

Modelado de Sistemas Orientados a Objetos

Guía didáctica

Unidad Académica Técnica y Tecnológica

Tecnología Superior en Transformación Digital de Empresas

Modelado de Sistemas Orientados a Objetos

Guía didáctica

Carrera	PAO Nivel
▪ <i>Tecnología Superior en Transformación Digital de Empresas</i>	II

Autores:

Jaramillo Hurtado Danilo Rubén
Sucunuta Manuel Eduardo



D S O F _ 1 0 7 5

Asesoría virtual
www.utpl.edu.ec

Universidad Técnica Particular de Loja

Modelado de Sistemas Orientados a Objetos

Guía didáctica

Jaramillo Hurtado Danilo Rubén

Sucunuta Manuel Eduardo

Diagramación y diseño digital:

Ediloja Cía. Ltda.

Telefax: 593-7-2611418.

San Cayetano Alto s/n.

www.ediloja.com.ec

edilojacialtda@ediloja.com.ec

Loja-Ecuador

ISBN digital - 978-9942-39-617-4



Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0)

Usted acepta y acuerda estar obligado por los términos y condiciones de esta Licencia, por lo que, si existe el incumplimiento de algunas de estas condiciones, no se autoriza el uso de ningún contenido.

Los contenidos de este trabajo están sujetos a una licencia internacional Creative Commons **Reconocimiento-NoComercial-CompartirIgual 4.0 (CC BY-NC-SA 4.0)**. Usted es libre de **Compartir – copiar y redistribuir el material en cualquier medio o formato. Adaptar – remezclar, transformar y construir a partir del material citando la fuente, bajo los siguientes términos: Reconocimiento- debe dar crédito de manera adecuada, brindar un enlace a la licencia, e indicar si se han realizado cambios.** Puede hacerlo en cualquier forma razonable, pero no de forma tal que sugiera que usted o su uso tienen el apoyo de la licenciante. **No Comercial-no puede hacer uso del material con propósitos comerciales. Compartir igual-Si remezcla, transforma o crea a partir del material, debe distribuir su contribución bajo la misma licencia del original.** No puede aplicar términos legales ni medidas tecnológicas que restrinjan legalmente a otras a hacer cualquier uso permitido por la licencia. <https://creativecommons.org/licenses/by-nc-sa/4.0/>

Índice

1. Datos de información	7
1.1. Presentación de la asignatura	7
1.2. Competencias genéricas de la UTPL.....	7
1.3. Competencias específicas de la carrera	7
2. Metodología de aprendizaje	8
3. Orientaciones didácticas por resultados de aprendizaje	10
 Primer bimestre.....	 10
 Resultado de aprendizaje 1.....	 10
 Semana 1	 10
 Unidad 1. Fundamentos de modelado.....	 10
1.1. Modelado.....	11
Actividad de aprendizaje recomendada.....	13
1.2. Importancia del modelado	14
Actividad de aprendizaje recomendada.....	14
Actividad de aprendizaje recomendada.....	16
1.3. Principios de modelado.....	17
Actividad de aprendizaje recomendada.....	17
 Semana 2	 17
1.4. Modelado orientado a objetos	17
Actividad de aprendizaje recomendada.....	19
Actividad de aprendizaje recomendada.....	19
 Semana 3	 27
1.5. Lenguaje Unificado de Modelado (UML).....	27
Actividad de aprendizaje recomendada.....	35
 Semana 4	 36
1.6. Desarrollo del proyecto software con UML.....	36

Resultado de aprendizaje 2.....	43
Semana 5	43
 Unidad 2. Modelado de Sistemas.....	43
2.1. Modelo “4+1”. Vistas de Kruchten.....	43
2.2. UML – Kruchten.....	44
2.3. Vista de escenarios	45
Actividad de aprendizaje recomendada.....	49
 Semana 6	50
2.4. Desarrollo del diagrama de caso de uso – Caso práctico	50
 Semana 7	52
2.5. Vista lógica.....	52
Actividad de aprendizaje recomendada.....	53
 Semana 8	59
Segundo bimestre	60
Resultado de aprendizaje 2.....	60
 Semana 9	60
2.6. Diagrama de comunicación	61
 Semana 10	64
2.7. Vista de procesos	64
 Semana 11	68
 Semana 12	72
2.8. Vista de despliegue	72
2.9. Vista física.....	77

Resultado de aprendizaje 3.....	81
Semana 13	81
 Unidad 3. Del modelo a la implementación.....	81
3.1. Generación de código a partir de los modelos	82
 Semana 14	88
3.2. Ambiente de desarrollo	88
 Semana 15	89
3.3. Implementación y pruebas.....	89
 Semana 16	90
4. Glosario.....	91
5. Recursos	92
6. Referencias bibliográficas	94
7. Anexos	95



1. Datos de información

1.1. Presentación de la asignatura



1.2. Competencias genéricas de la UTPL

- Compromiso e implicación social

1.3. Competencias específicas de la carrera

- Desarrolla aplicaciones empresariales aplicando enfoques centrados en la nube.
- Diseña modelos arquitectónicos de empresa para gestionar el alineamiento estratégico entre negocio y TI.



2. Metodología de aprendizaje

Para una adecuada participación de los estudiantes en las diferentes actividades planificadas para su aprendizaje, la presente asignatura utilizará una estrategia de aprendizaje basado en el estudio de casos. Esto permitirá al estudiante relacionar los conocimientos teóricos de manera práctica. Para esto se ha desarrollado el “Caso de estudio”, donde se plantean ejemplos que serán explicados por el tutor y luego el estudiante desarrolle las actividades que se plantean por cada uno de los componentes.

La descripción del caso sugiere realizar lo siguiente:

- Presentación preliminar del caso a los estudiantes a través de la tutoría virtual.
- Recopilación de opiniones, impresiones, juicios, posibles alternativas, por parte de los estudiantes para determinar el grado de comprensión del caso.
- Análisis de los aportes de los estudiantes para aclarar las incoherencias y ajustar la interpretación del caso.
- Conceptualización de los temas que requiere el caso, esto se realiza conforme se cubren los temas planificados en la presente asignatura.
- Desarrollo de cada modelo que se indica en el caso y que el tutor planifica semanalmente.

Esta estrategia de trabajo permite una formación teórico-práctica de los estudiantes logrando en ellos lo siguiente:

- Desarrollo de habilidades cognitivas como pensamiento crítico, análisis, y propuesta de soluciones.
- Aprendizaje de conceptos y aplicación de los mismos en casos, tanto de manera sistemática como por el desarrollo de escenarios reales.

- Habilidad para trabajar en grupo y la interacción con otros estudiantes, así como la actitud de cooperación, el intercambio y la flexibilidad, lo cual constituye una preparación eficaz el trabajo en equipo.
- La motivación por el aprendizaje, ya que los estudiantes por lo general encuentran el trabajo práctico más interesante que las lecciones magistrales y la lectura de libros de texto.



3. Orientaciones didácticas por resultados de aprendizaje



Primer bimestre

Resultado de aprendizaje 1

- Interpreta los principios del modelado de sistemas.

Contenidos, recursos y actividades de aprendizaje



Semana 1

Unidad 1. Fundamentos de modelado



1.1. Modelado

Empecemos el estudio analizando los conceptos sobre modelado desde una perspectiva general, para luego enfocarnos en los elementos esenciales del desarrollo del *software*.

El modelado no es tema nuevo, sino que es algo que se viene utilizando y desarrollando en distintas disciplinas. Consideremos el caso de la construcción de una casa, el arquitecto desarrolla los planos y los lleva al dueño para una revisión y poder ver si es lo que el dueño realmente necesita. Estos planos son el reflejo de una posible realidad, ya que permitirá tener varios puntos de vista, es decir, cómo está la fachada, la distribución de los espacios, donde estarán los baños, de manera tal que pueda tener una visión general de la casa. Dependiendo de esto, el cliente podrá solicitar rectificaciones hasta llegar a una aceptación para empezar la construcción. En el ámbito del desarrollo de sistemas la visión es muy parecida, pues los modelos que desarrolla el analista permitirán al usuario y equipo de desarrollo tener una visión de cómo deberá ser el nuevo sistema. Se requerirá desarrollar distintos modelos, cada uno de ellos orientados a elementos específicos del sistema, pero que en conjunto representan las características del *software*.

El modelado de *software* es una técnica con complejidad inherente a los sistemas. Ayuda al equipo de trabajo de desarrollo de *software* a visualizar el sistema de **información a construir**.

Los modelos:

- Ayudan a visualizar cómo es o queremos que sea un sistema.
- Permite especificar la estructura o el comportamiento de un sistema.
- Proporciona plantillas que nos guían en la construcción de un sistema.
- Permite documentar las decisiones que se toman.

Beneficios:

- Mejora la productividad.
- Reduce el número de defectos en el código.
- Facilita la comprensión.
- Facilita la descomposición y modularización del *software*.
- Facilita la evolución y mantenimiento del *software*.
- Mejora la reusabilidad.

El modelado es una técnica de ingeniería probada y aceptada, que consiste en el desarrollo de varias clases de modelos con distintos propósitos. Un modelo es una abstracción de algo, con el objeto de lograr una comprensión de lo que se desea construir. El modelado es la parte esencial de las actividades relacionadas con la producción del *software* de calidad. Se construyen los modelos para representar y comunicar la estructura deseada y el comportamiento del sistema, para visualizar y controlar la arquitectura del sistema, para comprender mejor el sistema que estamos construyendo, para descubrir oportunidades, para la simplificación y la reutilización y para controlar el riesgo (Booch, 1996).

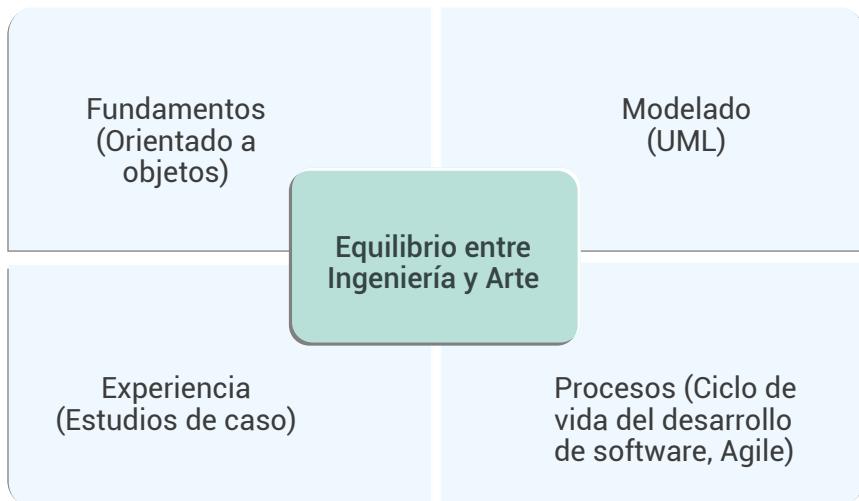
¿Qué es un modelo?

Un modelo es una simplificación de la realidad, al proporcionar los planos de un sistema. Al elaborar modelos, estos pueden ser muy detallados, como también representaciones que den un vistazo de alto nivel del sistema que se construirá, sin dejar de incluir aquellos elementos que son relevantes al nivel de abstracción seleccionado. Un buen modelo incluye elementos relevantes y que influyen en el sistema y omite aquellos elementos que por el momento son irrelevantes (Booch et al., 2006). Es tarea de los analistas descubrir todos los elementos necesarios para que los modelos que representan al *software* sean los apropiados.

En la figura 1, se puede apreciar al modelado como uno de los componentes fundamentales en el desarrollo del *software*.

Figura 1

Aprender y adoptar la ingeniería de software.



Nota. Adaptado de Software Engineering with UML (p. 3), por B. Unhelkar, 2018, CRC Press.

El modelado permite la creación de diagramas estandarizadas y especificaciones asociadas que contribuye en gran medida a mejorar la comunicación y aumentar la participación de todos los Interesados en el proyecto. La correcta participación de los Interesados mejora la calidad del software, reduce los errores y fomenta la fácil aceptación de la solución por parte de los usuarios.



Actividad de aprendizaje recomendada

Para tener una visión más específica del modelado en el contexto del desarrollo de software, en YouTube busque el video titulado “**Ingeniería del software II - Introducción al modelado del software**”, publicado por la Escuela Universitaria Politécnica, y analice cada uno de los conceptos que realiza el expositor.

Como pudo apreciar en el video, el modelado es uno de los componentes claves debido a que contribuye a la producción del buen software, al modelar podemos visualizar cómo es el estado actual, analizar su comportamiento y sobre todo tener una idea clara de cómo será el nuevo sistema.

1.2. Importancia del modelado

En el tema anterior ya hemos indicado de lo importante del modelado al momento del desarrollo del software. Precisamente debido a que el sistema software satisface necesidades que son articulados por los usuarios, por lo tanto, los usuarios son una parte integral del proceso de desarrollo. Los usuarios expresan necesidades, definen el sistema y brindan la visión del producto. El desafío consiste en capturar, modelar y traducir estas necesidades y requisitos en una solución aceptable.



Actividad de aprendizaje recomendada

Para ampliar los conceptos sobre la importancia del modelado, revise las diapositivas 6-9 del recurso educativo “Lenguaje Unificado de Modelado – UML” – [Tema 1. Introducción a UML](#).

Como lo pudo analizar, existen diferentes modelos que contribuye al desarrollo del software, desde una perspectiva de interés de quienes participan en este proceso. Es importante recalcar que, es importante tener todos los requisitos bien definidos antes de empezar con el desarrollo, esto permitirá reducir problemas al momento de la implementación.

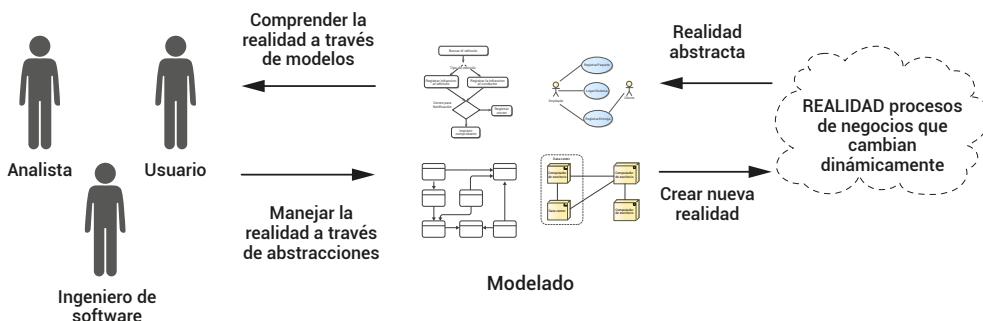
Todos los proyectos de desarrollo, indistintamente del proceso o metodología, inician con una comprensión de los objetivos de negocio, seguidamente se realiza el modelado y análisis de requisitos. Estas actividades permiten tener claridad y modelar las necesidades y deseos de los usuarios a través del modelo de requisitos. Estos modelos representan requisitos en el espacio del problema, diseños en el espacio de la solución y restricciones en el espacio arquitectónico. En general, un proyecto bien organizado y un adecuado modelado, permite que los equipos realicen todo lo que se necesita para que la codificación se desarrolle de forma correcta. Es decir, el equipo de desarrollo mediante el modelado le permitirá tener mejor comprensión del problema, podrá definir una adecuada solución antes de codificar la aplicación, reducirá los esfuerzos innecesarios y mejorará la calidad del software (Bhuvan, 2018).

Dada la importancia, el modelado considera dos propósitos principales:

1. Ayudar a comprender la realidad de negocio: a partir de las necesidades de los usuarios e interesados en especial, se desarrollan diferentes componentes que permiten mapear la situación actual del negocio.
2. Crear una nueva realidad del negocio: analizada la situación actual, se establecen las mejoras y la forma en que se desarrollarán las actividades.

Como se puede apreciar en la figura 2, estos dos propósitos nacen de las necesidades de los interesados y llegan a ellos mediante la propuesta de un modelo. Por lo tanto, la importancia del modelado consiste en: comprender los sistemas, aplicaciones y procesos existentes, seguido de la creación de nuevos procesos, proporcionando una base para probar y permitir comunicaciones efectivas con todos los interesados. El modelado permite una mejor comprensión de un problema y la solución antes de codificar la solución, lo que reduce los esfuerzos innecesarios y mejora la calidad del software.

Figura 2
Importancia de la ingeniería de software



Nota. Adaptado de *software Engineering with UML* (p. 3), por B. Unhelkar, 2018, CRC Press.

El modelado se utiliza para todo tipo de sistemas, para sistemas pequeños y no complejos son suficientes modelos básicos y generales, sin embargo, para sistemas grandes con un grado de complejidad alto, el modelado resulta imprescindible, por la simple razón de que la capacidad humana es limitada, por lo tanto, es necesario que los modelos ayuden a comprender

y manejar la complejidad. De esta manera, al modelar se persiguen cuatro objetivos (Booch et al., 2006):

1. Los modelos ayudan a visualizar cómo es o queremos que sea el sistema. El equipo de trabajo puede tener una misma visión del sistema que se está construyendo.
2. Los modelos permiten especificar la estructura o el comportamiento del sistema. Un modelo permite documentar lo necesario de la estructura del sistema antes de ser codificado.
3. Los modelos proporcionan guías o plantillas que guían la construcción del sistema. Estas plantillas con una herramienta de mucho valor durante la construcción. Por ejemplo, para el programador le ayudan a codificar la funcionalidad correcta.
4. Los modelos documentan las decisiones que se han adoptado. Para actividades a largo plazo y no depender de personas o de la memoria humana.

Cualquier proyecto se beneficia de estos objetivos, así sea un proyecto pequeño, considerar que con el paso del tiempo todos los sistemas tienen la tendencia natural de hacerse más complejos. Por este motivo, a pesar de que al inicio pueda pensar que no es necesario modelar, cuando el sistema evolucione se lamentará no haberlo realizado, entonces será demasiado tarde.



Actividad de aprendizaje recomendada

Para tener una idea general de lo que se realiza en el modelado, desde el enfoque del desarrollo de software, revise el texto de ingeniería de Software, de Ian Sommerville. Revise el tema “[Modelado de sistemas](#)” (páginas: 118-121).

Tal como lo habrá analizado, este apartado indica de forma general los diferentes elementos que se pueden crear al momento de desarrollar software, para cada una de las fases del desarrollo de software, estos modelos se basan en la visión 4+1 de la arquitectura de Kruchten.

1.3. Principios de modelado

Continuemos el estudio analizando lo relacionado con los principios del modelado. Empezar comentando que estos principios son el fundamento del modelado orientado a objetos y sobre los que se han desarrollado las diferentes propuestas de modelado.

El modelado tiene una rica historia en todas las disciplinas de ingeniería y precisamente cuatro principios de modelado son derivados de esa historia (Booch et al., 2006). En la siguiente infografía se detallan estos principios:

Principios de modelado



Actividad de aprendizaje recomendada

Para un mayor detalle de los principios, lea el recurso web "[Principio de modelado](#)".

Como habrá leído, estos principios influyen en los modelos que se deben desarrollar dependiendo de la naturaleza del proyecto. Si bien se proponen diferentes modelos, con base en vistas, no todos los sistemas requieren de todas las vistas.

Considerando la importancia de los principios de modelado, vamos a enfocarnos en el fundamento de los modelos orientados a objetos. En el siguiente apartado analizaremos este tema.



Semana 2

1.4. Modelado orientado a objetos

Empecemos el presente tema comprendiendo la importancia del Modelado Orientado a Objetos (MOO) y cómo este apoya el desarrollo de aplicaciones software. El MOO cubre aspectos del análisis y diseño orientado a objetos, proporcionando una forma muy productiva y práctica para el desarrollo de software. Permite representar las cosas mediante el uso de modelos

organizados en torno a conceptos del mundo real (objetos), lo que permite comprender problemas, comunicarse con expertos a distancia, modelar empresas y diseñar programas y bases de datos. Desarrollar un modelo para un sistema de software, antes de su desarrollo o transformación, es tan esencial, como tener un anteproyecto para la construcción de un gran edificio. Los modelos orientados a objetos se representan mediante diagramas. Un buen modelo siempre ayuda a la comunicación entre los equipos de proyecto y a asegurar la solidez arquitectónica(Booch et al., 2006).

El MOO es una técnica de modelado adecuada para el manejo de los sistemas complejos, permitiendo identificar y organizar la aplicación con respecto a su dominio, en lugar de su representación final en cualquier lenguaje de programación. Los desarrolladores de software tienen que pensar en términos del dominio de la aplicación durante la mayor parte del ciclo de vida de la ingeniería de software, por lo que, el desarrollador se ve obligado a identificar los conceptos inherentes a la aplicación. Primero, el desarrollador organiza y entiende el sistema correctamente y luego, los detalles de la estructura de datos y las funciones.

Hay varias ventajas y beneficios del modelado orientado a objetos, entre lo más destacado tenemos: la reutilización y el énfasis en la calidad, permiten el manejo de la resistencia al cambio, la encapsulación y la abstracción, entre otras. Por su propia naturaleza, todas estas características se suman al desarrollo de sistemas:

- Desarrollo más rápido.
- Mayor calidad.
- Mantenimiento más fácil.
- Reutilización de software y diseños, *frameworks*.
- Riesgos de desarrollo reducidos para la integración de sistemas complejos.

La estructura conceptual de la orientación a objetos ayuda a proporcionar mecanismos de abstracción para el modelado, que incluye: clases, objetos, generalización, asociación, etc.

Considerando el proceso de desarrollo de un sistema software orientado a objetos, el modelado considera las siguientes etapas: análisis, diseño de

sistema, diseño de objetos e implementación. A continuación, a manera de definición general vamos a ir comentando cada una de estas etapas.

Etapas de un sistema software orientado a objetos



Actividad de aprendizaje recomendada

Para conocer sobre el proceso de desarrollo del *software*, revise el recurso “Proceso de *software*” del texto “[Ingeniería de software de Ian Sommerville](#). Páginas 27-43.

Como pudo leer, para cada una de las fases del desarrollo de *software*, existen diferentes tipos de modelos, que deberá ser seleccionados de acuerdo con las necesidades del proyecto.

Amplíe su análisis sobre los conceptos relacionados con la POO, revisando las dispositivas “[Conceptos de POO](#)”.

Para realizar los modelos, es necesario disponer de la información necesaria, por ejemplo, si deseamos conocer la situación actual, será necesario levantar toda la información a partir de las personas (interesados), documentos existentes o sistemas de ser el caso. Ante esta situación, los procesos de desarrollo de *software* recomiendan desarrollar un proceso de descubrimiento, refinamiento, modelado y especificación. Estas actividades se conocen como ingeniería de requisitos. No vamos a enfocarnos en el desarrollo de estas actividades, pero si es importante saber que en el desarrollo de *software* son importantes estas actividades que aportan al desarrollo de los modelos.



Actividad de aprendizaje recomendada

Para identificar los conceptos asociados a la identificación de requisitos, revise el recurso “[Requisitos](#)”.

Como puede observar, para la definición de los requisitos es fundamental la información que se obtiene de diferentes fuentes, aplicando diferentes estrategias de obtención. Luego, utilizando estrategias de análisis se desarrollan los modelos que permiten conocer la situación actual. A partir

de estos modelos se procede con posibles ajustes que permitan diseñar la solución a desarrollar.

1.4.1. Tipos de modelos orientados a objetos

Analizada la importancia del MOO, el Modelado Orientado a Objetos, se identifican tres tipos de modelos para una descripción del sistema. Estos modelos son:

- Modelo de objeto.
- Modelo dinámico.
- Modelo funcional.

Los modelos de objetos se utilizan para describir los objetos en el sistema y su relación entre sí en el sistema. El modelo dinámico describe la interacción entre los objetos y el flujo de información en el sistema. Las transformaciones de datos en el sistema se describen mediante un modelo funcional. Los tres modelos son aplicables durante todas las etapas de desarrollo. Estos modelos tienen la responsabilidad de adquirir los detalles de implementación del desarrollo del sistema. Es importante tener en cuenta que no se puede describir un sistema completamente hasta que no se describan correctamente los tres modos.

Hay varias características del modelo orientado a objetos que son indispensables considerar al momento de construir los diagramas. Antes de enfocar estas características es fundamental abstraer las clases y los objetos. Por esta razón, a continuación vamos a analizar detenidamente los conceptos de objetos y clases. Elementos básicos sobre los que se desarrollarán las distintas representaciones y modelos.

1.4.2. Objetos

Los objetos son entidades, sujetos o cosas que encontramos en el dominio del problema, es decir, en situaciones de nuestro mundo cotidiano empresarial, organizacional o institucional. En sí el objeto es un componente que tiene un rol específico y que puede interactuar con otros, se trata de establecer equivalencias del mundo real con componentes software. En este sentido, el modelado para representar y resolver problemas del mundo real resulta sencillo e intuitivo.

A nuestro alrededor se encuentran muchos ejemplos de objetos del mundo real, por ejemplo, los libros, el escritorio, la televisión, etc. Todo lo que

el objeto de software sabe (estado) y puede hacer (comportamiento) se expresa mediante las variables y los métodos dentro de ese objeto. En otras palabras, todos los objetos comparten estados y comportamiento. Digamos que un objeto de software que modela una **bicicleta** del mundo real tendría variables que indicaran el estado actual de la bicicleta: su velocidad es de 20 mph y su marcha actual es la 3ra marcha, etc. Para mayor detalle de este ejemplo revise el video [Ejemplo de la bicicleta](#)

Todos los objetos presentan dos componentes: un conjunto de características y propiedades y un comportamiento determinado. Por ejemplo, pensemos en un **estudiante**, éste tiene identificación, nombre, edad. El comportamiento del estudiante puede ser descrito en función de las operaciones que puede realizar, por ejemplo: consultar, modificar o eliminar datos. De igual manera, para implementar un objeto puede tener atributos y puede definir operaciones que puede realizar.

Los atributos equivalen a las características de los objetos del mundo real y las operaciones se relacionan con su comportamiento. Para los valores y atributos considerar lo siguiente:

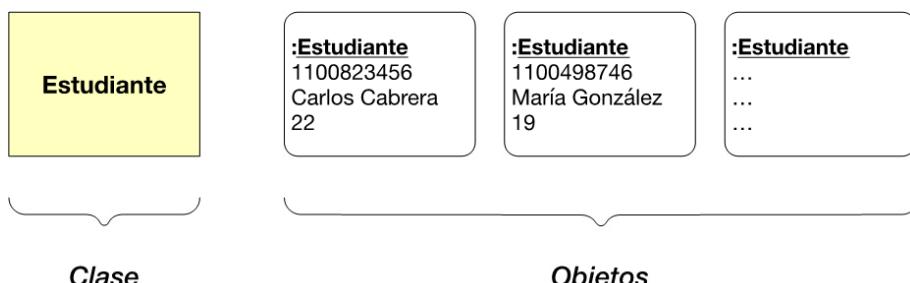
- Un atributo de objeto es una propiedad de una clase a la que se le asigna un nombre.
- Un valor es un elemento de información.
- Los modelos orientados a objetos se construyen sobre estructuras: clases y relaciones.
- Es importante no confundir valores y objetos. Los objetos tienen identidad, mientras que los valores no.
- En el análisis de un modelo no es necesario añadir un atributo que actúe como identificador interno de las instancias de la clase:
 - Los identificadores del objeto son implícitos a la metodología orientada a objetos.
 - Solo utilizamos aquellos atributos que tienen sentido para la aplicación.
- Durante la etapa de diseño puede ser necesario utilizar identificadores internos, por ejemplo, cuando se utiliza un modelo relacional.

Ahora, veamos cómo las características que comparten las clases se capturan como atributos y operaciones. Estos términos se definen de la siguiente manera:

- **Los atributos** son elementos con nombre que sirven para almacenar valores que pertenecen a una clase. Diferentes objetos de una clase determinada suelen tener al menos algunas diferencias en los valores de sus atributos.
- **Las operaciones** representan servicios que un objeto puede solicitar para afectar el comportamiento del objeto o del propio sistema.

En la figura 3, se muestra los objetos estudiante, que se indica anteriormente.

Figura 3
Clase y objetos



Nota. Sucunuta, M., Jaramillo, D., 2022

1.4.3. Clases

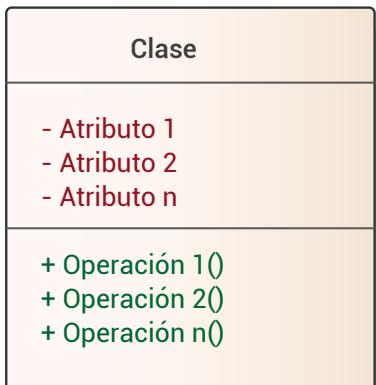
Una clase es una colección de cosas o conceptos que tienen las mismas características. Cada una de estas cosas o conceptos, se llama objeto. El uso de un conjunto de clases como el vocabulario central de un proyecto de software tiende a facilitar en gran medida la comprensión y el acuerdo sobre los significados de los términos y otras características de los objetos en el sistema.

Las clases pueden servir como base para el modelado de datos. En el modelo orientado a objetos, el término **clase** es la base a partir de la cual las herramientas de modelado visual (como Rational Rose XDE, Visual Paradigm, Enterprise Architect, etc.) funcionan y diseñan el modelo de sistemas.

La notación estándar de una clase es un cuadro con tres secciones. La sección superior contiene el nombre de la clase en negrita, la sección central contiene los atributos que pertenecen a la clase y la sección inferior contiene las operaciones de la clase, como puede ver en la figura 4.

Figura 4

Notación de una clase



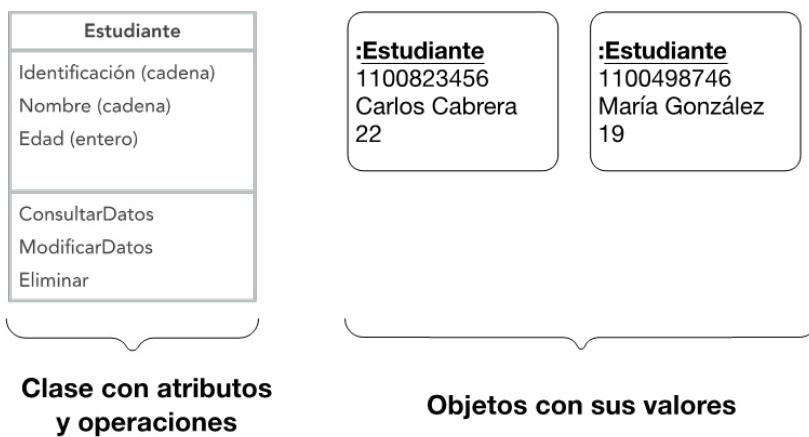
Nota. Sucunuta, M., Jaramillo, D., 2022

Consideré la siguiente convención de nomenclatura para las clases:

- Los nombres de las clases son sustantivos simples o frases nominales.
- Los nombres de atributos en una clase son sustantivos simples o frases nominales. La primera palabra no está en mayúscula, pero las palabras posteriores pueden estar en mayúscula.
- Los nombres de las operaciones son verbos simples. Al igual que con los atributos, la primera palabra no se escribe en mayúscula y las palabras siguientes pueden estar en mayúscula.

En la figura 5, se muestra la notación de modelado de objetos. La clase **estudiante** tiene como atributos: **identificación**, **nombre** de tipo cadena, y **edad** de tipo entero. Y como operaciones tiene: **ConsultarDatos**, **ModificarDatos** y **Emiminar**. Un objeto de la clase **estudiante** tiene el valor **1100823456** como identificación, el valor **Carlos Cabrera** como nombre y el valor **22** como edad.

Figura 5
Atributos y valores



Nota. Sucunuta, M., Jaramillo, D., 2022

1.4.4. Comunicación por paso de mensajes

Estará de acuerdo en que un objeto por sí solo generalmente no es muy útil. Los objetos suelen aparecer como componentes de un programa o sistema más grande. A través de la interacción de estos objetos, se logra la funcionalidad de los sistemas. Los objetos de software interactúan y se comunican entre sí mediante el paso de mensajes. Cuando el objeto X quiere que el objeto Y realice uno de los métodos del objeto Y, el objeto X envía un mensaje al objeto Y. El paso de mensajes proporciona dos beneficios significativos.

- Las características de un objeto se expresan a través de sus métodos, por lo que el paso de mensajes admite todas las posibles interacciones entre objetos.
- Cierra la brecha entre los objetos. No es necesario que los objetos estén en el mismo proceso, o incluso en la misma máquina, para enviar y recibir mensajes entre sí.

Ejemplo de clases y objetos

El ejemplo que vamos a utilizar se refiere a la facturación. Todos conocemos una factura. En la figura 6, se muestra la imagen de una factura emitida por un supermercado a un determinado cliente por la compra de ciertos productos.

Figura 6

Ejemplo de una factura

The image shows a scanned receipt from a supermarket. The receipt is divided into several sections:

- Factura:** At the top right, there is a red oval around the header information: RUC: 1790016919001, FACTURA N°: 086 - 105 - 000383186, NO. DE AUTORIZACIÓN: 1908202201179001691900120861050003831860446037113, FECHA Y HORA DE AUTORIZACIÓN: 19/08/2022 23:6:40, AMBIENTE: PRODUCCIÓN, EMISIÓN: NORMAL, and CLAVE DE ACCESO: 1908202201179001691900120861050003831860446037113. An arrow points from this section to the label "Factura".
- Cliente:** In the middle left, there is a red oval around the client information: RAZÓN SOCIAL / NOMBRES Y APELLIDOS: Carlos Alberto Jumbo, RUC/CI: 1100312345, and FECHA DE EMISIÓN: 19/08/2022. An arrow points from this section to the label "Cliente".
- Productos:** In the middle right, there is a red oval around the product table. The table has columns: Cod Principal, Cant, Descripción, Precio unitario, Descuento, and Precio total. It lists three items: MAGGI SALSA DE TOMATE (1.77), MASTER GUANTE LIM PROFUNDA 8 1/2 (0.32), and SX. FUNDA ROLLO COCINA (2.36). An arrow points from this section to the label "Productos".
- INFORMACIÓN ADICIONAL:** On the left, there is a table showing additional information: DEDUCIBLE ALIMENTACION: 1.77, AHORRO AFILIADO: 0.32, AHORRO POR DESCUENTOS: 0.37, and AHORRO TOTAL: 0.69.
- DESCUENTO:** On the right, there is a table showing the breakdown of discounts: SUBTOTAL 0%: 0,00, SUBTOTAL 12%: 5,45, SUBTOTAL NO SUJETO DE IVA: 0,00, SUBTOTAL SIN IMPUESTOS: 5,45, DESCUENTO: 0,33 (highlighted in red), ICE: 0,00, IVA 12%: 0,65, PROPINA: 0,00, and VALOR TOTAL: 6,10.

Nota. Sucunuta, M., Jaramillo, D., 2022

Para identificar los objetos es recomendable contar con la descripción del problema. Algunos de estos temas estarán en: la especificación de los casos de uso, historias de usuario, requerimientos, documentación del cliente, etc., en donde debemos localizar los nombres nominales, los mismos que tengan un significado dentro de la aplicación que se va a desarrollar, además un objeto tiene existencia propia y puede ser identificado, es decir se distingue de otros objetos. Tal como se observa en la figura 6, los datos que permiten identificar a los objetos son:

- Datos del cliente (RUC/CI: 1100312345, nombres y apellidos: Carlos Alberto Jumbo).
- Datos de los productos que se venden (Cod principal: 786100123856, Cant: 1, Descripción: MAGGI salsa de tomate, Precio unitario: 1.7679, Descuento: 0, Precio total: 1,77).

- Datos de la factura (RUC: 1790016919001, Factura Nº: 086-105-000383186, Subtotal 0%: 0, Subtotal 12%: 5.45, Subtotal no sujeto de IVA: 0, Descuento: 0.33, Valor total: 6.10).

Los objetos que se identifican son:

Figura 7
Identificación de objetos

Cliente	Producto
- Identificación = 1100312345 - Nombre = Carlos Alberto Jumbo	- Cantidad = 1 - Código = 786100123856 - Descripción = Maggi salsa de ... - ValorUnitario = 1.7679

Nota. Sucunuta, M., Jaramillo, D., 2022

Las clases que se definen son: cliente, producto, factura y detalle_factura. La clase **cliente** que tendrá toda la información de los clientes. La clase **producto** tendrá la información de todos los productos que se venderán. La clase **factura** tendrá toda la información de las facturas emitidas. Finalmente, tenemos la clase **detalle_factura** que tiene la información de todos los productos que se venden en esta factura.

1.4.5. Características básicas de sistemas orientado a objetos

Analicemos las características básicas en torno a las cuales se desarrollan los sistemas orientados a objetos, tal como se indica en el siguiente recurso.

Características básicas de sistemas orientado a objetos



1.5. Lenguaje Unificado de Modelado (UML)

1.5.1. Visión general de UML

El Lenguaje Unificado de Modelado (Unified Modeling Language, UML), este lenguaje nació como un estándar para la representación y visualización de los objetos, es decir, que los miembros de los equipos de desarrollo tengan una misma interpretación de un determinado modelo. Es un lenguaje que es utilizado para el análisis y diseño orientado a objetos. Se utiliza para: visualizar, especificar, construir y documentar. Los componentes (bloques) básicos de UML utilizados son: elementos, relaciones y diagramas.

Básicamente, el lenguaje de modelado unificado se centra en los conceptos de Booch, OMT e ingeniería de software orientada a objetos. El resultado de estos conceptos es un lenguaje de modelado único, común y ampliamente utilizable para los usuarios de estos y otros métodos(Booch et al., 2006).

UML es un lenguaje para:

- **Visualizar:** UML ofrece un conjunto de símbolos y estándares con una semántica bien definida, de manera tal que un desarrollador pueda escribir un modelo y otros lo puedan interpretar fácilmente sin ambigüedades.
- **Especificar:** UML permite construir modelos precisos, sin ambigüedades y completos, permitiendo la especificación de los aspectos más importantes del análisis, diseño e implementación de un sistema software.
- **Construir:** aunque UML no es un lenguaje de programación visual, sus modelos pueden conectarse directamente a una variedad de lenguajes de programación, como es Java, C++, Visual Basic, o incluso a bases de datos relacionales.

- **Documentar:** UML a través de la construcción de los artefactos permite documentar a arquitectura del sistema con todos sus detalles. Entre los artefactos tenemos:
 - Requisitos.
 - Arquitectura.
 - Diseño.
 - Código fuente.
 - Planificación de proyectos.
 - Pruebas.
 - Prototipos.
 - Versionamiento, etc.

Tal como se mencionó, modelar es diseñar la aplicación *software* antes de escribir su código, de manera que, al realizar un modelo en las fases iniciales del desarrollo, ayuda a detectar fallos y posibles errores en los requisitos, lo que ayuda a realizar verificaciones antes de la implementación. UML es considerada como parte de la estrategia para documentar, por lo tanto, el uso de UML depende del proyecto y la metodología utilizada, por ejemplo, en las metodologías ágiles los requisitos suelen cambiar continuamente, en cuyo caso no conviene realizar una documentación exhaustiva, de igual manera tampoco son convenientes diagramas al detalle para equipos pequeños si nadie los va a revisar. Otras metodologías utilizan de distinta manera UML, por ejemplo, Cascada, en el análisis y diseño, los analistas capturan las necesidades y diseñan las funcionalidades mediante diagramas UML.

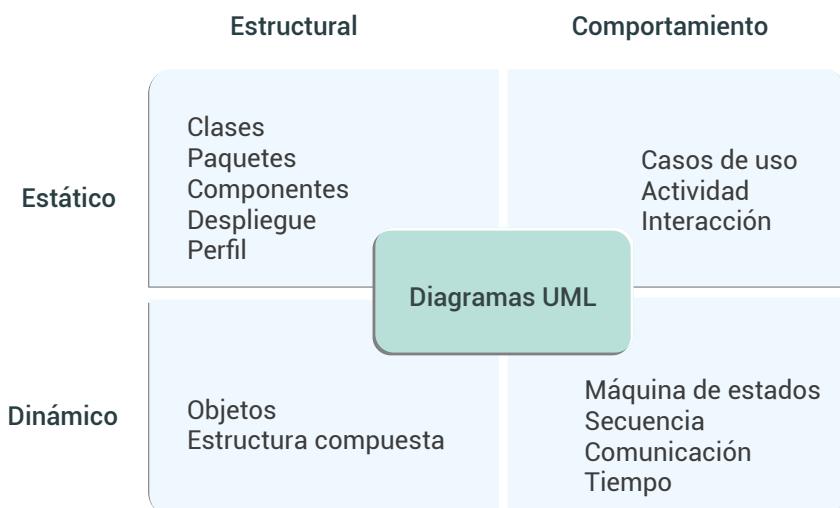
En consecuencia, UML no impone una metodología, sencillamente es un lenguaje que ayuda a documentar un sistema de forma concisa, utilizando una notación estándar para que todos los miembros del equipo y usuarios interpreten de igual manera el sistema.

1.5.2. Diagramas UML

Un diagrama UML es una representación gráfica, que muestra una serie de elementos conectados por relaciones. Los diagramas son una de las vistas más expresivas, tienen un nombre y tipo y se construye para una audiencia particular para transmitir una idea o para crear una descripción narrativa de parte del modelo (Aparx Systems, 2022).

Figura 8

Diagramas UML desde un punto de vista estructural-comportamiento y estático-dinámico



Nota. Adaptado de Object-Oriented Software Engineering Using UML, Patterns, and Java, por B. Bruegge, A. Dutoit, 2012, Prentice Hall.

La naturaleza de los diagramas UML se entiende desde el enfoque estructural versus comportamiento y estático frente a dinámico, tal como se muestra en la figura 8.

Estructural frente a comportamental: el aspecto estructural de un diagrama ilustra la forma en que se *organiza un sistema*, mientras que el aspecto comportamental modela *el flujo del sistema*. La estructura, por ejemplo, muestra cómo las clases se relacionan entre sí en un diagrama de clases. El comportamiento muestra la forma en que un usuario interactúa con el sistema, por ejemplo, a través de un caso de uso o un diagrama de actividad.

Estático frente a dinámico: el aspecto estático frente a dinámico representa la *dependencia temporal* del modelo. Un diagrama sin concepto de tiempo o movimiento es estático, mientras que uno que muestra cambios en el tiempo (o incluso una instantánea en el tiempo) se considera dinámico.

La naturaleza de los diagramas UML mencionados anteriormente tampoco es una clasificación hermética. Cada diagrama exhibe la naturaleza antes mencionada de los diagramas UML, pero algunas características se exhiben con mucha fuerza y otras pueden no existir en esos diagramas.

La especificación UML define catorce tipos de diagramas y enumera los elementos y las relaciones que se pueden incluir en cada diagrama. En la tabla 1, se indican a qué grupo pertenece cada diagrama.

Tabla 1

Clasificación de Diagramas UML por grupo.

Grupo	Descripción y diagramas
Diagramas estructurales.	Representan los elementos estructurales que componen un sistema o función, reflejando las relaciones estáticas de una estructura o arquitecturas en tiempo de ejecución. Los diagramas considerados son: <ul style="list-style-type: none">▪ Clases.▪ Estructura compuesta.▪ Componentes.▪ Despliegue.▪ Objetos.▪ Paquete.▪ Perfil.
Diagramas de comportamiento.	Muestran una vista dinámica del modelo, que representa las características de comportamiento de un sistema o proceso empresarial. Los diagramas son: <ul style="list-style-type: none">▪ Actividad.▪ Casos de uso.▪ Máquina de estados.▪ Tiempo.▪ Secuencia.▪ Comunicación.▪ General de interacción.
Diagramas extendidos.	Proporciona un conjunto de tipos de diagramas adicionales que amplían los diagramas UML básicos para modelos específicos de dominio.
Diagramas personalizados.	Admite tipos de diagramas específicos de MDG Technologies, incluidas las tecnologías integradas.

Nota. Adaptado de Unified Modeling Language (UML) por Aparx Systems, 2022

1.5.3. Herramientas para UML

Existen diversas herramientas que permiten realizar el modelado orientado a objetos con UML, ya sean gratuitas o de pago, encontrar la herramienta adecuada no es fácil, ya que, aunque existen innumerables proveedores de programas UML en la práctica, no todos ofrecen las mismas funciones, ciertas herramientas requieren pocos recursos, pero ofrecen pocas

funciones; otras permiten desarrollar cualquier tipo de diagrama y exportarlo a diferentes lenguajes de programación o importar un modelo desde un código existente.

El consorcio [Object Management Group](#) (OMG), que especifica los estándares UML, recomienda primero considerar exactamente lo que se desea representar con los diagramas UML. ¿Debe mostrarse la estructura o el comportamiento del sistema? Dependiendo de la respuesta, elegir la herramienta de modelador UML más adecuada para el proyecto. A continuación, en la tabla 2, se indican algunas herramientas considerando ciertos criterios.

Tabla 2
Usuarios-interesados del caso de estudio

Herramienta	Creador	Plataforma/ SO	Lenguaje generado	Licencia
Enterprise Architect	Sparx System	Windows	ActionScript, C, C#, C++, Delphi, Java, PHP, Python, Visual Basic, Visual Basic .NET, DDL, EJB, XML Schema, Ada, VHDL, Verilog, WSDL, BPEL, Corba IDL	Comercial
Lucidchart	Lucid Software	Windows, macOS, Linux, Solaris	Desconocido	Comercial/ Libre (educativa)
Modelio	Modeliosoft	Windows, Linux, macOS	Java, C++, C#, XSD, WSDL, SQL	Libre
Microsoft Visio	Microsoft	Windows, MacOS		Comercial
Start UML	KLabs Co.,Ltd.	Windows, MacOS, Linux	Java, C++, C#, SQL	Comercial

Nota. Sucunuta, M., Jaramillo, D., 2022

1.5.4. Representación de las características de MOO mediante UML

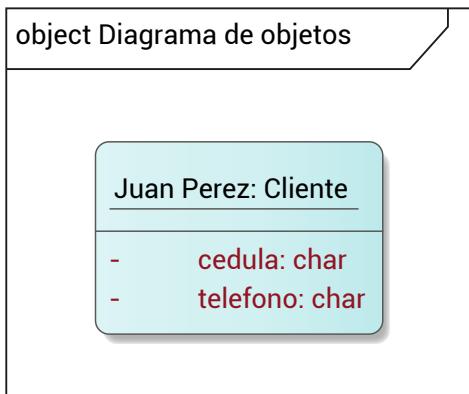
Las características del desarrollo orientado a objetos son el fundamento del modelado, por tal motivo a continuación se indica la forma en que se modelan utilizando UML. En este apartado ampliamos los conceptos analizados anteriormente, utilizando UML.

Objetos y clases

En UML un objeto se representa por un rectángulo en cuyo interior se escribe el nombre del objeto subrayado. Todos los objetos tienen tres

características o propiedades que sirven para definir a un objeto de forma única, estos son: un estado, un comportamiento y una identidad.

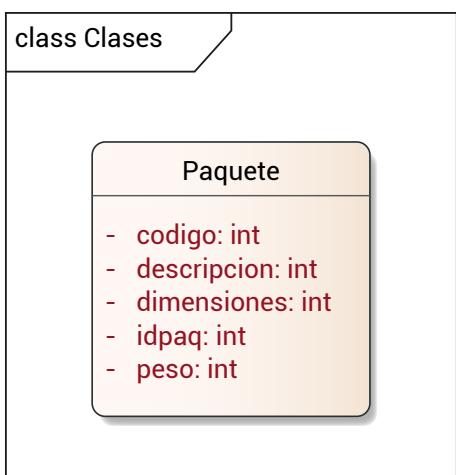
Figura 9
Representación de un objeto



Nota. Sucunuta, M., Jaramillo, D., 2022

En UML una clase se representa a través de una caja rectangular dividida en compartimentos. El primer compartimento se escribe el nombre de la clase, el segundo compartimento se indican los atributos y el tercer compartimento se escriben las operaciones.

Figura 10
Representación de una clase



Nota. Sucunuta, M., Jaramillo, D., 2022

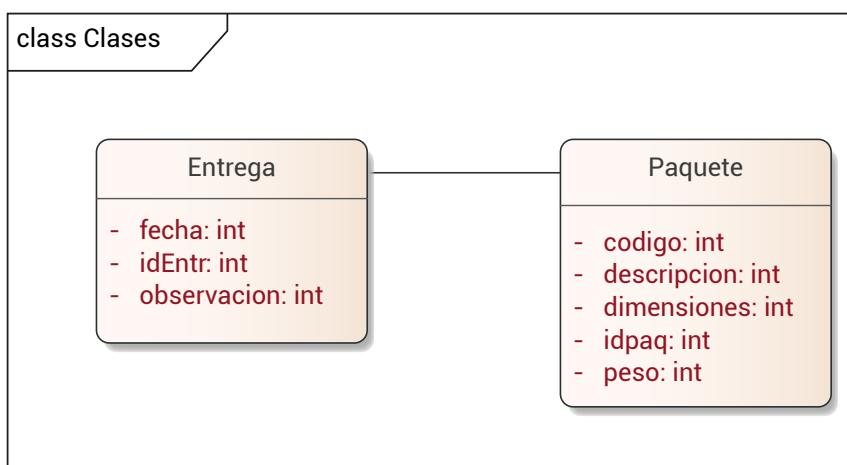
Las clases no pueden interactuar de manera aislada, se requiere que interactúen entre sí a través de las relaciones entre clases. Una relación es una conexión semántica que permite que una clase pueda conocer los atributos, operaciones y relaciones de otras clases. En UML la forma en que se conecta las clases entre sí, ya sea física o lógicamente, se modela como relaciones.

Asociación

La asociación y las clases son similares en el sentido de que las clases describen objetos y las asociaciones describen enlaces. La figura 11 muestra cómo podemos representar la asociación entre la clase entrega y la clase paquete.

Figura 11

Asociación



Nota. Sucunuta, M., Jaramillo, D., 2022

Las asociaciones pueden ser binarias, ternarias o de orden superior. En la práctica, la gran mayoría de asociaciones son asociaciones binarias o ternarias. Pero una asociación ternaria se forma por compulsión; no se pueden convertir en asociación binaria. Si una asociación ternaria se descompone en alguna otra asociación, se perderá cierta información.

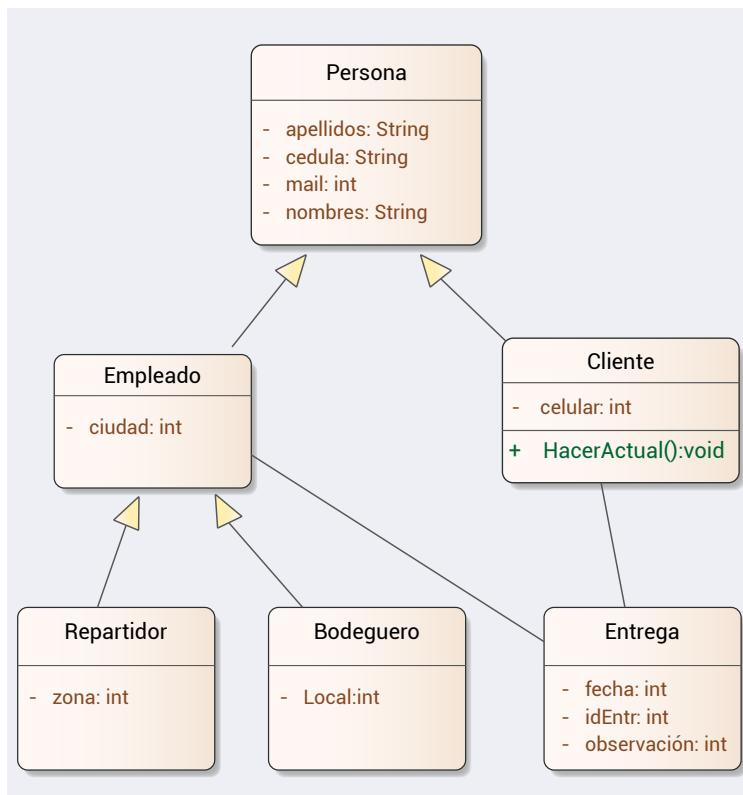
Generalización

Como se puede apreciar en la figura 12, se ejemplifica la representación en UML el concepto de herencia. Podemos observar una herencia entre la clase

padre persona y la clase hija cliente, donde la clase hija heredara todos los atributos/operaciones de su clase padre, e implementará otros atributos que diferenciaran de las otras clases que participan en la herencia.

Señales que aquí podemos encontrar una asociación entre empleado y entrega. El empleado provee una herencia con sus clases hijas, repartidor y bodeguero. Esto implica que la entrega la puede hacer cualquiera de estas instancias de la clase padre, es decir, si no lo hicieramos así deberíamos tener una doble asociación entre repartidor-entrega y bodeguero-entrega, en este caso cuando se realice la codificación de uno de estos dos atributos en la clase entrega deberá en un momento ser nulo.

Figura 12
Ejemplo de Herencia (UML) entre clases

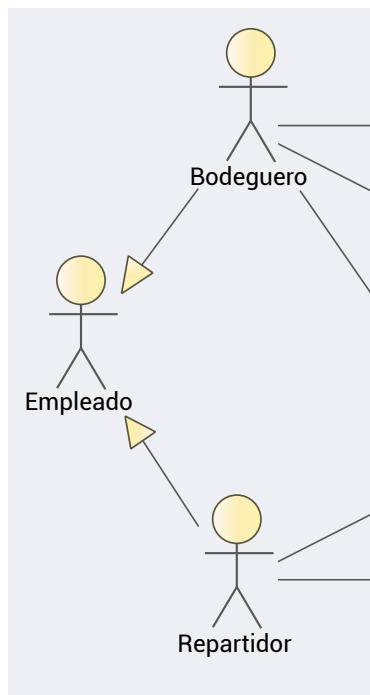


Nota. Sucunuta, M., Jaramillo, D., 2022

La figura 13 presenta una herencia entre actores que también es posible realizarla, esta representación en UML se comprenderá de mejor manera cuando se realicen diagramas de casos de uso.

Figura 13

Ejemplo de herencia (UML) entre actores



Nota. Sucunuta, M., Jaramillo, D., 2022



Actividad de aprendizaje recomendada

Familiarice con la herramienta de trabajo, desarrollando el diagrama de clases que se indica en la semana, para ello:

- Ingresar al laboratorio virtual (revise el manual de usuario sobre el acceso al laboratorio disponible en el CANVAS).
- Ingrese a la herramienta Enterprise Architect, cree un proyecto y realice la réplica de los diagramas de casos de uso y clases. Los conceptos asociados al diagrama, se explicarán en cada una de las semanas que se analicen estos temas.

Lectura recomendada

Revise los temas que se indican a continuación.



- Texto: revisar el tema “[Modeling with UML](#)” (Página: 29-43). Object-Oriented Software Engineering. Using UML, Patterns and Java.
- REA: “Lenguaje Unificado de Modelado - UML”, diapositivas 10-22, para conocer los fundamentos de UML.
- Sitio web: [¿Qué es UML?](#)



Semana 4

1.6. Desarrollo del proyecto software con UML

El desarrollo del software, se realiza mediante una planificación a nivel de proyecto, y es precisamente la naturaleza, el tipo y el tamaño del proyecto de software que determinan la aplicación de los modelos con UML. La versatilidad de UML permite que se utilice para capturar requisitos, modelar flujos de procesos, crear diseños de soluciones de software y desarrollar arquitectura. Es necesario comprender la situación actual del proyecto y luego aplicar cuidadosamente las técnicas relevantes de UML al proyecto.

El tamaño del proyecto se basa en aproximaciones de tiempo, costo y personal. Sin embargo, considerar el tamaño de un proyecto ayuda a comprender la medida y el nivel en el que se puede aplicar UML. En la tabla 3, se muestran algunos parámetros dependiendo del tipo de proyecto.

Tabla 3*Tamaño del proyecto y UML.*

Tipo proyecto	Número de personas	Tiempo	Presupuesto	Diagramas
Pequeños	5-15	3-6 meses	<\$ 2 millones	Diagramas de clase, secuencia y máquina de estado pueden ser muy útiles y, en ocasiones, pueden ser los únicos diagramas utilizados por un par de desarrolladores.
Mediano	15-50 personas	6-12 meses	3-10 millones	Estos proyectos modelan requisitos detallados con casos de uso y diagramas de actividad.
Grandes	+50 personas	Mayor a 1 año	>10 millones	Utilizan UML para el modelado en los espacios del problema y soluciones ampliamente. Además, los modelos UML en el espacio arquitectónico juegan un papel importante en este tipo de proyectos.
Colaborativos	+50 personas	Mayor a 1 año		Son proyectos de implementación y desarrollo basados en la nube. Las distintas ubicaciones, zonas horarias y valores de los equipos que especifican los requisitos y los que desarrollan las soluciones representan un gran desafío, y UML puede ayudar con este desafío a través de modelos visuales estandarizados.

Nota. Sucunuta, M., Jaramillo, D., 2022

1.6.1. Organizando el proyecto

El modelado con UML tiene sentido si el proyecto de software está debidamente organizado. Esta organización del proyecto se fundamenta en el proceso de desarrollo de software, ya que un proceso de software influye en la obtención de los requisitos del usuario, la realización de análisis, la creación de diseños de soluciones, el código de programación y las pruebas.

Una de las confusiones más comunes al realizar el análisis y diseño basado en UML es suponer que el UML en sí mismo es un proceso. Aunque UML es una parte extremadamente importante de un proceso, no es un proceso. UML es un conjunto de notaciones y diagramas, estandarizado y respaldado en la práctica por una herramienta de modelado. UML proporciona el contenido de los entregables, mientras que un proceso define las actividades y roles que producen los entregables. Los procesos en el desarrollo de software han evolucionado para incorporar también los factores socioculturales en la gestión de proyectos. Así, la motivación

del equipo, la experiencia, la colaboración, la confianza y la visibilidad son algunos de los factores que se incorporan a un proceso.

UML que define el contenido de los modelos, debe usarse dentro de los límites de un proceso de desarrollo de software. La ausencia de un proceso, UML corre el riesgo de convertirse en una herramienta para producir cantidades sustanciales de diagramas sin cohesión que son de poco valor para los ingenieros de software.

Bajo estas consideraciones y con el afán de realizar un desarrollo coherente del modelado, se plantea el siguiente esquema para realizar el modelado de un sistema orientado a objetos.

1. Establecer los objetivos de negocio.
2. Priorizar los requisitos.
3. Desarrollar el modelado.
 - Diagramas de Casos de uso.
 - Diagramas de clase.
 - Diagramas de comunicación.
 - Diagramas de actividad.
 - Diagramas de secuencia.
 - Diagramas de componentes.
 - Diagramas de paquetes.
 - Diagramas de despliegue.

En el siguiente recurso, se describe brevemente cada uno de estos diagramas, los mismos que se ampliarán en los siguientes apartados, considerando para el desarrollo el caso de estudio.

[Descripción de diagramas](#)

1.6.2. Ejemplo – Caso de estudio

Para desarrollar el modelado, se requiere de información que permita construir los diferentes componentes, para ello se ha considerado un caso de estudio que se indica a continuación y que se desarrollará en cada uno de los apartados que cubren la presente asignatura.

Las indicaciones específicas y sobre todo la forma en que se deben desarrollar los modelos las compartirá el tutor a través de las tutorías virtuales.

Descripción del caso

Para empezar el trabajo, revise el caso de estudio denominado TRAMIL, este documento lo encontrará en la sección de documentos del CANVAS. A continuación se plantea una descripción general del caso de estudio.

La empresa TRANMIL, está encargada de realizar la entrega de paquetes procedentes de compras (desde fuera de la ciudad) a cada uno de los domicilios de los clientes y que llegan a la bodega general de la empresa. Debido a la gran cantidad de paquetes que están llegando, junto a la cantidad de clientes que están utilizando los servicios y para mejorar los tiempos de entrega, el gerente opta pasar de un registro manual (hojas de Excel) a un proceso automatizado (SI). La empresa registrará cada uno de los paquetes que han llegado a la bodega para generar los listados de las entregas que se deben realizar por cada uno de los encargados de los sectores, los mismos que se ayudarán de dispositivos móviles que les permitan verificar el orden secuencial de las entregas y registrar las mismas, ellos serán los únicos que puedan hacer este proceso. Las entregas que se hagan en la oficina de la empresa deben ser controladas por el encargado de este proceso en las oficinas. Es necesario que los clientes puedan realizar el seguimiento al paquete desde el día que llega a la bodega hasta el momento de su entrega, esto con la finalidad de descongestionar las llamadas a las oficinas. Necesariamente, los clientes deben haberse registrado a través del sistema y una persona encargada del mismo debe confirmar que son clientes para darles de alta a través de un correo electrónico. Los usuarios podrán dar seguimiento desde cuando llega el paquete a la bodega hasta el momento de su entrega.



Objetivos del negocio

- Reducir los tiempos de entrega de los paquetes a los clientes en un 20 % dentro de los 6 meses posteriores al lanzamiento inicial del proyecto.

Considerando la descripción del caso, el objetivo que se plantea permitirá conocer si al desarrollar el proyecto, realmente cumple con la necesidad de los interesados, en el caso particular del caso, el gerente.

Identificación de usuarios

En este apartado debemos considerar que no todos los usuarios van a utilizar el sistema, es importante este apartado. Los usuarios serán quienes interactúen con el sistema. En el caso de estudio listamos los usuarios en la tabla

Tabla 4

Usuarios-interesados del caso de estudio

Usuario	Descripción
Us01	Gerente
Us02	Cliente
Us03	Bodeguero
Us04	Repartidor

Nota. Sucunuta, M., Jaramillo, D., 2022

Aquí podemos considerar para este ejemplo que el gerente va a ser únicamente un interesado o usuario que permitirá ayudar a determinar las necesidades que se deben suplir con el sistema, o sea la automatización de los procesos. Este usuario no aparece como actor del sistema. Este tema se especificará a partir de la siguiente unidad. Téngalo presente.

Identificación de necesidades

A continuación, se han seleccionado algunas necesidades que se encuentran en el enunciado del caso de estudio (tabla 5), recordar que las necesidades a veces pueden ser repetidas o no estar claramente especificadas, este proceso de abstracción lo debe hacer una persona de TI.

Tabla 5*Necesidades obtenidas del caso de estudio*

Necesidad	Descripción	Usuario
Nec01	Mejorar los tiempos de entrega de los paquetes que llegan a la empresa.	Us01
Nec02	Llevar un control de los movimientos de los paquetes desde que lleguen a la oficina hasta que es entregado al cliente, en la oficina o cuando se envía a su casa.	Us01
Nec03	Determinar el responsable de cada una de las actividades, llevando un registro de quien lo hizo, separando las actividades de cada empleado.	Us02
Nec04	Dar seguimiento de cada uno de sus paquetes que llegan desde cuando están en la oficina hasta que lo retira o se lo entregan.	Us02
Nec05	Llevar un control del registro por cada uno de los paquetes que se manejan en la empresa.	Us03
Nec06	Poder consultar la dirección de los clientes por zonas para poder acelerar la entrega de los paquetes.	Us04
Nec07	Registrar a quien se entrega el paquete cuando se lo hace en la casa, la fecha y la hora.	Us04
Nec08	Permitir que los clientes tengan acceso al sistema, que puedan registrarse desde su casa, pero para eso debemos tener a alguien que valide ese registro.	Us01
Nec09	Los clientes pueden cambiar de dirección de entrega si fuese el caso.	Us02

Nota. Sucunuta, M., Jaramillo, D., 2022

En la tabla 6, detallamos los requerimientos que se ha encontrado para nuestro caso de estudio, esto se relaciona con las necesidades que fueron encontradas con anterioridad, donde muchas veces los interesados no utilizan términos más específicos como si lo hace en los requerimientos, pues estos son realizados por especialistas.

Tabla 6*Necesidades obtenidas del caso de estudio*

Requerimiento	Descripción	Necesidad
Req1.	Registrar en el sistema la entrega de paquetes a los clientes al momento de realizar la entrega en la oficina .	Nec2.

Requerimiento	Descripción	Necesidad
Re12.	Registrar la entrega de paquetes a los clientes al momento de la entrega en sus casas .	Nec3.
Req3.	Registrar la entrega de paquetes a los clientes a través de la aplicación.	Nec2, Nec3
Req5.	Cliente da seguimiento del estado de entrega de sus paquetes, a través de su funcionalidad en un navegador.	Nec4.
Req6.	Actualizar y agregar direcciones por parte del cliente.	Nec5.
Req7.	Registro de clientes en el sistema.	Nec6.
Req8.	Autorizar el registro del cliente en el sistema.	Nec6.
Req9.	Notificar al cliente cada vez que cambia el estado de su paquete.	Nec7.
Req4.	Listados clasificados por sector.	Nec8.
Req.	Login de acceso al sistema .	Todas
Req	El cliente puede actualizar la dirección de entrega.	Nec2

Nota. Sucunuta, M., Jaramillo, D., 2022

Estimado estudiante, a partir de estos insumos que hemos obtenido del enunciado del caso de estudio, donde realizamos un proceso de abstracción, realizaremos el proceso de modelado desde la semana 5, con base en el modelo de Kruchten, el mismo que iremos desarrollando de forma paralela.

- Resultado de aprendizaje 2** ▪ Define modelos básicos de descripción arquitectónica

Contenidos, recursos y actividades de aprendizaje



Semana 5

Unidad 2. Modelado de Sistemas



Estimado estudiante, continuando con el estudio de la asignatura y luego de haber analizado los fundamentos del modelado, nos enfocamos en el desarrollo de los diferentes modelos utilizando UML considerando el modelo de vistas. Si bien no se abordan todos los componentes y diagramas que sugiere cada vista, consideramos aquellos que son fundamentales para el desarrollo de los sistemas software.

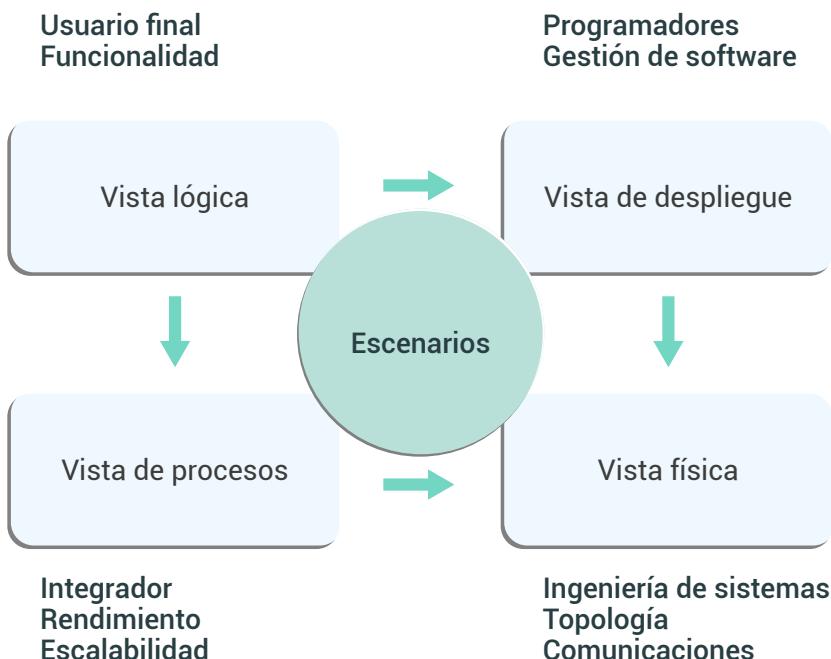
2.1. Modelo “4+1”. Vistas de Kruchten

El modelo “4+1” de Kruchten, es un modelo de vistas diseñado por Philippe Kruchten y que está acorde con el estándar IEEE 1471-2000 (Recommended Practice for Architecture Description of Software-Intensive Systems) que se utiliza para describir la arquitectura de un sistema software intensivo basado en el uso de múltiples puntos de vista.

Lo que propone Kruchten es que un sistema software se documente y muestre con 4 vistas bien diferenciadas y relacionadas entre sí con una vista más, llamada vista “+1”. Estas 4 vistas las denominó Kruchten como: vista lógica, vista de procesos, vista de despliegue y vista física y la vista “+1” que tiene la función de relacionar las 4 vistas citadas, la denominó vista de escenarios.

Cada una de estas vistas muestran toda la arquitectura del sistema software que se esté documentando, pero cada una de ellas ha de documentarse de forma diferente y ha de mostrar aspectos diferentes del sistema software. En la figura 14 se muestran las vistas del modelo.

Figura 14
Modelo de vistas 4+1 de Kruchten



Nota. Adaptado de “Architectural Blueprints – the “4+1” View Model of Software Architecture”, por P. Kruchten, 1995, IEEE Software.

2.2. UML – Kruchten

Antes de empezar con el estudio de cada una de las vistas del modelo de Kruchten siempre queremos asociar o buscar una relación dentro de modelos que se trabajan en cada una de las vistas. No existe un mapeo

igual entre las vistas y los diagramas, esto puede llevar a una confusión y a veces puede considerarse que la persona no está bien o no tiene el conocimiento claro.

Lo invito a leer el siguiente *link*, si no encuentra este documento ha sido cargado en el repositorio como “[mapeo.pdf](#)” donde aunque el análisis es realizado un tiempo atrás, ayuda a aclarar el tema. Así mismo, dentro de esta página encontrará algunos enlaces de interés.



Para terminar con este apartado lo invito a leer el artículo “Planos Arquitectónicos: el modelo de “4+1” Vistas de la Arquitectura del Software”, el mismo que está disponible en la web, en este artículo donde se menciona cada uno de los elementos que debe representarse en las vistas, esto se puede hacer con diferentes diagramas que nos presenta UML.

2.3. Vista de escenarios

Esta vista consiste en un pequeño conjunto de escenarios importantes (instancias de casos de uso más generales). Los escenarios son una abstracción de los requisitos más importantes, escriben secuencias de interacciones entre objetos, y entre procesos. Se utilizan para identificar y validar el diseño de arquitectura. Esta vista es también conocida como vista de escenarios o casos de uso. Esta vista es redundante con las otras (de ahí el “+1”), pero tiene propósitos distintos: como motor para descubrir los elementos arquitectónicos durante el diseño de la arquitectura y como una función de validación e ilustración.

Diagramas de casos de uso

Los diagramas de casos de uso son un tipo de diagrama de comportamiento que sirve para describir lo que debe hacer un sistema desde el punto de vista de quien lo va a utilizar. Un modelo de casos de uso se construye mediante un proceso iterativo durante las reuniones, entre los desarrolladores

del sistema y los clientes (y/o los usuarios finales) conduciendo a una especificación de requisitos sobre la que todos coinciden. Un caso de uso obtiene las acciones y comportamientos del sistema y cómo los actores interactúan.

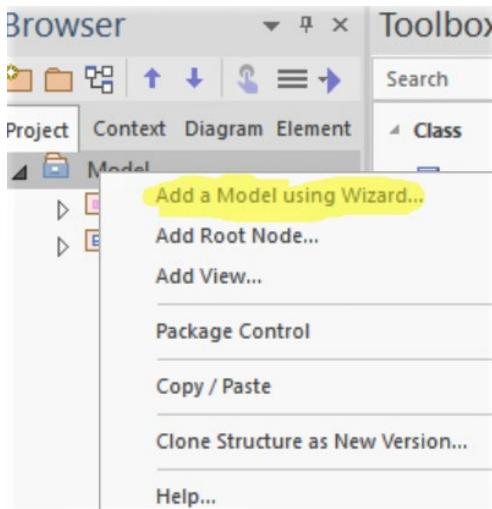
Los casos de uso son utilizados durante la elicitation y análisis para representar la funcionalidad del sistema, enfocándose en el comportamiento del sistema desde el punto de vista externo. Un caso de uso describe una función proporcionada por el sistema que produce un resultado visible para un actor. Un actor representa a cualquier entidad que interactúa con el sistema, por ejemplo: usuarios del sistema, otro sistema o el entorno físico del sistema.

Le invito a revisar los elementos del diagrama detallados en el siguiente recurso.

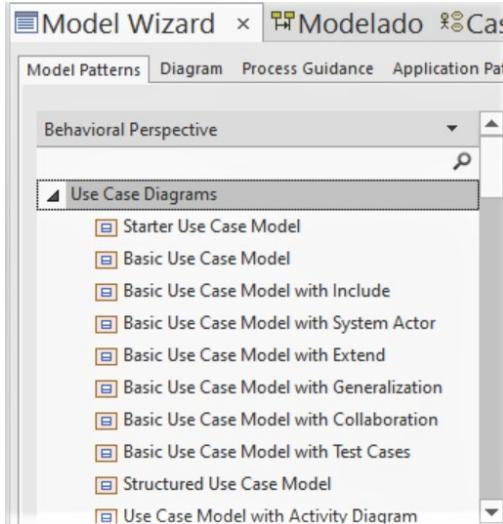
[Elementos del diagrama](#)

En la herramienta de trabajo EA, se utiliza varias plantillas para realizar un mismo diagrama, a continuación, se muestra algunas plantillas:

A partir de un nuevo modelo, puede crear un nuevo diagrama usando el wizard (asistente).



A partir de esto puede seleccionar en el wizard la plantilla que necesite:



Cada una de estas plantillas presenta una forma diferente de representar el diagrama:

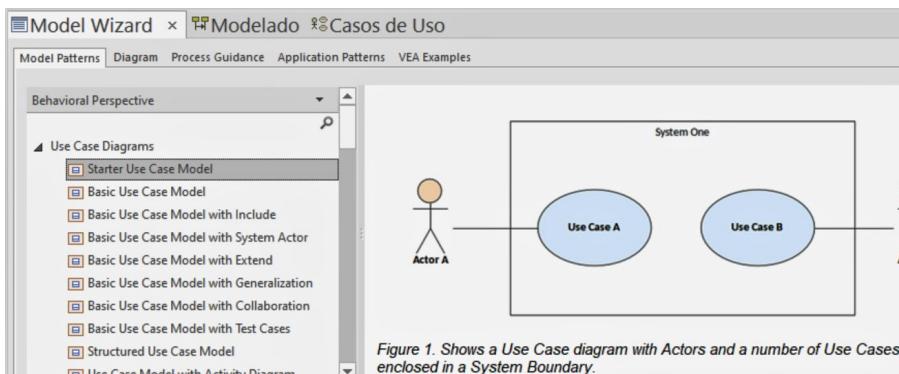
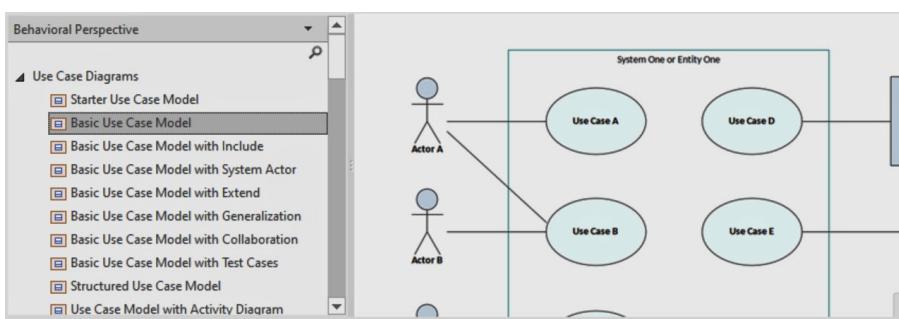


Figure 1. Shows a Use Case diagram with Actors and a number of Use Cases enclosed in a System Boundary.

Otra alternativa se presenta en la siguiente figura:



Actores

En la tabla 7 podemos encontrar los actores del sistema del caso de estudio, aquí una acotación, para poder establecer diferencias con los interesados no se ha considerado al usuario gerente, el mismo que no va a tener interacción con el sistema.

Tabla 7

Actores del caso de estudio

Actor	Nombre	Descripción
Act01	Bodeguero.	Persona que utilizará el sistema en la empresa
Act02	Repartidor	Persona que utilizará el sistema a través de un dispositivo, o
Act03	Cliente	Cliente externo a la organización. Dueño de los paquetes
Act04	Administrador	Encargada de las tareas de configuración y administración de funcionalidades del sistema

Nota. Sucunuta, M., Jaramillo, D., 2022

Casos de uso del caso de estudio

En la tabla 8 se presenta un listado de casos de uso identificados del caso de estudio, esta tabla puede cambiar por el proceso de abstracción que se da al momento de trabajar con el caso. Se mapean los elementos que ya han trabajado, actores y requerimientos.

Tabla 8

Casos de Uso - caso de estudio

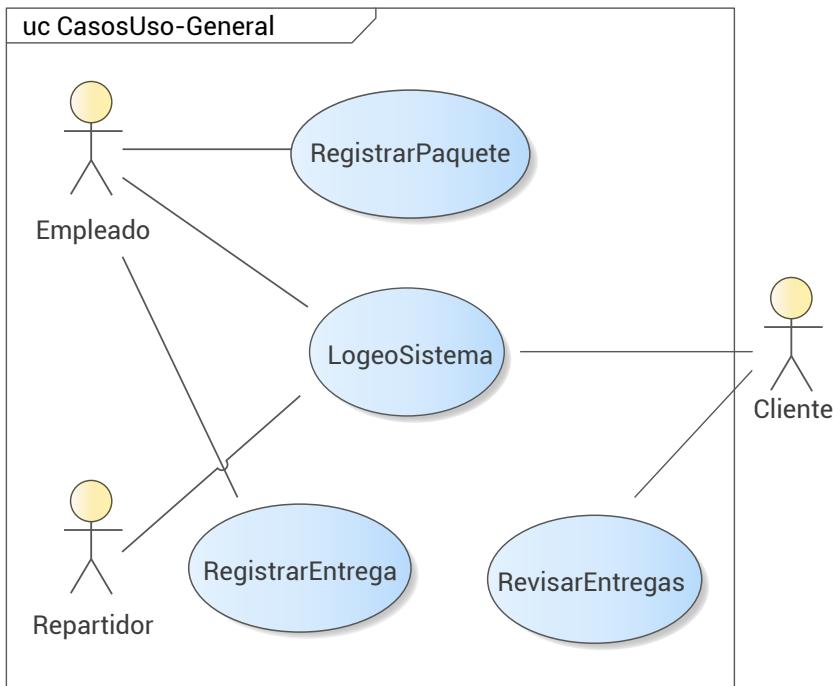
Caso Uso	Nombre	Actor	Req
CU01	RegistrarPaquete	Act01	Req01
CU02	RegistrarEntregaPaquete	Act01, Act02	Req04
CU03	RegistrarCliente	Act01	Req02
CU04	AgregarNuevaDireccion	Act01	Req03
CU05	NotificarCliente		Req04
CU06	RevisarEstadoPaquete	Act03	Req05
CU07	RegistrarseEnSistema	Act03	Req05
CU08	ValidarCliente	Act04	Req07
CU09	Administra Bodegueros	Act04	Req06
CU10	Administrar Repartidores	Act04	Req06

Nota. Sucunuta, M., Jaramillo, D., 2022

Un ejemplo de desarrollo de un diagrama de casos de uso podemos encontrar en la figura 15.

Figura 15

Diagrama inicial de casos de uso



Nota. Sucunuta, M., Jaramillo, D., 2022

De la misma manera, en el repositorio de la asignatura podrá encontrar un proyecto en EA que puede ayudarle sobre el caso de estudio para este ciclo académico. Como ejercicios de práctica se completarán este diagrama



Actividad de aprendizaje recomendada

Para un mayor detalle, revise el tema "[2.4.1 Use Case Diagrams](#)", del texto Object-oriented software engineering. (páginas: 44-50).



2.4. Desarrollo del diagrama de caso de uso – Caso práctico

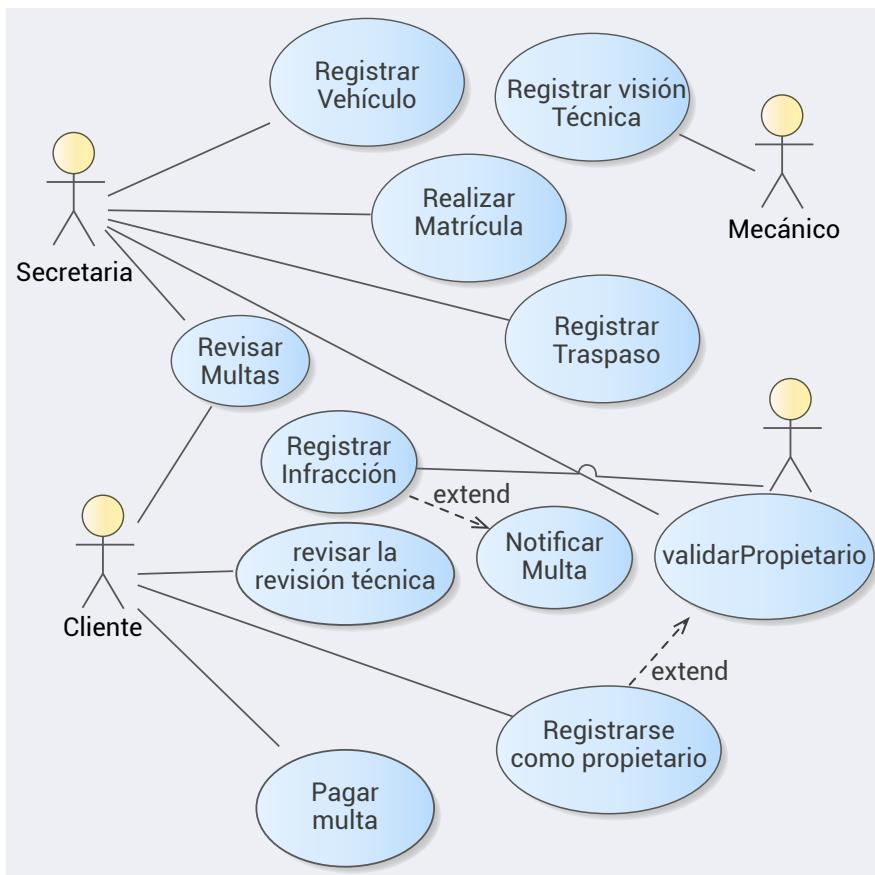
Para complementar el desarrollo del diagrama de casos de uso, vamos a desarrollar la siguiente actividad a través del siguiente ejemplo (caso de estudio).

Ejemplo: TRAMIL.

El caso de trabajo lo puede encontrar en el repositorio con el título: <<Caso de estudio guía – TRAMIL>>

Figura 16

Diagrama caso de estudio guía – TRAMIL



Nota. Sucunuta, M., Jaramillo, D., 2022

Para los estudiantes que no tenga un caso propio para el desarrollo. Podrá contar con el caso de estudio propuesto de acuerdo al ciclo en el repositorio de la asignatura, para ser trabajado en el transcurso.

Actividad interactiva



Con la asistencia del tutor, desarrolle el modelo de casos de uso, del caso práctico. Considere las indicaciones que dará el tutor.



2.5. Vista lógica

La vista lógica permite modelar en función de los servicios que ofrece el sistema a los usuarios, es decir, considera los requisitos funcionales. El sistema se descompone en un conjunto de abstracciones clave, tomadas del dominio del problema, en forma de objetos o clases de objetos. Explotan los principios de abstracción, encapsulación y herencia. Esta descomposición no es solo por lo bien del análisis funcional, sino que también sirve para identificar mecanismos comunes y elementos de diseño en las diversas partes del sistema (Kruchten, 2014).

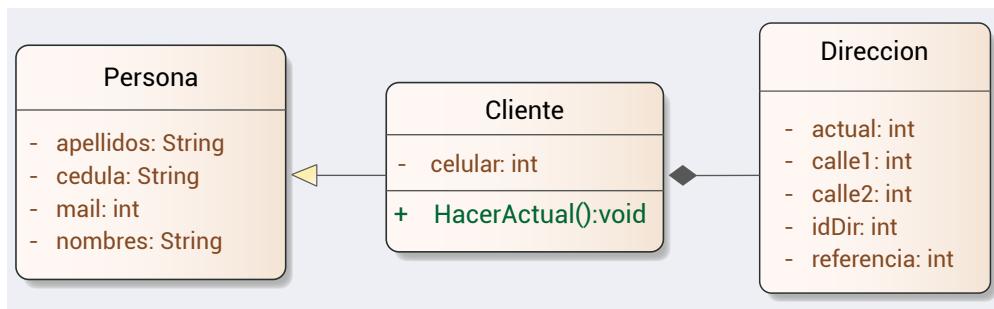
Diagramas de clases

Los diagramas de clases son un tipo de diagramas estructurales. Los diagramas de clases están compuestos por las clases que componen el sistema, su estructura interna y sus relaciones con otras clases.

Clases y objetos

Una clase es una descripción de un conjunto de objetos con las mismas propiedades (atributos) y el mismo comportamiento (operaciones). Tanto en programación orientada a objetos (POO) como en UML, una clase es una plantilla que representa un concepto del mundo real y se utiliza para crear objetos. Por ejemplo, podemos abstraer todos los productos de un supermercado en una clase **producto** junto con sus atributos (código, nombre, precio) y sus operaciones (obtener código, obtener precio, registrar un nuevo producto). A partir de una clase se pueden crear objetos, también llamados instancias de la clase en POO. Los objetos son concreciones de una clase: todos los objetos de una clase comparten las mismas operaciones, pero sus atributos pueden tener valores distintos. Las clases en UML se representan con un rectángulo dividido en 3 partes o “cajas”: la caja superior contiene el nombre de la clase, la del medio los atributos, y la inferior las operaciones.

Figura 17
Clases, atributos y operaciones



Nota. Sucunuta, M., Jaramillo, D., 2022



Actividad de aprendizaje recomendada

Realizar la definición de las clases que se indican en la figura 21, utilizando la herramienta Enterprise Architect.

Relaciones entre clases

Las relaciones pueden ser:

- Asociación.
- Agregación.
- Composición.
- Dependencia.

También puede ocurrir que varias clases tengan operaciones o atributos en común y se requiera abstraerlos utilizando mecanismos conocidos de los lenguajes OO como la herencia e interfaces. UML también permite representar estas relaciones de clases mediante:

- Generalización (equivalente a la herencia en POO).
- Realización (equivalente a interfaces en Java).

Asociación:

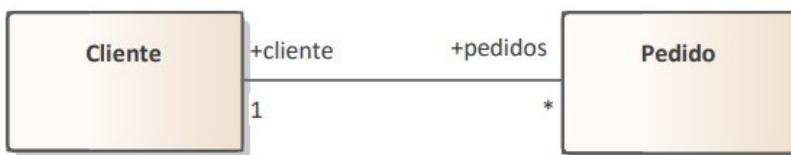
Se representa mediante una línea continua, que puede estar o no acabada en una flecha en “V”, según la navegabilidad. Además, se suelen indicar los roles y la multiplicidad. Si la flecha apunta de la clase A a la clase B, se lee

como: "ClaseA tiene una ClaseB" y se dice que la asociación o navegabilidad es unidireccional. Si no dibuja la flecha se lee "Clase A tiene una ClaseB y ClaseB tiene una ClaseA" y se dice que la asociación o navegabilidad es bidireccional.

Supongamos una aplicación de gestión con las clases cliente y pedido, tal como se indica en la figura 18, cliente guarda información sobre los pedidos y pedido tiene información sobre el cliente que lo realizó. La navegabilidad es, por tanto, bidireccional (18.a). Puede ser que la aplicación guarde la lista de pedidos en el objeto cliente, pero no a la inversa. En este caso la navegabilidad de la asociación sería unidireccional y su representación en UML sería como se indica en 18.b. Para el caso que los pedidos contienen información del cliente, pero los clientes no guardan pedidos, la asociación sería unidireccional, tal como se indica en 18.c.

Figura 18

Representación de asociación en el diagrama de clases.



Nota. Sucunuta, M., Jaramillo, D., 2022

Agregación y composición

Según (Booch et al., 2006) la agregación y la composición son “asociaciones que representan una relación entre un todo y sus partes”. Se puede leer como “ClaseB es un componente de ClaseA”, o también “ClaseA está compuesto por ClaseB”. Las diferencias entre ellas son: en la agregación, los objetos “parte” pueden seguir existiendo independientemente del objeto “todo”. Mientras que en la composición, la vida de los objetos compuestos está ligada a la del objeto que los compone, de manera que si el “todo” se destruye, las “partes” también se destruyen. La agregación se representa uniendo las dos clases (todo/parte) con una línea continua y poniendo un rombo hueco en la clase “todo”. La composición es igual, pero con el rombo relleno.

Figura 19

Ejemplo de agregación y composición



Nota. Sucunuta, M., Jaramillo, D., 2022

Tal como se muestra en la figura 19, la clase persona tiene una relación de agregación con la clase categoría, esto quiere decir que “una persona puede tener una categoría y que una categoría puede estar presente en muchas personas”. En este caso las categorías pueden existir por sí mismas. En la siguiente relación la clase persona tiene una relación de composición con la clase domicilio, ya que domicilio es una parte inseparable de la persona, por lo que si no existiera una persona entonces el domicilio de esta debería desaparecer. Esta relación de composición es una relación fuerte, ya que una instancia arrastra a la otra en caso de eliminación.

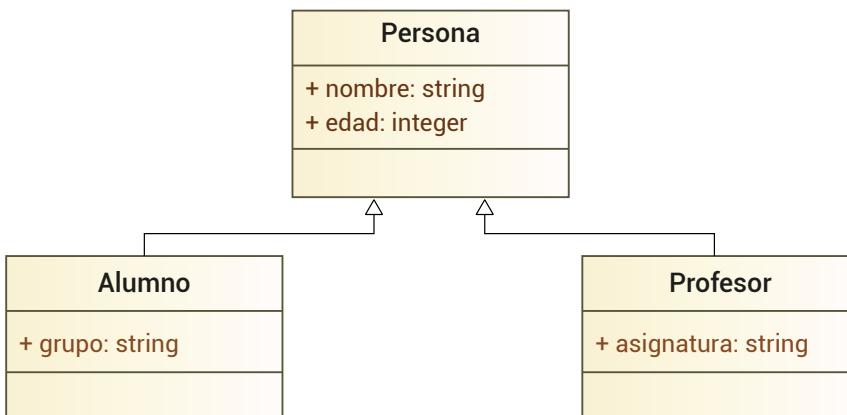
Generalización

La relación de generalización entre dos clases indica que una de las clases (la subclase) es una especialización de la otra (la superclase). Si ClaseA es la superclase y ClaseB la subclase, la generalización se podría leer como “ClaseB es una ClaseA” o “ClaseB es un tipo de ClaseA”. En Java (y la mayoría de los lenguajes orientados a objetos), la generalización se conoce

como herencia. La herencia se utiliza para abstraer en una superclase los métodos y/o atributos comunes a varias subclases. A la subclase también se le puede llamar clase hija o clase derivada. A la superclase también se le puede llamar clase padre o clase base. La generalización se expresa con una línea acabada en una flecha triangular hueca, dibujada desde la clase hija hacia la clase padre. Por ejemplo, en la figura 20, la generalización (herencia) alumno y profesor son especializaciones de la clase persona.

Figura 20

Ejemplo de Generalización (Herencia).



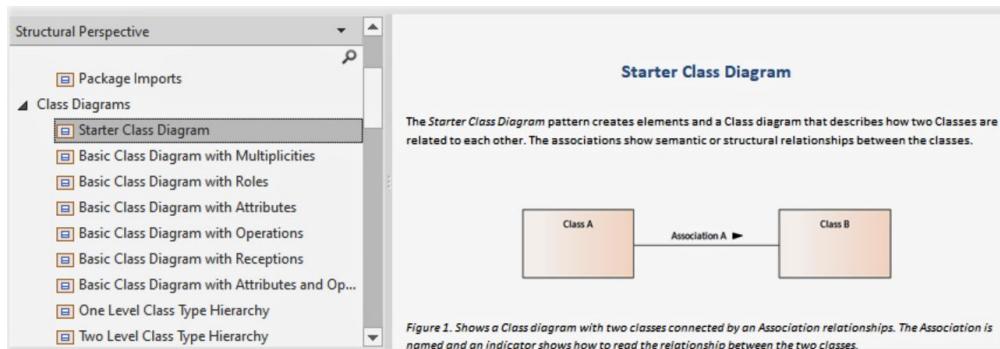
Nota. Sucunuta, M., Jaramillo, D., 2022

En Java, una clase padre puede tener métodos abstractos, lo que significa que para esos métodos no se proporciona ninguna implementación. Una clase abstracta es una clase que tiene al menos un método abstracto, al tener uno o más métodos sin implementación, las clases abstractas no se pueden instanciar. Una clase que extiende a una clase abstracta debe implementar los métodos abstractos o bien volverlos a declarar como abstractos, con lo que ella misma se convierte también en clase abstracta. En UML, tanto las operaciones como las clases abstractas se expresan poniendo el nombre de la operación o la clase en cursiva.

En la herramienta EA vamos a poder utilizar diferentes plantillas a través del asistente, a continuación, mostramos dos de las mismas:

Figura 21

Diagrama de clases, EA-Ejemplo1

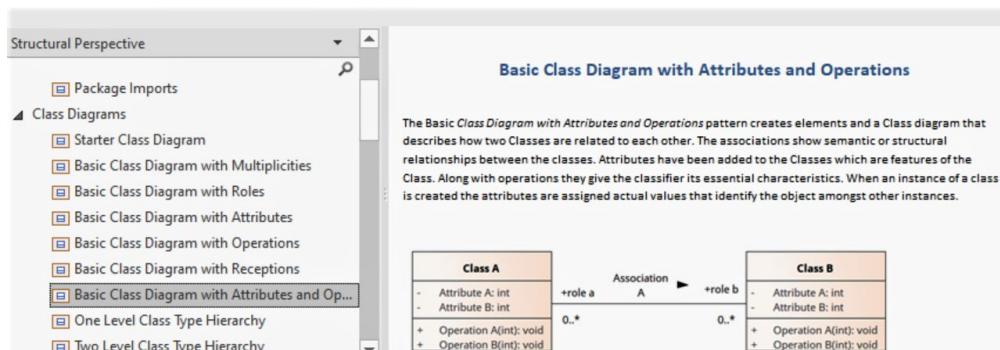


Nota. Sucunuta, M., Jaramillo, D., 2022

Otra plantilla que se puede seleccionar se muestra en la figura 22.

Figura 22

Diagrama de Clases, EA-Ejemplo2



Nota. Sucunuta, M., Jaramillo, D., 2022

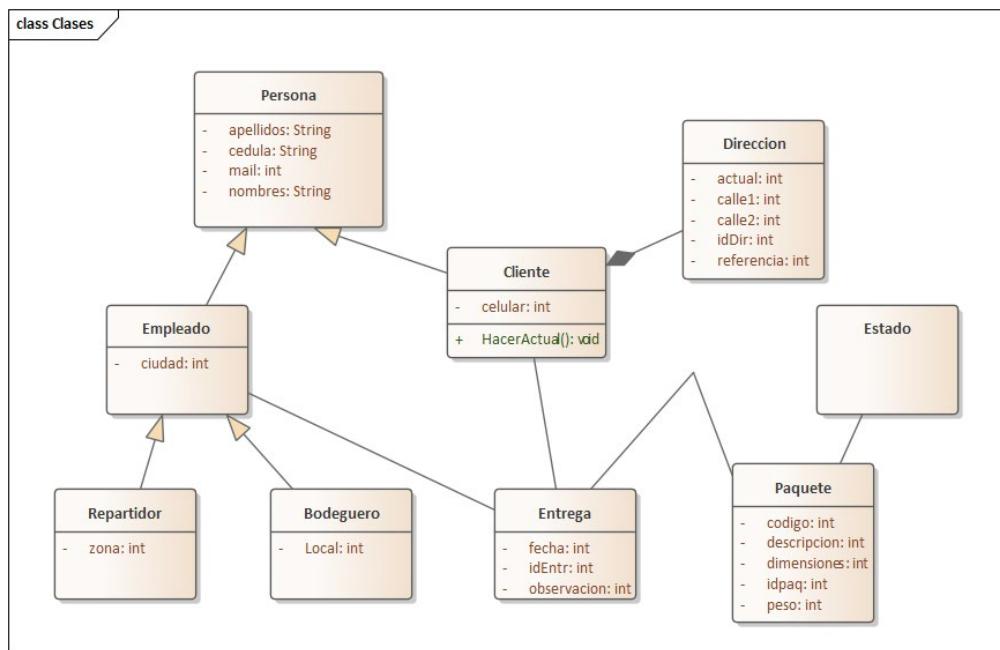
Actividad: desarrollo del siguiente caso.

Para desarrollar el diagrama de clases, es necesario realizar lo siguiente:

- Identificar las clases.
- Identificar las relaciones.
- Elaborar el diagrama de clases utilizando la herramienta EA.

Figura 23

Diagrama de clases.



Nota. Sucunuta, M., Jaramillo, D., 2022

Es esta aproximación inicial de la solución del caso de estudio.

Lectura recomendada

Revise los temas que se indican a continuación.



- Texto: revise el tema **Diseño de clases y objetos**: representaciones gráficas en UML. (páginas 383-405), del texto: programación en C, C++, Java y UML de Luis Joyanes Aguilar e Ignacio Zahonero Martínez.



Semana 8

En la presente semana corresponde realizar la preparación para la prueba del primer bimestre. La evaluación bimestral consiste en preguntas objetivas de la unidad 1 y la unidad 2 (Casos de uso y Diagramas de clase).

Sugiero realizar las siguientes actividades:

- Revisar los videos correspondientes a los temas del primer bimestre.
- Revise los ejemplos presentados en las tutorías por el docente.
- Realice las lecturas de los recursos que se indican en los temas.



Segundo bimestre

Resultado de aprendizaje 2

- Define modelos básicos de descripción arquitectónica.

Contenidos, recursos y actividades de aprendizaje



Semana 9

Unidad 2. Modelado de sistemas



Continuando con el estudio de los diagramas de UML con base en la referencia de la vista lógica del modelo de Análisis de Kruchten, nos encontramos otros de los diagramas que se puede trabajar en esta vista es el diagrama de comunicación.

Consideré este texto que se encuentra en el sitio: *"Hay que dejar claro que Kruchten no dice de qué manera se ha de documentar cada vista; sino que es lo que hay que documentar en cada vista, es decir que cuando se diga que la vista lógica se puede documentar de forma gráfica con un diagrama de clases de UML, no quiere decir que esa vista se tenga que documentar con ese diagrama, sino que ese diagrama (por sus características) puede documentar esa vista"*, esto tiene mucho sentido, pues en algunos casos un

mismo tipo de diagrama puede estar en otra vista y eso no quiere decir que se encuentre mal.

2.6. Diagrama de comunicación

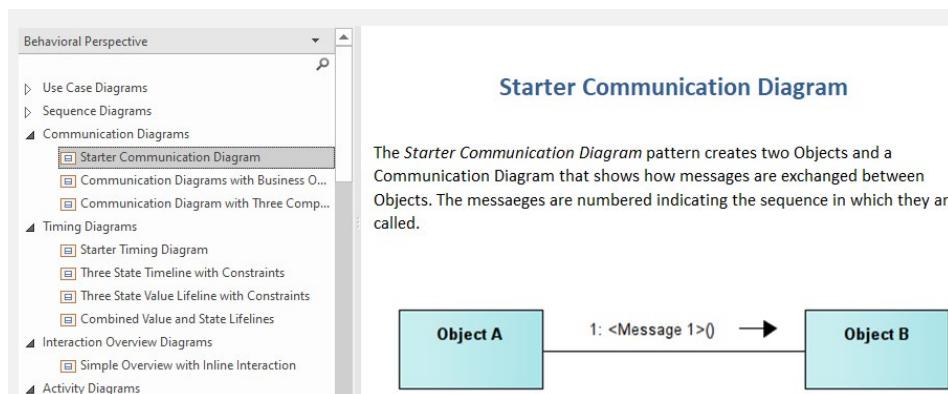
Un diagrama de comunicación se puede enunciar como la forma de representar interacciones entre objetos, se lo puede catalogar como alterno al diagrama de secuencias, también como la visión más simplificada del diagrama de colaboración.

Son un tipo de diagramas de comportamiento que representan la comunicación entre objetos del sistema y el orden en que se envían los mensajes. Son parecidos a los diagramas de secuencia, y de hecho se puede pasar de uno a otro fácilmente. La diferencia está en cómo se representa la información. Mientras que los diagramas de secuencia enfatizan el orden en que se envían los mensajes, los diagramas de comunicación destacan el flujo de control entre los objetos del diagrama.

Revise el contenido [Diagrama de comunicaciones UML 2](#), donde se presenta una explicación de este diagrama.

Su representación puede variar, de acuerdo con la plantilla que se quiera utilizar, puede elegir al momento de realizar este diagrama en la herramienta EA.

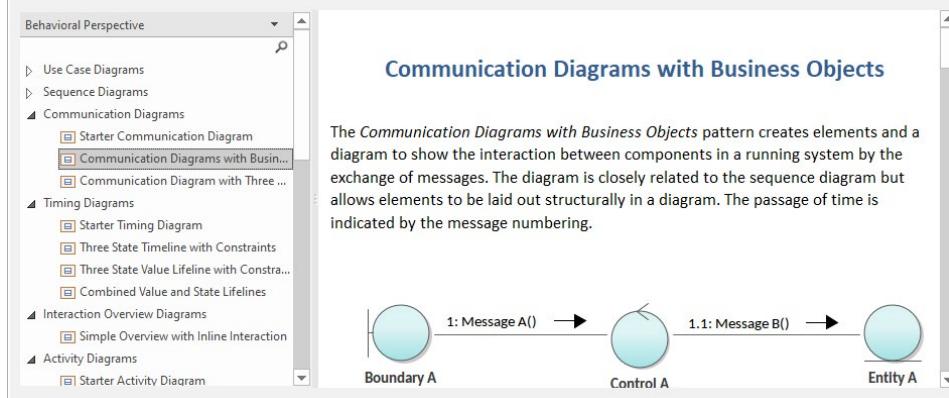
Figura 24
Diagrama de comunicación. Estilo 1



Nota. Sucunuta, M., Jaramillo, D., 2022

Este estilo presenta el diagrama de comunicaciones con objetos de negocios.

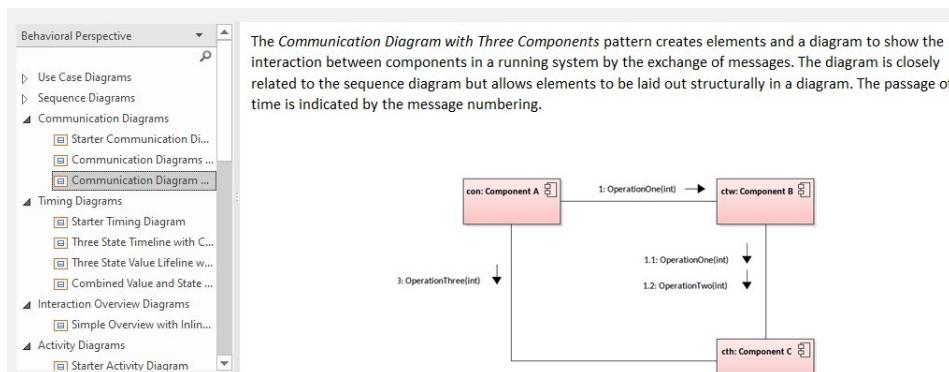
Figura 25
Diagrama de comunicación. Estilo 2



Nota. Sucunuta, M., Jaramillo, D., 2022

En el siguiente estilo del diagrama de comunicaciones presenta la interacción entre componentes y la relación/interacción entre los mismos.

Figura 26
Diagrama de comunicación. Estilo 3



Nota. Sucunuta, M., Jaramillo, D., 2022

Puede encontrar otras alternativas de herramientas para realizar un diagrama. En el siguiente [link Crear un diagrama de comunicación UML](#), puede encontrar el proceso para desarrollar este diagrama en la herramienta Visio.

Los elementos que participan en los diagramas de comunicación tenemos:

Tabla 9

Representación de una clase.

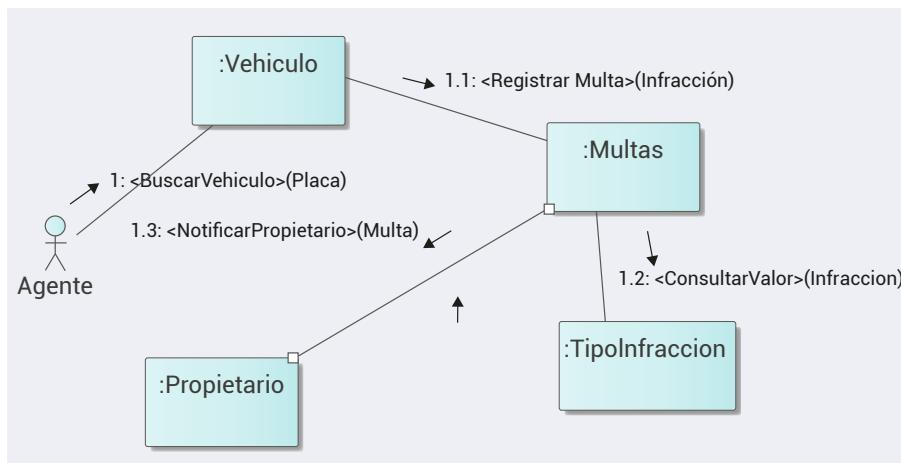
Elemento	Descripción
Mensajes	Cuando dos objetos se envían mensajes entre sí se unen con una línea recta, denominada canal de comunicación. A lo largo del canal de comunicación pondremos los mensajes que se van pasando indicados con una flecha en la dirección adecuada. Al lado de la flecha pondremos el nombre del mensaje o el valor de retorno.
Objetos	Se representan igual que en los diagramas de secuencia, con el nombre del objeto y la clase metidos dentro de una caja. En UML 2 se denominan nodos de comunicación.

Nota. Sucunuta, M., Jaramillo, D., 2022

Este podría ser un ejemplo de un diagrama de comunicaciones, sobre el << Caso de estudio guía – Matriculación>>

Figura 27

Diagrama de comunicaciones



Nota. Sucunuta, M., Jaramillo, D., 2022

Como se puede ver en la figura, podemos encontrar la comunicación que existirá entre los objetos, cuál será el mensaje que se enviará y el parámetro que permitirá hacer la selección. En las cajas se utilizan la instancia de los objetos que fueron trabajados en el diagrama de clases.



Nota. revise en el repositorio de la asignatura el ejemplo del caso de estudio que ha sido propuesta para este ciclo de estudios.



Semana 10

2.7. Vista de procesos

Un proceso es una agrupación de tareas que forman una unidad ejecutable. Los procesos representan el nivel en el que la arquitectura del proceso puede controlarse tácticamente. La vista de proceso se puede describir en varios niveles de abstracción, y cada nivel aborda diferentes preocupaciones. Esta vista se encarga de los aspectos dinámicos del sistema, explica el proceso y la manera en que se comunican. Se enfoca en el comportamiento del sistema en tiempo de ejecución. Esta vista considera aspectos de concurrencia, distribución, rendimiento, escalabilidad, etc. En UML se utiliza el diagrama de actividad o diagrama de secuencia (Kruchten, 2014).

2.7.1. Diagrama de actividades

Este diagrama nos permite determinar el flujo que muestra una actividad en el sistema. Estos diagramas presentan el flujo de trabajo que se ha realizado del punto inicial hasta el final. Aquí se muestran las decisiones que se podrán realizar en un determinado evento.

El diagrama de actividades es un flujo de acciones y objetos que nos permite modelar el comportamiento de un sistema, podemos modelar casos de uso, operaciones.

El diagrama de actividad presenta una variedad de elementos.

Tabla 10*Elementos del diagrama de actividades.*

Elemento	Descripción
	Actividad: se considera como un flujo de ejecución de acciones y objetos que expresan un comportamiento.
	Nodo inicial, nodo terminal: nos permiten determinar desde donde empieza y hasta modelamos el comportamiento.
	Nodo de decisión: permite determinar el camino que debemos seguir basados en una condición.
	Nodo de bifurcación, nos permite determinar caminos separados.

Nota. Sucunuta, M., Jaramillo, D. 2022

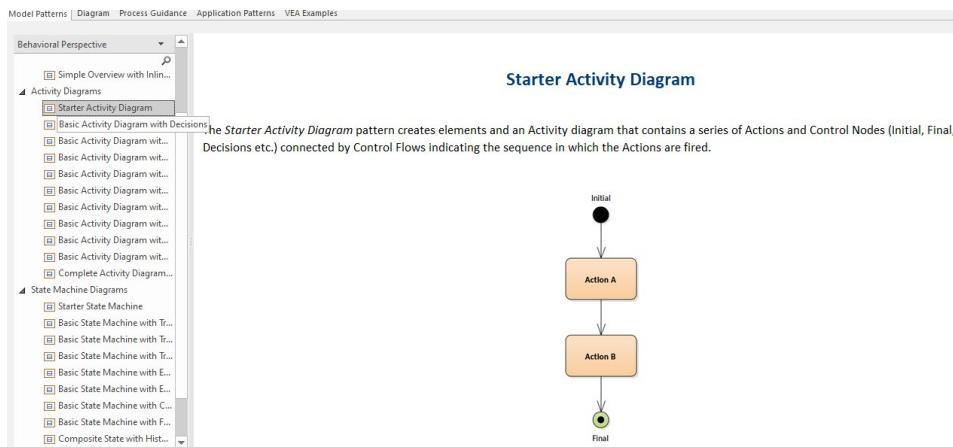
Para una mejor comprensión, revise el recurso, [UML 2 Tutorial - Activity Diagram](#).



Adicional, revise el video [Como elaborar un diagrama de actividad](#), esto le permitirá comprender el desarrollo de un diagrama de actividades.

Figura 28

Diagrama de actividades- EA

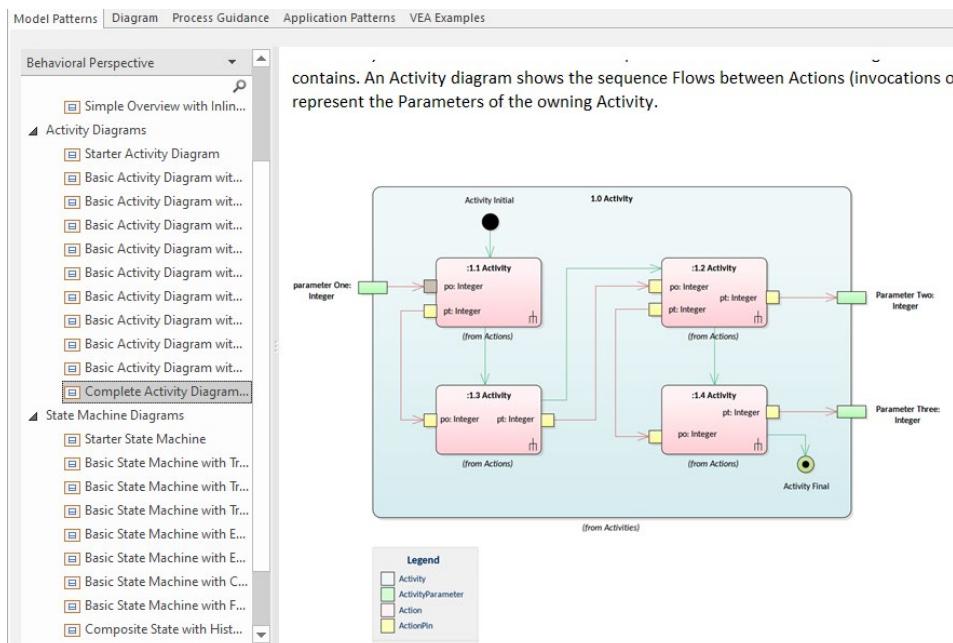


Nota. Sucunuta, M., Jaramillo, D., 2022

Una forma más amplia de comunicar las actividades se puede realizar como está en la figura 29.

Figura 29

Diagrama de actividades-EA-2

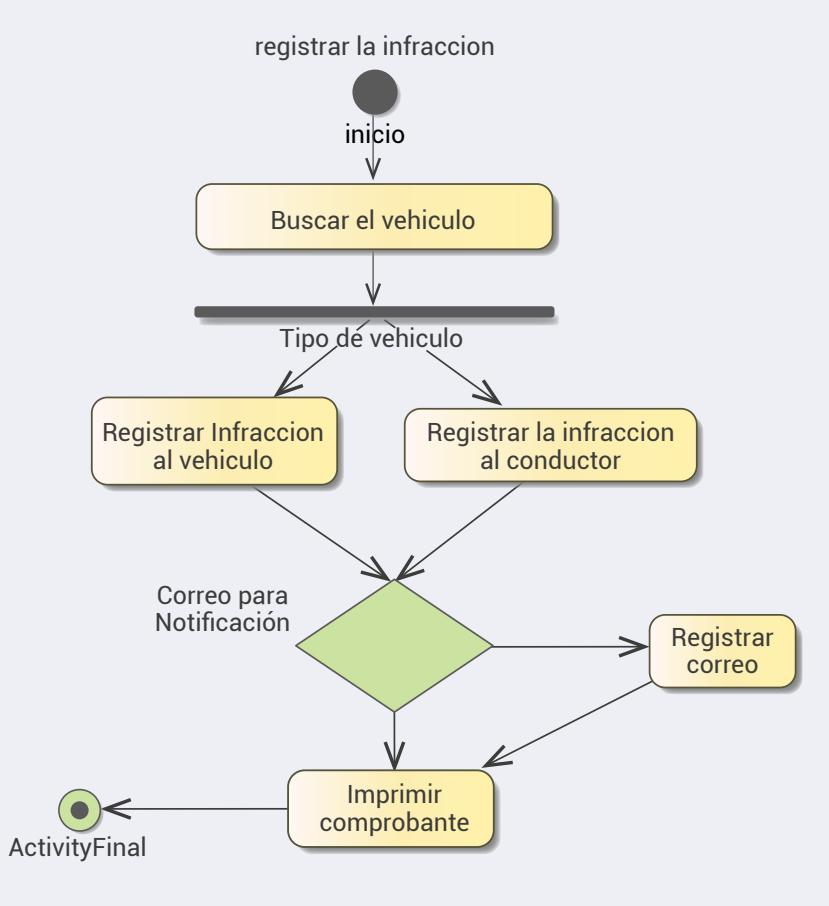


Nota. Sucunuta, M., Jaramillo, D., 2022

El siguiente diagrama muestra el proceso para realizar el registro de una multa en el sistema.

Figura 30

Proceso para realizar el registro de una multa en el sistema.



Nota. Sucunuta, M., Jaramillo, D., 2022

Revise sobre el caso de estudio para este ciclo lo relacionado con diagrama de actividades que se encuentra en el repositorio de la asignatura.



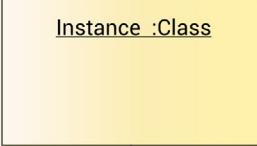
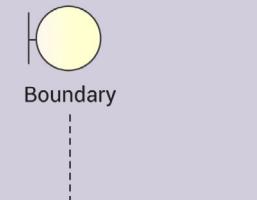
2.7.2. Diagrama de secuencia

Un diagrama de secuencia es un gráfico bidimensional donde el eje vertical representa el tiempo, el eje horizontal muestra objetos del sistema y la interacción entre los objetos se representa mediante flechas que van de unos objetos a otros, ordenadas cronológicamente de arriba abajo. Son un tipo de diagramas de interacción, que a su vez es una categoría de diagramas de comportamiento. Los diagramas de secuencia se utilizan cuando queremos expresar qué objetos se relacionan con qué objetos, enfatizando el orden en que lo hacen y qué tipo de mensajes se envían entre sí. También permiten expresar estructuras de control (condiciones y repeticiones), pero los diagramas de secuencia no están pensados para eso y debemos evitar incluir lógica procedural compleja en ellos.

Los elementos que participan en los diagramas de comunicación encontramos.

Tabla 11

Elementos que participan en los diagramas de comunicación.

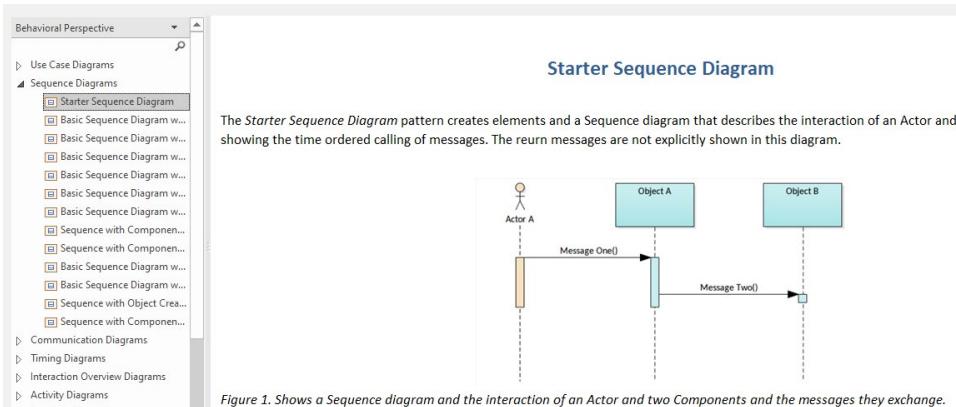
Elemento	Descripción
 Instance :Class	Objetos: son elementos que participan en el diagrama y son instancias de una clase. Se representan por un rectángulo con el nombre del objeto escrito.
 Boundary	Línea de vida: indica la existencia de un objeto (desde que se crea hasta que se destruye) mediante una línea discontinua.

Elemento	Descripción
<pre> sequenceDiagram participant Actor participant ObjectA participant ObjectB Actor->>ObjectA: return:= message(parameter) Actor->>ObjectA: message(parameter) ObjectA-->>Actor: message(return) </pre>	Mensaje: la comunicación entre objetos y activaciones.

Nota. Sucunuta, M., Jaramillo, D., 2022

En la herramienta de trabajo EA, nos permite utilizar varias plantillas, desde una forma sencilla de hacerlo como se muestra en la figura 35.

Figura 31
Diagrama de secuencias -EA-



Nota. Sucunuta, M., Jaramillo, D., 2022

Otra forma de realizar el diagrama de secuencias en EA lo muestra la figura.

Figura 32

Diagrama de secuencias - EA-2

Model Patterns | Diagram | Process Guidance | Application Patterns | VEA Examples

Behavioral Perspective

- > Use Case Diagrams
- > Sequence Diagrams
 - Starter Sequence Diagram
 - Basic Sequence Diagram w...
 - Sequence with Componen...
 - Sequence with Componen...
 - Basic Sequence Diagram w...
 - Basic Sequence Diagram w...
 - Sequence with Object Crea...
 - Sequence with Componen...
- > Communication Diagrams
- > Timing Diagrams
- > Interaction Overview Diagrams
- > Activity Diagrams
- > State Machine Diagrams
 - Starter State Machine
 - Basic State Machine with Tr...

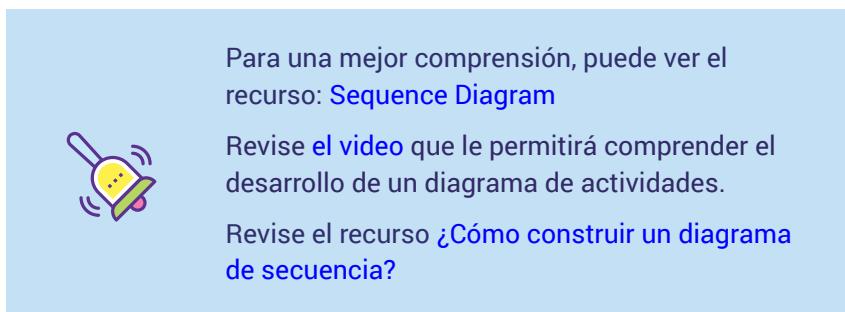
Basic Sequence Diagram with Boundary Control and Entity

The **Basic Sequence Diagram with Boundary Control and Entity** pattern creates elements and a Sequence diagram that describes the Actor and three business objects showing the time ordered calling of messages. The use of the Business Modeling Icons allows a moco pattern (three tier) interaction to be modeled. The Boundary typically represents a human-machine interface, the Control represents and the Entity represents the persistence of information or Objects.

The sequence diagram illustrates the interaction between four entities: Actor A (represented by a stick figure), Boundary A (represented by a light blue rounded rectangle), Control A (represented by a teal rounded rectangle), and Entity A (represented by a light green rounded rectangle). The interaction starts with Actor A sending a message to Boundary A. Boundary A then sends a message to Control A. Finally, Control A sends a message to Entity A.

Figure 1. Shows a Sequence diagram and the interaction of an Actor and three Business Objects and the messages they exchange.

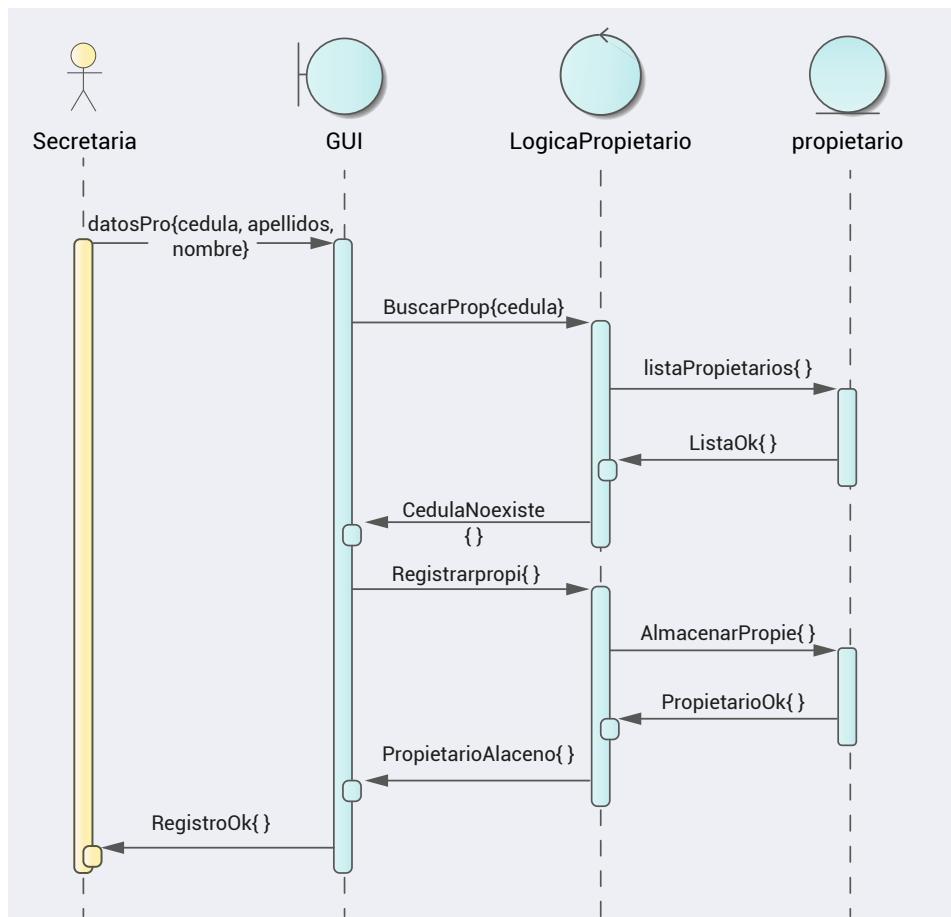
Nota. Sucunuta, M., Jaramillo, D., 2022



Un ejemplo del diagrama de secuencia lo podemos encontrar en la figura 33.

Figura 33

Ejemplo diagrama de secuencia



Nota. Sucunuta, M., Jaramillo, D., 2022

En el repositorio de la asignatura encontrará proyectos realizados en EA donde se han desarrollado algunos ejemplos del diagrama de secuencias.



2.8. Vista de despliegue

Esta vista permite ilustrar el sistema desde la perspectiva del desarrollador y está enfocado en la administración de los artefactos de software. Los diagramas UML que se desarrollan desde la vista despliegue son los diagramas de componentes y el diagrama de paquetes. Estos diagramas permiten representar la distribución física de los componentes software. Estos diagramas permiten y consideran lo siguiente:

- Identificar los nodos.
- Representar de forma clara la arquitectura de red.
- Dar una visión global.

2.8.1. Diagrama de componentes

Mediante este diagrama se expresan como las piezas del software que forman un sistema. Este diagrama presenta un nivel de abstracción inferior que el diagrama de clases.

Los elementos que participan en los diagramas de comunicación encontramos.

Tabla 12

Elementos en los diagramas de comunicación

Elemento	Descripción
 Product	Componente: se considera como una pieza ejecutable de software, por ejemplo .h, dll, Jar. Esto nos permite separar las partes independientes del tamaño de los mismos. El componente puede dar como solicitar servicios, lo que se conoce como contrato
 <<artifact>> <u>manual.jar</u>	Artefacto: unidades físicas donde se colocará información.

Elemento	Descripción
	Interfaz requerida, mostrar las dependencias entre componentes, nos permite recibir funciones, servicios o datos desde el exterior.
	Interfaz expuesta: proporciona al exterior función o servicios.
	Conector para expresar una relación de dependencia entre componentes.

Nota. Sucunuta, M., Jaramillo, D., 2022

En EA podemos encontrar varias plantillas para poder realizar el diagrama, como se muestra a continuación:

Figura 34
Diagrama de componentes-EA

The Starter Component Diagram pattern creates Components and a Component diagram that show: connector indicating that the two Components share information via interfaces. The Components have these have been made visible on the diagram

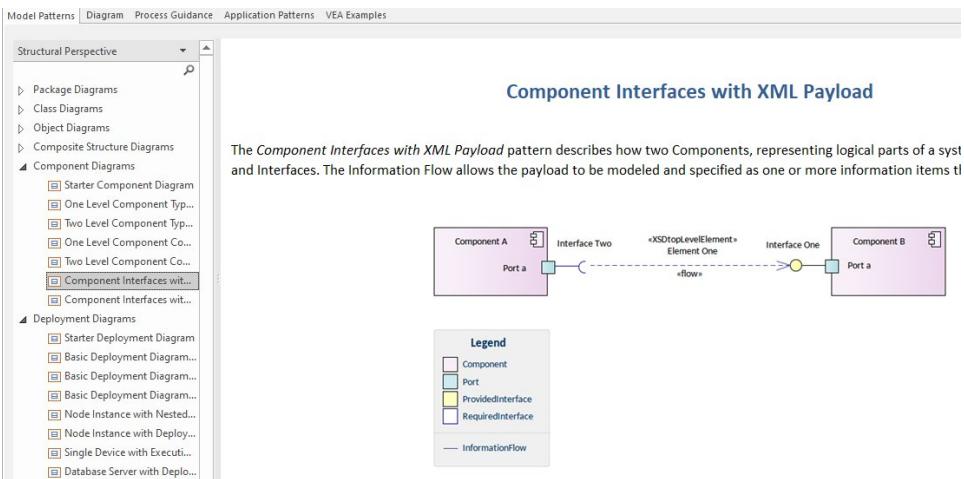
Discussion

Nota. Sucunuta, M., Jaramillo, D., 2022

La figura nos presenta otra forma de realizar el diagrama en la herramienta.

Figura 35

Diagrama de componentes-EA-2

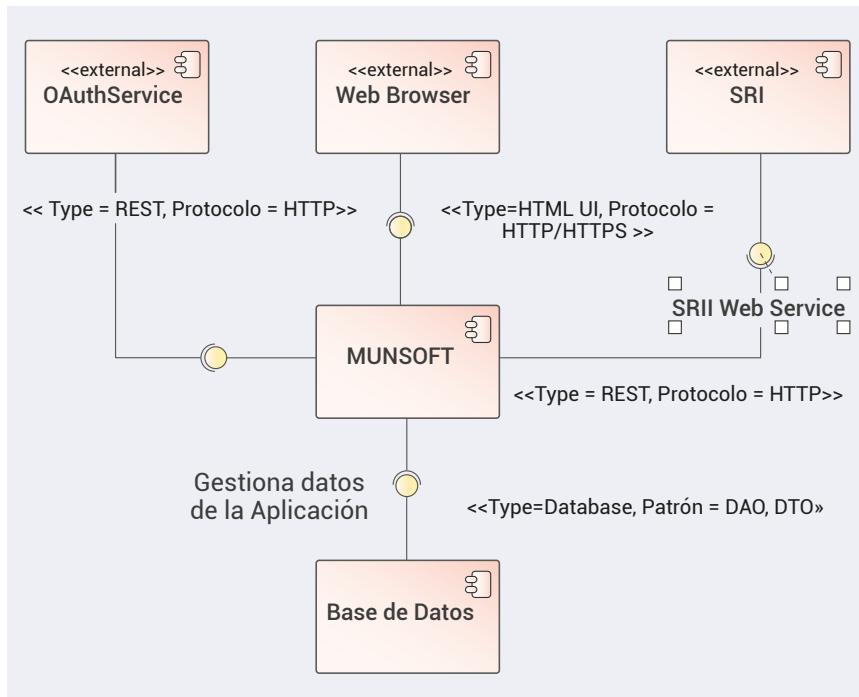


Nota. Sucunuta, M., Jaramillo, D., 2022

A continuación, un diagrama de componentes que se presenta como ejemplo.

Figura 36

Ejemplo de diagrama de componentes



Nota. Sucunuta, M., Jaramillo, D., 2022

Puede revisar la siguiente página para una mejor comprensión: [Diagrama de Componentes UML](#)

Así mismo, el siguiente video indica cómo realizar el [diagrama de Componentes](#)

En el repositorio podrá encontrar dentro del proyecto de caso de estudio del ciclo académico un ejemplo más detallado del mismo.



2.8.2. Diagrama de paquetes

El diagrama de paquete nos permite organizar de mejor manera. Cuando tenemos varias clases podemos ir organizándolas en paquetes, por ejemplo, un paquete de clases puras, un paquete donde colocamos las clases de interfaz y aquellas que las podemos considerar como generales, dependerá mucho cómo se quiera organizar los elementos que tengamos.

En el siguiente diagrama podemos encontrar:

Tabla 13

Elementos del diagrama de paquetes

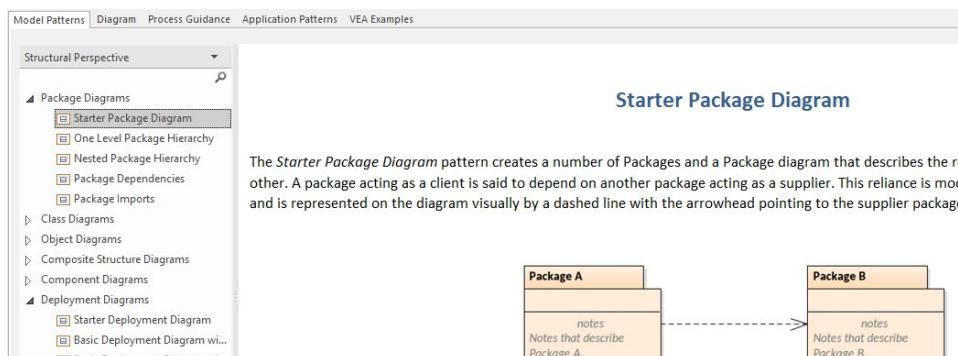
Elemento	Descripción
	Paquetes: se puede considerar como una carpeta donde se pueden agrupar varios elementos, por ejemplo clases.
	Relación de dependencia entre paquetes.

Nota. Sucunuta, M., Jaramillo, D., 2022

En la herramienta que estás utilizando podemos realizar este diagrama de varias formas, como se muestra en la parte izquierda de la figura, ahí se pueden seleccionar en este caso 5 diferentes opciones. A continuación, mostramos dos de ellas, una de la forma normal como se realiza y otra como un árbol jerárquico.

Figura 37

Diagrama de paquetes EA-1



The Starter Package Diagram pattern creates a number of Packages and a Package diagram that describes the relation between them. A package acting as a client is said to depend on another package acting as a supplier. This reliance is modeled and represented on the diagram visually by a dashed line with the arrowhead pointing to the supplier package.

Discussion

Figure 1. Shows a Package diagram with two packages that have dependency relationships indicating the reliance between them.

Nota. Sucunuta, M., Jaramillo, D., 2022

Figura 38

Diagrama de paquetes EA-2

Model Patterns | Diagram | Process Guidance | Application Patterns | VEA Examples

Structural Perspective

- Package Diagrams
 - Starter Package Diagram
 - One Level Package Hierarchy
 - Nested Package Hierarchy
 - Package Dependencies
 - Package Imports
- Class Diagrams
- Object Diagrams
- Composite Structure Diagrams
- Component Diagrams
- Deployment Diagrams
 - Starter Deployment Diagram
 - Basic Deployment Diagram wi...
 - Node Instance with Nested De...
 - Node Instance with Deploy De...
 - Single Device with Execution E...
 - Database Server with Deploye...
 - Node with Component and Ar...
 - Network Device Types

One Level Package Hierarchy

The One Level Package Hierarchy pattern creates a number of Packages and a Package diagram connected into a hierarchy.

Figure 1. Shows a package diagram where the contents of a package is described using the nesting connector.

Nota. Sucunuta, M., Jaramillo, D. 2022



En el proyecto que se presenta en el repositorio de la asignatura puede encontrar un ejemplo, así mismo puede revisar el siguiente video [diagrama de paquetes](#)

2.9. Vista física

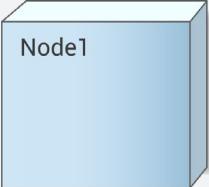
Permite desde el punto de vista un ingeniero en sistemas la descripción del mismo. En esta vista se muestra la conexión física de componentes como por ejemplo conexiones.

Diagrama de despliegue

Estos diagramas muestran las conexiones entre los elementos de *hardware* en forma de nodos. Además, en este diagrama se muestra la relación entre los nodos y elementos del *software*, así como los artefactos.

En estos diagramas podremos encontrar.

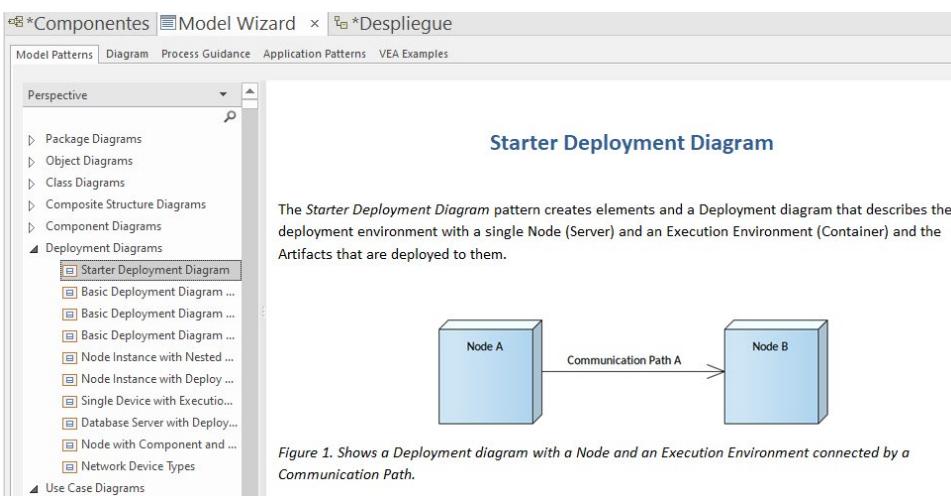
Tabla 14*Elementos del diagrama de despliegue*

Elemento	Descripción
	Nodo: elemento de <i>hardware</i> o <i>software</i> . Este puede estar representado de diferentes maneras (instancia-Estereotipos-Contenedores).
	Asociación: permite la relación entre dos nodos funciona de forma general como en todos los diagramas.

Nota. Sucunuta, M., Jaramillo, D., 2022

En la herramienta de Enterprise Architectur de la misma manera que todos los diagramas de UML que se pueden realizar, se presentan diferentes plantillas para el desarrollo de estos diagramas.

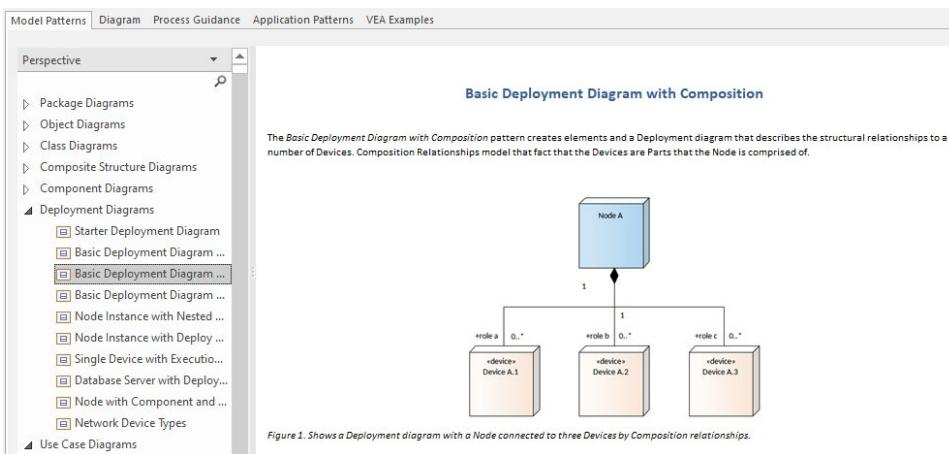
A continuación, en la figura 39 y 40 se presentan dos plantillas de cómo realizar el diagrama.

Figura 39*UML diagrama de despliegue-ejemplo 1.**Nota. Sucunuta, M., Jaramillo, D., 2022*

En la siguiente gráfica se muestra el diagrama de despliegue con una plantilla en jerarquía y un nivel de composición.

Figura 40

UML diagrama de despliegue-ejemplo 2.

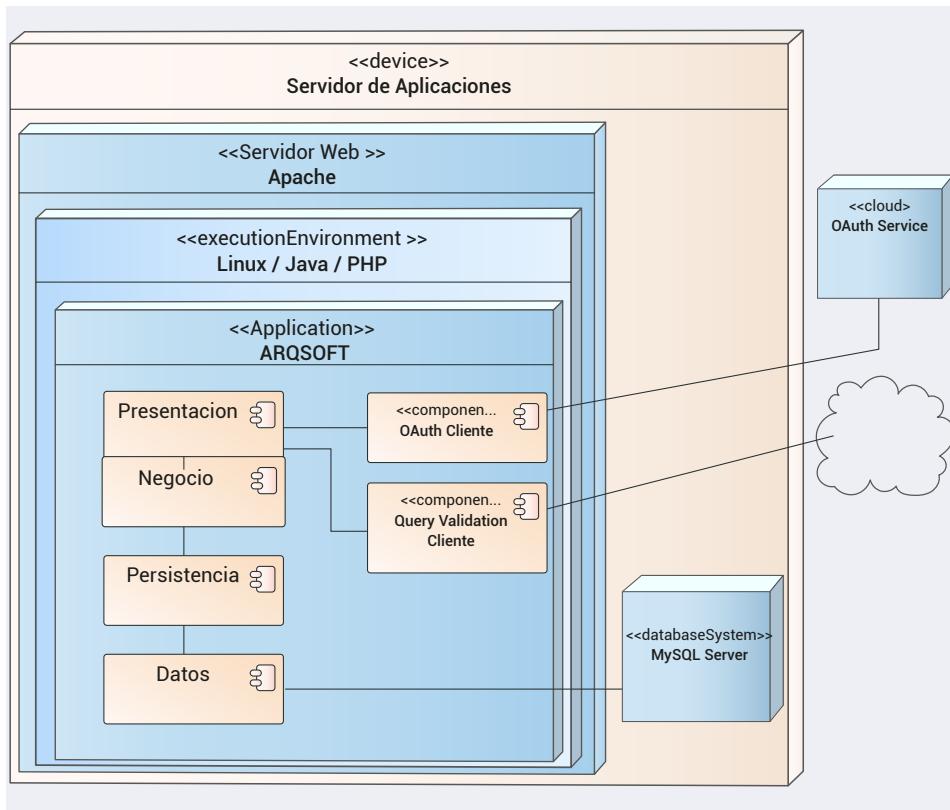


Nota. Sucunuta, M., Jaramillo, D., 2022

A continuación, un diagrama de despliegue sobre el caso de estudio <<Caso de estudio guía – Matriculación>>, donde también se podrá revisar otros ejemplos.

Figura 41

Diagrama de despliegue



Nota. Sucunuta, M., Jaramillo, D., 2022



Le invito a revisar el siguiente *link* donde puede encontrar un mejor detalle del [Diagrama de implementación](#)

De la misma manera, sobre el caso de estudio para el trabajo de este ciclo también puede revisar ejemplos del mismo.

- Resultado de aprendizaje 3**
- Transforma modelos de software en aplicaciones funcionales.

Contenidos, recursos y actividades de aprendizaje



Semana 13

Unidad 3. Del modelo a la implementación



La importancia de reflejar los modelos diseñados tras un análisis de la problemática que se resolverá con la etapa de construcción del sistema o en nuestro caso uno de los componentes, es muy importante.

Esto reflejará el esfuerzo que hemos realizado en la etapa de modelado y que se llegue a la construcción del sistema, es decir, vamos a llegar a la etapa de programación utilizando Java como el lenguaje base para la programación y la persistencia de la información.

En estas semanas de trabajo vamos a realizar este proceso y por su puesto la construcción de por lo menos una funcionalidad, la misma que cumpla con los estándares de programación que Ud. debe estar familiarizado para la realización de este proyecto web.

Empezaremos con la una explicación de los recursos que utilizaremos para la construcción, en el repositorio de la asignatura encontrará instaladores

de la versión de prueba o de lo contrario podrá utilizar los recursos que se encuentran instalados en nuestro laboratorio virtual:

- **El proyecto en Enterprise Architect** realizado durante el ciclo como base, ya sea el realizado con el profesor o el proyecto más personalizado seleccionado por el estudiante.

3.1. Generación de código a partir de los modelos

La primera parte de este trabajo será obtener el código base para la implementación de una determinada funcionalidad. Este código lo vamos a nombrar como modelo, puesto que estarán las clases en su versión inicial.

Es así que pasaremos desde el diagrama de clases de nuestro proyecto para la: generación y depuración del código que está basado en Enterprise Architect

Para ello vamos a partir del diagrama de clases que se haya realizado, donde deben constar:

- Atributos: con sus nombres y sus tipos.
- Operaciones.
- Relaciones: con su multiplicidad y la dirección de las mismas.

¿Tiene claro la diferencia de agregación y composición en código?

Vamos a poner en consideración algunos temas que se presentan en el diagrama de clases y deben estar reflejados en la generación del código desde este diagrama:

Navegabilidad

Es importante determinar la dirección que tiene cada relación.

- Si la flecha apunta de la ClaseA a la ClaseB, se lee como “ClaseA tiene una ClaseB” y se dice que la asociación o navegabilidad es unidireccional. Traducido a Java, esto significa que hay un atributo en la clase A que hace referencia a un objeto de la clase B.
- Si no dibujamos la flecha, se lee “ClaseA tiene una ClaseB y ClaseB tiene una ClaseA”, y se dice que la asociación o navegabilidad es

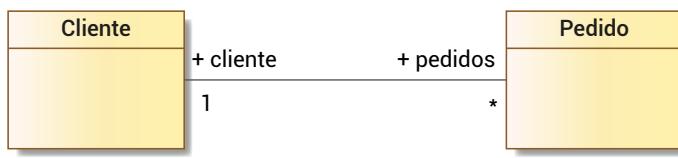
bidireccional. En Java, ambas clases tendrían atributos que se hacen referencia recíprocamente.

Ejemplo:

- Presentamos una asociación con las clases Cliente y Pedido. En el siguiente diagrama, Cliente guarda información sobre los pedidos y pedido tiene información sobre el cliente que lo realizó. La navegabilidad es, por tanto, bidireccional:

Figura 42

Asociación con las clases cliente y pedido



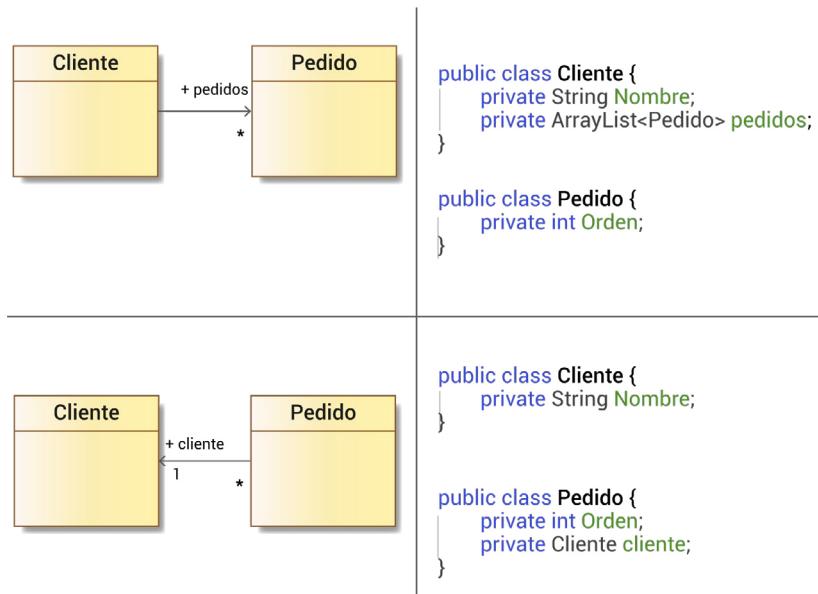
```
public class Cliente {  
    private String Nombre;  
    private ArrayList<Pedido> pedidos;  
}  
  
public class Pedido {  
    private int Orden;  
    private Cliente cliente;  
}
```

Nota. Sucunuta, M., Jaramillo, D., 2022

Ejemplo de asociación unidireccional

Figura 43

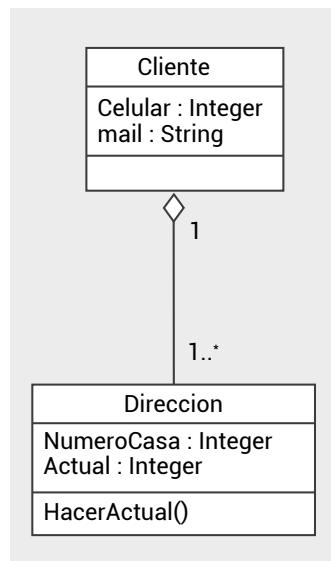
Ejemplo de asociación unidireccional



Nota. Sucunuta, M., Jaramillo, D., 2022

Agregación

Esta relación nos presenta que el objeto dirección existirá y que lo podremos tomar para relacionarlo con nuestro cliente.



En el código nos podemos fijar que estamos igualando el atributo dirección con un parámetro que llega llamado Mydireccion.

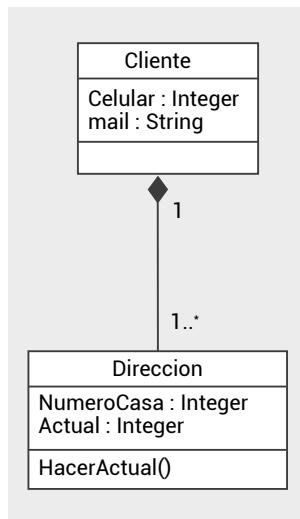
```
package CLA_tramil;

public class Cliente extends Personas {
    private String Celular;
    private String mail;
    private Direccion myDireccion;

    public Cliente(String Identificacion, String Apellidos, String Nombres, //padre
                  String Celular, String mail, Direccion myDireccion) { //hijo
        super(Identificacion, Apellidos, Nombres);
        this.Celular = Celular;
        this.mail = mail;
        this.myDireccion = myDireccion;
    }
}
```

Composición

La composición nos indica principalmente que el objeto dirección no podrá existir por sí solo, es decir que únicamente podrá existir si existe el objeto cliente. El constructor refleja eso, al momento de hacer new Direccion().



```
package CLA_tramil;

public class Cliente extends Personas {
    private String Celular;
    private String mail;
    private Direccion myDireccion;

    public Cliente(String Identificacion, String Apellidos, String Nombres, //padre
                  String Celular, String mail, //hijo
                  int casa, String telefono) { //dirección
        super(Identificacion, Apellidos, Nombres);
        this.Celular = Celular;
        this.mail = mail;
        this.myDireccion = new Direccion(casa, telefono);
    }
}
```

El código que se presenta la composicion y agregacion no está generando lo que está representado en el diagrama ¿Qué le falta? Lea la nota al pie¹

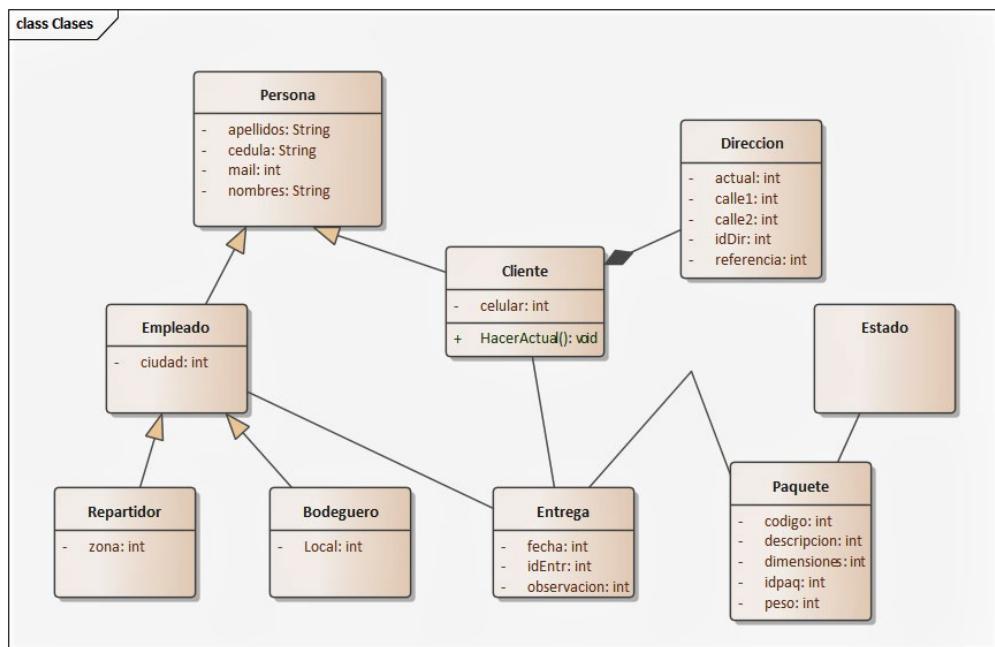
Recuerde la implementación de herencia.

Generación desde EA

Vamos a considerar el siguiente diagrama de clases:

Figura 44

Diagrama de clases

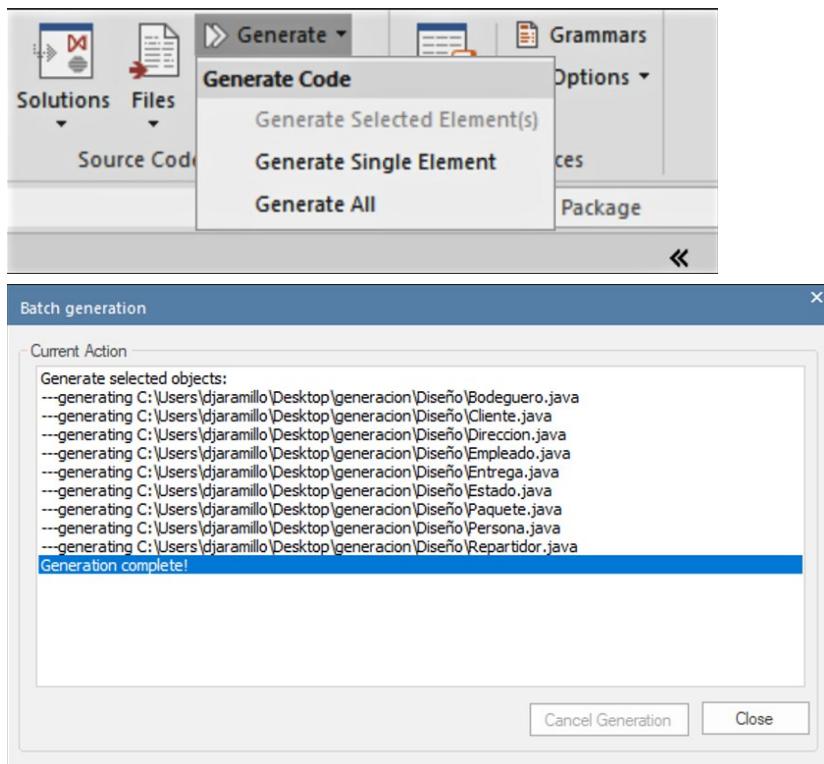


Nota. Sucunuta, M., Jaramillo, D., 2022

Podemos notar que existe relaciones de: herencia, composición y asociación, por lo que vamos a hacer el primer proceso de generación para poder determinar si lo que está representado y configurado en nuestro diagrama es realmente lo que queremos modelar o tenemos errores en la configuración de relaciones en esta herramienta.

¹ El constructor refleja una relación <<un cliente tiene una dirección>> 1 a 1 porque solo tenemos una dirección. Lo correcto es decir, el cliente tienen varias direcciones, para lo cual debemos colocar un array list de clientes.

El proceso para la generación lo podemos encontrar en la pestaña develop y luego en el ícono de generar, obteniendo las clases. Procederemos a generar todo el código y a continuación se presentará esta pantalla



Nota. Sucunuta, M., Jaramillo, D., 2022

Ahora vamos a analizar las mismas desde nuestro IDE

Este equipo > Escritorio > generacion > Diseño		
	Nombre	Fecha de modificación
ido	Bodeguero	1/8/2022 10:57
	Cliente	13/9/2022 9:16
Universidad Técnica P	ddd.eapx	2/8/2022 23:27
d Técnica Particular de	Direccion	1/8/2022 10:57
	Empleado	1/8/2022 10:57
o	Entrega	1/8/2022 10:57
s	Estado	1/8/2022 10:57
ntos	Paquete	1/8/2022 10:57
	Persona	1/8/2022 10:57
:	Repartidor	1/8/2022 10:57

Dónde se debe colocar, donde se generarán las clases, a partir de esto debemos revisar el resultado para realizar una corrección y obtener lo que se debe modelar. Este proceso puede resultar tedioso en los inicios, pero luego con el dominio de los temas se harán pocas generaciones.



Semana 14

Ya hemos llegado a tener el modelo de clases, este nos servirá para realizar la implementación de una funcionalidad.

3.2. Ambiente de desarrollo

Para la realización de esta actividad vamos a definir algunos elementos necesarios,



- Como motor de base de datos utilizaremos una versión de MySql standard instalada.
- Repositorio de trabajo y control de versiones: Gitgub, el mismo que permitirá la revisión de la actividad.
- IDE de desarrollo: Sprint tool suite² SpringBoot, como framework de implementación.
- El lenguaje base será JAVA.

Dentro del caso de estudio propuesto para esta asignatura hemos llegado .

No olvide tener claro algunos temas que ya fueron estudiados que a veces no consideramos el momento de realizar la implementación.

Debemos considerar también tener ya las clases generadas de forma correcta, esta validación la debe realizar cada uno de los estudiantes. No se

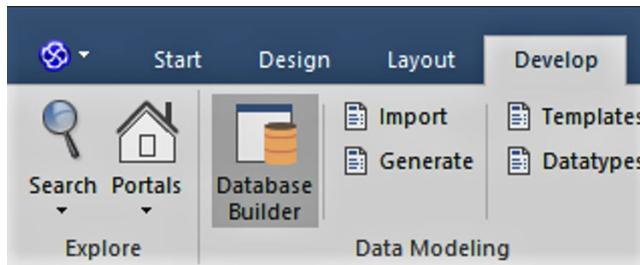
² En youtube puede buscar << Curso #SpringBoot - 1 Introducción>> encontrarán un curso sobre la herramienta donde se explica su instalación y configuración

asume porque la generación desde la herramienta no presentó errores, es lo correcto, se debe realizar la verificación correspondiente.

En la herramienta de trabajo se puede modelar todo lo relacionado con la base de datos, los beneficios que esta herramienta no son muy buenos, esto lo puede encontrar en:

Figura 45

EA- Construcción de la base de datos



Nota. Sucunuta, M., Jaramillo, D., 2022



Semana 15

3.3. Implementación y pruebas

En esta semana se realizará la implementación y pruebas de la funcionalidad que se ha trabajado.

En esta semana debemos probar la implementación realizada ya sea a través de una revisión manual donde veremos si la persistencia de los datos es correcta o a través de una herramienta para pruebas de funcionalidad.

Se debe considerar la elaboración de:

- Un *script* de creación de base de datos.
- *Script* de inserción para carga de datos iniciales en el proyecto.
- Actualizar su proyecto en el repositorio de GitHub, el mismo que permitirá compartir el código con sus compañeros.

- Detalle de la funcionalidad y plan de pruebas de la misma para realizar un trabajo colaborativo entre 2 o más grupos de trabajo.
- Considerar la realización de un video explicativo del trabajo de la funcionalidad para la realización de pruebas.



Semana 16

En la presente semana corresponde realizar la preparación para la prueba del segundo bimestre. La evaluación bimestral consiste en preguntas objetivas de la unidad 2 (Diagramas de comunicación, actividades, secuencia, componentes y paquetes) además de la unidad 3.

Sugiero realizar las siguientes actividades:

- Revisar los videos correspondientes a los temas del segundo bimestre.
- Revise los ejemplos presentados en las tutorías por el docente.
- Realice las lecturas de los recursos que se indican en los temas de este bimestre.



4. Glosario

UML: Unified Modeling Language a la traducción en español Lenguaje Unificado de Modelado.

MDA: (Model-Driven Architecture, Arquitectura basada en modelos). Es una propuesta para diseñar sistemas basados en el modelado del dominio.

EA: siglas de la herramienta Enterprise Architect que se utiliza para el desarrollo del caso práctico a través del Laboratorio Virtual.

MOO: son las siglas de Modelado Orientado a Objetos

POO: son las siglas de Programación Orientada a Objetos.

Kruchten: modelo diseñado por Philippe Kruchten para describir la arquitectura de un sistema software.

Enterprise Architect (EA): solución software para visualizar, analizar, modelar, probar y mantener todos los sistemas, software, procesos y arquitecturas.

CASE: (Computer Aided Software Engineering, Ingeniería de Software Asistida por Computadora). Conjunto de herramientas informáticas que apoyan en todos los aspectos del ciclo de vida del desarrollo del software.



5. Recursos

Recursos bibliográficos:

Libros:

- Joyanes, L., Zahonero, I. (2014). Programación en C, C++, Java y UML
Los autores realizan una importante relación entre los modelos UML y los lenguajes de programación. Esto permite al estudiante conocer el verdadero sentido de los modelos UML como parte del análisis y diseño.
- Unhelkar, B. (2018). Software Engineering with UML. CRC Press.
El texto presenta los conceptos básicos y desarrolla los modelos utilizando UML. Los conceptos son lo suficientemente claros que permiten relacionar con los ejemplos que se indican. Además, los casos que plantea son lo suficientemente didácticos que ayudan con el proceso de aprendizaje.

Recursos educativos

- López, P. Ruiz, F. Ingeniería de software I. Tema 2. Lenguaje Unificado de Modelado – UML. <https://ocw.unican.es/pluginfile.php/1403/course/section/1792/is1-t02-trans.pdf>.
- Lenguaje Unificado de Modelado: UML. OCW. <https://campusvirtual.ull.es/ocw/course/view.php?id=132>.
- ¿Cómo construir un diagrama de secuencia? Ejemplo de creación de objetos. <https://riunet.upv.es/handle/10251/83885>

Sitios web

- Una introducción a UML: https://developer.ibm.com/articles/an-introduction-to-uml/?mhsrc=ibmsearch_a&mhq=uml

- Sitio: Unified Modeling Language. <https://www.uml.org/>
- Sitio: Sparx systems. <https://sparxsystems.com/>
- Sitio: Biblioteca UTPL. <https://biblioteca.utpl.edu.ec/>



6. Referencias bibliográficas

- Aparx Systems. (2022). *Unified Modeling Languaje (UML)*. Enterprise Architect.
- Bhuvan, U. (2018). Software Engineering with UML. En *CRC Press Taylor & Francis Group*.
- Booch, G. (1996). Unified modeling language. *Performance Computing/Unix Review*, 14(13). https://doi.org/10.1007/978-3-319-64021-1_11
- Booch, G., Rumbaugh, J. & Jacobson, I. (2006). *El lenguaje unificado de modelado* (Segunda). Perason.
- Kruchten, P. (2014). *The 4+1 View Model of Architecture*.



7. Anexos

Caso de estudio.