Introducción

Hemos estructurado esta sección para ayudarte a aprender de la forma más fácil y rápida el proceso que conlleva adquirir las nociones del paradigma orientado a objetos, el cual dentro de la cursada se centrará en el modelado y programación de las propiedades de este paradigma. A lo largo de estas páginas aprenderemos la definición y la importancia del modelado de los objetos de nuestro juego mediante los diagramas de clases y los diagramas de secuencia. Finalmente explicaremos la metodología de resolución de problemas mediante la programación orientada a objetos

Paradigmas

Nuestro foco es la programación del producto (en este caso un videojuego) aplicando el paradigma de programación orientado a objetos, el cual es el más utilizado actualmente por las empresas que desarrollan software.

Esto de por si debiera ser un aspecto poderosamente motivante por cuanto significa que la mayor parte de los desarrollos que realices como Técnico Universitario en Diseño Integral de Videojuegos serán orientados a objetos.

Así que debemos iniciar este camino de profesionalización refiriéndonos al primer término que nos compete: **paradigma**.

El estadounidense Thomas Kuhn, un experto en Filosofía, una figura destacada del mundo de las ciencias y encargado de renovar la definición teórica de este término desde una concepción científica más acorde a los tiempos actuales nos brinda una respuesta contundente en su propia definición:

> *La serie de **prácticas** que trazan los lineamientos de una disciplina científica a lo largo de un cierto **lapso temporal**. El éxito de un paradigma es consecuencia de su efectividad para resolver algún problema.*

Si analizamos la primera oración suponemos que no debería causarnos conflictos de ningún tipo a excepción de dos palabras: prácticas y lapso temporal. ¿A qué se refiere Thomas con estas palabras?

Respecto de la primera, los propios científicos se han encargado de refinar o aclarar a que se refiere Kuhn; de esta manera hoy se acepta que hacen referencia a:

- Las leyes establecidas y los supuestos teóricos. Por ejemplo, las leyes de movimiento de Newton forman parte del paradigma newtoniano y las ecuaciones de Maxwell forman parte del paradigma que constituye la teoría electromagnética clásica.
- El instrumental y las técnicas instrumentales necesarias para hacer que las leyes del paradigma se refieran al mundo real. La aplicación en astronomía del paradigma newtoniano requiere el uso de diversos telescopios, junto con técnicas para su utilización y diversas técnicas para corregir los datos recopilados.
- Los principios generales propios del paradigma que guían el trabajo dentro del paradigma.

Lo anterior es esquematizado con el siguiente mapa conceptual

Ahora centrémonos en la otra palabra: lapso temporal. Para entender lo que Khun desea expresar con estos términos debemos analizarla en conjunto con la totalidad de la segunda oración. Esto significa que un paradigma es el resultado de un proceso social en el cual un grupo de personas desarrolla nuevas ideas y crea principios y prácticas alrededor de estas ideas. Entonces un paradigma no es solamente un conjunto de prácticas y conocimientos objetivamente validado. Como consecuencia cuando un paradigma ya no puede aplicarse para explicar un aspecto del mundo en base a las teorías y suposiciones científicas que modifican esa percepción significa que ese paradigma es obsoleto o no aplicable a esa situación.

Por los motivos expresados en los párrafos anteriores, se puede entender entonces por qué cada vez que surgen nuevos paradigmas es posible que se apliquen al mismo concepto: como consecuencia de un proceso natural de evolución o maduración del propio concepto.

¿No se entendió? Más fácil mediante este ejemplo: si quieres programar un videojuego debes realizarlo siguiendo un paradigma. Para ese tipo de problema ¿Hay un solo paradigma aplicable? Probablemente no. ¿Puede ser que un paradigma de programación no se pueda aplicar a un tipo de problema? Probablemente sí. ¿Puede ser que haya un paradigma más adecuado que otros para un tipo de problema? Seguramente que sí.

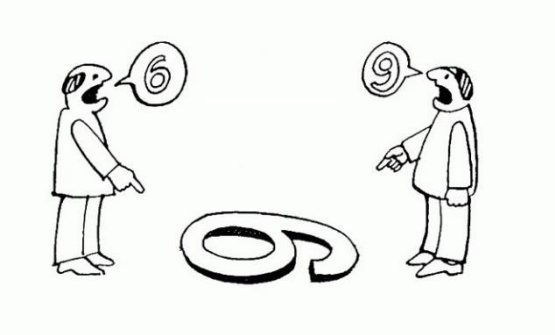
De esta manera podemos decir que se considera paradigma a:

- Los marcos de referencia que imponen reglas sobre cómo se deben hacer las cosas, indican qué es válido dentro del paradigma y qué está fuera de sus límites. Un paradigma

distinto implica nuevas reglas, elementos, límites y maneras de pensar, o sea implica un cambio.

- Patrones de pensamiento para la resolución de problemas. Un modelo o esquema fundamental que organiza nuestras opiniones con respecto a algún tema en particular. Los paradigmas establecen límites adoptados por los miembros de una comunidad científica para resolver problemas sustentados por los principios, leyes, supuestos teóricos y técnicas que la conforman.

Así, un paradigma científico establece aquello que debe ser observado; la clase de interrogantes que deben desarrollarse para obtener respuestas en torno al propósito que se persigue; que estructura deben poseer dichos interrogantes y marca pautas que indican el camino de interpretación para los resultados obtenidos de una investigación de carácter científico.



Observe la imagen de la derecha. Ante un problema diferentes paradigmas te ponen frente al mismo de manera diferente. Y no por ello la percepción de alguna de ellas esté equivocada. Pero lo que sí es probable es que alguno de esos paradigmas sea más efectivo que el resto en situaciones específicas.

Cuando un paradigma ya no puede satisfacer los requerimientos de una ciencia (por ejemplo, ante nuevos hallazgos que invalidan conocimientos previos), es reemplazado por otro.

En las ciencias sociales, el paradigma se encuentra relacionado al concepto de cosmovisión. El concepto se emplea para mencionar a todas aquellas experiencias, creencias, vivencias y valores que repercuten y condicionan el modo en que una persona ve la realidad y actúa en función de ello. Esto quiere decir que un **paradigma es también la forma en que se entiende el mundo**.

Paradigmas de Programación

Existen diferentes concepciones para los paradigmas de programación:

- Un proceso de diseño que va más allá de una gramática, reglas semánticas y algoritmos. Es un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas. Representan un enfoque particular o filosofía para la construcción del software.
- Son propuestas tecnológicas adoptadas por la comunidad de desarrolladores que se enfocan a resolver uno o varios problemas definidos y delimitados.

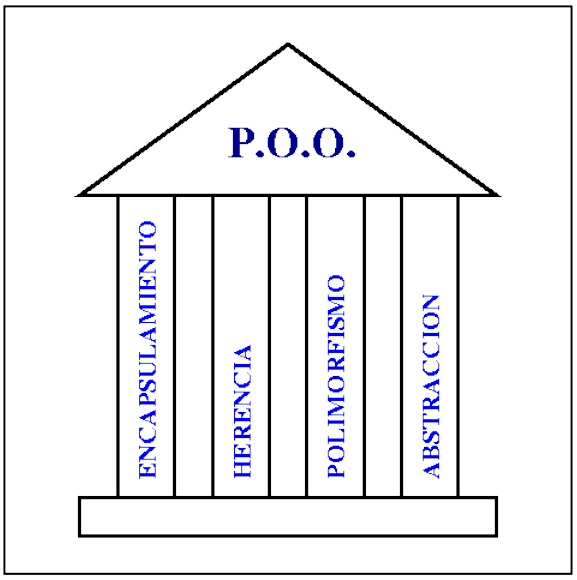
- Es un modelo básico de diseño y desarrollo de programas, que permite producir programas con unas directrices específicas, tales como: estructura modular, fuerte cohesión, alta rentabilidad, etc.
- Es una colección de modelos conceptuales que en conjunto modelan el proceso de diseño y determinan la estructura de un programa. Esa estructura conceptual de modelos está pensada de forma que los modelos determinan la forma correcta de los programas y controlan el modo en que el desarrollador piensa y formula soluciones, que luego son implementadas en un lenguaje de programación.
- Provee y determina la visión y métodos de un programador en la construcción de un programa o subprograma. Diferentes paradigmas resultan en diferentes estilos de programación y en diferentes formas de pensar la solución de problemas (con la solución de múltiples "problemas" se construye una aplicación o producto de software).

Existen tres cuestiones para tener en cuenta respecto de los paradigmas de programación:

1. Ningún paradigma es mejor que otro, sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.
2. Para que las características esenciales del paradigma sean efectivamente aplicadas, las características del lenguaje de programación utilizado para implementar la aplicación deben reflejar adecuadamente los modelos conceptuales de ese paradigma. Cuando un lenguaje refleja bien un paradigma particular, se dice que soporta el paradigma, y en la práctica un lenguaje que soporta correctamente un paradigma, es difícil distinguirlo del propio paradigma.
3. Los lenguajes de programación están basados en uno o más paradigmas, por ejemplo: Processing está basado en el paradigma orientado a objetos. El lenguaje de programación Scheme, en cambio, soporta solo programación funcional. Otros lenguajes, como C++ y Python soportan múltiples paradigmas.

Las nociones fundamentales del Paradigma Orientado a Objetos

Este paradigma se basa en la idea de que es posible representar un problema y su solución de una forma similar a la que el cerebro humano procesa la información que le rodea. En términos generales el paradigma representa la información del problema y su solución en la forma de entidades denominadas objetos. Estos objetos poseen un conjunto de características almacenadas y gestionadas por variables y un conjunto de operaciones u acciones que contienen la secuencia de pasos para que la operación se realice (algoritmo). Cuando un objeto necesita trabajar sobre una característica de otro objeto, o cuando requiere que otro objeto realice una operación debe enviarle un mensaje. Los mensajes son entonces, el mecanismo que utiliza el paradigma para establecer relaciones entre los objetos. Esta solicitud tiene un conjunto



de reglas y normas a seguir al momento de utilizarlas en un lenguaje de programación: el protocolo de mensajes. Finalmente, como todo paradigma, su filosofía se estructura sobre reglas, supuestos, técnicas y teorías que están organizadas en lo que se denominan Pilares o Propiedades del Paradigma Orientado a Objetos: la abstracción, la encapsulación, la herencia y el polimorfismo.

Estas propiedades estructuran por un lado la creación, uso y destrucción de los objetos alrededor del concepto de clase; mientras que por otro lado establece que el protocolo de mensajes es un contrato entre las clases que indica cómo una clase construye sus operaciones y cómo las debe poner a disposición de otras clases. De esta manera se introducen los conceptos de interfaz e implementación.

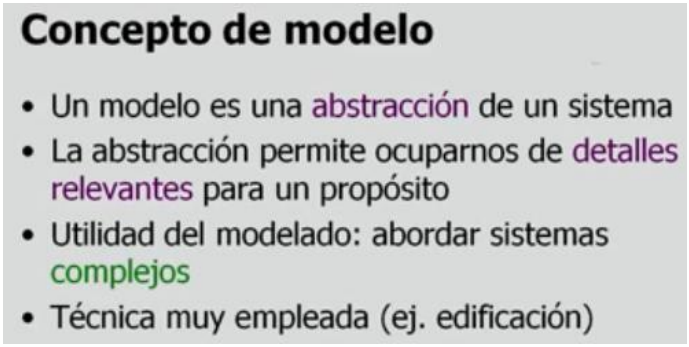
Más aún, estas propiedades definen formalmente que es un objeto, como se modelan los mismos y los mecanismos que permiten optimizar la reusabilidad, la facilidad de mantenimiento y realizar poderosas técnicas de programación.

Paradigma Orientado a Objetos: La abstracción y su relación con el modelado

Primero realizaremos una breve introducción al modelado del software y la importancia que este representa a la hora de desarrollar y construir aplicaciones informáticas. Posteriormente veremos como el paradigma orientado a objetos facilita modelar la realidad gracias a uno de sus pilares: la abstracción. Finalmente veremos algunos diagramas del Lenguaje de Modelado Unificado (UML) que permiten modelar eficientemente este paradigma. Con esto buscamos modelar los objetos de nuestro juego mediante los diagramas de clases y los diagramas de secuencia.

Un modelo es una abstracción de una realidad. En nuestro caso será la abstracción de un problema, luego la abstracción de la solución al problema, y finalmente la abstracción del sistema informático creado. La abstracción es una herramienta muy útil que consiste centrarnos únicamente en determinados detalles que son relevantes de nuestro sistema informático y desestimar otros detalles que pueden ser importantes pero que se pueden abordar en otro momento o que para los fines de nuestro sistema no serán abordados. La utilidad del modelado es fundamental a la hora de abordar la construcción de sistemas complejos como son la construcción de sistemas software.

La efectividad de los modelos no se discute, debido a que son utilizadas en muchas disciplinas. El ejemplo típico es la ingeniería civil o la arquitectura. El arquitecto utiliza diversos planos que son abstracciones de una parte del edificio construido o a construir. Dado que los profesionales de la construcción se han formado en la interpretación de estos planos, todos ellos interpretarán exactamente lo mismo (al menos que incurran en un error humano). Esto significa que los modelos son de interpretación unívoca, es decir no dan a lugar a ambigüedades o doble interpretación. Sin lugar a duda los profesionales de la construcción del software no deben desestimar el modelado como un aspecto central dentro del desarrollo de software.

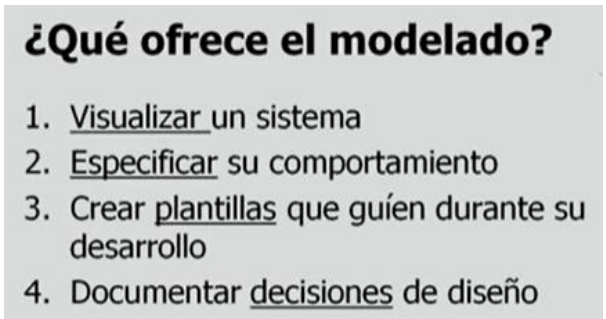


Sin embargo, es común una cierta resistencia en su uso ¿A qué se deberá? El factor principal dentro del mundo de los videojuegos radica en que el tiempo que se debe dedicar al modelado será mayor, por los ajustes que se seguramente se llegarán a realizar al avanzar el desarrollo del producto. La experiencia indica que lo importante es elegir el modelo adecuado al tipo de proyecto que se está por desarrollar; ya que existen modelos muy dinámicos. En particular el paradigma orientado a objetos se ajusta a modelos dinámicos (respecto de la rapidez de modificación) como veremos más adelante.

Entonces ¿Qué ofrece el modelado?

Nos ofrece básicamente 4 elementos esenciales:

1) Una visualización de un sistema: que no ha sido construido y de la que se puede obtener una aproximación a cómo será el mismo. (una aproximación de los objetos de nuestro juego y como ellos interactúan)
2) Una especificación de su comportamiento: nos permite indicar detalladamente como actúa el mismo ante diferentes situaciones.
3) Una plantilla que guíe a los desarrolladores durante su construcción: es decir son los "planos" que el arquitecto del software sigue para construir el producto.
4) Documentar decisiones de diseño: siempre las decisiones de diseño deben estar bien documentadas y justificadas. En algunas metodologías la documentación se dejaba al último; o como sucede de forma muy habitual está la tentación de no documentar por el tiempo que incurre. Los modelos permiten minimizar el tiempo de documentación.



Lenguajes de Modelado

Un lenguaje de modelado es una colección de técnicas de modelado. Una **técnica de modelado** posibilita escribir un modelo ya que consta de una colección de símbolos y un conjunto de reglas que definen cómo los símbolos pueden ser combinados de forma válida para formar un modelo:

Técnica de Modelado = colección de símbolos + reglas de composición

Dentro del ámbito del desarrollo de software es posible utilizar diferentes lenguajes de modelado, algunas opciones pueden ser:

- ✓ Código fuente: resulta una elección desaconsejable, puesto que se torna complejo de manejar. Asume que el desarrollador debe conocer el lenguaje de programación utilizado. Por otro lado, es difícil procesar y apreciar el comportamiento debido a la gran cantidad de líneas de código que presumiblemente posea el programa ¿Estoy refiriéndome a que no deba documentar el código fuente? Absolutamente no. Me estoy refiriendo a que no es aconsejable usar el código fuente como modelo de nuestra aplicación. El código siempre debe estar completamente documentado, pero eso no significa que el mismo deba usarse como modelo del sistema. Una opción más conveniente es usar otro lenguaje de modelado y de esa manera poder verificar que el código fuente es la implementación del modelo.
- ✓ Lenguaje Natural: Se refiere a crear diversos documentos en el cual se describe el comportamiento modelado. La característica principal de este tipo de lenguaje es que es propenso a errores, puesto que se generan documentos muy extensos; como consecuencia será difícil asegurar la consistencia y que no haya ambigüedad en nuestro modelado.
- ✓ Lenguaje Visual: es el que mayor éxito a tenido y el que se emplea (o se debería emplear) en la práctica por las empresas al momento de modelar software. Mediante anotaciones gráficas se construyen los modelos. Estas anotaciones tienen un significado concreto y unívoco que no da lugar a ambigüedad, y es una representación atractiva y rápida para observar el comportamiento del software debido que son fáciles de interpretar y procesar.

Con lo expuesto hasta ahora, resulta evidente que de tener que elegir un lenguaje de modelado a utilizar, la opción sería decantarse por los lenguajes de modelado visuales.

Teniendo esta primera decisión de diseño adoptada, la siguiente será determinar cuáles técnicas de modelado utilizar. Estas van desde aquellas que son puramente textuales hasta las puramente formales.

- Las técnicas más textuales tienen como principales ventajas que son muy fáciles de entender. Sin embargo, una desventaja es que tienden a ser imprecisos, ambiguos y no ofrecen verificación inteligente y control de calidad.
- Las técnicas puramente formales como, por ejemplo, el álgebra de procesos, ofrecen muchas posibilidades de verificaciones formales, pero como consecuencia son más difíciles de entender y utilizar.

En la asignatura usaremos el Lenguaje de Modelado Unificado (UML) porque ofrece lo que se denominan diferentes diagramas (un modelo gráfico). Sus diagramas se ubican en forma intermedia a las textuales y las formales. Así, por ejemplo, los diagramas de clases se ubican aproximadamente en el medio de estas dos clasificaciones. Tienen la ventaja de ser razonablemente fácil de entender, mientras que, al mismo tiempo, es lo suficientemente formal para lograr un buen nivel de precisión y posibilidades de verificación.

La propiedad de los objetos que ayuda a centrarse en el dominio del problema: la abstracción

Como se ha mencionado anteriormente, el paradigma orientado a objetos se sustenta en 4 pilares. El pilar denominado abstracción es la base para todo el desarrollo orientado a objetos. Consiste en la formalización de los conceptos necesarios que permiten generar soluciones

orientadas a objetos. La abstracción consiste en aislar elementos de su contexto o del resto de los elementos que lo acompañan para facilitar su estudio capturando las características esenciales del mismo.

En el ámbito de la programación orientada a objetos la abstracción permite representar un problema y/o su solución en términos de un modelo conformado por clases, objetos y la interrelación entre ellos.

El modelo orientado a objetos permite construir una representación para analizar, describir, explicar simular o predecir un fenómeno y se sustenta alrededor de las siguientes definiciones:

- **Clase**: modelo, molde, plano o maqueta a partir del cual se pueden generar objetos. Las clases son entidades utilizadas para analizar el problema y diseñar la solución. Toda clase posee 3 elementos importantes: el nombre de la clase, los atributos y las operaciones. Mediante las clases se pueden determinar los actores (clases) que participan en el problema estudiado (o en la solución) así como las características y acciones que estas entidades poseen y que contribuyen de alguna manera para que se cumplan los objetivos abordados por la solución planteada (los requisitos). Las clases son programadas en un lenguaje de programación orientado a objetos.
- **Objeto**: Los programas se ejecutan en memoria, adoptando el nombre de procesos. En el caso de los programas orientados a objetos estos procesos se denominan objetos que adquieren los atributos y operaciones de la clase a partir de la cual se ha creado, de esta manera en memoria pueden existir muchos objetos generados a partir de la misma clase pero cada una con una propiedad identidad que la separa de las demás (esto puede asemejarse al hecho de que a partir de un mismo plano se pueden construir varias viviendas, todas iguales por las especificaciones del plano pero en definitiva cada una es un ente individual)
- **Relaciones**: Indica la forma en que los objetos colaboran entre ellos para realizar alguna tarea específica. Esto genera un nuevo concepto que es el **mensaje**: es el mecanismo por el cual un objeto en memoria solicita a otro que ejecute una operación. Entonces las relaciones indican la forma en que los objetos envían mensajes a otros objetos.
- **Interfaz**: Las operaciones de un objeto que pueden ser solicitadas por otro objeto se denominan **servicios**. Para que los servicios de un objeto puedan ser invocados por otro objeto; el objeto que desea ponerlos a disposición de los otros objetos debe definir una interfaz. La interfaz es simplemente un mecanismo por el cual se determina si una operación se halla disponible para ser invocada por otro objeto (las operaciones dentro del objeto siempre son servicios para las otras operaciones del mismo objeto, esto es, las operaciones de un objeto siempre pueden ser invocadas por las otras operaciones del mismo objeto). El concepto de interfaz también se aplica a los atributos, esto es; si no se especifica una interfaz para un atributo, el mismo no podrá estar disponible para otros objetos.

Desde un aspecto conceptual, hemos descripto en que consiste la propiedad abstracción del paradigma orientado a objetos. Resulta natural para los desarrolladores que recién se inician en

este paradigma que haya elementos que no se comprendan. Por este motivo ahora las iremos detallando con mayor profundidad

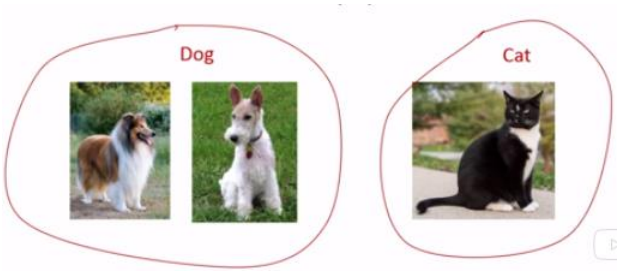
La clase

La noción de clase se basa en el principio de abstracción, el cual consta de tres premisas:

- Distinguir diferentes objetos
- Clasificar objetos entre objetos
- Focalizarse en las características y operaciones esenciales de esos objetos

El principio de abstracción es algo muy natural que aplicamos todos los días. Ya de niños, aprendemos cómo distinguir entre los no vivos y los seres vivos, entre cubiertos y juguetes, entre personas y animales (**Distinguir entre diferentes objetos**). Todos los días, **clasificamos los objetos que nos rodean en conceptos** como 'tenedor', 'cuchara', 'bloque', 'rueda', 'bolígrafo', 'bola'. Cuando categorizamos objetos, somos capaces de hacer abstracción de propiedades individuales, y **focalizar nuestra atención en las propiedades comunes esenciales**.

Por ejemplo, observe la siguiente imagen:



El perro Collie se ve muy diferente al perro Fox Terrier, sin embargo, entendemos que ambos son perros. Y, aunque el animal que se ubica a la derecha también tiene una piel peluda, cuatro patas, una cola, orejas y puede correr, saltar, mirar, comer etc. (lo mismo que los perros indicados anteriormente) sabemos que es un gato y no un perro.

En términos informáticos, un concepto que agrupa objetos con características y operaciones similares se denomina 'clase'.

Del mismo modo, para poder manejar los objetos en el universo de una organización (que presumiblemente debe representarse en un sistema de información), esos objetos deben ser categorizados en conceptos. Así que veamos un ejemplo:

En nuestra universidad hay mucha gente alrededor y podemos distinguirlos entre personal no docente, docentes y estudiantes:

- La clase “Estudiante” capturará los conceptos asociados a todos los estudiantes: se definirán las características y las operaciones (acciones que puede realizar) de un estudiante que son relevantes para la universidad. Además, en el mismo proceso abstraerá (no considerará) las características y operaciones que sean irrelevantes.
- Más importante aún, al definir la clase “Estudiante”, definiremos lo que distingue a un estudiante de cualquier otra persona dentro o fuera de la universidad: a saber, un estudiante está registrado en una carrera en la universidad. Las personas que no estén registradas en una carrera no pertenecerán a la clase “Estudiante”.
- Una universidad ofrece carreras. El concepto de carrera se refiere a una colección de materias, practicas, etc que puede seguir el estudiante para obtener un título. Al definir la clase “Carrera” definiremos la noción de “Programa de Estudio” de tal manera que entendemos la diferencia entre un programa de estudio y una colección aleatoria de cursos.

Si se desea construir un modelo para clases y otro modelo para objetos; podemos hacer la siguiente distinción:

- El nivel cero es el nivel de instancia o de objeto. Allí encontramos objetos individuales. Para la clase **Estudiante**, hace referencia a estudiantes reales, por ejemplo, dado que Juan Perez, David Vega o Ramiro Olarte son estudiantes inscriptos en sus respectivas carreras; se dirá que son instancias de clases (pertenecen al nivel cero de abstracción). Para la clase **Programa de Estudio**, encontramos diferentes programas de estudio como, por ejemplo, la Licenciatura en Ciencias Geológicas o Analista Programador Universitario son claros representantes de objetos.
- El nivel superior de abstracción, el nivel 1, es el nivel de modelo o diagrama. Ahí encontramos la definición de las clases, tales como la clase **Estudiante** o la clase **Programa de Estudio**.

Esto resulta particularmente importante, porque ud como TUDIVJ, lo que programará son modelos de nivel superior de abstracción, es decir las clases.

Además, modelar es abstraer, esto significa, realizar la transición del nivel 0 al nivel 1: Se pasa de las instancias a la definición de los conceptos.

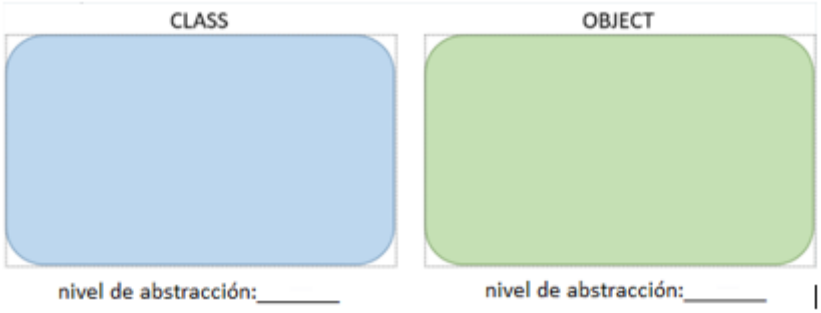
Por otro lado, cuando se desea **validar el modelo**, se debe hacer lo contrario: se tomará el modelo de nivel 1 y mediante un ejemplo se lo validará o refutará (es decir se realizará la prueba y se analizará su resultado). Entonces se comienza desde el nivel 1 y se intentará ilustrar ese nivel 1 encontrando un ejemplo en el nivel 0 que pruebe o refute la validez de ese modelo.

Ahora, lo invito a que realice la siguiente actividad para que pueda confirmar si los conceptos, en conjunto con los ejemplos le han permitido asimilar los conocimientos asociados a las clases y los objetos:

> *Los profesores pueden enseñar varias asignaturas. Una asignatura, a su vez, puede ser impartida por varios profesores. Por ejemplo, Fundamentos de Programación Orientada a Objetos es impartida por Ariel Vega y Carolina Apaza, mientras que Introducción al Desarrollo de Videojuegos es impartida solo por Mario Tejerina y Emanuel Jara. Cuando hay demasiados estudiantes registrados en una asignatura, se*

puede distribuir el examen en varias salas diferentes. Por ejemplo, el cuestionario de opción múltiple de Fundamentos de Programación Orientada a Objetos lo toman 385 estudiantes en San Salvador distribuidos en la sala A, sala B y sala C (son una bocha), mientras que en San Pedro se toma en una única sala de forma oral, lo cual supone además que hay diferentes modalidades de evaluación para la materia. Juan es uno de los estudiantes que rindió con el Profesor Vega y obtuvo una nota sobresaliente.

Identifique los sustantivos del enunciado anterior y arrastre los diferentes elementos a las zonas apropiadas y luego una las clases con sus objetos. Además, indique a qué nivel de abstracción corresponden:

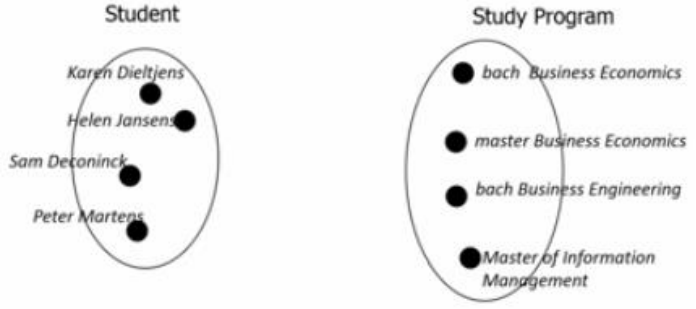


Una técnica de modelado para clases: el diagrama de clases de UML

Ahora debemos adoptar una técnica para modelar las clases. Como se anticipó en secciones anteriores, se adoptará el diagrama de clases de UML. Los diagramas de clases constituyen la vista más común e importante del diseño realizados por los desarrolladores de software. Se lo concibe como un modelo estático, porque no describen acciones; sino que muestran las clases y sus relaciones. En este diagrama una clase se representa como un rectángulo con el nombre de la clase dentro de ella. Observe la siguiente imagen

Se pueden distinguir dos clases: la clase ESTUDIANTE y la clase PROGRAMA DE ESTUDIO.

Como se pudo observar anteriormente, cada clase representa un conjunto de objetos. La clase ESTUDIANTE representa un conjunto que contiene a todos los estudiantes individuales. Y del mismo modo, la clase PROGRAMA DE ESTUDIO representa el conjunto de programas de estudio que contiene todas las diferentes instancias de programas de estudio, Esto se puede observar en el siguiente esquema ejemplificativo

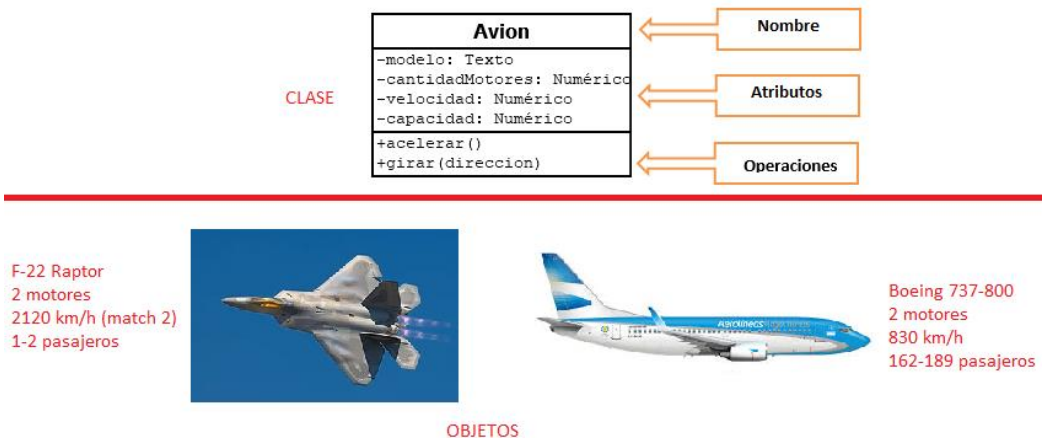


Una 'clase' tiene dos funciones:

1. Por un lado, la clase será la plantilla o un modelo para un grupo de objetos del mundo real que son similares. La clase define un tipo de instancias y, por lo tanto, también podemos denominarla Tipo de Objeto. Ej: Helen Jansen es un tipo de objeto de la clase Estudiante.
   Esto genera lo que se denomina "intent": definición de los miembros al que refiere la clase. En nuestro ejemplo, una persona es solo un estudiante si esa persona está suscrita al menos a un programa de estudios en la Universidad.
2. La plantilla capturará las características (y operaciones) relevantes sobre los objetos de la clase. En el caso de los estudiantes de la universidad, su nombre, su fecha de nacimiento, su domicilio y su dirección de correo electrónico son ejemplos de características que son relevantes para el concepto de estudiante en el contexto de la universidad. De la misma manera inscribirse a un final o solicitar constancia de regularidad son ejemplos de operaciones (acciones) relevantes que puede realizar el estudiante en el contexto de la universidad.
   Como se está abstrayendo, la definición de una clase omitirá los aspectos irrelevantes. Las características irrelevantes de los estudiantes desde la perspectiva de la universidad son: el color de su cabello, el color de sus ojos, su altura, su peso, etc. Las operaciones irrelevantes de los estudiantes desde la perspectiva de la universidad son: comer, vestirse, etc.
   Al mismo tiempo, la clase representa una colección de objetos que se ajustan a su “intent”. Esta se denomina 'extent' de la clase.

Finalmente tenga en cuenta que las clases pueden representar objetos tangibles e intangibles. Los estudiantes son por ejemplo objetos tangibles mientras que los programas de estudio son objetos intangibles. Ambos son tipos de objetos y se pueden representar como clases en el diagrama de clases UML.

Además del nombre, en una clase se pueden indicar atributos y operaciones. Los atributos son características relevantes de los objetos que se instancian a partir de una clase, mientras que las operaciones son las acciones que pueden realizar los objetos instanciados de una clase. Por ejemplo:



El diagrama de clases establece una notación para la definición de nombres, atributos y operaciones que no se puede violar.
Nomenclatura para Definición de nombres:

1. Deben ser sustantivos en singular
2. La primera letra del nombre inicia en mayúscula
3. No se admite espacios en los nombres. En el Diagrama de clases en fase de análisis se admite el uso de guión bajo. En el Diagrama de clases en fase de diseño es una práctica común que la primera letra de la siguiente palabra esté en mayúscula.

Nomenclatura para definición de atributos:

1. Inician con letra minúscula
2. No se admite espacios en los nombres de atributos. En el Diagrama de clases en fase de análisis se admite el uso de guión bajo. En el Diagrama de clases en fase de diseño es una práctica común que la primera letra de la siguiente palabra esté en mayúscula

Nomenclatura para la definición de operaciones:

1. Deben ser verbos en infinitivo
2. Inician con la primera letra en minúscula
3. No admiten espacios en los nombres. En el Diagrama de clases en fase de análisis se admite el uso de guión bajo. En el Diagrama de clases en fase de diseño es una práctica común que la primera letra de la siguiente palabra esté en mayúscula.

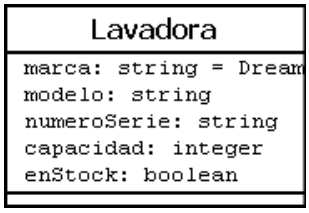
Los tipos de datos admitidos para los atributos y operaciones en UML (versión 3 para adelante)

1. Cadena o Texto (string)
2. Número de punto flotante (float)
3. Entero (integer)
4. Booleano (boolean)
5. Fecha (Date)
6. Hora (Time)

Puede indicar el tipo en español o en inglés. La industria suele indicarlos en inglés. También puede indicar un valor por defecto para el atributo con el signo igual (=).

A continuación, se planten diversos ejemplos de definición de clases, en base a problemas comunes con los que tendrá que lidiar como APU:

Ejemplo 1: Defina una clase Lavadora, e identifique sus atributos



Ejemplo 2: Sea Motor Bike un comercio dedicado a vender motocicletas, ATV (vehículos todo terreno), vehículos automotores para nieve y accesorios para todos estos tipos de vehículos. Identifique las clases y atributos

Observe que de la descripción surgen dos clases. En particular para Vehículo, partiendo de la oración del ejercicio, se puede detectar implícitamente un atributo que indica el tipo de vehículo que se vende. De la experiencia que se tenga en este tipo de problemas podemos deducir otros atributos o incluso operaciones.

Vehiculo
codigo: integer tipoVehiculo: string precio: float marca: string

Accesorio
codigo: integer nombre: string precio: float

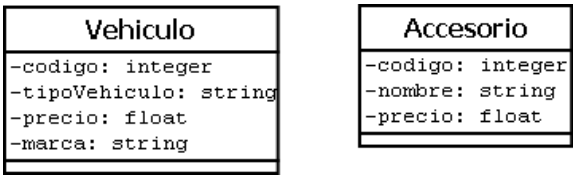
Ejemplo 3: Observe la siguiente imagen. Determine clases y objetos. Para las clases indique atributos.

La notación que ofrece UML para el diagrama de clases no se limita a la nomenclatura de los nombres de las clases, atributos y operaciones; sino que se extiende a otras áreas. Ahora estudiaremos la visibilidad de los atributos y las operaciones.

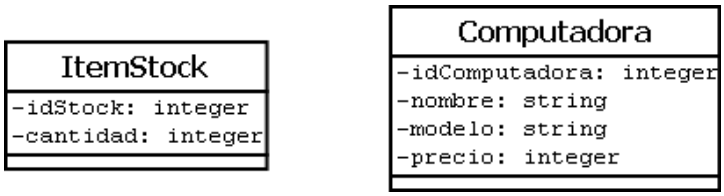
La visibilidad, establece la proporción en que otras clases podrán utilizar los atributos y operaciones de una clase dada (o en operaciones de una interfaz, concepto que se verá más adelante). En UML existen tres niveles de visibilidad:

1. Nivel Público: la funcionalidad se extiende a otras clases. En otras palabras, cualquier clase puede ver ese atributo u operación. Se representa con el signo (+).
2. Nivel Protegido: la funcionalidad se extiende únicamente a las subclases de la clase original (los conceptos de superclases y subclases se verán cuando se haga referencia a propiedad de los objetos denominada Herencia). Se representa con el signo (#)
3. Nivel Privado: En este nivel solamente la clase original puede acceder a los atributos y operaciones privadas. Se representa con el signo (-)

Ejemplo 4: Tome el ejemplo de Sea Motor Bike y represente la visibilidad los atributos de las clases como privadas.



Ejemplo 5: Idem para el ejemplo 3.



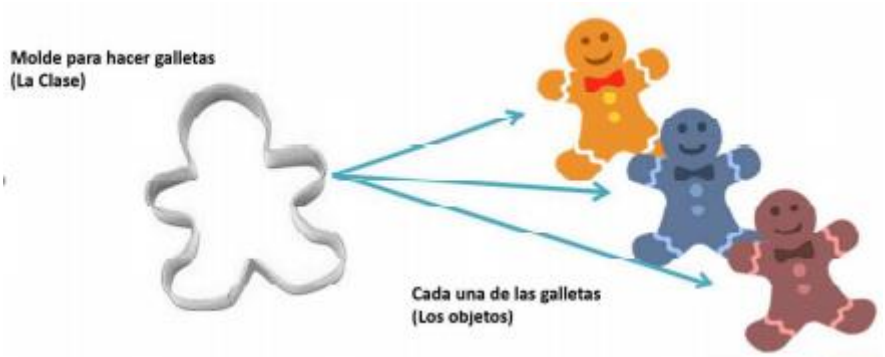
Hasta este momento se han realizado ejemplos de clases con sus respectivos atributos modelados mediante el diagrama de clases. ¿Y las operaciones? Hemos retrasado el tratamiento de este tema muy importante dentro del modelado de clases, a propósito, porque previamente debemos dar un marco conceptual completo alrededor del término **objeto**.

El Objeto

Mientras se describían los conceptos asociados al término “clase” se han nombrado y ejemplificado en diversos momentos la palabra objeto. Ambos conceptos están íntimamente relacionados, pero presentan diversas diferencias: mientras que la clase es una abstracción, el objeto es una entidad concreta que existe en el espacio y el tiempo. Se puede decir que la clase es la esencia del objeto, y es por ello por lo que se suele utilizar el verbo “**instanciar**” para indicar que un objeto se ha creado a partir de una clase.

Los objetos que comparten estructura (atributos) y operaciones similares, se agrupan en una clase. En el paradigma orientado a objetos, los objetos pertenecen siempre a una clase, de la que toman su estructura y operaciones.

Una forma común de expresar la relación entre clases y objetos es mediante ejemplos. Anteriormente esta relación se ejemplificó mediante las clases Estudiante, Programa de Estudio, Avion y Plano de la que se indicaron ejemplos de instancias de estas clases. Ahora se realizará lo mismo, pero haciendo énfasis en los objetos mediante un molde de galletas. Observe la siguiente imagen, de la que seguramente debería quedar claro lo que desea expresar



Entonces, ¿qué es un objeto? Desde el punto de vista del ser humano, puede ser cualquiera de estas tres representaciones:

1) Una cosa tangible y/o visible. Por ejemplo: auto, avión, lápiz, etc.
2) Un concepto comprensible intelectualmente, es decir intangible. Por ejemplo: cuenta bancaria, viaje en bote, etc.
3) Algo hacia lo que se dirige un pensamiento o una acción. Por ejemplo: listado de deudores, etc.

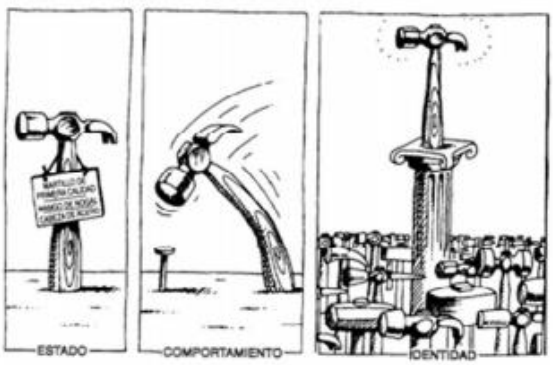
Una definición formal sería la siguiente:

Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un rol bien definido en el dominio del negocio.

La definición que usaremos en la asignatura es aquella ampliamente utilizada por los desarrolladores de software, que implícitamente toman todos los aspectos de la definición formal anterior

Objeto = Identidad + Estado + Comportamiento

La identidad es una propiedad interna de los objetos, por la cual todo objeto es único, irrepetible y diferente de otro objeto creado a partir de la misma clase. Por tanto, no pueden existir dos objetos iguales.



El estado representa el valor de todos los atributos de un objeto en un momento específico.

El comportamiento de un objeto hace referencia a las operaciones que posee objeto y que este realiza ya sea como una reacción al exterior, un cambio de estado interno o ante una solicitud de otro objeto. Dicho de otra forma, el comportamiento representa su actividad visible y comprobable exteriormente.

Una operación denota un servicio que un objeto de la clase a la que pertenece pone a disposición de otros objetos, una capacidad de acción que los objetos de esa clase poseen. Dentro del comportamiento de un objeto, se debe referenciar un concepto importante: el paso de mensajes. Representa la acción que realizar un objeto cuando requiere un servicio de otro. Si éste autoriza el pedido, entonces responde al mensaje ejecutando la operación solicitada.

El estado y el comportamiento de un objeto definen un conjunto de roles que representan al objeto en el mundo. Además, estos roles cumplen las responsabilidades de la abstracción.

En un diagrama de clases las operaciones se definen siguiendo la siguiente sintaxis

[visibilidad] nombre ([lista de parámetros])[:tipo de retorno]

Donde cada parámetro tiene su propia sintaxis

[dirección] nombre: tipo [multiplicidad][=valor]

Y donde los valores válidos para la dirección son:

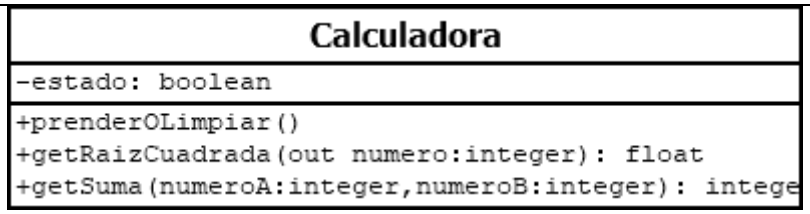
- ✓ in: la operación puede modificar el parámetro y quien invoca la operación no necesita volver a verlo.
- ✓ out: la operación cambia el parámetro y se lo devuelve a quien invoca la operación.
- ✓ inout: la operación utiliza el parámetro y puede cambiarlo para devolverlo.

En esta sintaxis los corchetes indican opcionalidad, por tanto, lo único requerido es el nombre de la operación.

Ejemplo 6: Mi lavarropa tiene la capacidad de centrifugar la ropa en varias velocidades, por defecto lo hace a 600 rpm. Modele esta descripción con el diagrama de clases.

Este es un ejemplo de un método de tipo servidor. Particularmente posee un parámetro cuya dirección es in. Este parámetro además por defecto usa el valor 600, pero eso no implica que no pueda enviarse por medio de él otro valor.

Ejemplo 7: Observe la siguiente imagen y determine al menos tres operaciones con diferentes cantidades de parámetros luego de definir la clase adecuada



El método *prenderOLimpiar()* limpia la pantalla si el estado es true, caso contrario prende la calculadora, por tanto es un método de tipo selector.

Observe que los otros métodos empiezan con *get*, que está en inglés y significa obtener. Es común usar esta notación cuando un método devuelve algún valor. Por otro lado, el parámetro tiene dirección **out** por cuanto lo altera devolviendo su raíz cuadrada. Esto significa que en memoria se usa el mismo espacio. Siendo más eficiente que generar una nueva variable para el resultado.

Ejemplo 7: Nuevamente visualice esta imagen

Para las clases determine sus operaciones.