OSCAR ERNESTO TRUJILLO

ID **UB210120**

NOMBRE DEL CURSO: PROGRAMACIÓN ORIENTADA A OBJETOS

FECHA: 04 DE FEBRERO DE 2013

LUGAR: PANAMA

**ATLANTIC INTERNATIONAL UNIVERSITY** INDICE

## Introducción………………………………………………………………… 3

1. Concepto……………………………….. ………………………………….. 5
2. Los paradigmas de la programación. …………………………………………… 5
3. Conceptos fundamentales**…………**…………………………………………… 7
4. Características de la programación orienta a objetos…………………………….. 9
5. Historia de los lenguajes de programación orientados a objetos …………………. 10
6. Aplicación…………………………………………………………………………. 11
7. Conclusiones………………………………………………………………………. 15
8. Bibliografía………………………………………………………………………… 16

## INTRODUCCIÓN

En un mundo globalizado, con los avances tecnológicos adelantando día con día, don la filosofía es hacer las cosas de una manera que sean más practicas para la humanidad, donde el control es importante para los resultados que se quieren y lo más importante que no implique mayor consto o dicho de otra forma, que podamos reducir los costos en cualquiera de los aspectos que se manejan el día con día, surge una nueva manera de cómo programar llamada PROGRAMACIÓN ORIENTADA A OBJETOS.

Este tipo de programación surge como una alternativa más viable y de menor costo en comparación a las programaciones existentes, nos permite tener muchas variables y elementos dentro los cuales se pueden ser llamados en cualquier parte del programa en el cual se le requiera. El comportamiento de dichos elementos es como tener un grupo de objetos los cuales se pueden mover a cualquier parte que uno desee y nunca perderá sus características, forma, esencia, y es este la idea principal de este tipo de programación.

En el siguiente en ensayo estudiaremos los aspectos principales de la programación orientada a objetos, como se compone dicha programación y como está estructurada, para tener una mejor idea de su comportamiento, también describiremos las estructuras principales de la programación ya traducidas a un lengua, teniendo esta visión, veremos qué tipo de programas utilizan esta manera de ver la programación y daremos un pequeño ejemplo aplicado a la realidad en uno de los lenguajes mas usados.

## CONCEPTO

La programación orientada a objetos es una extensión natural de la actual tecnología de programación, y representa un enfoque nuevo y distinto al tradicional. Al igual que cualquier otro programa, el diseño de un programa orientado a objetos tiene lugar durante la fase de diseño del ciclo de vida de desarrollo de software.

El diseño de un programa 00 es único en el sentido de que se organiza en función de los objetos que manipulará De hecho, probablemente la parte más difícil de la creación de software orientado a objetos es identificar las clases necesarias y el modo en que interactúan entre sí Desgraciadamente, no hay reglas fáciles para determinar las clases de un programa dado. La identificación de clases puede ser tanto arte como ciencia El proceso es algo impreciso, y por esta causa han surgido numerosos métodos que proporcionan reglas para la identificación de clases y las relaciones que existen entre ellas; estos métodos son los citados anteriormente.

La década de los noventa será, sin lugar a dudas, la década de la programación orientada a objetos Como Rentsch predijo, «la programación orientada a objetos será en los ochenta lo que la programación estructurada fue en la década de los setenta ». En la actualidad la programación orientada a objetos se ha hecho enormemente popular Escritores y diseñadores de software, junto a compañías importantes en el campo del software, se dedican de modo continuo a producir compiladores de lenguajes, sistemas operativos, bases de datos, etc, orientados a objetos.

¿Qué es la programación orientada a objetos? ¿Por qué es tan popular? La programación orientada a objetos es algo más que una colección de lenguajes de programación, tales como Smalltalk, Object Pascal, C++, etc . Se podría decir que este tipo de programación, es un nuevo medio de pensar sobre lo que significa computar (computadorizar), es decir, cómo se puede estructurar información en un computador.

COMENTARIO: La programación orientada a objetos además de ser un grupo de programas y es una filosofía de programación, en donde puedo tomar elementos aislados dentro de un programa darles características especiales y poderlos llamar en cualquier parte del mismo, cuestión que era muy difícil en otros tipos de programación, ya que se tenía que copiar todos el código de lo que se quería hacer cada vez que se necesitara.

## LOS PARADIGMAS DE LA PROGRAMACIÓN

Cuando usamos una computadora, lo que estamos buscando es simplemente una solución a un problema, y para que resuelva dicho problema debemos decirle cómo hacerlo, ya que, en rigor de la verdad, las computadoras no pueden más que sumar **bits** y mover **bytes** de un lugar a otro. Nuestra tarea como programadores es, entonces, indicarle a la computadora qué es lo que queremos que haga; para ello debemos utilizar un lenguaje particular.

El lenguaje que entiende una computadora se denomina **binario**, pero como éste es difícil de leer y escribir para nosotros, debemos usar un lenguaje intermedio que luego será traducido a binario. El lenguaje, entonces, es una herramienta que nos servirá para indicarle a la computadora qué pasos debe seguir para resolver el problema en cuestión. Ahora bien, el modo en que especifiquemos la solución dependerá del paradigma de programación que usemos.

Dicho paradigma no es más que un modelo que representa un enfoque particular para la construcción de sistemas. No hay uno mejor que otro, sino que cada uno tiene ventajas y desventajas. Por otro lado, hay situaciones en que un paradigma resulta más adecuado que otro. El paradigma que vamos a desarrollar en el presente libro es el de **Programación Orientada a Objetos** (POO).

## CONCEPTOS FUNDAMENTALES

La programación orientada a objetos es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

[**Clase**](http://es.wikipedia.org/wiki/Clase_(inform%C3%A1tica)): 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.

[**Herencia**](http://es.wikipedia.org/wiki/Herencia_(inform%C3%A1tica))**:** (por ejemplo, herencia de la clase C a la clase D) es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables publicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. Esto es así para mantener hegemónico el ideal de OOP.

[**Objeto**](http://es.wikipedia.org/wiki/Objetos_(programaci%C3%B3n_orientada_a_objetos)): entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa). Es una instancia a una clase.

[**Método**](http://es.wikipedia.org/wiki/M%C3%A9todo_(programaci%C3%B3n_orientada_a_objetos)): 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.

**Evento**: es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

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

**Propiedad o atributo**: contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

**Estado interno**: es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

**Componentes de un objeto**: atributos, identidad, relaciones y métodos.

**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.

## CARACTERISTICAS DE LA PROGRAMACIÓN ORIENTA A OBJETOS

La programación orientada a objetos puede describirse como el conjunto de disciplinas que desarrollan y modelizan software que facilitan la construcción de sistemas complejos a partir de componentes.

El atractivo intuitivo de la orientación a objetos es que proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible. Las ventajas de la orientación a objetos son muchas en programación y modelación de datos.

La programación orientada a objetos permite una representación más directa del modelo de mundo real en el código El resultado es que la transformación radical de los requisitos del sistema (definido en términos de usuario) a la especificación del sistema (definido en términos de computador) se reduce considerablemente.

El problema Utilizando técnicas convencionales, el código generado para un problema del mundo real consta de una primera codificación del problema y a continuación la transformación del problema en términos de un lenguaje de computador. Las disciplinas y técnicas orientadas a objetos manipulan la transformación automáticamente, de modo que el volumen de código codifica el problema y la transformación se minimiza. De hecho, cuando se compara con estilos de programación convencionales (procedimentales por procedimientos), las reducciones de código van desde un 40 por 100 hasta un orden de magnitud elevado cuando se adopta un estilo de programación orientado a objetos.

Se necesita un nuevo enfoque para construir software en la actualidad. Este nuevo enfoque debe ser capaz de manipular tanto sistemas grandes como pequeños y debe crear sistemas fiables que sean flexibles, mantenibles y capaces de evolucionar para cumplir las necesidades de cambio La tecnología orientada a objetos puede cubrir estos cambios y algunos otros más en el futuro.

La orientación a objetos trata de cumplir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos software. Estas tareas se realizan mediante la modelización del mundo rea. Los cinco elementos (propiedades) más importantes de este modelo son:

* Abstracción
* Encapsulación
* Principio de ocultación
* Poliformismo
* Herencia

[**Abstracción**](http://es.wikipedia.org/wiki/Abstracci%C3%B3n_%28programaci%C3%B3n_orientada_a_objetos%29): 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.

[**Encapsulamiento**](http://es.wikipedia.org/wiki/Encapsulamiento_%28programaci%C3%B3n_orientada_a_objetos%29): Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la [cohesión](http://es.wikipedia.org/wiki/Cohesi%C3%B3n) de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

[**Principio de ocultación**](http://es.wikipedia.org/wiki/Principio_de_ocultaci%C3%B3n): Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que especifica 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 pueden cambiar el estado interno de un objeto de maneras inesperadas, 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.

[**Polimorfismo**](http://es.wikipedia.org/wiki/Polimorfismo_%28programaci%C3%B3n_orientada_a_objetos%29): 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](http://es.wikipedia.org/wiki/Sobrecarga) de C++.

[**Herencia**](http://es.wikipedia.org/wiki/Herencia_%28programaci%C3%B3n_orientada_a_objetos%29): 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 reimplementar su comportamiento. 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; esta característica no está soportada por algunos lenguajes (como Java).

## HISTORIA DE LOS LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

El primer lenguaje de programación que introdujo el concepto de *clase* fue Simula-67, como entidad que contenía datos y las operaciones que manipulaban los datos. Asimismo, introdujo también el concepto de herencia.

El siguiente lenguaje orientado a objetos, y seguramente el más popular desde un enfoque conceptual exclusivamente de objetos, es Smalltalk, cuya primera versión comercial se desarrolló en 1976, y en 1980 se popularizó con la aparición de Smalltalk-80.

Entre los lenguajes orientados a objetos que se han desarrollado a partir de los ochenta destacan extensiones de lenguajes tradicionales tales como C++ y Objective-C (extensiones de C), Modula-2 y Object Pascal (extensión de Pascal) y recientemente Object Cobol.

Otro lenguaje orientado a objetos puros es Biffel, C1eado por Bertrand Meyer y que soporta todas las propiedades fundamentales de objetos. Hasta ahora no ha adquirido popularidad más que en ambientes universitarios y de

En los últimos años han aparecido lenguajes con soporte de objetos que cada vez se están popularizando más: Clipper 5-2, Visual BASIC, etc. De cualquier forma, existe un *rey* actual en los lenguajes Orientados a objetos: C + + La normalización por ANSI y A T& 1 de la versión 3 O Y las numerosas versiones de diferentes fabricantes, tales como Borland C++ *4.0j4* 5, Turbo

C++ *30j3* 1 Y 4 5, Microsoft *CjC++* 7.0, Visual C++ 1 *5j2,* Symantec *60j7.0,*etc, hacen que en la segunda mitad de los noventa será el lenguaje orientadoa objetos más popular y utilizado en el mundo de la programación.

## BENEFICIOS DE LAS TECNOLOGIAS DE OBJETOS (TO)

Una pregunta que hoy día se hacen muchos informáticos es: *¿Cuál es la razón para introducir métodos de* **POO** *en los procesos de desarrollo?* La principal razón,sin lugar a dudas, son los beneficios de dichas **POO**: aumento de la fiabilidad y productividad del desarrollador. La fiabilidad se puede mejorar, debido a que cada objeto es simplemente (una caja negra) con respecto a objetos externos con los que debe comunicarse Las estructuras de datos internos y métodos se pueden refinar sin afectar a otras partes de un sistema.

Los sistemas tradicionales, por otra parte, presentan con frecuencia efectos laterales no deseados. Las tecnologías de objetos ayudan a los desarrolladores a tratar la complejidad en el desarrollo del sistema. La productividad del desarrollador se puede mejorar, debido a que las clases de objetos se pueden hacer reutilizables de modo que cada subclase o instancia de un objeto puede utilizar el mismo código de programa para la clase Por otra parte, esta productividad también aumenta, debido a que existe una asociación más natural entre objetos del sistema y objetos del mundo real.

**COMENTARIO:** Dentro de los beneficios más grande que nos permite la programación orientada a objetos es la facilidad de su depuración luego de el proceso de desarrollo de cualquier programa, nos permite identificar rápidamente problemas de lógica dentro de los códigos y adicional a este permite menos tiempo ya sea en el mantenimiento o cambios que se tenga que hacer al programa, lo cual conlleva a un menor consto de mantenimiento, pudiendo competir con otros empresas las cuales no tienen esta metodología de programación.

## APLICACIÓN:

**Problemática:** Se requiere dentro de Fantastic dividir los clientes que visitan en nuestros casinos por tipos, los cuales son, Cliente VIP, Cliente Frecuente, Cliente normal, para tener un mejor control de su juego y de la atención por parte de la gente de la sala hacia ellos, por el momento se lleva pero manual y es bien difícil que un nuevo empleado los conozca inmediatamente.

**Propuesta de solución:** Hacer un programa para que guarde todos los datos de la persona y que se le pueda dar una clasificación en que tipo de cliente se encontrara, guardar estos datos en una base de datos y poder hacer consultas por medio del nombre y que los Slot puedan de esta manera identificar a que tipo de cliente se están enfrentando en cierto momento o en alguna problemática.

**Desarrollo:**

**Private Sub CrearBase(sBase As String)**

**Dim Db As Database**

**Dim Fd As Field**

**Dim Tb As New TableDef**

**Dim Idx As New Index**

**Dim i As Integer**

**Set Db = CreateDatabase(sBase, dbLangSpanish, dbVersion20)**

**Set Tb = Db.CreateTableDef("Clientes")**

**Set Fd = Tb.CreateField("NOMBRE", dbLong)**

**Fd.Attributes = dbAutoIncrField Or dbUpdatableField Or dbFixedField**

**Tb.Fields.Append Fd**

**'El resto de los campos**

**Set Fd = Tb.CreateField("APELLIDO", dbDate)**

**Tb.Fields.Append Fd**

**Set Fd = Tb.CreateField("DIRECCIÓN", dbDate)**

**Tb.Fields.Append Fd**

**Set Fd = Tb.CreateField("CEDULA", d dbDate)**

**Tb.Fields.Append Fd**

**Set Fd = Tb.CreateField("TELEFONO", dbDate)**

**Tb.Fields.Append Fd**

**Set Fd = Tb.CreateField("CORREO", dbDate)**

**Tb.Fields.Append Fd**

**Set Fd = Tb.CreateField("TIPO DE CLIENTE", dbDate)**

**Tb.Fields.Append Fd**

**Set Fd = Tb.CreateField("APUESTA PROMEDIO", dbDate)**

**Tb.Fields.Append Fd**

**Idx.Name = "PrimaryKey"**

**Idx.Unique = True**

**Idx.Primary = True**

**Idx.Fields = "NOMBRE"**

**Tb.Indexes.Append Idx**

**Db.TableDefs.Append Tb**

**Db.Close**

**MsgBox "Nueva base de datos " & sBase & " creada.", vbInformation**

**End Sub**

**Dim sSQL As String**

**Dim lID As Long**

**On Local Error Resume Next**

**If Len(sBusqueda) Then**

**sSQL = "SELECT \* FROM " & sTabla & " WHERE " & sBusqueda**

**Set RsBuscar = Db.OpenRecordset(sSQL, dbOpenSnapshot)**

**If Not RsBuscar.EOF Then**

**If buscAtras Then**

**RsBuscar.MoveLast**

**End If**

**lID = CLng(RsBuscar("NOMBRE"))**

**Else**

**lID = 0&**

**End If**

**Else**

**If buscAtras Then**

**RsBuscar.MovePrevious**

**Else**

**RsBuscar.MoveNext**

**End If**

**lID = CLng(RsBuscar("NOMBRE"))**

**End If**

**If (lID = 0&) Or (Err <> 0) Then**

**lID = 0&**

**If buscAtras Then**

**RsBuscar.MoveLast**

**RsBuscar.MoveNext**

**Else**

**RsBuscar.MoveFirst**

**RsBuscar.MovePrevious**

**End If**

**End If**

**Err = 0**

**BuscarEnBase = lNOMBRE**

**End Function**

**iFFAtras = True**

**If gsBuscar(sBuscar, cFFAc\_Buscar + cFFAc\_Atras, "Buscar datos") > cFFAc\_IDLE Then**

**sBuscar = Trim$(sBuscar)**

**If Len(sBuscar) Then**

**buscAtras = iFFAtras**

**'buscCompleta = iFFCompleta**

**YaEstoyAqui = True**

**LblStatus(1) = "Buscando " & sBuscar & "..."**

**DoEvents**

**qID = BuscarEnBase(Text1(ControlActual).DataField & " LIKE '" & sBuscar & "\*'")**

**If qID Then**

**Data1.Recordset.FindFirst "ID = " & CStr(qID)**

**If Data1.Recordset.NoMatch Then**

**qID = 0&**

**End If**

**End If**

**If qID = 0& Then**

**Beep**

**MsgBox "No se ha hallado el dato buscado en el campo: " \_**

**& Text1(ControlActual).DataField, vbOK + vbInformation, "Buscar"**

## CONCLUSIONES

* Una de las partes que dan el éxito a la Programación Orientada a Objetos, que más que un programa es una filosofía de programación.
* Una de las mayores ventajas que tiene es que nos reduce a los programadores las líneas de códigos, abonado que los programas como los Visual nos permiten hacer las cosas más fáciles el ahorro en la codificación nos da menor tiempo en la creación.
* Teniendo una menor cantidad de líneas de código se hace mucho más fácil en el momento de dar un mantenimiento, o cuando se van a realizar las pruebas para la implementación del software encontrar los problemas que usualmente de detectan en dicho momento.
* La filosofía de trabajar modularmente nos permite poder llamar en cualquier momento del programa un modulo y no tener que repetir toda la codificación, esto nos permite tener listados de códigos mucho más pequeños y poderlos revisar de manera fácil.
* Lo único que parece un poco enredado es a la hora de declarar las variables y darle los nombres a los módulos o sub-programas, ya que si se tiene poca experiencia en dicha filosofía queda un poco enredado y se comenten errores para que cumplan con todas las características de POO

## Bibliografía

* *PROGRAMACION ORIENTADA A OBJETOS*

***Luis Joyanes Aguilar***

*Director del Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software Facultad de Informática*

*Universidad Pontificia de Salamanca Campus Madrid*

* *http://www.joanballestermoragues.com/blog/ingenieria-software/analisis sistemas/programacion-orientada-a-objetos-*
* *http://www.pol.una.py/archivos/asi/paradigmas/01ConceptosOO.pdf*
* [*http://www.mcgraw-hill.es/bcv/guide/capitulo/8448148703.pdf*](http://www.mcgraw-hill.es/bcv/guide/capitulo/8448148703.pdf)
* [*http://frameworkphp.wordpress.com/2007/09/19/caracteristicas-de-la-programacion-orientada-a-objetos/*](http://frameworkphp.wordpress.com/2007/09/19/caracteristicas-de-la-programacion-orientada-a-objetos/)
* *http://es.wikipedia.org/wiki/Programaci%C3%B3n\_orientada\_a\_objetos*