UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACIÓN I

Unidad Temática N°1:

Introducción a la Programación con Objetos

Material Teórico

1° Año – 1° Cuatrimestre

Índice

INTRODUCCIÓN A LA PROGRAMACIÓN CON OBJETOS

Orientación 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 bebés 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 bebés 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.

¿Qué es la programación 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.

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.

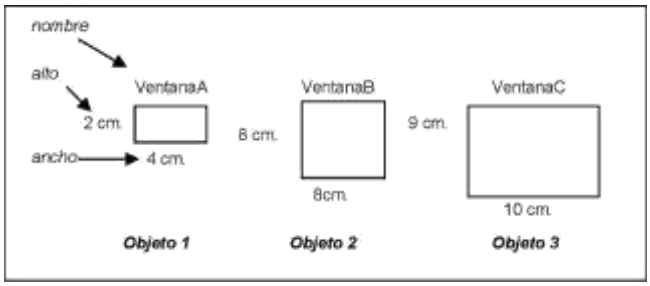

Gráfico 1: Elaboración propia

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 siguiente representa un objeto A (emisor) que envía un mensaje Test al objeto B (receptor).

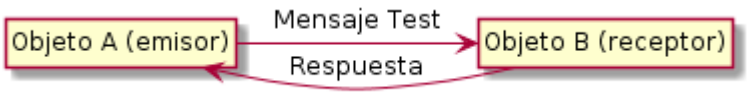

Gráfico 2: Elaboración propia

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:

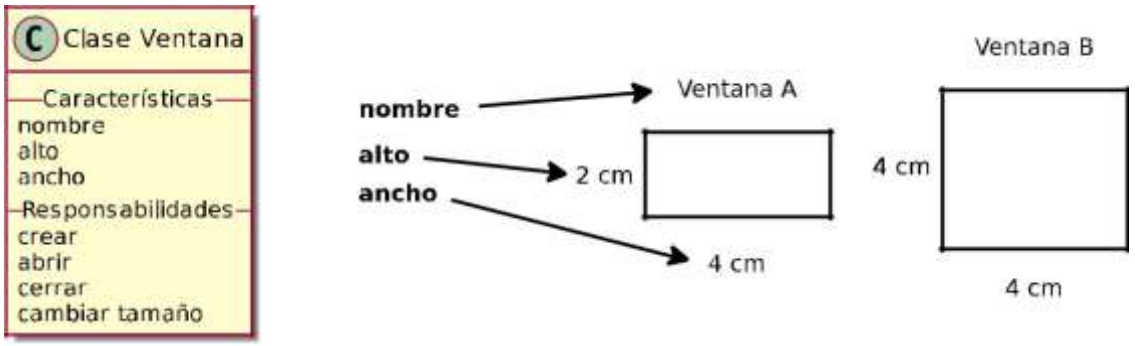

Gráfico 3: Elaboración propia

PRINCIPIOS DE LA POO

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.

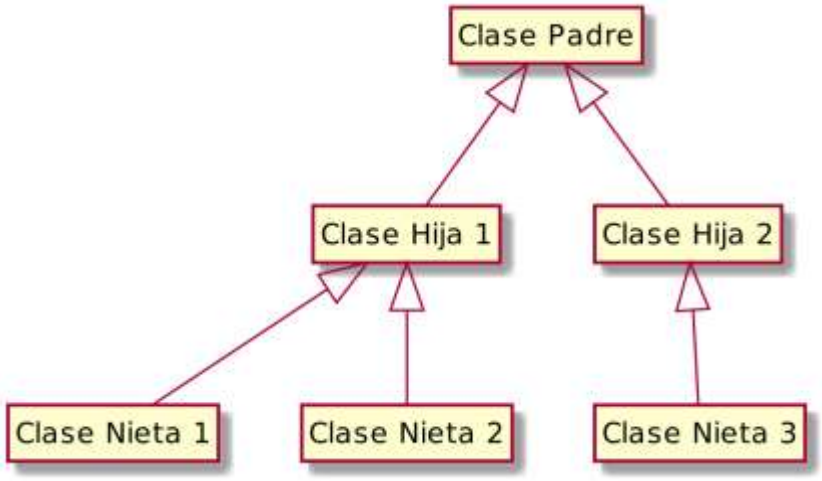

Gráfico 4: Elaboración propia

Ejemplo

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

Primer paso:Identificamos las características esenciales para cada entidad:

Gráfico 5: Elaboración propia

Segundo paso: Si observamos, los dos tienen características en común: Nombre, Domicilio y Teléfono 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:

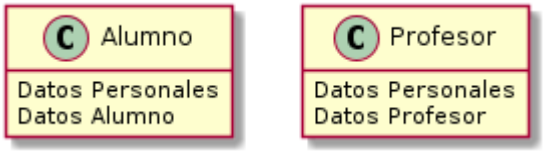

Gráfico 6: Elaboración propia

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

Gráfico 7: Elaboración propia

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.

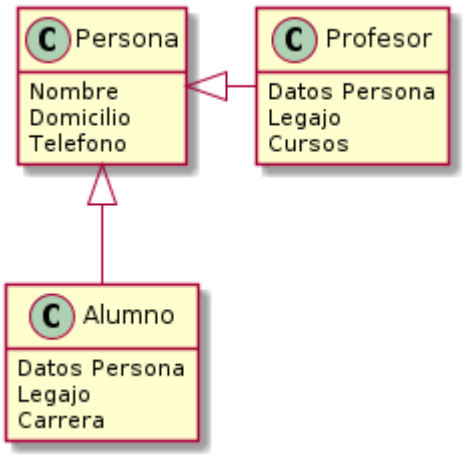

Gráfico 8: Elaboración propia

Polimorfismo

Dentro del ámbito de la programación orientada a objetos, podríamos definir al polimorfismo formalmente diciendo: dos objetos son polimórficos, 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 polimórficos 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 triángulo y un círculo. 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 círculo, como un cuadrado o un triángulo 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 círculo o un triángulo, porque al fin y al cabo, todos pueden responder a ese mensaje. En ese caso decimos que tanto los objetos “círculo”, como los “triángulo” y los “cuadrado” son polimórficos respecto del mensaje dibujar. Es importante notar que no es lo mismo dibujar una figura que otra, seguramente, cada figura implementa ese método de modo diferente.

Otro ejemplo: supongamos que en un sistema de un banco tenemos objetos que representan 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 a

esos mensajes y entonces serán polimórficos 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 realizará 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, él 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.

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

MODELADO DE CLASES CON UML

Lenguaje Unificado de Modelado (UML)

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el Object Management Group (OMG).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

¿Para qué necesitamos UML en programación?

Para comenzar a desarrollar nuestras soluciones orientadas a objetos es necesario primero identificar las clases principales del dominio del problema. Uno de los principales diagramas que nos ofrece UML es precisamente un **Diagrama de**

clases, mediante la cual es posible representar clases, atributos, métodos y relaciones.

Existen muchas herramientas que nos permiten trabajar con diagramas UML como MagicDraw, StarUML, ArgoUML o YUML. En particular esta última permite crear los diagramas a partir de unos comandos escritos en texto plano y dibujarlos directamente en un sitio Web, sin necesidad de instalar software en nuestros equipos. Esta herramienta es ideal para casos en los que necesitamos realizar de manera rápida unos diagramas sencillos para enviarlos a alguien o guardarlos. Lo bueno de esta herramienta es que al interpretar texto plano nosotros podemos generar y almacenar este texto y crear tantas modificaciones o copias como queramos.

Diagramas de clases con YUML

Estos diagramas sólo tienen un tipo de elemento que serían las **clases**. Estas clases se deben escribir entre corchetes [ ]. Es posible colorear una clase poniendo dentro unas llaves indicando bg:color.

Ejemplo: `[Clase{bg:green}]`.

Si se desea elaborar más cada clase y no quedarse únicamente con el nombre, se pueden indicar atributos y métodos utilizando el separador | (pipe).

Aunque hay pocos tipos de elementos, las relaciones entre estas son muy variadas. Estas relaciones son: > asociación simple, -texto> asociación direccional, 1-0..* cardinalidad, <>-1> Agregación, ++-1> composición, ^- herencia, ^-.- implementación y .> uso.

Veamos otro sencillo ejemplo con el siguiente texto.

```
[Vehiculo]<>-*>[Pasajeros]

[Vehiculo]^-[Coche]

[Vehiculo]^-[Moto]

[Conductor]-.->[Vehiculo]

[< <Desplazable>>]^-.-[Vehiculo]
```

El diagrama resultante es:

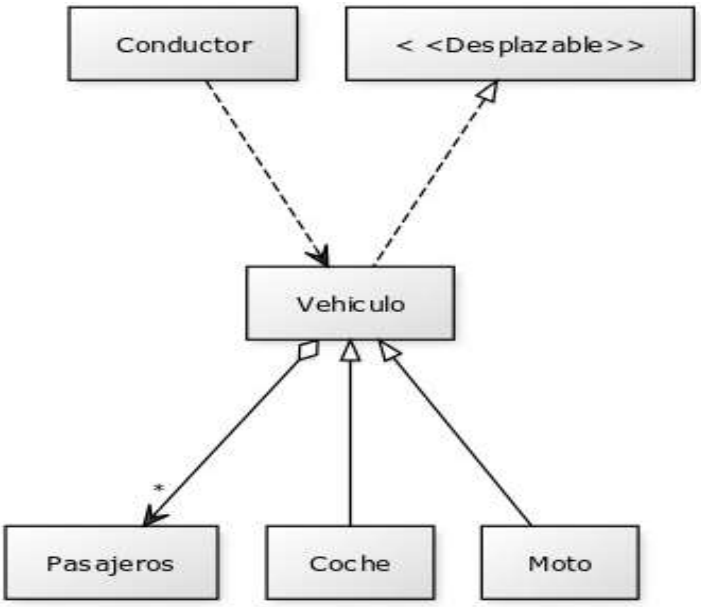

Gráfico 9: Elaboración propia

Podríamos modelar nuestra clase Ventana, citada anteriormente mediante el siguiente texto:

```
[ClaseVentana|-nombre; -alto;-ancho|+crear();+abrir();+cerrar(); +cambiarTamanio()]
```

El diagrama resultante es:

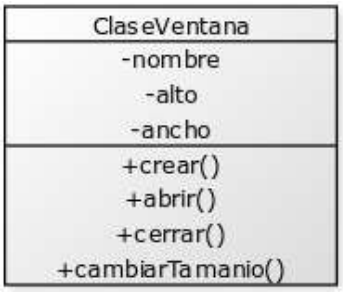

Gráfico 10: Elaboración propia

Nota: Ambos diagramas de clases fueron creados desde la página: YUML

BIBLIOGRAFÍA

Bishop, P. (1992) Fundamentos de Informática. Anaya.

Brookshear, G. (1994) Computer Sciense: An Overview. Benjamin/Cummings.

De Miguel, P. (1994) Fundamentos de los Computadores. Paraninfo.

Joyanes, L. (1990) Problemas de Metodología de la Programación. McGraw Hill.

Joyanes, L. (1993) Fundamentos de Programación: Algoritmos y Estructura de Datos. McGraw Hill.

Norton, P. (1995) Introducción a la Computación. McGraw Hill.

Prieto, P. Lloris A. y Torres J.C. (1989) Introducción a la Informática. McGraw Hill.

Tucker, A. Bradley, W. Cupper, R y Garnick, D (1994) Fundamentos de Informática (Lógica, resolución de problemas, programas y computadoras). McGraw Hill.