.



3.1.2.8 Abstracción

La abstracción es un mecanismo para representar un concepto u objeto en la realidad de manera simplificada. La abstracción es una propiedad de la orientación a objetos y sirve para modelar el sistema en forma real.

3.1.3 Sistema de software orientado a objetos

Un sistema de software orientado a objetos consiste de un conjunto de objetos de diferentes clases que interactúan entre sí utilizando métodos o servicios, todo ello con el propósito de que opere un producto de software.

Lo que hace diferente a un sistema orientado a objetos y a otro sistema de software, radica en que el orientado a objetos utiliza una serie de conceptos y propiedades que caracterizan al sistema orientado a objetos como: abstracción, encapsulación, comunicación por medio de mensajes, vida de un objeto, jerarquía de clases y polimorfismo; esto es posible utilizando lenguajes orientados a objetos para implementar

sistemas que utilizan clases y objetos. Por otro lado los sistemas no orientados a objetos no serán construidos utilizando objetos o clases.

3.1.3.1 Características y ventajas de la orientación a objetos

La técnica de orientación a objetos tiene una serie de características importantes que la distinguen de las otras y que cambiaron nuestra forma de pensar sobre los sistemas. Estos aspectos son los siguientes:

1. La forma de pensar y de ver al mundo con OO es más natural que otras técnicas de análisis y diseño. Debido a que todo lo que nos rodea esta formado por objetos. Y desde la infancia crecimos y aprendimos con ellos y descubrimos que tienen un determinado comportamiento. De igual forma, las empresas piensan de manera natural con términos de objetos, eventos y comportamientos.

2. Como los sistemas se desarrollan de acuerdo a objetos ya existentes dentro de la organización; esto permite una reutilización de ellos generando: un menor costo, menor tiempo de desarrollo y mayor confiabilidad del sistema.

3. La complejidad de los objetos cada vez es mayor. Al construir un objeto debemos tomar elementos de otros objetos.

4. Se cuenta con una serie de herramientas para las etapas de análisis, diseño e implementación OO. Las herramientas CASE son una opción para construir en menor tiempo un sistema y nos reduce el tiempo de desarrollo.

5. Las clases están diseñadas para reutilizarse, ya que están contenidas de otras clases con sus respectivos métodos. Si se crea una nueva clase se puede modificar con gran facilidad. Eso puede producir un mejor funcionamiento del sistema.

Los beneficios que nos proporciona el utilizar la técnica de OO al desarrollar un sistema son:

Ventajas
- Reutilización
- Estabilidad
- Confiabilidad
- Rapidez
- Calidad
- Mantenimiento
- Realista
- Comunicación

Reutilización Las clases están diseñadas para volverse a utilizar en otros sistemas. En el caso de que un sistema tenga la necesidad de crecer se logrará a través de la reutilización de clases y de código. Esta ventaja se debe de considerar al construir un software.

Estabilidad Son estables las clases aún cuando se han utilizado para reutilizarlas, es decir no sufren ninguna alteración.

Confiabilidad El software que fue desarrollado con clases estables es probable que tenga menos fallas que el software elaborado a partir de cero.

Rapidez El diseño es más rápido debido a que los componentes ya existen.

Calidad Los diseños como se integran a través de componentes ya probados cuentan con mayor calidad.

Mantenimiento Es más sencillo proporcionar el mantenimiento a un sistema. Debido a que las funciones de una clase son independientes de las demás clases.

Realista El análisis orientado a objetos es la más parecido a la empresa; así se pasa al diseño y a la implementación de manera directa, debido a que utilizan el mismo paradigma.

Comunicación Los profesionales en sistemas de información y los usuarios piensan en términos de objetos, eventos y políticas empresariales que definen el comportamiento de los objetos. Así que los usuarios comprenden el paradigma OO para compartir un modelo común con los creadores del software.

3.1.3.2 Metodología orientada a objetos

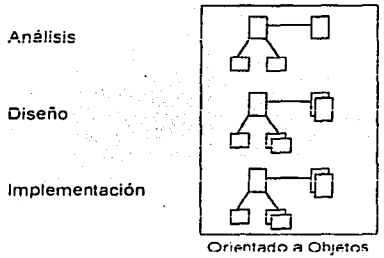
La metodología orientada a objetos se basa en modelar los sistemas como colecciones de objetos, es decir utiliza la misma técnica para cada etapa del proceso de desarrollo de software. De tal forma no se requiere ninguna forma de conversión para pasar de una etapa a otra. Por ejemplo, al final los programas desarrollados mostrarán similitud con los diagramas hechos en la fase del análisis y diseño.

Hoy en día existen infinidad de metodologías de desarrollo de software orientadas a objetos. Las metodologías en sus etapa de análisis y diseño requieren de una notación que muestre tanto el problema como su solución; así el programador se basa de esos diseños, que buscan ser simples y fáciles de entender.

Anteriormente cada metodología tenía su propia notación, esto era un problema, ya que un mismo símbolo podía representar diferentes cosas entre metodologías, en la actualidad ya no sucede en su mayoría, ya que algunas metodologías han adoptado a el UML como su notación, más aún al ser ya un estándar.

Entonces, la orientación a objetos desarrolla sus fases como sigue:

1. Empieza con un análisis *orientado a objetos*
2. Convierte el resultado del análisis en un diseño *orientado a objetos*
3. Convierte el diseño en programas *orientado a objetos*



Como se puede ver, la tecnología OO es usada en todo el proceso de desarrollo de software

3.1.3.3 Análisis y diseño orientado a objetos (ADOO)

El análisis y el diseño orientado a objetos consiste en situar el estado actual de un problema y su solución lógica dentro de la perspectiva de los objetos, con la finalidad de tener el dominio sobre ese problema.

El análisis orientado a objetos (AOO) es una de las fases iniciales dentro del ciclo de vida de software. En esta etapa se ve el mundo como objetos, los cuales deben de identificar y describir el problema que se presente. Dichos objetos cuentan con estructuras de datos y comportamientos que surgen a través de los requerimientos que presenten cada uno de los objetos proporcionados por un *modelo de objetos;* el cual consiste en definir los objetos que forman parte del sistema y el *modelo de comportamiento o estados* que describe el comportamiento de los objetos en relación con los eventos que existen dentro de un objeto.

Por ejemplo en el caso del sistema de contratación en una empresa, algunos objetos son: Empleado, Departamento, Sucursal.

Dentro de las técnicas de análisis orientado a objetos existen diferentes trabajos, algunos de ellos como Grady Booch define esta fase como:

> *"Un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema."*

Actualmente existe gran cantidad de tecnología que puede ser utilizada para desarrollar los requerimientos de ingeniería de software y las especificaciones. De igual modo los modelos pueden ser creados y mantenidos usando herramientas que representen los objetos y su comportamiento.

El diseño orientado a objetos (DOO) es importante en el desarrollo de un modelo orientado a objetos de un sistema de software para implementar los requerimientos especificados; esta fase se construye sobre los productos desarrollados durante el análisis orientado a objetos; por tal motivo, se definen los objetos lógicos del software que finalmente serán implementados en un lenguaje de programación orientado a objetos tal como: objetos candidatos dentro de las clases, definir mensajes para todos los objetos, definir estructuras de datos y procedimientos.

Muchas propuestas sobre DOO han sido descritos desde finales de los 80's. Las más populares metodologías incluyen a Booch, Buhr, Wasserman, entre otros.

Booch sintetiza el concepto de DOO como:

> *"El diseño orientado a objetos es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña."*

Tal como en el AOO, los artefactos o elementos del DOO son representados utilizando herramientas con terminología orientada a objetos.

3.1.3.4 Programación Orientada a Objetos (POO)

La programación orientada a o consiste en implementar en un lenguaje orientado a objetos los diseños que resultaron durante el desarrollo de software. En la actualidad hay una serie de lenguajes orientados a objetos, como: Smalltalk, Eiffel, C++, Ada, Objective C, Object Pascal, CLOS y Java.

3.1.3.4.1 Lenguaje Java

3.1.3.4.1.1 Introducción

Java es un lenguaje de programación para diseñar aplicaciones y programas. Hace uso del paradigma de orientación a objetos; cuya utilidad es creciente en el mundo del diseño del software.

3.1.3.5 Características

Las características del lenguaje son:

Orientado a objetos. Java es un lenguaje de programación orientado a objetos. Soporta objetos y sobre todos tiene las características y propiedades de la orientación a objetos.

Simple. Es un lenguaje que se diseñó para facilitar su aprendizaje. Se añaden y se eliminan características que otros lenguajes orientados a objetos no tienen.

Distribuido. Java está diseñado para el desarrollo de aplicaciones autónomas que residan en diferentes computadoras en una red. Proporciona librerías y herramientas para que los programas puedan ser distribuidos.

Independiente de la plataforma. Java está diseñado para que corra en plataformas y sistemas operativos distintos.

Robusto. Java realiza la búsqueda de errores, en tiempo de compilación y ejecución. La comprobación de tipos y la declaración explícita de métodos permiten detectar errores.

Seguro. Java mantiene la seguridad al prevenir el acceso a la memoria y protege las aplicaciones en internet.

3.1.3.6 Historia y evolución

En 1990 surge una investigación conocida como "Green Project" por parte de Sun, encabezada por Patrick Naughton junto con sus colegas James Grosling y Mike Sheridan. El objetivo principal fue crear un lenguaje de programación basado en C++, destinado al campo de la electrónica de consumo, especialmente electrodomésticos.

En 1991 surgió un compilador al que se le denominó Oak. Todo ello gracias a las aportaciones de los miembros del equipo. En 1993 aparece Mosaic y la World Wide Web comienza su transición al modo gráfico. Patrick Naughton, pone en libre distribución por la red el lenguaje Oak, más tarde desarrollan un decodificador, el cual se convertiría en Java. Se crea la primera aplicación para un ordenador personal, un browser para HTML, que demostraría toda la potencia de este nuevo lenguaje. En el se podría visualizar el primer applet Java que se escribió.

James Goslin estableció el nombre definitivo para el lenguaje Java, el 23 de Mayo de 1995 Sun Microsystems anuncia formalmente la aparición del nuevo lenguaje de programación Java y de HotJava, un navegador para éste, desarrollado con el mismo lenguaje.

3.2 Lenguaje de Modelado Unificado (UML)

El UML (Unified Modeling Language) es un lenguaje estándar para especificar, construir, visualizar y documentar los artefactos de un sistema de software orientado a objetos. Al utilizar UML, los desarrolladores y programadores pueden hacer un esquema de un proyecto, de hecho facilita el proceso de desarrollo de software.

UML es un lenguaje estándar con el que es posible modelar todos los componentes del proceso de desarrollo, y no define un modelo de desarrollo de software, incluso se puede utilizar para que una empresa pueda definir sus propios procesos. Y como lenguaje de modelado supone una abstracción de un sistema para llegar a construirlo en términos concretos. UML recomienda usar los procesos que utilizan las metodologías de desarrollo.

El objetivo central del lenguaje es abstraer cualquier tipo de sistema, sea informático o no, mediante diagramas. Un diagrama es una representación gráfica de una colección de elementos del modelo.

3.2.1 Evolución del UML

En los 90's surgen diferentes metodologías de desarrollo de software orientado a objetos, cada una con su propia notación; tres de ellas fueron las principales: la OMT desarrollada por Rumbaugh, la de Booch y la OOSE de Jacobson. Cada una tenía su propio valor y énfasis.

Posteriormente Booch escribió su segundo libro, tomando ideas de análisis de Rumbaugh y Jacobson. Mientras tanto Rumbaugh escribía una serie de artículos que en conjunto se convirtieron en la OMT-2, con técnicas de diseño utilizadas por Booch. Ambas metodologías comenzaron a converger, pero aún cada una con su propia notación.

Por la diferencia de notaciones se optó por adoptar una que contemplara todos los aspectos de las metodologías existentes, así fue que surgió el UML. Este lenguaje se desarrolló con la unificación de una serie de notaciones e ideas con enfoque orientado a objetos. UML fue introducido a la comunidad en 1995 con la versión 0.8; pero Ivar Jacobson tuvo aportaciones con las versiones 0.9 y 0.91 en 1996. Así la versión 1.0 fue presentada ante la OMG (Object Management Group) para su estandarización en 1997; meses después con algunas mejoras apareció la versión 1.1, la cual fue adoptada como un estándar en ese mismo año por la OMG.

Hubo aportaciones por parte de otras metodologías como:

Metodología	**Aportaciones**
Odell:	Clasificación
Shlaer and Mellor	Ciclo de vida de los objetos
Gamma	Patrones, notas y frameworks
Embly	Clases singleton
Fusion	Descripción de operaciones, mensajes enumerados
Wirfs and Brock	Responsabilidades
Harel	Cuadros de estados
Meyer	Pre-condiciones y pos-condiciones

3.2.2 Artefactos para el desarrollo de software

Dentro del contexto del UML un artefacto es una información que es utilizada o producida mediante un proceso de desarrollo de software. Un artefacto puede ser un modelo, una descripción o un software. Los artefactos en UML se representan en forma de diagramas, que junto con la documentación sobre el sistema constituyen los artefactos principales del sistema.

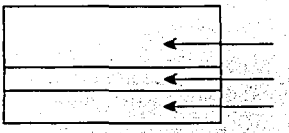
Los diagramas serán utilizados para crear diferentes vistas del mismo sistema. Crear diferentes vistas de un sistema satisface a los desarrolladores de software durante el proyecto. UML utiliza los diagramas para representar un sistema.

3.2.2.1 Elementos

Los elementos fundamentales del UML, son bloques de construcción que forman parte de los diagramas, son conocidos como elementos estructurales. Estos elementos contribuyen a la especificación de un sistema de software y representan abstracciones del sistema. Dentro de los elementos estructurales, existen los que encapsulan en conjunto el comportamiento del sistema y por otro lado los que definen como se relacionan los elementos. Conocer el uso de cada elemento facilita la creación de los diagramas.

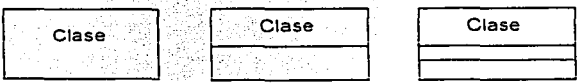
3.2.2.1.1 Clase

Una clase se representa en notación UML a través de un rectángulo con tres compartimentos horizontales.



El primero representa el nombre de la clase, el segundo muestra una lista de atributos de la clase y el tercer compartimento representa una lista de operaciones de la clase.

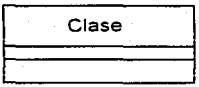
Existen variaciones de la notación en el caso de las clases, depende del nivel de detalle que se quiera representar; así que el segundo compartimento, el tercer compartimento o ambos pueden ser omitidos.



La clase llamada "Clase" puede ser representada gráficamente de tres formas. Los compartimentos son opcionales.

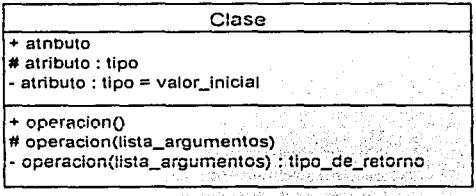
Las clases son bloques de código de programas en Java. Al implementar una clase de UML en Java, es necesario asegurarse que no tenga el estereotipo <<interface>>. La declaración de una clase consiste en un nombre de clase, una secuencia de variables y métodos, y las declaraciones de clases internas. La cláusula "*class*" permite declarar a una clase. La sintaxis de una clase es la siguiente:

[*Modificadores de Clase*] **class** *NombreDeLaClase* {
DeclaracionesDeMiembrosDeLaClase;
}



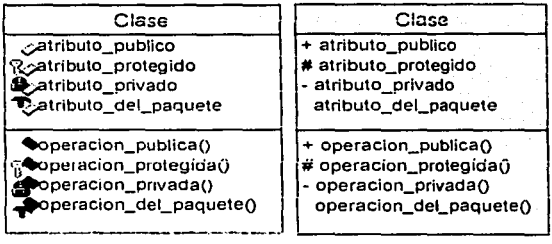

```
public class Clase {

}
```

Una clase en UML con atributos y operaciones quedaría de la siguiente manera:



Visibilidad y propiedades

La notación para representar la visibilidad o accesibilidad de los atributos y operaciones puede ser cualquiera de los siguientes: *public*, *protected*, *private* o *package* . El tipo de visibilidad se antepone mediante un icono o un símbolo:

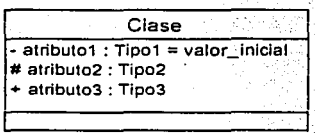


Símbolo	Visibilidad
o +	*public*
o #	*protected*
o -	*private*
	package

Atributos

Los atributos de una clase en UML corresponden a variables de instancia en una clase en Java. Y son descritos con la siguiente sintaxis en Java:

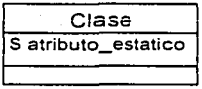
[*Modificadores*] *Tipo nombreDeLaVariable* [= *Valor Inicial*] ;




```
public class Clase {

  private tipo1 atributo1 = valor_inicial;
  protected tipo2 atributo2;
  public tipo3 atributo3;

}
```

Atributos estáticos

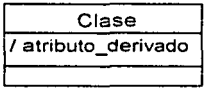
Si un atributo de una clase se le antepone el símbolo $ indica que es un atributo estático. Un atributo estático corresponde a una variable de clase en Java. Existe solamente una variable con ese nombre para todas las instancias de la clase. En Java, una variable de clase es indicada por el modificador "*static*".




```
public class Clase {

   static atributoEstatico;

}
```

Atributos derivados

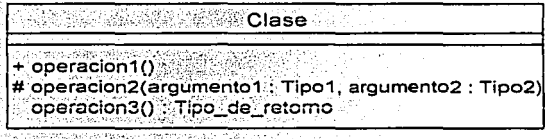
Un atributo derivado puede ser construido en base a otros atributos. Si un atributo tiene el símbolo / es derivado. Permite conocer ciertas condiciones de integridad. Java no soporta atributos derivados. Así que un atributo derivado se mapea como una variable de instancia.



Operaciones

Las operaciones de una clase en UML corresponden a métodos de una clase en Java. La sintaxis en Java es la siguiente:

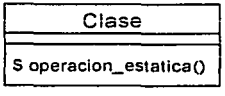
```
[ Modificadores ] TipoDeRetorno nombreDelMétodo ( [ Lista de Parámetros ] ) {
        cuerpoDelMétodo;
}
```




```
public class Clase {

  public void operacion1() {

  }
  protected void operacion2(tipo1 argumento1, tipo2 argumento2) {

  }
  package Tipo_de_retorno operacion3() {

  }

}
```

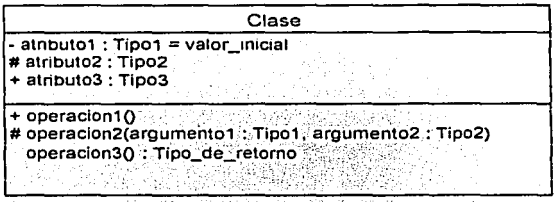
Operaciones estáticas

En UML una operación estática se indica con el prefijo $. Una operación estática se mapea a un método de clase en Java.




```
public class Clase {

  static void operacion_estatica() {;

  }

}
```

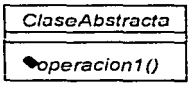
Una clase con todos sus atributos y operaciones en código Java queda de la siguiente manera:



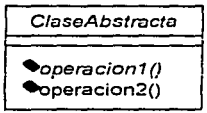

```
public class Clase {

  private Tipo1 atributo1 = valor_inicial;
  protected Tipo2 atributo2;
  public Tipo3 atributo3;

  public void operacion1() {

  }
  protected void operacion2(tipo1 argumento1, tipo2 argumento2) {

  }
  package Tipo_de_retorno operacion3() {

  }

}
```

Clase Abstracta

Una clase abstracta se representa igual que cualquier otra clase, sólo que su nombre se escribe con letras cursivas. Sus métodos se deben de especificar dentro del tercer compartimento y serán abstractos sólo si se especifican como tal. Los métodos abstractos también se representan con letra cursiva.



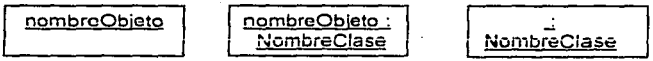
Para que una clase sea abstracta es necesario declarar el modificador de clase "*abstract*".




```
public abstract class ClaseAbstracta {

  public abstract void operacion1();

  public void operacion2() {

  }

}
```

3.2.2.1.2 Objeto

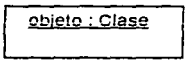
Se representa un objeto por medio de un rectángulo, pero para saber de que objeto se trata el contenido del rectángulo puede ser de tres formas:



La existencia de un objeto se puede representar de tres formas.

1. El nombre del objeto subrayado
2. El nombre del objeto seguido de dos puntos y del nombre de la clase al que pertenece, todos ellos subrayados
3. Dos puntos y el nombre de la clase a la que pertenece el objeto, ambos subrayados.

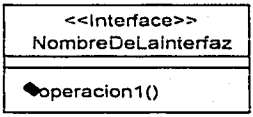
Un objeto se crea utilizando el operador "*new*" seguido del nombre de la clase a la que pertenecerá el objeto.




```
Clase objeto = new Clase();
```

3.2.2.1.3 Interfaz

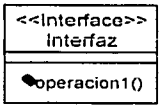
Una interfaz puede ser representada en UML de distintas formas, ya sea por un círculo que indica que es una interfaz o por una clase que incluye el estereotipo de <<interface>>.



NombreDeLaInterfaz

Una clase "interfaz" en UML se mapea como una interfaz en Java. La interfaz no tiene atributos y sus operaciones se representan como métodos abstractos. La declaración de una interfaz en Java se realiza utilizando la cláusula "*interface*" y el nombre de la interfaz. La sintaxis para declara una interfaz es:

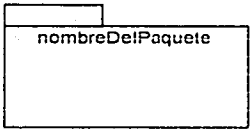
[*Modificadores de Clase*] **interface** *NombreDeLaInterfaz*
[**extends** *Interfaz1, Interfaz2 ...*] {
DeclaracionesDeMiembrosDeLaInterfaz;
}




```
public interface Interfaz {

   public abstract void operacion1();

}
```

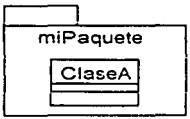
3.2.2.1.4 Paquete

Es un mecanismo de agrupación y pueden contener paquetes, clases y/o interfaces. Un paquete se representa como un folder, incluyendo su nombre.



Un paquete en UML sirve para organizar el modelo y se traducen directamente a Java dentro del sistema. La declaración de un paquete consta de la cláusula "*package*" y del nombre del paquete; esta declaración se introducirá en el código fuente de la clase que se encuentre dentro de ese paquete. Se declara un paquete como sigue:

package *nombreDelPaquete*




```
package miPaquete;

public class ClaseA {

}
```

3.2.2.1.5 Relaciones

Una relación es una conexión entre elementos. Las relaciones son de suma importancia por que nos ayudan a definir cómo estos elementos interactúan con algún otro. Comprender el significado de las relaciones en un modelo es el primer paso para entender como se implementarán en código Java.

3.2.2.1.5.1 Tipos de relaciones

Generalización Es un tipo de relación entre una clase general conocida como superclase o clase padre y una clase más específica de ese elemento llamada subclase o clase hija. La subclase es complemente consistente con la superclase y contiene información adicional. La generalización es una relación de tipo "*es un*" y es un elemento para modelar la herencia.

Dependencia Es una relación entre elementos, la cual implica un cambio en la especificación de un elemento que podría afectar a los elementos dependientes a el. En otras palabras, traduce algún tipo de referencia a una clase u objeto que no existe dentro del contexto.

Realización Es una relación entre dos elementos, en la cual un elemento especifica un contrato que otra entidad garantiza que cumplirá.

Asociación Es una relación entre objetos de la misma clase o diferentes clases. Una asociación implica que ambas clases tienen el mismo nivel conceptual. Existen dos tipos de relaciones de asociación:

- ***Unidireccional*** Es la relación que especifica la navegación en un solo sentido entre las clases conectadas.
- ***Bidireccional*** Es una asociación que implica la navegación en ambos sentidos entre las clases conectadas. Una asociación bidireccional se comporta como dos asociaciones unidireccionales separadas en direcciones opuestas.

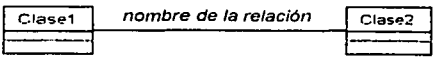
Agregación Es una forma de asociación que representa una relación "*es parte de*" entre dos clases, donde una clase es el "*todo*" y otra la "*parte*". Esto implica que el "*todo*" es conceptualmente de un nivel mayor que la "*parte*"

Composición Es una forma de agregación más fuerte, donde un "*todo*" es totalmente responsable de las partes y cada objeto "*parte*" es asociado sólo a un "*todo*". Por otro lado, la "*parte*" no necesita ser destruida cuando el "*todo*" es destruido, pero el "*todo*" es responsable de mantener viva a la "*parte*" o destruirla. La parte no puede ser compartida con otros "*todos*". El todo de cualquier forma puede transferir la pertenencia a otro objeto, el cual asume la responsabilidad.

3.2.2.1.5.2 Descripción de relaciones

No solo debe ser identificada la relación entre clases, sino que también se puede describir la relación por medio de:

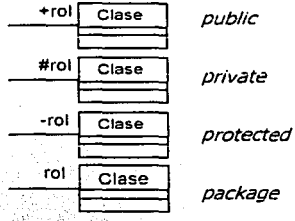
Nombre de la relación. El nombrar una relación sirve para clarificar su significado. Se representa con una etiqueta en cursivas en medio de la relación, por lo regular los nombres son un verbo o una frase. El nombre de la relación solo sirve para documentar en los diagramas y no tiene efecto en código.



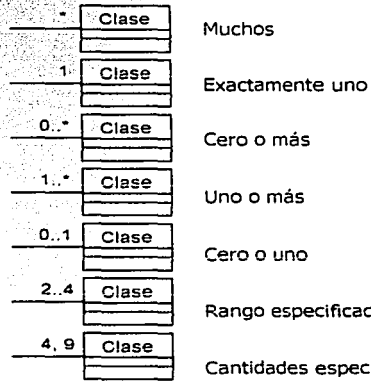
Navegación. Muchas de las relaciones tienen dirección. Se puede pensar como dirección como "quien puede ver a quien".

Rol. Es un nombre que va a identificar a un objeto de la clase al relacionarse con otra clase. Se describe el nombre del rol en la punta o puntas de la relación.

Un rol tiene un prefijo, el cual indica si el rol es accesible desde otras clases. Se conocen como modificadores y son los siguientes:



Multiplicidad. La multiplicidad indica el número de relaciones entre las instancias. Un indicador de multiplicidad es el número de objetos de una clase que hacen referencia a un objeto de otra clase. Hay un indicador de multiplicidad para cada fin de la relación. Los indicadores de multiplicidad son los siguientes:



La multiplicidad de cada dirección en la asociación determina el tipo de campo que se generará. Si la multiplicidad es 1 o no se especifica, un simple campo es generado; en el caso en que la multiplicidad sea un conjunto, existen dos opciones: declarar un arreglo para el campo o generar un campo basado en una clase contenedora.

Tanto el nombre de las relaciones, roles e indicadores de multiplicidad sólo aplican para las relaciones de asociación, agregación y composición.

Calificador. Atributo de una asociación cuyos valores son únicos dentro de un conjunto de objetos relacionado con un objeto en la asociación. Es decir, un objeto y un valor calificador identifican un único objeto a través de la asociación; ellos forman una llave compuesta.

3.2.2.1.5.3 Modelado de relaciones

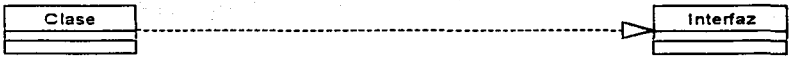
Generalización:

La generalización se diagrama en UML como una flecha continua que va de la clase hija a la clase padre.

Dependencia:

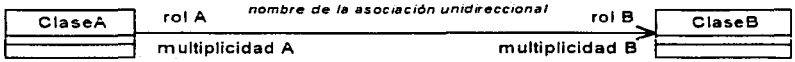
Gráficamente se representa como una línea discontinua dirigida a la clase de la cual se depende.

Realización:



Dentro de la notación UML la realización se representa como una flecha discontinua con una punta vacía.

Asociación unidireccional:



Una asociación unidireccional en UML se representa con una flecha continua que va de una clase a otra o a la misma.

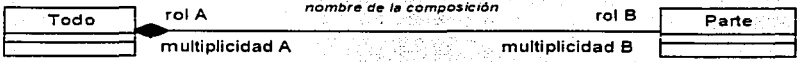
Asociación bidireccional:

Una asociación bidireccional se representa con una línea continua entre la misma clase o diferentes clases.

Agregación:

La agregación se representa con una línea con un rombo en el extremo del todo.

Composición:



Se representa a una relación de composición con una línea continua, la cual tiene un rombo relleno en el extremo del todo

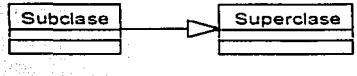
3.2.2.1.5.4 Implementación de relaciones

3.2.2.1.5.4.1 Generalización

Si una clase en UML tiene una superclase, al codificarla, la clase hereda de la clase padre. Java sólo soporta la herencia simple, esto es que una clase no puede heredar de más de una superclase. En código Java la relación de generalización se representa con la claúsula "*extends*". La claúsula "*extends*" acompaña a la declaración de una subclase que tiene una relación de generalización con una superclase. En otras palabras la generalización en UML se traduce a herencia en Java y se nota la jerarquía de clases. Este tipo de relación permite la reutilización de código.

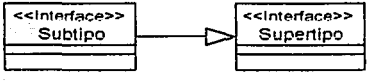
La sintaxis para declarar la herencia en Java es la siguiente:

[*Modificadores de Clase*] **class** *Subclase*
[**extends** *Superclase*] {
DeclaracionesDeMiembrosDeLaClase;
}




```
class Subclase extends Superclase {

}
```

La generalización entre interfaces se implementa de la siguiente forma:

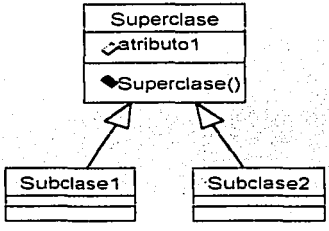



```
interface Subtipo extends Supertipo {

}
```

En Java, todas las clases son derivadas de una clase raíz. La clase *Object* es la raíz de la jerarquía de clases, dicha clase puede ser especificada o no. En el caso en que no se especifique la clase *Object* como superclase, Java genera la herencia implícitamente. Por tal motivo, *Object* no tiene una superclase.

Referencia a la superclase

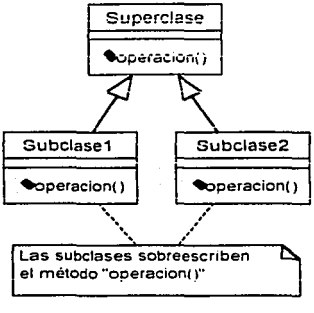
Se hace referencia a una variable o método de la clase padre con la palabra reservada "*super*".




```
public class Superclase {

   public tipo atributo1;
   Superclase() {
   }
}

public class Subclase1 {
   super.atributo1;
}

public class Subclase2 {
   super();
}
```

Sobreescritura de métodos

La sobreescritura es cuando métodos de clases diferentes tienen el mismo nombre de método, firma y tipo de retorno. Esto se da con las relaciones de generalización.

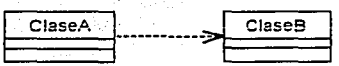



```
public class Superclase {

   public tipoDeRetorno operacion() {
   }
}

public class Subclase1 {

   public tipoDeRetorno operacion() {
   }
}

public class Subclase2 {

  public tipoDeRetorno operacion() {
   }
}
```

3.2.2.1.5.4.2 Dependencia

Las dependencias se utilizan para indicar que una clase utiliza a otra como argumento en la firma de una operación.

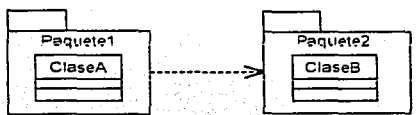
Este tipo de relación incluye a una variable local, la cual referencia a un objeto obtenido al llamar al método o a la referencia de un método estático de una clase, donde una instancia de esa clase no existe.




```
class ClaseA {

  void metodo(ClaseB parametro){

  }

}
```

Una dependencia también es utilizada para representar las relaciones entre paquetes. Porque un paquete contiene clases, nosotros podemos ilustrar a varios paquetes que tienen relaciones entre ellos basándose en las relaciones de las clases dentro de esos paquetes.

Las dependencias entre paquetes se implementan en Java utilizando la cláusula *"import"*. Dentro de la clase origen se incluye el paquete donde se encuentra la clase con la que se va a comunicar.

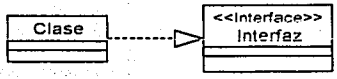



```
class ClaseA {

  import Paquete2;

}
```

3.2.2.1.5.4.3 Realización

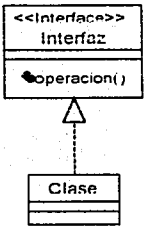
Una interfaz puede ser utilizada por una clase cuando en la declaración de la clase se agrega la cláusula "*implements*" seguida por el nombre de cada interfaz.

[*Modificadores de Clase*] **class** *Clase*
[**implements** *Interfaz1, Interfaz2...*] {
DeclaracionesDeMiembrosDeLaClase;
}




```
class Clase implements Interfaz {

}
```

Una clase que implementa una interfaz debe de declarar todos los métodos abstractos declarados en la interfaz, pero puede o no implementarlos. Un ejemplo es el siguiente código:

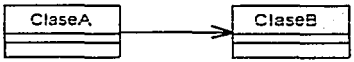



```
interface Interfaz {

  void operación(); // método abstracto

}

class Clase implements Interfaz {

  void operacion() {
  // implementación del método abstracto de la interfaz
  }

}
```

3.2.2.1.5.4.4 Asociaciones

3.2.2.1.5.4.4.1 Asociación unidireccional

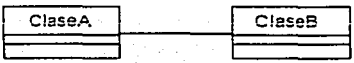
Una asociación unidireccional se traduce como el envío de mensajes en una sola dirección de la asociación; es decir se envían mensajes a una entidad pero no en viceversa. La implementación de una asociación unidireccional en Java se realiza agregando una variable de instancia de la clase destino en la clase que es el origen de la relación. El tipo de la variable de instancia corresponde a la clase destino.




```
class ClaseA {
  ClaseB nombreDeLaVariable;
}
```

3.2.2.1.5.4.4.2 Asociación bidireccional

Una asociación bidireccional se comporta como dos asociaciones unidireccionales separadas en direcciones opuestas. De esta forma las instancias de las clases participantes pueden enviarse mensajes. Se implementa una asociación bidireccional agregando una variable de instancia para ambas clases.



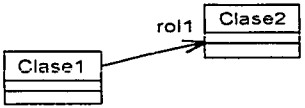

```
class ClaseA {
  ClassB variable1;
}
class ClaseB {
  ClaseA variable2;
}
```

3.2.2.1.5.4.4.3 Roles

Un nombre de un rol al implementarlo corresponde a el nombre de una variable de instancia de la clase. La notación correspondiente a *public/private/protected* corresponden a modificadores de miembros de clase en Java. El valor inicial de las variables puede ser especificado en la asociación.

Roles de una asociación unidireccional

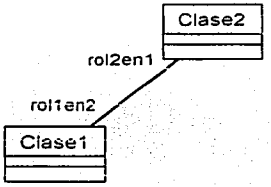
En una asociación unidireccional solo hay un rol, por lo cual se va a agregar una variable de instancia en la clase origen de la relación con el nombre del rol; esta variable va a ser del tipo de la clase destino.




```
class Clase1 {
  Clase2 rol1;
}
```

Roles de una asociación bidireccional

Para una asociación bidireccional deben de existir dos nombres de rol, uno para cada instancia de la clase. Ambos roles al codificar se convierten en variables de instancia según las clases correspondientes.

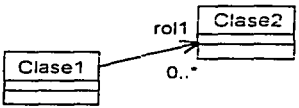



```
class Clase1 {
  Clase2 rol2en1;
}
class Clase2 {
  Clase1 rol1en2;
}
```

3.2.2.1.5.4.4.4 Multiplicidad

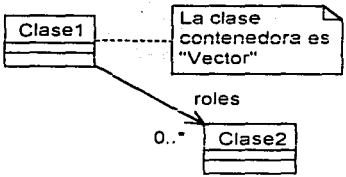
El tipo de multiplicidad que tenga una relación puede ser implementado en código Java. Si el indicador de multiplicidad es cero o uno se agrega una referencia simple. En el caso en que sea uno o más, cero o más o muchos; se agrega una "clase contenedora" como referencia. La clase contenedora va a guardar los objetos participantes en la relación. Por default la clase contenedora es un arreglo, pero se puede especificar alguna otra clase contenedora como: *java.util.Vector, java.util.Hashtable, etc.*

Multiplicidad de una asociación unidireccional



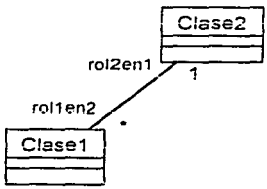

```
class Clase1 {
  Clase2 rol1[];
}
```

En el caso de que la clase contenedora se especifique en un dialogo, la variable de instancia será del tipo de la clase contenedora. Supongamos que se requiere utilizar a la clase Vector como clase contenedora:




```
class Clase1 {
  Vector roles;
}
```

Multiplicidad de una asociación bidireccional



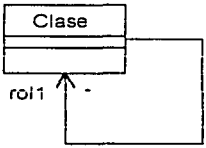

```
class Clase1 {
  Clase2 rol2en1;
}
class Clase2 {
  Clase1 rol1en2[];
}
```

3.2.2.1.5.4.4.5 Asociaciones reflexivas

Una asociación reflexiva hace referencia a objetos de la misma clase, esto indica que múltiples objetos de la misma clase colaboran juntos de alguna forma.

Asociación unidireccional reflexiva

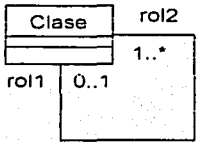
El resultado de codificar una asociación unidireccional reflexiva consiste en declarar al rol como una variable de instancia de la misma clase. La multiplicidad hace referencia a la misma clase.




```
class Clase{

  Clase rol1[];

}
```

Asociación bidireccional reflexiva

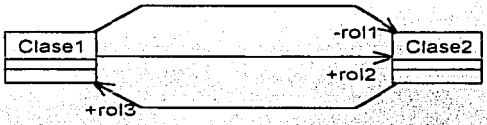
La implementación de una asociación reflexiva en el caso de ser bidireccional se realiza creando variables de instancia para cada uno de los roles, declarándolos en la misma clase.




```
class Clase {

  Clase rol1;
  Clase rol2[];

}
```

3.2.2.1.5.4.4.6 Asociaciones múltiples

Existen asociaciones múltiples cuando hay más de una relación entre dos clases. En Java se codifican creando una variable de instancia para cada uno de los roles.

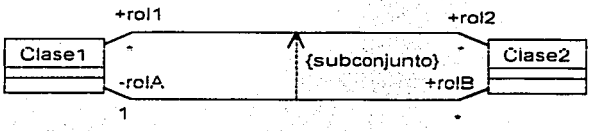



```
public class Clase1 {

  private Clase2 rol1;
  public Clase2 rol2;

}
```

```
public class Clase2 {

  public Clase1 rol3;

}
```

3.2.2.1.5.4.4.7 Restricciones

Una relación con restricciones requiere de aplicarle alguna regla. Por lo regular son utilizadas con asociaciones, agregaciones y composiciones.



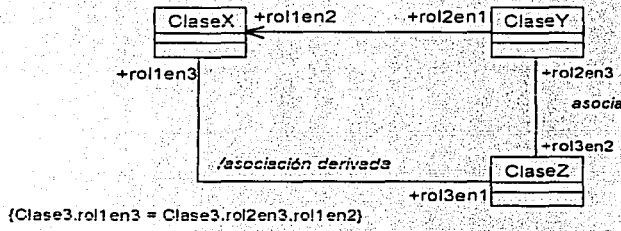

```
public class Clase1 {

  public Clase2 rol2[];
  public Clase2 rolB[];

}
```

```
public class Clase2 {

  public Clase1 rol1[];
  public Clase1 rolA;

}
```

3.2.2.1.5.4.4.8 Asociaciones derivadas

```
public class Clase1 {

   public Clase2 rol2en1[];

}
```

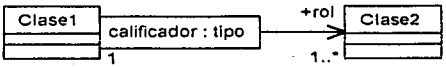
```
public class Clase2 {

   public Clase1 rol1en2;
   public Clase3 rol3en2[];

}
```




```
public class Clase3 {

   public Clase2 rol2en3;

}
```

3.2.2.1.5.4.4.9 Calificador

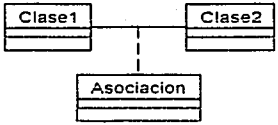
La implementación de una asociación con calificador es similar a la de una asociación con multiplicidad de muchos. El calificador es utilizado como una llave dentro de un método, dicho método va a recibir el calificador para almacenarse y regresar los valores guardados en una clase contenedora.




```
class Clase1 {
  public Vector rol;
  public Vector encontrarPorLlave (tipo calificador) {
  }
}
```

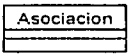
3.2.2.1.5.4.4.10 Clase asociación

Una clase asociación se utiliza cuando se define a la asociación. La clase asociación requiere de una asociación adicional que proviene de las clases que le dan origen.

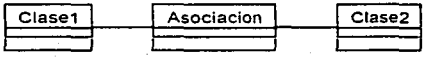


La implementación de una clase asociación en Java se basa en transformar la asociación realizando los siguientes pasos:

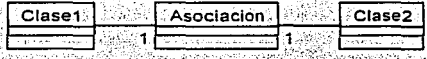
1. Se crea una nueva clase especial y se le da el nombre de la asociación.



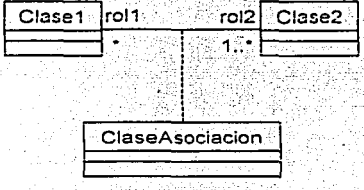
2. La relación se divide en dos.



3. Simplificar por una instancia especial para cada relación.



Al realizar los pasos anteriores queda codificada la clase asociación como sigue:

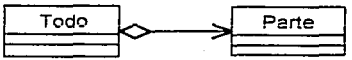



```
class Clase1 {

   ClaseAsociacion var;

}
class Clase2 {

   ClaseAsociacion var;

}
class ClaseAsociacion{

   Clase1 rol1[];
   Clase2 rol2[];

}
```

3.2.2.1.5.4.5 Agregaciones

3.2.2.1.5.4.5.1 Agregación unidireccional

Una agregación unidireccional no tiene significado en Java, por lo que se mapea de igual forma que una asociación unidireccional, es decir como una variable de instancia en la clase que representa el todo.




```
public class Todo {

   Parte variable;

}
```

3.2.2.1.5.4.5.2 Agregación bidireccional

Una relación de agregación bidireccional se traduce en Java con variables de instancia. La diferencia en Java entre una asociación bidireccional y una agregación bidireccional no es notable, sólo es una diferencia semántica. Si no se esta seguro cuando se quiera utilizar una agregación o una asociación, se recomienda utilizar la asociación. Las agregaciones en Java no se utilizan regularmente.


```
class Todo {

   Parte variable;

}

class Parte {

   Todo variable;

}
```

3.2.2.1.5.4.6 Composiciones

3.2.2.1.5.4.6.1 Composición unidireccional

Existen dos formas de representar a una composición unidireccional:

1. Como una variable de instancia en la clase que representa el todo. Es decir como asociación unidireccional.


```
class Todo {

  Parte variable;

}
```

2. Otra forma de implementar a la composición unidireccional es con una clase interna. Esta forma de implementar la composición puede limitar la reutilización.

```
class Todo {

  Parte variable;

  class Parte { // Parte es la clase interna

  }
}
```

3.2.2.1.5.4.6.2 Composición bidireccional

La composición bidireccional es una forma especial de agregación. No tiene significado en Java, por lo que se mapea como una asociación bidireccional.

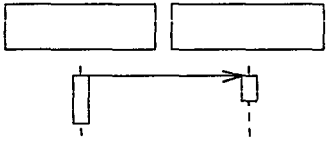

```
class Todo {

  Parte variable;

}

class Parte {

  Todo variable;

}
```

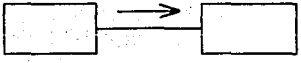
3.2.2.1.6 Mensajes

Un mensaje es un evento u operación que especifica la comunicación entre objetos al transmitir información. El paso de un mensaje, da como resultado una instrucción ejecutable como parte de una actividad dentro del sistema. Algunas de las acciones que pueden tener los mensajes sobre los objetos son: creación, envío de una señal, llamada de una operación, destrucción, etc.

3.2.2.1.6.1 Descripción de mensajes

Un mensaje se representa en UML con una flecha continua que va de un objeto a otro.



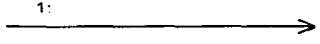


Un mensaje contiene información desde el origen de la invocación hasta el destino como:

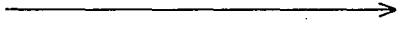
Número de secuencia. El enumerar los mensajes permite conocer el orden en que los mensajes se ejecutan. Al numerar la secuencia de los mensajes puede hacerse de dos formas

Ejemplo	Secuencia
1, 2, 3, 4, etc.	*alto nivel*
1.1, 1.2, 1.3, etc.	*jerárquica*

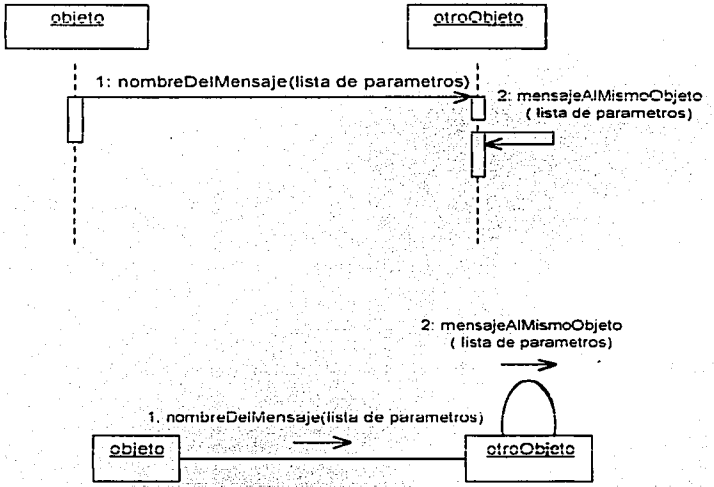
El número va arriba del mensaje



Nombre del mensaje y argumentos. Un mensaje tiene un nombre que va a describir a una operación y una posible lista de parámetros que va a recibir la operación.



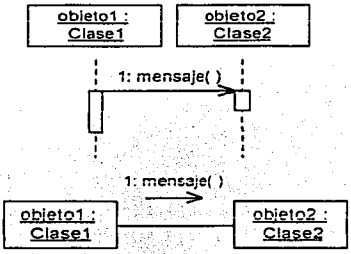
3.2.2.1.6.2 Modelado de mensajes



Un mensaje se representa gráficamente en UML como una flecha continua que va de un objeto a otro objeto. También existen mensajes enviados hacia el mismo objeto que lo invocó.

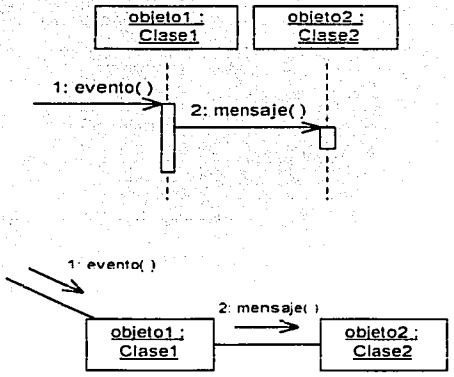
3.2.2.1.6.3 Implementación de mensajes

Los mensajes en UML se implementan en código Java como métodos. El método se declara en la clase a la que pertenece el objeto que recibe el mensaje.



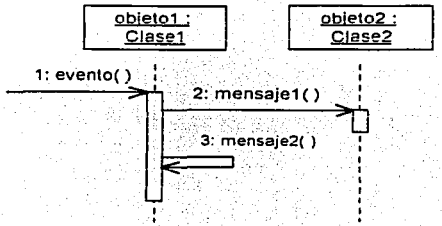

```
public class Clase2 {
  public void mensaje() {
  }
}
```

En el caso en que sea una secuencia de mensajes, se crea una variable de instancia dentro de la clase origen del objeto que recibe el mensaje. Posteriormente, se manda a llamar dentro de un método de la clase origen, el método de la clase que recibe el mensaje.

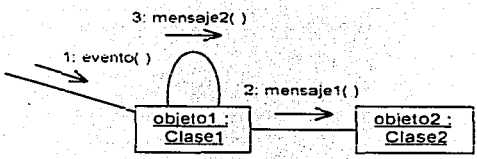



```
public class Clase1 {
  Clase2 objeto2;
  public void evento() {
    objeto2.mensaje();
  }
}

public class Clase2 {
  public void mensaje() {;
  }
}
```

Los mensajes que se dirigen hacia el mismo objeto son declarados como métodos en la clase a la que pertenece el objeto. El método es mandado a llamar dentro de la misma clase.

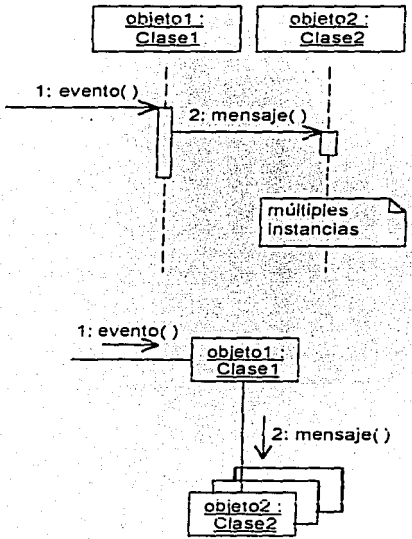



```
public class Clase1 {
  Clase2 objeto2;
  public void evento() {
    objeto2.mensaje1();
    mensaje2();
  }
}

public class Clase2 {
  public void mensaje1() {
  }
}
```



Se puede dar el caso de que un mensaje se dirija a un una clase que represente múltiples instancias de objetos. La implementación se realiza creando una variable de instancia del objeto que recibe el mensaje en la clase origen; como consiste de múltiples instancias se crea esa variable como un arreglo o del tipo de una clase contenedora.

Se implementa un ciclo para que se ejecute ese método a todas las instancias de la clase.



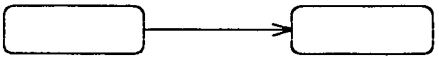

```
public class Clase1 {
  Clase2 objeto2[];
  public void evento() {
    for(int i=0; i < n; ++i)
    objeto2[i].mensaje();
  }
}

public class Clase2 {
  public void mensaje() {
  }
}
```

3.2.2.1.7 Estados y transiciones

Un estado es una posible condición en la que puede existir un objeto que pertenece a una clase. Es posible el pasar de un estado a otro a través de una transición; es decir, un cambio de un estado original a un estado sucesor.

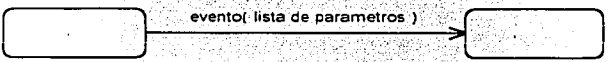
3.2.2.1.7.1 Descripción de estados y transiciones

Un estado de representa con un rectángulo redondeado de las puntas y una transición con una flecha continua que va de un estado a otro.

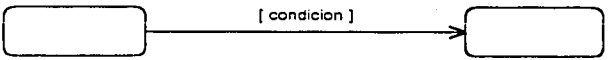


Una transición pueden contener:

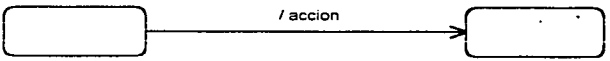
Evento. Es una ocurrencia que sucede en algún punto en el tiempo. se representa como parte de la transición.



Condición. Es una expresión boleana de un atributo que tiene un valor, y solo obedece a la transición si es verdadera. Se representa entre corchetes.

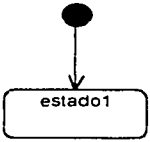


Acción. Es una operación que es asociada a una transición. A una acción se le antepone una diagonal.



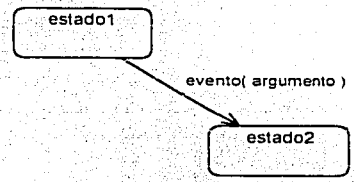
3.2.2.1.7.2 Implementación de estados

La idea básica en un diagrama de estados de una clase, es crear una variable de instancia en donde se guarde el registro del estado actual. En la definición de la clase se inicializa el estado en el constructor, según el estado del objeto al crearse.



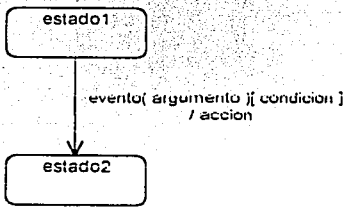

```
class Clase{

  String estado;

  Clase() {

    estado = "estado1";
  }
}
```

La información proporcionada por los diagramas de estados debe ser reflejada en los métodos de las clase, los eventos se implementan como métodos con sus respectivos argumentos.



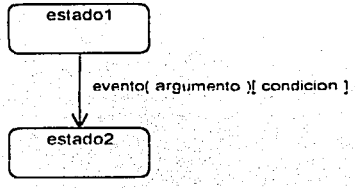

```
class Clase{

  String estado;

  void evento(argumento) {

    estado = "estado2";

  }

}
```

En el caso en el que el evento tenga una condición, esta se implementa con una sentencia "*if*" dentro del evento y se asigna el estado que adoptaría el objeto a la variable.



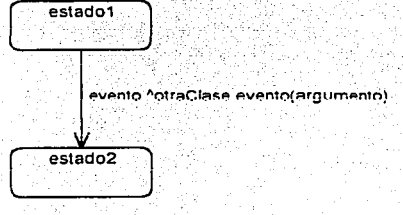

```
class Clase{

String estado;

  public void evento(argumento) {

    if (condicion){
      estado = "estado2";
    }
  }

}
```

Los eventos enviados a los objetos de la clase producen una reacción dependiendo del estado actual.



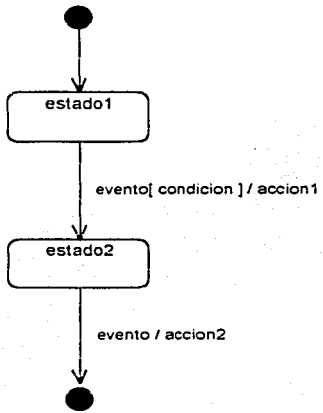

```
class Clase{

String estado;

  public void evento(argumento) {

    if (condicion) {

      estado = "estado2";
      accion;

    }

  }
}
```

Dentro del método de una clase se puede ejecutar el método a un objeto de otra clase.




```
class Clase{

String estado;

  public void evento() {

    estado = "estado2";
    otraClase.evento(argumento);

  }

}
```

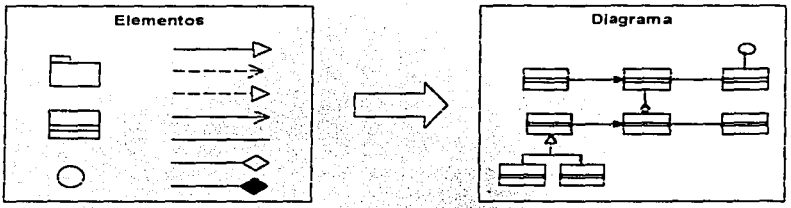
Al codificar los estados que tengan el mismo evento, se utiliza la sentencia "*switch*".




```
class Clase{

  public void evento() {

    switch (estado) {

      case estado1:
        if (condicion) {
          estado = "estado2";
          accion1;
        }
        break;

      case estado2:
        accion2;
        break;
    }

  }
}
```

3.2.2.2 Diagramas

Los diagramas en UML contribuyen a la especificación de un sistema de software. Se puede pensar en un diagrama como una composición de elementos, estos diagramas son el mecanismo que los desarrolladores utilizan para comunicar y resolver problemas en términos del sistema.



Un diagrama en UML está compuesto por un conjunto de elementos básicos de la notación.

Al comenzar a incorporar el UML en los desarrollos se sistemas, se utilizan diagramas individuales para comunicar la estructura del software y solucionar el problema a través de sus escenarios. Cada diagrama juega una parte importante dentro del desarrollo del sistema, y puede ser efectivo utilizar un diagrama en conjunción con algún otro para ayudar a entender el sistema.

Cada diagrama representa el sistema de una forma, los diagramas en UML se dividen en dos categorías:

Diagramas de comportamiento. Estos diagramas comunican los aspectos que contribuyen a satisfacer los requerimientos del sistema; es decir representan el aspecto dinámico del sistema mediante las colaboraciones entre elementos.

Hay cinco diagramas que pertenecen a esta categoría:

Casos de uso — Muestra un conjunto de actores y casos de uso, y las relaciones entre ellos. Un diagrama de casos de uso ayuda para organizar el modelo de acuerdo al comportamiento del sistema. Se centran en los procesos del negocio que la aplicación debe de soportar.

Actividad — Modela el flujo de actividades entre procesos. Representan visualmente el comportamiento capturado en un diagrama de casos de uso. No muestra la colaboración entre los objetos.

Estados Ilustra la secuencia de los estados de un objeto. Las transiciones entre estados ayudan a identificar y validar el comportamiento. Una clase puede tener más de un diagrama de estados

Secuencia Es un tipo de diagrama de interacción, el cual describe el orden de los mensajes entre objetos con respecto al tiempo. Semánticamente son el equivalente a los diagramas de colaboración.

Colaboración Es otro tipo de diagrama de interacción, el cual describe el comportamiento de los objetos cuando se envían y reciben mensajes. Semánticamente equivalen a un diagrama de secuencia.

Diagramas estructurales. Ilustran las relaciones que existen entre elementos físicos dentro del sistema, tal como las clases. dan a conocer los aspectos estáticos del sistema.

Los diagramas que pertenecen a esta categoría son tres:

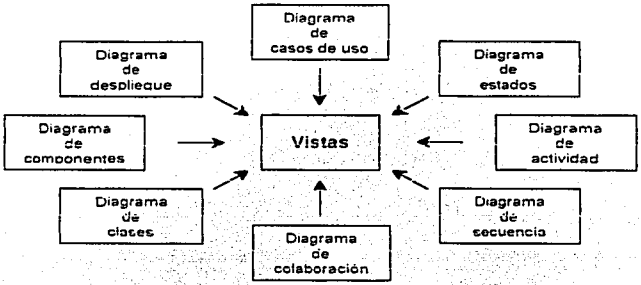
Clases Modelo estático, el cual ilustra un conjunto de clases, paquetes y sus relaciones. Detalla la estructura del sistema. Hay dos formas de diagramas de clases. El primero es el más común, esta formado por las clases que componen el sistema y de la estructura de esas clases. El segundo es el diagrama de paquetes, el cual representa los paquetes y las dependencias entre esos paquetes.

Componentes Representa las relaciones estáticas existentes entre los componentes de software dentro de la aplicación. Ejemplos de componentes: .exe, .dll, .ocx, y/o beans.

Despliegue Describe la topología física de un sistema y son de gran utilidad cuando la configuración del sistema es compleja. Incluye nodos, ya sean en forma de dispositivos (impresora, módem) o una computadora (servidor).

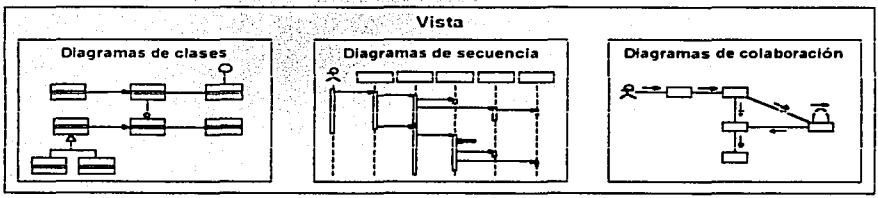
3.2.2.3 Vistas

Al utilizar una combinación de diagramas para comunicar el sistema se refiere a una vista. Una vista es una descripción del sistema desde una perspectiva en particular. Se representa el sistema desde diferentes perspectivas por la combinación de diagramas, esas combinaciones forman las vistas dentro del sistema.



Los aspectos que una vista represente van a ser capturados por una serie de diagramas.

Una vista es un artefacto que ayuda a simplificar la representación del sistema. Los diagramas en UML forman parte de diferentes vistas.



Una vista es una combinación de diagramas que permiten comunicar el sistema desde una perspectiva

Hay cinco vistas distintas que se asocian con el proceso de desarrollo de software.

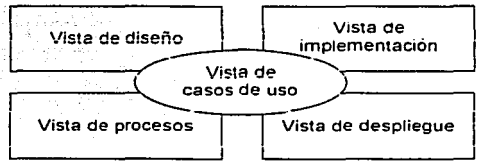
Casos de Uso Esta vista documenta el sistema desde la perspectiva del cliente. La terminología utilizada en esta vista debe ser del dominio específico. Los diagramas más comunes en esta vista son los diagramas de casos de uso y los diagramas de actividad.

Diseño Esta vista documenta el sistema desde la perspectiva de el diseñador y el arquitecto. Los diagramas más comunes en esta vista son los diagramas de clases y los diagramas de interacción (secuencia y colaboración).

Implementación Esta vista documenta los componentes por lo que esta compuesto el sistema. Esta vista contiene el diagrama de componentes. Esta vista es opcional, excepto para las aplicaciones Java más complejas.

Procesos Esta vista documenta los procesos que componen a la aplicación. Esos procesos son capturados en un diagrama de clases.

Despliegue Esta vista documenta la topología del sistema. El diagrama de desarrollo que forma parte de esta vista ilustra los nodos físicos y dispositivos que conforman a la aplicación, al igual que las conexiones entre ellos.



La arquitectura de un sistema de software esta compuesta por 4+1 vistas

Las diferentes vistas de un sistema son de suma importancia, por que los usuarios finales, desarrolladores y líderes de proyecto ven al sistema de forma diferente. Mientras que cada vista representa una perspectiva diferente, todas ellas representan al mismo sistema y deben ser consistentes en su información. Además cada vista puede ser utilizada para validar alguna otra.

Marco Metodológico

4. MARCO METODOLÓGICO

4.1 Variables

Independientes

Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos en UML en las etapas de análisis y diseño, los elementos de los modelos al comenzar la construcción del sistema pueden ser claramente representados en código Java.

Dependientes

- Facilitando la transición entre el diseño y la construcción del sistema.
- Permitiendo claridad y concordancia de los modelos con el código.
- Comunicando la arquitectura del sistema en base a objetos entre diseñadores y programadores.
- Permitiendo reducir los tiempos de elaboración del software al comenzar la etapa de construcción.

Tras haber realizado el Marco Teórico y el Marco Conceptual del tema de investigación, la variable independiente y las variables dependientes definidas en el Marco Problemático no sufrieron ningún cambio. Por tal motivo se retoman para aprobar y comprobar la hipótesis definitiva.

4.2 Variables de control

Para el tema de investigación no se necesita incluir las variables de control intervinientes y/o distorsionantes.

4.3 Hipótesis definitiva

De acuerdo a las relaciones hipotéticas mencionadas anteriormente, se dio origen a la que enseguida se presenta como mi hipótesis definitiva.

"Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos en UML en las etapas de análisis y diseño; los elementos de los modelos al comenzar la construcción del sistema pueden ser claramente representados en código Java; esto permite que la codificación no sufra ninguna forma de conversión de acuerdo a las características del diseño y que la implementación sea más fácil".

4.4 Definición del universo

El universo considerado para la investigación se compone de empresas involucradas en el desarrollo de sistemas de software orientado a objetos, dichas organizaciones deben de utilizar el UML como notación para documentar el análisis y diseño, y la implementación es realizada en lenguaje Java.

Por la limitación de recursos (tiempo, económicos e información disponible), no me es posible hacer una cuantificación próxima a la realidad, razón por la que la muestra se determinará como se explica en el siguiente punto.

4.5 Determinación de la muestra

Debido a que no pretendo hacer una investigación de prueba hipotética plena, el tipo de muestra seleccionada de acuerdo al tipo de investigación, corresponde a una muestra no probabilística por juicio.

La opinión de las personas dedicadas al desarrollo de software orientado a objetos con UML y Java, así como los conocimientos adquiridos durante el desarrollo de esta tesis, será lo que me permitirá concluir con un criterio para aprobar o no la relación hipotética.

4.6 Definición del método de la investigación

El método de investigación que se aplicará para la probanza de la hipótesis del trabajo de investigación, son las encuestas. A través de las encuestas se busca conocer las opiniones y puntos de vistas de profesionales en el tema.

4.7 Costo de la investigación

Tipo	Descripción	Parcial	Total
Recursos Humanos			
	Investigador	$ 70,000	
Total Recursos Humanos			$ 70,000
Bienes Inmuebles			
	Renta local	$ 20,000	
Total Bienes Inmuebles			$ 20,000
Bienes Muebles			
	Equipo de cómputo	$ 12,000	
	Impresora	$ 4,500	
Total Bienes Muebles			$ 16,500
Directos	Papel	$ 600	
	Cartuchos para impresora	$ 1,700	
	Disquete	$ 300	
Total Directos			$ 2,600
Indirectos	Teléfono	$ 2,300	
	Luz	$ 3,000	
	Internet	$ 3,600	
	Copias	$ 300	
	Otros	$ 500	
Total Indirectos			$ 9,700
Total			$ 118,300

4.8 Cuestionario

4.8.1 Construcción del cuestionario

Pregunta	Justificación	Respuesta esperada
1. Considera usted que sería fácil hacer la codificación de un sistema sin apoyarse de modelos?	Ver que tan importante es disponer de modelos y diagramas para comenzar a programar.	*No*
2. ¿Cree usted que el desarrollo de un sistema con un análisis, diseño y programación orientados a objetos facilita la transición entre una etapa con otra? y ¿Por qué?	Saber cómo es considerado el desarrollo de software orientado a objetos al pasar de una etapa a otra.	*Sí, porque se definen las clases y objetos que se utilizarán en el sistema y se trata el desarrollo en base a objetos.*
3. Por el motivo de que UML es un lenguaje de modelado a base de objetos y Java es un lenguaje de programación orientado a objetos, ¿Considera que hay un mapeo significativo cuando se convierten los modelos en UML a aplicaciones Java?	Verificar la relación existente entre UML como Java en el desarrollo de software.	*Sí, porque ambos lenguajes están basados en el paradigma de orientación a objetos*
4. ¿Cree que es útil el conocer que representa cada elemento de un diagrama en UML en código Java?	Mostrar que es necesario saber la relación UML/Java.	*Sí*

5. ¿Cree usted que se reduzcan los tiempos al comenzar a codificar por la interpretación de los modelos de UML a Java?	Conocer si se minimizan los tiempos de implementación al mapear diseños UML a Java.	*Sí*
6. ¿Cuáles principios y conceptos de orientación a objetos a utilizado tanto en UML como en Java?	Obtener información de los elementos específicos de UML que se pueden codificar en Java.	*Clase, objeto, herencia, polimorfismo, encapsulación, etc.*
7. ¿Cuáles diagramas en UML considera que representan código Java? y ¿Por qué?	Ver cuales diagramas en UML representan elementos en código Java.	*Diagrama de clases.* *Diagrama de secuencia.* *Diagrama de colaboración.* *Diagrama de estados.*
8. ¿En que proporción los modelos en UML generados en las etapas de análisis y diseño le han sido útiles al codificar en Java?	Evaluar cuánto son útiles los modelos al implementarlos.	*Mucha*
9. ¿Qué ventajas encuentra al utilizar modelos en UML para comenzar a codificar un sistema en Java?	Conocer las ventajas existentes al pasar del diseño a la implementación del sistema.	*Facilita el paso a la codificación.* *Se reduce el tiempo de la implementación.* *Existe una transición de modelos UML a código Java.*

4.8.2 Cuestionario piloto

1. ¿Considera usted que sería fácil hacer la codificación de un sistema sin apoyarse de modelos?

   ( ) Sí ( ) No

2. ¿Cree usted que el desarrollo de un sistema con un análisis, diseño y programación orientados a objetos facilita la transición entre una etapa con otra?

   ( ) Sí ( ) No

   Por qué: _____

3. Por el motivo de que UML es un lenguaje de modelado a base de objetos y Java es un lenguaje de programación orientado a objetos, ¿Considera que hay un mapeo significativo cuando se convierten los modelos en UML a aplicaciones Java?

   ( ) Sí ( ) No

   Por qué: _____

4. ¿Cree que es útil el conocer qué representa cada elemento de un diagrama en UML en código Java?

   ( ) SI ( ) NO

5. ¿Ha visto usted la reducción de tiempos al comenzar a codificar cuando se interpretan los modelos en UML a Java?

   ( ) SI ( ) NO

6. ¿Cuáles principios y conceptos de orientación a objetos ha utilizado tanto en UML como en Java?

   ( ) Clase ( ) Objeto ( ) Herencia
   ( ) Polimorfismo ( ) Encapsulación ( ) Mensaje

Otros: _____

7. ¿Cuáles diagramas en UML considera que representan código Java? y ¿Por qué?

Diagrama	**Razón**
( ) Actividad	_____
( ) Casos de Uso	_____
( ) Clases	_____
( ) Colaboración	_____
( ) Estados	_____
( ) Secuencia	_____

Otros: _____

8. ¿En qué proporción los modelos en UML generados en las etapas de análisis y diseño le han sido útiles al codificar en Java?

( ) Toda
( ) Mucha
( ) Medianamente
( ) Poca
( ) Nada

9. ¿Qué ventajas encuentra al utilizar modelos en UML para comenzar a codificar un sistema en Java?

_____

4.8.3 Cuestionario definitivo

1. ¿Considera usted que sería fácil hacer la codificación de un sistema sin apoyarse de modelos?

   ( ) Sí ( ) No

2. ¿Cree usted que el desarrollo de un sistema con un análisis, diseño y programación orientados a objetos facilita la transición entre una etapa con otra?

   ( ) Sí ( ) No

   Por qué: _____

3. Por el motivo de que UML es un lenguaje de modelado a base de objetos y Java es un lenguaje de programación orientado a objetos, ¿Considera que hay un mapeo significativo cuando se convierten los modelos en UML a aplicaciones Java?

   ( ) Sí ( ) No

   Por qué: _____

4. ¿Cree que es útil el conocer qué representa cada elemento de un diagrama en UML en código Java al comenzar a programar?

   ( ) SI ( ) NO

5. ¿Ha visto usted la reducción de tiempos al comenzar a codificar cuando se interpretan los modelos en UML a Java?

   ( ) SI ( ) NO

6. ¿Cuáles principios y conceptos de orientación a objetos ha utilizado tanto en UML como en Java?

   ( ) Clase ( ) Objeto ( ) Herencia
   ( ) Polimorfismo ( ) Encapsulación ( ) Mensaje

Otros: _____

_____

7. ¿Cuáles diagramas en UML considera que representan código Java? y ¿Por qué?

Diagrama	**Razón**
( ) Actividad	_____
( ) Casos de Uso	_____
( ) Clases	_____
( ) Colaboración	_____
( ) Estados	_____
( ) Secuencia	_____

Otros: _____

_____

8. ¿En qué proporción los modelos en UML generados en las etapas de análisis y diseño le han sido útiles al codificar en Java? Seleccione un porcentaje.

( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( )

0 10 20 30 40 50 60 70 80 90 100 %

Nada | Poca | Medianamente | Mucha | Toda

9. ¿Qué ventajas encuentra al utilizar modelos en UML para comenzar a codificar un sistema en Java?

_____

_____

_____

4.9 Realización de la investigación

Las personas entrevistadas que contestaron el cuestionario son:

Juan Carlos Chávez García
AAIDA Organismo Integrado de Distribución S.A. de C.V.

Claudia Herrera Cervantes
Infotec

Edgar Mendoza Arreola
Circulo Interware de México S.C.

Ma. Teresa Ventura Miranda
Subdirección de Sistemas, UNAM.

Milton Arturo García Lima
Becario DGSCA-UNAM

Roberto Rojas Plata
Radiomovil DIPSA, S.A. de C.V.

4.10 Captura de información

1. Considera usted que sería fácil hacer la codificación de un sistema sin apoyarse de modelos?

Juan Carlos Chávez	*Claudia Herrera*	*Edgar Mendoza*	*Ma. Teresa Ventura*	*Milton García*	*Roberto Rojas*
No	No	No	No	No	No

Conclusión: Todos los entrevistados señalan la importancia de basarse en modelos conceptuales para comenzar la implementación del sistema. Así la arquitectura propuesta por los diseñadores del sistema será utilizada por los programadores para comenzar a codificar.

2. ¿Cree usted que el desarrollo de un sistema con un análisis, diseño y programación orientados a objetos facilita la transición entre una etapa con otra?

Juan Carlos Chávez	*Claudia Herrera*	*Edgar Mendoza*	*Ma. Teresa Ventura*	*Milton García*	*Roberto Rojas*
Sí	Sí	Sí	Sí	Sí	Sí

¿Por qué?

Juan Carlos Chávez	Porque a través de modelos sencillos de interpretar por todas las personas que intervienen en cada una de las etapas mencionadas arriba, se incrementa la comunicación y esto se ve reflejado en el cumplimiento de los objetivos del sistema y el equipo de trabajo
Claudia Herrera	El manejo de objetos facilita el uso de estándares, ya sea para modelado, como es el caso de UML, en programación JAVA, XML, etc. Permite de manera muy precisa la comunicación entre programas, a través de las interfaces, el re-uso de componentes, etc. Existen metodologías, en el caso de la industria del software, en donde para la comunicación por etapas, el uso de objetos permite la continuidad y la comunicación, entre los diferentes perfiles que intervienen en el desarrollo de un proyecto en cada una de las etapas. En mi opinión las Best Practices de la industria del software no serían posibles de manera transparente sin el manejo de objetos.
Edgar Mendoza	Depende de la metodología utilizada, pero en general los productos generados en cada etapa fungen como una base sólida para las subsecuentes en un contexto de desarrollo iterativo.
Ma. Teresa Ventura	Sobretodo si se implementa en objetos también. Los modelos son la base conceptual del desarrollo; entre más detallados, más transparente es la implementación.

Milton García	Porque es fácil redireccionar los resultados de un modulo, función o método aplicado a determinada etapa del manejo de datos hacia etapas consecutivas o posteriores, además facilita la detección de errores y mejora la funcionalidad de los sistemas.
Roberto Rojas	Se tiene un modelo de referencia entre cada etapa.

Conclusión: Se coincide en que el desarrollo de software orientado a objetos facilita el pasar de una etapa a otra; lo cual, es debido a que se desarrollan las fases bajo el paradigma de objetos. Pero todos consideran necesario que el análisis, diseño y programación en realidad sean la base del desarrollo, para que al terminar una etapa se utilice lo generado para la etapa que comienza.

3. Por el motivo de que UML es un lenguaje de modelado a base de objetos y Java es un lenguaje de programación orientado a objetos, ¿Considera que hay un mapeo significativo cuando se convierten los modelos en UML a aplicaciones Java?

Juan Carlos Chávez	***Claudia Herrera***	***Edgar Mendoza***	***Ma. Teresa Ventura***	***Milton García***	***Roberto Rojas***
Sí	Sí	Sí	Sí	No	Sí

¿Por qué?

Juan Carlos Chávez	***Claudia Herrera***	***Edgar Mendoza***	***Ma. Teresa Ventura***	***Milton García***	***Roberto Rojas***
Es bastante claro el diseño de clases a partir de modelados con UML, ya que provee las características principales de los componentes que debe incluir.	Sí, de hecho las personas que desarrollaron UML, tenían la necesidad de crear una metodología que sirviera de herramienta para ese mapeo significativo, dado que son de los mejores programadores en OO.	Porque UML es lo suficientemente flexible como para modelar las estructuras del lenguaje JAVA, así como Java es lo suficientemente flexible como para permitir la implementación de un diseño orientado a objetos.	Entre más detallado sea el diseño, la codificación es un proceso de "traducción" que se facilita mucho más. Además UML originalmente surgió con un enfoque de implementación OO que actualmente se ha extendido a todas las fases del desarrollo.	Pocas herramientas hacen esta conversión y no es muy completa ya que la interpretación que el programador desea dar en un modelo de determinadas acciones que se deben realizar con los datos no la interpretan como se debe y dicho mapeo no resulta tan funcional.	Sin importar el lenguaje de programación que se ocupe, UML es una herramienta sólida para tener un diseño funcional adecuado.

Conclusión: Se afirma por parte de los entrevistados la interpretación de los modelos de UML a Java, ya que ambos representan elementos y conceptos de orientación a objetos y sobre todo hay una conversión o traducción entre modelo-código.

4. *¿Cree que es útil el conocer que representa cada elemento de un diagrama en UML en código Java al comenzar a programar?*

Juan Carlos Chávez	Claudia Herrera	Edgar Mendoza	Ma. Teresa Ventura	Milton García	Roberto Rojas
Sí	Sí	No	Sí	Sí	Sí

Conclusión: La mayoría opina que de alguna u otra forma sí tiene beneficio para el programador conocer la representación de los modelos en código.

5. *¿Cree usted que se reduzcan los tiempos al comenzar a codificar por la interpretación de los modelos de UML a Java?*

Juan Carlos Chávez	Claudia Herrera	Edgar Mendoza	Ma. Teresa Ventura	Milton García	Roberto Rojas
Sí	Sí	Sí	No	No	Sí

Conclusión: En este punto, los entrevistados creen en su mayoría que los tiempos de implementación efectivamente se reducen. Al momento en que el programador recibe durante el comienzo de la programación los modelos, diagramas y documentación de un sistema, puede agilizar el tiempo de desarrollo, todo ello dependiendo del conocimiento que tenga para interpretar los elementos del UML en código Java.

6. *¿Cuáles principios y conceptos de orientación a objetos a utilizado tanto en UML como en Java?*

	Juan Carlos Chávez	Claudia Herrera	Edgar Mendoza	Ma. Teresa Ventura	Milton García	Roberto Rojas
Clase	X	X	X	X	X	X
Objeto	X	X	X	X	X	X
Herencia	X	X	X	X	X	X
Polimorfismo	X	X	X	X	X	X
Encapsulación	X	X	X		X	X
Mensaje	X	X	X	X	X	

Conclusión: La clase, objeto, herencia, polimorfismo, encapsulación y mensaje han sido utilizados por los entrevistados durante el desarrollo de un sistema. Todos esos principios y conceptos se representan como parte de la notación UML y en el lenguaje de programación Java.

7. *¿Cuáles diagramas en UML considera que representan código Java? y ¿Por qué?*

	Juan Carlos Chávez	Claudia Herrera	Edgar Mendoza	Ma. Teresa Ventura	Milton García	Roberto Rojas
Actividad	X	X				
Casos de uso						
Clases	X	X	X	X	X	X

Colaboración	X	X	X	X		
Estados	X	X	X	X		
Secuencia	X	X	X	X		X

Conclusión: De todos los diagramas existentes como parte del UML, los cuatro que representan y especifican elementos en código Java son: el diagrama de clase, ya que representa clases con sus miembros, relaciones y paquetes; el diagrama de colaboración muestra los métodos y mensajes entre objetos respecto al tiempo, el diagrama de estados considera el valor de inicio de las variables, condiciones, etc.; y el diagrama de secuencia también muestra métodos y mensajes entre objetos.

8. ¿En que proporción los modelos en UML generados en las etapas de análisis y diseño le han sido útiles al codificar en Java?

	Juan Carlos Chávez	***Claudia Herrera***	***Edgar Mendoza***	***Ma. Teresa Ventura***	***Milton García***	***Roberto Rojas***
Toda						
Mucha	X	X	X			
Medianamente				X	X	X
Poca						
Nada						

Conclusión: Se consideran de mediana y mucha utilidad la información que proporcionan los modelos conceptuales; básicamente los diagramas antes mencionados muestran detalles de la programación.

9. ¿Qué ventajas encuentra al utilizar modelos en UML para comenzar a codificar un sistema en Java?

Juan Carlos Chávez	Llevar un orden correcto para darle seguimiento a todo el proyecto, incrementando la efectividad al programar y reducir el tiempo.
Claudia Herrera	Las ventajas principales son que los modelos en UML permiten considerar y visualizar tanto por el usuario, como el analista, programadores, etc los requerimientos del usuario para evitar omisiones, también permiten darle trazabilidad a dichos requerimientos. Los modelos de UML también permiten desde la etapa de análisis identificar las clases, interfaces, mensajes, etc. Y así permitir desde un inicio el contemplar la re-usabilidad, etc. y distribuir el trabajo entre los integrantes del equipo de manera más transparente para la optimización de tiempos.
Edgar Mendoza	La semántica de los métodos y atributos de las clases están definidos y documentados. Ayuda a identificar firmas de métodos y tipos de datos con facilidad. Simplifica la organización de las clases que conforman una pieza de software en paquetes. Reduce el tiempo de desarrollo pues permite que el programador se preocupe por implementar las clases evitándole en buena parte preocuparse por la forma en que debe organizarlas o incluso en la forma en que debe diseñarlas para cumplir con los

	requerimientos del software.
Ma. Teresa Ventura	Dependiendo del detalle del modelo. Pero de entrada, ya se cuenta con las clases, atributos y métodos; ya se tiene la visión de todos los componentes que hay que implementar; el comportamiento estático y dinámico del sistema, y los casos de uso nos dan además, el panorama de la funcionalidad del sistema.
Milton García	Da una idea general de lo que se debe codificar, tal vez separando cada actividad y a su vez mostrando todo lo que se realiza dentro de cada una de estas.
Roberto Rojas	Tener un diseño confiable para la etapa de desarrollo y a partir de este codificar el sistema.
Conclusión: Las ventajas convergieron básicamente en que la codificación no sufra ninguna forma de conversión de acuerdo a las características especificas del diseño, reducir los tiempos de elaboración del software en la etapa de construcción y definir clases y objetos de acuerdo al problema en cuestión.	

4.11 Análisis de los resultados

Los resultados obtenidos de los cuestionarios aplicados a los entrevistados se analizarán pregunta por pregunta, de tal forma que se emitirá una conclusión por cada una de las mismas.

4.12 Conclusiones

1. La importancia de disponer de modelos al comenzar a programar el sistema es de gran utilidad, debido a que sirven de apoyo para comunicar los requerimientos del sistema y sobre todo comunicar a los programadores la arquitectura del sistema.

2. La transición entre etapas del proceso de desarrollo de software orientado a objetos, permite que se desarrolle el sistema bajo el mismo paradigma. Es decir, la estructura y arquitectura del sistema se va a ver reflejada en objetos y clases.

3. El conocer la correspondencia entre modelos en UML a aplicaciones Java, es de gran utilidad debido a que se reducen tiempos al comenzar a programar el sistema y facilitan el paso a la codificación.

4. Los principios y conceptos de orientación a objetos utilizados tanto en UML y Java son: clase, objeto, herencia, polimorfismo, encapsulación y mensaje; todos ellos forman parte de los diseños en UML.

5. Los diagramas que representan código Java son el diagramas de: clases, secuencia, colaboración y estados.

4.13 Aprobación de la hipótesis

Al haber evaluado la investigación contenida en este trabajo y establecer las conclusiones precedentes, considero aprobatoria la hipótesis presentada.

"Durante el desarrollo de un sistema orientado a objetos, una vez que han sido generados los modelos en UML en las etapas de análisis y diseño; los elementos de los modelos al comenzar la construcción del sistema pueden ser claramente representados en código Java; esto permite que la codificación no sufra ninguna forma de conversión de acuerdo a las características del diseño y que la implementación sea más fácil".

Marco Instrumental

5. MARCO INSTRUMENTAL

5.1 Propuestas de acción

Al haber realizado esta investigación he considerado conveniente llevar a cabo las siguientes actividades:

1. Difusión de la información a compañeros de carrera y de trabajo e interesados en el tema, por medio de un archivo vía correo electrónico y disquete.

2. Construcción de una página web, con información correspondiente al tema de investigación de la tesis, dónde se describen e ilustran los elementos fundamentales del UML, los diagramas en los que se utilicen y como ellos son mapeados al lenguaje Java. La información esta ubicada en la dirección: *http://www.geocities.com/mapeo_disenios_uml_java/index.html*. Se agrega la impresión de la página web en el Anexo.

3. Elaboración de un artículo con los aspectos esenciales de la investigación. El artículo se entregó para su posible publicación a la Secretaria de Divulgación y Fomento Editorial de la Facultad de Contaduría y Administración, UNAM. Como parte del Anexo se incluye el oficio entregado.

4. Preparación de una serie de diapositivas; con la finalidad de realizar una presentación del tema de investigación a alumnos de la licenciatura en informática de la Facultad de Contaduría y Administración, UNAM. La propuesta se realizó a la División de Informática de la FCA, UNAM. El oficio entregado se encuentra en el Anexo.

5. Definición de temas relacionados con el desarrollo de software orientado a objetos, para proponer la inclusión de la materia de "Desarrollo de software orientado a objetos con UML y Java" como materia optativa al plan de estudios de la Licenciatura en Informática, ya que en la actualidad no existe dicha materia. Dicho plan se entregó a la División de Informática de la Facultad de Contaduría y Administración, UNAM. El oficio entregado se agrega en la parte de anexos.

Conclusiones

6. CONCLUSIONES

1. Los modelos generados en las etapas de análisis y diseño utilizados al comenzar a codificar cualquier sistema de información, facilitan a los programadores el trabajo de generación de código en la etapa de implementación, debido a que proporciona la arquitectura del sistema.

2. Más aún, al realizar un desarrollo de software orientado a objetos permite que en cada una de las etapas como en el análisis, diseño e implementación, el paso entre una etapa a otra sea con base a objetos, lo que permite el que se comunique el sistema de la misma forma.

3. UML es un lenguaje de modelado orientado a objetos y Java un lenguaje de programación orientación a objetos, por tal motivo los elementos de un diagrama pueden ser claramente representados en código; el conocer la interpretación de elementos UML-Java puede ser de gran utilidad al comenzar la codificación del sistema, ya que se reducen los tiempos de implementación al iniciar esta etapa.

4. Los modelos en UML una vez que han sido diseñados pueden representar detalles específicos de Java; con la finalidad de especificar y detallar cuestiones de implementación; esto en el caso de conocer que la codificación del sistema se va a realizar en ese lenguaje de programación.

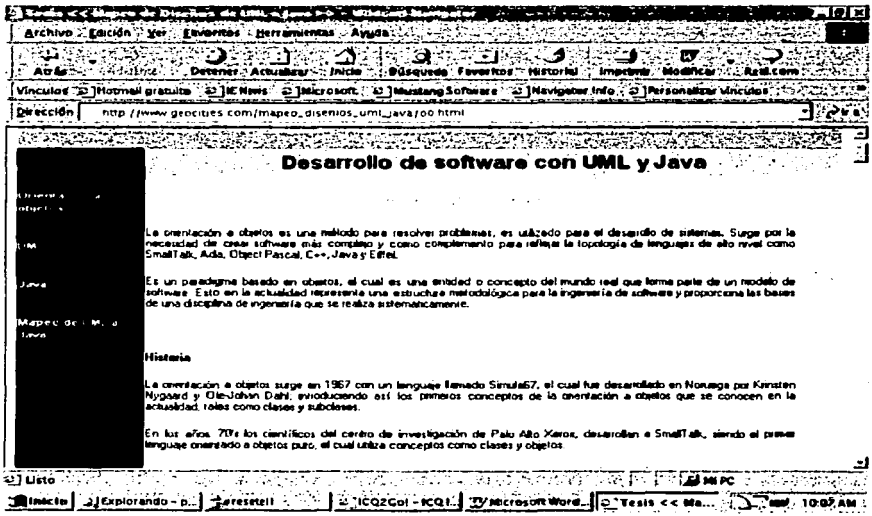
5. Algunos de los elementos modelados en un diagrama en UML nunca llegarán a ser código, por ejemplo los actores en los diagramas de casos de uso. Sin embargo, los modelos en UML, como los diagramas de clase, colaboración, secuencia y estados; proporcionan un mapeo significativo cuando se convierten a aplicaciones Java, en esta traducción modelo-código los más utilizados son los diagramas de clases, posteriormente algunos utilizan los diagramas de interacción (secuencia y colaboración) y por último a los diagramas de estados.

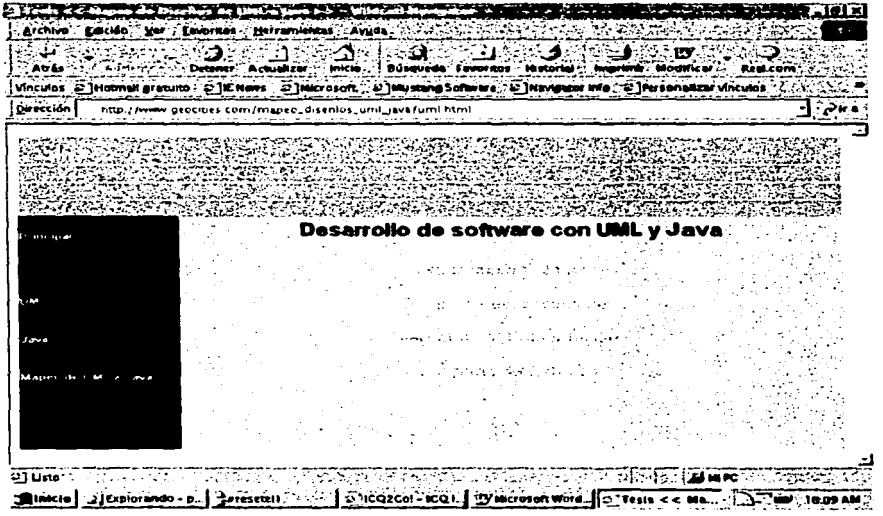
6. En los modelos en UML se representan el detalle de las clases, como atributos y métodos, así como mensajes entre objetos y la transición de estados de un objeto, proporcionando al implementar el sistema reducir tiempos de codificación y comunicar la organización del sistema.

Anexos

7. ANEXOS

Página web


Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Detener Actualizar Inicio Búsqueda Favoritos Historial Imprimir Modificar Real.com
Vínculos Hotmail gratuito IE News Microsoft Mustang Software Navigator Info Personalizar vínculos
Dirección http://www.geocities.com/mapeo_disenios_uml_java/oo.html
Desarrollo de software con UML y Java
UML
Java
Mapeo de UML a Java
La orientación a objetos es una método para resolver problemas, es utilizado para el desarrollo de sistemas. Surge por la necesidad de crear software más completo y como complemento para reflejar la topología de lenguajes de alto nivel como SmallTalk, Ada, Object Pascal, C++, Java y Eiffel.
Es un paradigma basado en objetos, el cual es una entidad o concepto del mundo real que forma parte de un modelo de software. Esto en la actualidad representa una estructura metodológica para la ingeniería de software y proporciona las bases de una disciplina de ingeniería que se realiza sistemáticamente.
Historia
La orientación a objetos surge en 1967 con un lenguaje llamado Simula67, el cual fue desarrollado en Noruega por Kirnsten Nygaard y Ole-Johan Dahl, introduciendo así los primeros conceptos de la orientación a objetos que se conocen en la actualidad, roles como clases y subclases.
En los años 70's los científicos del centro de investigación de Palo Alto Xerox, desarrollan a SmallTalk, siendo el primer lenguaje orientado a objetos puro, el cual utiliza conceptos como clases y objetos.
Listo
Mi PC
Inicio
Explorando - p...
presetel
ICQ2Go! - ICQ t...
Microsoft Word...
Tesis << Ma...
10:07 AM


Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Detener Actualizar Inicio Búsqueda Favoritos Historial Imprimir Modificar Real.com
Vínculos Hotmail gratuito IE News Microsoft MyMustang Software Navegador Info Personalizar vínculos
Dirección http://www.geocities.com/mapec_disenios_uml_java/uml.html
Desarrollo de software con UML y Java
UML
Java
Lista
Inicio
Explorando - D...
Presentell
ICQ2Go! - ICQ I...
Microsoft Word
Tesis << Ma...
Mi PC
10:09 AM

Articulo

Mapeo de diseños de UML a Java

Autor Jesús Cuauhtémoc García Flores

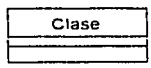
Durante el desarrollo de software la utilización de modelos permite representan diferentes vistas de los sistemas. La claridad de los modelos es indispensable para construir sistemas robustos. Así surge el UML, el cual es un lenguaje de modelado que esta compuesto de una notación. Los elementos de la notación aparecen en diferentes diagramas y son utilizados por diferentes métodos para expresar su diseño. Por otro lado Java es un lenguaje de programación orientado a objetos, el cual ha crecido notablemente para el diseño de aplicaciones y programas.

Debido a que UML es un lenguaje de modelado y Java es un lenguaje de programación orientado a objetos, existe un mapeo significativo cuando se convierten los modelos en UML a aplicaciones Java. Al proceso de comenzar a generar código a través de modelos se le conoce como ingeniería directa, y es posible realizarlo con UML y Java.

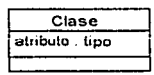
Diagrama de clases

Clases, atributos y operaciones

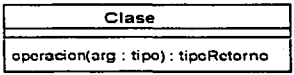
Un diagrama de clases muestra la existencia de clases y como están relacionadas entre sí. Un diagrama de clases consiste de clases y relaciones. En UML una clase se representa como un rectángulo con tres compartimentos, en el primer compartimento va el nombre de la clase; el segundo compartimento corresponde a los atributos de la clase, los cuales equivalen a variables en Java; y en el tercero, las operaciones de la clase, estas en Java se mapean a métodos




```
class Clase {

}
```




```
class Clase {

  tipo atributo;
}
```




```
class Clase {

  tipoRetorno operacion(tipo arg) {
  }
}
```

Relaciones

Una relación es una conexión entre elementos. Comprender el significado de las relaciones en un modelo es el primer paso para entender como se implementarán en código Java. Los tipos de relaciones son los siguientes:

Generalización. Es un tipo de relación entre una clase general conocida como superclase o clase padre y una clase más especifica de ese elemento llamada subclase o clase hija. En Java la subclase va acompañada de la cláusula "*extends*".

Dependencia. Es una relación que traduce algún tipo de referencia a una clase u objeto que no existe dentro del contexto.

Realización. Es una relación entre dos entidades, en la cual una entidad especifica un contrato que otra entidad garantiza que cumplirá. Al codificar, a la declaración de la clase se le agrega la cláusula "*implements*", seguida por el nombre de una o varias interfaces.

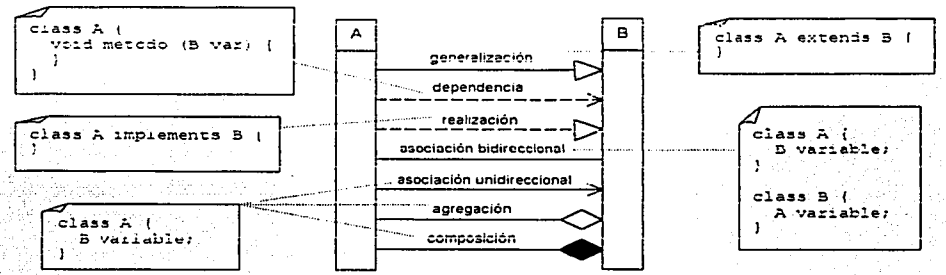
Asociación. Es una relación entre objetos de la misma clase o diferentes clases. Existen dos tipos de relaciones de asociación:

Unidireccional. Es la relación que especifica la navegación en un solo sentido entre las clases conectadas. La clase hacia donde se dirige la navegación de la relación se convierte en una variable de instancia de la clase que da origen a la relación.

Bidireccional. Es una asociación que implica la navegación en ambos sentidos entre las clases conectadas. Corresponde a dos asociaciones unidireccionales en sentidos opuestos.

Agregación. Es una forma de asociación que representa una relación "*es parte de*" entre dos clases, donde una clase es el "*todo*" y otra la "*parte*". No tiene significado, por lo que al codificar es igual que una asociación unidireccional.

Composición. Es una forma de agregación más fuerte. Por otro lado, la "*parte*" no necesita ser destruida cuando el "*todo*" es destruido, pero el "*todo*" es responsable de mantener viva a la "*parte*" o destruirla. Equivale en código a una asociación unidireccional.



Diagramas de interacción

Los diagramas de interacción muestran la interacción entre instancias de objetos. Existen dos tipos de diagramas de interacción:

1. ***Diagramas de secuencia.*** Muestran los objetos participantes en la interacción por sus líneas de vida y los mensajes que ellos intercambian arreglados en una secuencia de tiempo.

2. ***Diagramas de colaboración.*** Representa las interacciones entre los objetos, este tipo de diagramas están organizados a través de objetos con ligas hacia algún otro objeto.

Tanto los diagramas de secuencia y de colaboración representan las interacciones entre objetos, y se envían y reciben mensajes. Los mensajes se codifican a Java como métodos.

Diagrama de secuencia

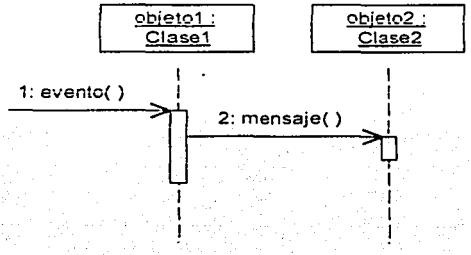
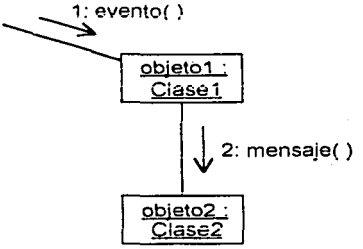


Diagrama de colaboración




```
public class Clase1 {
  Clase2 objeto2;
  public void evento() {
    objeto2.mensaje();
  }
}
```

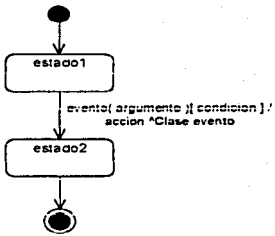
Diagramas de estados

La idea básica en un diagrama de estados de una clase, es crear una variable de instancia en donde se guarde el registro del estado actual. En la definición de la clase se inicializa el estado en el constructor, según el estado del objeto al crearse.

La información proporcionada por los diagramas de estados debe ser reflejada en los métodos de la clase, los eventos se implementan como métodos con sus respectivos argumentos.

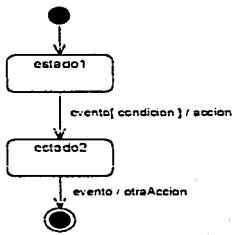
En el caso en el que el evento tenga una condición, esta se implementa con una sentencia "*if*" dentro del evento y se asigna el estado que adoptaría el objeto a la variable.

Los eventos enviados a los objetos de la clase producen una reacción dependiendo del estado actual.




```
class Clase{

  String estado;

  Clase() {
    estado = "estado1";
  }

  void evento (argumento) {
    if (condicion) {
      estado = "estado2";
      accion;
      Clase.evento();
    }
  }
}
```

Las transiciones entre estados con el mismo evento (operación a realizar), al codificarse estos eventos en Java, se utiliza la sentencia "*switch*", dentro de él se realizarán las condiciones y acciones según el estado del objeto.




```
class Clase{

  public void evento() {

    switch (estado) {

      case estado1:
        if (condicion) {
          estado = "estado2";
          accion1;
        }
        break;

      case estado2:
        accion2;
        break;
    }
  }
}
```

Plan de estudios

DESARROLLO DE SOFTWARE ORIENTADO A OBJETOS CON UML Y JAVA

CLAVE:
PLAN:
LICENCIATURA(SEMESTRE): INFORMÁTICA
DEPTO. ACADÉMICO: INFORMÁTICA AVANZADA
ÁREA: SISTEMAS
REQUISITOS:

CRÉDITOS: 8
HORAS POR CLASE: 2
CLASES POR SEMANA: 2
HORAS POR SEMESTRE: 68

OBJETIVO GENERAL: AL FINALIZAR EL CURSO EL ALUMNO SERÁ CAPAZ DE CONOCER EL PARADIGMA DE ORIENTACIÓN A OBJETOS Y PONER EN PRÁCTICA LA INGENIERÍA DE SOFTWARE ORIENTADA A OBJETOS CON UML Y JAVA EN UN MODELO DE DESARROLLO.

HORAS	TEMÁTICA	OBJETIVOS EDUCACIONALES	SUGERENCIAS DIDÁCTICAS	REFERENCIAS BIBLIOGRÁFICAS
10	I. ORIENTACIÓN A OBJETOS 1. Construcción y mantenimiento de sistemas 2 Requerimientos y características de los desarrollos actuales 3. Crisis del software. 4 Enfoque de la metodología de objetos. A) OMT - James Rumbaugh B) Booch - Grady Booch C) OOSE - Ivar Jacobson D) RUP Proceso de Desarrollo Unificado 5 Paradigma orientado a objetos 6. Principios y conceptos de la orientación a objetos. A) Clases y objetos B) Estructura y encapsulamiento C) Mensajes y métodos D) Jerarquía de clases E) Herencia F) Polimorfismo	Dar a conocer el desarrollo de software en la actualidad Evaluar la problemática de la crisis del software. Difundir los métodos orientados a objetos más importantes. Presentar los elementos primordiales de la orientación a objetos.	Exposición audiovisual. Trabajos de revisión bibliográfica de libros y publicaciones no referidos en esta unidad.	1, 2, 3

30	II MODELADO ORIENTADO A OBJETOS CON UML 1. Evolución del UML 2. Modelado de entidades 3. Modelado de diagramas A) Casos de uso B) Actividad C) Clases D) Interacción a) Secuencia b) Colaboración E) Estados F) Diseño G) Implementación H) Procesos I) Despliegue	Identificar conceptos y principios de orientación a objetos en UML Elaborar y ejemplificar diagramas en UML	Exposición audiovisual. Trabajos de revisión bibliográfica de libros y publicaciones no referidos en esta unidad. Ejercicios en clase propuestos por el profesor y por el alumno	4, 5, 6
28	III LA TECNOLOGÍA JAVA 1. El lenguaje de programación Java 2 Tipos de programas en Java 3 Compilación y ejecución de programas 4 Creación de aplicaciones con el JDK 5 Estructura del lenguaje A) Estructura básica de un programa B) Identificadores C) Variables primitivas y tipos de datos D) Expresiones y operadores E) Palabras clave F) Bloques y sentencias G) Comentarios 6 Objetos en Java A) Definición de clases B) Creación de objetos C) Variables y Métodos D) Metodos sobrecargados E) Modificadores de acceso F) Implementando la herencia G) Polimorfismo H) Clases abstractas I) Interfaces J) Manejo de paquetes	Presentar el entorno de programación Java Introducir las bases y elementos del lenguaje Java Representación y programación de conceptos y principios en Java.	Exposición audiovisual. Prácticas de laboratorio de cómputo Ejercicios en clase	7, 8

BIBLIOGRAFÍA:

BÁSICA

1. BOCCH, Grady, **Análisis y diseño orientado a objetos,** Addison-Wesley
2. MARTIN, James, ODELL James J., **Análisis y diseño orientado a objetos,** Prentice Hall
3. JACOBSON, Ivar, BOOCH, Grady, RUMBAUGH, James, **El Proceso de desarrollo unificado de software,** Adisson-Wesley
4. JACOBSON, Ivar, BOOCH, Grady, RUMBAUGH, James, **El Lenguaje de Modelado Unificado,** Adisson-Wesley
5. JACOBSON, Ivar, BOOCH, Grady, RUMBAUGH, James, **El Lenguaje de Modelado Unificado. Manual de referencia,** Addison-Wesley
6. LARMAN, Craig, **UML y patrones: Introducción al análisis y diseño orientado a objetos**, Addison-Wesley Longman
7. ECKEL, Bruce, **Thinking in Java**, Prentice Hall
8. JAWORSKI, Jamie, **Java 1.2 al descubierto**, Prentice Hall

Ofícios

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

L.A. GUSTAVO ALMAGUER PÉREZ
SECRETARIO DE DIVULGACIÓN Y FOMENTO EDITORIAL
FCA, UNAM

P R E S E N T E

Por este medio, me permito hacerle de su conocimiento que tras haber realizado el desarrollo e investigación del tema de tesis "Generación de código Java a partir de diseños en UML", se elaboró el artículo "Mapeo de diseños de UML a Java", el cual contiene los aspectos esenciales que resultaron de mi investigación.

Por tal motivo, es grato presentarle el artículo para su posible publicación, todo ello como parte de una serie de actividades para difundir la información dentro de la Facultad de Contaduría y Administración, UNAM.

A T E N T A M E N T E
México, D.F. a 28 de Agosto de 2002.

Jesús Cuauhtémoc García Flores
Estudiante de Informática, UNAM
Tel. Particular 5643 3695

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

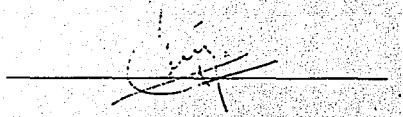
M. en I. GRACIELA BRIBIESCA CORREA
JEFE DE LA DIVISIÓN DE INFORMÁTICA

P R E S E N T E

Por este medio, me es grato presentarle una propuesta de temas para la materia de "Desarrollo de software orientado a objetos con UML y Java", con la finalidad de poder incluirla como materia optativa al plan de estudios de la carrera de informática. Todo ello tras haber realizado el desarrollo e investigación para el tema de tesis "Generación de código Java a partir de diseños en UML".

Por otro lado, también me pongo a su disposición para la presentación del tema de investigación (tesis) ante grupos de informática de la Facultad de Contaduría y Administración, UNAM

A T E N T A M E N T E
México, D.F. a 28 de Agosto de 2002.



Jesús Cuauhtémoc García Flores
Estudiante de Informática, UNAM
Tel. Particular 5643 3695

GLOSARIO

8. GLOSARIO

abstracción	Mecanismo que denota las características esenciales de un objeto que lo distinguen de los demás.
acción	Es una operación instantánea que va asociada a un suceso; las acciones también pueden representar operaciones internas de control, tales como dar un valor a un atributo o generar otros sucesos, estas acciones son mecanismos para estructurar el control dentro de una implementación.
actor	Cosa o persona que interactúa con el sistema. Un actor forma parte de un diagrama de casos de uso.
Ada	Lenguaje de programación con una estructura de bloque y un mecanismos de tipo de datos con extensiones para aplicaciones de tiempo real y distribuidas.
agregación	Es un tipo de relación en la que se describen los componentes que forman a un todo.
análisis	Identificación de los requerimientos del usuario.
análisis y diseño orientado a objetos / ADOO	Consiste en situar el estado actual de un problema y su solución lógica dentro de la perspectiva de los objetos, con la finalidad de tener el dominio sobre ese problema.
análisis orientado a objetos / AOO	Etapa del desarrollo de software orientado a objetos en la que se identifica y describe el problema en la perspectiva de objetos. Se describen los requerimientos del sistema y los casos de uso, se definen los conceptos del sistema utilizados por los usuarios y expertos.
analista	Es el profesional capacitado para diseñar, detectar y analizar sistemas de información, así como el investigar y desarrollar técnicas específicas para el análisis de sistemas de información.
aplicación	Un programa escrito en Java para correr en un interprete, tal como el JDK.
applet	Programa escrito en lenguaje Java, que se ejecuta en cualquier ordenador que tenga un navegador compatible con Java.

argumento	Un dato especificado en la llamada a un método. Un argumento puede ser un valor literal, una variable o una expresión.
arquitectura / arquitectura del sistema	Colección de componentes computacionales junto con una descripción de las interacciones entre estos componentes
arreglo	Colección secuencial de elementos de tipos idénticos, en la cual cada posición de los elementos se le designa un único índice.
artefacto	Información que se utiliza o produce mediante un proceso de desarrollo de software.
asociación	Ruta de comunicación entre objetos conectados.
asociación bidireccional	Es una asociación que implica la navegación en ambos sentidos entre las clases conectadas.
asociaciones múltiples	Situación que se da cuando existe más de una relación entre dos clases.
asociación reflexiva	Es una asociación dirigida hacia la misma clase.
asociación unidireccional	Es el tipo de asociación que especifica la navegación en un solo sentido entre las clases conectadas.
atributo / campo	Un miembro dato o característica de una clase. Un atributo no es estático.
AWT / Abstract Window Toolkit	Se trata de una biblioteca de clases Java para el desarrollo de Interfaces de Usuario Gráficas. AWT permite hacer interfaces gráficas mediante artefactos de interacción con el usuario, como botones, menús, texto, botones para selección, barras de deslizamiento, ventanas de diálogo, selectores de archivos, etc.
browser / navegador	Es un programa que actúa como una interfaz entre el usuario y los contenidos de Internet, específicamente la web. Los navegadores también se conocen como clientes web, o clientes universales.
C++	Es un lenguaje de programación orientado a objetos diseñado partiendo del lenguaje C.

caso de uso	Es una función proporcionada por el sistema, y será documentado con su descripción, flujo principal y flujos alternativos.
ciclo de vida de un objeto	Es el tiempo de existencia de un objeto y sus etapas son: cuando se crea, cuando envía mensajes y cuando se elimina.
clase	Plantilla o molde a partir de las cuales se pueden crear objetos con el mismo conjunto de atributos y el mismo comportamiento. Es decir, una clase es una descripción de un conjunto de objetos.
clase abstracta	Una clase que contiene uno o más métodos abstractos y no puede ser instanciado. Otras clases pueden extender de ella e implementar los métodos abstractos.
clase singleton	Clase de la que puede haber una sola instancia.
clase contenedora	Clase en la que una de sus instancias contiene un conjunto de objetos.
clasificador	Describe características estructurales y de comportamiento
colaboración	Dos o más objetos que participan entre sí con la finalidad de prestar un servicio.
COM / Component Object Model	Proporciona un juego de interfaces que permiten a los clientes y servidores comunicarse dentro del mismo ordenador.
compilador	El compilador es un programa que nos permite traducir código fuente en un programa para ser ejecutado en una computadora.
comportamiento	Como un objeto actúa y cual es su reacción hacia otros objetos.
composición	Es un tipo de relación en donde se organizan varios objetos en uno general.
condición	Es una expresión boleana de un atributo que tiene un valor.
constructor	Un método que crea un objeto. En Java, los constructores son métodos de instancia con el mismo nombre que el de la clase.
constante	Variable donde su valor se inicializa cuando se declara.

CORBA / Common Object Request Broker Architecture	Modelo industrial standard para objetos distribuidos.
DCOM / Distributed Component Object Model	Conceptos e interfaces de programa de Microsoft en el cual los objetos de programa del cliente pueden solicitar servicios de objetos de programa servidores en otros ordenadores dentro de una red.
dato	Números, caracteres, imágenes u otro método de registro, el cual puede ser manipulado por el hombre y en especial como entrada hacia una computadora, para ser almacenada y procesada. Un dato por si solo no dice nada sobre el porqué de las cosas, y por sí mismo tiene poca o ninguna relevancia o propósito.
dependencia	Es un tipo de relación en la que hay algún tipo de referencia a una clase u objeto que no existe dentro del contexto.
diagrama	Es una representación gráfica de una colección de elementos del modelo.
diagrama de actividad	Representan el flujo para una operación del sistema.
diagrama de casos de uso	Representa el comportamiento y funcionamiento del sistema a los usuarios y/o clientes.
diagrama de clases	Un modelo que muestra la existencia de clases y como están relacionadas entre sí.
diagrama de colaboración	Representación de la secuencia de los mensajes que implanta una operación o transacción.
diagrama de componentes	Muestra las relaciones entre paquetes de componentes y componentes.
diagrama de despliegue	Muestra como se asignan los procesos en el hardware.
diagrama de estados	Muestra el espacio del estado de una clase, los sucesos que origina la transición de un estado a otro, y las acciones que resultan de un cambio de estado.
diagrama de secuencia	Representación que se centra en la ejecución de una interacción en el tiempo.

diseño	Desarrollo de una solución de software para las necesidades del usuario.
diseño orientado a objetos / DOO	Solución a un problema dentro de la perspectiva de objetos en la ingeniería de software. Se realiza la arquitectura del sistema, se definen las interfaces de usuario, los objetos y la base de datos
distribuido	El desarrollo de aplicaciones autónomas que residan en diferentes computadoras en una red.
Eiffel	Eiffel es un avanzado lenguaje orientado a objetos, creado a mediados de los años 80 por Bertrand Meyer, de la compañía Interactive Software Engineering.
encapsulación	Es el ocultamiento del funcionamiento interno de un objeto, esto permite que sólo la interfaz sea percibida y no la implementación.
enlace / liga	Ruta de comunicación entre los objetos conectados.
escenario	Es una secuencia de sucesos que se producen durante una ejecución completa de un sistema, el ambiente puede incluir a todos los sucesos o solo a aquellos que afecten a algunos objetos del sistema.
especialización	Consiste en crear subclases que especialicen a una superclase.
estado	Posible condición en la que se encuentra un objeto.
estereotipo	Es un tipo de elemento.
evento	Es una ocurrencia que pasa en algún punto del tiempo
excepción	Un evento durante el programa en ejecución que prevé que el programa continúe de manera normal; generalmente es un error.
firma	Consiste de el nombre del método y el número y tipos de parámetros de el método.
flujo principal	Conjunto de actividades básicas dentro de un caso de uso.
flujo alternativo	Secuencia de actividades alternas que se pueden realizar dentro de una descripción de un caso de uso.

framework	Programa reusable muy complejo comprendido por muchos objetos que proporcionan un servicio a la aplicación.
generalización	Identificar características y comportamiento comunes entre clases.
herencia	Es cuando una clase comparte la estructura y comportamiento definidos a una o a más clases.
herencia múltiple	Permite que una clase tenga más de una superclase y que herede características de todas ellas, esto permite mezclar información procedente de dos o mas fuentes, una clase con más de una superclase se denomina clase unión.
herencia simple	Es cuando una clase hereda de una sola superclase su estructura y comportamiento.
herramientas CASE / CASE	Conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en alguna de sus fases.
HotJava	Navegador para Java
HTML	Archivo, para documentos de texto en internet. Es muy simple y aloja imágenes, sonidos, video, formas y testo.
identidad	Característica de los objetos que consiste en que cada objeto es único.
identificador	Palabras que el programador puede definir dentro del lenguaje para nombrar variables, métodos, clases y objetos.
implementación / construcción / codificación / programación	Fase del ciclo de desarrollo de software donde se escribe el código del software. Se comienza con la programación del sistema, por medio de los diagramas.
indicador de multiplicidad	Número de objetos de una clase que hacen referencia a un objeto de otra clase.
información	Conjunto de datos que han sido procesados por algún sistema de procesamiento.

ingeniería de software	Es el tratamiento sistemático de todas las fases del ciclo de vida del software, abordando el desarrollo de sistemas de información de forma similar a los proyectos de ingeniería. Esto implica la identificación de las tareas a realizar (establecidas según una metodología de desarrollo), de los productos a obtener y de las técnicas y herramientas a utilizar.
interfaz	Grupo de métodos que pueden ser implementados por varias clases.
interfaz gráfica de usuario / GUI	Se refiere a las técnicas correspondientes al uso de gráficos que interactúan con el teclado y mouse, y dan una interfaz de fácil uso al programa.
internet	Red que consiste de una serie de computadoras interconectadas de muchas organizaciones y países alrededor del mundo.
Java	Java es un lenguaje de programación orientación a objetos para diseñar aplicaciones y programas.
lenguajes de alto nivel	Lenguajes diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Un programa escrito en lenguaje de alto nivel es independiente de la máquina (las instrucciones no dependen del diseño del hardware o de una computadora en particular), por lo que estos programas son portables o transportables. Los programas escritos en lenguaje de alto nivel pueden ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras.
Lenguaje de Modelado Unificado / Unified Modeling Language / UML	Lenguaje estándar para especificar, construir, visualizar y documentar los artefactos de un sistema de software orientado a objetos.
mantenimiento	Proceso utilizado para detectar y solucionar errores en el software.
mapeo	La programación de la aplicación debe ser compatible con los elementos generados en el análisis y diseño del sistema.
mensaje	Solicitud que se le envía a un objeto para que lleve a cabo una acción de la cual puede regresar una respuesta.

método / operación	Comportamiento que tiene un objeto dependiendo del valor que presenten sus datos y de las operaciones que se realicen sobre esos datos.
método abstracto	Un método que no tiene implementación.
método de clase / método estático	Un método que es invocado sin referencia a un objeto en particular. Los métodos de clase afectan a la clase como un todo, y no a una instancia en particular de la clase.
método de instancia	Cualquier método que es invocado con respecto a una instancia de una clase.
método orientado a objetos / metodología orientada a objetos	Conjunto de técnicas, herramientas y procesos que se basan en modelar los sistemas con el paradigma orientado a objetos durante el proceso de software.
miembro / miembro de clase	Un campo o método de una clase. A menos que se especifique otra cosa, un miembro no es estático.
modelo	Abstracción de algo que se elabora para comprender ese algo antes de construirlo.
modelo conceptual	Conjunto de conceptos significativos en un dominio del problema, identificando los atributos y las asociaciones, y es la herramienta más importante del análisis orientado a objetos.
Mosaic	Un programa que proporciona una simple GUI que habilita el fácil acceso a los datos almacenados en internet. Estos datos pueden ser simples archivos o documentos de texto.
multiplicidad	Indica el número de relaciones entre las instancias.
notación	Medio para capturar la forma de funcionar, el comportamiento y la estructura de un sistema.
objeto / instancia / instancia de clase	Elemento o cosa independiente que forma parte de nuestro mundo, tiene un conjunto de características propias y realiza una serie de tareas específicas. Un objeto puede ser real o abstracto.
objetos distribuidos	Conjunto de objetos que interactúan sobre diferentes plataformas a través de una red.

orientación a objetos / OO	Consiste en representar la realidad de manera más natural, lo cual facilita las actividades de análisis y diseño de sistemas, así como su programación.
paquete	Es un mecanismo de agrupación de paquetes, clases y/o interfaces.
paradigma	Conjunto de técnicas utilizadas para explicar o aclarar algo.
patrón	Es una descripción esquemática de soluciones a problemas comunes dentro del diseño de software.
portable	Es la facilidad de transferencia de productos de software entre varias plataformas de hardware y software.
polimorfismo	Habilidad de diferentes objetos de responder cada uno de su propia forma a mensajes idénticos; es decir, cada patrón representa una solución genérica a un problema
proceso / proceso de desarrollo/ proceso de software / desarrollo de software	Conjunto de actividades y resultados asociados a un producto de software.
programa	Es el conjunto de instrucciones que permiten realizar una tarea o tareas, para la computadora se escriben utilizando lenguajes especiales, especificando las operaciones que debe contener y el orden de ejecución.
programación orientada a objetos / POO	Es un paradigma de programación que se basa en la construcción de sistemas de software en módulos obtenidos a partir de los objetos que manipula.
realización	Tipo de relación en la cual una entidad especifica un contrato que otra entidad garantiza que cumplirá.
referencia	Un tipo de dato el cual su valor es una dirección de memoria.
relación	Es una conexión entre elementos que ayudan a definir como estos elementos interactúan con algún otro.
responsabilidad	Es un servicio que ofrece un objeto dentro del sistema.

reutilizar	Es la habilidad de utilizar elementos de software para construir diferentes aplicaciones.
RIM / Remote Method Invocation	Arquitectura que soporta objetos distribuidos escritos enteramente en Java.
robusto	Habilidad de un sistema de software de reaccionar apropiadamente ante condiciones anormales.
rol	Es un nombre que va a identificar a un objeto de la clase al relacionarse con otra clase.
sistema de información	Conjunto de elementos físicos, lógicos, de comunicación, datos y personal que, interrelacionados, permiten el almacenamiento, transmisión y proceso de la información.
sistema de software / software / producto de software	Es todo aquello que no es físico. Es el conjunto de programas que utiliza la computadora.
sistema de software orientado a objetos / sistema orientado a objetos	Conjunto de objetos de diferentes clases que interactúan entre sí utilizando métodos o servicios, todo ello con el propósito de que opere un producto de software.
sobreescritura	Método de una subclase que tiene la misma firma y tipo de retorno que la clase que originalmente definió al método, pero se le proporciona diferente implementación.
subclase / clase derivada / clase hija	Una clase que recibe la estructura y comportamiento de una clase en particular.
superclase / clase base / clase padre	Una clase que proporciona su estructura y comportamiento de otra clase.
transición	Cambio entre un estado y otro
variable	Nombre de un identificador que hace referencia a una localidad en memoria donde puede ser almacenado un valor.
variable de clase / variable estática	Un dato asociado a una clase en particular como un todo, y aplica a todas las instancias de la clase.

variable de instancia	Cualquier dato que es asociado con un objeto en particular. Cada instancia de una clase tiene su propia copia de variables de instancia definidas en la clase.
visibilidad / accesibilidad	Mecanismo que permite el control de acceso previniendo que se conozcan los detalles de implementación de una clase o paquete.
vista	Combinación de diagramas que permiten comunicar el sistema desde una perspectiva.
vista de casos de uso	Consiste en conocer el entono y entendimiento del sistema, se representa con los modelos de casos de uso.
vista de diseño	representa la funcionalidad del sistema, esto es a través del diagrama de clases.
vista de implementación	Se basa en la organización modular del software dentro del ambiente de desarrollo y es a través del diagrama de componentes.
vista de procesos	Muestra la estructura del sistema al momento de ejecutarse.
vista de despliegue	Muestra la configuración del hardware del sistema, la representan los diagramas de desarrollo.

Siglas y acrónimos

ADOO	Análisis y Diseño Orientado a Objetos
AOO	Análisis Orientado a Objetos
AWT	Abstract Window Toolkit
CASE	Computer Aided Software Engineering
CLOS	Common Lisp Object System
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DOO	Diseño Orientado a Objetos
GUI	Graphical User Interface
HTML	HyperText Markup Language
JDK	Java Development Kit
OMG	Object Management Group
OMT	Object Modeling Tecnique
OO	Orientación a Objetos
OOSE	Object Oriented Software Engineering
POO	Programación Orientada a Objetos
RIM	Remote Method Invocation
RUP	Rational Unified Process
TCP/IP	Transfer Control Protocol/Internet Protocol
UML	Unified Modeling Language

BIBLIOGRAFÍA

9. BIBLIOGRAFÍA

Libros

The Essence of Object-Oriented Programming with Java and UML
Bruce E. Wampler, Ph.D
Addison-Wesley
Primera edición
ISBN 0-201-73410-9

Object-Oriented Software Development in Java: Principles, Patterns, and Frameworks
Xiaoping Jia
Addison-Wesley
Primera edición
ISBN 0-201-35084-X

Object Oriented Software Design & Construction with Java
Dennis Kafura
Prentice Hall
Primera edición
ISBN 0-13-011264-X

El Lenguaje de Modelado Unificado
Ivar Jacobson, Grady Booch y James Rumbaugh
Addison-Wesley
Primera edición
84-7829-028-1

Visual Modeling with Rational Rose 2000 and UML
Terry Quatrani
Addison-Wesley
Primera edición
ISBN 0-201-69961-3

Análisis y Diseño Orientado a Objetos
Grady Booch
Addison-Wesley
Segunda edición
ISBN 9-684-44352-8

El Proceso Unificado de Desarrollo de Software
Ivar Jacobson, Grady Booch y James Rumbaugh
Ed. Addison-Wesley
Primera edición
ISBN 0-201-57169-2

Estadística matemática con aplicaciones
William Mendenhall, Dennis D. Wackely, Richard L. Scheafer
Iberoamérica
Segunda edición
ISBN 970-625-016-6

Tesis

Documentación del análisis y diseño de sistemas orientados a objetos con UML
Jiménez Herrada, Jenny
2001
Informática
Facultad de Contaduría y Administración, UNAM
001-00623-J1-2001

El software orientado a objetos
Preciado López, José Guillermo
1996
Ciencias de la Computación
Facultad de Ciencias, UNAM
001-00321-P4-1996

Una aplicación de la metodología OMT para diseño orientado a objetos
Oliver Suárez, Isabel Laura
1998
Ciencias de la Computación
Facultad de Ciencias, UNAM
001-00321-O2-1998

RUP y UML como elementos clave en el desarrollo de sistemas orientados a objetos

2002
Informática
Facultad de Contaduría y Administración, UNAM

Revistas

JavaPro - Guide to Middleware
209 Hamilton Ave., Palo Alto, CA 94301
Octubre
Vol. 3, No.: 13

JavaPro
209 Hamilton Ave., Palo Alto, CA 94301
Junio/Julio
Vol. 2, No.: 3

EETimes (Electronic Engineering Times)
600 Community Drive Manhasset, NY 11030

Using UML to drive Java can alleviate chaos
Bill Graham
1 de Abril de 2002

Artículos

Mapping UML desings to Java
William Harrison, Charles Barton, Mukund Raghavachari
IBM

A Quick Introduction to UML and Java
Rob McGovern
1 de Febrero de 2002

An overwiew of object relationships / The basics of UML and Java associations
Scott W. Ambler
IBM - DeveloperWorks
8 de Febrero de 2001

Implementing object relationship with a singular multiplicity
Scott W. Ambler
IBM - DeveloperWorks
2 de Marzo de 2001

The fundamentals of one-to-many bidireccional associations
Scott W. Ambler
IBM - DeveloperWorks
8 de Marzo de 2001

Using collections to manage many-to-many object relationship
Scott W. Ambler
IBM - DeveloperWorks
15 de Marzo de 2001

URL's

http://ami.ics.hawaii.edu/courses/665/notes/mapping.html
http://ami.ics.hawaii.edu/courses/665/notes/uml_class.html
http://ami.ics.hawaii.edu/courses/665/notes/subclassing_composition.html
http://www.kirkk.com/ReferenceCard.html
http://www.holub.com/class/uml/uml.html
http://cannes.rhon.itam.mx/Alfredo/Espaniol/Publicaciones/IngSW.htm
http://spider.ipac.caltech.edu/staff/jchavez/public/talks/tcc.html
http://ugweb.cs.ualberta.ca/~c301/Fall2001/