UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN

Licenciatura En Informática

Informática IV

(Programación Orientada a Objetos)

Autor: L.I. María de Lourdes Isabel Ponce Vásquez

SEPTIEMBRE 2020 - ENERO 2021

Contenido

OBJETIVO GENERAL

Al terminar el curso, el alumno conocerá la filosofía de la orientación a objetos, así como su implementación en un lenguaje de programación.

El alumno deberá estar familiarizado con los fundamentos básicos de las estructuras de datos y el análisis y diseño orientado a objetos.

UNIDAD 1. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

Objetivos Específicos

- Conocer brevemente los orígenes de la programación orientada a objetos
- Definir qué es la orientación a objetos
- Definir qué es el paradigma orientado a objetos
- Conocer los principios fundamentales de la programación orientada a objetos
- Describir algunas de las características de los lenguajes orientados a objetos

1.1 Introducción

En los 70's se hablaba de la "crisis del software", lo que dio lugar en el campo de la informática a una tendencia estructurada (programación estructurada, análisis estructurado, diseño estructurado, etc.) se pensaba que lo estructurado era la solución a todos los problemas de desarrollo de software ya que aparecieron metodologías que promovían una solución jerárquica, disciplinada, organizada y planificada para el desarrollo de software. Se introdujeron algunas mejoras en los lenguajes de programación para facilitar estas características, como estructuras de control para eliminar el goto, algunos recursos de modularidad y de organización de bibliotecas de programas.

Sin embargo, desafortunadamente pocos sistemas lograban terminarse, pocos se terminaban cumpliendo los requerimientos iniciales y no todos los que se terminaban cumpliendo los requerimientos iniciales se usaban según lo planeado. El problema (mal llamado de mantenimiento) era la falta de adaptabilidad a nuevos requerimientos que no habían sido planeados. La "estructuración" facilitaba la revisión y modificación de cientos de líneas de código, pero no lo impedía.

En la realidad es prácticamente imposible planear absolutamente todo, puesto que aprendemos con la experiencia, no de la planificación. Actualmente es necesario desarrollar prototipos rápidamente sin tener que esperar a que los requerimientos iniciales estén totalmente definidos; es preciso aprovechar el software existente y tener facilidad de adaptación de este software a otros usos diferentes a los originales, sin tener que modificar el código existente, con los riesgos que implica que algo que funcionaba ya no funcione.

La orientación a objetos vino a suplantar la tendencia estructurada en los 90's, con los objetivos de hacer frente a una mayor productividad, funcionalidad y sencillez en el desarrollo de aplicaciones (programas, interfaces y bases de datos).

Actualmente la frase "orientado (a) a objetos", es muy popular, escuchándose a cada momento frases como "sistemas operativos orientados a objetos", "lenguajes orientados a objetos", "programación orientada a objetos" (POO), "bases de datos orientadas a objetos", etc. Sin embargo, el concepto tiene más de 50 años, cuando se dio su nacimiento con la creación del lenguaje Simula en 1967, este lenguaje fue creado para aplicaciones de simulación. Su redescubrimiento y popularidad se deben a lenguajes como Smalltalk creado a finales de los 70s, Java y C++, lenguaje que fue creado por Bjärne Stroustrup y fue basado en un lenguaje que es usado ampliamente como C.

La Orientación a Objetos constituye una metodología de diseño y desarrollo de software barato. Esta metodología de análisis, diseño y programación que configura las fases fundamentales del ciclo de vida

de un sistema informático, se asentó como la estructura metodológica de los años 90, y se considera actualmente como un paradigma en el desarrollo de aplicaciones.

1.2 El Paradigma de la Orientación a Objetos

1.2.1 ¿Qué es la OO (Orientación a Objetos)?

La **Orientación a Objetos** se puede definir como "*una disciplina de ingeniería de desarrollo y modelado de software que permite construir más fácilmente sistemas complejos a partir de componentes individuales*".

La Orientación a Objetos permite una representación más directa del mundo real, reduciendo fuertemente la transformación radical normal desde los requerimientos del sistema, definidos en términos del usuario, a las especificaciones del sistema, definidas en términos de la computadora. En vez de tratar de modelar un problema en algo familiar a la computadora se trata de acercar la computadora al problema, es decir, modelar el problema mediante entidades independientes que interactúan y cuyas fronteras no están determinadas por su instrumentación computacional sino por la naturaleza del problema.

Actualmente, la tendencia de la Informática es la de producir componentes reutilizables para ensamblarlos a otros y obtener así, el producto. Estos elementos reutilizables son denominados "Componentes Integrados de Software" (cis) por su teórica similitud con los "componentes integrados de hardware" (chips), innovación que revolucionó la industria informática hace ya muchos años.

El **Paradigma Orientado a Objetos** (PaOO) es pues *una filosofía de desarrollo y empaquetamiento de software que permite crear unidades funcionales extensibles y genéricas donde las estructuras de datos y el comportamiento de éstas, se incluyen en la misma representación, de forma que el usuario las pueda aplicar según sus necesidades y de acuerdo con las especificaciones del sistema a desarrollar*.

1.2.2 Ventajas de la OO

Las técnicas OO fueron creadas para mejorar la capacidad del profesional de cómputo de diversos modos. Es recomendable comprender todos sus beneficios potenciales e intentar conseguirlos todos, en vez de sólo algunos. Los beneficios que la Orientación a Objetos proporciona son:

- Reutilización: las clases pueden diseñarse para que se reutilicen en otros sistemas, al modificar y ampliar con facilidad la implementación de estos componentes sin afectar al resto de su estructura. También se pueden construir componentes reutilizables de software y bibliotecas específicas de estos componentes fácilmente extensibles. Se debe construir un repositorio de clases con esta finalidad.
- Estabilidad y confiabilidad: las clases diseñadas para reutilización repetida se vuelven estables y es probable que estas clases tengan menos fallas que el software creado a partir de cero. Aunque no implica que estén libres de errores.
- Encapsulamiento: el diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel, el encapsulamiento oculta los detalles y hace que las clases complejas sean fáciles de usar (las clases pueden verse como cajas negras).
- Escalabilidad: ya que permite construir clases cada vez más complejas; las clases se construyen a partir de otras existentes y probadas, esto permite construir componentes complejos de software que a su vez se convierten en bloques de construcción de software más complejo.
- Diseño de mayor calidad: puesto que se integran a partir de componentes ya existentes y probados, que han sido verificados y pulidos varias veces.

- Integridad: los objetos sólo pueden usar métodos específicos, por y de la forma que se definió en su creación, esto tiene particular importancia en los sistemas cliente-servidor y los sistemas distribuidos, donde los usuarios desconocidos podrían intentar el acceso al sistema.
- Programación más sencilla: Los programas se elaboran a partir de piezas pequeñas, cada una de las cuales, puede crearse y ensamblarse fácilmente, generalmente. El programador crea un método para una clase a la vez. El método cambia el estado de los objetos en formas que suelen ser sencillas cuando se les considera en sí mismas.
- Mantenimiento más sencillo: ya que, al dar mantenimiento, se modifica un solo método de una clase a la vez y cada clase efectúa sus funciones independientemente de las demás.
- Modelo más realista: modela la empresa o área de aplicación de un modo más cercano a la perspectiva del usuario, lo que permite realizar la transformación casi transparente en el desarrollo.
- Mejor comunicación entre los profesionales de los sistemas de información y los empresarios: Los empresarios comprenden más fácilmente los términos de eventos, objetos y políticas que describen el comportamiento de los objetos ya que se interactúa fácilmente con un entorno computacional, usando metáforas familiares.
- Independencia de diseño: las clases están diseñadas para ser independientes del ambiente de plataformas, hardware y software, lo que permite emplearlas en diferentes proyectos.
- Mejores herramientas: existen herramientas CASE y Frameworks que aceleran el desarrollo y permiten controlar los proyectos.

1.3 Principios fundamentales de la OO

Algunos autores centran los principios fundamentales que configuran el paradigma orientado a objetos, en tres características:

- Abstracción
- Herencia
- Identidad de los Objetos

Otros más estiman que la OO se basa sólo en:

- Herencia
- Encapsulamiento o encapsulación
- Polimorfismo

Mientras que otros señalan siete u ocho de los siguientes aspectos como básicos a considerar para una verdadera orientación al objeto:

- Modularidad
- Abstracción
- Objetos
- Clases
- Encapsulamiento
- Herencia
- Polimorfismo y enlace dinámico
- Persistencia

Otros conceptos como la generalización y manipulación de excepciones no son propios del modelo de objetos, pero encuentran en este paradigma una mejor forma de expresión. Para efectos del curso tomaremos los ocho elementos anteriores como las características fundamentales para obtener una buena representación orientada a objetos.

1.3.1 Modularidad

La **modularidad** consiste en *dividir un programa en partes llamadas módulos, los cuales pueden trabajarse por separado*. En términos de programación, los módulos pueden compilarse por separado y la división no depende de cierto número de líneas sino es una división en términos de integrar en un módulo un conjunto de procedimientos relacionados entre sí, junto con los datos que son manipulados por tales procedimientos.

El objetivo de la modularidad es reducir el costo de elaboración de programas al poder dividir el trabajo entre varios programadores y aumentar la productividad.

Por ejemplo, un automóvil está constituido por un conjunto de módulos tales como un sistema eléctrico, uno mecánico y uno de frenado. Cada módulo se trabaja por separado y el especialista sólo conoce la forma en que se relaciona el módulo con los otros, pero no tiene por qué saber los detalles de funcionamiento de otros módulos o sistema.

Estos conceptos no son exclusivos de la OO, pues se han desarrollado desde la programación estructurada, sólo que en ésta se pueden omitir (desde luego bajo responsabilidad del programador), pues hacerlo, lleva a tener grandes programas en un solo archivo y sin estructura alguna, lo cual causa grandes pérdidas de tiempo al desear modificar tal programa.

La OO no puede lograrse sin hacer uso de los mecanismos antes mencionados y para ello, proporciona los medios para ir creando módulos llamados *clases*. En las clases, cada tipo no simple es un módulo, y cada módulo de alto nivel es un tipo.

1.3.2 Abstracción

La **abstracción** es una *descripción o especificación simplificada de un sistema que hace énfasis en algunos detalles significativos y suprime los irrelevantes*.

La abstracción es esencial para construir cualquier sistema, sea o no orientado a objetos y debe enfocarse más en qué es un objeto y qué hace, antes de pensar en la implementación.

Por ejemplo, un automóvil puede abstraerse como un objeto que sirve para desplazarse a mayor velocidad, sin importar cómo lo haga.

Una característica de la abstracción es que un objeto puede abstraerse de diversas formas, dependiendo del observador. Así, el automóvil que se mencionaba puede ser visto como un objeto de colección por un coleccionista, una herramienta de trabajo por un corredor profesional, una mercancía por un vendedor, etc.

Las abstracciones en un sistema OO ayudan a representar los diferentes puntos de vista incorporados en el sistema a desarrollar. Los objetos son descritos como implementaciones de tipos de datos abstractos. En conjunto las abstracciones forman una jerarquía que muestra cómo las diferentes perspectivas de un sistema se relacionan unas con otras.

Como sabemos, la abstracción es un concepto básico en la solución de problemas, ya que permite definir conceptos generales a partir de objetos particulares que representan algún aspecto del mundo real. Los TDA son independientes de la implementación, lo cual hace posible centrarse en los datos y operaciones sin pensar en el lenguaje de programación y posteriormente traducirlos al lenguaje deseado.

1.3.3 Objetos

A pesar de que el punto central en esta nueva metodología de programación es el concepto de objeto, resulta difícil tratar de definirlo. En un diccionario se puede encontrar la siguiente definición:

> "Un objeto es cualquier cosa que se ofrece a la vista y afecta los sentidos. Es una entidad tangible que exhibe algún comportamiento bien definido".

En términos de programación, un objeto no es necesariamente algo tangible (por ejemplo: un proceso). Por lo que un objeto es: "*un concepto, abstracción o cosa con límites bien definidos y significado para una aplicación*". Lo que sí puede decirse de todo objeto es que tiene *estado, comportamiento e identidad*.

El **estado** de un objeto abarca todas las *propiedades o características distintivas del mismo y los valores de cada una de estas propiedades*. En términos de programación, puede decirse que las propiedades son las variables que sirven para describir tal objeto.

El **comportamiento** es la *forma como actúa o reacciona un objeto en términos de cambio de estado, envío y recepción de mensajes*. Está formado por la definición de las operaciones (funciones y procedimientos) que puede realizar este objeto. Los tipos más comunes de operaciones, o en POO métodos, son: modificar, seleccionar, iterar, construir y destruir. El conjunto de operaciones que un objeto puede realizar sobre otro, se conoce como protocolo.

Identidad es la *propiedad de un objeto que lo distingue de todos los demás*. En un programa, normalmente se trata de un identificador. La identidad de objetos se refiere al hecho de que los datos son organizados en entidades discretas y distinguibles denominadas objetos.

Cada objeto en un sistema OO usualmente tiene un **nombre**, también llamado **referencia** o **asa** (handle); este nombre distingue a un objeto de otros.

En resumen, un *objeto es un conjunto de localidades en memoria con un conjunto de subprogramas (en POO se conocen como métodos) que definen su comportamiento y un identificador asociado*.

Por ejemplo, un automóvil puede tener diversos estados, puede estar estacionado o en circulación. Un comportamiento asociado con el auto puede incluir encenderlo, lo cual modifica su estado, y generalmente tiene una identidad marca, modelo, número de serie del motor y placa.

1.3.4 Clasificación (Clases)

Lo más común es que el programa tenga más de un objeto con propiedades y comportamientos similares, así que en lugar de repetir la definición de un objeto se agrupan las características comunes de los objetos en una clase. La OO emplea la clasificación para agrupar objetos que tienen atributos y comportamientos en común.

Por ejemplo, podríamos tener un conjunto de objetos: un auto compacto, un auto de carreras, un perro maltés, un perro chihuahua, un barco y un triángulo. En este caso se podrían agrupar el auto compacto y

el de carreras en una clase; el perro maltés y el chihuahua en otra clase; el barco en otra y el triángulo en una última.

El grupo de perros puede ser considerado una clase, ya que es posible asociar atributos con la clase perro, tales como color, tamaño o raza y también se pueden asociar operaciones o comportamientos con esta clase: el perro ladra, se alimenta y se baña. Similarmente, la clase auto tiene sus propios atributos (color, tamaño y cantidad de pasajeros) y operaciones (arrancar, circular y detenerse).

La clasificación es un concepto muy antiguo se cree que Aristóteles fue el primero en clasificar a los animales. Es importante notar que esta forma de agrupar refleja la perspectiva de la persona o grupo de trabajo que los han definido. Sin embargo, una clasificación igualmente válida podría ser que el barco y los autos formaran una clase, la clase vehículos de transporte. Esto es importante, ya que una definición de clases y jerarquías está pensada para representar un problema en particular o su solución; dos representaciones diferentes pueden ser igualmente útiles o correctas dependiendo del problema a tratar.

Se dice que un objeto *es una* ***instancia*** *de una clase*. Cada instancia tiene sus propios atributos (es decir, tiene formas de describir su propio estado en cualquier instante de tiempo), pero comparte nombre y comportamientos con las otras instancias de la clase.

Una **clase** *es un módulo y un tipo de datos abstracto que definen un grupo de objetos que comparten características comunes*. Las clases se caracterizan por los mensajes que pueden recibir. Para facilidad de manejo y mantenimiento, las clases se agrupan en paquetes. En relación a su alcance, las clases pueden ser públicas, aquellas que pueden ser vistas desde fuera de un paquete, o privadas, las que sólo son visibles dentro de su paquete.

Por ejemplo, se puede hablar de autos como una clase y mencionar propiedades compartidas tales como los asientos o parabrisas, pero también se pueden emplear los valores de los atributos para distinguir a un miembro de la clase de otro (un auto de carreras corre a una mayor velocidad). Así, una clase describe un conjunto de objetos que comparten una estructura común y tienen comportamientos comunes, pero los valores de los atributos permiten distinguir a unos de los otros.

Los conceptos de clase y objeto están profundamente relacionados, no se puede hablar de un objeto sin prestar atención a su clase. Pero tienen diferencias:

- Objeto es una entidad concreta que existe en el tiempo y el espacio (se crea al momento de ejecución del programa), una clase representa sólo una abstracción, la “esencia” de un objeto (se crea al momento de compilar el programa).
- Los objetos son conceptos dinámicos porque los objetos muestran un comportamiento e interactúan entre ellos, las clases son estáticas, no interactúan entre ellas.

1.3.5 Encapsulamiento

La clase representa la esencia del objeto y el objeto es una entidad que existe en el tiempo y el espacio. El objeto se define también como instancia de la clase a que pertenece. La clase tiene dos vistas: la exterior, en la cual se hace énfasis en la abstracción y se oculta en la estructura y secretos de comportamiento, y la vista interior o implementación. Aquí se nota que es indispensable hacer uso del concepto de encapsulación.

Típicamente, la información acerca de un objeto está encapsulada por su comportamiento. Esto significa que un objeto mantiene datos acerca de cosas del mundo real a las que representa en su sentido verdadero.

Generalmente a un objeto se le debe “pedir” o “decir” que cambie sus propios datos con un mensaje, en vez de esperar que procesos externos cambien la naturaleza de un objeto.

Al encapsular u ocultar los atributos y comportamiento de un objeto, se separan los aspectos externos de un objeto (los accesibles para todos) y se ocultan los detalles de implementación (los accesibles para nadie). Sin embargo, encapsulamiento no es lo mismo que ocultamiento de información; de hecho, el límite de protección del encapsulamiento puede ser **transparente** ó **translúcido**. Esto es, en algunos casos, se puede ver dentro del límite de protección (es transparente) pero otras veces está oculto (es translúcido). De hecho, la abstracción es una técnica que ayuda a identificar qué información debe ser visible y cuál debe ocultarse.

Por lo tanto, el **encapsulamiento** es una *técnica para empaquetar la información de tal forma que se oculte lo que debe ocultarse y haga visible lo que está pensado para serlo*.

El encapsulamiento permite una alta cohesión y un bajo acoplamiento entre módulos (clases).

Con esto se trata de lograr que al tener algún cambio en la implementación de un objeto no se tengan que modificar los programas que utilizan tal objeto.

Siguiendo con el ejemplo del automóvil, se sabe que existen diversos mecanismos para que funcione, en particular se tiene el sistema de frenado que todo mundo sabe que sirve para detener el auto al pisar el pedal de freno, pero sólo el mecánico sabe los detalles de la implementación. Por otro lado, si en algún momento se cambia, para el conductor es transparente.

1.3.6 Herencia

Evidentemente, algunos atributos son compartidos entre miembros de una clase. Por ejemplo, los autos compactos y los autos de carreras tienen asientos, pero difieren en otras características como la velocidad. Las clases pueden ser organizadas jerárquicamente de acuerdo con las semejanzas y diferencias entre ellas; esta jerarquía exhibe la estructura de herencia de clases de la OO.

La **herencia** que es la *noción de que las características y el comportamiento general de una superclase puede darse (compartirse o derivarse) a sus subclases*, permite que una clase sea definida como una extensión o restricción de otra y es la contribución más importante de la POO, pues mediante este mecanismo es posible lograr la principal meta de la OO que es la reutilización de código.

La herencia permite definir una jerarquía de clases. En tal jerarquía, algunas clases son subordinadas a otras llamadas subclases. Para construir la jerarquía se comienza definiendo ampliamente una clase, para refinarla luego en subclases más especializadas.

Una subclase define el comportamiento de un conjunto de objetos que heredan algunas de las características de la clase padre, pero adquieren características especiales no compartidas por el padre, en este sentido se dice que la subclase es una especialización de la clase padre.

En ocasiones se usa una clase abstracta para simplificar la jerarquía, de manera que no se pueda definir ningún objeto de la clase abstracta si no es como una instancia de una subclase.

La herencia define *relaciones entre clases, donde una clase comparte la estructura o comportamiento definidos en una o más clases.*

La **herencia múltiple** y repetida permite que se pueda declarar una clase como heredera de varias, e incluso de ella misma.

La **herencia simple** permite heredar sólo de una clase padre. En estos casos, se implementan las llamadas **interfaces** que permiten simular la herencia múltiple.

1.3.7 Polimorfismo

Un comportamiento es una acción o transformación que un objeto realiza a la cual está sujeto. Un comportamiento de un objeto se activa por la recepción de un mensaje o por la entrada en un estado particular. En ocasiones el mismo comportamiento se manifiesta de manera diferente en diferentes clases o subclases, propiedad que se denomina polimorfismo.

El **polimorfismo** es la *capacidad de tener métodos con el mismo nombre, pero con una implementación diferente*.

Por ejemplo, al tratar de frenar un auto siempre se debe oprimir el pedal del freno y el vehículo se detendrá sin importar si los frenos son de tambor o de disco.

Una *implementación específica de un comportamiento para una cierta clase* se denomina método. En un sistema polimórfico, un comportamiento puede tener más de un método que lo implemente. Por ejemplo, existirá un método frenar cuando se trata de frenos de disco y otro cuando son de tambor, pero el método sigue siendo frenar.

Una forma de polimorfismo en OO se da al usar un operador para aplicarlo a elementos de diferente tipo. Por ejemplo, al pretender sumar enteros, reales o complejos, se emplea el mismo símbolo +, esto se conoce como **sobrecarga de operadores**. En este caso el compilador se encarga de determinar cuál es el método que se está invocando de acuerdo a los objetos involucrados en la operación.

Un lenguaje de programación OO se diseña para seleccionar automáticamente el método correcto para implementar la operación, a partir de los datos asociados con la operación como parámetros, y el nombre de la clase de objeto, esto es llamado **enlace dinámico** (**dinamic binding**). El enlace dinámico, permite que las entidades del programa puedan referenciar en tiempo de ejecución a objetos de diferentes clases y está íntimamente relacionado con el concepto de herencia. En el ejemplo del frenado, el objeto seleccionará el método de frenado apropiado, basado en los parámetros que describen el tipo de frenos.

El polimorfismo permite agregar nuevas clases sin cambiar el código existente. Así, un nuevo tipo de transporte, como un camión puede agregarse con facilidad a la jerarquía. Sin embargo, hay que señalar que será necesario revisar los métodos.

1.3.8 Persistencia

La última propiedad asociada con sistemas OO es la **persistencia**, que es la *capacidad del nombre, estado y comportamiento de un objeto para trascender el espacio o el tiempo*. En otras palabras, el nombre, estado y comportamiento de un objeto se pueden conservar aún cuando el objeto es transformado o destruido.

Una vez almacenado en memoria secundaria, se puede reconstruir para usarlo durante la ejecución (materialización del objeto).

Por ejemplo, se puede desear guardar información del mantenimiento de un auto, para poder comparar los ajustes y modificaciones que se le han realizado. En este caso, el objeto persiste aun cuando sus atributos se transformen.

1.3.9 Otros Conceptos

El paradigma orientado a objetos presenta diferente terminología según la etapa del ciclo de vida que se contempla.

Así, en las fases de análisis (análisis orientado a objetos-AOO) y de diseño (diseño orientado a objetos-DOO), se utilizan los conceptos: Identificación del Objeto, Identificación de las Estructuras y Materias, Definición de Elementos, Definición de Atributos, Conexión de Mensajes, Instancias, Definición de Servicios.

Mientras que en la etapa de programación (programación orientada a objetos-POO), los términos a utilizar son: Tipo de Dato Abstracto, Objeto, Mensaje, Clase, Sobrecarga, Enlace Dinámico, Polimorfismo, Herencia, Variables de Clase o de Instancia, Métodos, Constructores, Destructores, Genericidad, Aserción, Invariante, etc.

Todos estos términos son metáforas del paradigma, y configuran el lenguaje o "jerga" que utiliza el ingeniero informático en el diseño y desarrollo de software aplicando esta metodología.

Las Ciencias de la Computación están llenas de metáforas que han llegado a trascender el círculo especializado del informático y son utilizadas de forma rutinaria por la inmensa mayoría del público, debido a la incorporación de la computadora en la actividad diaria. Dos excelentes ejemplos son los términos "memoria" y "ventanas". Estas palabras presentan poca semejanza real con los conceptos que representan en el lenguaje habitual. No obstante, los términos informáticos han quedado acuñados y los utilizamos indiscriminadamente, aparentemente sin gran dificultad.

El Paradigma Orientado a Objetos se describe a menudo usando el concepto Objeto/ Mensaje, en el que cada objeto (elemento autónomo de información creado en tiempo de ejecución) es solicitado para realizar un determinado servicio el usuario mediante el envío a ese objeto del mensaje apropiado. El solicitante no precisa conocer cómo el objeto proporciona el servicio pedido, la implementación es interna al objeto y la gestiona el suministrador del objeto. El énfasis se produce en qué se puede obtener, más que en cómo se obtiene.

1.4 Lenguajes Orientados a Objetos

Los lenguajes de programación orientados a objetos (LOO) tienen dos vertientes, aquellos que nacieron siendo **orientados a objetos desde su origen** como Smalltalk, Eiffel, Java, Phyton y aquellos que son **extensiones de lenguajes imperativos** como C++, Pascal OO, ADA OO, Object LISP, COBOL OO, estas extensiones no consideran a los objetos como entidades activas que reciben mensajes, más bien invocan procedimientos o funciones que pasan objetos; no obstante, incorporan las características más relevantes de POO, sin embargo, en todos ellos se emplea el concepto de clase (class) o tipo de dato abstracto para crear objetos.

En realidad, muchos de los conceptos creados con la programación estructurada como las estructuras de control, la modularidad, la definición de tipos (o estructuras) de datos y la definición del alcance de las variables persisten en estos lenguajes, agregando la mayoría de las características citadas en el punto anterior.

En algunos LOO como Smalltalk y Phyton no hacen chequeo estático de tipos, sino que los tipos son verificados en tiempo de ejecución. Esto es menos restrictivo durante la compilación, pero impone una sobrecarga en tiempo de ejecución, ya que durante esta se determina si un objeto puedo o no dar respuesta a un mensaje; además, esto permite que se generen errores difíciles de detectar al momento de ejecución.

Algunos LOO son llamados **híbridos**, como C++ y Java, en estos lenguajes, casi todo es un objeto, pero existen elementos que no son objetos, como los tipos de datos primitivos. En cambio, los LOO **puros**, todo es un objeto.

En la mayoría de los casos, los LOO todavía no cuentan con soporte adecuado para mantener la persistencia de los objetos de manera transparente al usuario, todavía es responsabilidad del programador proveer a los sistemas de persistencia para los objetos que lo requieran.

Existe una clasificación de los lenguajes relacionados con objetos:

- **Lenguajes basados en objetos**, son los que soportan encapsulamiento e identidad del objeto; es decir, los métodos y atributos se ocultan dentro y se hacen privados a los objetos, y estos objetos tienen identificadores únicos. Estos lenguajes tienen poco o ningún soporte para las clases, en el sentido de que no hay soporte para la abstracción en conjunto. En otras palabras, los objetos no pertenecen a las clases abstractas, las cuales tienen una identidad independiente. Además, no hay herencia. El lenguaje Ada original es típico de este tipo.
- **Lenguajes basados en clases**, como CLU, incluyen la noción de abstracción en conjunto a nivel instancia/clase, pero no soportan la herencia entre clases abstractas, las cuales no pueden tener instancias concretas. Los lenguajes basados en clases incluyen todas las características de los lenguajes basados en objetos.
- **Los lenguajes orientados a objetos** se caracterizan porque toman las características de los lenguajes basados en objetos y de los basados en clases y tienen además herencia completa entre clases e instancias y entre clases y clases o, lo que es lo mismo, las instancias y las clases heredan los métodos y atributos de las clases a las que pertenecen.

Un lenguaje es orientado a objetos si, y sólo si satisface los siguientes requisitos:

- **Apoya los tipos de datos abstractos**. Los TDA se basan en las clases de datos diseñadas previamente. Cada TDA protege los datos del uso inadecuado, al ofrecer una cantidad de operaciones permisibles. Para esto, tanto la representación de los datos como las operaciones permisibles están protegidos ocultando los detalles de su implementación. Si un lenguaje de programación está orientados a los tipos de datos abstractos, los tipos identificados para un sistema de aplicación se asocian de manera directa con las unidades de código; el abismo entre el concepto y la realidad se reduce en gran medida.
- **Soporta la selección de métodos**. Al seleccionar el método, el usuario sólo debe especificar la operación a aplicarse a cada objeto (o en lenguajes más amplios, a uno o más registros). Así, el sistema elegirá el método adecuado para los parámetros especificados. En otras palabras, el usuario sólo debe especificar lo que debe hacerse y el selector del método determina cómo debe aplicarse. El polimorfismo es una de las aplicaciones más comunes de la selección de método.
- **Soporta la herencia de tipo**. Con la herencia de tipo, los sistemas se amplían y afinan por medio de componentes de tipos ya existentes. La herencia de tipo permite construir los sistemas a partir de jerarquías de tipo ya existentes; proporciona los mecanismos para la construcción y reutilización del software. De esta forma, no es necesario reescribir el código, sólo aquella parte que es diferente. La herencia impone un mecanismo en los TDA que reduce en gran medida la complejidad de los sistemas resultantes.

En concreto, podemos señalar que el entorno de programación orientado a objetos es un entorno:

- Procedimental, que incorpora el concepto de encapsulación de datos.
- Extensible, ya que permite ampliar los componentes software sin necesidad de reprogramar todo el componente.
- Reutilizable. por permitir crear un programa en base a una serie de componentes definidos en bibliotecas de componentes.
- Interactivo a través de una interfaz gráfica de usuario que proporciona la funcionalidad necesaria en materia de edición, puesta a punto de programas y archivo de información.

1.5 Campos de Aplicación

Al igual que el paradigma imperativo, el paradigma orientado a objetos puede emplearse para resolver casi cualquier tipo de problema. El mismo tipo de problemas resueltos con el paradigma imperativo encuentra una solución más fácil y ordenada en el paradigma orientado a objetos, sin embargo, este paradigma es particularmente útil para desarrollo de aplicaciones con manejo de interfaces de usuario, imágenes, sonido (multimedia) y bases de datos multimedia, sistemas de tiempo real, simulación y modelado, BDOO, hipertexto e hipermedia, IA y sistemas expertos, redes neuronales, programación paralela y CAD/CAM.

La tecnología orientada a objetos como herramienta para analizar, diseñar e implementar aplicaciones permite obtener aplicaciones más fáciles de modificar y extender a partir de componentes reutilizables, esto disminuye el tiempo de desarrollo y aumenta la confiabilidad de los sistemas, estas características son particularmente importantes al momento de crear, de modo más sencillo, aplicaciones muy grandes y complejas.

Tarea.

Leer en otras fuentes sobre los temas vistos en esta unidad, hacer un resumen de la unidad (máximo 1 cuartilla) y explicar cuáles son las diferencias entre la programación orientada a objetos y la estructurada (no olvidar conclusiones y bibliografía).
Explicar:

1. En qué consiste el paradigma orientado a objetos.
2. Qué es la orientación a objetos.
3. Cuáles consideras que son las tres ventajas principales de la orientación a objetos.
4. Cuáles son los principios fundamentales del paradigma orientado a objetos.
5. Qué es la modularidad, cómo se logra en el paradigma orientado a objetos y en que difiere del paradigma imperativo.
6. Que es la abstracción y para que se emplea en el paradigma orientado a objetos.
7. En programación OO qué es un objeto y cuáles son los elementos que lo componen.
8. Qué es la persistencia de un objeto.
9. Cuál es la diferencia entre un LOO híbrido y uno puro.
10. Cuál es la diferencia entre los lenguajes basados en objetos, los lenguajes basados en clases y los orientados a objetos.
11. Cuáles son sus campos de aplicación.