

PROGRAMACIÓN ORIENTADA A OBJETOS

Índice

Objetivos	2
Bibliografía	2
1. Introducción	3
Historia de los lenguajes orientados a objetos	3
2. Características Generales del Diseño Orientado a Objetos	5
Elementos básicos de la orientación a objetos	9
Características fundamentales de un objeto	10
Interacciones entre objetos	13
Diagrama de Comunicación	13
Diagrama de secuencia	14
Fragmentos combinados	15
Objetos con estereotipos	16
4. CLASES	19
5. MÉTODOS Y MENSAJES	22
6. HERENCIA	24
7. RELACIONES ENTRE CLASES	32
Asociación	33
Agregación	36
Relación de Agregación	37
Relación de Composición	37
Diagrama de clases	39
8. Conceptos Claves	44
Polimorfismo	44
Abstracción	45
Persistencia	45
Ventajas del Diseño Orientado a Objetos	45

UNIDAD 1: Diseño Orientado a Objetos

Objetivos

Al término de esta unidad se espera:

- Que el estudiante distinga las ventajas de la orientación a objetos, frente a otros tipos de diseño.
- Que el estudiante interprete los conceptos de encapsulamiento, abstracción, reusabilidad y herencia.
- Que el estudiante diseñe y organice adecuadamente una jerarquía de clases, atendiendo a los distintos tipos de relaciones entre clases.

Bibliografía

Fontela, Carlos - Programación Orientada a Objetos. Técnicas Avanzadas de Programación. Fontela, Carlos. Buenos Aires. Editorial Nueva Librería. 2003.

Louden, Kenneth C. - Lenguajes de Programación Principios y Práctica- Mexico, D.F. Editorial Thomson. 2004.

Pierre-Alain, Muller - Modelado de objetos con UML -- Barcelona: Gestión 2000. 1997.

Rumbaugh, James. y otros - Modelado y diseño orientados a objetos. Madrid : Prentice-Hall, 1997.

Pratt, Terrence – Zelkowitz, Marvin - Lenguajes de Programación Diseño e Implementación. Editorial Prentice Hall. Hispano americana

Libro Programación Orientada a Objetos - Universidad Nacional del Sur - se encuentra disponible en sitio web de la asignatura.

Doug Rosenberg, Kendall Scott - Applying Use Case Driven Object Modeling with UML - An Annotated e-Commerce Example-Addison-Wesley Professional (2001)

Craig Larman - Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and the Unified Process-Prentice Hall (2001)

Sitios Web :

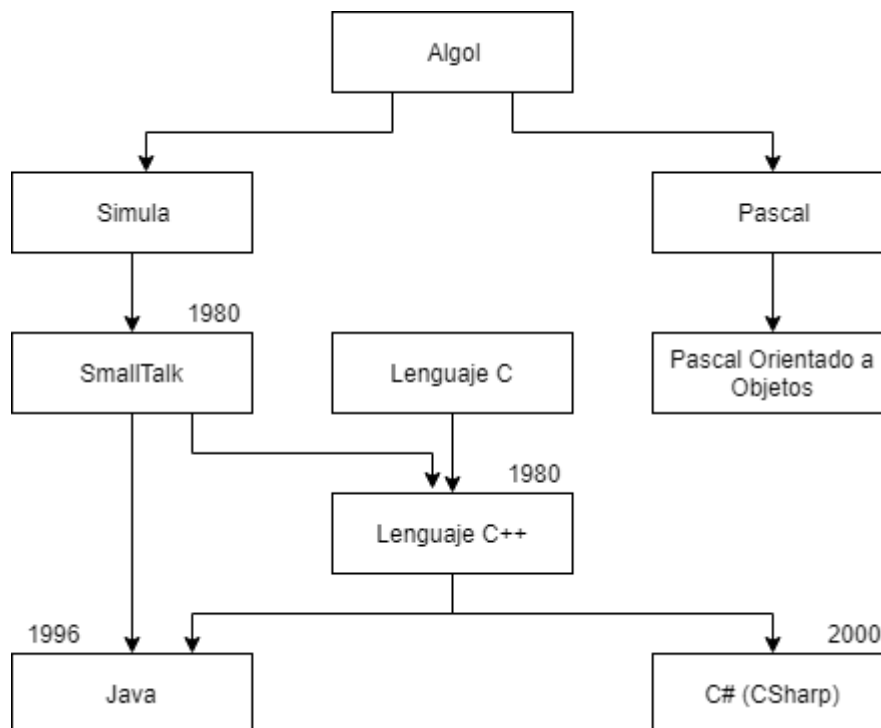
<http://www.vc.ehu.es/jiwotvim/IngenieriaSoftware/Teoria/Bloquell/UML-1.pdf>

1. Introducción

En la década actual, las posibilidades del software están retrasadas respecto de las del hardware en un mínimo de dos generaciones de procesadores. En general las herramientas técnicas y abstracciones del software convencional se convierten rápidamente en inadecuadas mientras que los sistemas que requieren software son cada vez más grandes y más complejos. Además, la disponibilidad de aplicaciones para una mayor base de usuarios, requiere que el software sea más flexible y más fácil de usar. Con el tiempo ha ido creciendo el consenso de que el paradigma¹ de orientación a objetos puede ayudar a controlar esta complejidad y aprovechar al máximo el entorno de sistemas en aplicaciones más útiles.

La tecnología orientada a objetos nace en los programas y luego se traslada a distintas áreas, principalmente análisis y diseño. Si bien esta tecnología es aplicable a Base de Datos, Hojas de Cálculo, Sistemas Operativos, etc., se debe ser muy cuidadoso en el análisis de estos productos pues muchas veces lo único que poseen es una única clase, sin soportar los conceptos de orientación a objetos.

Historia de los lenguajes orientados a objetos



Un cierto número de lenguajes de programación ha contribuido a la evolución de los actuales lenguajes orientados a objetos. Simula, desarrollado en los años 60 como lenguaje para programar simulaciones, contribuyó con el concepto de clase y los

¹ Paradigma: El filósofo e historiador de la ciencia, [Thomas S. Kuhn](#) dio a **paradigma** su significado contemporáneo cuando lo adoptó para referirse al conjunto de prácticas y saberes que definen una [disciplina científica](#) durante un período específico.

mecanismos de herencia. La fuerza principal que impulsó el desarrollo comercial de los lenguajes orientados a objetos fue el proyecto Smalltalk, a principios de los años 70. Los objetivos de Smalltalk incluían el concepto de clases y un interfaz gráfico. A través de los años fue evolucionando, y en la actualidad se ha convertido en uno de los lenguajes puros de mayor aplicación.

A pesar de este movimiento temprano hacia los lenguajes orientados a objetos, solamente se lograron pequeñas incursiones en la comunidad de la programación general. El progreso fuera de los lenguajes procedimentales sucedía en forma lenta. El progreso reciente se ha acelerado debido a las disponibilidades de las extensiones orientadas a objetos de dos lenguajes de mucho uso, C y PASCAL, como así también a las extensiones para otros lenguajes populares como BASIC y COBOL.

Los lenguajes orientados a objetos se dividen en dos grandes grupos: el grupo de lenguajes puros orientados a objetos y el grupo de lenguajes híbridos orientados a objetos. En el grupo de los lenguajes puros orientados a objetos se encuentra el Smalltalk, Eiffel y Java. Este último adquirió mayor notoriedad a partir de su utilización en la generación de componentes (Applets) de las páginas WEB.

En el grupo de lenguajes híbridos, las construcciones orientadas a objetos se añaden a un lenguaje procedimental. Los miembros de este grupo incluyen C++ y las diferentes variantes del lenguaje Pascal orientado a objetos. Una gran cantidad de código de programas ya está escrito en C y Pascal, por lo que añadir técnicas orientadas a objetos a un programa ya existente en un lenguaje conocido es mucho más fácil que volver a escribir una aplicación en un lenguaje completamente nuevo.

En general los lenguajes puros orientados a objetos resaltan el prototipo rápido, mientras que los híbridos ponen el énfasis en la velocidad de ejecución y facilitan la incorporación de extensiones orientadas a objetos para el programador orientado a procedimientos. Los lenguajes orientados a objetos más maduros, como Smalltalk, ofrecen también robustas librerías de clases y excelentes conjuntos de herramientas de desarrollo. Estas capacidades se están incorporando a los lenguajes híbridos.

Respecto a C# (CSharp), es un lenguaje creado por Microsoft y luego estandarizado. Está muy asociado a la tecnología .NET de Microsoft, pero nada impide que sea implementado en otras plataformas.

Algunos autores dicen que C# es un clon de Java. De hecho, se parece a un Java con notación C++: maneja interfaces, no soporta herencia múltiple, tiene un modelo de memoria con referencias, las excepciones y clases derivan de un ancestro común, la transformación de tipos está controlada y soporta multihilos en forma estándar.

Por el resto, es un lenguaje genuinamente orientado a objetos. Sin embargo, su gran potencia viene de la enorme cantidad de clases que Microsoft ha construido en el framework .NET, por lo cual es difícil imaginarlo como un lenguaje independiente de esta tecnología.

De todas maneras, .NET es más una plataforma que un lenguaje, como también puede afirmarse de Java. Como C# es el lenguaje estrella de la plataforma, podemos analizarlo en conjunto con ella. Debemos decir que es cierto que .NET toma el modelo de Java en muchos aspectos. Sin embargo, Microsoft ha hecho un esfuerzo importante para que un programa C# corriendo sobre el marco de trabajo de .NET sea más rápido que el programa equivalente en Java corriendo sobre una JVM en el mismo entorno.”²

² Fontela, Carlos. Programación Orientada a Objetos – Técnicas Avanzada de Programación – Pág. 142

2. Características Generales del Diseño Orientado a Objetos

Dice Carlos Fontela³: “En la programación imperativa tradicional, los algoritmos se describen en base a procesos (ya sea en forma secuencial o mediante concurrencia). Por ejemplo, se pueden escribir en Pascal o C una serie de subprogramas para el manejo de pantallas, y serán exactamente eso: procedimientos o funciones que describen cómo se abre una ventana, cómo se la cierra, cómo se la limpia, etc. Así, los algoritmos se expresan mediante procesos y éstos como una secuencia de tareas a realizar por la computadora. Por eso este tipo de programación se llama procedimental o imperativa.

La programación modular buscó cubrir algunos baches que surgían de aplicar la programación estructurada en el caso de sistemas de tamaño mediano a grandes, pero buscando objetivos similares. Así, la programación modular permitió dividir el programa en módulos autónomos que se pudieran programar, verificar y modificar individualmente. Como ya se ha analizado en otras asignaturas muchas son las ventajas de esta metodología cuando se hace un buen uso de la misma. No obstante, a mediados de 1980, esta programación no resolvía todos los problemas.

Cuando se trabaja con la programación modular, unos de los problemas que se presenta están en el uso de la abstracción. Esto era así porque el usuario se tentaba de hacer uso indebido de los datos, ya que tendía a usar conceptos propios de la implementación que no deberían ser de su incumbencia.

Mas adelante veremos como el **ocultamiento de la información** (o de implementación) en el desarrollo de software es un concepto es clave en la programación orientada a objetos (POO). Este paradigma encara la resolución de cada problema desde otra óptica: la óptica del objeto.

Así, para resolver el ejemplo de manejo de pantallas, se define un objeto pantalla y se definen todas las acciones (denominadas métodos) que ese objeto puede realizar (comportamiento), cada vez que se le envía un cierto mensaje o solicitud.

Los métodos definen las únicas acciones que se le pueden aplicar a un objeto. En síntesis es el comportamiento de los objetos frente a mensajes o solicitudes provenientes de otros objetos.

Tipo de datos abstractos

En este paradigma quedan de manifiesto la importancia de los conceptos de **abstracción** y **ocultamiento de información** lo cual permite definir la implementación de tipos de datos abstractos.

Un tipo de datos abstracto (TDA) hace referencia a:

1. un tipo de datos definido por el programador,
2. un conjunto de operaciones abstractas sobre objetos de datos,
3. y el encapsulamiento de los objetos de ese tipo, de tal manera que el usuario final del tipo no pueda manipular esos objetos excepto a través del uso de las operaciones definidas.

³ Fontela, Carlos. Programación Orientada a Objetos – Técnicas Avanzada de Programación – Pág. 27- 28.

Puede verse así una de las ventajas del ocultamiento de la implementación, que casualmente es permitir cambios de implementación de un TDA

Por ejemplo si se desarrolla un TDA que maneje una pila de números enteros implementadas a través de un arreglo, y se le diera al usuario acceso a la implementación, éste podría hacer uso de ciertas facilidades propias de la forma de implementación. Luego, si por algunas cuestiones del sistema, se decide que es conveniente hacer la implementación sobre una lista enlazada a través de punteros, no podría hacerse sin obligar al cliente a modificar sus programas.

Así, la primer ventajas del ocultamiento de la implementación es que al no poder el cliente usar la estructura de datos y algoritmos subyacente, en el futuro se podrá alterar dicha estructura sin que sea necesario cambios en los programas de los clientes⁴.

Programación orientada a objetos

Durante el proceso de desarrollo de requerimientos de un sistema de software el analista es responsable de identificar los objetos del problema y caracterizarlos a través de sus atributos. En la etapa de diseño se completa la representación modelando también el comportamiento de los objetos.

Un objeto del problema es una entidad, física o conceptual, caracterizada a través de atributos y comportamiento. El comportamiento queda determinado por un conjunto de servicios que el objeto puede brindar y un conjunto de responsabilidades que debe asumir.

Cuando el sistema de software está en ejecución, se crean objetos de software.

*Un **objeto de software** es un **modelo**, una representación de un objeto del problema. Un objeto de software tiene una **identidad** y un **estado interno** y recibe **mensajes** a los que responde ejecutando un **servicio**. El estado interno mantiene los valores de los atributos.⁵*

Desde el punto de vista de los objetos, un programa es un conjunto de objetos colaborando, o bien, un conjunto de objetos enviando mensajes y respondiendo a otros mensajes.

Dado que esos métodos son, en la práctica, muy similares a los procedimientos de la programación imperativa tradicional y los mensajes se podrían pensar como invocaciones a esos procedimientos, se podría pensar que simplemente se ha inventado un nuevo nombre para algo que ya existía anteriormente.

Esto es cierto: las ideas de objetos provienen en gran parte de la abstracción de datos. Sin embargo, el paradigma de objetos introduce conceptos nuevos. Además, es una técnica más estructurada que los intentos anteriores de estructuración. Y, por lo expresado, es más modular y abstracta que los intentos previos de abstracción de datos y ocultamiento de implementación. Es un paradigma de programación no imperativa, no procedimental. Se basa en técnicas previas, como la abstracción y el ocultamiento de la implementación, e incorpora la herencia, el polimorfismo y otras de menor entidad.”

En el paradigma de orientación a objetos los programas se centran en una entidad básica, **el objeto**, el cual combina los datos (atributos del objeto) con los procedimientos (métodos) que actúan sobre dichos datos. En segundo lugar, y a diferencia de los

⁴ Fontela. Op. Cit. pág. 18-20

⁵ Libro Programación Orientada a Objetos UNSur - Op. Cit. pág. 10

programas tradicionales que utilizan procedimientos que actúan sobre un conjunto de datos pasivos, los objetos interactúan entre sí enviando mensajes. El programador orientado a objetos puede agrupar características comunes de un conjunto de objetos en **clases**, a través de un proceso de abstracción. Los descendientes de estas clases se construyen por medio del mecanismo de subclasificación, permitiendo que sean heredados los métodos programados anteriormente y debiendo programar solamente las diferencias.

La Programación Orientada a Objetos (POO) no se puede desligar de todo el paradigma de orientación a objetos. En efecto, no hay que confundir POO con el uso de un lenguaje que soporte POO. Para hacer buena POO hay que desarrollar todo el sistema utilizando el paradigma, empezando por un análisis y un diseño orientados a objetos.”

En realidad, la orientación a objetos debería impactar en todas las tareas propias del desarrollo de software, no sólo en la programación (**requerimientos, análisis, diseño, implementación y prueba**). Esto se debe a que la POO es más natural si se parte del análisis y el diseño con objetos. Además, esta tecnología incluye una serie de mejoras que permite resolver problemas que presentaban las antiguas metodologías.

ACTIVIDAD 1:

- Agregue nuevas cuestiones inherentes a los conceptos de abstracción y ocultamiento de información que a su criterio no están presentes en este documento.⁶
- Teniendo en cuenta que uno de los objetivos de la orientación a objetos es la calidad del software, defina calidad de software e indique factores para obtener dicha calidad.
- Cuando se habla de productividad del software, se habla de reducir los costos del software y el tiempo de desarrollo. Investigue sobre cualidades relacionadas a la productividad del software.⁷

Lenguaje de Modelamiento Unificado

Para comprender mejor las aplicaciones orientadas a objetos, vamos a incluir algunos diagramas que nos permiten entender el sistema de software desde distintas perspectivas y resolver los problemas que se plantean.

Para eso, utilizaremos como notación a UML ((Unified Modeling Language o Lenguaje de Modelamiento Unificado).

UML es un lenguaje que permite la visualización, especificación y documentación de sistemas. No es una metodología sino una notación, que aglutina distintos enfoques de orientación a objetos.

UML 2 construido en el estándar altamente exitoso UML 1.x, el cual se ha convertido en un estándar de la industria, tanto para el modelado, diseño y construcción de sistemas de software así como también más negocios generalizados y procesos científicos.

UML 2 define 13 tipos básicos de diagramas, divididos en dos grupos generales:

⁶ Fontela, Carlos. Programación Orientada a Objetos – Técnicas Avanzada de Programación – Pág. 16 a 23

⁷ Libro Programación Orientada a Objetos UNSur – Pág. 5 a 7

1. Diagramas de Modelado Estructurado⁸

Los diagramas estructurales definen la arquitectura estática de un modelo. Ellos se usan para modelar las “cosas” que constituyen un modelo – los componentes de clases, objetos, interfaces y físicos. Además se usan para modelar las relaciones y dependencias entre elementos.

- Los diagramas de paquetes se usan para dividir el modelo en contenedores lógicos o “paquetes” y describen las interacciones entre ellos a un nivel más alto.
- Los diagramas estructurales o de clases definen los bloques de construcción básica de un modelo: los materiales generales, clases y tipos que se usan para construir un modelo completo.
- Los diagramas de objetos muestran cómo las instancias de elementos estructurales se relacionan y usan en tiempo de ejecución.
- Los diagramas de estructuras compuestas proveen un medio de ordenar una estructura de elementos y focalizar en detalles, construcciones y relaciones internas.
- Los diagramas de componentes se usan para modelar estructuras de alto nivel o más complejas, usualmente construidas desde una o más clases, y provee una interfaz bien definida.
- Los diagramas de despliegue muestran la disposición de artefactos significativos dentro de una configuración real.

2. Diagramas de Modelado de Comportamiento

Los diagramas de comportamiento capturan las variedades de interacción y el estado instantáneo dentro de un modelo mientras se “ejecuta” a través del tiempo.

- Los diagramas de Casos de Uso se usan para modelar las interacciones del usuario/sistema. Ellos definen el comportamiento, requisitos y restricciones en la forma de scripts o escenarios.
- El diagrama de actividades tiene un amplio número de usos, desde definir un flujo de programa básico, hasta capturar los puntos de decisión y acciones dentro de cualquier proceso generalizado.
- Los diagramas de máquina de estados son esenciales para el entendimiento de la condición momento a momento o “estado de ejecución” de un modelo cuando éste se ejecuta.
- El diagrama de comunicación muestra la red y la secuencia de mensajes de comunicaciones entre objetos en tiempo de ejecución durante una instancia de colaboración.
- Los diagramas de secuencia están estrechamente relacionados a los diagramas de comunicaciones y muestran la secuencia de mensajes pasadas entre los objetos usando una línea de tiempo vertical.
- Los diagramas de tiempos fusionan los diagramas de secuencia y estados para proveer una vista de un estado del objeto a través del tiempo y los mensajes que modifican ese estado.
- Los diagramas de descripción de la interacción fusionan los diagramas de actividades y secuencia para permitir que los fragmentos de interacción sean fácilmente combinados con los puntos y flujos de decisión.

⁸ http://www.sparxsystems.com.ar/resources/uml2_tutorial.html

Elementos básicos de la orientación a objetos

3. OBJETOS

Un objeto es una unidad atómica que encapsula estado y comportamiento. La encapsulación en un objeto permite una alta cohesión y un bajo acoplamiento.

Atributos de un objeto: Los atributos describen la abstracción de características individuales que posee un objeto.

Ejemplo: un objeto avión puede tener como características o atributos a la matrícula, el nombre del avión, la capacidad de carga y la cantidad de asientos. Matrícula es el nombre de un atributo cuyo valor puede ser el 1553-ALFA-33.

Comportamientos: Los comportamientos de un objeto representan las acciones que pueden ser realizadas por un objeto.

Ejemplo: Un objeto avión puede tener los siguientes comportamientos: aterrizar, despegar, cambiar de ruta aérea, pedir información a la torre de control, etc.

En general, los objetos pueden caer dentro de las siguientes categorías:

Cosas tangibles: avión, televisor, libro, auto, etc.

Roles representados por personas: alumno, gerente, médico, paciente, etc.

Organizaciones: empresa, división, equipo, etc.

Interacciones que relacionan dos o más objetos del modelo: Por ejemplo, un objeto acta de examen, el cual relaciona un objeto materia con un objeto alumno.

Lugares: sala de embarque, muelle de carga, supermercado, etc.

Descripción de objetos⁹

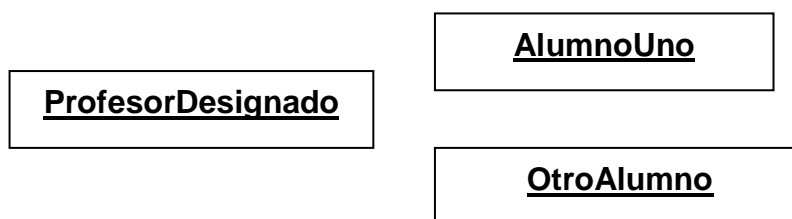
Los objetos informáticos definen una representación abstracta de las entidades de un mundo real o virtual, con el objetivo de controlarlos o simularlos.

En comparación con el ciclo de vida de los seres vivos, los objetos del mundo real que nos envuelven nacen, viven y mueren.

Los objetos se definen según el **contexto** del problema, lo que puede ser un objeto apropiado para una situación, puede no serlo para otra. Del mismo modo, las características que se observan para un objeto en un contexto, pueden no ser relevantes en otro contexto. Por lo tanto, la existencia de un objeto y las características que se observan, dependen del problema.

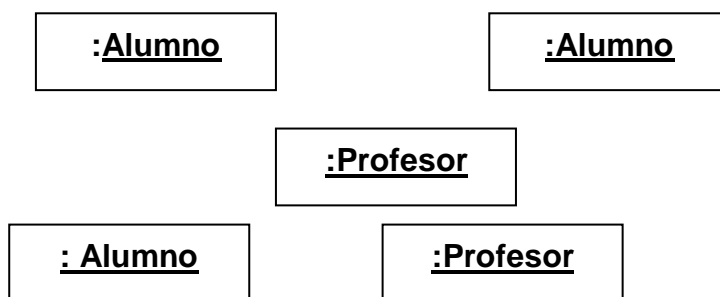
En UML, la notación general para un objeto es un rectángulo, que contiene el nombre del objeto subrayado. El diagrama siguiente representa tres objetos:

⁹Modelado de Objetos con UML. Pierre Alain Muller. Pág. 16.



A menudo es difícil encontrar un nombre para designar cada objeto; por ello la notación permite indicar un nombre genérico en lugar de su nombre individual. Este artificio permite hablar de objetos en términos generales.

El diagrama siguiente muestra objetos alumnos y profesores. Los dos puntos precisan que se trata de objetos anónimos, de género *Alumno* y *Profesor*.

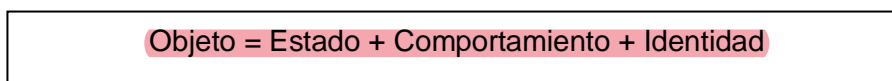


La notación general para el objeto se extiende incluyendo el nombre del objeto seguido del género. Como muestra el siguiente diagrama.



Características fundamentales de un objeto

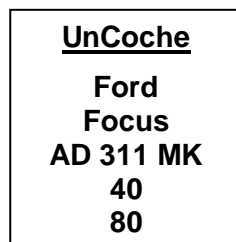
La presentación de las características fundamentales de un objeto permite responder de manera más formal a la cuestión: ¿qué es un objeto? Todo objeto presenta las tres características siguientes: un estado, un comportamiento y una identidad.



El estado

El estado agrupa los valores instantáneos de todos los atributos de un objeto, sabiendo que un atributo es una información que cualifica al objeto que la contiene. Cada atributo puede tomar un valor en un ámbito de definición dado. El estado de un objeto, en un instante dado, corresponde a una selección de valores, entre todos los valores posibles de los diferentes atributos.

El siguiente diagrama muestra un objeto coche que contiene, en un instante dado, los valores de sus atributos: marca, patente, cantidad de combustible, temperatura.



El estado evoluciona con el tiempo; así, cuando un coche circula, la cantidad de combustible disminuye y la temperatura del mismo varía. Ciertos atributos del estado pueden ser constantes, tal es el caso de la marca y el número de patente. Sin embargo, por regla general, **el estado de un objeto es variable y puede verse como la consecuencia de sus comportamientos pasados.**

El comportamiento

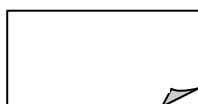
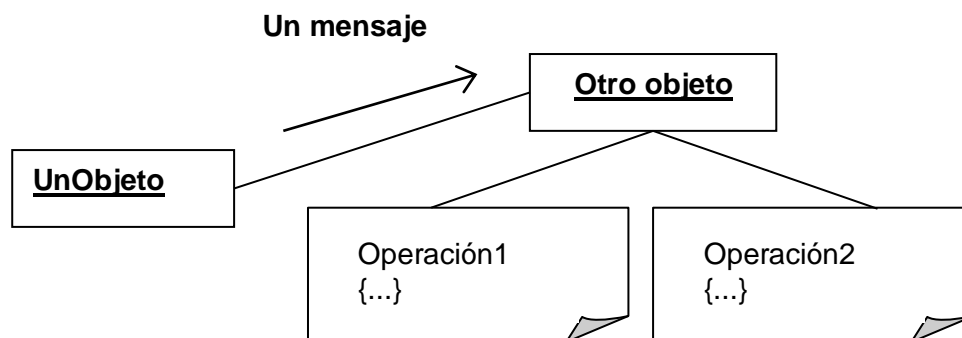
El comportamiento describe las acciones y reacciones de ese objeto.

Cada comportamiento se llama operación. Las operaciones de un objeto se desencadenan a consecuencia de un estímulo externo, representado en forma de un mensaje enviado por otro objeto.

Un **mensaje**, es una solicitud que pide al objeto que se comporte de una manera determinada.

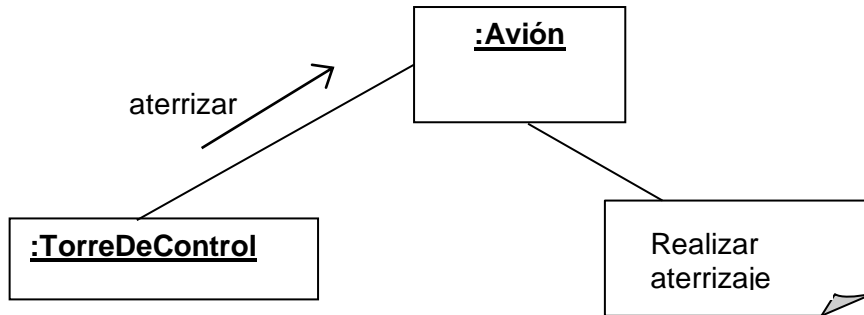
Todo mensaje está compuesto por la **identidad del receptor**, el **comportamiento a ejecutar** y opcionalmente **información adicional** para realizar el comportamiento.

En el siguiente diagrama, según el mensaje, se desencadena la *Operación1* o la *Operación2*.



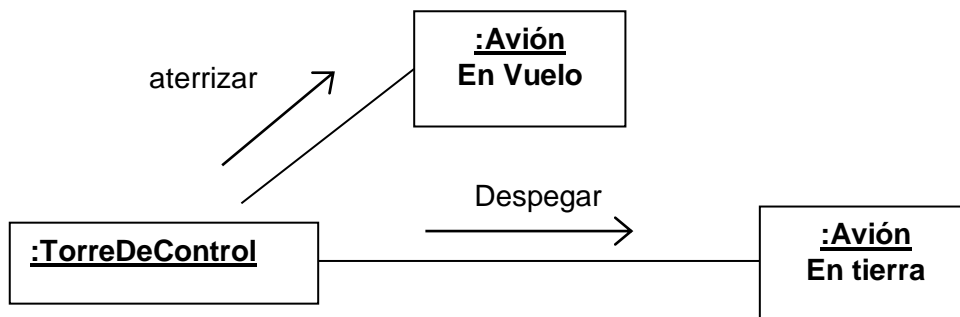
Este diagrama presenta un rectángulo cuya esquina inferior derecha está plegada: se trata de la representación de una nota, es decir, de una información de clarificación opcional expresada en un formato libre, a fin de facilitar la comprensión del diagrama.

Ejemplo:



Como veremos más adelante, las interacciones entre objetos se representan por medio de diagramas en los que los objetos que interactúan están vinculados entre sí por líneas continuas llamadas enlaces. La presencia de un enlace significa que un objeto conoce o ve a otro objeto. En el ejemplo anterior, el objeto perteneciente a la clase TorreDeControl conoce al objeto perteneciente a la clase Avión.

El **estado y el comportamiento están relacionados** ya que el comportamiento en un instante dado depende del estado actual, y el estado puede ser modificado por el comportamiento. Solo es posible hacer aterrizar un avión cuando está volando, es decir, el comportamiento *Aterrizar* sólo es válido si la información *En Vuelo* es válida. Tras el aterrizaje, la información *En Vuelo* pasa a ser no válida, y la operación *Aterrizar* deja de tener sentido. El ejemplo muestra la relación entre el estado y el comportamiento.



La identidad

Además de su estado, un objeto posee una identidad que caracteriza su propia existencia. La identidad **permite distinguir los objetos de forma no ambigua**, independientemente de su estado. Ello permite, entre otras cosas, distinguir dos objetos en los que todos los valores de los atributos son idénticos.

*El concepto de identidad se refiere al hecho de que **cada objeto es único en el mundo**, por más que su conjunto de atributos y sus valores sean exactamente iguales a los de otros objetos. Por ejemplo, dos autos del mismo modelo, color, motor, salidos de la misma línea de producción el mismo día no dejan de ser dos autos diferentes, por más que su conjunto de atributos y sus valores sean iguales. La única posibilidad de que dos objetos sean iguales es que sean el mismo objeto.*

La identidad es un concepto, no se representa de manera específica en modelado. Cada objeto posee una identidad de manera implícita.

En casos como el citado anteriormente, si es necesario diferenciar objetos, a menudo puede utilizarse un identificador procedente naturalmente del ámbito del problema. Este tipo de identificador, llamado también clave natural, puede añadirse al estado de los objetos para distinguirlos.

Por ejemplo, el DNI puede ser una clave natural para distinguir personas, el número de patente para distinguir automóviles, etc.

Interacciones entre objetos

Desde el enfoque del diseño orientado a objetos, una aplicación se va a modelar como un conjunto de objetos que interactúan entre sí para dar solución al pedido de un usuario. UML utiliza diagramas para modelar el sistema desde distintos aspectos, separados pero relacionados, de modo que permiten tener una visión más clara y detallada del sistema.

Una interacción expresa el comportamiento que resulta de la comunicación de un grupo de objetos. Una interacción la podemos visualizar según el punto de vista del tiempo o según el punto de vista del espacio.

Las interacciones entre objetos las podemos representar a través de dos tipos de diagramas: diagrama de comunicación y diagrama de secuencia.

Actores

La interacción de los objetos es iniciada por el usuario. Por lo que, para modelar un sistema es necesario identificar quien o quienes son sus usuarios. A partir de esto, se define el concepto de actor, **que es una entidad externa al propio sistema**, pero que necesita intercambiar información con él.

Los actores no están restringidos a ser personas físicas, también pueden representar a sistemas externos al sistema que se está modelando.

En el caso de los usuarios que son personas reales, el actor representa la función que la persona realiza, por ejemplo en un sistema de facturación de un supermercado los principales usuarios son los cajeros, por lo tanto, existirá un actor “cajero”. También es probable que exista un actor “gerente de sucursal” que consulta lo facturado, “gerente general”, en un contexto de una cadena de supermercados, entre otros actores.

Un actor se representa como se muestra a continuación.

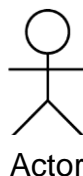


Diagrama de Comunicación

Estos diagramas **muestran interacciones entre objetos** en la estructura espacial estática que permite la comunicación entre objetos. El tiempo no se representa de manera explícita, por lo tanto los mensajes se numeran para indicar el orden de los envíos.

Por ejemplo en el contexto de una biblioteca, una de las principales funciones que realiza es el préstamo del material a los socios.

Cuando un socio solicita un determinado material en calidad de préstamo, implica que se lleven a cabo una serie de acciones que se enuncian brevemente a continuación.

El bibliotecario busca el material solicitado, constata que está disponible. Busca la ficha del socio y verifica que no esté suspendido. Luego registra el préstamo del material en la ficha del socio, y se establece el material como no disponible. Finalmente se le informa al bibliotecario que el préstamo se ha registrado.

Registrar préstamo de material

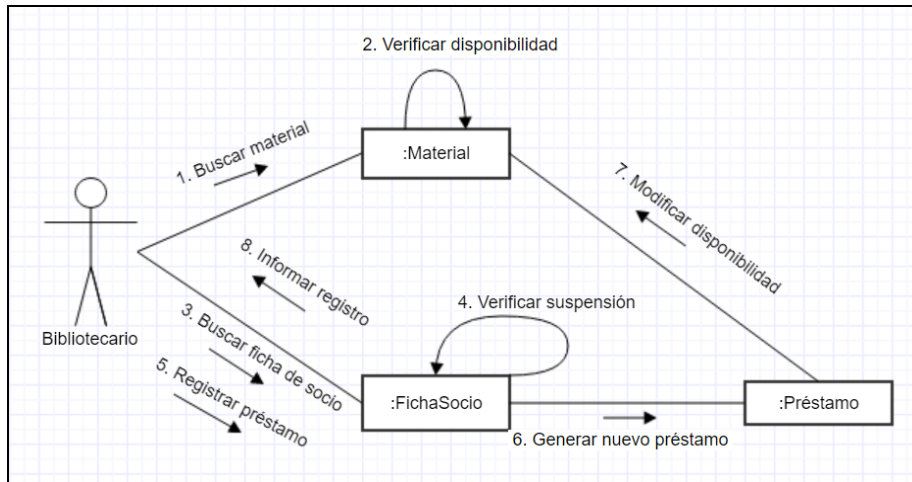


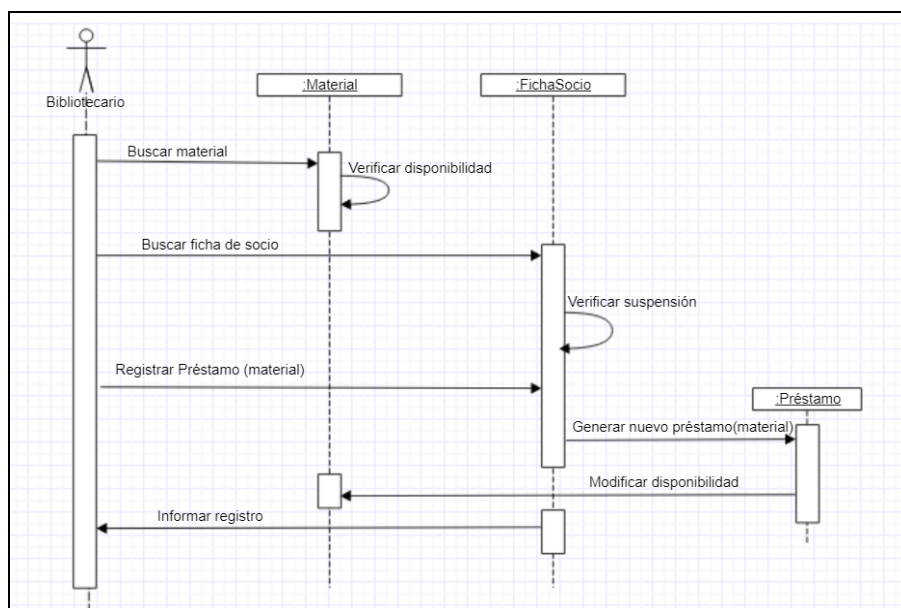
Diagrama de secuencia

Estos diagramas muestran interacciones entre objetos según un punto de vista temporal. Muestra la cronología de los envíos de mensajes.

Un objeto se representa por un rectángulo, una barra vertical llamada línea de vida del objeto. Un mensaje se representa con una flecha, y el orden de envío de los mensajes está dado por la posición sobre el eje vertical.

Ejemplo 1:

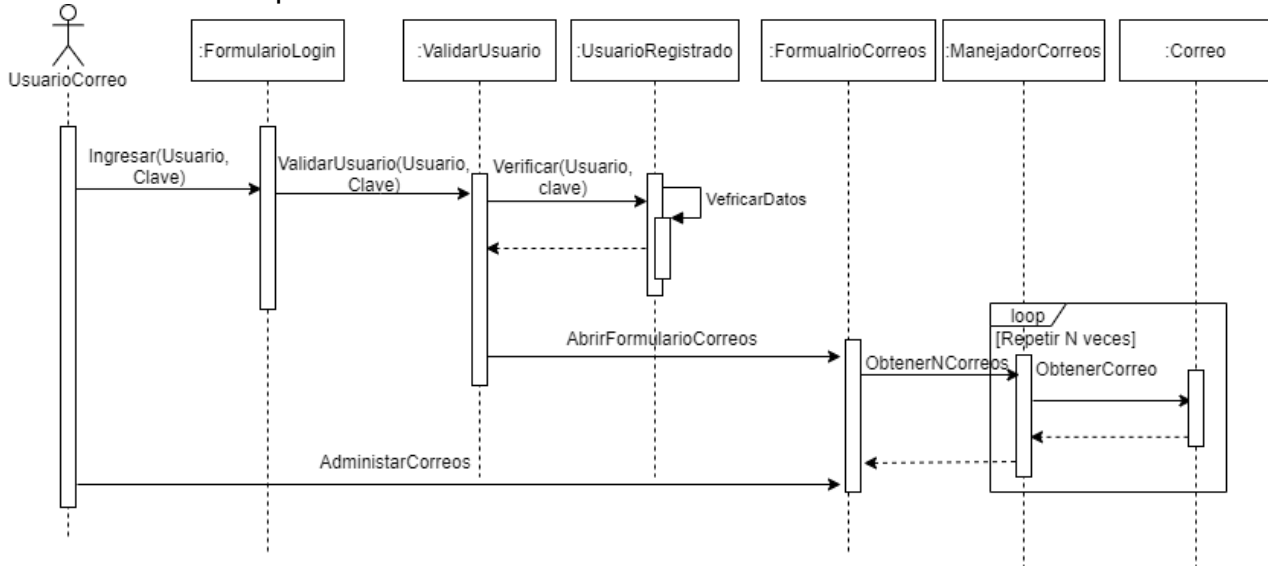
El siguiente ejemplo muestra el diagrama de secuencia referido al diagrama de comunicación antes expuesto.



Ejemplo 2:

Para ingresar a la aplicación web de correo electrónico de **Gmail**, un usuario debe loguearse; este proceso implica desplegar una interfaz (formulario web) en el que se solicita dirección de correo y clave.

El usuario, una vez ingresados estos datos, presiona sobre el botón Entrar, luego los datos del usuario son verificados y en caso de que los datos ingresados sean válidos, se permite ingresar a la aplicación de correo electrónico que despliega un nuevo formulario conteniendo las carpetas de correo electrónico.



Fragmentos combinados¹⁰

Existen mecanismos que permiten agregar un grado de lógica de procedimientos a los diagramas y se denominan fragmentos combinados. Un fragmento combinado es una o más secuencias de procesos incluidas en un marco y ejecutadas bajo circunstancias específicas.

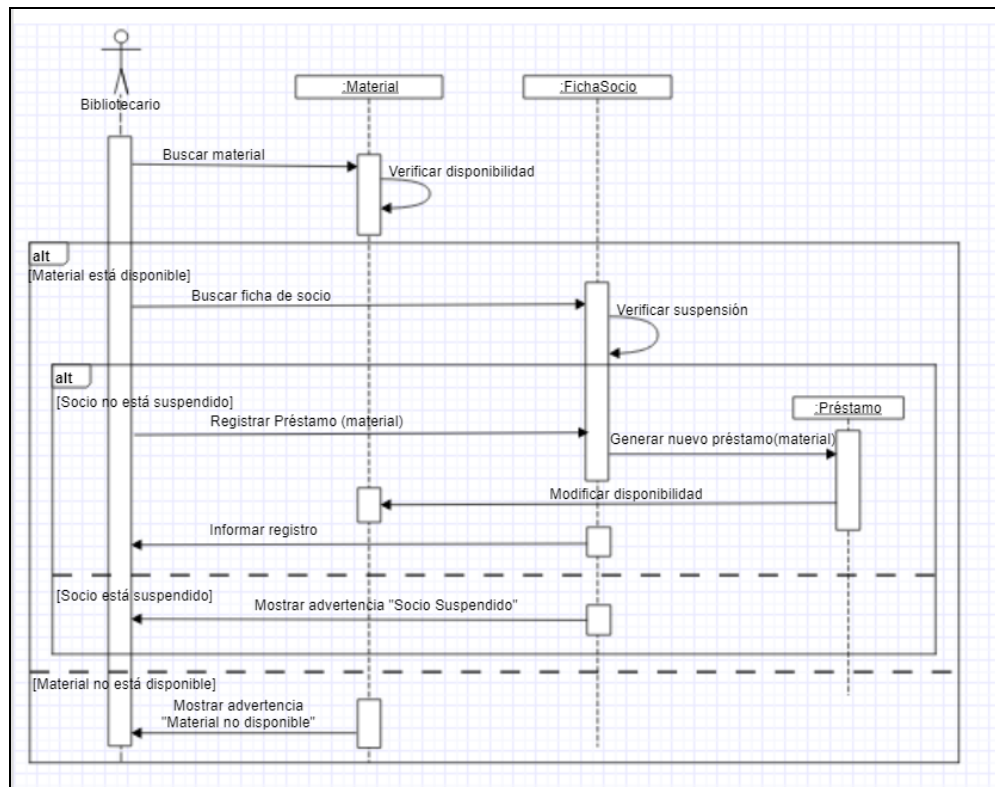
Existen varios fragmentos disponibles, pero para los fines de la asignatura solo resultan de interés 2 (dos) de ellos:

- Fragmento de alternativa (denotado “**alt**”): modela la elección de una interacción entre objetos, a través de una condición de guarda, es decir, modela estructuras condicionales del tipo if...then...else.
- Fragmento de iteración o bucle (denotado “**loop**”): el fragmento incluye un conjunto de mensajes que se ejecutan múltiples veces, según lo indique la condición de guarda

Ejemplo 3:

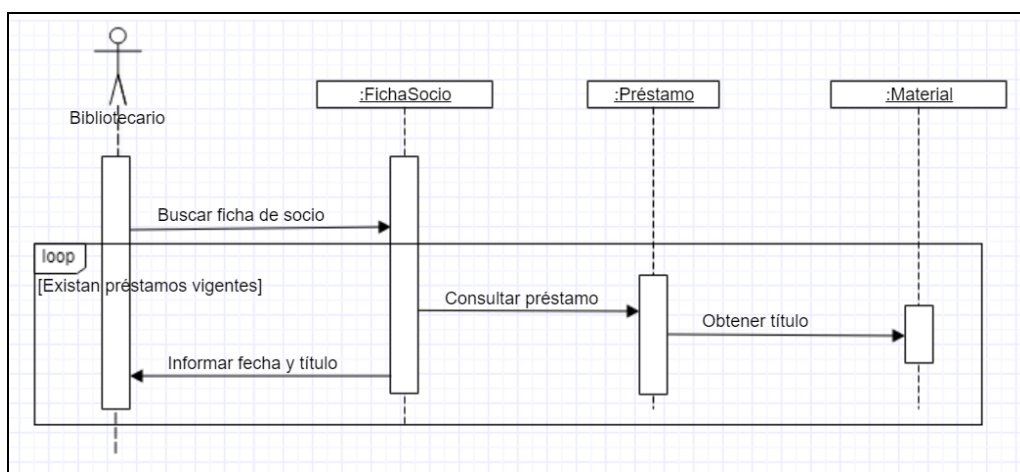
Retomando el ejemplo 1, se puede utilizar un fragmento combinado de alternativa para que el diagrama de secuencia modele la situación de que el material no está disponible y/o que el socio esté suspendido.

¹⁰ A partir de UML 2.0



Ejemplo 4:

El bibliotecario necesita conocer para un determinado socio, el título del material y la fecha de devolución de los préstamos vigentes (material aún no devuelto).



Objetos con estereotipos

El ejemplo 2 trata del proceso de logueo de un usuario en una aplicación web de correo electrónico. Es claro que los objetos del dominio del problema son los usuarios y los

correos. Pero, es muy común que en las aplicaciones (en especial las aplicaciones web) usen otros objetos que no pertenecen al dominio del problema, algunos surgen para presentar una interfaz al usuario (actor) de la aplicación, otros para controlar la lógica interna de la ésta.

En el diagrama de secuencia del ejemplo 2 figuran los objetos “FormularioEntrada” y “FormularioCorreos”, estos objetos conforman la interfaz de usuario. Por ello este diagrama de secuencia puede transformarse representando las instancias de los objetos a través de símbolos estereotipados¹¹. Esto permite distinguir claramente los distintos tipos de objetos.

Los estereotipos son:

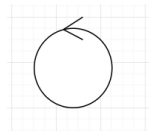
- a. Objeto de Interfaz (Boundary): representa un elemento (ejemplo un formulario web, ventana, pantalla, menú, cuadro de diálogo) con el cual interactúa el actor (por ejemplo el usuario).

Notación



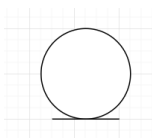
- b. Objeto de Control (Controllers): se ocupa de organizar y controlar la lógica requerida para alcanzar el objetivo de una determinada funcionalidad. Media entre los objetos de interfaz y los de entidad. Sirven de conexión entre los usuarios y los datos almacenados. Aquí es donde se capturan las reglas y políticas del negocio, que cambian con frecuencia, con la idea de localizar los cambios en estos objetos, sin interrumpir la interfaz de usuario ni el esquema de la base datos.

Notación



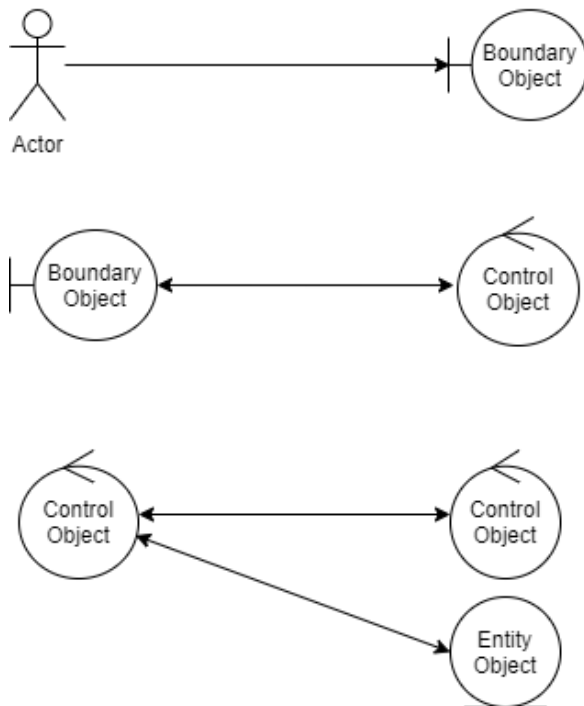
- c. Objeto de Entidad (Entity): para cada uno de los objetos (entidades) del dominio requeridos para realizar la funcionalidad.

Notación

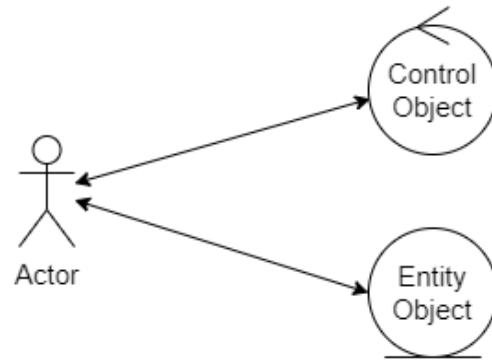


¹¹ Es un mecanismo de extensión del lenguaje que permite complementar la semántica de un elemento.

Relaciones permitidas

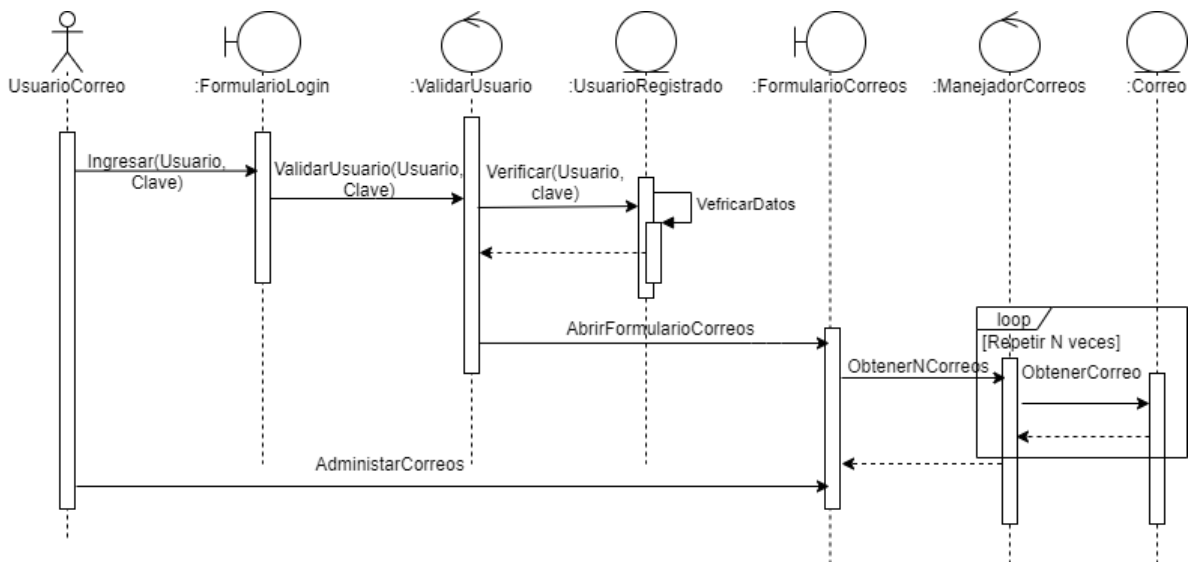


Relaciones NO permitidas



Ejemplo 5

El diagrama de secuencia del ejemplo 2 usando objetos estereotipados sería el siguiente:



Esta notación también se puede usar para el diagrama de comunicación.

ACTIVIDAD 2:

Un cine requiere que se implemente una aplicación informática que registre la información de las películas que se proyectan en distintas funciones en las salas que el cine dispone. La aplicación debe permitir al empleado responsable de programación consultar la información de la cartelera y mantenerla actualizada. También debe realizar la venta de entradas por parte de los vendedores.

a) Redacte la secuencia de acciones y elabore un diagrama de comunicación

El responsable de programación requiere conocer las salas en las que se proyecta una película determinada.

Una película se proyecta en distintas funciones. Una función se lleva a cabo en un día y horario establecido. Cada función tiene asignada una sala. Una función se proyecta siempre en la misma sala. Las funciones de una película pueden tener asignadas salas distintas.

b) Construya el diagrama de secuencia

La venta de entradas se puede realizar para cualquier función de la semana. Además, se ofrece el beneficio de promociones (descuentos) para aquellas personas que son titulares de determinadas tarjetas (por ejemplo Nevada, Santander Río, etc.). El descuento se aplica sólo sobre una entrada.

Cuando una persona le solicita al vendedor comprar entrada/s, el vendedor elegirá la opción "Venta". La aplicación mostrará, por cada película en cartelera, el nombre de la película junto con la fecha y la hora de las distintas funciones; el vendedor seleccionará una de ellas. A continuación, ingresará la cantidad de entradas a vender; en caso de que requiera el beneficio de promoción, la aplicación listará las tarjetas que están incluidas y el vendedor seleccionará una de ellas. La aplicación calculará el importe final de la venta y lo mostrará al vendedor, este seleccionará "Aceptar venta" y a continuación se registrará la venta.

4. CLASES

Una **clase** es una descripción de un conjunto de objetos, ya que consta de comportamientos y atributos que resumen las características comunes del conjunto.

La clase especifica el ámbito de definición de un conjunto de objetos. Cada objeto pertenece a una clase.

La posibilidad de definir clases es una de las ventajas de la orientación a objetos; definir clases significa colocar código reutilizable en un depósito común en lugar de redefinirlo cada vez que se necesite.

La **encapsulación** de una clase permite la cohesión y presenta distintas ventajas básicas:

- Se protegen los datos de accesos indebidos
- El acoplamiento entre las clases se disminuye
- Favorece la modularidad y el mantenimiento

Ejemplo: si se necesita trabajar con un conjunto de libros, para un sistema que automatiza una biblioteca, no es necesario definir el comportamiento para cada libro, será suficiente con agrupar ese comportamiento en la clase LIBRO.

Una clase es una especificación genérica para un número arbitrario de objetos similares. Los objetos que responden a la especificación de una clase son llamados **instancias** de una clase; en nuestro ejemplo cada objeto libro es una **instancia de la clase Libro**.

Libro
código : entero título: cadena autor: cadena cantPáginas: entero
CambiarAutor(); MostrarDatos();

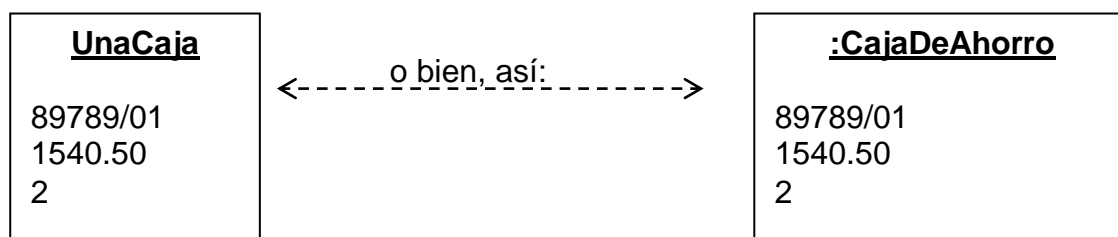
La **instanciación** es la operación que se realiza para obtener un objeto de una clase. La instanciación relaciona una clase y un objeto.

La clase tiene como atributos: código del libro, título, autor y cantidad de páginas. Los comportamientos contemplados para un objeto de esta clase son entre otros: cambiar el autor y mostrar los datos del libro.

Ejemplo: si se necesita automatizar el manejo de las cajas de ahorro de una entidad bancaria, tenemos la clase Caja de Ahorro, con las siguientes características.

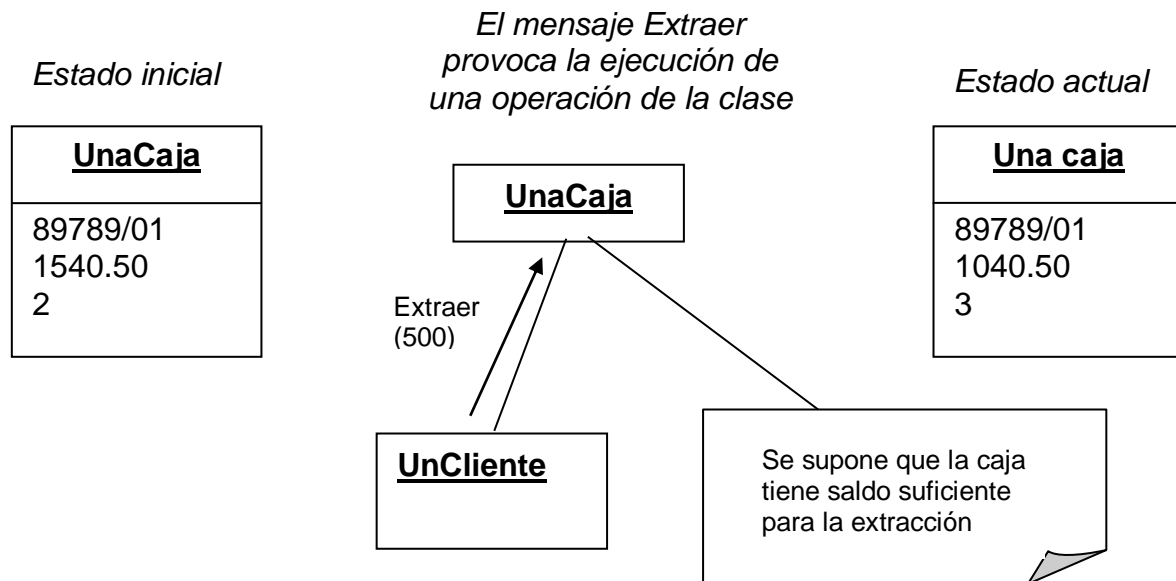
CajaDeAhorro
número de cuenta: cadena saldo: real cantExtracciones: entero
Depositar(importe); Extraer(importe) ConsultarSaldo();

Un objeto de la clase caja de ahorro, puede representarse de la siguiente forma¹²:



¹² Como explicaremos más adelante: para distinguir entre una clase (el tipo) y un objeto (una instancia del tipo), un objeto se muestra subrayado.

El siguiente esquema muestra el cambio de estado del objeto, debido a la operación extraer.



Nótese la diferencia fundamental entre un objeto y una clase. Un objeto es algo concreto, puede ser real o abstracto. Una clase es una abstracción de un concepto que engloba todos los objetos. LIBRO es una clase mientras “Programación Orientada a Objetos – Técnicas Avanzada de Programación” es un objeto de la clase LIBRO

Visibilidad de los atributos:

Cómo antes se explicitó, los atributos de una clase no deberían ser manipulables directamente por el resto de objetos, no obstante existen distintos **niveles de encapsulación** también llamados niveles de visibilidad.

La visibilidad indica si el atributo y/o comportamiento es visible o puede ser accedido desde otras clases.

Los tres niveles distintos de visibilidad utilizados son: *private*, *protected* y *public*¹³.

El nivel de visibilidad puede apreciarse en las representaciones gráficas de las clases mediante los caracteres +, # y -, que corresponden respectivamente a los niveles público, protegido y privado. Los mismos niveles de visibilidad usados para los atributos pueden aplicarse a los comportamientos.

¹³ Estos niveles se corresponden con los propuestos por el lenguaje de programación C++:

Reglas de visibilidad
+ Atributo público # Atributo protegido - Atributo privado
+ Método público # Método protegido - Método privado

El nivel más fuerte es el llamado nivel privado (private). En este caso, la parte privada de la clase es totalmente inaccesible desde otras clases.

Es posible lograr una visibilidad intermedia, colocando ciertos atributos en la parte protegida de la clase. Estos son visibles para las subclases. Para las demás clases, siguen siendo inaccesibles.

El nivel más bajo se obtiene colocando algunos atributos en la parte pública de la clase. Esto equivale a romper la noción de caja negra y hacer visibles los atributos para todas las clases.

Un análisis similar puede realizarse para la visibilidad de los métodos de una clase.

El diagrama siguiente muestra la representación de la clase Caja de Ahorro.

CajaDeAhorro
-número de cuenta: cadena -saldo: real -cantExtracciones: entero
+Depositar(importe : real); +Extraer(importe: real); +ConsultarSaldo():real;

Cómo vemos, lo conveniente es que los métodos sean de acceso público y los atributos privados. De este modo se logra el **Encapsulamiento**, entendiendo por esto a la conjunción de la abstracción y el ocultamiento de la información.

5. MÉTODOS Y MENSAJES

Los objetos tienen la posibilidad de actuar. La acción sucede cuando un objeto recibe un **mensaje**, que es una solicitud que pide al objeto que se comporte de manera determinada. Cada objeto recibe, interpreta y responde a mensajes enviados por otros objetos.

El conjunto de mensajes al que un objeto puede responder se llama protocolo del objeto.

Los comportamientos u operaciones que caracterizan un conjunto de objetos residen en la clase y se llaman **métodos**. En el ejemplo anterior **ConsultarSaldo ()**, es un método de la clase **CajadeAhorro**.

Los métodos son el código que se ejecuta para responder a un mensaje, y el mensaje es la llamada o invocación a un método.

Los métodos determinan cómo actúa el objeto cuando recibe un mensaje y manipulan los valores de los atributos del objeto. De esta forma, los métodos proporcionan la única forma de modificar los datos de un objeto. La estructura de un objeto (atributos y métodos), está oculta a usuarios del objeto, de manera tal que los mensajes que recibe un objeto proporcionan la única forma de conectar al objeto con el mundo exterior. Estas características de la orientación a objetos fomentan la modularidad (ya que es clara la frontera entre los distintos objetos), explícita la comunicación entre los distintos objetos y oculta los detalles de implementación.

Representación usando UML

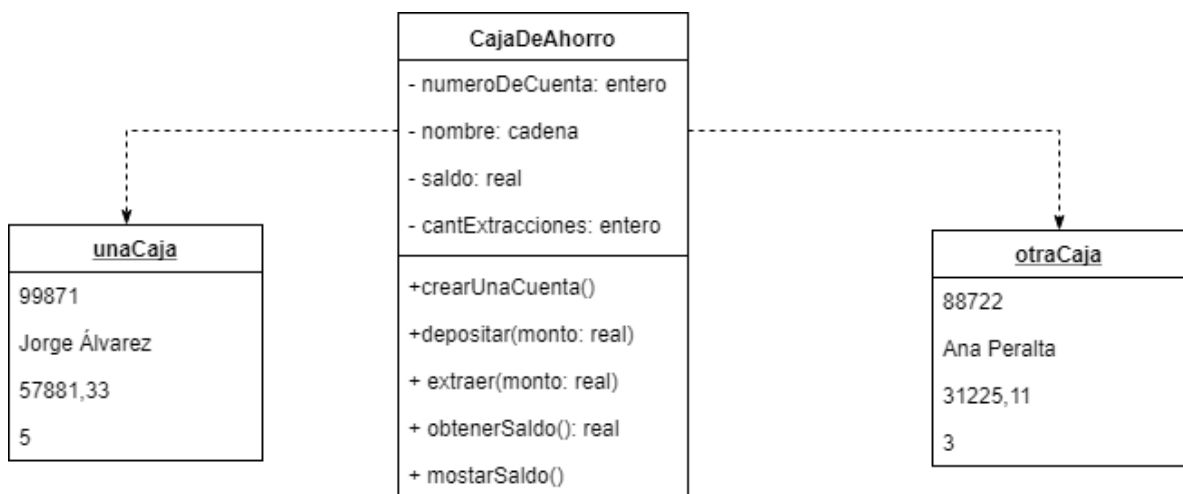
Para representar gráficamente clases y objetos utilizaremos la simbología propuesta por UML. En UML, una clase es representada por un rectángulo que posee tres divisiones:

Superior: Contiene el nombre de la clase.

Intermedia: Contiene los atributos que caracterizan a la clase.

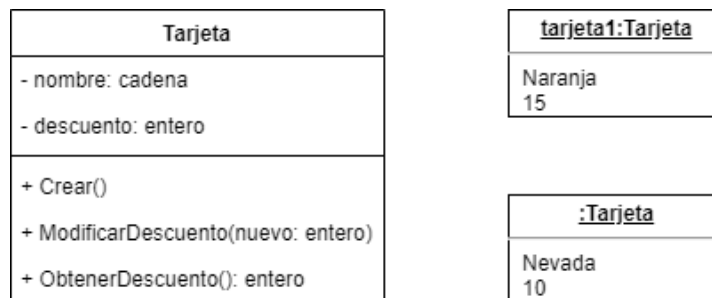
Inferior: Contiene los métodos, los cuales indican la forma como interactúa el objeto con su entorno.

En UML, **un objeto** se representa bajo la forma de un rectángulo; el nombre del objeto se subraya. La primera parte del rectángulo indica el nombre y la segunda los valores asociados a los atributos.



ACTIVIDAD 3

A – A partir del diseño de la situación planteada en la ACTIVIDAD 2 se obtuvieron los siguientes elementos.



A.1. Identifique clase y objeto.

A.2. Verifique que el/los objetos estén completos, en caso de no estarlo, complételos.

A.3. En la/ clase/s identificada/s indique el/los comportamiento/s que modifican el estado de un objeto.

B. Identifique dos clases que intervienen en la realidad planteada en la ACTIVIDAD 2.

B.1. Realice el diseño de ellas usando UML, indicando a su criterio los atributos y comportamientos que pueden incluir.

B.2. Represente en UML dos objetos de cada una de las clases diseñadas.

6. HERENCIA - Conceptos de Generalización y Especialización¹⁴


La generalización y la especialización son puntos de vista centrados en las jerarquías de clases.

La **generalización** consiste en factorizar los elementos comunes (atributos, métodos y restricciones) de un conjunto de clases en una clase más general llamada superclase. Como las clases se ordenan según una jerarquía, una superclase es una abstracción de sus subclases.

La generalización es un mecanismo complejo porque exige una buena capacidad de abstracción. La depuración de una jerarquía óptima es delicada e iterativa. Los árboles de clases no crecen a partir de una raíz. Por el contrario, se determinan partiendo de las hojas porque éstas pertenecen al mundo real mientras que los niveles superiores son abstracciones construidas para ordenar y comprender.

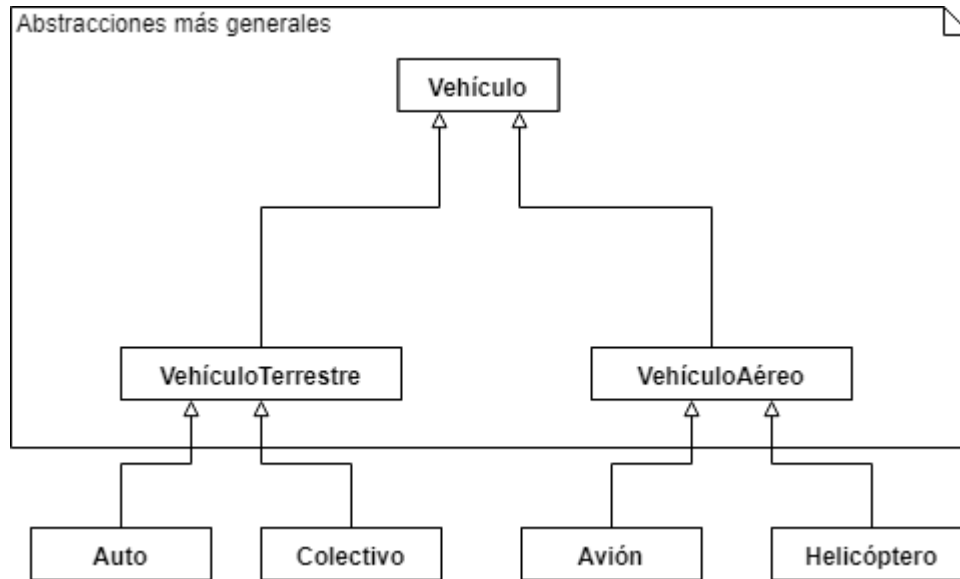
La **especialización** permite capturar las particularidades de un conjunto de objetos no discriminados por las clases ya identificadas. Las nuevas características se representan por una nueva clase, subclase de una de las clases existentes. La especialización es una técnica eficaz para la extensión coherente de un conjunto de clases.

Una **subclase** identifica el comportamiento de un conjunto de objetos que hereda las características de la clase padre y adicionan algunas específicas que ésta no posee.

UML utiliza la flecha  para indicar la **Especialización /Generalización**, esta indica que una subclase comparte los métodos y atributos especificados en una Clase Superior, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos de la Clase Superior.

¹⁴ Modelado de Objetos. op. cit. Pág.46

El ejemplo siguiente muestra una jerarquía de medios de transporte.

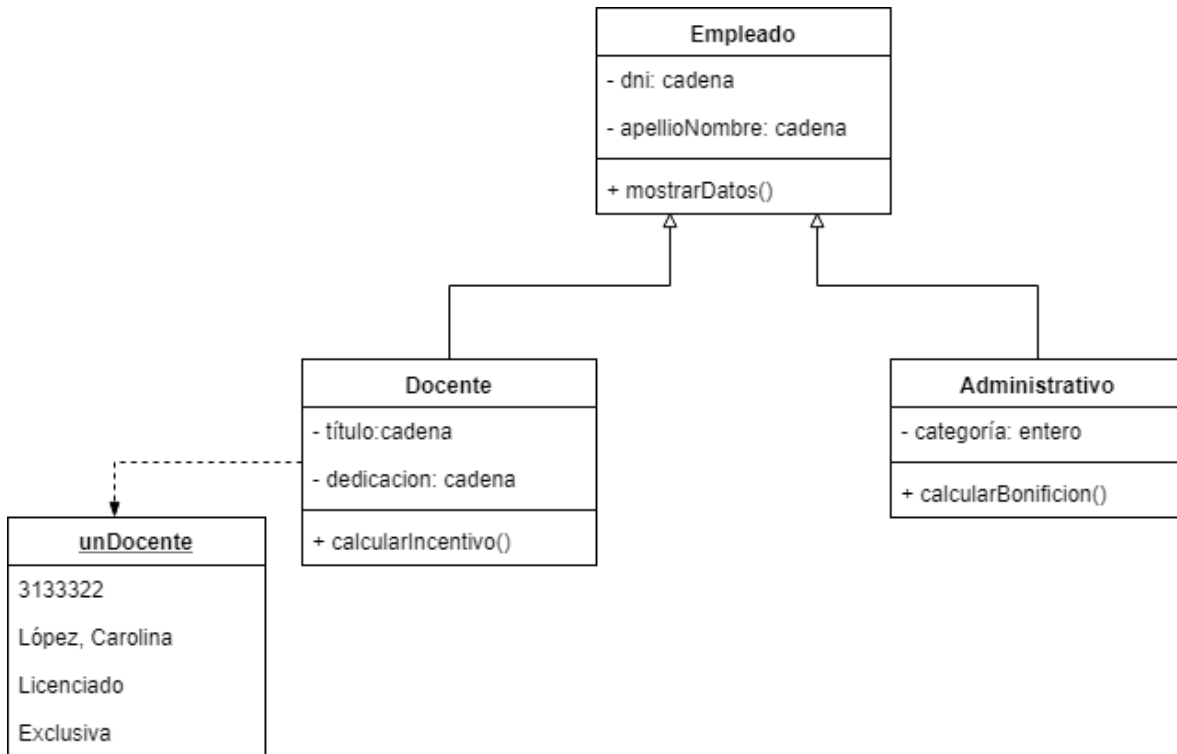


La **herencia** es el mecanismo que permite compartir automáticamente métodos y datos entre clases y subclases. Los conceptos de **Especialización /Generalización** se objetivan a través de la **herencia**.

Este mecanismo potente, permite crear nuevas clases a partir de clases existentes programando solamente diferencias.

La generación de subclases es la creación de una nueva clase que hereda de una clase existente. La única tarea de la subclase es indicar las diferencias de comportamiento y atributos entre ella y la superclase.

Ejemplo: A continuación se muestra una jerarquía de clases que forma parte de un Sistema de Administración de empleados de una universidad.



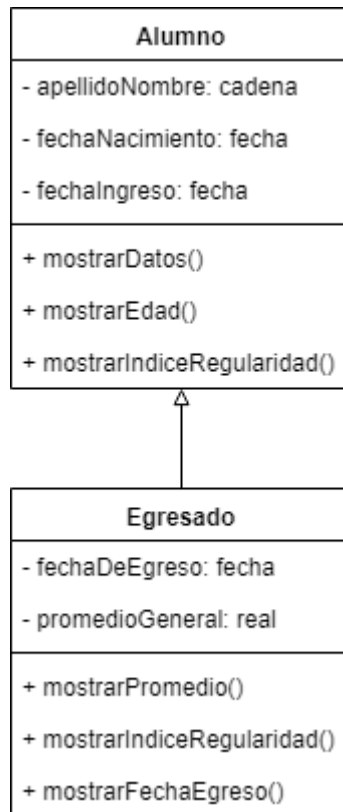
Las subclases **Docente** y **Administrativo** comparten los atributos y métodos de la clase superior **Empleado**. La subclase **Docente** por ejemplo, agrega atributos y comportamientos propios.

Clase Abstracta: Una clase es abstracta cuando no existe un objeto que sea instancia directa de ella, pero sí existe una subclase de ella que es instanciable.

Generalmente se utilizan para resumir los comportamientos comunes a un conjunto de subclases. El nombre de las clases abstractas va en cursiva. Por ejemplo, la **clase Empleado** de la jerarquía anterior es abstracta.

Clase Concreta: Una clase concreta es aquella que es instanciable, es decir que existe al menos un objeto de esa clase. La **clase Docente** es una clase concreta. El objeto **unDocente** representa una instancia de la clase **Docente** que además de los atributos de la clase empleado tiene los propios de la clase de la cual es instancia.

Ejemplo: Suponiendo el siguiente diagrama de clases, que responde a un Sistema de Pedido de Becas, consideremos solamente la jerarquía alumno - egresado.



A partir de la clase **Alumno**, se ha construido la clase **Egresado**, con quien comparte atributos y comportamientos. En esta jerarquía tanto **Alumno** y **Egresado** son clases concretas, pues ambas son instanciables.

La clase **Egresado** comparte características de la clase **Alumno**, contiene por herencia los atributos *nombre*, *fechaNacimiento* y *fechaIngreso*, y los métodos *mostrarDatos()*, *mostrarEdad()*. Agrega además características particulares a través de los atributos *fechaEgreso* y *promedioGeneral*, y de los métodos *mostrarPromedio()*, *mostrarIndiceRegularidad()* y *mostrarFechaEgreso()*.

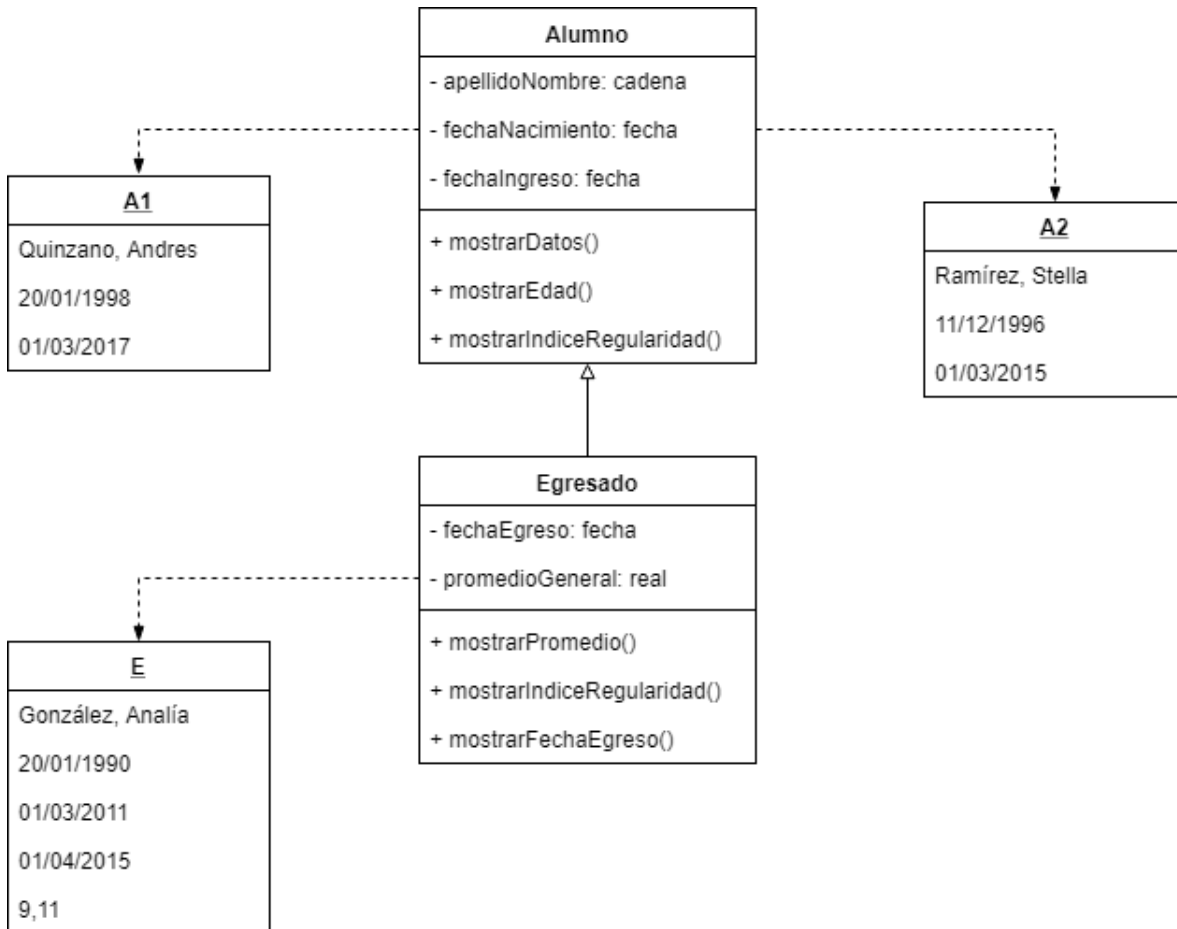
Por otro lado el método *mostrarIndiceRegularidad()*, que calcula el índice de regularidad, aparece tanto en la clase superior (**Alumno**) como en la clase descendiente (**Egresado**). Esto se debe a que el cómputo del índice de regularidad varía según la clase a la que pertenezca el objeto que recibe el mensaje. Por lo tanto, si bien el mismo método aparece en las dos clases, el método de la clase **Egresado** redefine al de la clase **Alumno** a través de un código distinto. Esto es, el método **mostrarIndiceRegularidad** de la clase **Egresado** oculta al método **mostrarIndiceRegularidad** de la clase **Alumno**.

Así, cuando un objeto de la clase **Egresado** recibe el mensaje **mostrarIndiceRegularidad**, ejecuta el método **Egresado.mostrarIndiceRegularidad**. Como conclusión podemos decir que, dentro de una jerarquía de clases, la herencia propaga las características de la clase superior en sus clases descendientes, de modo que varias clases pueden compartir una misma descripción.

Sean:

A1, A2: Alumno; // objetos de la clase **Alumno**

E: Egresado; // objeto de la clase **Egresado**



- **A1.mostrarEdad():** este **mensaje** produce como respuesta la ejecución del **método Alumno.mostrarEdad()** definido en la clase Alumno. Este método calcula la edad a partir de la fecha de nacimiento 20/01/1998.
- **A2.mostrarEdad():** este **mensaje** produce como respuesta la ejecución del **método Alumno.mostrarEdad()** definido en la clase Alumno. En este caso el método calcula la edad a partir de la fecha de nacimiento 11/12/1996.
- **E.MostrarEdad():** en este caso el sistema busca el método **mostrarEdad** en la clase Egresado; como la clase no contiene ese método busca en la clase superior y, en respuesta al mensaje recibido, ejecuta el código asociado al **método**. En este caso el método calcula la edad a partir de la fecha de nacimiento 20/01/1990.
- **mostrarPromedio():** este **mensaje** produce como respuesta la ejecución del **método E.mostrarPromedio()** definido en la clase Egresado. Muestra el valor 9,11.

Variables de clase y variable de instancia

Se denominan variables o atributos de clases a aquellos atributos que tienen el mismo valor para cada objeto de la clase. A los atributos que no son de clase se los llama de instancia. Así, el valor de las variables de clase será el mismo para cualquier instancia de la clase. En una lista de objetos, una variable de clase, podría ser la cantidad de nodos, que se actualiza en dos momentos: cada vez que se crea un nodo, suma uno a la cantidad, cada vez que se elimina un nodo de la lista, se decrementa en uno la cantidad, esta variable de clase se puede utilizar para validar que se haga la limpieza correcta de la memoria utilizada por los nodos de la lista, esto es, cuando termina el programa, si la variable termina en cero, significa que se liberó toda la memoria solicitada para el almacenamiento de los nodos, caso contrario, quedan referencias a memoria que no se utiliza. Si el valor de la variable de clase es cambiado para una instancia, el mismo cambia para todas las instancias de la clase y subclase.

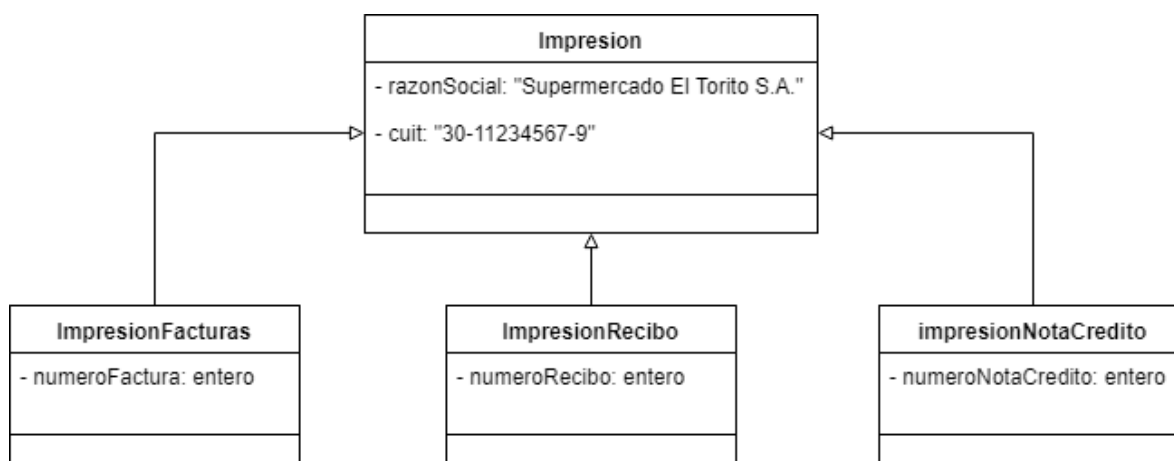
Una variable de clase es un elemento de información que define un atributo de toda una clase. La variable se aplica para la clase por sí misma y para todas sus instancias, de modo que solo se almacena un valor, no importa cuántos objetos de clase hayan sido creados.

Se denominan variables o atributos de instancia a aquellos atributos cuyo valor cambiar para cada objeto de la clase.

Una variable de instancia es un elemento de información que define un atributo de un objeto en particular.
La clase del objeto define qué tipo de atributo es y cada instancia almacena su propio valor para ese atributo. A las variables de instancia también se les denomina variables de objeto.

Por ejemplo:

La siguiente jerarquía de clase modela la impresión de todo documento escrito que emita el “Supermercado EL Torito”.



La jerarquía muestra una clase superior abstracta, que resume las características comunes de las subclases.

La clase **Impresión** posee al menos dos variables de clase, **razonSocial** y **cuit**. Estos atributos deben estar presentes en cualquier documento escrito de la empresa.

La clase **ImpresiónFacturas** posee al menos tres atributos, dos son variables de clase, **razonSocial** y **cuit**, y uno es de instancia que es **numeroFactura**.

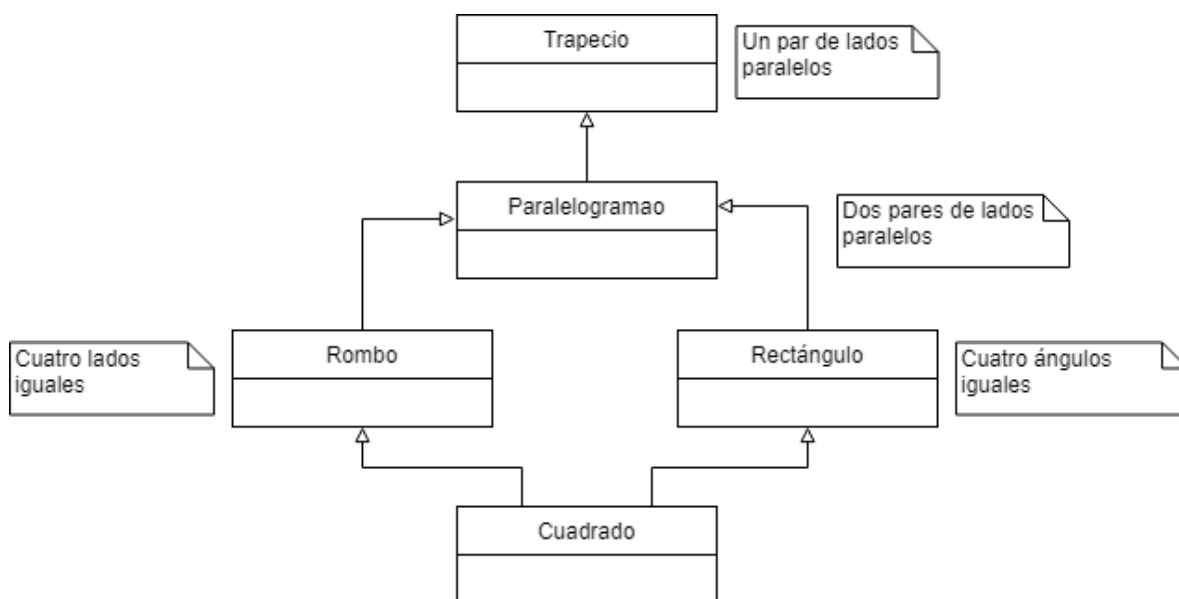
Tipos de herencia

Existen dos tipos de herencia: *herencia simple* y *herencia múltiple*.

Con la herencia simple una subclase puede heredar datos y métodos de una única clase y también puede añadir nuevo comportamiento. Son ejemplos de herencia simple, la jerarquía Alumno-Egresado y la jerarquía Impresión-Factura, antes presentadas.

Con la herencia múltiple una subclase puede adquirir los datos y métodos de más de una clase padre, como lo muestra la siguiente gráfica.

El siguiente ejemplo muestra, dentro de los cuadriláteros, una jerarquía que clasifica los distintos tipos de cuadriláteros trapecios.



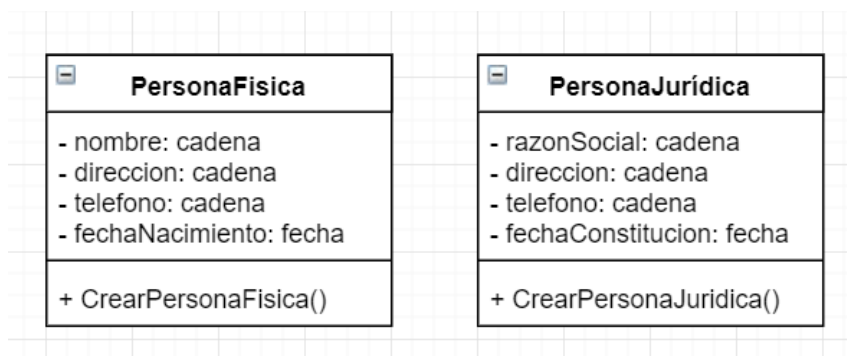
En resumen:

Los elementos del mundo a ser modelado y su comportamiento son transformados en objetos. Los objetos con comportamientos comunes son organizados en clases. Las clases se organizan en jerarquías y los mecanismos de herencia proporcionan a cada subclase los métodos y datos de la clase padre. Al desarrollar una nueva aplicación se consideran las clases provistas o generadas a partir de las cuales se generan las subclases necesarias.

ACTIVIDAD 4

A- **Contexto:** Un estudio contable tiene como clientes personas físicas y personas jurídicas. Es necesario disponer de un registro de datos de clientes para agilizar la búsqueda cuando estos son requeridos.

1. A partir de las clases dadas, aplique generalización o especialización para diseñar una jerarquía de clases adecuada para el contexto planteado.



Nota: indique que mecanismo aplicó (generalización o especialización).

2. En la jerarquía diseñada, identifique clases concretas y clases abstractas.

3. Cree una instancia de cada clase.

B- **Contexto:** Una fábrica de pastas frescas elabora los siguientes productos:

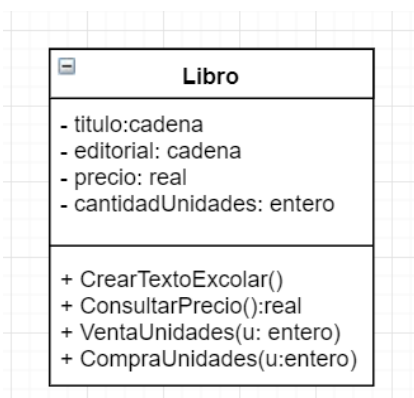
- fideos frescos: se elaboran dos tipos comunes y para celíacos. A su vez, cada tipo de fideo se presenta en caja de 500grs. y de 1000grs. El precio se establece de acuerdo con el tipo y al peso.
- raviolos: se elaboran de variedad pollo, ricota y cuatro quesos. Cualquiera sea la variedad, se presenta en caja de 500grs. y 1000grs. El precio se establece de acuerdo con el peso.
- tapas de empanadas: se presentan en paquete de 12 y 24 unidades cada uno. Se fabrican con masa criolla y masa hojaldre. El precio del paquete se establece de acuerdo con la cantidad de unidades y el tipo de masa.
- tapas de tarta: se presentan en paquete de 2 unidades cada uno, con separador y sin separador, esto última marca una distinción en el precio.

De acuerdo con el contexto planteado:

1. Identifique las características de cada tipo de producto y diseñe una clase para cada uno de ellos.
2. Aplique un mecanismo (generalización o especialización) al conjunto de clases del ítem anterior y diseñe una jerarquía de clases. Especifique el mecanismo aplicado.
3. Señale clases abstractas y concretas.
4. Represente en UML un objeto de cada clase.

C- **Contexto:** La librería “Sr Leo”, vendía sólo libros de textos escolares para los distintos niveles (primario, secundario, etc.). Con el objetivo de incrementar las ventas, los dueños han decidido ampliar el rubro. A partir de este año incorpora la venta de novelas. Actualmente se usa un sistema informático que registra los libros disponibles para la venta. Los dueños han solicitado modificar el sistema actual para que permita el registro de los datos las novelas; y además registre otras características de los libros escolares (además de los datos que ya se registran), porque con el uso del sistema actual han detectado que esto necesario.

El sistema actual funciona con la siguiente clase para representar los libros de textos escolares.



Ahora, los usuarios requieren:

1. Para el libro escolar además de los datos que ya se han registrado, se almacene el nivel (primario, secundario, etc.) al que va dirigido.
2. Para una novela registrar los mismos datos que hasta el momento el sistema registra para los libros escolares (título, editorial, precio, cantidad de unidades) en conjunto con el autor (en caso de ser más de uno, se registran los nombres separados por guion medio) y el tipo (romántica, crónica, etc.).

Para el contexto planteado:

1. Reutilizando la clase existente, diseñe una jerarquía de clases que contemple los requerimientos de los usuarios. Especifique el mecanismo (generalización o especialización) aplicado.
2. Señale clases abstractas y concretas.
3. Represente en UML los siguientes libros. Nota: de cada uno hay 10 unidades disponibles



7. RELACIONES ENTRE CLASES

En la primera etapa del DOO, se deciden las características de cada clase, es decir se identifican las clases. Una vez realizado esto, se trata de decidir el modo en que las clases se vinculan otras clases. Aunque algunas clases pueden existir aisladas, la mayoría no puede, y deben cooperar unas con otras.

Las relaciones entre las clases expresan una forma de acoplamiento entre ellas. Según el tipo de acoplamiento que presentan las clases podemos distinguir distintos tipos de relaciones. Entre ellas podemos citar:

- ✓ Asociación
- ✓ Agregación y Composición
- ✓ Herencia

Multiplicidad de las relaciones

En UML la multiplicidad indica el número de instancias (cardinalidad) que participan en la relación, y se anota en cada extremo de la relación.

Los valores de multiplicidad más comunes son:

1 uno y sólo uno

0..1 cero o uno

m..n de m a n (m y n enteros naturales)

***** de cero a varios

0..* de cero a varios

1..* de uno a varios

Asociación

La relación de asociación representa una conexión semántica bidireccional entre dos clases.

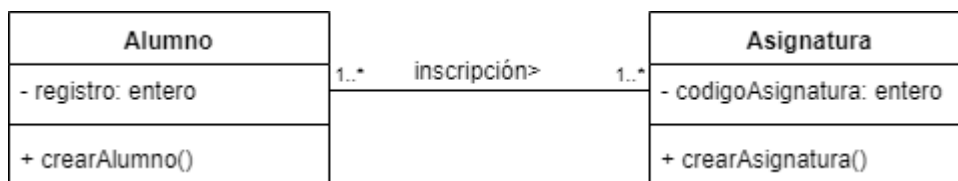
Una asociación es una abstracción de los vínculos que existen entre objetos instancias de las clases asociadas.

La asociación no es contenida por las clases, ni subordinada a las clases asociadas; es el reflejo de una conexión que existe en el ámbito de la aplicación.

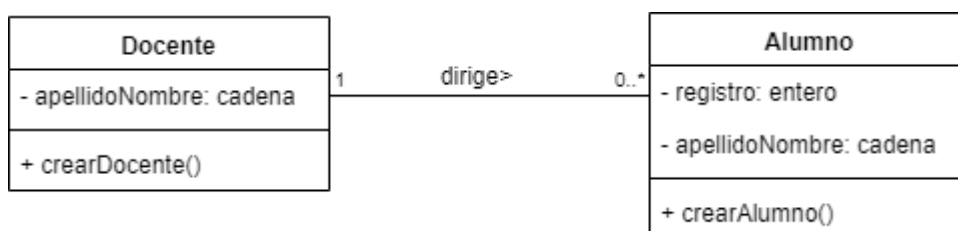
Las asociaciones pueden ir acompañadas de un nombre que permite una mejor legibilidad del diagrama y se representa por una línea continua entre las clases asociadas.

Ejemplos:

Un alumno puede inscribirse en varias asignaturas, una asignatura permite la inscripción de varios alumnos.



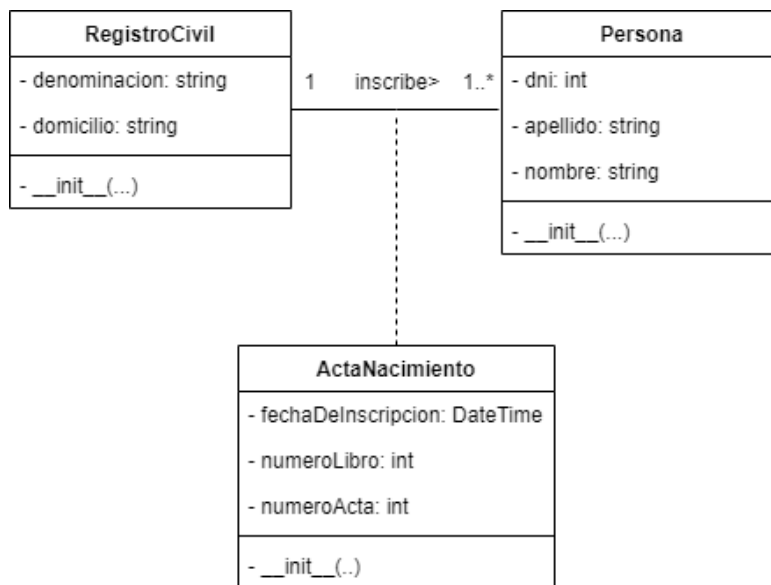
Un docente puede ser director de tesis de varios alumnos, un alumno puede tener un solo director de tesis.



Clase de Asociación

En algunas situaciones es necesario mantener la información que es específica de la propia asociación.

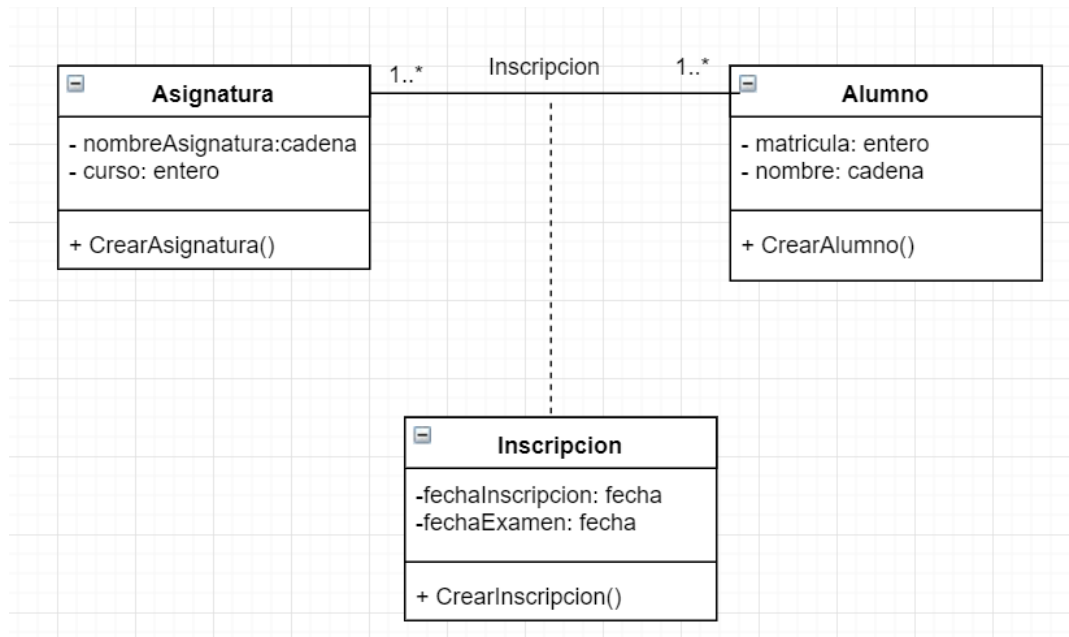
Supongamos un sistema que almacena información sobre los registros civiles de la provincia de San Juan, y las personas nacidas en la jurisdicción del mismo. Los nacimientos se registran en un acta de nacimiento, que posee entre otros datos: número de acta, fecha de inscripción, número de libro, etc. Como se puede observar en este ejemplo, si bien, hay una asociación entre Registro Civil y Persona, los datos propios del acta de nacimiento, no son datos inherentes al Registro Civil, ni a la Persona, esta información debe ser almacenada como atributos de una **clase asociación**.



Principio Fundamental:

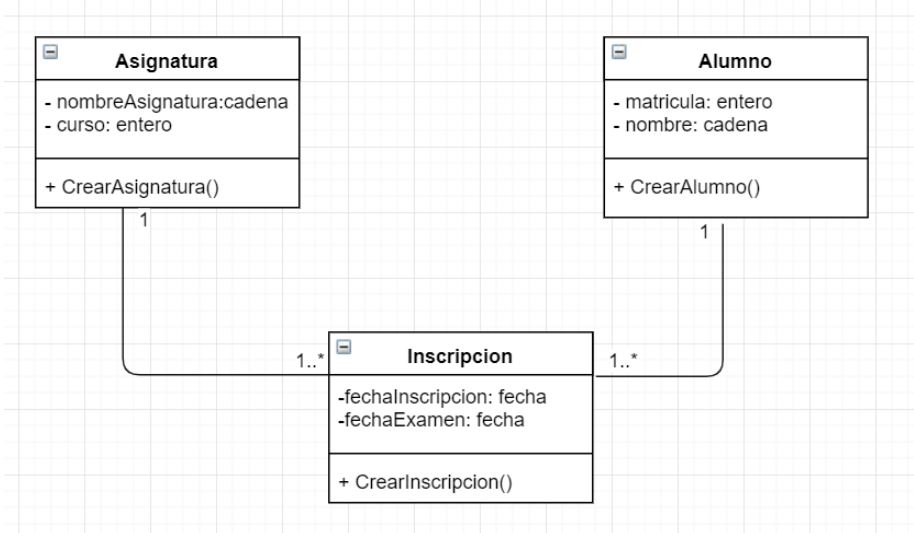
Una dupla de objetos, instancias de cada una de las clases que participan en la asociación, se relaciona con una **única** instancia de la clase asociación.
No importa la multiplicidad en ambos extremos.

Analicemos un sistema contempla las inscripciones a **exámenes por turno (por ejemplo turno del mes de julio)**, donde cada alumno puede inscribirse en una o más asignaturas, y por cada inscripción recibe un comprobante.



Para la asociación entre las clases **Asignatura** y **Alumno** de este sistema, puede crearse una clase asociación –**Inscripcion**– que guarde información de la inscripción a examen, que realiza el alumno (fecha de inscripción, fecha de examen, etc.). Un objeto instancia de **Inscripcion** está asociado a un único y alumno y una única materia.

Lo anterior, cambia si el sistema contempla **TODAS LAS MESAS DE EXÁMENES DE JULIO** (dos turnos) donde el alumno podría inscribirse dos veces en la misma asignatura.



Para el par formado por un determinado objeto de la clase Asignatura y un determinado objeto de la clase Alumno, existe más de un objeto de la clase Inscripcion asociado a ella. En este caso no podemos definir a Inscripcion como una clase asociación, sino que existe una relación de asociación entre Asignatura y Alumno **que se modela a través de** Inscripcion.

ACTIVIDAD 5

Narrativa

La librería “Sr Leo” (Actividad 4-C) requiere una nueva modificación en el sistema que le permita registrar nuevos datos de las editoriales y los autores de los libros. Estos son:

- 1- Nombre y lugar de la sede central de cada editorial.
- 2- Nombre y nacionalidad de cada autor.

Para el contexto planteado se pide:

- 1- Diseñar una clase para representar las editoriales.
- 2- Diseñar una clase para representar los autores.
- 3- Modificar las clases existentes en caso de ser necesario (jerarquía diseñada en la actividad 4)
- 4- Relacionar las clases teniendo en cuenta lo siguiente:
 - a) Un libro puede tener uno o más autores, y un autor puede escribir uno o más libros.
 - b) Una editorial edita distintos libros, y un libro editado por una editorial.
 - c) La editorial puede editar un libro más de una vez.
 - d) De cada vez que la una editorial ha editado un libro es necesario registrar el número de edición, el año y la cantidad de ejemplares.

Agregación ¹⁵

La forma más simple de reutilizar una clase es simplemente haciendo una **nueva clase que la contenga**. Esta técnica se llama Agregación.

Así, una agregación es una asociación *no simétrica* (todo/parte) en la que una de las clases cumple un papel predominante respecto de la otra. Por otra parte, la agregación declara una dirección a la relación todo/parte. Gráficamente se representa colocando un rombo del lado de la clase agregado.

Existen dos formas de este tipo de relaciones, **la agregación y la composición**.

- Con la composición el objeto parte puede pertenecer a un todo único.
- En la relación de composición, si muere el objeto que dispara la composición, los objetos que lo componen mueren con él.
- En la agregación, pasa lo contrario.

Algunos criterios para detectar agregaciones¹⁶:

- Los objetos de una clase están subordinados a los objetos de otra clase.
- Una acción sobre una clase implica una acción sobre otra clase
- Los valores de los atributos de una clase se propagan en los valores de los atributos de otra clase

En caso de duda son preferibles las asociaciones, ya que en general siempre hay que elegir una solución que implique el acoplamiento más débil.

¹⁵ Resulta interesante ver los ejemplos planteados en
http://www.unirioja.es/cu/jeaansa/0910/archivos/EIPR_Tema02.pdf

¹⁶Modelado de Objetos. op. cit. Pág103

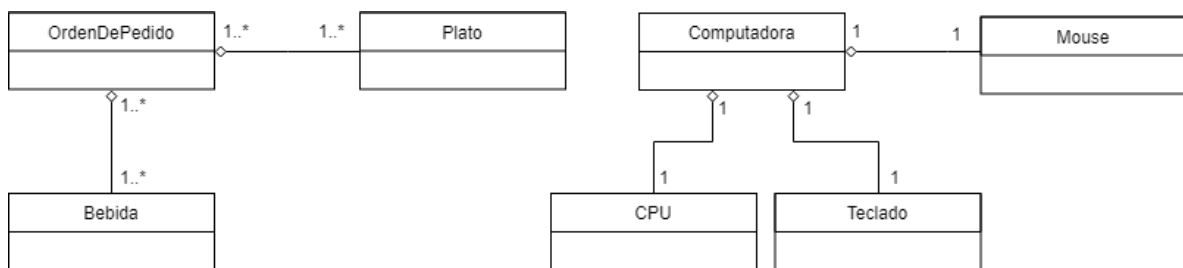
Relación de Agregación¹⁷

Este tipo de relación se presenta en aplicaciones en las cuales un objeto contiene como partes a objetos de otras clases., pero de tal modo que la destrucción del objeto continente no implica la destrucción de sus partes.

En este caso se dice que el objeto continente o contenedor incluye referencias a objetos de otras clases y que los objetos contenidos pueden existir independientemente del objeto contenedor. Por consiguiente los tiempos de vida de esos dos objetos no están tan estrechamente acoplados, de modo que se pueden crear y destruir instancias de cada clase independientemente.

La agregación se representa por una flecha con un rombo sin relleno

Ejemplos:



Una orden de pedido incluye uno o más objetos de Bebida, y uno o más objeto de Plato. La destrucción de un objeto Orden de Pedido, no implica la destrucción de los objetos Bebida o Plato. Análisis similar para un objeto computadora. Los objetos de las clases CPU, teclado y pantalla tienen existencia propia independiente de la clase Computadora.,

Como se observa, no hay restricciones sobre la multiplicidad del agregado,

Relación de Composición¹⁸

Siguiendo con lo antes expuesto, hay agregaciones donde los objetos agregados no tienen sentido fuera del objeto resultante.

Este es el caso de la composición la cual aparece cuando los atributos de una clase hacen referencia a objetos de otra clase y éstos están físicamente contenidos por el agregado.

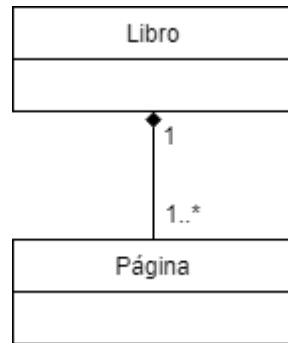
Así, un objeto de la nueva clase (contenedor) incluye o contiene objetos de otras clases ya definidas pero cuya existencia solo tienen sentido en este contexto.

En este caso, *la vida de los objetos incluidos depende de la vida del objeto que los incluye*. La composición se representa con un **rombo relleno** que apunta al agregado.

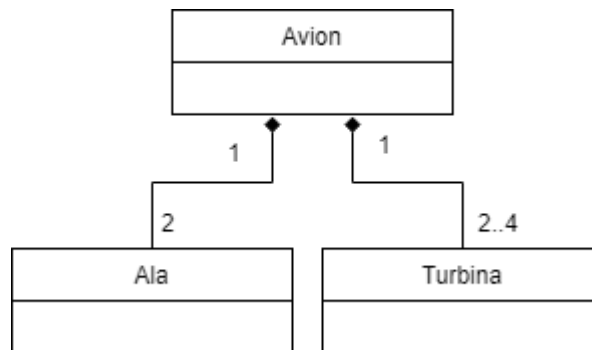
¹⁷ En UML 1 se la denomina *agregación de contenido por referencia*

¹⁸ En UML 1 se la denomina *Agregación de contenido por valor*

Ejemplo:



Para explicar porque es una relación de composición (agregación fuerte) cabe hacer notar que los objetos de la clase página tienen sentido en tanto y en cuanto forman parte de un objeto de la clase libro (que tiene asignado su ISBN, autor, editorial). Fuera de esa de esa clase no ofrecen información de mayor utilidad.



En este ejemplo un Avión es un objeto compuesto de otros objetos, incluye alas, turbinas, tren de aterrizaje etc. Esto significa que el objeto ala o hélice no existe independientemente de la instancia de Avión que los contiene. La vida de los objetos Avión y ala o hélice están interrelacionadas, de modo que cuando se crea una instancia de Avión se crea una instancia de la clase ala o hélice, y cuando se destruye un objeto de Avión, por implicación se destruyen también los correspondientes objetos ala o hélice.

En este caso de agregaciones, la multiplicidad del lado agregado puede tomar el valor 0 o 1. El valor 0 se refiere a un atributo no explicitado.

Reutilización y extensión¹⁹:

Se denomina reutilización al uso de clases u objetos desarrollados y probados en un determinado contexto, para incorporar esa funcionalidad en una aplicación diferente a la de origen. En este sentido el uso de una clase ya constituye reutilización del código.

La extensión se basa en aprovechar las clases desarrolladas para una aplicación, utilizándolas para la construcción de nuevas clases, en la misma u otra aplicación. De todas formas, como la extensión es una forma particular de reutilización –usando una

¹⁹ <http://apdaza-poo.blogspot.com.ar/2008/04/reutilizacin-del-codigo-y-relaciones.html>

acepción más amplia de término-, se suele englobar bajo el concepto común de reutilización.

Un caso típico de extensión se logra a través de la herencia, como ya se ha visto anteriormente. Así, a partir de una clase desarrollada pueden generarse una o más subclases que posean características particulares, que heredan atributos y comportamientos de la clase padre.

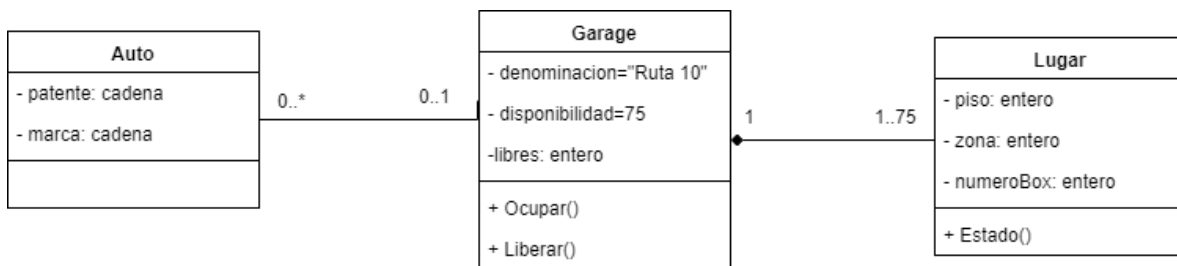
La forma más simple de reutilizar una clase es simplemente haciendo una **nueva clase que la contenga**.

Diagrama de clases

Un diagrama de clases es un diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos

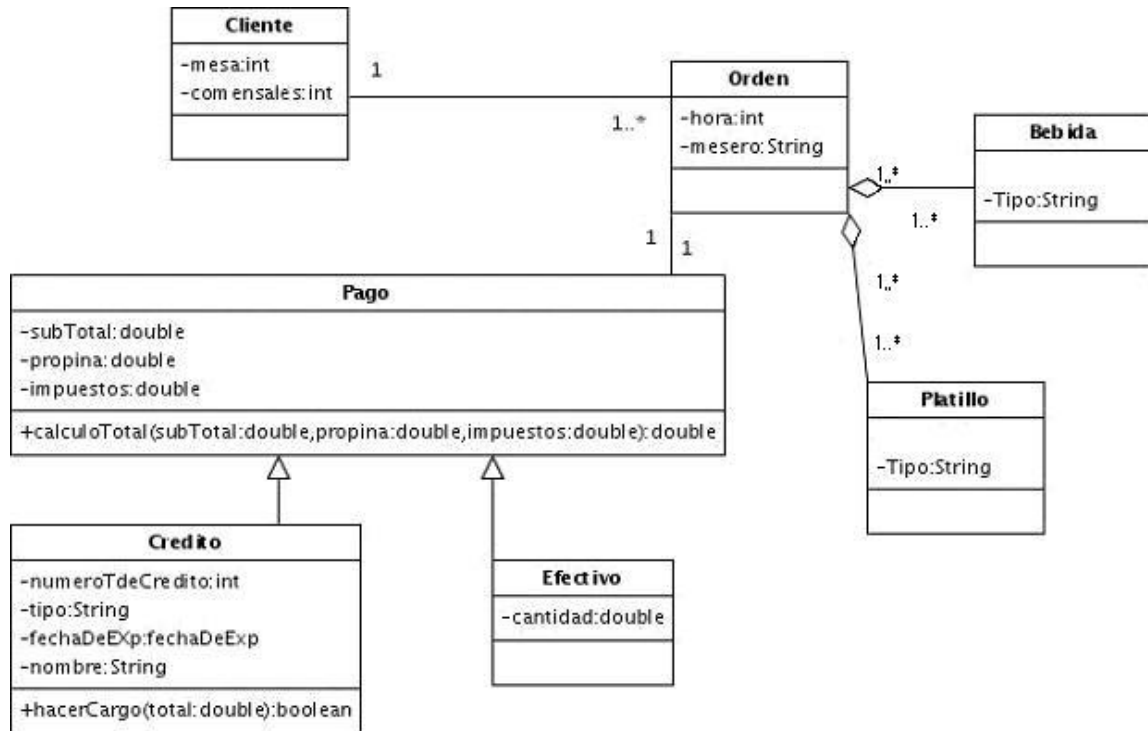
Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas.

Ejemplo 1: El siguiente diagrama de clases modela un sistema que registra el estacionamiento en el garaje “Ruta 10”, que tiene espacio para estacionar 75 autos.



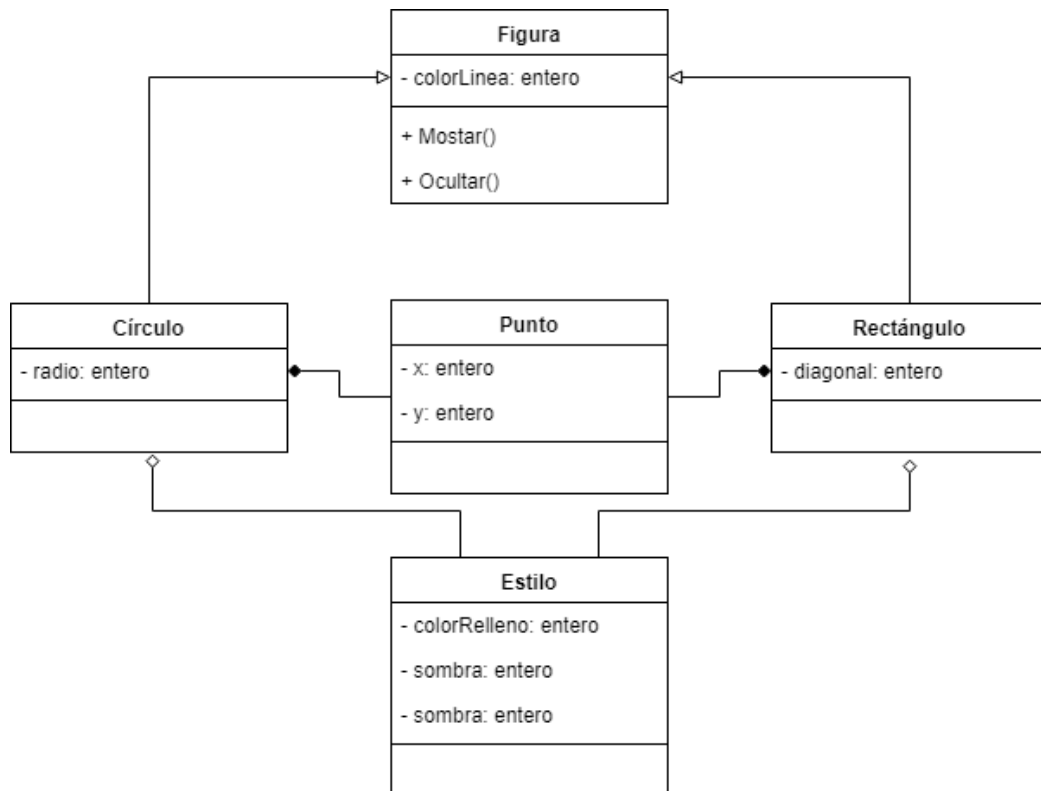
En este ejemplo, todos los atributos de la clase **Auto**, al igual que los de la clase **Lugar**, son variables de instancia. La clase **Garage** tiene dos atributos que son variables de clase: **denominacion** y **disponibilidad**.

Ejemplo 2: El siguiente diagrama de clases modela un sistema de facturación en un restaurante.



Ejemplo 3: El siguiente diagrama modelo una aplicación que grafica figuras que entre sus atributos tiene un estilo (características propias del diseño) y un punto (centro de la figura).

Coloque la multiplicidad.



ACTIVIDAD 6: Relacione las clases dadas de acuerdo con el contexto planteado.

Contexto:

El jardín maternal “Lucecita” requiere un sistema para manejar la información de los niños que asisten en el presente año.

Se reciben niños desde los 45 días hasta los 3 años de edad. De cada uno de ellos se registra los datos de al menos una persona responsable del niño (padre, madre y/o tutor). Pudiendo ser una misma persona responsable de más de un niño.

El jardín dispone de distintas salas. Un niño se asigna a una sala según la edad, y se mantiene en la misma sala durante todo el año.

La cantidad de docentes a cargo de una sala varía en función de la cantidad de niños que asisten a la sala. En un día, una docente solo puede estar a cargo de una sala, pero en el transcurso del año, una misma docente puede estar a cargo de diferentes salas. Por ello es necesario contar con un registro diario de periodo de tiempo que una docente estuvo a cargo de una sala.

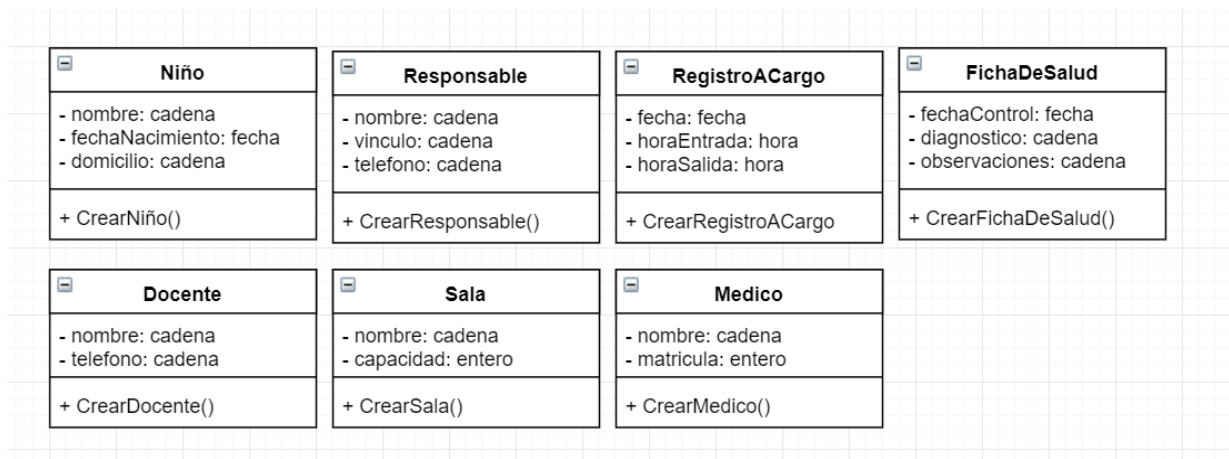
El jardín cuenta con la asistencia médica de un grupo de pediatras para que controlen el estado general de salud de cada niño que se inscribe en el jardín. Cuando el médico controla a un niño completa la ficha de salud del niño. El niño es controlado solo por un médico.

El sistema requerido para manejar la información del jardín debe dar solución a los siguientes requerimientos de los dueños del jardín.

- Alta, baja, modificación para un niño, para un responsable, de una ficha de salud, de un registro diario de docente a cargo.
- Dado el DNI de un niño, mostrar las datos las personas responsables del niño.
- Dado el nombre de una sala, listar los nombres de los niños que asisten a la sala dada.
- Dada el nombre de una sala y una fecha, mostrar los nombres de las docentes que estuvieron a cargo de la sala en la fecha dada.
- Dado el DNI de un niño, mostrar los datos de la ficha de salud del niño con DNI dado y el nombre del médico que lo controló.

No es de interés resguardar los datos de los años anteriores al actual.

Para el contexto planteado se han identificado las siguientes clases:



ACTIVIDAD 7

En cada uno de los contextos presentados:

- 1 Identifique las clases que intervienen.
- 2 Diseñe el diagrama de clases (relaciones entre clases) de acuerdo con lo que el contexto planteado requiere.
- 3 Realice el **diagrama de secuencia** para la funcionalidad que se indica, utilizando estereotipos.
- 4 A partir del diagrama de secuencia construido, identifique los métodos necesarios y agréguelos en la clase correspondiente.

A- Contexto: Agencia de Turismo

La agencia de turismo “Por el mundo” que tiene sucursales en todo el país, lo ha contratado para desarrollar una aplicación que mantenga información de los paquetes turísticos ofrecidos por sus sucursales.

De cada sucursal se almacena el nombre, teléfono y dirección. Cada sucursal ofrece diferentes paquetes turísticos formados por uno o más destinos. De cada uno de los paquetes se debe almacenar un código y la duración en días. Mientras que para los destinos se necesita mantener su nombre, la provincia en la que se encuentra y el país.

Las sucursales pueden vender el mismo paquete para distintas fechas, por lo que para cada venta que una sucursal hace de un paquete se debe registrar el precio, fecha de salida y fecha de regreso.

Además, un paquete puede incluir distintos servicios. Existen dos tipos de servicios: alquiler de vehículo y excursiones. De un alquiler de vehículo se debe conocer la fecha, tipo de vehículo y cantidad máxima de pasajeros. De las excursiones se registra la fecha, actividad a realizar y vestimenta recomendada.

Restricciones:

- Un paquete se puede vender en una o varias sucursales y cada sucursal puede vender uno o varios paquetes
- Una venta tiene un paquete y una sucursal.
- Un paquete puede incluir uno o más servicios.
- Un servicio puede ofrecerse en distintos paquetes.
- Un paquete puede incluir uno o más destinos.
- Un destino puede pertenecer a varios paquetes.

Considerando el diagrama de clases elaborado en el ítem anterior y la descripción de la funcionalidad dada a continuación construya el **diagrama de secuencia**.

Funcionalidad: Un usuario selecciona la opción de “Ver paquete”, la aplicación debe solicitar el código del paquete. El usuario ingresa el código del paquete y la aplicación deberá mostrar la duración, el nombre de cada destino incluido en el paquete y la fecha y tipo de cada servicio incluido en el paquete.

A- Contexto: empresa de servicio de streaming

La empresa FlixNet presta el servicio de streaming de contenido multimedia bajo demanda por Internet a los usuarios registrados. Actualmente la empresa sólo provee series.

Para mejorar la experiencia de sus usuarios, requiere una aplicación que permita llevar el registro de los capítulos que un usuario aún no termina de ver. Este registro consiste en mantener el minuto del capítulo en que se detuvo la reproducción por parte de un usuario.

No hay restricciones en cuanto a la reproducción de los capítulos, esto significa que un usuario puede haber detenido la reproducción (en distinto momento) de más de un capítulo, de la misma serie o de distintas series.

Una serie puede tener una o más temporadas. Una temporada tiene uno o más capítulos. Toda serie tiene un género.

De cada género se registra el nombre.

De una serie se registra el nombre, el género, la cantidad de temporadas y sus temporadas.

De una temporada se registra el número, la cantidad de capítulos y sus capítulos.

De un capítulo se registra un código interno de identificación, el número, el título (si lo tiene), la duración en minutos y una breve descripción.

De un usuario registra su nombre, apellido, correo electrónico y clave de acceso.

Cuando un usuario reproduce un capítulo se registra el minuto del capítulo en que se detuvo la reproducción.

Considerando el diagrama de clases elaborado en el ítem anterior y la descripción de la funcionalidad “Ver serie” dada a continuación construya el **diagrama de secuencia**.

Funcionalidad: Un usuario selecciona la opción de “Ver serie”, la aplicación debe mostrar el nombre de cada género. El usuario selecciona un género y la aplicación deberá listar, de cada serie que pertenece al género seleccionado, el nombre y la cantidad de temporadas. El usuario selecciona una serie. La aplicación muestra por cada temporada, el número de temporada y de cada uno de sus capítulos el número, el título, la duración y, en caso de tratarse de un capítulo que el usuario ha visto, el minuto en el que detuvo la reproducción. El usuario selecciona un capítulo y la aplicación muestra el mensaje de “Espere mientras se carga el capítulo seleccionado”.

B- Contexto: aplicación de juego

Se requiere de una aplicación para un juego en el cual el jugador participa a través de un personaje.

El juego consiste en que el personaje supere una serie de niveles. Cada nivel se conforma por un conjunto de misiones. El nivel se ha superado cuando el personaje ha cumplido todas las misiones que lo conforman.

Cuando un personaje cumple una misión se registra el desempeño en el cumplimiento.

Según este desempeño será la cantidad de dinero que gana. Cada vez que el personaje supera un nivel recibirá vidas.

Cuando un personaje cumple una misión, el desempeño logrado puede ser bueno, muy bueno o brillante. Un personaje puede cumplir una misión más de una vez; y se debe tener en cuenta que, para cada vez, que el personaje cumple una misión se registra el desempeño alcanzado y la fecha en que lo hizo.

El personaje adquiere recursos que son necesarios para cumplir misiones. Estos recursos pueden ser materiales o herramientas. De un recurso material, por ejemplo: madera, sogas, etc., se registra el nombre del recurso y el precio que debe pagar el personaje para adquirirlo. De un recurso herramienta, por ejemplo: martillo, sierra, etc. se registra el nombre del recurso y la función de la herramienta (rompe piedras, corta madera, etc.). Además, cada herramienta requiere niveles superados para obtenerla. Por ejemplo, para adquirir el martillo se requiere haber superado el nivel 1 y 3. De cada recurso adquirido

por el personaje se registra la cantidad adquirida. Un personaje puede adquirir el mismo recurso más de una vez, en ese caso la cantidad adquirida se suma a la cantidad que disponía previamente.

De un nivel se registra el número de nivel y la cantidad de vidas que obtendrá el personaje cuando lo supere. De una misión se registra el nombre de la misión y la cantidad de dinero que gana el jugador para cada uno de los posibles desempeños (bueno, muy bueno y brillante).

De un personaje se registra su nombre, la imagen (nombre del archivo que contiene la imagen), cantidad de vidas disponibles y cantidad de dinero disponible.

A partir del diagrama de clases elaborado anteriormente, realice el diagrama de secuencia para la siguiente funcionalidad:

Adquirir un recurso material: El personaje solicita adquirir un material y se le muestra un listado con todos los materiales. El personaje elige uno de ellos e ingresa la cantidad a adquirir. En ese momento se le muestra el precio total (precio del material * cantidad ingresada) del material que desea adquirir y, en caso de que el dinero que el personaje dispone no sea suficiente para adquirirlo, se muestra la cantidad de dinero faltante. En cambio, si es suficiente, pide que confirme la compra. El personaje confirma la compra y se registra la cantidad adquirida del recurso y se actualiza la cantidad de dinero disponible.

8. Conceptos Claves

Polimorfismo

El polimorfismo puede definirse como la capacidad que tienen objetos de clases diferentes de una jerarquía, a responder de forma distinta a una misma llamada de un método.

La idea es que los objetos de distintas clases – derivadas de otra – puedan ser tratados de la misma manera. Esto es, se le apliquen los mismos mensajes aunque las implementaciones particulares de los métodos sean diferentes. Estos métodos tienen la misma semántica y distinta implementación.

Para esto se deberá trabajar con vinculación tardía o vinculación en tiempo de ejecución. Esto se aborda en la unidad 3.

El polimorfismo fomenta la extensibilidad, software escrito para invocar comportamiento polimórfico se escribe en forma independiente del tipo de los objetos a los cuales los mensajes son enviados. Por lo tanto, nuevos tipos de objetos, que pudieran responder a mensajes existentes, pueden ser agregados en dicho sistema sin modificar el sistema base.

Encapsulamiento

Término formal que describe al conjunto de métodos y de datos de un objeto de manera tal que el acceso a los datos se permite solamente a través de los métodos propios del objeto.

La comunicación entre los distintos objetos se realiza solamente a través de mensajes explícitos.

Abstracción

La orientación a objetos fomenta que los programadores y usuarios piensen en las aplicaciones en términos abstractos.

A partir de un conjunto de objetos, se piensa en los comportamientos comunes de los mismos para situarlos en superclases, las cuales constituyen un depósito para elementos comunes y reutilizables.

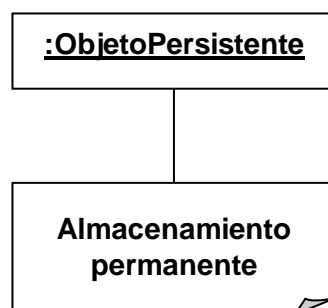
Persistencia

La persistencia se refiere a la permanencia de un objeto, es decir, al tiempo en que se le asigna espacio y permanece accesible. Cuando un objeto ya no es necesario, es destruido, y el espacio que este ocupaba es recuperado. La recuperación automática del espacio de memoria se denomina recolección de basura.

Algunos autores estudian la posibilidad de que haya una continuidad de la existencia de un objeto luego de la ejecución de un programa, por ejemplo almacenándolo en una base de datos. Si bien hoy existen Bases de Datos Orientadas a Objetos, en la práctica estas bases de datos se basan en otros modelos como el relacional.

La persistencia de los objetos²⁰

La persistencia designa la capacidad de un objeto de trascender el tiempo o el espacio. Un objeto persistente conserva su estado en un sistema de almacenamiento permanente, de modo que es posible parar el proceso que lo ha creado sin perder la información representada por el objeto (pasivación del objeto). En adelante, el objeto puede ser reconstruido (activación del objeto) por otro proceso y se comportará exactamente como en el proceso inicial. Los objetos no persistentes se llaman transitorios o efímeros. De modo predeterminado, los objetos no se consideran como persistentes.



En conjunto, los lenguajes orientados a objetos no proponen soporte directo para asegurar la persistencia de los objetos. Es una lástima, y ello obliga a recurrir a artificios externos para asegurar la persistencia de los objetos. Los fabricantes de bases de datos proporcionan soluciones para guardar objetos, bien totalmente orientadas a objetos, o bien híbridas.

Ventajas del Diseño Orientado a Objetos

Un lenguaje de programación que soporta el paradigma de orientación a objetos beneficia al desarrollador de software proporcionando una forma natural de modelar un fenómeno

²⁰ Extraído del libro Método de Objetos con UML

del complejo mundo real. Programar no significa solamente escribir líneas de código, sino desarrollar modelos utilizando clases.

Con la programación orientada a objetos los programas tienen menos líneas de código.

Las bibliotecas de clases predefinidas que contienen los lenguajes maduros como el Smalltalk aumentan las ventajas de utilizar lenguajes orientados a objetos, ya que la tarea de programar consiste en gran parte en encontrar dentro de las bibliotecas las clases y métodos adecuados y combinarlos convenientemente.

Otra ventaja de la programación orientada a objetos la representa la herencia, ya que subclases pueden heredar o redefinir estructuras de datos y métodos de clases existentes. Los objetos posibilitan integrar los datos y los métodos que actúan sobre dichos datos, lo que simplifica el mantenimiento del programa y sus posibles actualizaciones. Por el contrario en los lenguajes tradicionales siempre existe la posibilidad de que una actualización de una subrutina provoque efectos no deseados en subrutinas relacionadas.

Un problema potencial de los lenguajes orientados a objetos es que el programador debe conocer una extensa biblioteca de clases, pero este tiempo está compensado con el tiempo ahorrado en la reutilización de código, en lugar de tener que volverlo a crear.

Booch define la diferencia entre programar en forma tradicional (programación procedimental) y programar con objetos (programación orientada a objetos), de la siguiente manera:

“Lea las especificaciones del software que desea desarrollar. Subraye los verbos si desea código procedimental o los nombres (sustantivos), si desea código orientado a objetos.”

Conclusiones

La programación orientada a objetos es producto de un proceso evolutivo permanente de los lenguajes de programación, esto significa que no se descartan viejos conceptos sino que se van incorporando algunos nuevos. Conceptos como los de **abstracción** y **encapsulamiento**, pilares de la POO, ya fueron incorporados por el **Diseño Modular**, mientras que el concepto de **herencia** surge en esta nueva metodología.

En este nuevo paradigma de orientación a objetos, los **objetos** y las **clases** son los pilares y los **métodos**, los **mensajes** y la **herencia** son mecanismos primarios. Hasta ahora la creación de un programa consistía en la definición de procesos que actuaban sobre un conjunto independiente de datos.

En la POO los **objetos** son módulos autocontenidos que hacen referencia tanto a los datos como a los procedimientos que actúan sobre dichos datos. Los procedimientos de un objeto se llaman ahora métodos y representan el comportamiento del mismo. Las características comunes de un conjunto de objetos que tienen un mismo comportamiento se agrupan en una clase, a partir de ella pueden crearse nuevas clases (subclases) que heredan los métodos y los atributos de la clase superior, permitiéndole al programador programar solamente las diferencias.