



INTRODUCCIÓN

La abstracción es la propiedad más importante del paradigma orientado a objetos. Hace referencia a un **proceso** mediante el cual se determinan tanto **las características más relevantes**, así como los **comportamientos comunes más relevantes** de una porción de la **realidad**. Mediante este proceso es posible **definir las entidades** que conforman y representan el sistema estudiado.

Este proceso es clave a la hora de realizar el análisis y diseño orientado a objetos, porque permite **determinar un conjunto de CLASES que modelan la realidad** o el problema que se quiere solucionar.

En pocas palabras mediante la abstracción representamos el mundo mediante clases; que es en definitiva lo que se programará en un lenguaje de programación orientado a objetos.

LA CLASE

La clase es una entidad que describe un conjunto de objetos con estructura y comportamiento similares. Comúnmente se asocia el concepto de clase por analogía a un “molde” desde el cual se pueden obtener objetos, tal como se observa en la imagen de la derecha:

Así como los moldes para hornear permiten obtener galletas con una forma específica, a partir de las clases se pueden obtener los objetos del sistema informático.



El objetivo fundamental de usar clases responde al principio de abstracción del paradigma orientado a objetos, que consiste en tres premisas:

- Distinguir diferentes objetos
- Focalizarse en las características y operaciones esenciales de esos objetos
- Clasificar los objetos en base a sus características y operaciones comunes en clases

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 (Clasificar objetos).

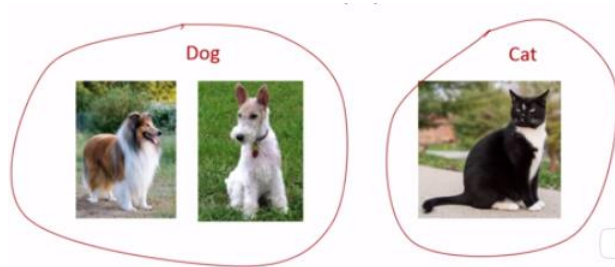
Todos los días, diferenciamos los objetos que nos rodean en conceptos como 'tenedor', 'cuchara', 'bloque', 'rueda', 'bolígrafo', 'bola' (Distinguir diferentes objetos).

Finalmente, cuando realizamos alguna actividad concreta somos capaces de hacer abstracción de propiedades individuales, y focalizar nuestra atención en las propiedades comunes esenciales.

Por ejemplo, si realizamos un deporte, por ejemplo, fútbol y estamos participando de un partido, nos focalizaremos en ciertos objetos: los compañeros más cercanos a quienes realizar un pase para lo cual observaremos si las personas cercanas tienen el mismo color de camiseta; estar atento a no perder el dominio de la pelota, y dependiendo de la situación dar un pase, patear al arco rival o decidir correr con la pelota un tramo más.

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. Como puede apreciar acaba de aplicar el principio de abstracción.



Veamos un par de ejemplos para que queden expuestos de manera explícita estos conceptos.

Ejemplo 1: Observe la siguiente imagen y determine las clases, atributos y operaciones que se pueden definir a partir de los objetos que en ella están presentes.



Como puede notar tenemos una imagen de frutas. Mediante la palabra “fruta” hemos podido agrupar un conjunto de objetos que comparten características similares. Entonces, la palabra **Fruta representaría la clase**. A **partir de esta clase podemos obtener diversos objetos**: la uva, la sandía, el kiwi, etc, los cuales son efectivamente frutas.

¿Cuáles serían las características similares de estos objetos?

Como puede observar cada fruta tiene un color o colores específicos, así como un nombre que la identifica. Podemos identificar que cada una de ellas posee una textura, tamaño

y sabor definidos. Estas serían **las características de la clase**; o lo que en términos del paradigma orientado a objetos conocemos **como atributos de los objetos**. Dado que una clase es un molde para obtener objetos es natural pensar que posee atributos, y que **los objetos que se creen a partir de esa clase poseerán esos atributos**.

Para hacerlo más explícito podemos representarlo así, donde se completan algunos valores, dejando al estudiante la tarea de completar los faltantes

Clase Fruta					
Atributos					
Objetos	Nombre	color/colores	tamaño	textura	sabor
	Banana	amarillo	mediano	fibrosa	dulce
	Uva	morado	pequeño	suave	dulce
	Sandia	Verde por fuera/rojo por dentro	grande	fibrosa	dulce
	Durazno				dulce
					Dulce
					Dulce
					dulce
					dulce

Algo interesante a considerar es que estos objetos no tienen comportamiento, ya que no realizan ninguna acción u operación. En el siguiente ejemplo además de identificar las clases y características, se identificarán sus comportamientos.

Ejemplo 2: Nuevamente observe la siguiente imagen y determine clases, características y comportamiento



En este caso podemos observar varias personas y elementos físicos. En primer lugar, se debe recordar que **la abstracción toma los elementos esenciales o relevantes**. La imagen muestra claramente que el centro de esta es un profesor enseñando a varios estudiantes. Justamente, poder identificar estos aspectos ayuda a identificar las clases relevantes, que **en este caso serían Profesor y Estudiante**; con lo cual las sillas y mesas se descartan.

Respecto del **Profesor** podemos ver **que tiene ciertas características**: un nombre, una altura, un tipo de vestimenta, un color de tez, etc. De todas estas características señaladas también debemos determinar las que serían relevantes (de ahora en adelante diremos abstraer), con lo cual podemos señalar que el nombre del profesor, así como la materia que imparte sería lo único necesario para poder interactuar con él.

¿Cuál sería el comportamiento relevante del Profesor?: explicar tema, escuchar preguntas, responder consultas. Como se ha señalado anteriormente, el **comportamiento representa las operaciones que puede realizar la clase** (y por ende, los objetos que se crean a partir de ella).

Respecto de los estudiantes ¿Cuáles serían los atributos y operaciones resultantes del proceso de abstracción?

LA ABSTRACCIÓN EN LA PROGRAMACIÓN DE VIDEOJUEGOS

Y estos aspectos ¿se necesitan en el mundo del desarrollo de videojuegos? Absolutamente sí. Programar juegos, significa programar clases de las cuales se generan los objetos del juego. Cada objeto de un videojuego se denomina GameObject. Se crean a partir de las clases que representan el sistema informático detrás del videojuego.

Ejemplo 3: En la siguiente pantalla de juego identifique las clases, con sus respectivos atributos y operaciones; a partir de las que se crearon los GameObjects.

Algo que debe considerar es que una pantalla de juego es una interfaz gráfica de usuario de un juego “ya programado”. Así que todos los elementos serán relevantes porque ya pasaron por el proceso de abstracción. Lo importante es poder identificar las clases de los objetos que vemos en la pantalla.





En este ejemplo podemos ver al GameObject que representa a Mario. **Probablemente exista una clase que se denomine PersonajePrincipal o Jugador.** Si únicamente nos basamos en lo que se puede observar en la pantalla, podemos indicar que **algunos de los atributos de Mario son cantidad de monedas recolectadas, el tipo de Mario** (Mini Mario, Super Mario o Fairy Mario), **y el puntaje de Mario.** Por otro lado, **sus operaciones serían: correr, saltar, golpear bloque.**

Otras clases presentes son Goomba, Bloque, Escenario, Tiempo. De estos indique atributos y operaciones.

Ejemplo 4: El GDD (Game Design Document) es una fuente muy importante de la información del juego para realizar la abstracción de las clases. Observe las siguientes extracciones de un GDD e indique las posibles clases, con sus atributos y operaciones.

GDD Atacantes del Espacio PadreGamer

Descripción del Proyecto (Project Description)

Atacantes del Espacio es un juego 2D "shoot' em up" (disparar y matar) basado en el famoso juego llamado "**Space Invaders**" lanzado en Japón en junio de 1978 y diseñado por Toshihiro Nishikado para la Empresa Japonesa Taito¹. "Space Invaders" es uno de los videojuegos más importantes de la historia considerándose uno de los primeros shooters creados y por esta razón PadreGamer quiere rendir un homenaje al mismo mediante la creación de tutoriales en el canal de YouTube: <https://www.youtube.com/c/padregamer> para enseñar a todos los interesados como crear un juego tanto en Unity 2017.3 como en GameMaker Studio 2 que se acerque lo más posible al concepto del juego original.



Personajes (Characters)

- **Cañón Laser:** controlado por el jugador representa el último punto de defensa de la invasión alienígena en la tierra y de ser destruido la tierra será invadida por los aliens.



- **Alienígenas:** representan a los enemigos, hay tres tipos de aliens organizados en 5 filas de 11 aliens cada una. La misión de los aliens es ir acercándose cada vez más rápido a la tierra para invadirla mientras disparan al jugador. El valor en puntos por destruir a cada alien es inferior en las filas de abajo y mayor en las de arriba.



- **Nave Misteriosa:** es un OVNI (objeto volador no identificado) el cual aparece aleatoriamente en la parte superior de la pantalla y si el jugador lo destruye proporcionara un bono aleatorio de puntos



Progresión de la Historia (Story Progression)

Aunque atacantes del espacio no tiene una historia, la progresión simplemente consiste en ir destruyendo ola tras ola de alienígenas las cuales se irán haciendo cada vez más difíciles.



Explicación del Juego (GamePlay)

Esta probablemente sea la sección más importante del GDD, aquí es donde se describe cómo se jugará el juego con todas sus mecánicas

Debido a que Atacantes del Espacio tratará de acercarse lo más fiel posible al concepto original del juego "Space Invaders" entonces se procede a continuación a explicar las mecánicas y conceptos generales de dicho juego:

Es un juego "shoot' em up" fijo (el movimiento del jugador está restringido) en 1 dimensión donde el jugador controla un cañón láser que sólo se mueve horizontalmente en la parte inferior de la pantalla y puede disparar a un grupo de aliens que van descendiendo de la parte superior. El objetivo del juego consiste en destruir 5 filas de 11 aliens cada una. Los aliens se mueven horizontalmente y al llegar al borde de la pantalla descienden un poco para entonces empezar a moverse en el sentido horizontal contrario repitiéndose así el ciclo.

Al destruir un alien, logrando acertar el disparo del cañón láser, el jugador va obteniendo puntos. A medida que más y más aliens son destruidos, el movimiento de estos, así como la música del juego va incrementando su velocidad. Destruir todos los aliens trae una nueva ola (nivel) de éstos con una mayor dificultad, lo cual representa un ciclo sin fin.

Los aliens intentan destruir el cañón laser disparándole a medida que van descendiendo. Si llegan a alcanzar la parte inferior de la pantalla, la invasión alienígena es exitosa y el juego se termina aun cuando el jugador tenga vidas disponibles. Una "nave misteriosa" ocasionalmente aparecerá en la parte superior de la pantalla moviéndose de un lado al otro y si es destruida proporcionará un bono de puntuación al jugador. El cañón laser está parcialmente protegido por unos bunkers de defensa estacionarios los cuales son gradualmente destruidos por los disparos tanto del jugador como de los aliens. El jugador tiene un número definido de cañones laser (vidas) por lo que el juego también se acaba al ser destruido el último de ellos.

Habilidades del Usuario (User Skills)

- Presionar un botón para disparar
- Utilizar las flechas izq y der o la palanca del control para mover al cañón
- Estimar el momento exacto del disparo para destruir el objetivo
- Administrar el uso de los bunkers de defensa
- Esquivar los proyectiles de los enemigos

Bien, para este caso, se ha seleccionado como clase a describir el Cañon Laser. Los subrayados rojos indican la fuente de los atributos detectados mientras que los azules permiten obtener las operaciones para este cañón. Así tenemos

- Clase: Cañon Laser
- Atributos: color, ubicación actual, cantidad de vidas, puntaje
- Operaciones: mover a la izquierda, mover a la derecha, disparar

Como puede observar resulta adecuado partir del GDD para obtener las clases del sistema. Más adelante se observará que este GDD también brinda información valiosa para determinar cómo interactúan las clases detectadas.

Determine las demás clases (con sus respectivos atributos y operaciones) que existen en el GDD.

EL MODELADO DE LA ABSTRACCIÓN

Hasta este momento se han brindado diferentes alternativas para señalar las clases obtenidas a partir de un proceso de abstracción. Sin embargo, existe un mecanismo estandarizado para representar las clases. El proceso de representar las clases de manera estandarizada se denomina modelado de clases. El estándar para el modelado de la abstracción de los objetos es UML y específicamente para esta sección utilizaremos el Diagrama de Clases.

FAMILIAS DE DIAGRAMAS

UML plantea un conjunto de diagramas que se estructuran en dos grandes familias: diagramas de estructuras y diagramas de comportamiento.

Los diagramas de estructuras hacen referencia a lo que es el soporte estructural, la parte arquitectónica de datos del sistema mientras que **los diagramas de comportamiento** se centran en distintas expresividades orientadas a especificar el comportamiento de un sistema.

DIAGRAMAS DE CLASE: ASPECTO GENERAL

Es el diagrama esencial en UML porque desde un punto de vista estructural es el más usado para especificar la estructura datos de un sistema de información. En relación con este aspecto, **este diagrama constituye la construcción principal de cualquier solución** en un ámbito de modelado orientado a objetos.

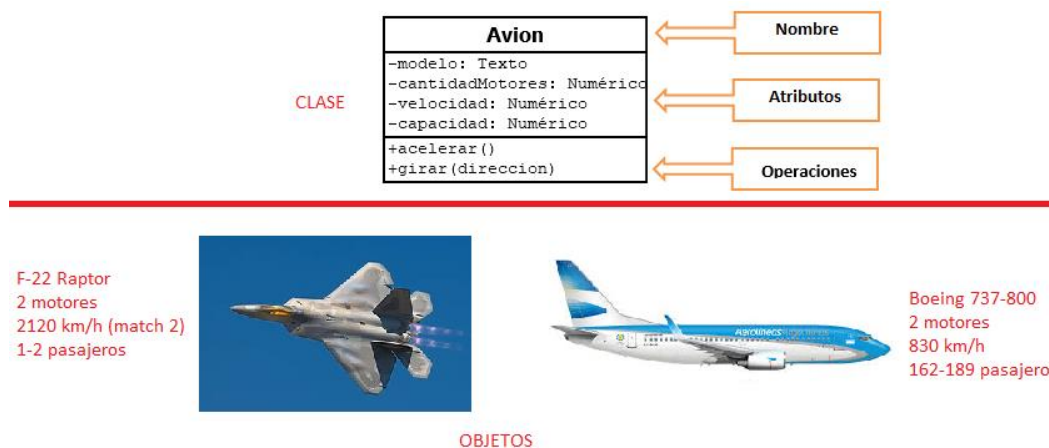
Por estos motivos, en él se incluye el conjunto de clases que van a conformar lo que es el esqueleto del sistema, **distribuyendo de manera adecuada** en ese conjunto de clases; **por medio de sus atributos; los datos que el sistema gestiona**.

Si las clases y sus atributos constituyen el esqueleto de un sistema; en analogía con el cuerpo humano, podemos señalar que este diagrama también permite definir y ubicar los músculos que permiten que ese esqueleto se pueda movilizar. Esto es, **en cada clase es posible explicitar un conjunto de operaciones, así como un conjunto de relaciones entre clases que determinan QUE HACE el sistema**.

Para que un diagrama de clases cumpla con lo expuesto en los párrafos precedentes, una clase se representa como una caja que puede estar dividida de 1 a 3 partes. La primera parte siempre hará referencia al nombre de la clase. Si la clase se divide en tres partes, la segunda sección estará reservada para los atributos, mientras que la tercera alojará el conjunto de operaciones. Sin embargo, si la clase solo posee dos secciones, la segunda parte puede hacer referencia a los atributos o a las operaciones.

Ejemplo 5: Diagrame una clase UML a partir de una clase Avion. Además, proponga dos objetos que se obtienen a partir de esa clase.

En este ejemplo puede ver de manera gráfica como se representa una clase. Se observa de manera explícita las tres partes de una clase. Si bien existe una nomenclatura para definir el nombre de una clase, así como reglas específicas para la definición de los atributos y operaciones; es muy probable que pueda interpretar la clase modelada sin mayores inconvenientes: La clase se denomina Avión, posee 4 atributos (modelo, cantidad de motores, velocidad y capacidad) y finalmente 2 operaciones (acelerar y girar en una dirección).



La nomenclatura del diagrama de clases

En este apartado vamos a definir la nomenclatura que se usará en la asignatura para definir los diferentes elementos de una clase. Cabe acotar que aquellos que se coloquen en cursiva hacen referencia a aspectos no obligatorios en UML pero que se consideran en el lenguaje de programación que usaremos (en este caso los derivados de JAVA aplicados a Processing), de tal manera que los mismos se apliquen desde el principio.

Nomenclatura para definición de nombres de clases

1. Deben ser sustantivos en singular.
2. La primera letra del nombre inicia en mayúscula, el resto continúa en minúsculas.
3. No se admite espacios en los nombres. En UML para durante la fase de análisis se admite el uso de guion bajo (_) para separar palabras que forman el nombre de la clase. En fase de diseño es una práctica común que la primera letra de la siguiente palabra esté en mayúscula. *Esta última recomendación la aplicaremos en **todos** los diagramas de clases que se realicen en la asignatura.*

Ejemplo 6: Revise las siguientes imágenes y determine que palabras indicadas son clases y cuales son objetos. En el caso de las clases indique cuales cumplen con la nomenclatura y en caso de que no la cumplan, defínala de manera correcta. Finalmente modele las clases de las imágenes.



Ryu: No es una clase, porque su nombre hace referencia a un peleador específico. Si se tomara como clase, cada objeto que se creara debería ser únicamente Ryu. Como sabemos en Street Fighter hay muchos peleadores. En pocas palabras el nombre Ryu no es adecuado para representar una clase desde la que se puedan crear objetos.

Ken: Sucede exactamente lo mismo que con Ryu.

jugador: Es un buen nombre para extraer objetos que sean jugadores (por ejemplo, los objetos Ryu y Ken), pero no cumple con la nomenclatura, ya que la primera letra del nombre debe estar en Mayúscula (**Jugador**).

Peleador: Es buen nombre para representar una clase desde la cual se puede extraer los objetos tales como Ryu y Ken.

Barra_de_energía: Es un buen nombre para representar las barras de energía de cada peleador, pero no cumple con la nomenclatura, ya que no se utilizarán guiones bajo para separar las palabras del nombre de la clase. Se puede usar **BarraDeEnergia**.

Escenario: Es un buen nombre para representar el lugar donde ocurre el enfrentamiento entre Ryu y Ken. Cumple con la nomenclatura.

BarrasCounter: Es un buen nombre para representar las barras counter de cada peleador, pero se debe recordar que una clase es un molde desde el cual se extraen los objetos. Por ese motivo, la nomenclatura indica que debe ser un nombre en singular. Por lo tanto, un nombre correcto es **BarraCounter**. De esta clase se extraerán las barras counter para cada uno de los peleadores.

Finalmente procedemos a diagramar las clases involucradas en la imagen:



Ahora realice de manera similar este ejercicio (es obligatorio) de manera correcta y completa.



Pinki:
 Fantasma:
 PacMan:
 Guinda:
 Vida:
 puntajeActual:
 píldora:
 PrimerNivel:
 SuperPíldora:

Características avanzadas de una clase

Hasta el momento hemos referido que una clase consta de tres partes importantes: el nombre los atributos y las operaciones. Además de estos elementos, **una clase (también denominada “clasificador”)** cuenta con algunas propiedades que permiten precisar con mayor detalle aspectos propios de la clase modelada. Estos son: **visibilidad**, alcance, multiplicidad y elementos abstractos, raíces, hojas y polimórficos, que permiten especificar el ámbito del clasificador.

En este momento nos centraremos en la visibilidad, ya que se aplican tanto a los atributos como a las operaciones, cuyas nomenclaturas vamos a encarar inmediatamente.

La visibilidad establece **la disponibilidad que ofrece una clase a otras clases con respecto al uso o acceso de sus atributos y operaciones**. Esto es, en caso de requerirlo se puede determinar si un atributo u operación está totalmente disponible a otra clase; o caso contrario se puede

determinar que un atributo u operación no esté disponible para ninguna otra clase. Existen algunos niveles intermedios entre estos dos extremos, principalmente a nivel del lenguaje de programación que se use.

En UML existen cuatro niveles de visibilidad:

1. Nivel Público: la visibilidad 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 visibilidad se extiende únicamente a las subclases de la clase original (los conceptos de superclases y subclases se verán cuando se haga referencia a una 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 (-)
4. Nivel Paquete (package): Incorporado en UML 2.5 por la influencia de los lenguajes de programación que la incluyen. La visibilidad se extiende a las clases que comparten el mismo paquete en el cual se ha definido la clase origen; por lo cual cualquier clase fuera de este paquete no podrá ver el atributo u operación marcado con este nivel; a menos que sean subclases. Está representada por el literal (~).

Ejemplo 7: La siguiente es una imagen de Animal Crossing: New Horizons para Nintendo Switch, uno de los mejores juegos del año 2020 para la consola.



Teniendo en cuenta que solamente sabe que una clase UML puede dividirse en tres partes y que la visibilidad se puede aplicar tanto a atributos y operaciones, realice una clase del personaje principal con al menos 3 atributos y 2 operaciones; donde la visibilidad de uno de los atributos sea privada, otra package y la última pública. Respecto de las operaciones que una de ellas sea pública y la otra protegida.

Teniendo en cuenta lo solicitado una posible respuesta al ejercicio sería la siguiente

PersonajePrincipal
~ nombre : string
- edad : integer
+ fechaNacimiento : date
+ realizarLanzamientoCaña()
recogerCaña()

Como puede observar la visibilidad es ubicada a la izquierda de los atributos y de las operaciones sobre las cuales se aplica.

La visibilidad es una característica controversial, en el sentido de que se ha incluido por la influencia de los lenguajes de programación orientados a objetos. A pesar de que UML es independiente de los lenguajes de programación y de que se aplica al modelado en etapas anteriores a la programación; con lo cual UML se debería aplicar únicamente al modelado

conceptual; resulta que actualmente resulta común usar UML en todas las etapas de desarrollo software.

La siguiente tabla permite observar la accesibilidad respecto de la visibilidad establecida para los atributos y operaciones de una clase.

Accesibilidad desde	public (+)	private (-)	protected (#)	package (~)
La misma clase	Si	si	si	si
Subclases	Si	no	si	si
Otras clases	Si	no	no	Solo si están en mismo paquete

DEFINICIÓN DE ATRIBUTOS

Ahora entramos en un terreno un poco más complejo, debido a que la definición de un atributo involucra muchos aspectos para tener en cuenta. **Un atributo es una característica específica de un objeto; que en definitiva es una variable que almacena un valor.** Como hemos estudiado anteriormente una variable consta de un identificador y un tipo de datos. Por lo tanto, se hace necesario, poder definir los tipos de datos predeterminados que posee UML con el objetivo de poder tenerlos a disposición al momento de definir los atributos de una clase de manera completa y precisa. En UML se distinguen los siguientes tipos de datos predefinidos:

Nombre	Descripción	Ejemplo
Boolean	Indica que solo puede tomar un valor lógico. Es decir, puede almacenar el valor verdadero o falso.	Suponga una variable que indica si ha finalizado el tiempo de una fase de Super Mario Bros. En UML sería: <code>isFinishedTime: boolean</code>
Byte	Indica un tipo numérico para enteros con un rango de valores pequeño. Generalmente entre -128 a 127.	Suponga una variable que almacena la cantidad de vidas de un personaje. En UML sería: <code>cantidadVidas: byte</code>
Short	Indica un tipo numérico para enteros con un rango más grande que el anterior.	El puntaje de bonus que se obtenía en cada nivel del mítico juego Duck Hunt por eliminar a todos los patos era de 30000. Entonces en UML para poder representar un atributo que pueda almacenar ese valor sería <code>puntajeBonus: short</code>
Integer	Indica un tipo numérico para enteros con rangos muy grandes	El juego Duck Hunt contaba con 99 niveles, pero si realizabas 28 juegos perfectos podías alcanzar el nivel máximo del juego que es 999.999. Esto se debió a las limitaciones de memoria de esa época. Si hoy se programara el mismo juego el puntaje máximo debería definirse de esta manera <code>puntajeAcumulado: integer</code>
Long	Indica un tipo numérico para enteros muchos más grandes que el integer. Generalmente	Suponga que desea expresar la distancia en km que existe desde nuestro planeta hasta Saturno. Un buen tipo de datos a aplicar sería long, de la siguiente manera:

	se utiliza para aplicaciones que realizan cálculos científicos muy grandes.	<code>distanciaAsaturno: long</code>
Float	Indica un tipo numérico para números reales. Normalmente será suficiente para la mayoría de las aplicaciones de videojuegos que requieran manipular decimales	Suponga que realiza un juego educativo en el cual se plantea en diversos puntos la posibilidad de avanzar, únicamente si el jugador responde el valor correcto a operaciones matemáticas básicas. Como ud sabe la división puede generar como resultado valores reales, por lo cual una buena variable sería <code>resultadoOperacion: float</code>
Double	Indica un tipo numérico para números reales muy grandes. Normalmente se utilizan para aplicaciones que realizan cálculos científicos.	El tiro parabólico que describe una bala disparada por un cañón puede ser aproximados por métodos numéricos, que utilizan diversos cálculos basados en la derivada de una función. El resultado de la operación puede almacenarse en una variable de esta manera <code>derivada: double</code>
Date	Indica un tipo numérico para valores que representan fechas	La ficha de los peleadores de Street Fighter incluye su fecha de nacimiento. La clase Peleador puede incluir un atributo que se denomine <code>fecNac: date</code>
String	Indica un tipo que representa cadenas de caracteres. Las cadenas de caracteres permiten formar palabras usando las letras del abecedario, incluyendo espacios vacíos y símbolos.	Otro de los elementos que involucra la ficha de peleadores de Street Fighter es la nacionalidad. La clase Peleador puede incluir un atributo que se denomine <code>Nacionalidad: string</code>
Char	Indica un tipo que representa a un único carácter.	Normalmente en los juegos que utilizan el movimiento basado en la posición arriba, abajo, izquierda y derecha, suelen almacenar la posición seleccionada desde el teclado mediante la siguiente configuración: A: IZQUIERDA D: DERECHA W: ARRIBA S: ABAJO Para almacenar el movimiento realizado por un personaje se podría usar la siguiente variable <code>teclaSeleccionada: char</code>

Con esta información se puede indicar la sintaxis para definir atributos en una clase:

```
<<estereotipo>> visibilidad nombre : tipo [multiplicidad] = valor_inicial
{cadena-de-propiedades}
```

Donde:

- Estereotipo: Característica de UML que permite enriquecer la expresividad del lenguaje introduciendo clasificadores con semántica adicional (no se estudiarán en la asignatura)
- Visibilidad: aquí se introduce el símbolo que indica la visibilidad que tendrá el atributo.
- Nombre: Indica el identificador o nombre de la variable que representa el atributo.
- Multiplicidad: La multiplicidad hace referencia al número al número de ocurrencias del atributo que puedo tener. Este concepto se abordará con mayor detalle más adelante.
- Valor inicial: Representa el valor inicial o por defecto que tendrá un objeto al momento de ser creado.
- Cadena de propiedades: Permite especificar, de una manera más específica, qué propiedades quiero que el atributo tenga. Entre ellas destacan, por ejemplo:


frozen, que quiere decir que el atributo no va a poder ser modificado.

addOnly, que hace referencia a atributos que tiene una multiplicidad mayor que uno y que indica que sólo se puede añadir nuevos valores a ese atributo, pero no se puede modificar o borrar.

changeable, es la habitual y que quiere decir que un atributo puede cambiar su valor como consecuencia de la ejecución de las operaciones de la clase.

Existen otras propiedades como las restricciones en las cuales se utilizan operadores relaciones pero que no se ahondarán en esta materia.

Ejemplo 8: Bien ahora brindaremos ejemplos de atributos en base a imágenes

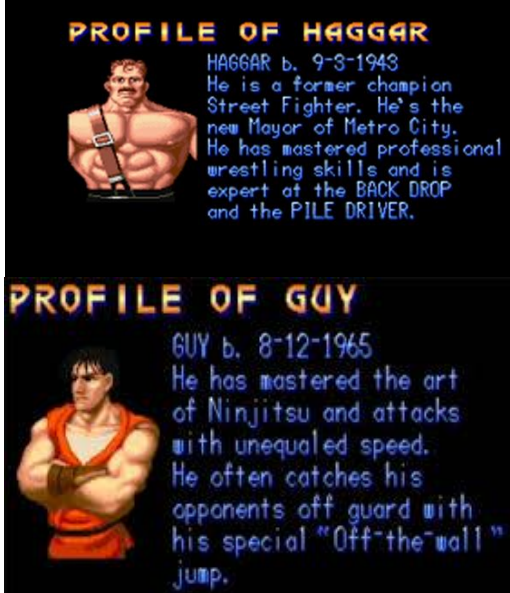
	<p>Bloque</p> <ul style="list-style-type: none"> - color : Color - tamaño : integer - premio : string 	<p><<type>> Color</p> <ul style="list-style-type: none"> + value : undefined {Value:(255,128,0)}
---	---	--

Bien, en este ejemplo tenemos un típico bloque de premios del Juego Super Mario Bros. Se debe recordar que según la narrativa del juego estos son en realidad aldeanos del reino Champignon convertidos en bloques por Bowser. Para ayudar a Mario, ellos tienen la capacidad de otorgarle un premio cada vez que los golpean.

El nombre de la clase es **Bloque**, tiene un atributo denominado tamaño de tipo numérico entero. En premio se guarda mediante una cadena de caracteres un que valor podría ser “moneda”, “hongo rojo”, “hongo verde”, “estrella” o “flor de fuego”.

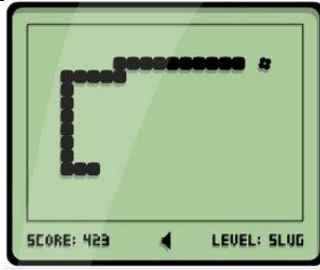
Finalmente observe que el bloque tiene un color. Muchas veces el color dentro del mundo de los videojuegos es provisto por una clase. Por ese motivo se crea la clase **Color** y se aplica el estereotipo <<type>>, que indica que se usa la clase como un tipo de datos.

Observe, además, que el valor por defecto (naranja) se establece como un color RGB, es decir una combinación entre una tonalidad de rojo, verde y azul. Si bien lo normal hubiera sido que se escriba de la siguiente manera **-color: Color = (255,128,0)**; resulta que existen herramientas UML que por alguna razón no soportan el = **valor**. En esos casos se usa la cadena de propiedades {**Value:** } para indicar el valor por defecto.



Peleador
- nombre : string
- fechaNacimiento : date
- profesion : string
- movimientosEspeciales : string [1..*]

En este ejemplo tenemos los perfiles de los peleadores del juego Final Fight. En el juego se pueden elegir a Haggar, Guy y a Cody (aunque de este último no se incluye imagen). Se podrían considerar a Haggar y Guy objetos de una clase **Peleador**. Esta clase deberá tener los atributos necesarios para poder almacenar los datos del perfil de cada peleador. El nombre de la clase es **Peleador**, tiene un atributo denominado *nombre* de tipo string que permite almacenar el nombre del peleador. El atributo *fechaNacimiento* registra la fecha de nacimiento de cada peleador, mientras que la profesión de cada uno de ellos se registra en *profesion*. Note como los nombres de los atributos indican que van a almacenar. Observe que Guy posee un único movimiento especial, mientras que Haggar registra dos. En este caso podemos indicar, a través de la multiplicidad esta característica; así [1..*] indica que el peleador puede tener 1 o más movimientos especiales.



Nivel	Posicion	Snake	Cuadrado
- nombre : string	- posX : integer - posY : string	- puntaje : integer - cuadrados : Cuadrado [1..*] {addOnly} - posicion : Posicion	- lado : integer
PorcionAlimento			
- posicion : Posicion			

En este ejemplo tenemos una imagen del famoso juego Snake. En las primeras versiones del juego la víbora estaba formada por cuadrados. La longitud de la víbora aumentaba al incrementarse la cantidad de cuadrados que conforman su cuerpo (al comer una porción de alimento ubicada de manera fija en la pantalla aumenta en una unidad). Imagine que debe programar este Snake clásico, lo primero será poder realizar la abstracción del juego identificando clases y atributos (ya que todavía no hemos visto el modelado de operaciones). Como podemos observar se ha identificado cinco clases. En primer lugar, el **Nivel**, del cual se ha definido un atributo que representa el *nombre* del nivel. En segundo lugar, se ha definido la clase **Cuadrado**, el cual posee un atributo *lado*, que contiene el tamaño de los cuadrados que conforman el cuerpo del Snake. Por esta misma razón, se ha creado una clase **Snake** que posee dos atributos: por un lado, el *puntaje* que va adquiriendo la víbora a medida que avanza



sobre el juego, mientras que por el otro posee un atributo denominado *cuadrados* que representa el cuerpo del Snake. Como se puede observar mediante la multiplicidad se puede indicar que **Snake** puede estar formado por uno o más cuadrados. Además, mediante la propiedad de cadenas *{addOnly}* sabemos que su cuerpo solamente puede crecer.

En esta definición se ha agregado un tipo de datos que resulta muy común en los juegos 2D, que es la posición del objeto; en la forma de una clase **Posicion** con estereotipo <<type>> que es usado como atributo tanto en **PorcionAlimento**, como en **Snake**.

Nomenclatura adoptada para la definición de atributos

Probablemente se haya percatado que, en los ejemplos anteriores, se ha utilizado una nomenclatura para la definición del identificador de un atributo. Esta nomenclatura está basada en el lenguaje de programación JAVA y será la adoptada para la cursada de la materia. Es probable que en otras materias esto cambie. A continuación, se describe la nomenclatura utilizada:

1. *Inician con letra minúscula*
2. No puede iniciar con un número
3. El nombre debe ser representativo del significado de la característica de la clase que se desea exponer. Por ejemplo, suponga que tiene una clase **NaveEspacial** y quiere indicar que tiene una cantidad de vidas por atributo; un atributo válido sería *cantVidas*, mientras que un atributo inválido sería *v*, o *cv*.
4. No se admite espacios en los nombres de atributos. En el Diagrama de clases en fase de análisis se admite el uso de guion 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*. En la asignatura usaremos siempre esta segunda opción.

DEFINICIÓN DE OPERACIONES

Como se ha señalado anteriormente, **una operación denota una acción que puede realizar el objeto creado a partir de una clase**. Esto significa que todas las operaciones que se definan en una clase podrán ser realizadas por los objetos que se creen a partir de ella. En términos formales, **una operación es un “servicio” que un objeto pone a disposición de otros objetos, una capacidad de acción que puede ser solicitada por algún otro objeto**. La nomenclatura para la definición de operaciones que se usará en la asignatura se corresponde con la que es usada en el lenguaje Java:

1. Deben ser verbos en infinitivo
2. *Se escribe en minúscula.*
3. No admiten espacios en los nombres. En el Diagrama de clases en fase de análisis se admite el uso de guion 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*.

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

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

Los **[]** indican que son opcionales, por tanto, lo único requerido es el nombre de la operación.


Cada parámetro de la lista de parámetros 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.

Ejemplo 9: En base a las siguientes imágenes modele clases con sus respectivos atributos y operaciones

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><th style="text-align: center;">Nivel</th></tr> <tr><td>- numNivel : byte</td></tr> <tr><td>- longitudCamino : integer</td></tr> <tr><td>+ mostrarDatos(in auto: Auto)</td></tr> <tr><td>+ mostrarPosicionAuto(in posicionAuto: Posicion)</td></tr> </table>	Nivel	- numNivel : byte	- longitudCamino : integer	+ mostrarDatos(in auto: Auto)	+ mostrarPosicionAuto(in posicionAuto: Posicion)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><th style="text-align: center;">Auto</th></tr> <tr><td>- puntaje : integer</td></tr> <tr><td>- velocidad : integer</td></tr> <tr><td>- nivelCombustible : integer</td></tr> <tr><td>- color : Color</td></tr> <tr><td>- posicion : Posicion</td></tr> <tr><td>+ acelerar()</td></tr> <tr><td>+ frenar()</td></tr> <tr><td>+ mover(in direccion: char)</td></tr> </table>	Auto	- puntaje : integer	- velocidad : integer	- nivelCombustible : integer	- color : Color	- posicion : Posicion	+ acelerar()	+ frenar()	+ mover(in direccion: char)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><th style="text-align: center;">Color</th></tr> <tr><td>+ value : undefined</td></tr> </table> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><th style="text-align: center;">Posicion</th></tr> <tr><td>- posX : integer</td></tr> <tr><td>- posY : string</td></tr> </table>	Color	+ value : undefined	Posicion	- posX : integer	- posY : string
Nivel																					
- numNivel : byte																					
- longitudCamino : integer																					
+ mostrarDatos(in auto: Auto)																					
+ mostrarPosicionAuto(in posicionAuto: Posicion)																					
Auto																					
- puntaje : integer																					
- velocidad : integer																					
- nivelCombustible : integer																					
- color : Color																					
- posicion : Posicion																					
+ acelerar()																					
+ frenar()																					
+ mover(in direccion: char)																					
Color																					
+ value : undefined																					
Posicion																					
- posX : integer																					
- posY : string																					
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><th style="text-align: center;">SuperHeroe</th></tr> <tr><td>+ volar()</td></tr> </table>	SuperHeroe	+ volar()	<div style="display: flex;">  <div style="width: 70%;"> <p>El juego Road Fighter es un clásico del NES (o Family Game como se conoce por estos pagos). Observe que se han definido las clases Auto, Nivel y SuperHeroe como aquellas que poseen operaciones; mientras que Color y Posición las conocemos de ejemplos anteriores.</p> <p>¿Qué acciones puede realizar un objeto de tipo Auto? <i>acelerar()</i>, <i>frenar()</i>, <i>mover()</i> son ejemplos claros de operaciones que cumplen la nomenclatura. Todos son verbos en infinitivo, todos indican claramente lo que realizará el objeto.</p> <p>Observe el método <i>mover(in direccion:char)</i>. Es un método que recibe parámetros. Esto significa que para poder moverse el auto requiere que le indique a que dirección se moverá. Un parámetro es una variable que recibe la operación. Puede imaginarse que se usará los valores 'a', 'w', 's' y 'd' para determinar la dirección. Entonces la operación generará el movimiento de acuerdo con este valor.</p> </div> </div>																		
SuperHeroe																					
+ volar()																					

	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><th style="text-align: center;">Calculadora</th></tr> <tr><td>-estado: boolean</td></tr> <tr><td>+prenderOLimpiar()</td></tr> <tr><td>+getRaizCuadrada(out numero:integer): float</td></tr> <tr><td>+getSuma(numeroA:integer,numeroB:integer): integer</td></tr> </table>	Calculadora	-estado: boolean	+prenderOLimpiar()	+getRaizCuadrada(out numero:integer): float	+getSuma(numeroA:integer,numeroB:integer): integer
Calculadora						
-estado: boolean						
+prenderOLimpiar()						
+getRaizCuadrada(out numero:integer): float						
+getSuma(numeroA:integer,numeroB:integer): integer						

Saliendo un poco de los ejemplos de videojuegos, analicemos una calculadora desde el enfoque de modelado de clases.

La operación *prenderOLimpiar()* limpia la pantalla si el estado es true (verdadero), caso contrario prende la calculadora, además seguramente cambia el valor del atributo estado. Esta operación sucede cuando se presiona el botón de color naranja.

Observe que las otras operaciones empiezan con **get**, que está en inglés y significa obtener. Es común usar esta notación cuando un método devuelve algún valor.

En particular centrémonos en *getRaizCuadrada()*; esta operación se activa cuando se presiona la tecla que tiene el símbolo $\sqrt{\quad}$. Esta operación recibe un parámetro, que es el valor que ingresó la persona en la calculadora. La operación toma ese valor, obtiene su raíz cuadrada (de ser posible) y devuelve el valor generado. Esto último se refleja en el hecho de que la operación finaliza con **:float**, es decir devuelve un valor.

Pero, además, como el parámetro *numero* tiene una dirección **out**. **Esto significa que se alterará su valor con el resultado** y tanto el parámetro como el resultado comparten el mismo espacio en memoria, siendo más eficiente que generar una nueva variable para el resultado. **Sin embargo, en este caso hay un error conceptual, por cuanto los tipos no coinciden.**

Ahora observe la operación *getSuma()*. Recibe dos parámetros, los cuales se separan mediante comas. **Si no indica la dirección de los parámetros por defecto se asumen in**. En este caso la operación tomará ambos parámetros, los sumará y devolverá el resultado en una variable tipo integer.