ALGORITMOS Y ESTRUCTURAS DE DATOS
SEMANA Nº 1

OBJETIVOS DE LA PRIMER SEMANA

Clase teórica
- Presentación de la materia, condiciones de cursado, regularización, examen, etc.
- Objetivos de la materia.
- Resolución de problemas (Análisis del problema, Diseño del algoritmo y resolución del problema por computadora).
- Análisis y programación orientada a objetos.

Clase práctica
- Ejercicios de identificación de entidades.
- Repaso de matemática discreta (diagramas de flujo).

CLASE TEORICA

OBJETIVOS DE LA MATERIA

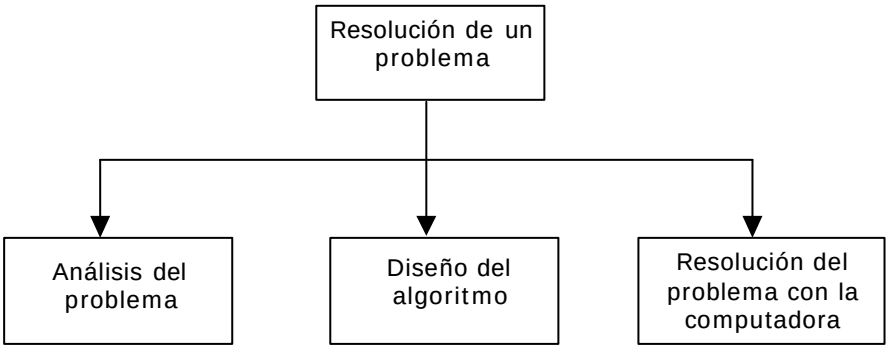
- Desarrollar la capacidad de razonamiento y lógica necesarias para Programar.
- Abordar problemas reales, analizarlos, definir la estrategia de su resolución, explicitar los algoritmos necesarios y codificarlos en un lenguaje de programación.
- Lograr entrenamiento en:
    1. Análisis del problema
    2. Diseño de la solución
    3. Programación de la solución (traducción a un lenguaje de programación).

RESOLUCION DE PROBLEMAS

La principal razón para que las personas aprendan a programar en general y los lenguajes de programación en particular es utilizar la computadora como una herramienta para la resolución de un problema se puede dividir en tres partes importantes:

1. Análisis del problema
2. Diseño o desarrollo del algoritmo
4. Resolución del algoritmo en la computadora.

El primer paso, análisis del problema, requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. Una vez analizado el problema, se debe desarrollar el algoritmo, procedimiento paso a paso para solucionar el problema dado. Por último, para resolver el algoritmo mediante una computadora se necesita codificar el algoritmo en un lenguaje de programación, convertir el algoritmo en programa, ejecutarlo y comprobar que el programa soluciona verdaderamente el problema. Las partes del proceso de resolución de un problema mediante computadora se ve en el gráfico.



1. ANALISIS DEL PROBLEMA

El propósito del análisis de un problema es ayudar al programador para llegar a una cierta comprensión de la naturaleza del problema. El problema debe estar bien definido si se desea llegar a una solución satisfactoria.

Para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas con detalle. Una buena definición del problema, junto con una descripción detallada de las especificaciones de entrada y salida, son los requisitos más importantes para llegar a una solución eficaz.

El análisis del problema exige:

1. Elección de la forma de encarar el problema y su resolución a través de un método adecuado, como por ejemplo: estructurado, orientado a objetos, etc.
2. Una vez elegida la forma de trabajo realizar una lectura previa del problema a fin de obtener una idea general de lo que se solicita.
3. La segunda lectura deberá servir para responder a las preguntas: ¿Qué información debe proporcionar la resolución del problema? ¿Qué datos se necesitan para resolver el problema?

El paso 1 es obligado ya que todos los precedentes dependen de esta elección. En nuestra materia veremos el análisis y la programación desde el punto de vista orientado a objetos.

La respuesta a la primera pregunta indicará los resultados deseados o las *salidas del problema.* La respuesta a la segunda indicará qué datos se proporcionan o las *entradas del problema.*

Por ejemplo:

PROBLEMA

Leer el radio de un circulo y calcular e imprimir su superficie y longitud.

ANALISIS

Para resolver este problema utilizando la filosofía de objetos podríamos seguir los siguientes pasos:

1. Identificar el objeto: identificar la entidad que engloba al problema, en nuestro ejemplo es el círculo.

2. Identificar sus características (atributos) : podemos decir que todo círculo tiene como característica esencial el radio, además, nuestro problema en particular necesitará el valor de la superficie y de la longitud, que son datos calculables a partir del radio y que pueden figurar como atributos o no dependiendo si se quiere guardar su estado en el objeto. Como nuestro enunciado no nos restringe la forma de implementarlo vamos a definir como características o atributos: el radio, la superficie y la longitud.

3. Identificar sus responsabilidades (métodos) : las responsabilidades es algo que la entidad hace, generalmente se traducen en operaciones (métodos) que podemos realizar con los atributos, por ejemplo, podemos enumerar:

Inicializar el radio: Inicializar en 0, por ejemplo, el radio.
Inicializar la superficie: Inicializar en 0, por ejemplo, la superficie.
Inicializar la longitud: Inicializar en 0, por ejemplo, la longitud.
Ingresar el radio: Se ingresa solo el valor del radio ya que el resto de los atributos se calculan a partir de este.
Mostrar el radio: Mostrar el valor del radio.
Mostrar la superficie: Mostrar el valor de la superficie.
Mostrar la longitud: Mostrar el valor de la longitud.
Calcular la superficie: formula que devuelve la superficie
Calcular la longitud: formula que devuelve la longitud.

2. DISEÑO DEL ALGORITMO

El análisis del problema nos resuelve el problema de identificar las entidades (objetos), junto con sus características (atríbutos) y sus comportamientos (métodos), pero a partir de allí hay que comenzar a desarrollar el cuerpo de los métodos, para ello se necesitará un algoritmo.

El significado de algoritmo es similar al de receta, método, técnica, procedimiento o rutina, y se define como “Un conjunto finito de reglas diseñadas para crear una secuencia de operaciones para resolver un tipo específico de problemas.

También se define algoritmo como una estrategia que exige precisión en las instrucciones (descripción de los pasos a seguir). Donde cada instrucción debe ser:

- Clara y precisa para que no surja ninguna duda en cuanto a su ejecución.
- Sencilla para que no surja ninguna duda en su comprensión y pueda ser ejecutada sin dificultad.

Una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la máquina constituyen el *algoritmo*

Los algoritmos se pueden representar a través de herramientas tales como: diagramas de flujo, pseudocódigos o diagramas N.-S.

3. RESOLUCION DEL PROBLEMA POR COMPUTADORA

Una vez que las entidades y el algoritmo están diseñados y representados gráficamente se debe pasar a la fase de resolución práctica del problema con la computadora.

Esta fase se descompone a su vez en las siguientes subfases:

1. Codificación del algoritmo en un programa.
2. Ejecución del programa.
3. Comprobación del programa.

En el diseño del algoritmo, éste se describe en una herramienta de programación tal como un diagrama de flujo o pseudo código. Sin embargo, el programa que implementa el algoritmo debe ser escrito en un lenguaje de programación y siguiendo las reglas gramaticales o sintaxis del mismo. La fase de conversión del algoritmo en un lenguaje de programación se denomina *codificación,* ya que el algoritmo en lenguaje de programación se denomina código.

ANALISIS Y PROGRAMACION ORIENTADA A OBJETOS

1. INTRODUCCIÓN A LA PROGRAMACIÓN CENTRADA EN EL CONCEPTO (ORIENTADA A OBJETOS)

La orientación a objetos es una forma natural de pensar en relación con el mundo y de escribir programas de computación. Mire a su alrededor. Por todas partes: *¡objetos!* Personas, animales, plantas, automóviles, aviones, edificios, cortadoras de pastos, computadoras y demás. Cada una de ellas tiene ciertas características y se comporta de una manera determinada. Si las conocemos, es porque tenemos el ***concepto de lo que son***. Conceptos persona, objetos persona. Los seres humanos **pensamos en términos de objetos**. Tenemos la capacidad maravillosa de la ***abstracción***, que nos permite ver una imagen en pantalla como personas, aviones, árboles y montañas, en vez de puntos individuales de color.

Todos estos objetos tienen algunas cosas en común. **Todos tienen atributos**, como tamaño, forma, color, peso y demás. Todos ellos exhiben **algún comportamiento.** Un automóvil acelera, frena, gira, etcétera. El objeto persona habla, ríe, estudia, baila, canta ...

Los seres humanos aprenden lo relacionado con los objetos estudiando sus atributos y observando su comportamiento. Objetos diferentes pueden tener muchos atributos iguales y mostrar comportamientos similares. Se pueden hacer comparaciones, por ejemplo, entre bebes y adultos, entre personas y chimpancés. Automóviles, camiones, pequeños carros rojos y patines tienen mucho en común.

La ***programación orientada a objetos*** *(POO)* **hace modelos de los objetos del mundo real** mediante sus contrapartes en software. Aprovecha las relaciones de clase, donde objetos de una cierta clase, como la clase de vehículos, tienen las mismas características. Aprovecha las relaciones de *herencia,* donde clases recién creadas de objetos se derivan heredando características de clases existentes, pero también poseyendo características propias de ellos mismos. Los bebes tienen muchas características de sus padres, pero ocasionalmente padres de baja estatura tienen hijos altos.

La programación orientada a objetos nos proporciona una forma más **natural e intuitiva** de observar el proceso de programación, es decir *haciendo modelos* de objetos del mundo real, de sus atributos y de sus comportamientos. POO también hace modelos de la comunicación entre los objetos. De la misma forma que las personas se envían *mensajes* uno al otro los objetos también se comunican mediante mensajes.

La **POO *encapsula* datos (atributos) y funciones (comportamiento)** en paquetes llamados ***objetos***; los datos y las funciones de un objeto están muy unidos. Los objetos tienen la propiedad de *ocultar* la *información.* Esto significa que aunque los objetos puedan saber cómo comunicarse unos con otros mediante *interfaces* bien definidas, a los objetos por lo regular no se les está permitido saber cómo funcionan otros objetos. Los detalles de puesta en práctica quedan ocultos dentro de los objetos mismos. (*Casi estamos diciendo que existe entre ellos el respeto a la intimidad ...*) A esto se le llama ***Encapsulamiento***.

2. QUE ES LA PROGRAMACION ORIENTADA A OBJETOS

Es una técnica o estilo de programación que:

- Utiliza objetos como bloque esencial de construcción. Los programas se organizan como colecciones de **objetos** que colaboran entre sí enviándose mensajes. Solo se dispone de "***objetos que colaboran entre sí***". Por lo tanto un programa orientado a objetos viene definido por la ecuación:

  Objetos + Mensajes = Programa

- Facilita la creación de software de calidad debido a que potencia el mantenimiento, la extensibilidad y la reutilización del software generado bajo este paradigma.
- Trata de amoldarse al modo de pensar del hombre y no al de la máquina.

3. COMPONENTES BÁSICOS

Objetos

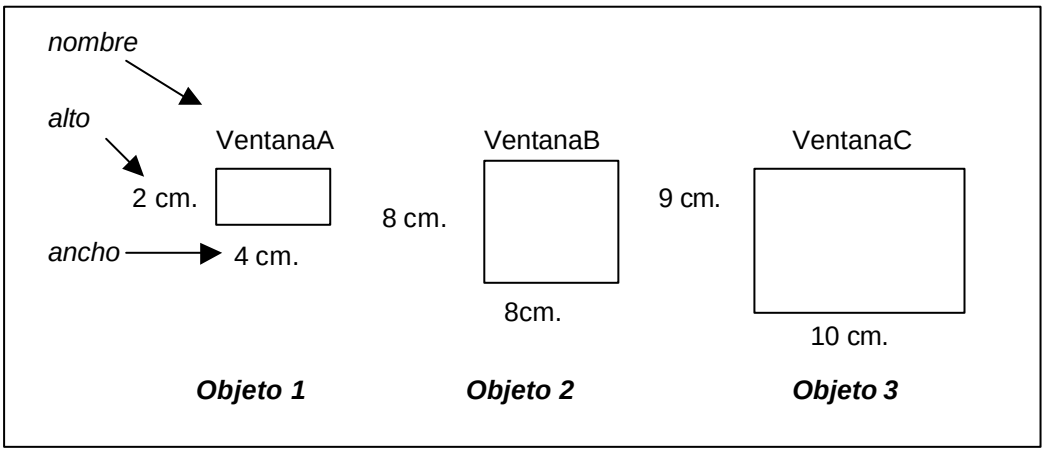
Es el elemento fundamental de la programación orientada a objetos. Es una entidad que posee atributos y métodos, los atributos definen el estado de mismo y los métodos definen su comportamiento.

Cada objeto forma parte de una organización, no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

¿ Qué son objetos en POO? La respuesta es cualquier entidad del mundo real que se pueda imaginar:

- *Objetos físicos*
  - Automóviles en una simulación de tráfico.
  - Aviones en un sistema de control de tráfico aéreo.
  - Componentes electrónicos en un programa de diseño de circuitos.
  - Animales mamíferos.
- *Elementos de interfaces gráficos de usuarios*
  - Ventanas.
  - Objetos gráficos (líneas, rectángulos, círculos).
  - Menús.
- *Estructuras de datos*
  - Vectores.
  - Listas.
  - Arboles binarios.
- *Tipos de datos definidos por el usuario*
  - Números complejos.
  - Hora del día.
  - Puntos de un plano.

Como ejemplo de objeto, podemos decir que una ventana es un objeto que puede tener como atributos: **nombre, alto, ancho, etc.** y como métodos: **crear la ventana, abrir la ventana, cerrar la ventana, mover la ventana, etc.**



Métodos

Los métodos definen e implementan el comportamiento del objeto.

Especifican la forma en que se controlan los datos de un objeto.

Un método es un conjunto de instrucciones escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Un método asociado con el objeto factura, por ejemplo, podría ser aquel que calcule el total de la factura. Otro podría transmitir la factura a un cliente. Otro podría verificar de manera periódica si la factura ha sido pagada y, en caso contrario, añadir cierta tasa de interés.

Mensajes

Un objeto solo no es muy útil. Un objeto aparece por lo general como un componente de un programa donde aparecen muchos objetos.

Mediante la interacción de estos objetos los programadores consiguen funcionalidades complejas.

Los objetos se comunican los unos con los otros mediante el **paso de mensajes**.

Un mensaje es simplemente una petición de un objeto a otro para que éste se comporte de una determinada manera, ejecutando uno de sus métodos.

Un mensaje consta de 3 partes:

- El objeto que recibe el mensaje (**receptor**),
- El nombre del método a realizar (**selector**) . Debe ser un método del receptor.
- 0 o más **argumentos** necesitados por el método.

La figura 1. Representa un objeto A (emisor) que envía un mensaje Test al objeto B (receptor).

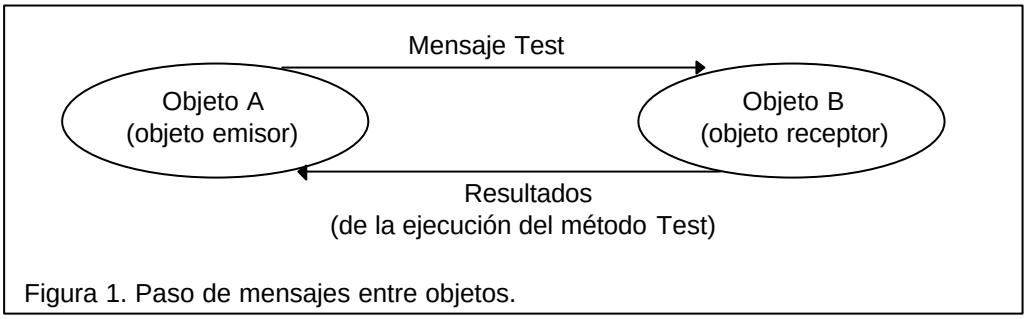


Figura 1. Paso de mensajes entre objetos.

Usando el ejemplo anterior de ventana, podríamos decir que algún objeto de Windows encargado de ejecutar aplicaciones, por ejemplo, envía un mensaje a nuestra ventana para que se abra. En este ejemplo el objeto emisor es un objeto de Windows y el objeto receptor es nuestra ventana, el mensaje es la petición de abrir la ventana y la respuesta es la ejecución del método abrir ventana.

Clases

Una clase es simplemente un modelo que se utiliza para describir uno o más objetos el mismo tipo.

Cada vez que se construye un objeto de una clase, se crea una instancia de esa clase. Por consiguiente, los objetos son instancias de clases. En general, los términos objetos e instancias de una clase se pueden utilizar indistintamente.

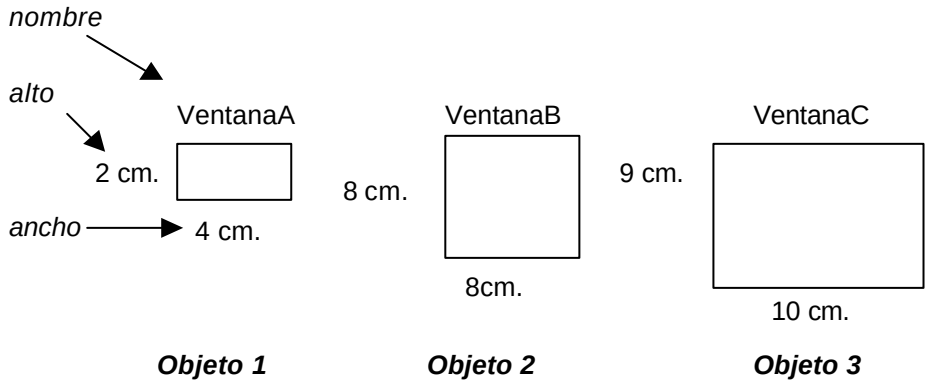
Una clase puede tener muchas instancias y cada una es un objeto independiente.

Siguiendo con el ejemplo de ventana, el modelo o clase para todas las ventanas sería:

Clase ventana

Características: nombre, alto, ancho.
Responsabilidades: crear, abrir, cerrar, cambiar tamaño.

Objetos de tipo ventana



4. CARACTERISTICAS

Abstracción

La abstracción se define como la "*extracción de las propiedades esenciales de un concepto*". Permite no preocuparse de los detalles no esenciales. Implica la identificación de los atributos y métodos de un objeto.

Es la capacidad para encapsular y aislar la información de diseño y ejecución.

Encapsulamiento

Es el mecanismo por el cual los objetos protegen sus variables bajo la custodia de sus métodos. De esta manera el usuario puede ver los objetos como cajas negras que proporcionan servicios.

Toda la información relacionada con un objeto determinado está agrupada de alguna manera, pero el objeto en sí es como una caja negra cuya estructura interna permanece oculta, tanto para el usuario como para otros objetos diferentes, aunque formen parte de la misma jerarquía. La información contenida en el objeto será accesible sólo a través de la ejecución de los métodos adecuados.

Beneficios de la encapsulación:

Modularidad: el código fuente de un objeto se puede escribir y mantener independientemente del código fuente de otros objetos. También hace que pueda reutilizarse.

Ocultación de la información: Un objeto tiene una interfaz pública que otros objetos pueden usar para comunicarse con él. El objeto puede mantener información privada y métodos que pueden cambiar sin que esto afecte a otros objetos que dependen de él.

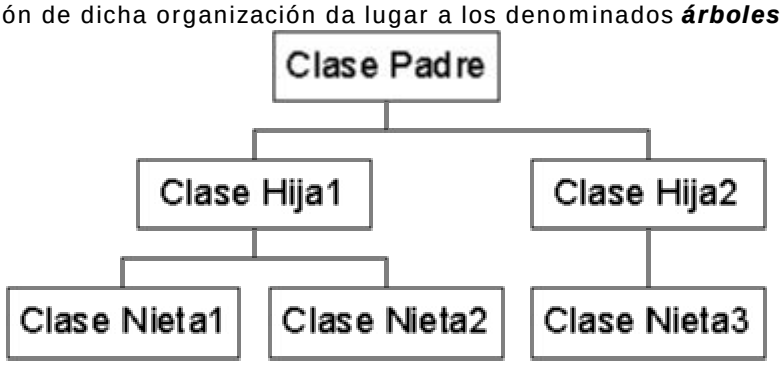
Herencia

Nosotros vemos de manera natural nuestro mundo como objetos que se relacionan entre sí de una manera jerárquica. Por ejemplo, un perro es un mamífero, y los mamíferos son animales, y los animales seres vivos...

La herencia es la propiedad que permite a los objetos construirse a partir de otros objetos. El concepto de herencia está presente en nuestras vidas diarias donde las clases se dividen en subclases. Así por ejemplo, las clases de animales se dividen en mamíferos, anfibios, insectos, pájaros, etc. La clase de vehículos se divide en automóviles, autobuses, camiones, motocicletas, etc.

El principio de este tipo de división es que cada subclase comparte características comunes con la clase de la que se deriva. Los automóviles, camiones autobuses y motocicletas -que pertenecen a la clase vehículo- tienen ruedas y un motor; son las características de vehículos. Además de las características compartidas con otros miembros de la clase, cada subclase tiene sus propias características particulares: autobuses, por ejemplo, tienen un gran número de asientos, un aparato de televisión para los viajeros, mientras que las motocicletas tienen dos ruedas, un manillar y un asiento doble.

Del mismo modo, las distintas clases de un programa se organizan mediante la ***jerarquía de clases***. La representación de dicha organización da lugar a los denominados ***árboles de herencia***:



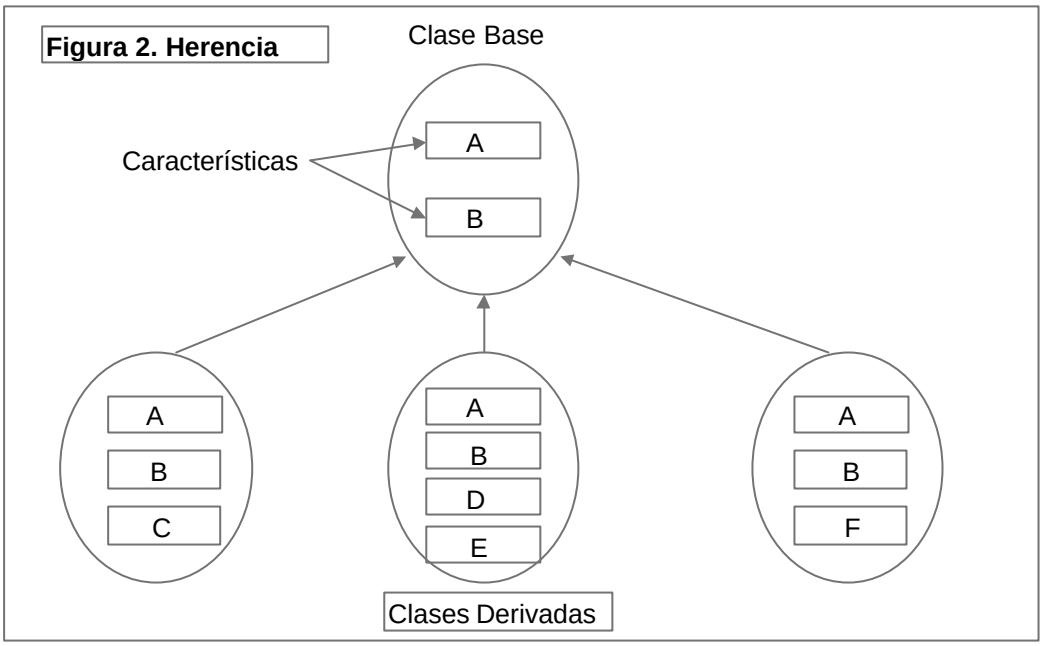
Mediante la *herencia* una *clase hija* puede tomar determinadas propiedades de una *clase padre*. Así se simplifican los diseños y se evita la duplicación de código al no tener que volver a codificar métodos ya implementados.

Al acto de tomar propiedades de una clase padre se denomina **heredar**.

La idea de herencia se muestra en la figura 2. Observen en la figura que las características A y B que pertenecen a la clase base, también son comunes a todas las clases derivadas, y a su vez estas clases derivadas tienen sus propias características.

La herencia impone una relación jerárquica entre clases en la cual una clase *hija* hereda de su clase *padre*. Si una clase sólo puede recibir características de otra clase base, la herencia se denomina *herencia simple*.

Si una clase recibe propiedades de más de una clase base, la herencia se denomina *herencia múltiple*.



Como ejemplo, supongamos que tenemos que registrar los datos de alumnos y profesores, para ello vamos a analizar el problema y refinar la solución en varios pasos:

Primer paso: Identificamos las características esenciales para cada entidad.

Alumno	**Profesor**
Nombre	Nombre
Domicilio	Domicilio
Telefono	Telefono
Legajo	Legajo
Carrera	Titulos
Materias	Cursos

Segundo paso: Si observamos, los dos tienen características en común: Nombre, Domicilio y Telefono que se corresponden con los datos personales de cualquier persona, y datos particulares para el alumno y el docente. Entonces podríamos escribir los atributos de la siguiente forma:

Alumno	**Profesor**
Datos personales	Datos personales
Datos del alumno	Datos del profesor

Tercer paso: Si agrupamos los datos personales en una clase llamada Persona con los datos comunes a las dos entidades, formaremos lo siguiente:

Persona
Datos Personales

Alumno	**Profesor**
Es una persona	Es una persona
(porque tiene datos de persona)	(porque tiene datos de persona)
Tiene datos del alumno	Tiene datos del profesor

Cuarto paso: Al completar las características de todas las clases, vamos a observar que hemos ahorrado características, ya que en el primer paso había características que se repetían en las dos entidades y ahora hay tres entidades en donde cada una tiene las características que le corresponden sin repetición, pero alumno y profesor van a heredar de persona las características que le correspondan.

Persona
Nombre
Domicilio
Telefono

Alumno	**Profesor**
Hereda los datos de persona	Hereda los datos de persona
Legajo	Legajo
Carrera	Titulos
Materias	Cursos

Polimorfismo

Dentro del ámbito de la programación orientada a objetos, podríamos definir al polimorfismo formalmente diciendo: Dos objetos son polimorfitos, respecto de un conjunto de mensajes, si ambos son capaces de responderlos. En otras palabras podemos decir que si dos objetos son capaces de responder al mismo mensaje, aunque sea haciendo cosas diferentes, entonces son polimorfitos respecto de ese mensaje.

Por ejemplo, supongamos que en un programa debemos trabajar con figuras geométricas y construimos algunos objetos que son capaces de representar un rectángulo, un triangulo y un circulo. Si nos interesa que todos puedan dibujarse en la pantalla, podemos dotar a cada uno de un método llamado "dibujar" que se encargue de realizar esta tarea y entonces vamos a poder dibujar tanto un circulo, como un cuadrado o un triangulo simplemente enviándole al mensaje "dibujar" al objeto que queremos mostrar en la pantalla. En otras palabras, vamos a poder hacer X.dibujar() sin importar si X es un cuadrado un circulo o un triangulo, porque al fin y al cabo, todos pueden responder a ese mensaje. En ese caso decimos que tanto los objetos "circulo", como los "triangulo" y los "cuadrado" son polimorfitos respecto del mensaje dibujar. Es importante notar que no es lo mismo dibujar una figura que otra, seguramente, cada figura implementa esa método de modo diferente.

Otro ejemplo: supongamos que en un sistema de un banco tenemos objetos que representas cuentas corrientes y cajas de ahorro. Si bien son diferentes, en ambas pueden hacerse depósitos y extracciones, entonces podemos hacer que cada objeto implemente un método "extraer" y uno "depositar" para que puedan responder a esos mismos mensajes. Sin duda, van a estar implementados de modo diferente porque no se hacen los mismos pasos para extraer o depositar dinero de una cuenta corriente que de una caja de ahorro, pero ambos objetos van a poder responder e esos mensajes y entonces serán polimorfitos entre si respecto del conjunto de mensajes formado por "depositar" y "extraer".

La gran ventaja es que ahora podemos hacer X.depositar() y confiar en que se realizara un depósito en la cuenta que corresponde sin que tengamos que saber a priori si X es una cuenta corriente o una caja de ahorro.

Otro ejemplo: un usuario de un sistema tiene que imprimir un listado de sus clientes, el puede elegir imprimirlo en el monitor, en la impresora, en un archivo en disco o en una terminal remota. De esta forma el mensaje es imprimir el listado pero la respuesta la dará el objeto que el usuario desee, el monitor, la impresora, el archivo o la terminal remota. Cada uno responderá con métodos (comportamientos) diferentes.

Mas adelante, al analizar en profundidad el mecanismo de herencia, vamos a ver que Java posee una forma bastante simple que permite resolver el polimorfismo en tiempo de ejecución, o sea, decidir a que objeto hay que enviarle el mensaje en el momento en que el programa se ejecuta.

5. Resumen

Como vimos anteriormente, el análisis de un problema es el paso más importante para resolver un problema, existen varias formas de plantear las soluciones, pero hay dos grandes formas que se están utilizando hoy en día, que son el análisis estructurado y el orientado a objetos, el primero se está dejando de usar, sin embargo aún quedan muchos sistemas resueltos bajo esta óptica que hay que mantener o actualizar. Y la segunda forma es la que nosotros vamos a usar ya que es la tendencia del mercado, es la que mejor rendimiento tiene, se está usando desde hace varios años y es la que tiene mayor soporte en cuanto a tecnologías de análisis, diseño y desarrollo de software.

A continuación vamos a realizar una breve comparación entre la programación estructurada y la orientada a objetos que nos servirá para entender mejor como tendremos que resolver los problemas bajo la óptica orientada a objetos.

Programación estructurada

La programación estructurada tiende a ser ***orientada a* la *acción.*** Los programadores se **concentran en escribir funciones**, que son grupos de acciones que ejecutan alguna tarea común y que se agrupan para formar programas. La **unidad fundamental de la programación es la función**. Es cierto que los datos son importantes, pero la óptica es que los datos son materia prima para las acciones que las funciones ejecutan. Semánticamente las acciones son verbos. Los ***verbos*** en una especificación de sistema ayudan al programador a determinar el conjunto de funciones que juntas funcionarán para ponerlo en marcha.

Programación orientada a objetos

La programación orientada a objetos como su nombre lo indica está ***orientada al objeto (concepto).*** El **concepto** se implementa en clases (*class*). **La clase es la unidad básica de programación.** Es una unidad que contiene los atributos y todas las funciones necesarias a su tratamiento.

Entonces cada clase contiene datos junto con un conjunto de funciones que manipula dichos datos. Los componentes de datos de una clase se llaman ***datos miembros*** o *atributos.* El comportamiento de la clase se implementa mediante ***funciones miembro***.

El foco de atención está sobre los objetos, en vez de sobre las funciones. Los ***sustantivos*** en una especificación de sistema ayudan al programador a determinar el conjunto de clases, a partir de las cuales serán creados los objetos que funcionarán conjuntamente para poner en práctica el sistema.

Es muy importante que a la hora de analizar un problema utilicemos sustantivos y no verbos.

A continuación resolveremos un problema bajo la óptica orientada a objetos:

PROBLEMA

Dado el valor de los tres lados de un triángulo, calcular el perímetro.

ANALISIS

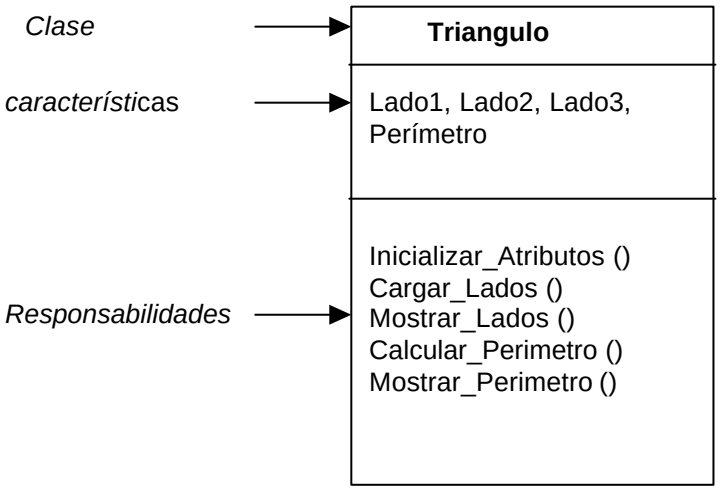
1. Identificar el objeto: identificar la entidad que engloba al problema, en nuestro ejemplo es el triángulo.

2. Identificar sus características: podemos decir que todo triángulo tiene como datos esenciales los tres lados, además, nuestro problema en particular necesitará el valor del perímetro, que es un datos calculable a partir de los tres lados y que puede figurar como atributo o no dependiendo si se quiere guardar su estado en el objeto. Como nuestro enunciado no nos restringe la forma de implementarlo vamos a definir como características los tres lados y el perímetro.

3. Identificar sus responsabilidades: las responsabilidades a elegir generalemente dependerán de las operaciones que podemos realizar con los atributos, entonces podemos enumerar:

Inicializar los atributos: Inicializar en 0, por ejemplo, los lados y el perímetro.
Ingresar los lados: Se ingresa el valor de cada lado.
Mostrar los lados: Mostrar el valor de los lados.
Calcular el perímetro: formula que devuelve el perímetro del triángulo.
Mostrar el perímetro: Mostrar el valor del perímetro.

Graficando el resultado



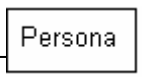
LECTURA COMPLEMENTARIA

Para lectura de los alumnos

ANÁLISIS DE LA ESTRUCTURA DE OBJETOS.

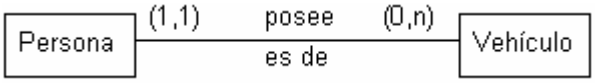
El análisis de la estructura de objetos (AEO) define las categorías de los objetos que percibimos y las formas en que los asociamos.

Objetos y Tipos de Objetos: En el análisis se trata de identificar los tipos de objeto más que los objetos individuales en un sistema. Los tipos de objetos se definen en base a la comprensión del analista de nuestro mundo.



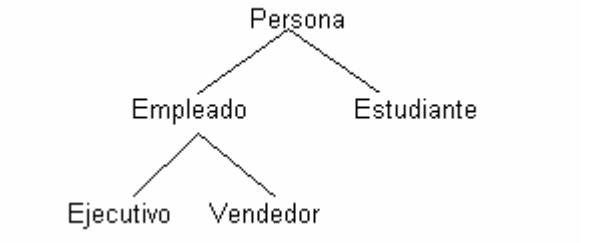
Representación para el Objeto Persona.

Asociaciones de Objetos: Es importante modelar la forma como los objetos se asocian entre sí. Además es necesario identificar el significado de la asociación y la cantidad de objetos con los que un objeto dado puede y debe asociarse (cardinalidad). Por ejemplo, un objeto del tipo persona posee cero o muchos objetos del tipo vehículo. Un objeto del tipo vehículo es de un y sólo un objeto del tipo persona.



Representación para la Asociación entre dos Tipos de Objetos.

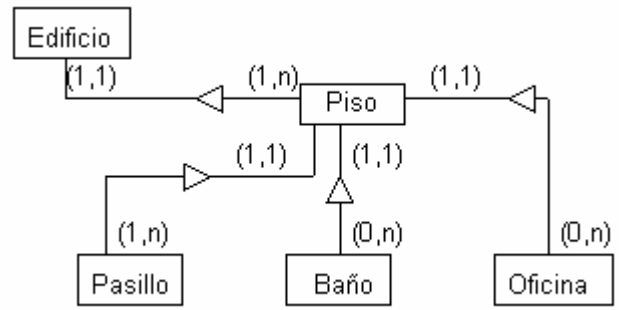
Jerarquías de Generalización: Una de las vías de sentido común por las que el hombre organiza su volumen de conocimiento es el de las jerarquías, de lo más general a lo más específico. Por ejemplo:



Representación de una Jerarquía de generalización, para el tipo de objeto Persona.

En las jerarquías se habla de subtipo o especialización de un supertipo o generalización. En el caso anterior, persona es el supertipo para Empleado y Estudiante, que son sus subtipos. Por otra parte, Empleado es el supertipo para los subtipos Ejecutivo y Vendedor. Los subtipos (niveles inferiores de la jerarquía) heredan las características de sus supertipos, además, cada instancia de un tipo de objeto lo es también de sus supertipos.

Jerarquías Compuestas: Un objeto se denomina complejo si está formado por otros. Las jerarquías Compuestas permiten realizar agregaciones de objetos. Por ejemplo: Un objeto del tipo edificio se compone de a lo menos un objeto del tipo piso. A su vez un objeto del tipo piso se compone de a lo menos un objeto del tipo pasillo, podría tener varios (o ninguno) objetos del tipo baño y oficina.



Representación de una Jerarquía Compuesta.

Diagramas de relación entre los objetos: Los tipos de objetos están relacionados con otros tipos de objeto. Por ejemplo, un empleado trabaja en una sucursal, o un cliente realiza un pedido de varios productos.

Un objeto del tipo cliente puede ordenar muchos objetos del tipo pedidos, y un objeto del tipo pedido es ordenado por un y sólo un objeto del tipo cliente. Un objeto del tipo producto está en muchos o ningún objeto del tipo pedido, mientras que un objeto del tipo pedido tiene al menos un objeto del tipo producto.

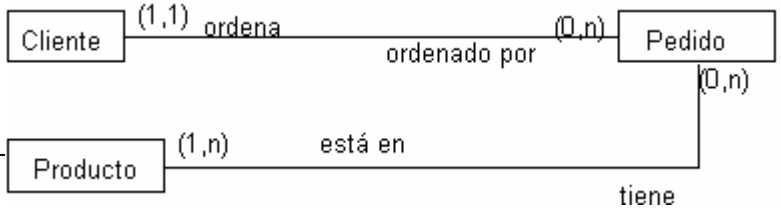


Diagrama de Relación entre objetos.

ESQUEMAS DE OBJETOS

La comprensión de un modelo suele ser más sencilla si los tipos de objetos y relaciones se presentan mediante un diagrama de relación entre objetos; los supertipos y subtipos se presentan en un diagrama de jerarquías de generalización y las estructuras compuestas en un diagrama compuesto. Sin embargo, para los usuarios más sofisticados puede ser útil presentarlo todo en un mismo diagrama, el que se denomina esquema de objetos.

MODELO DE OBJETOS

Existen una serie de principios fundamentales para comprender cómo se modela la realidad al crear un programa bajo el paradigma de la orientación a objetos. Estos principios son: la abstracción, la encapsulación, la modularidad, y el polimorfismo.

Principio de Abstracción

Mediante la abstracción la mente humana modela la realidad en forma de objetos. Para ello busca parecidos entre la realidad y la posible implementación de *objetos del programa* que simulen el funcionamiento de los *objetos reales*.

Los seres humanos no pensamos en las cosas como un conjunto de cosas menores; por ejemplo, no vemos un cuerpo humano como un conjunto de células.

Los humanos entendemos la realidad como objetos con comportamientos bien definidos.

No necesitamos conocer los detalles de por qué ni cómo funcionan las cosas; simplemente solicitamos determinadas acciones en espera de una respuesta.

Pero la abstracción humana se gestiona de una manera jerárquica, dividiendo sucesivamente sistemas complejos en conjuntos de subsistemas, para así entender más fácilmente la realidad. Esta es la forma de pensar que la orientación a objeto intenta cubrir.

Principio de Encapsulación

La encapsulación permite a los objetos elegir qué información es publicada y qué información es ocultada al resto de los objetos. Para ello los objetos suelen presentar sus métodos como interfaces públicas y sus atributos como datos privados e inaccesibles desde otros objetos.

Para permitir que otros objetos consulten o modifiquen los atributos de los objetos, las clases suelen presentar niveles de acceso para sus atributos y sus métodos. De esta manera el acceso a los datos de los objetos es controlado por el programador, evitando efectos laterales no deseados.

Con el encapsulado de los datos se consigue que las personas que utilicen un objeto sólo tengan que comprender su interfaz, olvidándose de cómo está implementada, y en definitiva, reduciendo la complejidad de utilización.

Principio de Modularidad

Mediante la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.

Esta fragmentación disminuye el grado de dificultad del problema al que da respuesta el programa, pues se afronta el problema como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

Principio de Polimorfismo

Polimorfismo quiere decir "un objeto y muchas formas". Esta propiedad permite que un objeto presente diferentes comportamientos en función del contexto en que se encuentre.
También se refiere a la posibilidad de las variables a referenciar entidades de distinto tipo.

RELACIONES ENTRE OBJETOS

Durante la ejecución de un programa, los diversos objetos que lo componen han de interactuar entre sí para lograr una serie de objetivos comunes.

Existen varios tipos de relaciones que pueden unir a los diferentes objetos, pero entre ellas destacan las relaciones de: asociación, agregación / composición, y generalización / especialización.

Relaciones de Asociación

Serían relaciones generales, en las que un objeto realiza llamadas a los servicios (métodos) de otro, interactuando de esta forma con él. Representan las relaciones con menos riqueza semántica.

Relaciones de Agregación / Composición (clientela)

Muchas veces una determinada entidad existe como conjunción de otras entidades, como un conglomerado de ellas. La orientación al objeto recoge este tipo de relaciones como dos conceptos; la agregación y la composición.

En este tipo de relaciones un *objeto componente* se integra en un *objeto compuesto*.

La diferencia entre agregación y composición es que mientras que la composición se entiende que dura durante toda la vida del objeto componedor, en la agregación no tiene por qué ser así.

Esto se puede implementar como un objeto (*objeto compuesto*) que cuenta entre sus atributos con otro objeto distinto (*objeto componente*).

Relaciones de Generalización/Especialización (herencia)

A veces sucede que dos clases tiene muchas de sus partes en común, lo que normalmente se abstrae en la creación de una tercera clase (*padre* de las dos) que reúne todas sus características comunes.

El ejemplo más extendido de este tipo de relaciones es la herencia, propiedad por la que una clase (*clase hija*) recoge aquellos métodos y atributos que una segunda clase (*clase padre*) ha especificado como "heredables".

Este tipo de relaciones es característico de la programación orientada a objetos.

En realidad, la generalización y la especialización son diferentes perspectivas del mismo concepto, la generalización es una perspectiva ascendente (*bottom-up*), mientras que la especialización es una perspectiva descendente (*top-down*).

PARTE PRACTICA

Repaso teórico

Objetivo de la materia

1. Analizar el problema
2. Diseñar una solución.
3. Programar dicha solución (traducirla al lenguaje Java)

Análisis orientado a objetos

- Básado en objetos que son entidades de la vida real que tienen características (atributos) y responsabilidades (métodos). Por ejemplo :

  *Se tiene el objeto **Alumno** de nombre **juan**, legajo **100.000** y tel: **4588987**, puede responder notificando su nombre, legajo y telefono y puede recibir cambios en su teléfono.*

- Los objetos no estan aislados, forman parte de una organización y sus atributos y métodos están directamente relacionados con el entorno donde se desarrollan, por ejemplo:

  Un alumno de la universidad tecnológica posiblemente tendrá algunos atributos distintos a un alumno de una escuela primaria o a un alumno de un curso terciario, si bien pueden tener características generales a cualquier alumno, dependiendo del contexto donde se use el alumno, necesitará más o menos características especiales.

- Los objetos se comunican a través de mensajes (petición entre los objetos). Por ejemplo:

  *Un objeto **Docente** puede solicitarle al objeto **Alumno** su legajo, el **Alumno** responderá con la ejecución del método que devuelva su legajo.*

- Los objetos se modelizan a través de clases (modelos o moldes para describir objetos). Por ejemplo:

  *El objeto **Alumno juan** pertenece a la clase **Alumno** que tiene las características y comportamientos generales a cualquier alumno, por ejemplo:*
  ***características**: Nombre, Legajo, Teléfono.*
  ***Responsabilidades**: Notificar el Nombre, Notificar el Legajo, Notificar el teléfono y Cambiar el Teléfono.*

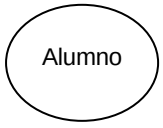
- Los objetos se pueden construir a partir de otros, reutilizar la estructura de otras clases (herencia). Por ejemplo:

  *Un **Alumno** y un **Docente** tienen las características y los comportamientos deuna persona, por lo tanto se pueden construir a partir de una clase genérica **Persona**.*

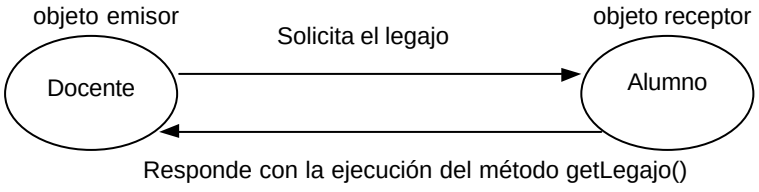
Convenciones de trabajo

1. Notación gráfica

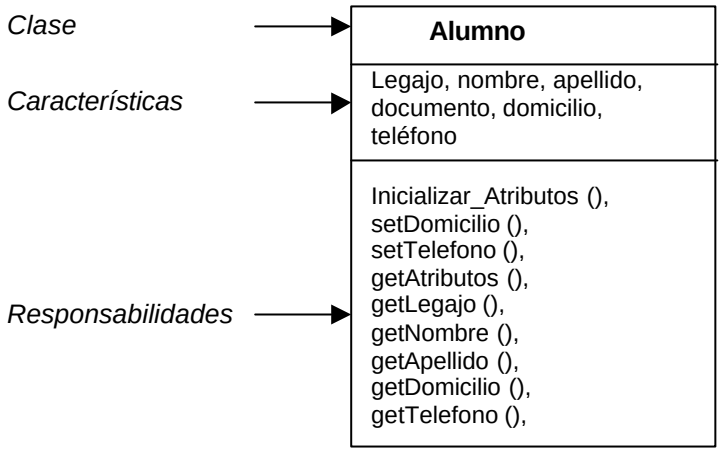
Objetos



Comunicación entre objetos

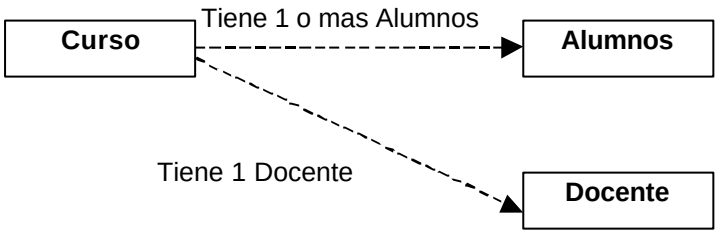


Clases

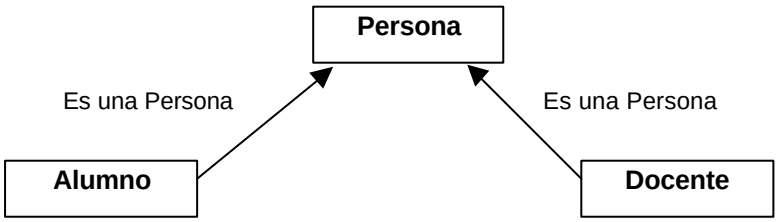


Relaciones entre clases

Relaciones tiene un : Se representan con linea punteada



Relaciones es un : Se representan con linea recta



2. Nombre de los métodos

En una clase se pueden presentar lo siguientes métodos:

- Métodos de interface:
  - Entrada al objeto : Reciben lo posibles cambios que se originan del exterior del objeto, como por ejemplo cambiar algún atributo.
  - Salida del objeto : Las respuestas propias del objeto, como por ejemplo mostrar el valor de algún atributo.
- Métodos de control :
  - Internos: Son métodos internos del objeto que se ejecutan por pedidos del propio objeto, no requieren interface con otros objetos, por ejemplo el cálculo interno de algún atributo.
  - Externos: Son métodos que se ejecutan por pedidos de objetos externos, no requieren interface de entrada/salida, por ejemplo la actualización de algún cálculo.

1. **Para los métodos de entrada** vamos a usar la palabra set+Nombre del atributo, por ejemplo para cambiar el valor del atributo nombre podemos escribir setNombre().
2. **Para los métodos de salida** vamos a usar la palabra get+Nombre del atributo, por ejemplo para retornar el valor del atributo nombre podemos escribir getNombre().
3. **Para los métodos de control** se puede usar cualquier nombre acorde a la misión del método.

Ejercicios prácticos

1. Identificación de entidades, enunciando sus caracteristicas y responsabilidades, representarlas con un diagrama de clase simple.
2. Dado un contexto real identificar y graficar el conjunto de clases y las relaciones existentes entre ellas (diseñar la estructura de clases).

Por ejemplo:

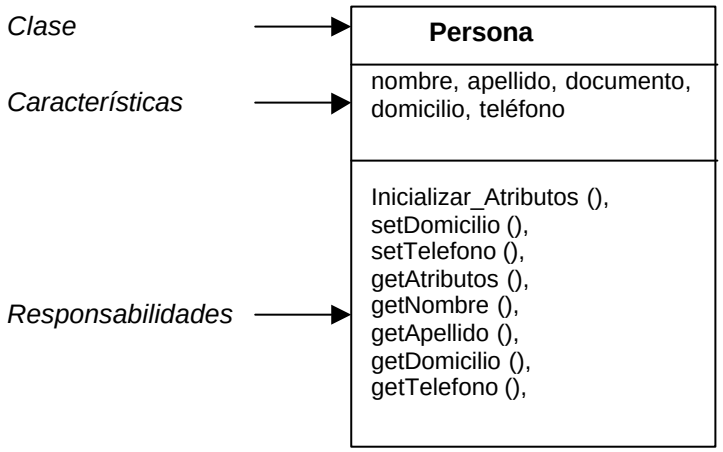
1. Dados las siguientes entidades definirlas, identificar la clase, las características, y las responsabilidades correspondientes, representarlas con un diagrama de clase simple.

1. Persona 2. Cliente 3. Proveedor 4. Producto
5. Mesa 6. Automóvil 7. Avión
8. Cuenta bancaria 9. Edificio
10. Definir cualquier entidad de la vida real, que se utilice en la facultad, trabajo, hogar, etc.

Resoluciones

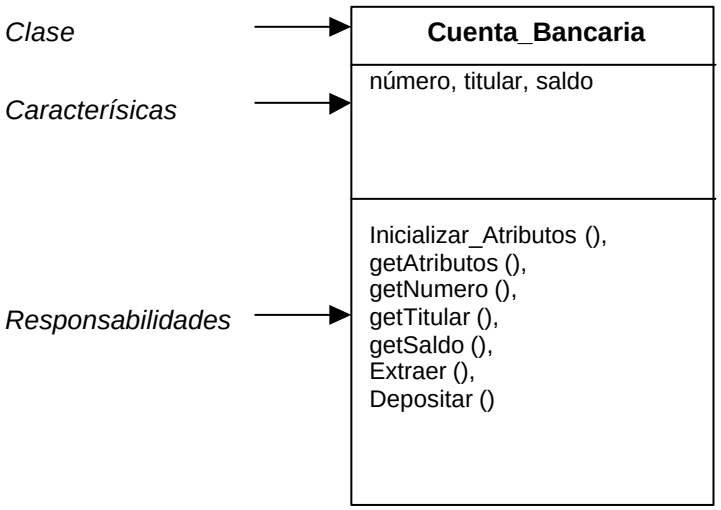
1.1. Persona

Características : nombre, apellido, documento, domicilio y teléfono
Responsabilidades : Inicializar atributos, cambiar domicilio, cambiar teléfono, notificar todos los atributos, notificar Nombre, notificar documento, notificar domicilio, notificar teléfono, etc.



1.9. Cuenta bancaria

Características : Numero, Titular, Saldo
Responsabilidades : Inicializar atributos, notificar numero, notificar titular, notificar saldo, depositar, extraer, etc.



2. Dados las siguientes entidades y/ situaciones, identificar y graficar el conjunto de clases intervinientes.

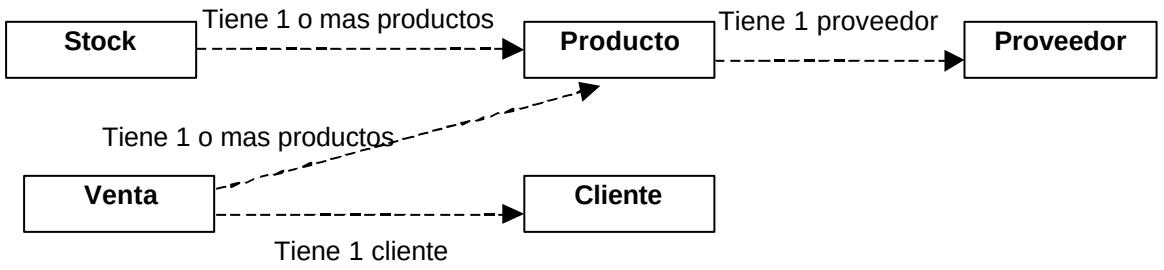
1. Factura
2. Deposito de autos
3. Stock de productos
4. Pago de haberes a los empleados
5. Listado de asistencia a los alumnos

6. Identifique los entidades intervinientes en un negocio estandar de compra venta, grafique las clases y las relaciones entre ellas.

Clases intevinientes: Cliente, Proveedor, Producto, Stock, Ventas.

Cliente	**Proveedor**	**Producto**
código, nombre, apellido, documento, domicilio, teléfono	código, nombre, apellido, documento, domicilio, teléfono	código, nombre, precio, cantidad, proveedor
Inicializar_Atributos (), Cambiar_Domicilio (), Cambiar_Telefono (), Mostrar_Atributos (), Mostrar_Nombre (), Mostrar_Apellido (), Mostrar_Domicilio (), Mostrar_Telefono ()	Inicializar_Atributos (), Cambiar_Domicilio (), Cambiar_Telefono (), Mostrar_Atributos (), Mostrar_Nombre (), Mostrar_Apellido (), Mostrar_Domicilio (), Mostrar_Telefono ()	Inicializar_Atributos (), Cambiar_Nombre (), Cambiar_Cantidad (), Cambiar_Precio (), Cambiar_Proveedor (), Mostrar_Atributos (), Mostrar_Nombre (), Mostrar_Cantidad (), Mostrar_Precio (), Mostrar_Proveedor ()

Stock	**Ventas**
nombre, lista de productos	numero, cliente, lista de productos
Inicializar_Atributos (), Cambiar_Productos (), Cambiar_Telefono (), Mostrar_Atributos (), Mostrar_Nombre (), Mostrar_Productos ()	Inicializar_Atributos (), Cambiar_Productos (), Cambiar_Cliente (), Mostrar_Atributos (), Mostrar_Numero (), Mostrar_Cliente(), Mostrar_ Productos ()



7. Diseñe la estructura de un sistema que maneje la información de productos en una librería, que vende libros técnicos y novelas.

Las novelas se clasifican como de ciencia ficción, romance, misterio, juveniles y policiales. Los libros técnicos se clasifican como de ingeniería, ciencias naturales o ciencias sociales.

Cada libro tiene un título, uno o más autores, una editorial, un año de edición y un formato (tapas duras o edición económica). Los libros técnicos tienen además un código ISBN y capítulos, los que tratan una o más materias.

La librería obtiene los libros por medio de proveedores que representan a una o más editoriales. De cada libro se tiene un stock (que puede ser cero). Al venderse un libro, el stock se actualiza. Si un cliente requiere un libro cuyo stock es cero, se puede realizar un encargo por parte del cliente. Esto significa que se pide el libro a un proveedor de la editorial del libro.

8 El jefe de una organización de desarrollo de software desea administrar el desarrollo de los proyectos.

Cada proyecto de desarrollo de software está a cargo de un jefe de proyecto, el cual lidera y coordina al equipo de proyecto, conformado por un grupo de análisis y diseño, un grupo de construcción y un representante de los usuarios que solicitó el software.

Cada proyecto contempla el desarrollo de una o más softwares (aplicaciones) que se componen a su vez de elementos de software (aplicaciones módulo). Cada elemento de software tiene un nombre, un responsable del análisis y diseño (miembro de ese grupo), un responsable de la construcción (miembro del grupo de construcción), una fecha de actualización (la última) y un plan de test (a desarrollar por el equipo de SQA).

Un plan de test es una declaración narrativa de las pruebas a aplicar al elemento de software para aprobarlo o no. El equipo SQA pertenece a la organización, pero debe ser independiente del equipo de desarrollo del software que contiene al elemento a evaluar.

Se desea poder manipular esta información y generar reportes de todos los proyectos en desarrollo.