Nombre del alumno:
Victor Hugo López Moreno

Nombre del profesor:
Andrés Alejandro Reyes Molina

Nombre del trabajo:
Supernota

Materia:
Fundamentos y lógica de programación

Grado: 3°

Características del modelo orientado a objetos.

Abstracción

Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar "cómo" se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos, y, cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción. El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

Encapsulamiento

Significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión (diseño estructurado) de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

Polimorfismo

Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O, dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Herencia

Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los objetos en clases, y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase, se dice que hay herencia múltiple; siendo de alta complejidad técnica por lo cual suele recurrirse a la herencia virtual para evitar la duplicación de datos.

Modularidad

Se denomina "modularidad" a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la modularidad de diversas formas.

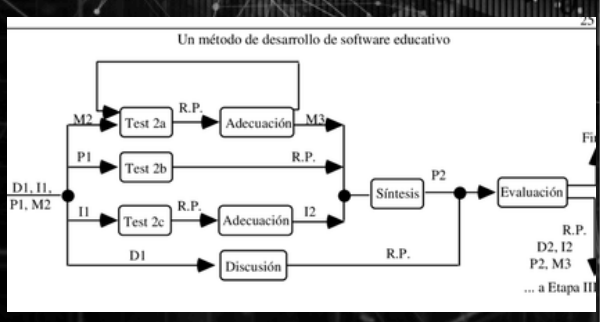
Principio de ocultación

Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una "interfaz" a otros objetos que detalla cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

Recolección de basura

La recolección de basura (garbage collection) es la técnica por la cual el entorno de objetos se encarga de destruir automáticamente, y por tanto desvincular la memoria asociada, los objetos que hayan quedado sin ninguna referencia a ellos. Esto significa que el programador no debe preocuparse por la asignación o liberación de memoria, ya que el entorno la asignará al crear un nuevo objeto y la liberará cuando nadie lo esté usando. En la mayoría de los lenguajes híbridos que se extendieron para soportar el Paradigma de Programación Orientada a Objetos como C++ u Object Pascal, esta característica no existe y la memoria debe desasignarse expresamente.

Representación gráfica del diseño.



El diseño gráfico de sistemas (DGS, o GSD por sus siglas en inglés) es un método actual de diseño, prototipaje y creación de sistemas integrados el cual combina programación gráfica con hardware COTS para simplificar el desarrollo, dando como resultado diseños de gran calidad y la posibilidad de diseños personalizados. De esta manera alguien no experto en diseño de sistemas integrados, puede hacer un diseño de lo que necesita sin tener que recurrir a un experto.
Esta forma de diseño se encuentra a un nivel más alto (de abstracción) que el diseño de Sistemas Electrónicos (ESL por sus siglas en inglés).

El diseño gráfico de sistemas nos permite diseñar un sistema electrónico completo, usando software intuitivo como son los lenguajes de programación gráficos y sin la necesidad de tener que hacer uso del hardware que necesitará nuestro diseño, nos permitirá crear prototipos y la posibilidad de hacer pruebas de simulación.

Relación entre la programación orientada a objetos y la estructurada.

CUADRO COMPARATIVO

Enfoque Estructurado	*Enfoque Orientado a Objetos*
El análisis está orientado a los Procesos del sistema.	El análisis está orientado a los Objetos.
En este análisis se llega solo a la fase de integración y no toma en consideración los cambios que ocurren dentro del sistema en el proceso de análisis y diseño de sistemas.	Un programa que se usa en un ambiente real necesariamente debe cambiar. Los cambios difieren un poco de los requeridos en evolución, pues contemplan la introducción de nuevas funcionalidades no previstas en el problema original.
Consta de 5 Fases (Análisis, Diseño, Codificación, Pruebas e Integración).	Consta de 4 Fases (Análisis, Diseño, Evolución y Modificación).
El Diseño inicia una vez que ha culminado la fase de análisis de sistema.	El Diseño inicia aún antes de concluir con la etapa de análisis. Se recomienda analizar un poco y diseñar. Esta etapa debe concluir una vez que se establecieron claves y mecanismos importantes.
Se consideran los elementos o perspectivas básicas del análisis (Entrada-Proceso-Salida), en función del Software.	Se consideran los conceptos básicos como el Objeto y el Atributo, el todo y sus partes (software), clases y miembros. Modela los objetos que son parte de él.

Diseño de la solución (modelado).

La POO es una forma de programación que organiza el software en torno a objetos, en lugar de en torno a funciones y lógica. Un objeto es una entidad que contiene datos y comportamientos únicos. En la POO, se define una clase, que es una plantilla para crear objetos, y se utilizan instancias para crear objetos concretos.

La POO es una forma de programación que organiza el software en torno a objetos, en lugar de en torno a funciones y lógica. Un objeto es una entidad que contiene datos y comportamientos únicos. En la POO, se define una clase, que es una plantilla para crear objetos, y se utilizan instancias para crear objetos concretos.

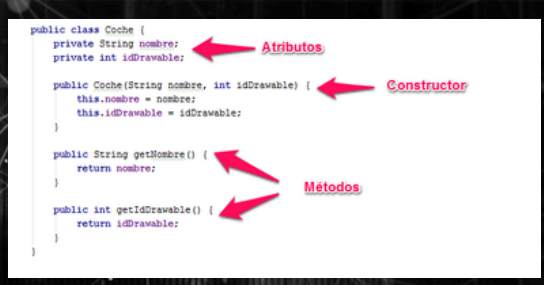
Aplicaciones prácticas

La POO se utiliza ampliamente en el desarrollo de software en una variedad de campos. Los desarrolladores utilizan la POO para crear programas de gran escala, aplicaciones de escritorio, juegos, aplicaciones móviles y más.

En Certitec somos programadores de .NET 100% y contamos con tecnología orientada a objetos que permite desarrollar y modificar software de gestión basado en un modelo de arquitectura .NET.

estructura de una clase en programación

En informática, una clase es una plantilla para el objetivo de la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje. Cada clase es un modelo que define un conjunto de variables y métodos apropiados para operar con dichos datos. Cada objeto creado a partir de la clase se denomina instancia de la clase.
La programación orientada a objetos es la base principal para los tipos de objetos. Permiten abstraer los datos y sus operaciones asociadas al modo de una caja negra. Los lenguajes de programación que soportan clases difieren sutilmente en su soporte para diversas características relacionadas con clases. La mayoría soportan diversas formas de herencia. Muchos lenguajes también soportan características para proporcionar encapsulación, como especificadores de acceso.
Una clase también puede tener una representación (metaobjeto) en tiempo de ejecución, que proporciona apoyo en tiempo de ejecución para la manipulación de los metadatos relacionados con la clase.



Elementos de una clase

Palabras clave reservadas
Estas palabras clave fueron seleccionadas para tener un significado especial en un lenguaje de programación, son usadas únicamente para definir estructuras de control y operaciones en el código, y no pueden ser utilizadas como identificadores. Algunos ejemplos de palabras reservadas son: "if", "else", "for", "while", "break", "return", entre otros.

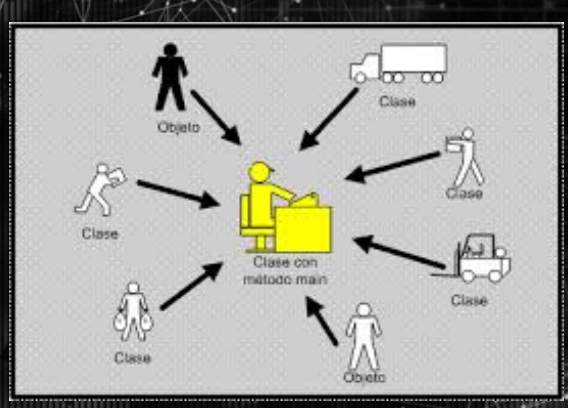
Estructuras de control
Las estructuras de control inician con palabras clave y son un elemento utilizado para controlar el flujo del programa, decidir que acciones realizar en función de las condiciones que se determinen en el código. Las estructuras de control comunes incluyen "**if**", "**else**", "**while**", "**for**" y "**switch**".

Identificadores
Los identificadores son nombres que se emplean para referirse a variables, funciones, clases, objetos, entre tantos otros… pero siempre deben ser únicos y descriptivos para que el código sea fácil de leer y entender.
Los identificadores pueden ser cualquier combinación entre letras, números y caracteres especiales como "**_**" o "**$**". Algunos ejemplos de identificadores son: "**suma**", "**edad_usuario**", "**mi_funcion**", "**MiClase**", etc.

clase principal

CLASE CON EL MÉTODO MAIN: CLASE PRINCIPAL, INICIADORA O “PROGRAMA PRINCIPAL”
Hasta ahora hemos visto código implementando clases y que las clases definen tipos, es decir, nos permiten crear objetos del tipo definido por la clase. Pero a todas estas, ¿dónde está el programa? Todavía no hemos visto ningún programa, y ya es hora de que abordemos este asunto.

CLASE CON EL MÉTODO MAIN: CLASE PRINCIPAL, INICIADORA O “PROGRAMA PRINCIPAL”
Hasta ahora hemos visto código implementando clases y que las clases definen tipos, es decir, nos permiten crear objetos del tipo definido por la clase. Pero a todas estas, ¿dónde está el programa? Todavía no hemos visto ningún programa, y ya es hora de que abordemos este asunto.



El método main normalmente no será muy extenso en cuanto a líneas de código respecto al resto del código (clases). Si esto ocurriera, posiblemente sería indicador de que este método tiene más protagonismo del que debe, y en nuestro símil el árbitro no debe ser protagonista del partido. En algunos ejemplos de código que desarrollemos quizás la clase con el método main contenga gran parte del código total, pero si esto es así será porque lo estamos utilizando con fines didácticos. En programas profesionales esto no debe ocurrir.

Fuentes

https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos#:~:text=Algunas%20caracter%C3%ADsticas%20clave%20de%20la,%2C%20polimorfismo%2C%20acoplamiento%20y%20encapsulamiento.

https://es.wikipedia.org/wiki/Dise%C3%B1o_gr%C3%A1fico_de_sistemas#:~:text=El%20dise%C3%B1o%20gr%C3%A1fico%20de%20sistemas%20nos%20permite%20dise%C3%B1ar%20un%20sistema,posibilidad%20de%20hacer%20pruebas%20de

https://immune.institute/blog/programacion-orientada-a-objetos/

https://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)#:~:text=2%20V%C3%A9ase%20tambi%C3%A9n-,Componentes,de%20manipulaci%C3%B3n%20de%20dichos%20datos.

https://prezi.com/td6u_whbiy9g/elementos-principales-de-la-programacion-orientada-a-objetos/#:~:text=Una%20clase%20se%20compone%20por,la%20clase%20con%20su%20entorno.