

Gestión de requisitos

**Breve descripción:**

Este componente formativo aborda la ingeniería de requisitos y el levantamiento de requerimientos para prever el comportamiento del software. Utiliza casos de uso, historias de usuario y Lenguaje de Modelado Unificado para modelar funcionalidades e interacciones. Además, introduce herramientas CASE, esenciales para visualizar y construir software.

**Junio 2024**

Tabla de contenido

[Introducción 4](#_Toc168431935)

[1. Características del Lenguaje Unificado de Modelado UML 6](#_Toc168431936)

[Versión oficial de UML 7](#_Toc168431937)

[Características generales del UML 7](#_Toc168431938)

[2. Especificación y análisis de requisitos 10](#_Toc168431939)

[Diagrama de casos de uso 10](#_Toc168431940)

[Actor 11](#_Toc168431941)

[2.1. Relaciones de casos de uso 16](#_Toc168431942)

[Especificación de casos de uso 19](#_Toc168431943)

[2.2. Casos de uso reales (Prototipos) 21](#_Toc168431944)

[3. Historias de usuario 24](#_Toc168431945)

[Criterios de aceptación: 25](#_Toc168431946)

[4. Diagrama de actividades 31](#_Toc168431947)

[Elementos de un diagrama de actividades 31](#_Toc168431948)

[Ejemplo de diagrama de actividades 34](#_Toc168431949)

[5. Diagramas de clase 36](#_Toc168431950)

[Control de acceso 38](#_Toc168431951)

[Relaciones entre clases 39](#_Toc168431952)

[5.1. Asociaciones 39](#_Toc168431953)

[5.2. Herencia 43](#_Toc168431954)

[Visibilidad en las relaciones de herencia 47](#_Toc168431955)

[5.3. Agregación 47](#_Toc168431956)

[5.4. Composición 48](#_Toc168431957)

[6. Herramientas CASE 50](#_Toc168431958)

[Clasificación de las Herramientas CASE 51](#_Toc168431959)

[Síntesis 54](#_Toc168431960)

[Material complementario 55](#_Toc168431961)

[Glosario 56](#_Toc168431962)

[Referencias bibliográficas 57](#_Toc168431963)

[Créditos 59](#_Toc168431964)

Introducción

La especificación de requisitos es uno de los procesos más importantes en las fases de desarrollo de software. Constituye la fase inicial y es crucial para que el proyecto culmine exitosamente. Su objetivo principal es comprender qué esperan los usuarios y clientes del sistema, empleando un lenguaje natural para obtener una perspectiva clara y evitar errores costosos en las fases subsiguientes del desarrollo del producto.

El análisis de requerimientos propone un conjunto de técnicas e instrucciones que facilitan una representación precisa y completa de las funcionalidades que debe ejecutar un sistema para satisfacer los requerimientos de clientes y usuarios. Actualmente, algunas de las técnicas más utilizadas incluyen los diagramas de casos de uso, historias de usuario y diagramas de actividades.

Para la construcción de software, es fundamental la utilización de modelos, ya que representan la dirección y aspecto de una necesidad específica. Esta "necesidad" puede estar ya existente, en desarrollo o en fase de planificación. Los diseñadores del modelo deben levantar los requerimientos del producto, que pueden abarcar áreas como funcionalidad, rendimiento y fiabilidad. El modelo se estructura en varias vistas, cada una describiendo un aspecto específico del producto o sistema en desarrollo. Un modelo permite documentar la estructura y comportamiento de un sistema antes de ser codificado.

Este componente formativo presentará diversas técnicas y diagramas para el análisis y especificación de requisitos, con el fin de organizar adecuadamente los requerimientos. Esto permite una representación efectiva de cada una de las interacciones del usuario con el sistema, resultando en una especificación eficaz que reduce costos y riesgos en el desarrollo del software, y contribuye a alcanzar un producto de calidad.

# Características del Lenguaje Unificado de Modelado UML

Con el nacimiento de la Programación Orientada a Objetos (POO), se establecieron una variedad de lenguajes, métodos y simbologías utilizadas para detallar el desarrollo de diferentes sistemas. Indudablemente, esta situación generó desconcierto entre los programadores de software. Para solucionar este inconveniente, surgió el Lenguaje de Modelado Unificado (Unified Modeling Language, UML).

* **El lenguaje de Modelado Unificado UML**

Proporciona un conjunto de herramientas que facilitan el modelado (análisis y diseño) de sistemas orientados a objetos. Establece una serie de normas técnicas que representan de manera gráfica el camino y la estructura de un sistema determinado para programadores, analistas, desarrolladores, diseñadores, etc.

El UML es una de las herramientas más emocionantes en el mundo actual del desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas generar diseños que capturen sus ideas de forma convencional y fácil de comprender, facilitando la comunicación de estas a otras personas. (Schmuller, 2000, p.24)

* **Evolución del Lenguaje Unificado de Modelado UML**

El UML tuvo su origen a finales de los años 80, inicialmente basado en el método Booch, desarrollado por el diseñador de software Grady Booch. En 1994, el científico de la computación James Rumbaugh implementó el modelado de objetos en este método. En 1995, se añadieron a estas bases otros principios y metodologías, algunas de ellas introducidas por Ivar Jacobson. (Booch, 2006).

### Versión oficial de UML

La primera versión oficial, UML 1.0, se lanzó en 1997 bajo el liderazgo de Booch, Rumbaugh y Jacobson. Esta versión ofrecía una metodología que incluía varios estándares con el principal objetivo de asegurar la coherencia entre todos los diagramas. En 2005, se anunció la versión UML 2.0, cuyo contenido estaba principalmente orientado hacia la visión de la Programación Orientada a Objetos. Desde entonces, se han realizado nuevas versiones basadas en UML 2.0.

* Grady Booch, ingeniero de sistema e investigador, antes del 95.
* Jim Rumbaugh (OMT), antes del 95.
* Ivar Jacobson, UML 0.8 y 0.91 1996.

### Características generales del UML

Como se mencionó anteriormente, el UML es un lenguaje que proporciona un conjunto de reglas que facilitan la comunicación de un sistema mediante representaciones gráficas, indicando cómo realizar la creación y lectura de los modelos. Aunque existen muchas características del UML, según Booch (2006), se pueden resumir en las siguientes, las cuales se detallan a continuación:

1. **Visualizar**

UML facilita la representación de un sistema de forma gráfica, con el fin de que se pueda entender fácilmente por otra persona.

1. **Especificar**

UML permite detallar cuáles son las necesidades de un sistema previo a su construcción.

1. **Construir**

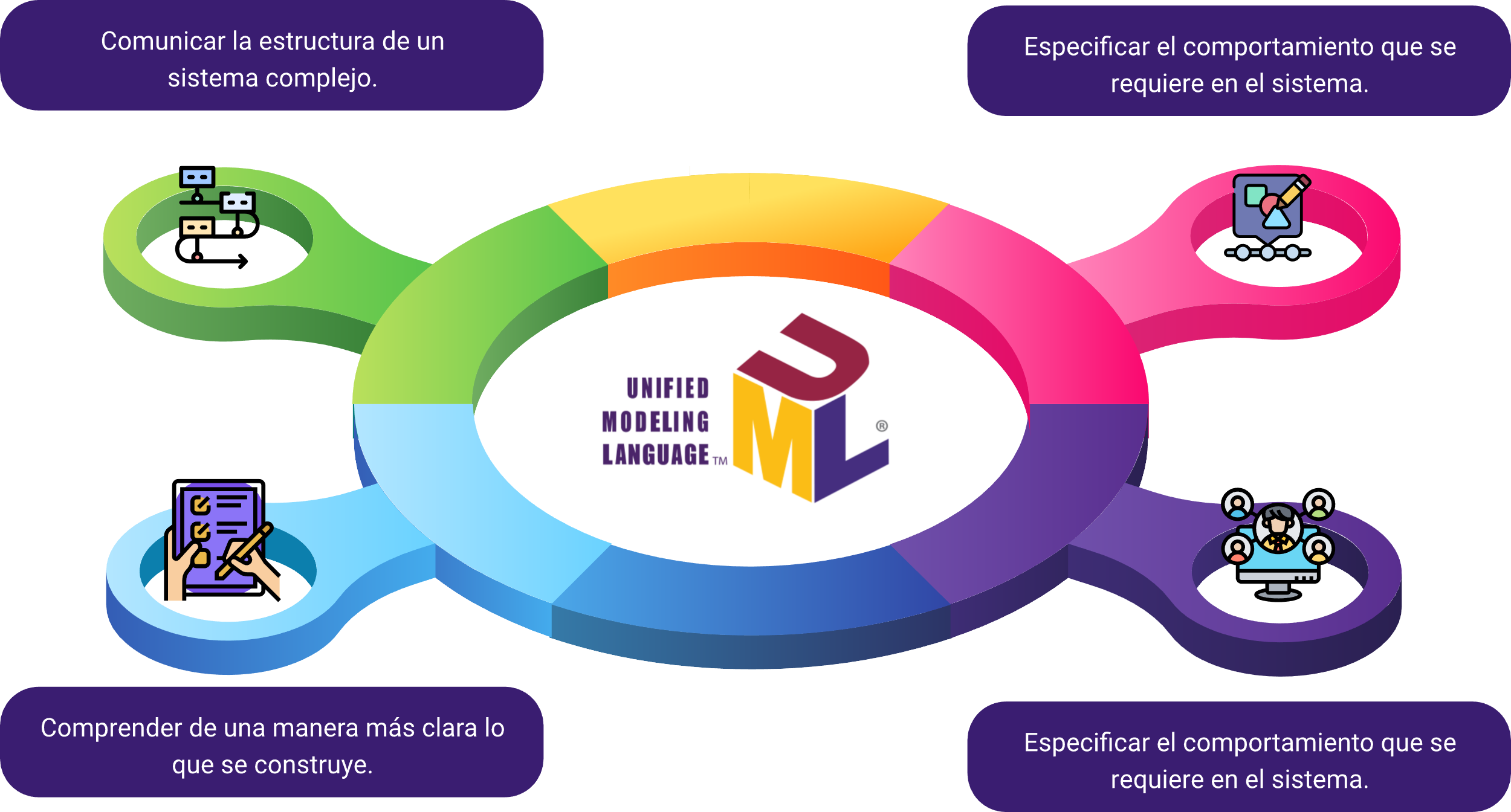
La construcción del sistema diseñado se realiza tomando los modelos especificados en el lenguaje UML.

1. **Documentar**

Cuando se diseñan las notaciones gráficas del sistema desarrollado, estas se utilizan como documentación. Esto permite brindar soporte para nuevas revisiones o empalmes, facilitando que los nuevos desarrolladores comprendan el sistema.

La versión 1.0 de UML fue lanzada en enero de 1997 y desde entonces ha sido utilizado con éxito en la construcción de sistemas para diversas industrias alrededor del mundo, incluyendo hospitales, bancos, comunicaciones, aeronáutica y finanzas, entre otras. UML es actualmente el lenguaje de modelado de sistemas de software más conocido y utilizado, permitiendo visualizar, especificar, construir y documentar sistemas con una perspectiva orientada a objetos (Rumbaugh et al., 2004).

1. Modelo para la construcción de software UML



UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Entre los tipos de diagramas que ofrece UML, se encuentra el diagrama de clase, que es parte de los diagramas de estructura. Estos últimos enfatizan los elementos que deben existir en el sistema de modelado.

# Especificación y análisis de requisitos

La función más importante de la especificación de requisitos es servir de intermediario para que los clientes, los analistas de requisitos, los desarrolladores y los usuarios puedan comunicarse eficazmente. En esta fase es fundamental recolectar tanto los requerimientos del usuario y del cliente como los del software a desarrollar, para lograr la satisfacción del cliente. Por ello, es crucial el uso de técnicas que permitan representar de manera visual e intuitiva estos requerimientos.

Cuando se realiza la recolección de requerimientos de un cliente, el UML facilita el modelado de estos a través de sus casos de uso. Los stakeholders (partes interesadas) que están involucrados en el producto son incluidos en el modelado con las funcionalidades requeridas (casos de uso), que a su vez son modelados incluyendo relaciones y asociaciones jerárquicamente organizadas entre ellos.

Los casos de uso y sus actores son quienes especifican los requerimientos de un cliente, detallando las expectativas que cada uno tiene del sistema, sin centrarse únicamente en la funcionalidad que se implementará. Los análisis de requisitos se realizan para diversos procesos, no necesariamente solo para desarrollos de software. Entre los diagramas más relevantes y utilizados en UML se encuentran los diagramas de casos de uso.

### Diagrama de casos de uso

Los casos de uso permiten describir cada una de las funciones de las aplicaciones a desarrollar desde el punto de vista del usuario. Su utilidad se centra en expresar las funcionalidades que el producto de software debe ejecutar y, definir a las personas que serán responsables de realizar estas funcionalidades. Esta técnica implica la creación de un diagrama con su respectiva notación. El caso de uso representa el comportamiento del software en la interacción con el usuario, con el objetivo de que este alcance una meta específica. Por tanto, un caso de uso se compone de los elementos que forman parte de su notación.

* **Componentes de los diagramas de casos de uso**

La notación de un diagrama de caso de uso incluye tres componentes principales que facilitan la representación simbólica de las funcionalidades del sistema, así como de las personas involucradas y las interacciones entre ellas.

Estos componentes son: los actores, que son los usuarios o roles que interactúan con el sistema; los casos de uso, que describen las funciones del sistema; y las relaciones, que especifican las conexiones entre actores y casos de uso.

### Actor

Define un rol desempeñado por un usuario, representa a una persona o grupo de personas que interactúan con un software. Se simboliza mediante la figura de un "stick man", junto al nombre del actor, que es obligatorio. Los actores se utilizan para identificar la clase de usuario que participará en el sistema y será responsable de ejecutar determinadas funciones. Por lo general, los nombres de los actores comienzan con mayúscula.

* **Principales**

Son actores especificados para identificar los objetivos de los usuarios, los cuales son guiados por los casos de uso.

* **De apoyo**

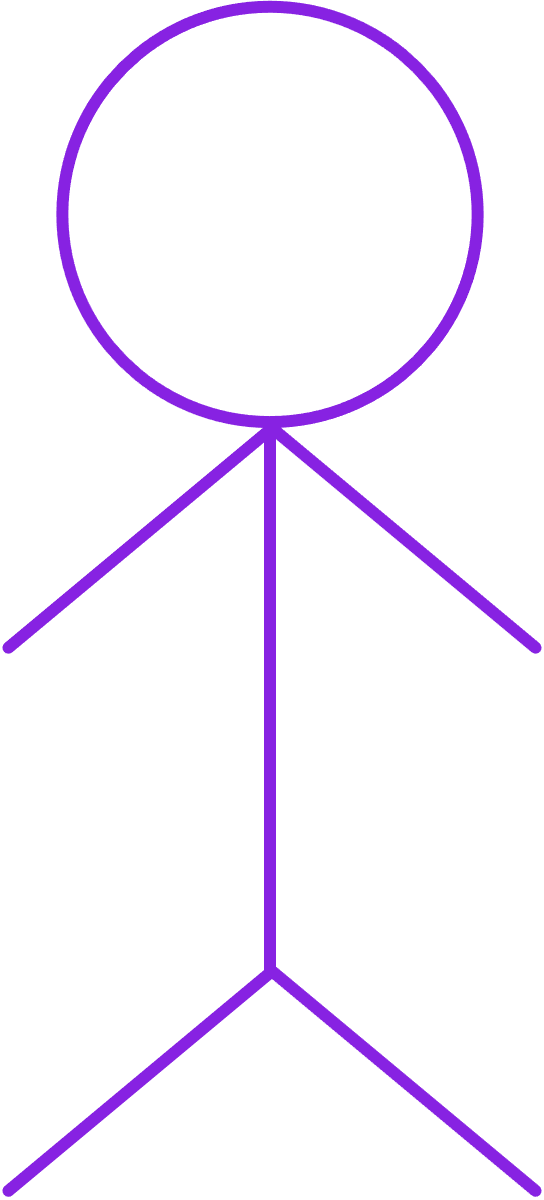
Proveen servicios al sistema informático y son definidos para esclarecer las interfaces externas.

* **Los pasivos**

Se interesan en el funcionamiento de los casos de uso, aunque no son considerados actores principales.

En la figura que se presenta a continuación, se identifica cómo se representa de manera gráfica un actor en un UML, al cual también se le adiciona el nombre del actor correspondiente para dar mayor claridad en el proceso.

1. Representación gráfica de un actor en UML



Además, es importante considerar que los actores no son necesariamente personas, aunque en los diagramas de casos de uso se representen mediante formas humanas. En este contexto, los sistemas externos también cumplen el rol de actores, pues pueden requerir información del sistema real o interactuar con él, como ocurre con una base de datos.

Al determinar los casos de uso, resulta esencial identificar al actor. Este término se refiere al usuario cuando desempeña un rol específico, implicando que dicho usuario ejecuta los casos de uso. Dado lo anterior, cada actor tiene la capacidad de participar en diversos casos de uso y, a su vez, cada caso de uso puede involucrar a diferentes actores. Para identificar a un actor, basta con visualizar los roles que interactúan dentro del sistema.

**Ejemplo 1:**

Para explicar cada uno de los elementos de un caso de uso, se presenta el siguiente requerimiento:

Una empresa solicita un sistema que facilite el proceso de generación de facturas de venta para comerciantes. Estas facturas deben ser enviadas a través de un sistema externo encargado del proceso de facturación.

Los actores identificados en los requisitos del ejemplo 1 son el comerciante y el sistema externo.

#### Relación

La interacción entre un actor y un caso de uso se denota mediante una relación, simbolizada por una línea o una flecha. Se describen varios tipos de relaciones según los actores mencionados anteriormente.

* La relación de comunicación entre un actor y un caso de uso puede ser activa o pasiva.
* Un actor activo inicia la ejecución, con la flecha apuntando hacia el caso de uso correspondiente.
* Un actor pasivo, en cambio, no inicia la ejecución y la flecha apunta hacia él.
* El tipo de comunicación o relación entre actores y casos de uso establece una asociación directa.
* Esta asociación especifica que el actor es el ejecutor del proceso descrito.

1. Representación gráfica de una relación de caso de uso en UML



Continuando con el ejemplo 1 se identifican las siguientes relaciones de comunicación:

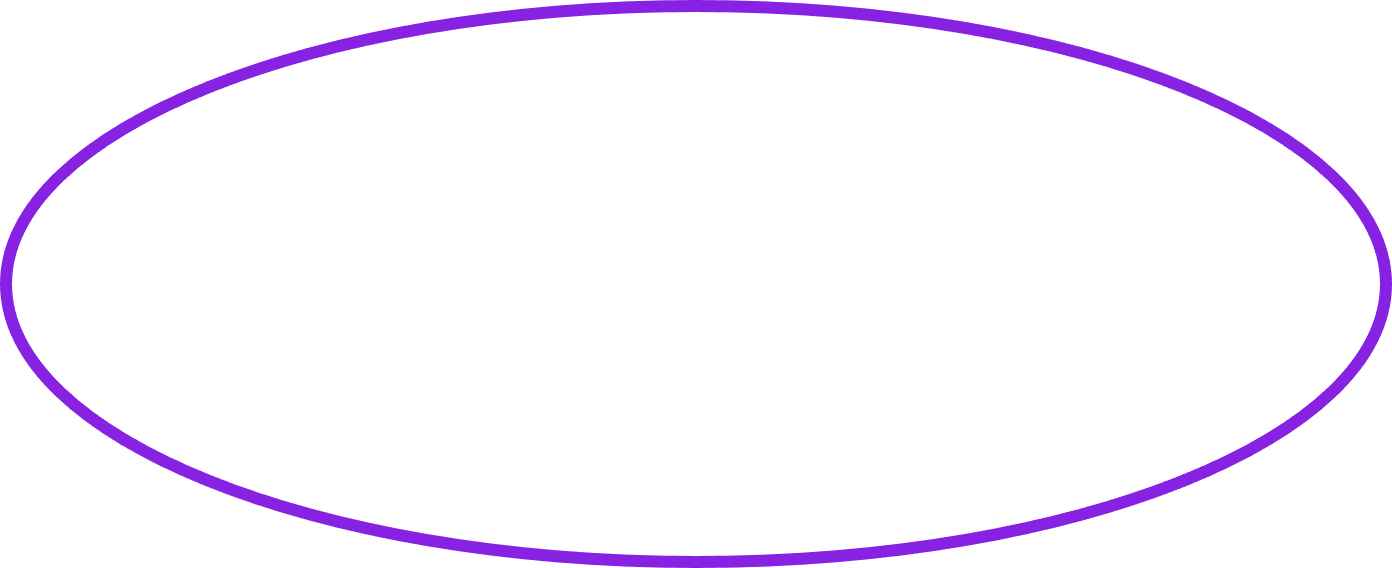
* Entre el comerciante y la generación de la factura.
* Entre el comerciante y el envío de la factura.
* Entre la factura y el sistema externo.

#### Caso de uso

Los casos de uso se identifican mediante los verbos o acciones que el actor ejecuta o ejecutará. Estas acciones se describen y representan de la siguiente manera:

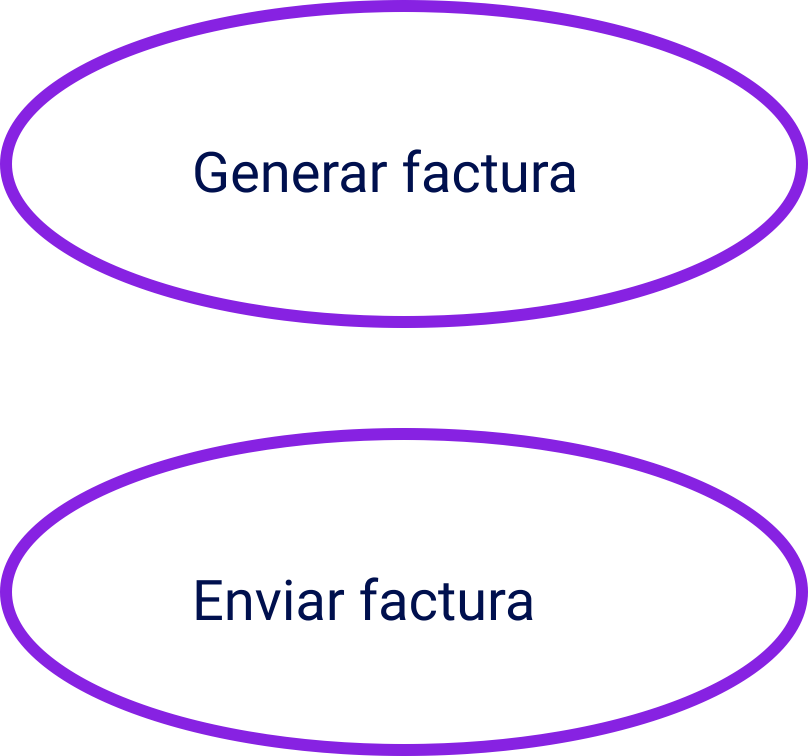
* Se simbolizan con un óvalo, indicando una función que el sistema debe proporcionar.
* El caso de uso debe ser de fácil lectura e interpretación, tanto para los desarrolladores como para el cliente. Para el nombre, se recomienda utilizar un verbo en infinitivo acompañado de un complemento.

1. Representación caso de uso



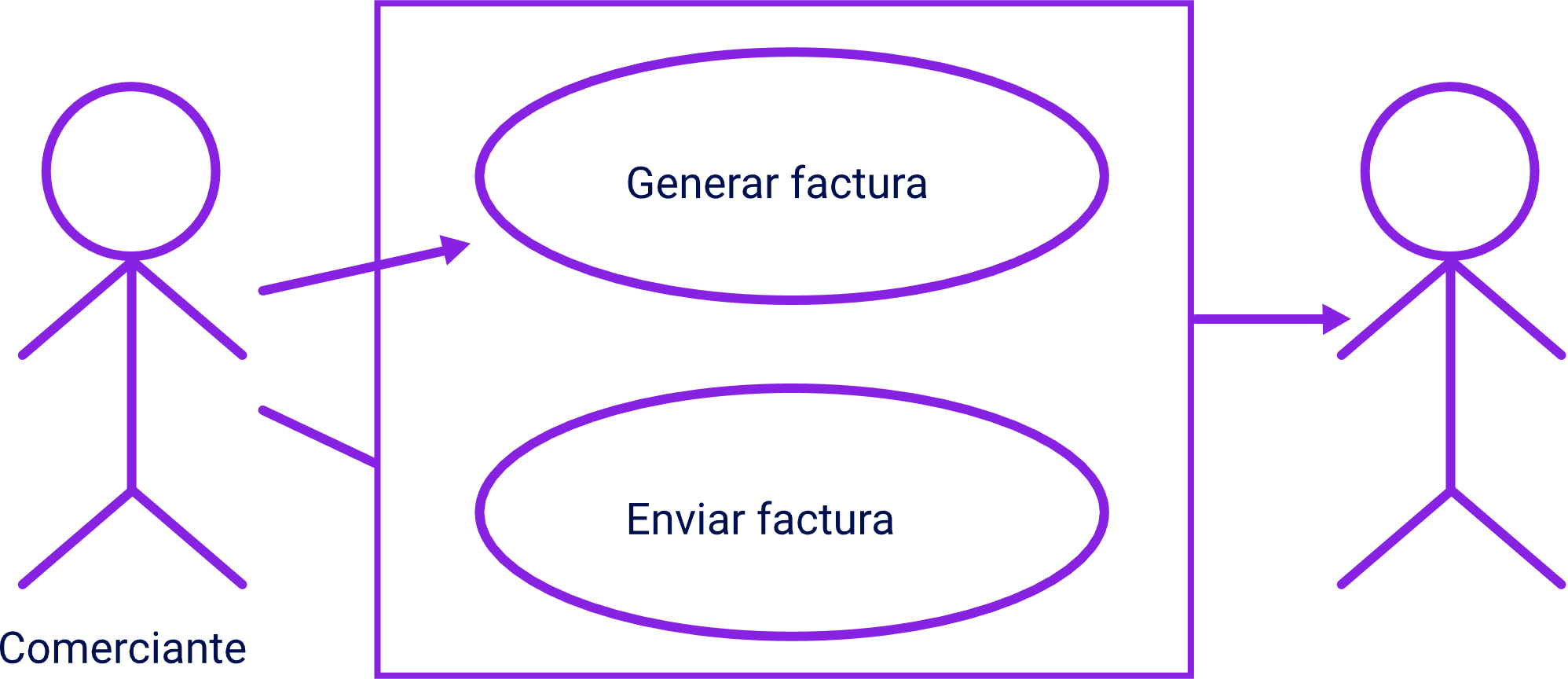
Los casos de uso que se identifican para el ejemplo 1 son: generar factura y enviar factura, los cuales se ven representados gráficamente en la siguiente figura.

1. Ejemplo caso de uso



A continuación, se presenta un ejemplo de la representación gráfica del diagrama de caso de uso, que incluye los actores, relaciones y el caso de uso del problema planteado en el Ejemplo 1.

1. Representación ejemplo 1



Por otro lado, los pasos para diseñar el diagrama de casos de uso del sistema de ventas, mostrado en la figura anterior, están detallados a continuación:

* Se identifican los casos de uso según requerimientos, para el caso del ejemplo son: (generar factura, enviar factura).
* Se identifican los posibles actores que interactuaran con el sistema: para el caso del ejemplo: (comerciante y sistema externo).
* Se identifican las relaciones de comunicación entre comerciante y los casos de uso, generar factura y enviar factura.

Además de la relación básica de comunicación mostrada anteriormente, existen otras relaciones en los casos de uso que detallan con mayor precisión las interacciones entre ellos.

## Relaciones de casos de uso

Durante el proceso de desarrollo, se pueden definir diferentes casos de uso que no son independientes entre sí. Es posible establecer relaciones de dependencia mutua entre los casos de uso. Las principales relaciones consideradas por UML son:

* **Inclusión (include)**

Se emplea para extraer comportamientos comunes de un caso de uso. Un caso de uso puede incluir a otro.

* **Extensión (extend)**

Utilizada para ampliar las funcionalidades de un caso de uso existente, es especialmente útil para incorporar nuevos requisitos del sistema.

* **Generalización (generalization)**

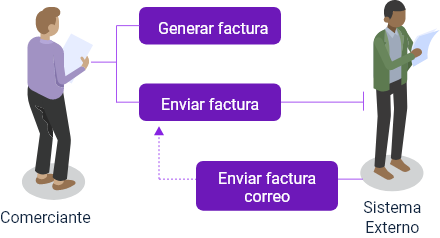
Esta relación ocurre entre un caso de uso general y uno más específico, donde el específico hereda propiedades del caso de uso base.

Tomando como referencia el Ejemplo 1, con el objetivo de perfeccionar el diagrama de casos de uso, se pueden introducir nuevos casos de uso que ilustren las relaciones mencionadas anteriormente.

* **Ejemplo extends**

Se establece la relación extends cuando un caso de uso es similar a otro, pero incluye funciones adicionales. Por ejemplo, en el caso mencionado anteriormente, el caso de uso enviar factura podría ampliarse a enviar factura por correo electrónico. Ambos casos implican la misma acción básica de enviar la factura; sin embargo, en la extensión, además se envía un correo electrónico con la factura. Esto se representaría de la siguiente manera:

1. Relación extends

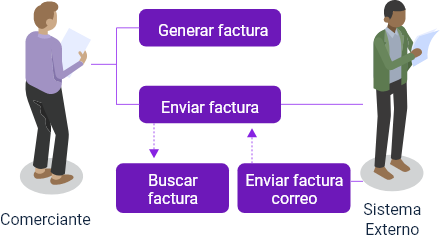


La relación se interpreta así: el caso de uso enviar factura por correo extiende del caso de uso enviar factura.

* **Ejemplo include**

Otro aspecto importante a considerar se presenta en ejemplo include; esta relación se define cuando parte del comportamiento o acción es común en más de un caso de uso, eliminando la necesidad de duplicar dichas acciones. Por ejemplo, en el caso del Ejemplo 1, se añade el siguiente requisito funcional: antes de enviar la factura, el usuario puede realizar una búsqueda de facturas para seleccionar y enviar. Esto hace necesario incluir la relación include. La representación sería la siguiente:

1. Relación include



### Especificación de casos de uso

La especificación de casos de uso se refiere al proceso de descripción detallada de cada caso, incluyendo los flujos y eventos que interactúan con el sistema y los actores involucrados. Generalmente, el analista encargado del diseño identifica los requerimientos y los documenta.

El propósito de la especificación de casos de uso incluye:

* Describir la interacción de un actor con el sistema.
* Explicar el rol que desempeñan los actores en su interacción con el sistema y sus flujos.
* Detallar los casos de uso.

A continuación, se presenta las partes esenciales y las directrices básicas para la documentación de casos de uso.

1. Plantilla de casos de uso

| Componente | Descripción de una especificación de caso de uso |
| --- | --- |
| Nombre del caso de uso | Indica el nombre del caso de uso. Normalmente, el nombre expresa el resultado objetivo y observable del caso de uso, como por ejemplo "Imprimir factura" en el caso de un sistema de ventas. |
| Autor | Persona que diseña el caso de uso. |
| Fecha | Aquí se diligencia la fecha en la que se realizó el caso de uso. |
| Descripción | Se realiza una corta descripción del caso de uso. |
| Actores | Describe los nombres del actor o actores del caso de uso. |
| Precondiciones | Son aquellas condiciones que deben existir para que se cumpla el caso de uso. |
| Flujo normal | Son los pasos normales de ejecución del caso de uso. |
| Flujo alternativo | Flujos alternativos de ejecución de casos de uso. |
| PosCondiciones | Son aquellas condiciones que se cumplen cuando finaliza un caso de uso. |

A continuación, se presenta un ejemplo de especificación de casos de uso para el caso de uso “Crear Foro” utilizando la plantilla propuesta posteriormente.

1. Descripción de una especificación de caso de uso ejecutado

| Componente | Descripción de una especificación de caso de uso |
| --- | --- |
| Nombre del caso de uso | Crear foro |
| Autor | Mario Vera Cruz |
| Fecha | 20/09/2021 |
| Descripción | Crea un nuevo mensaje. |
| Actores | Alumno/Docente. |
| Precondiciones | Al iniciar el usuario debe estar autenticado en el sistema. |
| Flujo normal | 1. El actor da clic en la opción nuevo mensaje.  2. Se muestra un área de texto para escribir el mensaje.  3. El actor introduce el mensaje.  4. El sistema verifica datos y los almacena.  5. El profesor acepta y publica. |
| Flujo alternativo | 4a. Se hace una comprobación de los datos, si no son correctos muestra mensaje de corrección.  4b. El profesor rechaza el mensaje de modo que no es publicado sino devuelto. |
| PosCondiciones | Se publica y almacena el mensaje satisfactoriamente. |

## Casos de uso reales (Prototipos)

El propósito principal de los casos de uso reales es detallar el proceso de un sistema de información descrito mediante un caso de uso, lo que incluye la interacción de objetos y la definición de las transacciones de las interfaces y clases de los diferentes procedimientos secundarios de diseño. Por lo tanto, es fundamental completar los entornos recolectados del análisis con las clases de diseño correspondientes, sin dejar de lado las restricciones del entorno tecnológico.

Los casos de uso reales representan el diseño preciso del caso de uso desde una tecnología específica. Por ejemplo, si se trata de un diseño gráfico para el usuario, se incluirá el diseño de las ventanas y una descripción de la interacción con los objetos de la interfaz. Los casos de uso reales proporcionan diseños de pantalla y explicaciones sobre la interacción entre los objetos de la interfaz.

Para aclarar las definiciones anteriores, se utiliza un ejemplo de un sistema de ventas que se presentará posteriormente. En este ejemplo, se realiza el diseño del caso de uso real y se describe la interacción entre los artefactos del diseño. A continuación, se presenta una tabla con los datos de un ejemplo para un sistema de ventas:

1. Sistema de ventas

| Caso de uso adquirir productos | Adquirir productos |
| --- | --- |
| Actores | Cliente (quien inicia el proceso), Cajero. |
| Tipo | Primario, real. |
| Descripción | • En el almacén de ropa el cliente se dirige a la caja registradora con los productos a comprar.  • El empleado-cajero hace el registro de los productos que el cliente compra y recibe el dinero en efectivo.  • Al finalizar el proceso, el cliente se retira con los artículos comprados. |

A continuación, se presenta el ejemplo del sistema de ventas:

1. Ejemplo Sistema de ventas



1. El cliente se acerca a la caja registradora con los artículos que desea llevar.
2. El cajero ingresa el código del producto en A de la Ventana-1.

* Opcional: Si el cliente tiene más de un producto, el cajero puede registrar la cantidad en E.
* Una vez finalizado el registro de todos los productos, se presiona H.

1. El sistema almacena los productos registrados.

* La descripción y el precio del producto se muestran en B y en F de la Ventana-1.

1. El cajero presiona el botón I para dar por finalizada la venta.
2. El sistema realiza el cálculo y muestra en C el total de la venta.

# Historias de usuario

Las historias de usuario son una forma sencilla de representar los requisitos de un sistema de información. Se crean en una o dos oraciones y se redactan en un lenguaje común y legible para el usuario. Estas historias, conocidas como User Stories, se han convertido en un estándar ampliamente utilizado en la definición de requisitos. Surgieron en XP (1999) y fueron impulsadas de manera concluyente por Mike Cohn en su libro "User Stories Applied: For Agile Software Development" (2004), estableciendo el patrón para definirlas. La simplicidad es la razón principal de su popularidad. Las historias de usuario están compuestas por tres elementos:

* Cómo [Rol del usuario]
* Quiero [Objetivo]
* Para poder [Beneficio]

A continuación, se hace la descripción de los elementos de una historia de usuario según Cohn, M. (2018):

1. **Tarjeta (Card)**

Es una frase que sintetiza una necesidad dada en una conversación entre el usuario y el dueño del producto. Escrita en una tarjeta, comúnmente sigue el formato rol-objetivo-beneficio. Su estructura es:

* Como [tipo de usuario] -> ¿quién?
* Quiero [necesidad] -> ¿qué?
* Para [beneficio esperado] -> ¿para qué? Con esta estructura simple de tres líneas se identifican: ¿Quién? (roles de valor), ¿Qué? (problema a resolver) y ¿Para qué? (objetivo a alcanzar).

1. **Conversación**

Es el proceso de comunicación entre las personas que requieren las necesidades (dueño del producto) y las personas que saben cómo solucionarlas (desarrolladores). Se realiza mediante preguntas y utilizando recursos como gráficas y prototipos para clarificar el contenido de las historias de usuario.

1. **Confirmación**

Es la verificación del acuerdo con relación a lo que se quiere construir. Permite validar si se terminó o no, o si se cumplió con lo esperado. Esta confirmación recibe el nombre de criterios de aceptación, los cuales ayudan a tener claro el contexto de la historia de usuario y a determinar fácilmente si está terminada o no.

A continuación se presenta un ejemplo, el usuario desea comprar una boleta para una película en el cine. los criterios de aceptación que se pueden establecer son:

### Criterios de aceptación:

1. Cuando se agoten las boletas, se debe mostrar un mensaje informando que no hay disponibilidad para la venta.
2. El sistema debe validar la edad del comprador para permitir la venta de boletas según la clasificación de la película.
3. Tras un pago en efectivo, el sistema debe mostrar el valor del cambio a devolver; si no hay cambio, deberá mostrar cero.

Cuando se realizan historias de usuario es muy importante cumplir con las siguientes reglas:

* Ser concisa
* Concreta
* Realizar criterios de aceptación claros y específicos

Así como estas reglas son necesarias, también lo es una buena redacción de las historias de usuario. Por este motivo, a continuación se presenta un método que facilita mejorar la calidad en la escritura de una historia de usuario: el método “Invest”.

**Método invest**

A continuación, se hace una descripción de cada uno de los elementos que intervienen en este método:

* **Independiente**

Las historias de usuario deben ser lo más independientes posible para evitar dependencias.

* **Negociable**

Las historias de usuario deben ser flexibles y negociables, siempre en acuerdo con el cliente.

* **Valiosa**

Es crucial que cada historia de usuario aporte valor al cliente. Idealmente, el cliente debería participar activamente en su redacción.

* **Estimable**

La estimación de la historia debe ser precisa y realizable, teniendo en cuenta el tamaño y la complejidad, para facilitar la planificación por parte del equipo.

* **Small (Pequeña)**

El tamaño de la historia de usuario debe ser pequeño para simplificar y mejorar la precisión de la estimación. Es ideal que se pueda completar en periodos cortos, como días o semanas.

* **Testeable (Comprobable)**

La historia de usuario debe ser comprobable, con criterios claros de aceptación para considerarse completada.

Al utilizar este método, se garantiza que las historias de usuario sean independientes entre sí, facilitando así su planificación y desarrollo sin sobrecargarlas con detalles técnicos que puedan limitar las negociaciones entre clientes y desarrolladores.

**Ejemplos**

A continuación, se presentan algunos ejemplos prácticos de historias de usuario aplicando el método invest:

1. **Historia de usuario: “Crear Usuario”**

En seguida, se presenta dos opciones:

Primera opción:

* **Rol**: como cliente de un supermercado.
* **Objetivo**: quiero hacer compras mediante una aplicación web.
* **Beneficio**: para poder comprar sin salir de casa.
* **Comentario**: en esta historia de usuario, se identifica claramente lo que el cliente desea hacer, aunque no se especifican los detalles de cómo se realizarán las compras en línea. Estos detalles podrían aclararse en reuniones posteriores. Esta historia cumple con las características del método INVEST: es sencilla, clara y satisface una necesidad específica, facilitando así la estimación del tiempo de desarrollo.

Segunda opción:

* **Rol**: como cliente de un supermercado.
* **Objetivo**: quiero acceder al registro mediante un formulario que me permita escribir mi correo electrónico y asignar una contraseña.
* **Beneficio**: para asegurar mi privacidad.
* **Comentario**: esta historia de usuario contiene especificaciones excesivas en la sección de objetivo y no clarifica suficientemente el beneficio. Se recomienda simplificar y aclarar el propósito para alinearse mejor con los criterios de INVEST.

1. **Ejemplo de una historia épica**

* **Rol**: como cliente de un supermercado.
* **Objetivo**: quiero ingresar a una página web.
* **Beneficio**: para hacer compras en línea.
* **Comentario**: esta historia no es independiente y se considera una Épica, porque es muy amplia y puede subdividirse en múltiples historias más pequeñas y manejables. Para visualizar mejor una historia de usuario Épica, se recomienda consultar recursos de aprendizaje específicos sobre este tema.
* Hacer un registro
* Un ingreso
* Seleccionar productos
* Pagar, etc.

**Plantillas de historias**

Para visualizar una historia del usuario épica, lo invitamos a consultar el PDF "**Plantillas de historias**", el cual se encuentra en la carpeta Anexos.

Por lo anterior, tampoco se cumpliría con la característica de estimación, ya que no se podría determinar cuándo terminaría la historia. Como se identificó en los ejemplos anteriores, el método INVEST nos proporciona reglas para optimizar la redacción de las historias de usuario, permitiendo así satisfacer las necesidades reales del cliente.

Para realizar una buena gestión de proyectos, es habitual en las metodologías ágiles utilizar una serie de herramientas para crear las historias de usuario, tales como:

* Trello
* Jira Software
* Taiga
* Excel

1. **Plantillas de historias de usuario**

Las plantillas de historias de usuario pueden adaptarse según las necesidades específicas sin perder la flexibilidad y sencillez que caracteriza a una historia de usuario. Los equipos de desarrollo colaboran utilizando herramientas en línea para compartir y trabajar conjuntamente en las plantillas. A continuación, se presenta un ejemplo del uso de una plantilla de Excel para la gestión de productos, descargada de la web.

1. Ejemplo plantilla historia de usuario

| ID de la historia | Rol | Funcionalidad | Razón/Resultado | Criterio de aceptación |
| --- | --- | --- | --- | --- |
| Id.01 | Como cliente | Quiero ver una lista de categorías de productos. | Para poder realizar búsquedas de productos por categorías. | Despliegue de listado de categorías a seleccionar con productos asociados. |
| Id.02 | Como cliente | Quiero ver una lista de precios. | Para poder elegir el producto más barato. | Despliegue una lista ordenada de productos por precio de menor a mayor. |

# Diagrama de actividades

Estos son diagramas de comportamiento que se utilizan para representar una sucesión de actividades. Explican el flujo de operaciones desde el punto de inicio hasta el final, definiendo una variedad de caminos de decisiones en el desarrollo de eventos que abarca una actividad. Según Booch, G. (1994), estos diagramas permiten visualizar un caso de uso específico a un nivel más preciso, ilustrando el flujo de actividades definidas en un sistema.

A continuación, se describen los beneficios del diagrama de actividades:

Beneficios:

* Explica la lógica de un algoritmo.
* Representa cada uno de los procesos realizados en los diseños de casos de uso en UML.
* Instruye procesos entre el sistema y los usuarios en un flujo de trabajo o negocios.
* Aporta claridad, optimizando y facilitando los casos de uso complejos.

### Elementos de un diagrama de actividades

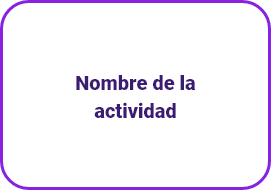
Un diagrama de actividades contiene fundamentalmente los siguientes elementos:

1. **Actividad**

Es la descripción o el detalle de una sucesión de conductas parametrizadas, simbolizada como un flujo de acciones ordenadas. El modelado de este flujo se realiza mediante nodos de actividad que se conectan a través de flujos de control. Las actividades pueden generar jerarquías de peticiones, ya sea llamando a diferentes actividades o, como último recurso, solventando operaciones propias. La representación gráfica de las actividades se realiza mediante un rectángulo con bordes redondeados, dentro del cual se especifica el nombre de la actividad.

Para nombrar las actividades se usan verbos del modelo de negocio como, por ejemplo:

1. Nombre de la actividad



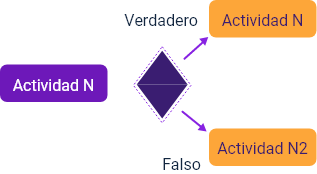
* Buscar factura
* Realizar pago
* Renovar lista

1. **Flujo entre actividades**

Este permite realizar los enlaces entre objetos y nodos dentro de una actividad, incorporando flujos de objetos y flujos de control. La representación gráfica de estos flujos se realiza mediante una flecha con punta abierta, que indica el orden en la ejecución de las actividades. Frecuentemente, se añade una descripción en la flecha para facilitar la comprensión.

Cuando los flujos de actividades incluyen condiciones para su ejecución, estos se representan mediante un rombo y se denominan nodos de decisión.

1. Flujo entre actividades

****

1. **Nodo inicial**

Como su nombre lo indica, el nodo inicial se encarga de empezar un flujo cuando se llama a una actividad. En cada diagrama existe un solo nodo inicial, el cual se representa con un círculo pequeño.

1. Nodo inicial

La figura representa la grafica de un nodo inicial.

1. **Nodo final**

También es un nodo de control final. Su función es detener todos los flujos de una actividad. En una actividad pueden existir más de un nodo final. Este se representa con un rectángulo, como se presenta a continuación:

1. Nodo final

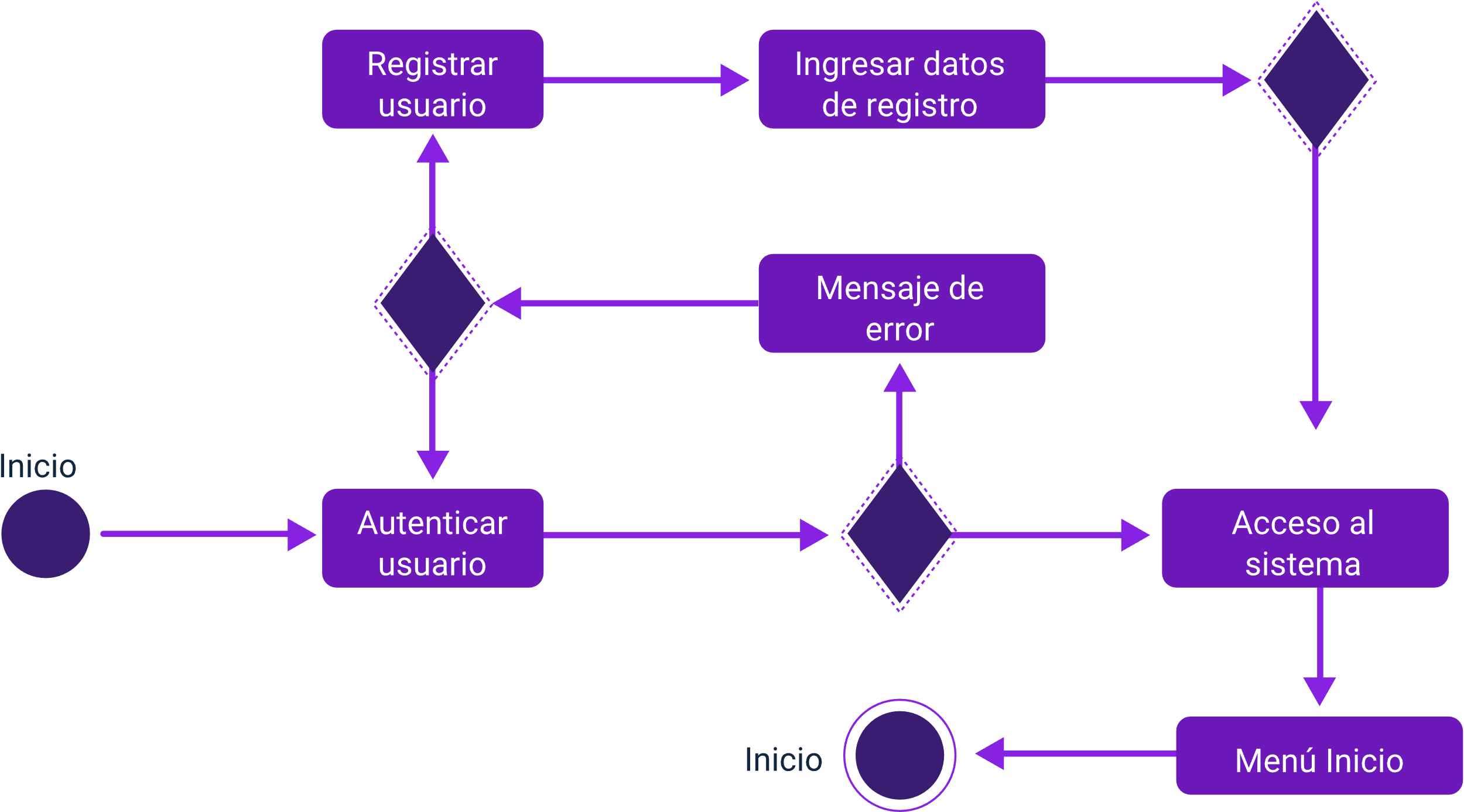
La figura representa la grafica de un nodo final.

Después de realizar la descripción de los elementos que componen un diagrama de actividades, se presenta el siguiente ejemplo de un diagrama de clase sobre un proceso de registro e inicio de sesión.

### Ejemplo de diagrama de actividades

A continuación, se presenta un ejemplo de un diagrama de actividades que incluye las funcionalidades de iniciar sesión y registrar en un sistema.

1. Ejemplo de diagramas de actividades.



Como se presenta en el ejemplo anterior, el flujo inicia con la actividad de autenticar usuario y pasa por un nodo de decisión que verifica si el usuario existe o no. Si el usuario existe, pasa a un nodo de decisión donde se valida el usuario; si no es correcto, muestra un mensaje de error; si es correcto, da acceso a la aplicación y muestra el menú de inicio. En caso de no existir el usuario, pasa a la actividad de registrar usuario y después ingresa datos. Luego sigue a un nuevo nodo de decisión, donde válida los datos y da acceso a la aplicación, continuando el flujo de actividad mostrando la interfaz del menú del sistema y finalmente termina la actividad con un nodo final.

# Diagramas de clase

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellas. Se utiliza durante el proceso de análisis y diseño de sistemas para crear el diseño conceptual de la información que se manejará y los componentes que se encargarán de su funcionamiento y la interacción entre ellos. En un diagrama de clases, se pueden distinguir principalmente dos elementos: las clases y sus relaciones (Flores Cueto & Bertolotti Zúñiga, 2013).

El objetivo de un diagrama de clases es representar las clases que conforman el modelo de un sistema determinado. Este diagrama se construye y se refina durante las fases de análisis y diseño, y sirve como referencia en la implementación del sistema.

Los diagramas de clases pueden utilizarse desde tres perspectivas diferentes:

* **Conceptual**

El diagrama de clase representa los conceptos dentro del dominio del problema en estudio.

* **Especificación**

El diagrama de clase muestra las interfaces de las clases sin detallar su implementación. Las clases se asemejan a tipos de datos, representando una interfaz con múltiples implementaciones posibles.

* **Implementación**

Esta vista muestra las clases tal y como se manifiestan en el entorno de implementación.

Los elementos que componen un diagrama de clase son:

* **Clase**

Es la unidad básica que encapsula la información para definir un objeto (siendo un objeto una instancia de una clase). Mediante la clase, es posible modelar entidades del dominio de estudio, como una casa, un auto o una cuenta corriente.

* **Atributos**

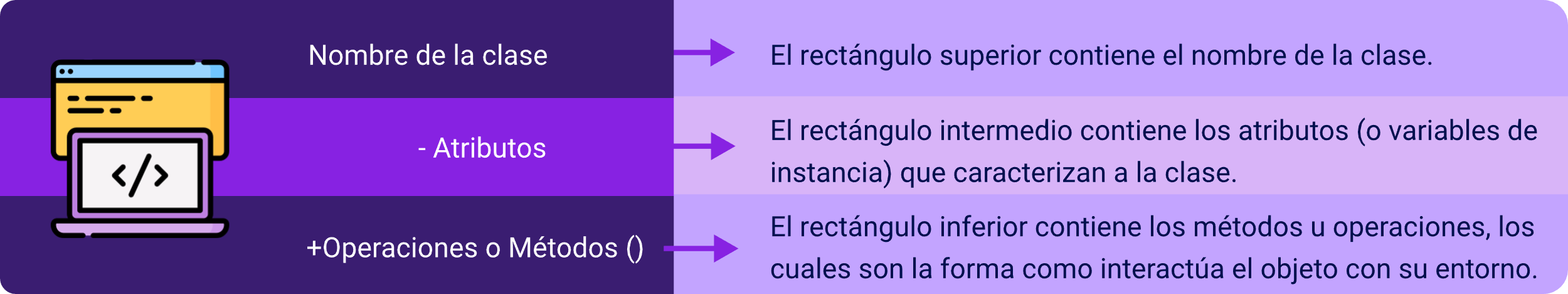
Son las características o propiedades específicas de una clase.

* **Métodos**

Son las operaciones o funciones a través de las cuales una clase interactúa con sus datos.

En UML, una clase es representada por un rectángulo que posee tres divisiones, en donde:

1. Representación de una clase en UML



Los atributos y los métodos de las clases tienen características que definen su grado de comunicación y visibilidad con el entorno, a lo que se denominan controles de acceso.

### Control de acceso

El control de acceso, también conocido como visibilidad, determina la accesibilidad de los atributos o métodos de una clase. Los atributos y métodos pueden ser privados, protegidos o públicos. Cada uno de estos niveles de acceso se representa con los símbolos especificados a continuación:

* **Público**

Se define un atributo o método como público cuando debe ser visible o accesible por todas las clases.

Símbolo: -

* **Privado**

Se define un atributo o método como privado cuando su visibilidad o acceso se restringe exclusivamente a la clase en la que se declara.

Símbolo: +

* **Protegido**

Se define un atributo o método como protegido cuando se desea que sea accesible solamente desde la clase que lo define y desde aquellas clases que heredan de ella.

Símbolo: #

Por ejemplo, se puede representar una clase denominada Producto que contiene tres atributos: código de tipo entero, nombre de tipo String y precio de tipo double. Además, incluye dos métodos: obtenerPrecio y calcularTotal. En esta representación, todos los atributos de la clase son privados, lo cual se indica con el símbolo -, y los métodos son públicos, identificados con el símbolo +.

* Producto

- codigo: int

- nombre: String

- precio: double

+ obtenerPrecio()

+ calcularTotal()

### Relaciones entre clases

Las relaciones entre las diferentes clases indican cómo interactúan los objetos pertenecientes a esas clases. Existen distintos tipos de relaciones:

* Asociación (conexión entre clases).
* Dependencia (relación de uso).
* Generalización (relaciones de herencia).

## Asociaciones

La relación de asociación se refiere a las conexiones entre clases. Este tipo de relación se representa gráficamente mediante una línea que conecta dos clases y presenta las siguientes características:

* **Nombre de la asociación**

Es necesario asignar un nombre a la asociación, que es obligatorio para establecer el contexto de la relación.

* **Rol**

Cada clase tiene un rol asociado que describe su función en la asociación. Los roles son opcionales y no se necesitan si el nombre de la clase describe claramente su papel en la relación.

* **Navegabilidad**

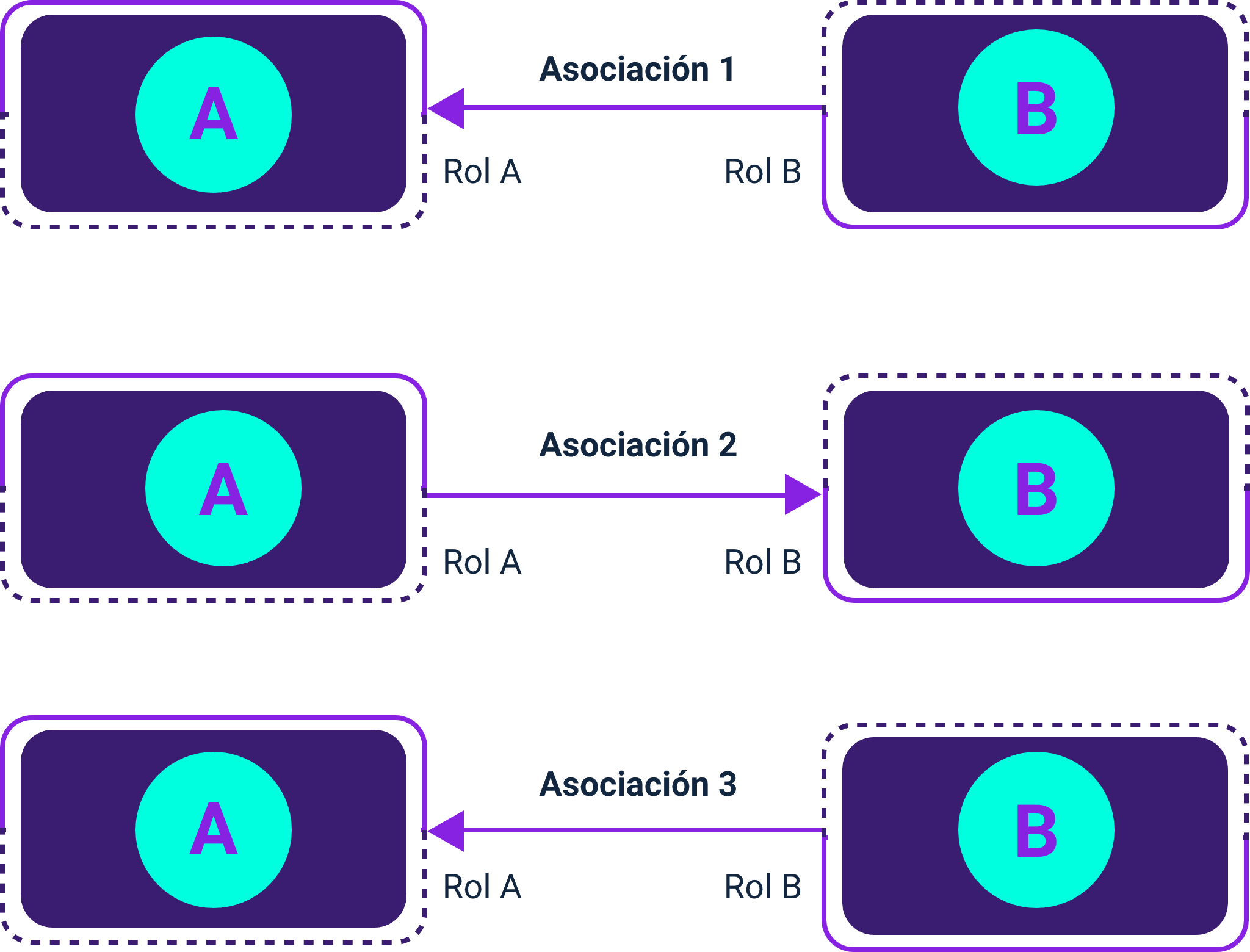
Define si la relación es unidireccional o bidireccional. Se representa gráficamente con flechas que indican la dirección de la asociación.

* **Multiplicidad**

Indica el número de instancias de una clase que pueden asociarse con una instancia específica de la otra clase en la relación.

A continuación, se presentan algunos ejemplos gráficos:

1. Ejemplo de asociación



En el ejemplo anterior, se pueden identificar tres diferentes ejemplos de asociación entre la clase A y la clase B, en todos los ejemplos la Clase A asume el rol A en el contexto de asociación, mientras que la clase B asume el rol B en el contexto de la relación. Adicionalmente, se identificó que los dos primeros ejemplos presentan navegabilidad unidireccional, lo que indica que la relación se establece en una sola dirección. En el primer caso, la 'Asociación1' vincula la Clase B, como origen, con la Clase A, como destino. En el segundo caso, 'Asociación2' vincula la Clase A, como origen, con la Clase B, como destino. En el último ejemplo, el vínculo es bidireccional; es decir, funciona en ambos sentidos: desde la Clase A hacia la Clase B y viceversa.

Una relación de asociación debe contar, como mínimo, con tres elementos:

* El nombre de la asociación
* La navegabilidad
* La multiplicidad.

En cuanto a la multiplicidad, esta indica el número de instancias de las clases que participan en la relación. Aunque este número puede variar, es común utilizar el valor de '1' o '\*' para denotar múltiples instancias.

Para facilitar la comprensión y la interpretación, es importante considerar la navegabilidad. La lectura se realiza tomando como referencia una instancia de la clase de origen de la asociación con respecto al valor 'N' de instancias de la clase de destino, donde 'N' representa la multiplicidad en el extremo opuesto.

Por ejemplo, se podría representar una clase llamada 'Figura' que contiene dos atributos ('lado1' y 'lado2') y tres métodos (un constructor 'Figura', un método 'área' y un método 'perímetro') de la siguiente manera:

1. Ejemplo de variaciones de asociación entre clases



En el ejemplo anterior, se presentan tres variaciones de relaciones de asociación entre la clase Instructor y la clase Aprendiz. En los tres casos, la navegabilidad varía en función de lo que se representa en la relación, tal como se indica en su nombre.

* **En el primer caso**

Existe una relación de asociación denominada Enseña, con una navegabilidad que va del Instructor al Aprendiz, indicando que un instructor enseña a varios aprendices.

* **En el segundo caso**

La relación se interpreta de la siguiente manera: un aprendiz es evaluado por un instructor.

* **En el tercer caso**

De esta representación gráfica, se ilustra una relación bidireccional; es decir, se pueden realizar lecturas desde cada extremo. En este ejemplo, un instructor comparte conocimientos con muchos aprendices y un aprendiz comparte experiencias con un instructor.

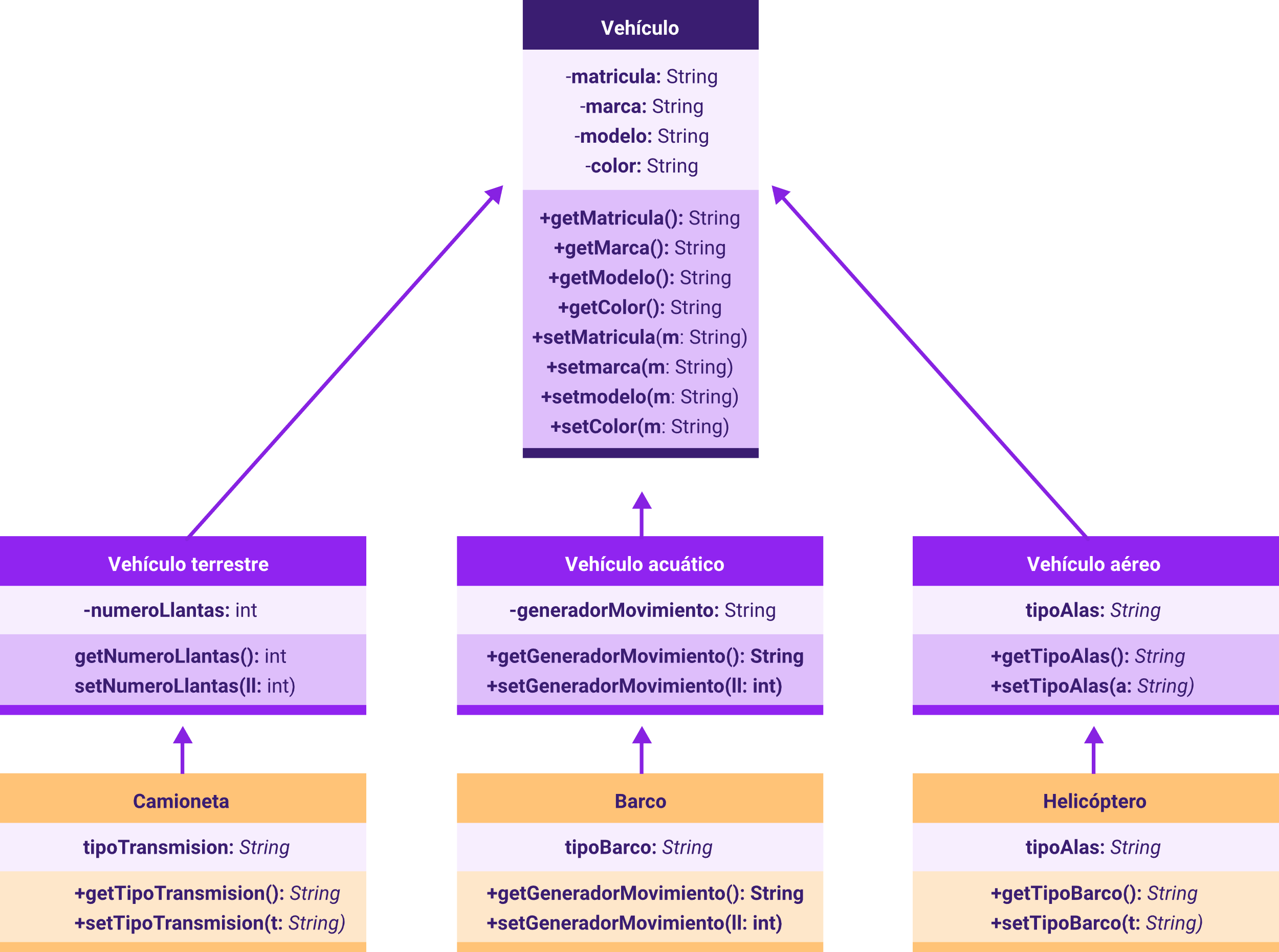
## Herencia

La herencia es una de las relaciones más comunes en la práctica del paradigma orientado a objetos. Como se mencionó anteriormente, se pueden construir relaciones que van desde conceptos (clases) generales a conceptos (clases) específicos (especialización), o de conceptos específicos a generales (generalización), según Schmuller (2001).

La herencia es un tipo especial de asociación que cuenta con clases principales o superclases (que son las más generales en la relación) y clases secundarias o subclases (que son las más especializadas). Este tipo de relación tiene implícito el nombre 'es un' o 'es una'. También, implícitamente, es una relación que vincula una instancia de la superclase con una instancia de la subclase; por lo tanto, no se asignan nombres ni multiplicidad a este tipo de relación.

En una relación de herencia, las subclases heredan las características (atributos) y comportamientos (métodos) de las superclases.

1. Ejemplo de herencia

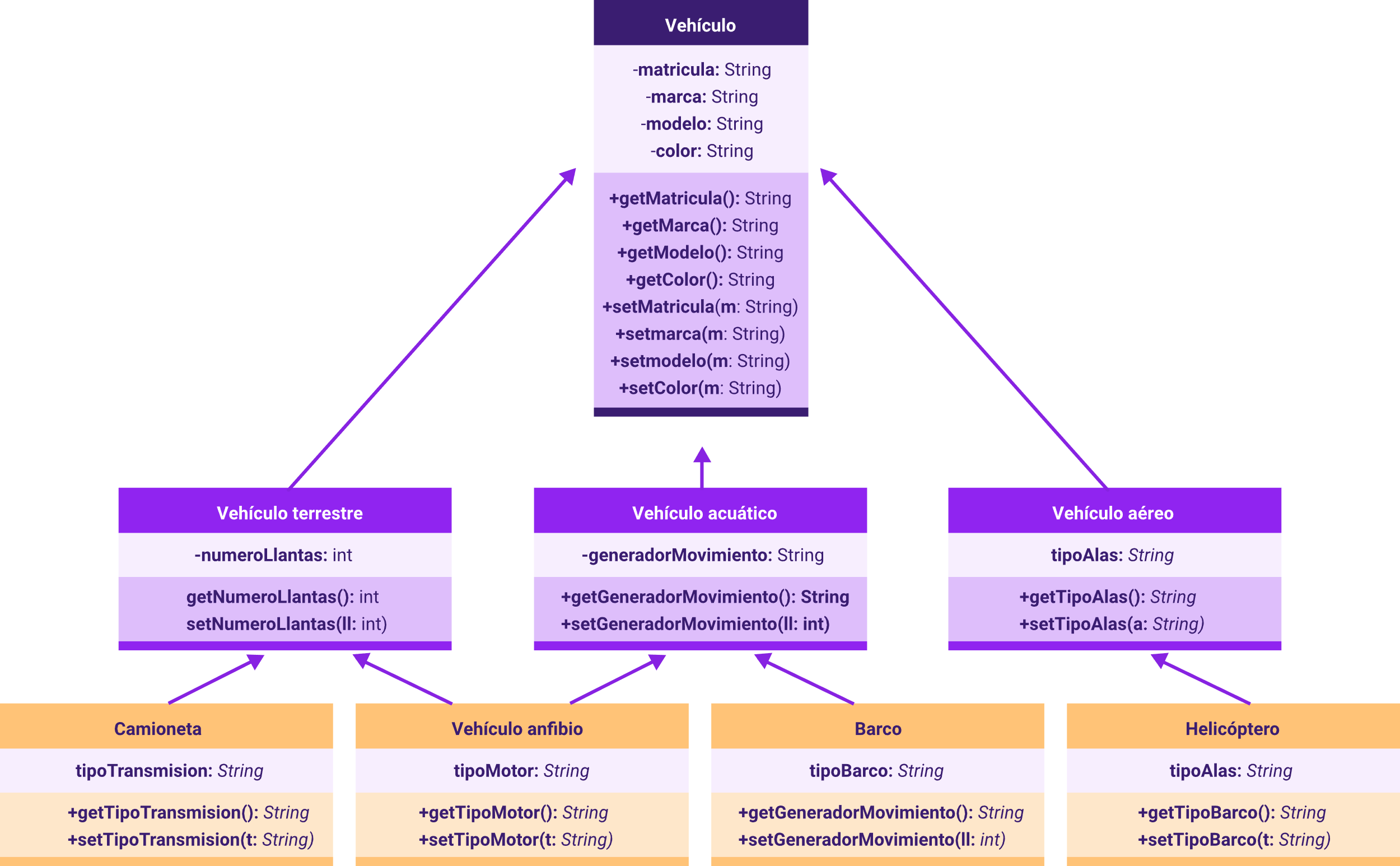


En la figura anterior, se presenta una relación de herencia entre siete (7) clases distintas. Las clases de tipo 'supertipo' están situadas en los niveles superiores de la jerarquía y representan las clases más generales cuyas definiciones serán heredadas por las subclases. En las relaciones de herencia, se pueden establecer varios niveles, y en cada nivel existen la correspondiente superclase y sus subclases. Es importante notar que no se asigna un nombre a estas relaciones porque se entiende implícitamente como una relación del tipo 'es un' o 'es una', la cual se interpreta desde las subclases hacia las superclases y siempre en relaciones uno a uno.

Por ejemplo, una Camioneta es un Vehículo Terrestre, que a su vez es un Vehículo. La clase Vehículo Terrestre hereda todas las características y comportamientos de la clase Vehículo. A su vez, la clase Camioneta hereda todas las características y comportamientos de la clase Vehículo Terrestre. De manera indirecta, la clase Camioneta está heredando las características y comportamientos de la clase Vehículo al estar en la misma línea de herencia. La clase Barco no hereda las características de la clase Vehículo Terrestre porque no se encuentra en la misma línea de herencia.

Hay dos tipos de herencia en función de la cantidad de clases supertipo directas asociadas: si una clase hereda características de un solo supertipo, es una herencia simple; si una clase hereda características de más de un supertipo, es una herencia múltiple.

1. Ejemplo de herencia múltiple



En el ejemplo de herencia múltiple, el diagrama de clases presenta varias relaciones de herencia. Por ejemplo, la clase Vehículo Anfibio es un caso de herencia múltiple, ya que hereda características de dos clases supertipo directas y diferentes: Vehículo Terrestre y Vehículo Acuático.

La herencia múltiple es admitida solo por algunos lenguajes de programación, siendo Python, Perl y C++ los más conocidos que la implementan. Sin embargo, otros lenguajes orientados a objetos ofrecen funcionalidades similares a la herencia múltiple mediante el uso de interfaces.

### Visibilidad en las relaciones de herencia

Para definir la visibilidad de los atributos y métodos de una clase, se utilizan los identificadores de acceso: private, public y protected. En el contexto de las relaciones de herencia, es crucial ser cuidadoso para evitar resultados no deseados, respetando las siguientes normas:

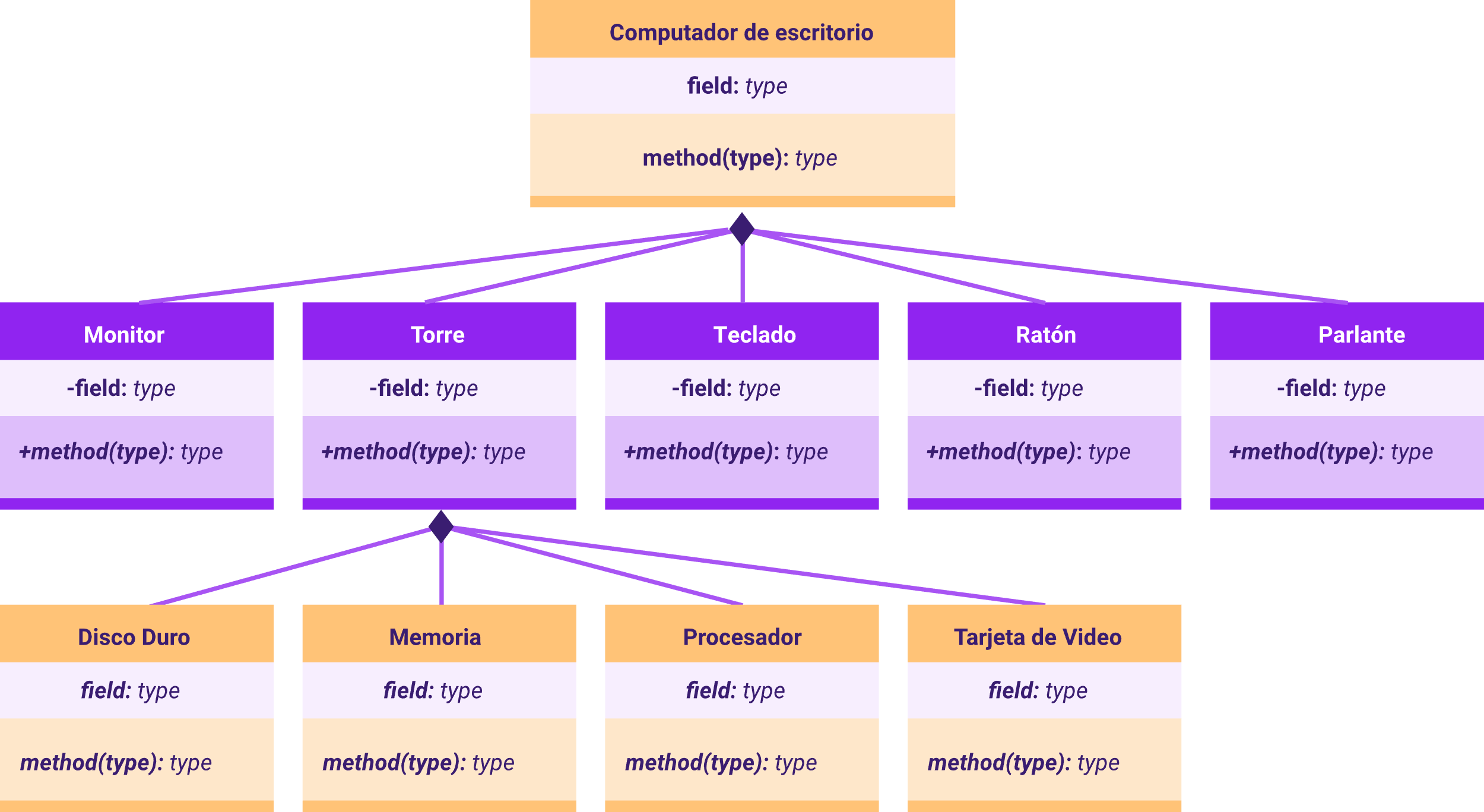
* Un atributo o método private en una superclase es inaccesible desde las subclases
* Un atributo o método public en una superclase permanece público en las subclases.
* Un atributo o método protected en una superclase se mantiene protegido en las subclases si existen más niveles de herencia, o actúa como privado en caso contrario.

## Agregación

La agregación es un tipo de asociación que indica que un conjunto de clases forma un todo. Por ejemplo, existen clases agregadas que simbolizan el todo y están compuestas por un conjunto de clases componentes.

Esta relación se representa con una línea que conecta la clase agregada con sus clases componentes, teniendo un rombo en el lado de la clase agregada que representa la relación de agregación. Aunque esta relación no tiene un nombre explícito, ya que se comprende la existencia de clases agregadas y componentes, es necesario establecer la multiplicidad en el lado de los componentes.

1. Ejemplo de relación de Agregación

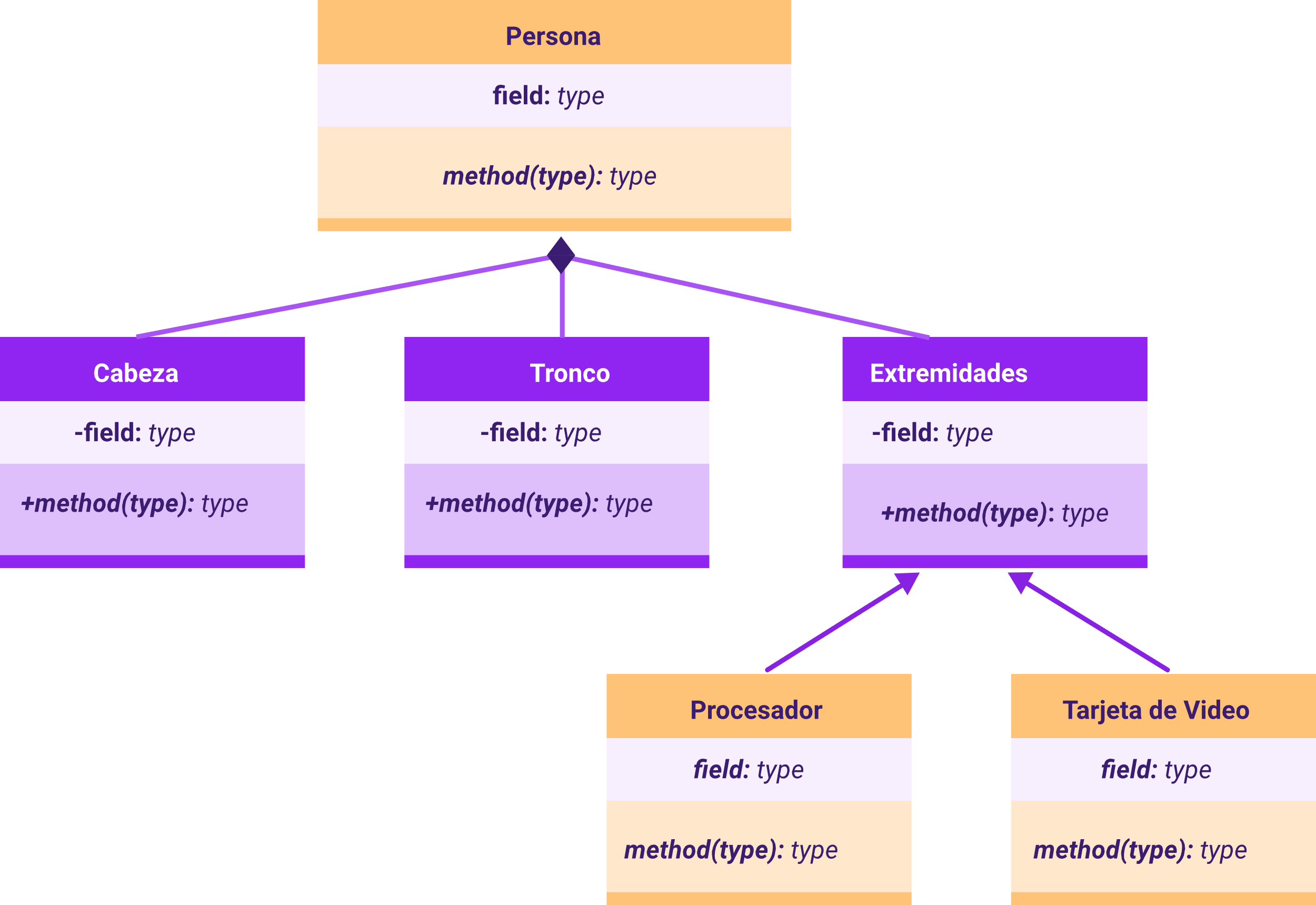


En la figura se presenta un ejemplo de relación de agregación. Como se puede apreciar, la clase Computador de Escritorio está compuesta por un monitor, una torre, un teclado, un ratón y dos parlantes. Además, la clase Torre consta de varios discos duros, memorias, un procesador y una tarjeta de video. Los componentes en este diagrama pueden ser partes de otras clases. Asimismo, una instancia de cualquier componente puede ser sustituida con facilidad, lo cual sugiere que la asociación con la clase agregada no es muy fuerte, característica distintiva de una agregación.

## Composición

La composición es un tipo particular de relación de agregación donde los componentes no pueden pertenecer a más de una relación de agregación; es decir, son exclusivos de la composición establecida. Para distinguir una relación de composición de una de agregación, el rombo representativo se rellena completamente de negro. (Schmuller, 2001).

1. Ejemplo de relación de Composición



En la figura anterior, se identifica un ejemplo de composición. En este caso lo que se representa es una relación mucho más fuerte que la agregación ya que los elementos que componen la clase agregada no pueden formar parte de otra clase.

# Herramientas CASE

Según Pressman, en el pasado, la ingeniería de software era principalmente una actividad manual, donde las herramientas se utilizaban solo en las etapas finales del proceso. Sin embargo, los ingenieros de software contemporáneos destacan la necesidad de herramientas más sofisticadas que las manuales para satisfacer las demandas de los sistemas informáticos actuales (Pressman, 1998).

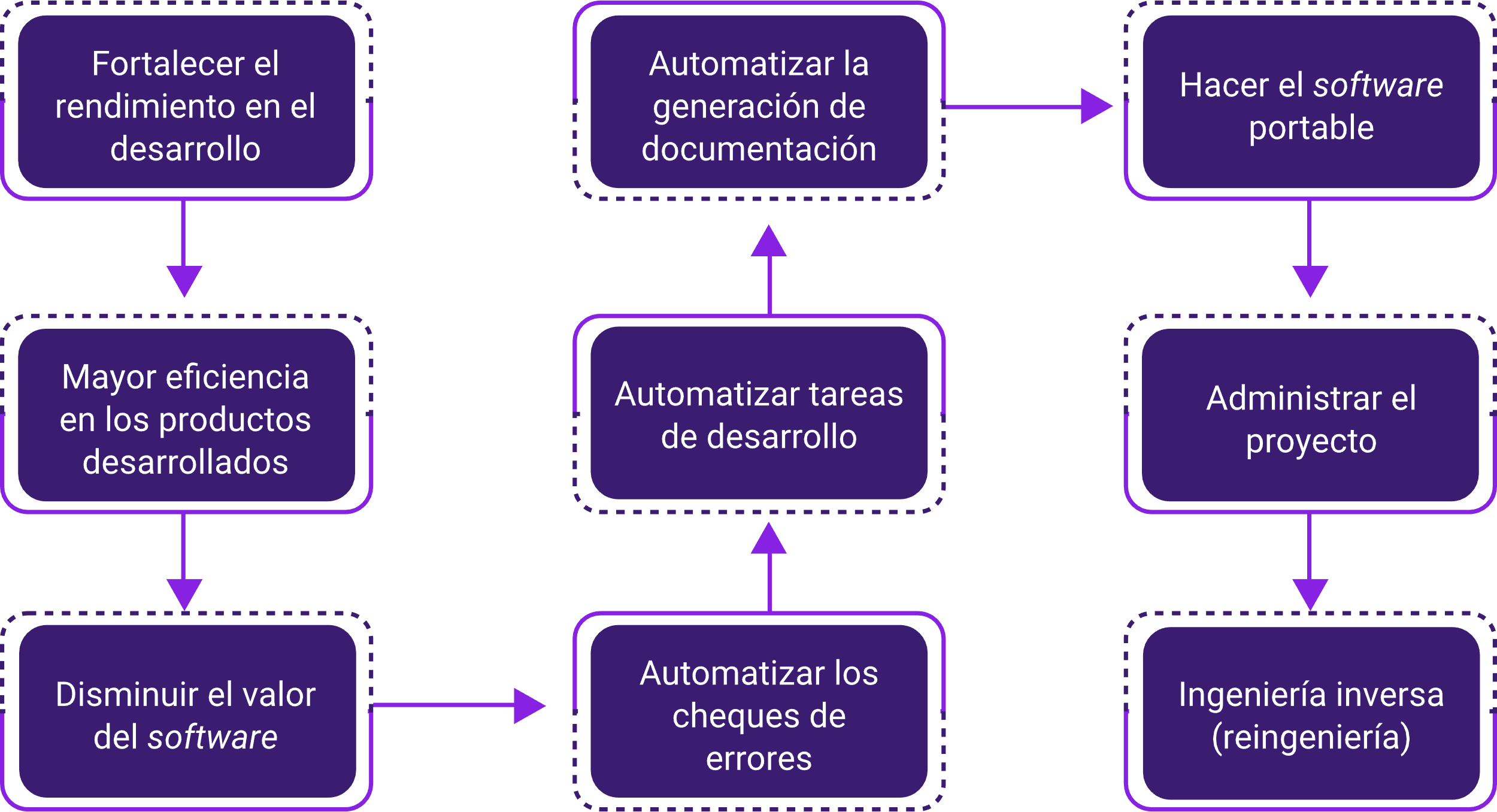
Las herramientas CASE son un complemento en la caja de herramientas de un ingeniero de software proporcionando la capacidad de automatizar tareas que tradicionalmente se ejecutaban de forma manual. Esto ayuda a garantizar que la calidad se integre desde la fase de diseño, mucho antes de proceder a la construcción del producto.

CASE es un acrónimo de "Computer Aided Software Engineering", que traducido al español significa "Ingeniería de Software Asistida por Computadora".

La tecnología CASE se refiere a la ingeniería de Software apoyada por herramientas computacionales. Consiste en un conjunto de herramientas que automatizan las tareas de desarrollo de software, con la finalidad de ofrecer un sistema integrado que conecte y automatice las fases del ciclo de vida del desarrollo (Cuevas, 1991).

De acuerdo con Pressman (1998), los objetivos de las herramientas CASE son:

1. Objetivos de la herramienta CASE



### Clasificación de las Herramientas CASE

Las herramientas CASE en función de las fases del ciclo de vida que abarcan, se pueden agrupar de la siguiente forma:

* **Herramientas integradas, I-CASE (Integrated CASE, CASE integrado)**

Abordan todos los aspectos del ciclo de vida del desarrollo de sistemas. Son llamadas también CASE de trabajo en equipo.

* **Herramientas de alto nivel, U-CASE (Upper CASE - CASE superior) o front-end**

Se utilizan para la automatización y soporte de las actividades de las etapas tempranas del desarrollo: análisis y diseño.

* **Herramientas de bajo nivel, L-CASE (**Lower CASE **- CASE inferior) o back-end**

Ayudan a las etapas finales en el ciclo de vida del desarrollo: construcción e implementación.

* **Juegos de herramientas o Tools-Case**

Son el tipo más simple de herramientas CASE. Automatizan una tarea o actividad del ciclo de vida del desarrollo de sistemas que se convierte en herramienta o herramientas independientes, constituyendo un caso de multi-herramienta.

A continuación, se presentan ejemplos de las herramientas CASE por ámbitos:

* **Análisis y diseño**

Workbench

GUI Design Studio

DIA

StarUML

* **Diseño de archivos y base de datos**

Power Designer

Oracle Designer

EasyCase

* **Programación**

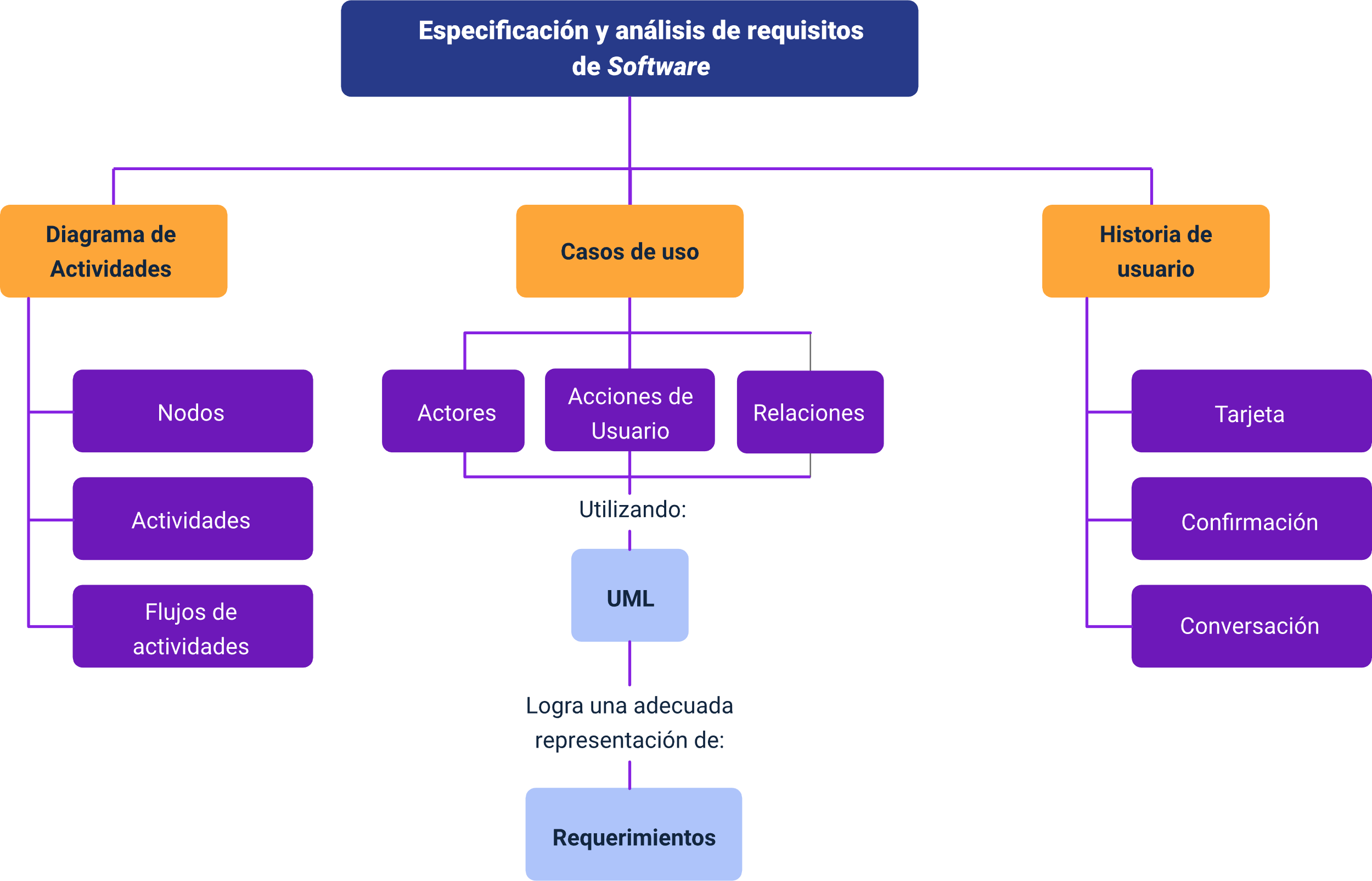
NetBeans

Eclipse/Omondo

Los diagramas de clase se diagraman con las herramientas que hacen parte del grupo de Herramientas de alto nivel, U-CASE, entre las que encontramos StartUML, Lucichart, ArgoUML, MagicDraw.

Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo:



Material complementario

| Tema | Referencia | Tipo de material | Enlace del recurso |
| --- | --- | --- | --- |
| Diagrama de casos | Nicosiored. (2017, November 22). Diagrama de Casos de Uso II - 5 - Tutorial UML en español. YouTube. | Video | <https://youtu.be/DUjBnEvIm1M?feature=shared> |
| Diagrama de actividades | Nicosiored. (2018, January 31). Diagrama de Actividades - 15 - Tutorial UML en español. YouTube. | Video | <https://youtu.be/GoYdpOVhDRc?feature=shared> |
| Diagramas de clase | TodoCalls & DataScience (2017).Curso UML Diagrama de Clases. YouTube. | Video | <https://www.youtube.com/watch?v=7WRYH2ei5Rw&t=106s> |
| Lista Herramientas Case | Herramientas CASE (2008, July 11). Herramientas CASE más utilizadas. | Página web | <https://herramientascase.wordpress.com/las-mas-utilizadas/> |

Glosario

**Ágil**: conjunto de tareas o acciones para producir y mantener productos y lograr objetivos de proceso. incluye procedimientos, estándares, políticas y objetivos para crear y modificar productos de trabajo.

**CASE**: ingeniería de software asistida por computación.

**Herramienta CASE**: una herramienta software que automatiza una parte de las fases del ciclo del desarrollo de sw.

**Método**: indica cómo construir técnicamente el software, incluyendo técnicas de modelado y otras técnicas descriptivas.

**Metodología**: colección de métodos para resolver un tipo de problemas.

**Requerimiento**: petición de algo que se solicita.

**Requisito**: condición que debe cumplir algo, en general, esto cumple con el requerimiento.

**Stakeholders**: individuo u organización que comparte, reclama o le interesa un sistema, o le compete una característica que satisface sus necesidades y expectativas.

**Tecnología CASE**: conjunto de instrumentos y técnicas software para automatizar una disciplina de la ingeniería, incluyendo metodologías estructuradas y herramientas.

Referencias bibliográficas

Booch, G. (1994). Object-oriented analysis and design. Redwood City.

Booch, G., Rumbaugh, J., Jacobson, I., Molina, J. J. G., & Martínez, J. S. (2006). El lenguaje unificado de modelado: guía del usuario. Addison-Wesley.

Cohn, M. (2004). User stories applied: For agile software development. Addison-Wesley Professional.

Cohn, M. (2018). User Stories and User Story Examples by Mike Cohn. [online] Mountain Goat Software. <https://www.mountaingoatsoftware.com/agile/user-stories>

Cuevas Agustín, Gonzalo. (1991). Ingenieria del Software.Práctica de la programación, Editorial RA-MA, 1 impresión.

Flores Cueto, J. J., & Bertolotti Zúñiga, C. (2013). Diagrama de clases en uml. 6.

Gutiérrez, J. (s.f.). Diagramas UML de casos de uso y de requisitos.

Larman, C. (2002). “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process”. 2nd Ed. Prentice Hall.

Norris & Rigby. (1994). “Ingeniería de software explicada”, 1 edición Editorial Megabyte-Noriega editores, México.

Pressman, Roger. (1998). Ingeniería del Software. Un enfoque práctico, Editorial Mc. Grraw-Hill, 4ta edición, 1998.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). El Lenguaje Unificado de Modelado. Elements, 30.

Schmuller, J. (2001). Aprendiendo UML en 24 horas. Pearson Educación.

Schmuller, J., & Garza Marín, A. D. (2000). Aprendiendo UML en 24 horas.

Wake, W.C. (2003). “Extreme Programming Explored” y “Refactoring Workbook”

Créditos

| Nombre | Cargo | Centro de Formación y Regional |
| --- | --- | --- |
| Milady Tatiana Villamil Castellanos | Responsable del Ecosistema | Dirección General |
| Olga Constanza Bermúdez Jaimes | Responsable de Línea de Producción | Centro de Servicios de Salud - Regional Antioquia |
| Zulema Yidney León Escobar | Experta Temática | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| Jonathan Guerrero Astaiza | Experto Temático | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| Paola Alexandra Moya Peralta | Evaluadora Instruccional | Centro de Servicios de Salud - Regional Antioquia |
| Andrés Felipe Herrera Roldán | Diseñador de Contenidos Digitales | Centro de Servicios de Salud - Regional Antioquia |
| Edwin Sneider Velandia Suárez | Desarrollador Fullstack | Centro de Servicios de Salud - Regional Antioquia |
| Edgar Mauricio Cortés García | Actividad Didáctica | Centro de Servicios de Salud - Regional Antioquia |
| Luis Gabriel Urueta Álvarez | Validador de Recursos Educativos Digitales | Centro de Servicios de Salud - Regional Antioquia |
| Margarita Marcela Medrano Gómez | Evaluador para Contenidos Inclusivos y Accesibles | Centro de Servicios de Salud - Regional Antioquia |
| Daniel Ricardo Mutis Gómez | Evaluador para Contenidos Inclusivos y Accesibles | Centro de Servicios de Salud - Regional Antioquia |