U. T. 4 P.O.O.



Introducción

La programación Orientada a objetos se define como un paradigma de la programación, una manera de programar específica, donde se organiza el código en unidades denominadas clases, de las cuales se crean objetos que se relacionan entre sí para conseguir los objetivos de las aplicaciones. La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.

Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser reutilizados se creó la posibilidad de utilizar módulos. El primer módulo existente fue la función, que somos capaces de escribir una vez e invocar cualquier número de veces.

Sin embargo, la función se centra mucho en aportar una funcionalidad dada, pero no tiene tanto interés con los datos. Es cierto que la función puede recibir datos como parámetros, pero los trata de una forma muy volátil. Simplemente hace su trabajo, procesando los parámetros recibidos y devuelve una respuesta.

Con la Programación Orientada a Objetos se buscaba unificar los datos y su gestión en una única estructura, creando unas mejores condiciones para poder desarrollar aplicaciones cada vez más complejas, sin que el código se volviera un caos. Además, se pretendía dar unas pautas para realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar.

> la Programación Orientada a Objetos, un paradigma de programación que permite desarrollar aplicaciones complejas manteniendo un código más claro y manejable que otros paradigmas anteriores

Cómo se piensa en clases y objetos

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo, vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además, tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcar.

Pues en un esquema POO "el coche" sería lo que se conoce como "Clase". Sus características, como el color o el modelo, serían propiedades y las funcionalidades asociadas, como ponerse en marcha o parar, serían métodos.

La clase es como un libro, que describe como son todos los objetos de un mismo tipo. La clase coche describe cómo son todos sus coches, qué propiedades tienen y qué funcionalidades deben poder realizar. A partir de una clase podemos crear cualquier número de objetos de esa clase. Un coche rojo que es de la marca Ford y modelo Fiesta, otro verde que es de la marca Seat y modelo Ibiza.

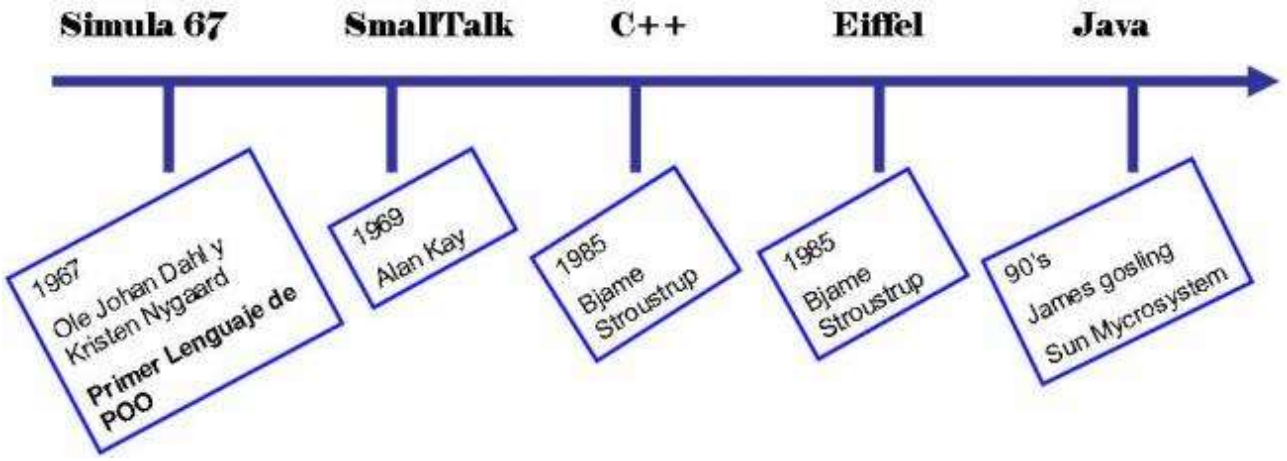
Identificar las clases es una de las tareas más importantes en la programación orientada a objetos y de esta tarea dependerá la calidad del producto final.

Orígenes

Los conceptos de la programación orientada a objetos tienen origen en Simula 67, un lenguaje diseñado para hacer simulaciones, creado por Ole-Johan Dahl y Kristen Nygaard, del Centro de Cómputo Noruego en Oslo. En este centro se trabajaba en simulaciones de naves y La idea surgió al agrupar los diversos tipos de naves en diversas clases de objetos, siendo responsable cada clase de objetos de definir sus propios datos y comportamientos. Fueron refinados más tarde en Smalltalk, desarrollado en Simula en Xerox PARC (cuya primera versión fue escrita sobre Basic) pero diseñado para ser un sistema completamente dinámico en el cual los objetos se podrían crear y modificar “sobre la marcha” (en tiempo de ejecución) en lugar de tener un sistema basado en programas estáticos.

La programación orientada a objetos se fue convirtiendo en el estilo de programación dominante a mediados de los años ochenta, en gran parte debido a la influencia de C++, una extensión del lenguaje de programación C. Su dominio fue consolidado gracias al auge de las Interfaces gráficas de usuario, para las cuales la programación orientada a objetos está particularmente bien adaptada. En este caso, se habla también de programación dirigida por eventos. Las características de orientación a objetos fueron agregadas a muchos lenguajes existentes durante ese tiempo, incluyendo Ada, BASIC, Lisp y Pascal, entre otros.

La adición de estas características a los lenguajes que no fueron diseñados inicialmente para ellas condujo a menudo a problemas de compatibilidad y en la capacidad de mantenimiento del código. Los lenguajes orientados a objetos "puros", por su parte, carecían de las características que muchos programadores se habían acostumbrado a utilizar. Para evitar este obstáculo, se hicieron muchas tentativas para crear nuevos lenguajes basados en métodos orientados a objetos, pero permitiendo algunas características imperativas de maneras "seguras". El Eiffel de Bertrand Meyer fue un temprano y moderadamente acertado lenguaje con esos objetivos, pero ha sido reemplazado por Java, en gran parte debido a la aparición de Internet y a la implementación de la máquina virtual de Java en la mayoría de navegadores. PHP en su versión 5 se modificó, soportando una orientación completa a objetos, cumpliendo todas las características propias de la orientación a objetos.



Características de la POO

La POO posee muchas e importantes características que aporta con respecto a otros paradigmas para el desarrollo de aplicaciones, entre las que podemos destacar las siguientes:

- Permite enfocar los problemas de una forma mucho más intuitiva de lo que lo hacen otros paradigmas.
- Mientras que, en la programación estructurada datos y código están totalmente separados, en la POO se encuentran unidos en una única entidad.
- Se persigue trabajar con un modelo más cercano al mundo real en el que los diferentes objetos interaccionan entre sí.
- Permite generar programas modulares y muy escalables.
- La mejor opción para conseguir una buena experiencia de usuario.
- Su capacidad para permitir reutilizar código sin necesidad de reescribirlo nuevamente facilitó mucho la tarea de los programadores, que vieron en esta tendencia una gran oportunidad de rentabilizar el esfuerzo realizado.

Las propiedades de la POO son:

- **Abstracción**. Es un proceso mental por el que se ignoran las características de algo, quedándonos con lo que realmente nos importa. La abstracción es algo que hacemos constantemente los humanos en nuestro día a día. Si no lo realizáramos nuestro cerebro se colapsaría con toda la información que nos rodea. Una forma de manejar la complejidad del software es conseguir escribir código de tal manera que permita la abstracción.

- **Encapsulamiento**. Es el proceso por el cual se integran en una única estructura los datos y los mecanismos de gestión sobre esos datos, creando un todo. Una clase se compone tanto de variables (atributos) como de funciones y procedimientos (métodos).
- **Ocultación**. Hay una zona oculta al definir las clases (zona privada) que sólo es utilizada por esa clase y por alguna clase relacionada. Hay una zona pública (llamada también interfaz de la clase) que puede ser utilizada por cualquier parte del código. Una clase define la visibilidad de sus datos y métodos para que otras clases tengan acceso a ellos y tener un control total sobre la gestión de los mismos.
- **Polimorfismo**. Cada método de una clase puede tener varias definiciones distintas.
- **Herencia**. Una clase puede heredar propiedades (atributos y métodos) de otra creando de esta manera una jerarquía de clases. La herencia se puede hacer desde un único padre (herencia simple) o desde varios padres (herencia múltiple).

Ventajas y desventajas de la POO

Ventajas	Desventajas
Reusabilidad	Cambio en la forma de pensar, más difícil
Facilidad de desarrollo y mantenimiento	Curva de aprendizaje más larga
Facilidad de modificación	Ejecución más lenta
Aislamiento de los errores	No todos los problemas se pueden resolver con este enfoque
Reducción de código redundante	No es productivo para tareas simples
Fácil entendimiento de la lógica	
Cada componente tiene independencia de los demás	

Conceptos Fundamentales POO

Para hacer un desarrollo en POO debemos saber los conceptos fundamentales de la misma y los principios de programación. Según Robert C. Martin existen cinco principios básicos que constituyen la programación orientada a objetos son:

- **Principio de responsabilidad única**. Cada clase debe tener una única responsabilidad, y esta responsabilidad debe estar contenida únicamente en esa clase. Cada responsabilidad es el eje y la razón de cambio. Para contener la propagación del cambio, se deben separar las responsabilidades.
- **Principio de abierto-cerrado**. Una entidad (clase, módulo, función, etc.) debe quedarse abierta para su extensión, pero cerrada para su modificación. Si un cambio impacta a varios módulos, entonces la aplicación no está bien diseñada. Se deben diseñar módulos que procuren no cambiar y así, reutilizar el código más adelante(extensión).
- **Principio de sustitución**. Si en alguna parte de un programa se utiliza una clase, y esta clase es extendida, se puede utilizar cualquiera de las clases hijas sin que existan modificaciones en el código. Cada clase que hereda de otra puede usarse como sus padres sin necesidad de conocer las diferencias entre ellas.
- **Principio de segregación de interfaz**. Hace referencia a que muchas interfaces cliente específicas son mejores que una interfaz de propósito general. Se aplica a una interfaz amplia y compleja para dividirla en otras más pequeñas y específicas, de tal forma que cada cliente use solo aquella que necesite pudiendo así ignorar al resto.
- **Principio de inversión de dependencias**. Los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de abstracciones.



La herramienta **Dia** es muy útil para hacer el diseño bajo POO

Pasaremos a ver los conceptos fundamentales de la POO en los siguientes apartados.

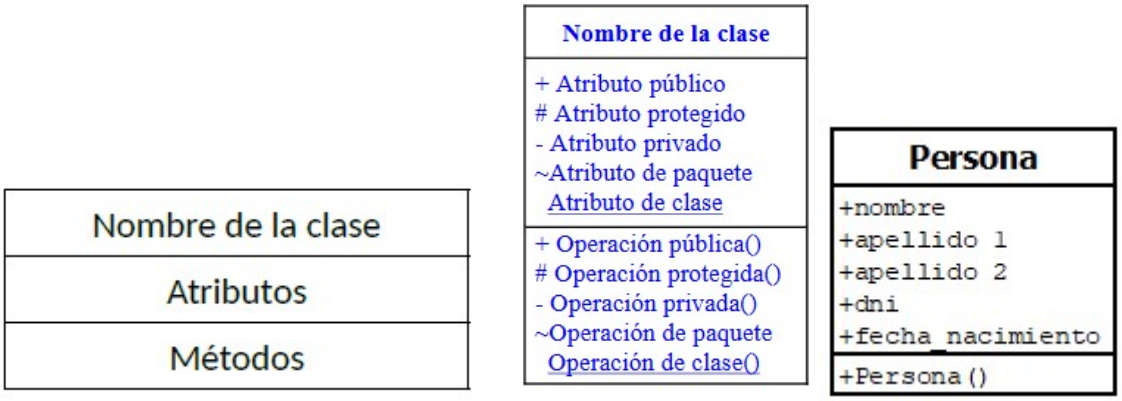
Clases

Una clase describe un grupo de objetos que contienen una información similar (atributos) y un comportamiento común (métodos), este conjunto de objetos se denomina **interfaz**. El interfaz público es el conjunto de atributos y métodos accesibles a los demás objetos, es muy importante definirlo correctamente ya que si cambiamos la firma (parámetros y tipos de parámetros) dejaremos a los objetos que los utilicen inservibles.

Las definiciones comunes (nombre de la clase, los nombres de los atributos, y los métodos) se almacenan una única vez en cada clase, independientemente de cuántos objetos de esa clase estén presentes en el sistema. Una clase es como un molde a partir de cual se pueden crear objetos.

Antes de poder utilizar un objeto se debe definir la clase a la que pertenece, esa definición incluye:

- **Propiedades**. Las variables miembros de la clase.
- **Métodos**. Las funciones miembros de la clase.
- **Visibilidad**. Tanto atributos como métodos tienen una visibilidad con respecto a ella misma y a las demás clases del programa.



Propiedades o atributos

Una propiedad es una característica única del objeto, se asemejan a las variables de la programación estructurada, recoge información de la clase. La propiedad puede contener cualquier valor, incluido otro objeto de una clase. Las propiedades pueden ser reales o generadas. Las propiedades reales almacenan valores en memoria, las propiedades generadas simulan una propiedad a través de un método. El valor de estas últimas propiedades es la capacidad de control que generan y la posibilidad de crear valores calculados complejos, no solo los almacenados. En el ejemplo anterior las propiedades serían: *dni, nombre, apellido1, apellido2* y *fecha_nacimiento*.

Métodos

Una clase es una estructura de datos que incluye las operaciones necesarias para gestionarla. Esas operaciones son los llamados métodos. Un método no es más que una función o procedimiento de gestión de la clase que puede ser utilizado por esta o por otras. En el ejemplo anterior, el método sería Persona().

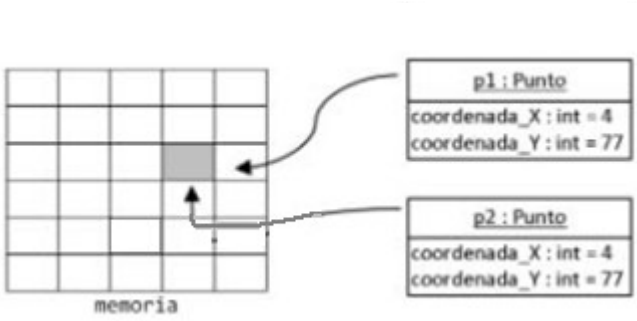
Todas las clases implementan métodos, pero hay dos que se implementarán siempre independientemente de que nosotros los creemos o no. Estos métodos son el constructor de la clase y el destructor. El constructor se usa en el momento que creamos un objeto y será llamado por el sistema una vez la estructura se haya creado en memoria para inicializarlo. Del mismo modo, el destructor es el último método que se ejecuta cuando hayamos solicitado la eliminación de un objeto para liberar recursos.

Ejemplo: Define gráficamente una clase Bombero considerando las siguientes propiedades de clase: nombre (Cadena), apellidos (Cadena), edad (Entero), casado (Booleano), especialista (Booleano).

Define un constructor (método con el mismo nombre que la clase) que reciba los parámetros necesarios para la inicialización y los métodos para poder establecer y obtener los valores de los atributos.

Objetos

Las clases son el molde que determina la estructura de "algo" que podemos utilizar. El objeto es la instanciación de una clase, es la creación de ese "algo". Un objeto se crea a partir de la clase con los métodos y propiedades que ésta determina. Un objeto de una clase es independiente de otro objeto de la clase y el estado de un objeto será diferente al de otro objeto de la misma (se conoce como estado de un objeto al valor de todas sus propiedades).



Dos objetos no serán nunca el mismo, incluso aunque su estado sea el mismo, a menos que apunten a la misma área de memoria (es lo que se denomina clonado del objeto).

Los objetos se crear un se destruyen de forma dinámica a través de las herramientas del lenguaje elegido, los utilizamos durante todo el desarrollo de la aplicación para guardar información, realizar acciones sobre ellos, etc.

Las clases se utilizarán cuando queramos acceder a datos o métodos compartidos entre todos los objetos de la clase.

Ejemplo: Crea la siguiente estructura bajo Python. Y haz uso de ella.



Encapsulación y Ocultación

La clase forma un todo, aúna los datos y las operaciones en una única estructura de datos, permite la abstracción, la ocultación, el encapsulamiento, la herencia y el polimorfismo.

La encapsulación es el mecanismo por el que una clase agrupa todos los datos y operaciones dentro de una única estructura. Con la encapsulación un objeto puede ocultar la información que contiene al mundo

exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada, es lo que se denomina visibilidad.

Visibilidad	Public	Private	Protected
Desde la misma clase	✓	✓	✓
Desde una subclase	✓	⊘	✓
Desde otra clase (no subclase)	✓	⊘	⊘

La visibilidad permite que los métodos y propiedades de una clase sean utilizados de forma correcta por otros elementos del programa. La visibilidad puede ser pública (el signo **+** se utiliza para representar la visibilidad en una definición de clase), privada (**-**) o protegida (**#**). En caso de visibilidad pública cualquier parte del programa tendrá acceso, tanto la propia definición de clase como partes externas a la misma. Si la visibilidad es privada, solo la clase tiene acceso a dichos datos u operaciones, ni partes externas ni otras clases podrán utilizarla. Por último, en caso que sea protegida, tanto la propia clase, como cualquiera que herede de ella podrán utilizar el método o propiedad, estando prohibido el uso a otras clases o partes externas.

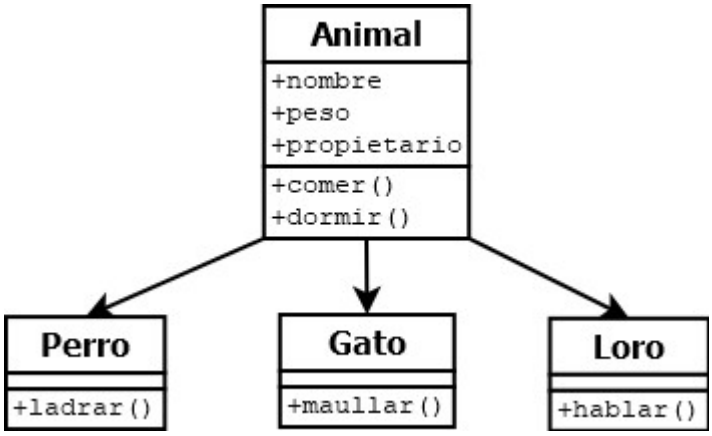
Abstracción

La abstracción es el mecanismo por el que podemos reducir la complejidad de la información, recogiendo solo aquella información que es relevante. Consiste en la generalización conceptual de los atributos y comportamiento de un determinado conjunto de objetos.

Pongamos un ejemplo para comprenderlo mejor. Queremos modelizar una clínica veterinaria en la que vamos a tratar varios tipos de animales: perros, gatos, loros, por simplicidad. Si hacemos un estudio llegamos a la conclusión que necesitamos tres clases diferentes para cada animal.



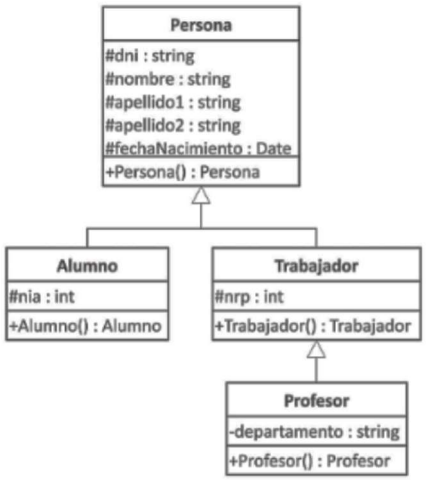
Como podemos observar casi todos los métodos y propiedades son similares, por lo que podemos **abstraer** lo importante de las tres clases en otra que llamaremos animal, dejando lo particular en cada una de ellas.



Herencia. Tipos de Herencia

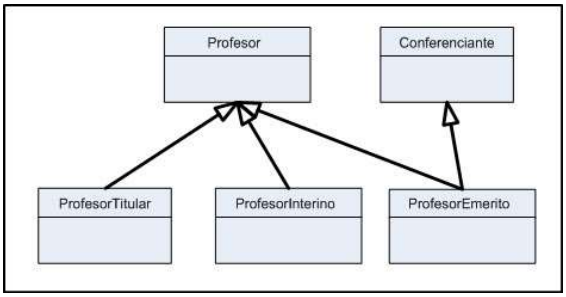
La herencia es la capacidad que tienen las clases de usar las características ya declaradas en otras clases para su propia declaración. La herencia genera una relación de padres e hijos en las clases, una jerarquía, en la que las clases que heredan se sitúan en la parte inferior de la jerarquía, y las clases de las que se heredan en la parte superior.

Como vemos en el ejemplo anterior, las clases perro, gato y loro sería las clases hijas, que heredan las propiedades y métodos de la clase padre animal. Esta jerarquía se puede extender todo lo que se necesite y crear todo lo compleja que el programa requiera.



La herencia se puede presentar en dos vertientes en función de cuántos padres se herede:

- **Herencia simple.** Cada clase hija solo tendrá un único ancestro o padre. Java tiene este tipo de jerarquía.
- **Herencia múltiple.** Cada clase hija podrá heredar de tantos ancestros como requiera. Python tiene este tipo de jarquía.



En el momento que creamos una relación de herencia estamos creando una jerarquía de clases que podremos utilizar para documentar el desarrollo.

Ejemplo: Implementa las estructuras anteriores bajo Python

Ejemplo Definir los siguientes términos:

- Abstracción.
- Ciclo de vida.
- Encuesta.
- Herencia.
- Metodología de desarrollo.
- Observación.
- Polimorfismo.
- Visibilidad.
- Atributo.
- Encapsulamiento.
- Entrevista.
- Método.
- Objeto.
- Orientación a Objetos.
- Sistema.

Ejemplo: Para la siguiente situación debes listar los objetos detectados, realizar la clasificación de clases de los objetos y determinar sus atributos.

José de 30, María de 25 y Pedro de 19 años de edad, todos expertos en alguna rama de la ingeniería, pertenecientes al grupo "IPL", postulan un proyecto "Inversión de Polos electromagnéticos por medio del plástico", ya habían postulado otro proyecto el 2001 y otro más el 2014. En este proyecto el grupo se adjudicó 12 millones de pesos para el desarrollo, el monto fue definido por el concurso "Grandes ideas de chilenos para chilenos", cuyo periodo de postulación duró 6 meses, e inició el 10 de enero del 2016, los otros proyectos fueron presentados en otros fondos y concursos.

Ejemplo: Para la siguiente situación debes listar los objetos detectados, realizar la clasificación de clases de los objetos y determinar sus atributos.

La maratón de Nueva York, es una de las maratones más famosas de la historia, pertenece al selecto grupo de los World Marathon Majors, al cual pertenecen otras importantes maratones mundiales. Inició en 1970, y se ha corrido todos los años, excepto el 2012, cuando fue cancelada por el huracán Sandy. El año 2015 convocó a más de 50.000 atletas. Cubre unos 42.195 kilómetros por toda la ciudad. La carrera inicia en Staten Island, pasa por Brooklyn y Queens, entra a la isla Manhattan y luego avanza por la Primera Avenida, el Bronx, vuelve a Manhattan por la Quinta Avenida, y finaliza en Central Park. El campeón del año 2015 fue Stanley Biwott, de Kenia, con un tiempo de 2:10:34. La campeona fue Mary Keitany, también de Kenia, con un tiempo de 2:24:25.

Ejemplo: Para la siguiente situación debes listar los objetos detectados, realizar la clasificación de clases de los objetos y determinar sus atributos.

El estudio Marvel, presento su Universo Cinemático, con grandes propuestas, para los amantes de los comics, y de las películas de acción, fantasía y ciencia ficción. Esta inició el 2008 con la presentación de Iron Man, esta película fue protagonizada por Robert Downey Jr. Actor norteamericano, en el papel de Tony Stark, la película fue dirigida por Jon Favreau, este año se ha estrenado Captain America: Civil War, protagonizada por Chris Evans, en el papel del Capitán América, y un gran reparto de otros actores y personajes ya presentados en otras películas, esta fue dirigida por los hermanos Russo. Las películas de Marvel se agrupan en fases de entrega, por ejemplo, esta última corresponden a la fase 3, la primera fase inció en el 2008, y se espera que la serie de películas finalice en el 2019 ó 2020.

Soluciones

Actividad N°1: Definir los siguientes términos:

- **Abstracción**: es la acción de aislar a un elemento desde su contexto para realizar un análisis dirigido y delimitado. Interesando mucho más el ¿qué hace? en vez de ¿Cómo lo hace?
- **Atributo**: cualidad, adjetivo que define o declara una característica de un objeto, por ejemplo, para una clase alumno, el nombre y la edad, serían atributos.
- **Ciclo de vida**: se les llama así a las metodologías para desarrollo de software, que inician con la toma de requerimientos para el desarrollo, luego diseñan y construyen, para luego probar y realizar procesos de mantenimiento para preservar un sistema informático a través del tiempo, hasta el momento en que será reemplazado.
- **Encapsulamiento**: es la técnica utilizada para que las clases y objetos sean utilizados a través de los métodos que ellas estimen, evitando que el usuario conozca la estructura y la forma de procesar que tiene, esto asegura que la clase quede protegida por modificaciones externas.
- **Encuesta**: técnica de captura de requerimientos, que se basa en un conjunto de preguntas de rápida resolución, mayormente cerradas o de respuestas pre-escritas. Idealmente utilizada cuando el volumen de las personas a evaluar es alto, o se pretende capturar impresiones, pero no sugerencias y nuevas ideas. La mayoría de las encuestas apuntan a la mejora de la percepción del producto, interfaces, pantallas, documentos y dispositivos.
- **Entrevista**: técnica de captura de requerimientos, que se basa en un conjunto de preguntas de conversación y respuesta amplia, idealmente se utilizan para cuando el volumen de entrevistados es pequeño, o es un conjunto selecto como usuarios finales, focus group, o stakeholders, donde sus impresiones y sugerencias son significativas para determinar los alcances del sistema.
- **Herencia**: capacidad de una clase para adaptar los atributos y métodos de clases superiores, en el caso de Java de una clase superior (no permite herencia múltiple), la herencia presenta las clases en una estructura jerárquica.
- **Método**: es el comportamiento programado de una clase, los atributos de una clase son tratados a través de sus métodos. Los métodos son las acciones que pueden realizar las clases.
- **Metodología de desarrollo**: es el conjunto ordenado de las fases para desarrollar una solución de software. Incluyen el Análisis, el Diseño, la Construcción, las Pruebas y el Mantenimiento.
- **Objeto**: cualquier elemento del mundo real, que nos entregue información a través de sus características propias (atributos) o de lo que hace (métodos), son capaces de comunicarse e interactuar con otros objetos, logrando una simulación de la vida.
- **Observación**: técnica para captar requerimientos en un ambiente laboral, que consiste en ser parte del trabajo para conocer a fondo el actuar de las personas que laboran, para determinar los aspectos automatizables y que, a la vez, provoquen menor impacto en la organización.
- **Orientación a Objetos**: la Programación Orientada a Objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación. Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir nuestros programas en términos de objetos, propiedades, métodos y otras cosas que veremos rápidamente para aclarar conceptos y dar una pequeña base que permita soltarnos un poco con este tipo de programación.
- **Polimorfismo**: es la capacidad de los objetos de implementar sus métodos y que estos reaccionen de manera distinta según la naturaleza de los objetos creados o de los estímulos que esta reciba.
- **Sistema**: un conjunto de elementos que interactúan entre sí para lograr un objetivo común.
- **Visibilidad**: la capacidad de las clases, atributos o métodos, para ser invocadas y utilizadas por otros.

Actividad N°2: Para la siguiente situación debes listar los objetos detectados, realizar la clasificación de clases de los objetos y determinar sus atributos.

Objetos:

- José, María, Pedro.
- IPL
- "Inversión de Polos electromagnéticos por medio del plástico"
- "Grandes ideas de chilenos para chilenos"

Clases y Atributos:

- Postulante (clase)
  - nombre Postulante
  - edad Postulante
  - especialidad Postulante
- Grupo (clase)
  - nombreGrupo
- Proyectos (clase)
  - nombreProyecto
  - añoPresentacionProyecto
  - situacionProyecto
- Concurso (clase)
  - nombreConcurso
  - fondoConcurso
  - duracionPostulacionConcurso
  - fechaInicioConcurso

Actividad N°3: Para la siguiente situación debes listar los objetos detectados, realizar la clasificación de clases de los objetos y determinar sus atributos.

Objetos:

- Maratón de Nueva York.
- World Marathon Majors.
- Huracán Sandy.
- Staten Island, Brooklyn, Queens, Isla Manhattan, Primera Avenida, Quinta Avenida, Central Park, El Bronx.
- Stanley Biwott, Mary Keitany.
- Kenia.

Clases y atributos

- Maratón (clase)
  - nombreMaraton
  - distanciaMaraton
  - cantidadParticipantesMaraton
  - añoInicioMaraton
  - inicioMaraton
  - terminoMaraton

- Agrupación (clase)
  - nombreAgrupacion
- Evento (clase)
  - nombreEvento
  - tipoEvento
  - añoEvento
- Localidad (clase)
  - nombreLocalidad
  - tipoLocalidad
- Campeón.
  - nombreCampeon
  - sexoCampeon
  - tiempoCampeon
  - nacionalidadCampeon
- País (clase)
  - nombrePais
  - banderaPais

Actividad N°4: Para la siguiente situación debes listar los objetos detectados, realizar la clasificación de clases de los objetos y determinar sus atributos.

Objetos:

- Iron Man, Captain America: Civil War.
- Robert Downey Jr., Chris Evans
- Jon Favreau, Hermanos Russo
- Fase 3

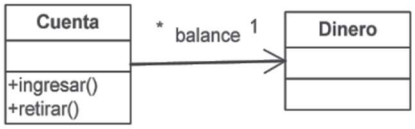
Clases y Atributos:

- Película (clase)
  - nombrePelicula
  - añoPelicula
- Actor (clase)
  - nombreActor
  - nacionalidadActor
  - papelActor
- Director (clase)
  - nombreDirector
  - nacionalidadDirector
- Fase 3 (clase)
  - nombreFase3
  - añoFase3

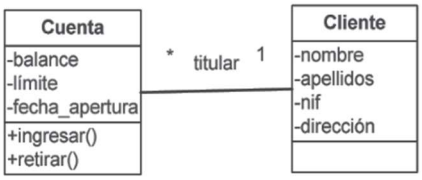
Relaciones entre las clases

Las clases, además de las jerarquías crean otro tipo de relaciones entre ellas en función del problema. Estas relaciones se reflejarán como propiedades o métodos dentro de las clases para poder realizar la relación. Las relaciones se describen adecuadamente a través UML y su diagrama de clases.

- **Asociación.** Diremos que dos (o más) clases tiene una relación de asociación cuando una de ellas tenga que requerir o utilizar alguno de los servicios (es decir, acceder a alguna de las propiedades o métodos) de las otras.



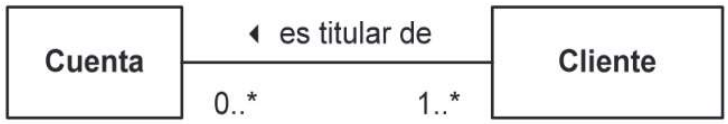
Asociación unidireccional



Asociación bidireccional

Multiplicidad	Significado
1	Uno y sólo uno
0..1	Cero o uno
N..M	Desde N hasta M
*	Cero o varios
0..*	Cero o varios
1..*	Uno o varios (al menos uno)

La asociación presenta además una multiplicidad en la que indica cómo se relacionan las clases de la asociación. Podemos ver en la tabla anterior el significado de las cardinalidades.



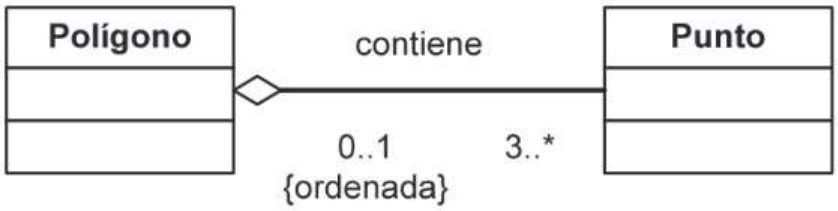
Relación opcional
Un cliente puede o no ser titular de una cuenta

Relación obligatoria
Una cuenta ha de tener un titular como mínimo

Se pueden dar relaciones de asociación entre la misma clase como se ve a continuación.

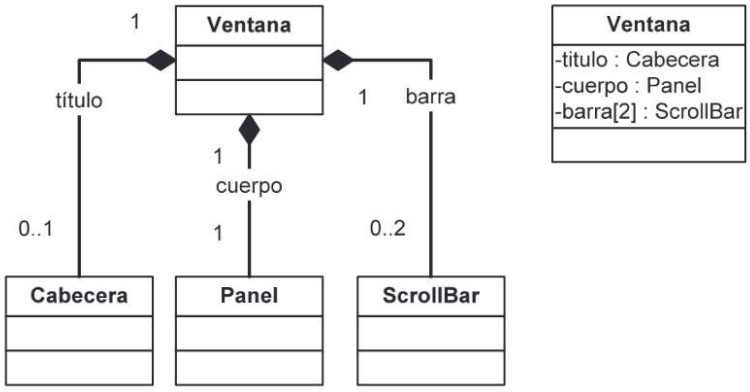


- **Agregación**. La agregación es una relación entre clases donde una clase forma parte de otra, pero no de forma exclusiva, pudiendo formar parte de otras relaciones de agregación.

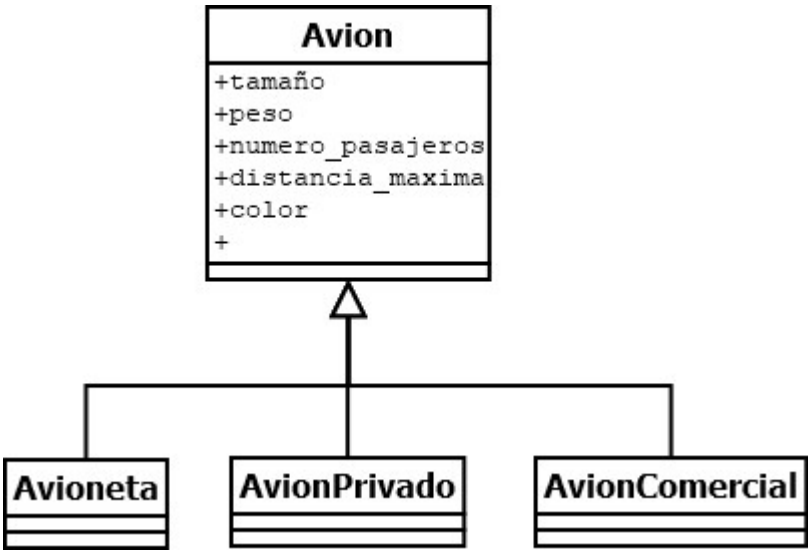


En este ejemplo punto forma parte de polígono, pero nada le impide formar parte también de otra clase, por ejemplo, círculo.

- **Composición**. La composición es una relación de agregación en la que las partes constituyentes forman parte de forma exclusiva de la relación, no pudiendo formar parte de otras relaciones de agregación o composición. Además, cada parte no tiene nada en común con las otras clases constituyentes, son disjuntas y solo existen asociadas a la relación.

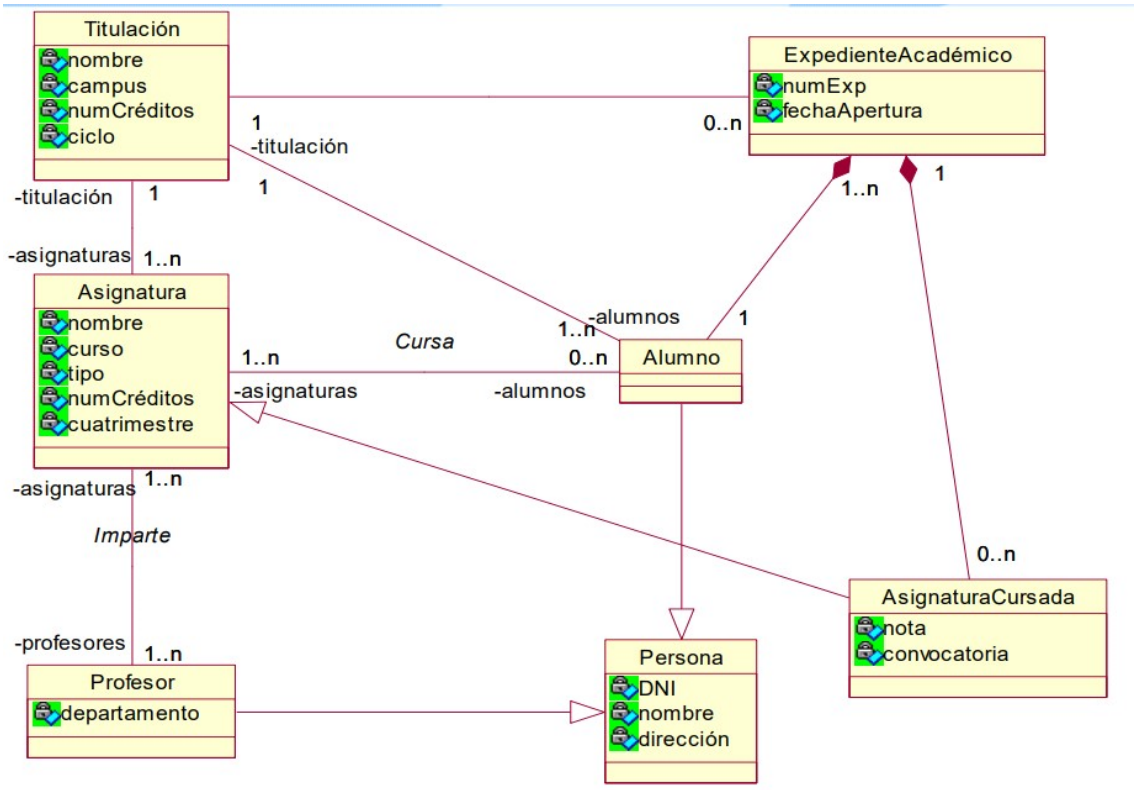


Ejemplo: Considera que queremos representar mediante un programa los aviones que operan en un aeropuerto. Crea un esquema análogo para aviones. Define cuáles podrían ser las clases y cuáles podrían ser algunos objetos de una clase.



Ejemplo: Se desea desarrollar una aplicación para la gestión académica de una universidad. Especificaciones: El sistema debe ser capaz de gestionar todos los expedientes académicos de los alumnos dando la posibilidad de realizar las operaciones típicas de altas, bajas, modificaciones y consultas de los datos del mismo. Debemos dar un número de expediente único en el sistema y una fecha de apertura. La información mínima que se debe guardar de un alumno son el DNI, nombre, dirección, la titulación en la que está matriculado, así como las asignaturas que está cursando actualmente. También debemos almacenar su historial académico, donde deben aparecer todas las asignaturas cursadas y sus respectivas notas y convocatoria.

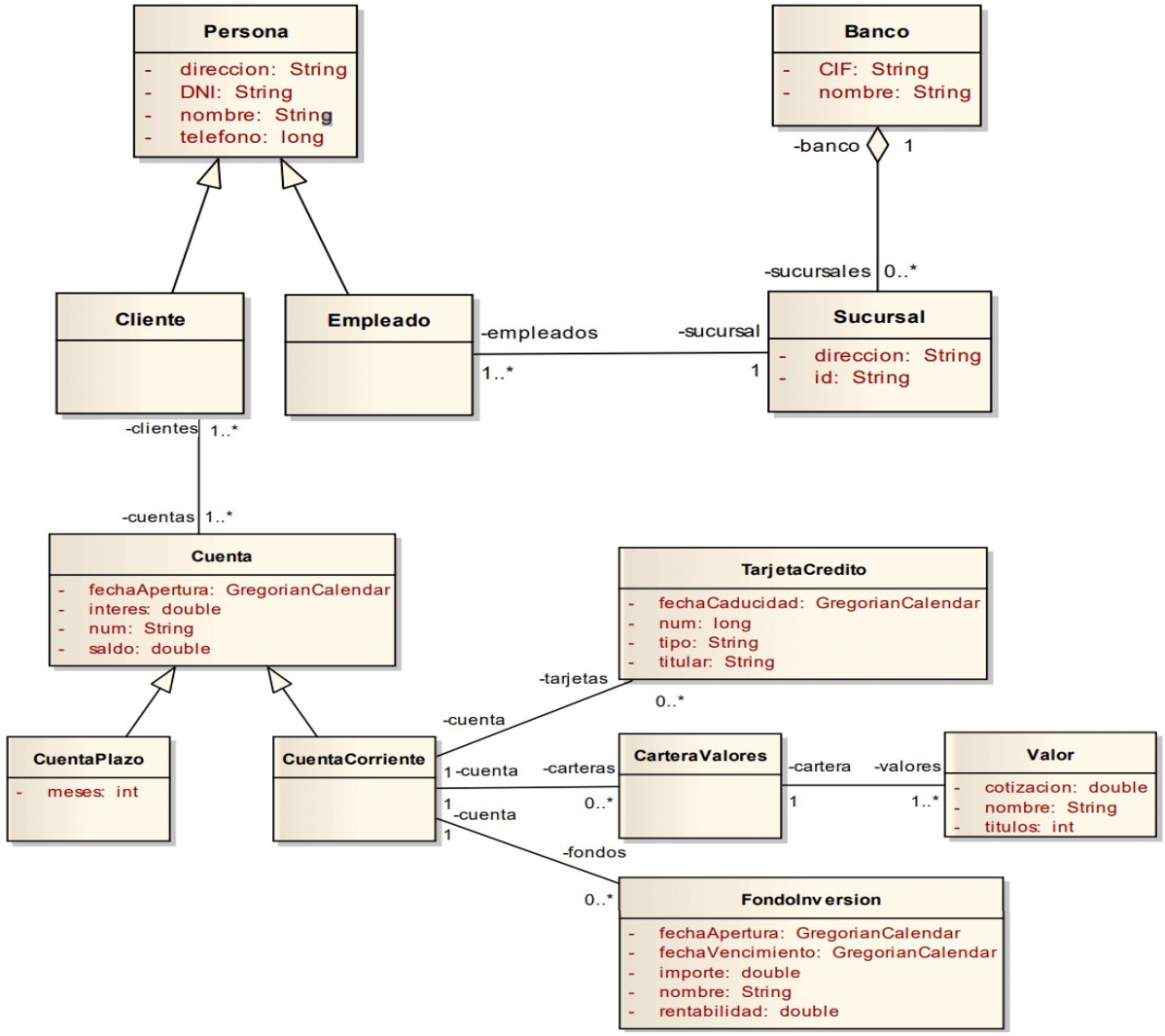
Se debe realizar la gestión de las distintas titulaciones que existen en la universidad teniendo en cuenta que una titulación sólo se da en un campus determinado y los datos que podemos consultar son el nombre, el número de créditos, si es de primer o segundo ciclo, etc. Se tienen que gestionar las asignaturas que se imparten en una titulación, teniendo en cuenta que una asignatura solo se puede dar en un único curso. Algunos de los datos que se pueden consultar de una asignatura son: el nombre, número de créditos, cuatrimestre en el que se imparte y su tipo (obligatoria, troncal, optativa). Se debe guardar la información de los profesores que imparten las distintas asignaturas de la titulación. Se debe almacenar como mínimo su DNI, nombre, dirección y departamento al que pertenece. También se podrá consultar las distintas asignaturas que imparte.



Ejercicio: Implementa las estructuras anterior y siguiente bajo Python

Ejemplo: Se desea desarrollar una aplicación de gestión bancaria. Especificaciones: El sistema debe ser capaz de gestionar una serie de productos asociados a los clientes del banco. Los productos que gestiona el banco son: cuentas bancarias, fondos de inversión y carteras de valores. Las cuentas deben tener: número de cuenta, fecha de apertura, saldo y tipo de interés y los datos de sus clientes. El banco tiene dos tipos de cuentas: corrientes y a plazo. Las cuentas corrientes pueden tener tarjetas de crédito asociadas. Solo éstas cuentas pueden tener el resto de productos asociados. Las cuentas a plazo deben tener el número de meses que estará abierta

De los clientes y los empleados se debe almacenar la siguiente información: DNI, nombre, dirección y teléfono. De los empleados necesitamos saber en qué sucursal trabajan. Cada sucursal tendrá un identificador y una dirección. Los fondos de inversión deben tener un nombre, importe, rentabilidad y la fecha de apertura y vencimiento. Las carteras de valores están compuestas por los valores asociados, almacenando el nombre del valor, el número de títulos y el precio de cotización. Las tarjetas de crédito deben almacenar el tipo (Visa, MasterCard, etc.), el número, el titular y la fecha de caducidad.



```
class Persona:
    def __init__(self, dni, direccion=None,
                 nombre=None, telefono=None):
        self.dni = dni
        self.direccion = direccion
        self.nombre = nombre
        self.telefono = telefono

    def __str__(self):
        return f"Dº {self.nombre} ({self.direccion})"


class Cliente(Persona):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        # La relación original es N a N, pero no lo hemos visto
        # todavía, un cliente N cuentas no al revés
        self.cuentas = []

    def add_cuenta(self, cuenta):
        if isinstance(cuenta, Cuenta):
            self.cuentas.append(cuenta)

    def get_cuenta(self, identificador):
        for index, cuenta in enumerate(self.cuentas):
            if cuenta.num == identificador:
                return self.cuentas[index]
        return None

    def print_cuentas(self):
        str_ret = ""
        for cuenta in self.cuentas:
            str_ret += str(cuenta) + "\n"
        return str_ret


class Empleado(Persona):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)


class Sucursal:
    def __init__(self, direccion, identificador):
        self.direccion = direccion
        self.id = identificador
        self.empleados = []

    def __str__(self):
        return f"Sucursal: {self.direccion} ({self.id})"

    def add_empleado(self, empleado):
        if type(empleado) == Empleado:
            self.empleados.append(empleado)

    def get_empleado(self, identificador):
        for index, empleado in enumerate(self.empleados):
            if empleado.dni == identificador:
```

```
                return self.empleados[index]
        return None

    def print_empleados(self):
        str_ret = ""
        for empleado in self.empleados:
            str_ret += str(empleado) + "\n"
        return str_ret


class Banco:
    def __init__(self, cif=None, nombre=None):
        self.cif = cif
        self.nombre = nombre
        self.sucursales = []  # relacción

    def __str__(self):
        str_ret = ""
        for sucursal in self.sucursales:
            str_ret += "|----" + str(sucursal) + "\n"

        return f"Banco:{self.nombre}\n" + str_ret

    def add_sucursal(self, sucursal):
        if type(sucursal) == Sucursal:
            self.sucursales.append(sucursal)

    def get_sucursal(self, identificador):
        for index, sucursal in enumerate(self.sucursales):
            if sucursal.id == identificador:
                return self.sucursales[index]
        return None


class Cuenta:
    def __init__(self, num=0):
        self.fecha_apertura = ""
        self.interes = 0
        self.num = num
        self.saldo = 0

    def __str__(self):
        return f"Cuenta: {self.num} ({self.saldo})"

    def add_saldo(self, saldo):
        if saldo > 0:
            self.saldo += saldo

    def del_saldo(self, saldo):
        if saldo > 0:
            self.saldo -= saldo


class CuentaPlazo(Cuenta):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.meses = 0
```

```
    def __str__(self):
        return f"CuentaPlazo: {self.num} ({self.saldo})"

class CuentaCorriente(Cuenta):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.tarjetas = []
        self.fondos = []
        self.carteras = []

    def __str__(self):
        return f"CuentaCorriente: {self.num} ({self.saldo})"

    def add_tarjeta(self, tarjeta):
        if type(tarjeta) == TarjetaCredito:
            self.tarjetas.append(tarjeta)

    def get_tarjeta(self, identificador):
        for index, tarjeta in enumerate(self.tarjetas):
            if tarjeta.num == identificador:
                return self.tarjetas[index]
        return None

    def print_tarjetas(self):
        str_ret = ""
        for tarjeta in self.tarjetas:
            str_ret += str(tarjeta) + "\n"
        return str_ret

    def add_fondo(self, fondo):
        if type(fondo) == Fondo:
            self.fondos.append(fondo)

    def get_fondo(self, identificador):
        for index, fondo in enumerate(self.fondos):
            if fondo.nombre == identificador:
                return self.fondos[index]
        return None

    def print_fondos(self):
        str_ret = ""
        for fondo in self.fondos:
            str_ret += str(fondo) + "\n"
        return str_ret

    def add_cartera(self, cartera):
        if type(cartera) == CarteraValores:
            self.carteras.append(cartera)

    def get_cartera(self, identificador):
        for index, cartera in enumerate(self.carteras):
            if cartera.nombre == identificador:
                return self.carteras[index]
        return None

    def print_carteras(self):
```

```
        str_ret = ""
        for cartera in self.carteras:
            str_ret += str(cartera) + "\n"
        return str_ret

class TarjetaCredito:
    def __init__(self, num):
        self.fecha_caducidad = None
        self.num = num
        self.tipo = ""
        self.titular = ""

    def __str__(self):
        return f"Tarjeta Número {self.num} ({self.titular})"


class Fondo:
    def __init__(self, nombre):
        self.nombre = nombre
        self.fecha_apertira = None
        self.fecha_vencimiento = None
        self.importe = 0
        self.rentabilidad = 0

    def __str__(self):
        return f"Fondo {self.nombre} ({self.importe})"


class Valor:
    def __init__(self, nombre):
        self.nombre = nombre
        self.cotizacion = 0
        self.titulos = 0

    def __str__(self):
        return f"Valor {self.nombre} ({self.titulos})"


class CarteraValores:
    def __init__(self, num):
        self.num = num
        self.valores = []

    def __str__(self):
        return f"Cartera de Valores: {self.num}"

    def add_valor(self, valor):
        if type(valor) == Valor:
            self.valores.append(valor)

    def get_valor(self, identificador):
        for index, valor in enumerate(self.valores):
            if valor.nombre == identificador:
                return self.valores[index]
        return None

    def print_valores(self):
```

```
        str_ret = ""
        for valor in self.valores:
            str_ret += str(valor) + "\n"
        return str_ret


if __name__ == "__main__":
    cliente = Cliente(direccion="Hola", nombre="JP", dni=99)
    print(cliente)

    # Pruebas de banco y sucursales
    banco = Banco("B123456789", "Banco UNO")
    for value in range(5):
        # Creamos 5 sucursales, su id será el número
        sucursal_aux = Sucursal("Direccion " +
                                str(value), value)
        banco.add_sucursal(sucursal_aux)

    # Añadimos empleados a una sucursal
    print(banco)
    SUCURSAL_A_MODIFICAR = 3
    print(banco.get_sucursal(SUCURSAL_A_MODIFICAR))
    banco.get_sucursal(SUCURSAL_A_MODIFICAR).add_empleado(
                Empleado(nombre="Empleado 1", dni=1))
    banco.get_sucursal(SUCURSAL_A_MODIFICAR).add_empleado(
                Empleado(nombre="Empleado 2", dni=2))
    print(banco.get_sucursal(
              SUCURSAL_A_MODIFICAR).print_empleados())
    print(banco.get_sucursal(
              SUCURSAL_A_MODIFICAR).get_empleado(2))
    print()

    # Creamos cuentas en el cliente
    cuenta_aux = CuentaPlazo(num=1)
    cuenta_aux.add_saldo(100)
    cliente.add_cuenta(cuenta_aux)
    cuenta_aux = CuentaCorriente(num=2)
    cuenta_aux.add_saldo(1000)
    cliente.add_cuenta(cuenta_aux)
    print(cliente.print_cuentas())
    print(cuenta_aux)
    cuenta_aux.add_tarjeta(TarjetaCredito(0))
    cuenta_aux.add_tarjeta(TarjetaCredito(1))
    cuenta_aux.add_fondo(Fondo("Nombre 1"))
    cuenta_aux.add_fondo(Fondo("nombre 2"))
    print(cuenta_aux.print_tarjetas())
    print(cuenta_aux.print_fondos())
    cuenta_aux.add_cartera(CarteraValores("Una cartera"))
    cv = CarteraValores("Cartera dos")
    cuenta_aux.add_cartera(cv)
    print(cuenta_aux.print_carteras())
    cv.add_valor(Valor("val 1"))
    cv.add_valor(Valor("val 2"))
    print(cv.print_valores())
```

Sobrecarga

Una de las restricciones de los primeros lenguajes estructurados (fuertemente tipados) es que las funciones o procedimientos solo podían esperar el argumento de un tipo específico. De tal manera que, si una función esperaba como primer parámetro un entero, si se le pasaba cualquier otro tipo de parámetro generaba un error de ejecución.

Para solucionar este problema se definió en la POO la sobrecarga, o la creación de métodos que se nombran igual, pero difieren en la firma (número y tipos de parámetros), de tal manera que se determinará en tiempo de ejecución (enlace dinámico o unión tardía) al método a llamar en función de los tipos de los parámetros.

```java
public class Calculos{

    public int Suma(int a, int b){
        return a + b;
    }

    public double Suma(double a, double b){
        return a + b;
    }

    public long Suma(long a, long b){
        return a + b;
    }

}
```

Los lenguajes dinámicamente tipados adolecen de la sobrecarga al admitir sus variables y parámetros cualquier tipo.

Polimorfismo y Enlace Dinámico

El polimorfismo es una relajación del sistema de tipos, de tal manera que una referencia a una clase (atributo, parámetro o declaración local o elemento de un vector) acepta direcciones de objetos de dicha clase y de sus clases derivadas (hijas, nietas, …)

Dicho de otra manera, la posibilidad que un objeto de una clase pueda ser utilizado como si fuera un objeto de la clase padre de la misma. El sistema de POO hace una “conversión” entre el objeto hijo y el objeto padre y se pueden utilizar de manera indistinta.

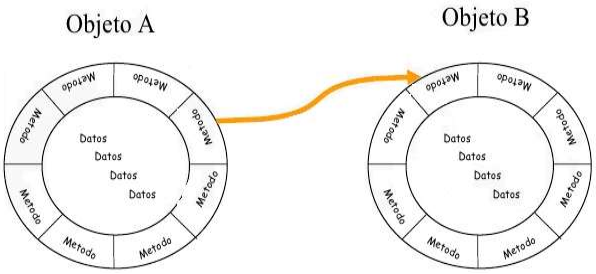


El único requerimiento para usar el polimorfismo es que hagamos uso de los métodos heredados exclusivamente para tratar los objetos hijos.

En este ejemplo, cualquier método que espere como parámetro una clase vehículo, podrá aceptar también cualquiera de sus hijas (coche, moto y bus) siempre que solo haga uso del interfaz (conjunto de métodos definidos en una clase) de la clase padre.

Mensajes

En la POO se denomina mensaje al uso que un objeto hace de un método de un segundo objeto facilitándole los parámetros esperados, y el primer objeto recoge los resultados. El símil con la programación estructurada es la llamada a una función o procedimiento.



El mecanismo de mensajes es la base de la comunicación de datos entre los objetos y de realización de tareas. Para que una aplicación funcione los objetos se tienen que intercambiar entre sí mensajes, haciendo que los objetos reaccionen a esos mensajes y efectúen las funciones para las que han sido programados.

Lenguajes

Un lenguaje es orientado a objetos si cumple entre otras características con lo siguiente:

- Soporta objetos que son abstracciones de datos con una interfaz de operaciones con nombre y un estado local oculto.
- Los objetos tienen un tipo asociado (la clase).
- Los tipos (clases) pueden heredar atributos de los supertipos (superclases)

La relación de lenguajes orientados a objetos es inmensa, podemos destacar C++, Java y Python como algunos exponentes de los mismos, pero en la siguiente dirección encontraremos una lista más amplia.

https://en.wikipedia.org/wiki/List_of_object-oriented_programming_languages

Ejercicios

1. Representa mediante un diagrama de clases la siguiente especificación relacionada con los alquileres de cámaras en una tienda de fotografía.
   - La tienda alquila cámaras fotográficas analógicas.
   - Las cámaras se caracterizan por su marca, modelo y soporte flash (si, no).
   - Cada cámara es compatible con uno o más tipos de películas.
   - Las películas se caracterizan por su marca, nombre, sensibilidad ISO (50, 100, 200, 400, 800, 1600) y formato (35 mm, 110 mm, 120 mm).
   - Para cada marca con la que trabaja la tienda se conoce la dirección del servicio de reparación más cercano.
   - La tienda dispone de varios ítems de cada modelo de cámara.
   - Cada ítem tiene una pegatina con una referencia, y puede estar en la tienda, alquilado, con retraso o en reparación.
   - Los clientes pueden tener un máximo de 1 cámara en alquiler.
2. Representa mediante un diagrama de clases la siguiente especificación sobre las personas que participan en una película
   - De cada película se almacena el título, la sinopsis, el año, el género al que pertenece (drama, comedia, acción, terror, romance, aventura, scifi) y el país.
   - Sobre las personas que participan en la película es necesario conocer el nombre, los apellidos, la fecha de nacimiento y la nacionalidad.
   - Una persona puede participar en una película como actor, director, productor o guionista.
   - Una película tiene al menos un director y un guionista.
   - Una persona se considera actor si ha actuado al menos en una película.
   - Una película puede tener asociados varios trailers que son editados por una o más personas. No puede existir el trailer de una película hasta que existe la película.
3. Representa mediante un diagrama de clases la siguiente especificación relacionada con un sistema para gestionar series:
   - Las series se caracterizan por su título, año de inicio, sinopsis y género al que pertenece (acción, aventura, animación, comedia, documental, drama, horror, musical, romance, ciencia ficción)
   - Las series se organizan en temporadas ordenadas que tienen una fecha de producción y una fecha de estreno de televisión a nivel mundial.
   - Cada temporada está a su vez formada por capítulos ordenados que tienen un título, una duración y una sinopsis.
   - Los usuarios se caracterizan por su nombre y apellidos, dirección de correo electrónico y fecha de nacimiento.
   - Si un usuario ha visto algún capítulo de una temporada el sistema la marca como empezada, si ha visto todos los capítulos de la temporada la marca como vista y un usuario en cualquier momento puede dar una temporada como cancelada, es decir, indicar que la deja de seguir.
   - Además, el sistema registra qué capítulos ha visto el usuario.
4. Representa mediante un diagrama de clases la siguiente especificación sobre una empresa:
   - Una aplicación necesita almacenar información sobre empresas, sus empleados y sus clientes.
   - Ambos se caracterizan por su nombre y edad.
   - Los empleados tienen un sueldo bruto, los empleados que son directivos tienen una categoría, así como un conjunto de empleados subordinados.
   - De los clientes además se necesita conocer su teléfono de contacto.
   - La aplicación necesita mostrar los datos de empleados y clientes.
5. Construir un diagrama de clases para un hospital con un conjunto de pacientes y un conjunto de empleados sanitarios (médicos y enfermeros) siguiendo las siguientes especificaciones:
   - Un paciente es atendido por uno o más médicos y es asistido por un grupo de enfermeros
   - Cada paciente se describe por su número de historia clínica, su nombre y dirección

- En la base de datos se mantiene información del personal sanitario referente a su número de empleado, nombre y tipo. Para los médicos hay que indicar además su especialidad
- Para cada paciente se mantiene un registro de los análisis realizados identificados por un número de referencia, además se indica el tipo de análisis, la fecha en la que se realizó, el médico que solicitó el análisis, él o los enfermeros que realizaron la prueba y los resultados que se obtuvieron en la misma.

Ejercicio (Repaso)

Introducción

Implemente la funcionalidad básica de la tienda: gestionar el almacén, gestionar los clientes y realizar ventas.

Descripción del proyecto

La práctica a desarrollar consistirá en diseñar e implementar completamente el sistema de gestión de una tienda virtual. Se deberá realizar un diagrama de clases adaptado a la funcionalidad requerida en esta práctica e implementar la funcionalidad posteriormente. Se valorará un diseño que aplique adecuadamente los principios de orientación a objetos (jerarquías de clase, modularidad, etc.)

El almacenamiento se realizará en Arrays.

El sistema a implementar consistirá en una aplicación que simulará el comportamiento de una tienda online que venderá distintos tipos de artículos (en el ejemplo a implementar libros, discos y películas

Existirán dos posibles perfiles de usuario:

- Un perfil administrador que se encargará de gestionar el almacén.
- Y un perfil cliente que será el que realice las compras en la tienda.

Las posibles acciones para el administrador serán, al menos, las siguientes:

- Mostrar el inventario completo de la tienda/almacén, ordenados por id de artículo
- Mostrar el inventario de un tipo de artículo concreto ordenado por descripción de artículo
- Realizar búsquedas de artículos
- Aumentar el stock de un artículo
- Mostrar el listado de clientes de la tienda

Por su parte, las acciones para los clientes serán, al menos:

- Registrarse como usuario en la tienda.
- Aumentar su saldo para compras.
- Realizar búsquedas de artículos concretos (queda a disposición del alumno el definir los mecanismos de búsqueda que proporcionará la aplicación).
- Añadir y eliminar artículos de su carrito de la compra.
- Comprar los artículos que ha añadido al carrito (si no hay saldo suficiente a la hora de realizar la compra ésta no será posible)
- Visualizar su histórico de compras entre dos fechas dadas

En la tienda de prueba que se implementará se gestionarán al menos tres posibles tipos de artículos: libros, discos y películas. Cada uno de ellos tendrá los siguientes atributos:

- Libro: id interno, título, autor, editorial.
- Disco: id interno, título, intérprete, año de publicación.
- Película: id interno, título, género, director.