**UNIVERSIDAD TECNOLÓGICA DEL SUROESTE DE GUANAJUATO**

**Nombre y firma del Alumno:**

**David Mendoza Aguirre**

**Esmeralda Morales Conejo**

**Materia:**

**“Programación de Aplicaciones”**

**Carrera:**

**Tecnologías de la Información y Comunicación**

**Nombre del profesor(a):**

**MI. Ariana Gómez Constreras**

**Fecha:**

**Viernes 23 de agosto de 2019**

**Introducción**

La programación orientada a objetos de basa en el hecho de que se debe dividir el programa, no en tareas, si no en modelo de objetos físicos o simulados, si se escribe un programa en un lenguaje orientado a objetos, se está creando un modelo de alguna parte del mundo, es decir, se expresa un programa como un conjunto de objetos que colaboran entre ellos para realizar tarea.

La POO difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada sólo se escriben funciones que procesan datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

**Objeto**

Un objeto es algo que se visualiza, se utiliza y que juega un papel o un rol en el dominio del problema del programa. La estructura interna y el comportamiento de un objeto, en consecuencia, no es prioritario durante el modelado del problema.

En otras palabras, un objeto no es más que un conjunto de atributos (variables o datos) y métodos (o funciones) relacionados entre sí.

**Identificación de un objeto**

Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

En comparación con un lenguaje imperativo, una "variable", no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto

**Clase**

Una clase es la implementación de un tipo abstracto de datos y describe no solo los atributos (datos) de un objeto sino también sus operaciones (comportamiento).

Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

**Instancia**

Una instancia es un elemento de una clase (un objeto). Cada uno de los objetos o instancias tiene su propia copia de las variables definidas en la clase de la cual son instanciados y comparten la misma implementación de los métodos. Sin embargo, cada objeto asigna valores a sus atributos y es totalmente independiente de los demás.

**Métodos**

Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

la declaración hay que indicar cuál es el tipo del parámetro que devolverá el método o *void* en caso de que no devuelva nada.

También se debe especificar el tipo y nombre de cada uno de los parámetros o argumentos del método entre paréntesis. Si un método no tiene parámetros el paréntesis queda vacío (no es necesario escribir *void*). Los parámetros de los métodos son variables locales a los métodos y existen sólo en el interior de estos.

**Sobrecarga de métodos**

La sobrecarga de métodos permite usar el mismo nombre de un método en una clase, pero con diferentes argumentos (y, opcionalmente, con diferentes valores de retorno). Los métodos sobrecargados hacen más cómoda la implementación de un objeto y, por ende, la utilización de los métodos del mismo.

Los métodos sobrecargados:

* Deben tener el mismo nombre.
* Deben cambiar la lista de argumentos.
* Pueden cambiar el valor de retorno
* Permiten cambiar el nivel de acceso

**Mensajes**

Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

La acción puede ser el envío de otros mensajes, el cambio de su estado, o la ejecución de cualquier otra tarea que se requiera que haga el objeto. Los objetos de un programa interactúan y se comunican entre ellos por medio de mensajes.

**Constructores**

Un constructor es un método que tiene el mismo nombre que la clase y cuyo propósito es inicializar los atributos de un nuevo objeto. Se ejecuta automáticamente cuando se crea un objeto o instancia de la clase.

Las reglas para los constructores son:

* El constructor tiene el mismo nombre que la clase.
* Puede tener cero o más parámetros.
* No devuelve ningún valor (ni si quiera void).
* Toda clase debe tener al menos un constructor.

**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, unas variedades 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**

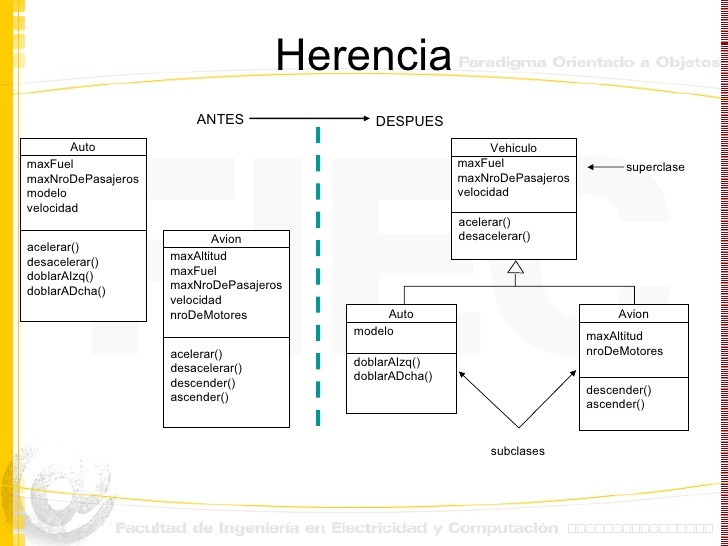
Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear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

La encapsulación describe el hecho de que los objetos se usan como cajas negras. Así, un objeto encapsula datos y métodos, que están dentro del objeto.

Interfaz pública de una clase: Es el conjunto de métodos (métodos de servicio) que sirve para que los objetos de una clase proporcionen sus servicios. Estos servicios son los que pueden ser invocados por un cliente.  
  
Métodos de soporte: Métodos adicionales en un objeto que no definen un servicio utilizable por un cliente, pero que ayudan a otros métodos en sus tareas.  
  
La encapsulación es un mecanismo de control. El estado (el conjunto de propiedades, atributos o datos) de un objeto sólo debe ser modificado por medio de los métodos del propio objeto.

**Herencia**

Las clases no están 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.



**Buenas practicas**

**Definición de variables por default.**

*string var1= String.empty;  
int var2 = 0;  
boolean = true;*

La declaración de las variables locales de un método, deberá realizarse en el bloque superior de éste. Todas las variables deben estar declaradas en el mismo lugar y deben ser inicializadas para limpiar su valor predeterminado.

**Declaración de variables**

Declaración de variables C# tipos definidos, ya que muchas veces usamos los tipos definidos en las clases de Framework de .NET

*int var1;  
string var2;  
object var3;*

No usar

Int, String, Object, Boolean.

Utilizar prefijos como “is” o similares al nombrar variables de tipo bool.

*private bool isFinished;*

**Comparaciones en C#**

Es preferible utilizar las clases para manejo de string como se define a continuación:

*string.Equals(var)*

*string.Contrains(Var)*

\* Siempre que se comparan se recomienda hacerlos en mayúsculas o minúsculas.

**Métodos y propiedades**

Evitar métodos y propiedades públicas, a menos que sea estrictamente necesario accederlas desde afuera de la clase. Utilizar el keyword internal si estos miembros deben ser accedidos desde dentro del mismo assembly.

**Abrir y cerrar conexiones**

Al abrir conexiones a bases de datos, Sockets, Streams, etc. siempre cerrarlos en los bloques finally. Esto asegurará que aún ante la eventualidad de una excepción, estos accesos serán cerrados. Se puede usar el Using en caso de no tener que capturar una excepción en particular dentro del catch.

**Declaración de clases**

Utilizar Pascal Casing para declarar nombres de clases.  
Utilizar el prefijo “I” con Pascal Casing para nomenclar Interfaces ( Ejemplo: IUser).

**Manejo de excepciones**

Nunca “capturar una excepción y no hacer nada”. Si se oculta una excepción, nunca se sabrá si la excepción sucedido. Muchos desarrolladores utilizan este método para ignorar errores no significativos. Trate de evitar las excepciones comprobando todas las condiciones de error mediante programación. En el peor de los casos, usted debe registrar la excepción en un log y continuar.