Apuntes de Fundamentos de Programación.

FRANCISCO RÍOS ACOSTA
Instituto Tecnológico de la Laguna
Blvd. Revolución y calzada Cuauhtémoc s/n
Colonia centro
Torreón, Coah; México
Contacto : friosam@prodigy.net.mx

I N D I C E.

1 Conceptos básicos del modelo orientado a objetos.

La orientación a objetos es una forma de hacer frente a la comprensión y solución de problemas, usando modelos organizados a partir de conceptos del mundo real. Su pieza fundamental es el objeto, el cual combina en una sóla entidad, los datos que lo identifican y su comportamiento. En particular, nosotros utilizaremos la orientación a objetos para analizar, diseñar e implementar programas de computadora, es decir, vamos a efectuar programación orientada a objetos. Entonces, lo primero que tenemos que aprender si queremos hacer programas orientados a objetos, es el concepto de lo que es un objeto.

Objeto.- es "algo" que tiene sentido en el contexto de una aplicación. A nosotros nos servirá para 2 propósitos : (1) nos ayudarán a entender el mundo real cuando analicemos un problema, y (2) nos proporcionan una base –modelo- para su implementación en una computadora. Los objetos pueden ser de 2 tipos :

- *Concretos*
- *Conceptuales*

Ejemplos de *objetos concretos*.- una bicicleta, una manzana, una memoria USB, un archivo de computadora, un carro, un alumno, un edificio, una puerta, un profesor.
Ejemplos de *objetos conceptuales*.- un programa de computadora, una variable, una lectura, el amor, un pensamiento.

Atributo.- En el primer párrafo mencionamos que el objeto encapsula –agrupa en una sóla entidad- a los datos que lo identifican y a su comportamiento. A los datos que identifican a un objeto se les llama *atributos*. Un *atributo* es un valor mantenido por un objeto, por ejemplo, un alumno es un objeto cuyos atributos son el número de control, su nombre, y su calificación final. Observemos que un alumno puede tener otros atributos mas, pero si el contexto de la aplicación es el obtener una lista de calificaciones finales de un grupo de alumnos, los atributos antes mencionados serán los únicos que nos interesen.

Imaginemos que queremos construir una agenda con los datos de nuestros amigos. Entonces ¿ qué atributos interesarían de nuestros amigos?. Los atributos que podríamos pensar son : el `nombre`, su `teléfono`, su `dirección`, su `correo electrónico`.

Otro ejemplo es el caso de ciertas bicicletas en una bodega. Los atributos de una bicicleta podrían ser : `rodada`, `tipo` –montaña, de carreras-, `material de construcción`, `marca`, `no. de velocidades`.

Veamos el ejemplo de atributos para un objeto conceptual. El amor, sus atributos podrían ser : `número de amor`, `tipo` –de conveniencia, de estudiante, segundo aire, primera vista-, `intensidad`.

El caso de objetos que se refieran a figuras geométricas como los polígonos, por ejemplo el rectángulo, el triángulo, el pentágono. Los atributos que nos podrían interesar son : el `número de vértices`, el `color del borde`, el `color de relleno`.

Comportamiento.- Se refiere al conjunto de acciones o transformaciones que un objeto ejecuta o a las cuales está sujeto. Al comportamiento también se le denomina de otras maneras : *Operaciones* o *Métodos*.
Por ejemplo, el caso de los objetos alumno requieren de algunas acciones y/o transformaciones : `AsignarCalificación`, `Visualizar`, `LeerNoControl`, `LeerNombre`. Las 4 acciones y/o transformaciones representan el comportamiento de los objetos alumno. En el caso de `AsignarCalificacion`, `LeerNoControl` y `LeerNombre`, se refieren a transformaciones, ya que modificarán el valor de la calificación final, del número de comntrol y del nombre del alumno. `Visualizar` representa una acción ya que sólo toma los atributos del alumno para mostrarlos en pantalla o en una hoja impresa.

Para el ejemplo de las bicicletas, definiríamos la acción y/o transformaciones de `Mover`, `Ajustar`, `Aceitar`, `Reparar`, entre otras, de acuerdo a lo que queramos limitar el estudio.

Para el amor, algunas acciones serían : `Proporcionar`, `Limitar`, `Quitar`.

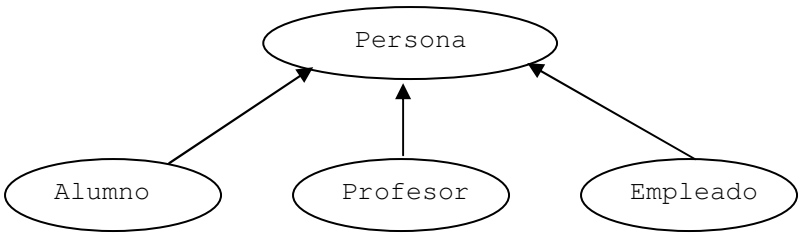
Resumiéndo lo anterior, podemos decir que cada vez que pensemos orientado a objetos, debemos enfocarnos a un proceso de identificación de objetos, de sus atributos y de su comportamiento –métodos-.

Características de los objetos.-

- *Identidad*. Se refiere a que cada objeto conserva de manera inherente su propia identidad. O sea, 2 objetos son distintos aún si el valor de todos sus atributos son idénticos. Por ejemplo, los 8 peones negros de un juego de ajedrez, son todos

negros, tienen las mismas dimensiones, textura, pero todos son diferentes, existen y tienen su propia identidad. Dos gotas de agua es otro ejemplo de la característica de identidad de los objetos.

- *Clasificación*. Se refiere a que los objetos con los mismos atributos y comportamiento –métodos-, son agrupados en *clases*. Cada objeto perteneciente a una clase, se dice que es una *instancia* de la clase. Así que una clase, representa a un posible conjunto infinito de objetos individuales. Por ejemplo a todos los alumnos que aparecerán en la lista de calificaciones finales, los clasificamos en la clase `Alumno`. A todos los amigos que registramos en nuestra agenda, los podemos clasificar en la clase *`Persona`*. Notemos que los alumnos también tienen dirección, teléfono y correo electrónico, pero no los agrupamos en la clase `Persona` ya que estos atributos para el caso de la aplicación de obtener la lista de calificaciones, no se requieren. La pregunta es ¿Los amigos podrían ser alumnos?. Discútelo con tus compañeros del grupo y con tu profesor. La *clase* describe en una sóla entidad a los atributos y comportamiento –métodos-, de todos los objetos que pertenezcan a ella.
- *Polimorfismo*. Es una característica de la orientación a objetos, que significa que un mismo método puede tener diferente manera de realizarse, en las diferentes clases que tengamos bajo estudio. Cada objeto perteneciente a una clase, "sabe como" ejecutar sus propios métodos. Cuando programamos orientado a objetos, el lenguaje de programación automáticamente selecciona el método correcto para efectuar una cierta acción o transformación sobre el objeto al que se aplica. Por ejemplo, si tenemos los objetos `bicicleta`, `carro`, `barco`, y les aplicamos la operación `Mover`, la acción se ejecuta de manera diferente para cada objeto.¿Por qué?. Otro ejemplo típico es el de los objetos de la clase `Figura`, digamos `círculo`, `rectángulo`, `triángulo`. Apliquémosles la acción de `Dibujar`, cada uno de ellos tendrá su propio método `Dibujar` definido, ya que la acción debe implementarse de manera diferente para cada objeto.
- *Herencia*. Es una de las características que hacen a la oriientación a objetos mas eficiente, mas poderosa. Se refiere a compartir atributos y métodos entre clases, que se relacionan de manera jerárquica. Un ejemplo sería definir la clase `Persona` con atributos : `nombre, edad` y `sexo`. Esta clase se denomina clase *base*, ya que de aquí podríamos definir otras clases `Alumno`, `Profesor`, `Empleado`, que heredan los atributos y métodos de la clase `Persona`. A estas 3 clases se les denominan clases *derivadas*. Entonces, en herencia las clases *base* proporcionan atributos y métodos a otras clases, las clases *derivadas*. Para visualizar la herencia, se construyen los *árboles de herencia*. A cada clase sea base o derivada se denota con un elipse, y la herencia se denota usando una flecha con dirección de la clase derivada hacia la clase base.



Si las clases derivadas tienen los atributos :

- clase `Alumno : número de control, calificación`,
- clase `Profesor` : `número de tarjeta, grado`,
- clase `Empleado` : `salario, nombre del puesto`.

Al aplicar la herencia, los objetos de las clases derivadas tendrán ahora además de sus atributos antes enumerados, los atributos que heredan de la clase base `Persona`.

- clase `Alumno : número de control, calificación`, **`nombre, edad, sexo`**
- clase `Profesor` : `número de tarjeta, grado`, **`nombre, edad, sexo`**
- clase `Empleado` : `salario, nombre del puesto`, **`nombre, edad, sexo`**

Lo mismo sucede con los métodos en la clase base `Persona`, que serían heredados por las clases derivadas `Alumno`, `Profesor`, `Empleado`. Supongamos que la clase Persona tiene definido los métodos `AsignaNombre`, `AsignaEdad`, `AsignaSexo`. Las clases derivadas también heredan a estos métodos, por lo tanto un objeto alumno puede utilizar estos métodos sin necesidad de redefinirlos en la clase `Alumno`. Por esto, la herencia cuando es utilizada se dice que estamos reusando código. El reuso de código es una característica muy poderosa en la orientación a objetos.

1.1 Reconocimiento de objetos y clases en el mundo real y la interacción entre ellos.

Ahora que ya conocemos los objetos, sus tipos, sus atributos, su comportamiento –métodos-, sus características, podemos seguir con el reconocimiento de ellos dentro de la especificación de problemas que deseamos solucionar aplicando la orientación a objetos. Veamos algunos ejemplos.

Ejemplo 1.1.1.
En un sistema de cómputo identifica a los objetos y clases que lo conforman. Agrega los atributos y comportamiento para cada clase que hayas reconocido.

Objeto	Clase	Atributos	Métodos
ratón	Mouse	tipo conexión dimension peso noBotones	Mover Limpiar Reparar Conectar
teclado	KeyBoard	noTeclas empotrado conexión tipo	Conectar Limpiar
touchpad	Mouse2	dimension noBotones	Deslizar Desplazar
pantalla	Monitor	tipo resolucion dimension marca	Limpiar Encender Apagar
impresora	Impresora	tipo puerto noPagPorMin marca	Limpiar Reparar Alimentar Alinear Probar

Agrega mas objetos, clases, atributos y métodos al ejercicio.

Ejemplo 1.1.2
Identifica los objetos, clases, atributos, métodos para un edificio del Tec Laguna.

Objeto	Clase	Atributos	Métodos
salon de clase	Aula	noParedes noPuertas noVentanas noMesaBancos noMesas noSillas dimension tipoLuz refrigeración	Limpiar Reparar Asignar Iluminar
mesaBancos	Silla	color paleta portaLibros	Pintar Usar Reparar Limpiar Mover
muro	Pared	material noVentanas color dimensión	Pintar Limpiar Tapizar
mingitorio	Baño	tipo color material automatico	Limpiar Usar Descargar Aromatizar

Ejemplo 1.1.3

Identifica los objetos, clases, atributos, métodos para un Departamento Académico del Tec Laguna.

Objeto	Clase	Atributos	Métodos
alumno	Alumno	noControl nombre cardex telefono nombreTutor	Inscribir Examinar RegAsistencia

Ejercicios propuestos.

1. Encuentra los objetos, clases, atributos y métodos en un juego de futbol.
2. Encuentra los objetos, clases, atributos y métodos en un curso de fundamentos de programación.
3. Encuentra los objetos, clases, atributos y métodos en tú recámara.
4. Encuentra los objetos, clases, atributos y métodos en un carro.
5. Encuentra los objetos, clases, atributos y métodos en una familia.

La interacción entre objetos se efectúa mediante *mensajes*. Un mensaje se envía a un objeto utilizando un método de la clase. En ocasiones un objeto envía un mensaje a otro objeto de la misma clase o de otra clase, a veces un objeto se envía asimismo el mensaje. Por ejemplo, un `chofer` –objeto- envía el mensaje `Avanzar` a un objeto `carro`. La forma del mensaje sería : `carro.Avanza`. Vemos que la sintaxis para construir un mensaje está compuesta de 2 partes : el nombre del objeto y el nombre del método. Ambos deben pertenecer a la clase, tanto el objeto como el método.

```
identificadorDelObjeto.Metodo
```

Otro mensaje podría ser el que usa el método `Arrancar`, `carro.Arrancar`. Los métodos reciben parámetros para caracterizar de manera definida a su acción. Por ejemplo, cuando usamos el mensaje `carro.Avanzar`, podemos enviar al método `Avanzar` un parámetro que indique la velocidad, de manera que ahora escribiríamos el mensaje de la siguiente forma :

`carro.Avanzar(primera)` ó
`carro.Avanzar(reversa)` ó
`carro.Avanzar(segunda)`, y así por el estilo.

Observemos que el parámetro se ha incluido entre paréntesis. Un método que no reciba parámetros, puede escribirse con los paréntesis vacios. Por ejemplo :

```
carro.Encender()
```

Cuando escribimos programas orientados a objetos, éstos están compuestos de mensajes.

1.2 La abstracción y el encapsulamiento como un proceso natural.

Hemos establecido que si usamos la orientación a objetos para enfrentar la solución de un problema, debemos pensar en reconocer objetos, sus atributos y sus métodos, terminando por agrupar a los objetos reconocidos en clases. Al proceso de identificar o reconocer objetos que existen en el contexto de una aplicación –análisis de un problema-, se le denomina proceso de abstracción.

Abstracción. consiste de una examinación colectiva de ciertos aspectos de un problema, con el objetivo de aislar aquellos aspectos que son importantes para algún propósito y suprimir aquellos que no lo son.

Características de la abstracción.

- Para un mismo problema pueden existir diferentes abstracciones, dependiendo del propósito.
- Todas las abstracciones son incompletas e inexactas.

- La abstracción permite limitar el universo de manera que podamos construir modelos.
- No existen abstracciones correctas, sólo existen abstracciones adecuadas o inadecuadas.

Encapsulamiento. este concepto se refiere a lo que ya habíamos comentado anteriormente, acerca de juntar –encapsular- en una sóla entidad –la clase-, tanto atributos como métodos. Sólo imaginate que ves a una muchacha que acapara tu atención, de inmediato ves en ella atributos que ella como objeto perteneciente a la clase `Mujer`, los contiene. Por ejemplo el color de sus ojos, el color de su tez, el largo de su cabello, la curvatura de sus … brazos, su nombre, su edad. También la clase a la que ella pertenece, encapsula el comportamiento, por ejemplo el `Sonreir`, `Caminar, Observar, Platicar, Estudiar` por decir algunos métodos.

Para entablar interacción con ella, deberás pensar –abstraer- mensajes. Por ejemplo, le enviarías para conocer su nombre el mensaje : `ellaMujer.ComoTeLLamas`. Para saber su edad tal vez deberías usar el mensaje `ellaMujer.RetornaTuEdad`.

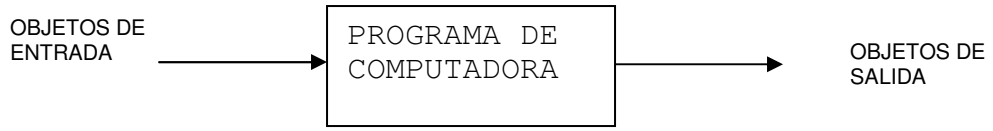
El *encapsulamiento* tiene otro objetivo en la orientación a objetos, que consiste en la separación de los aspectos externos de un objeto, los cuales son accesibles por otros objetos, de los detalles de implementación interna del objeto, los cuales se ocultan a otros objetos. Esto lo veremos mas tarde cuando enumeremos los beneficios de la orientación a objetos.

Ejercicio propuesto.

1. Construye mensajes para objetos de la clase Alumno, con atributos mencionados en las secciones anteriores.
2. Construye mensajes para objetos de la clase Arbitro de cualquier deporte.

1.3 La programación orientada a objetos y la complejidad del software.

Primero comentaremos que al referirnos a la programación orientada a objetos, estamos hablando de construir programas cuyas instrucciones manejan objetos de entrada, procesan mensajes que envían a los objetos, de manera que se produzcan objetos de salida.



Software es una palabra en inglés que significa programación, conjunto de programas, de acuerdo al contexto en el que se aplique. En este caso, la complejidad del software se refiere a lo difícil que resulta construir programas orientados a objetos. La programación orientada a objetos viene a sustituir a otra forma de construir programas, que durante mucho tiempo –desde 1973- fue la manera por mucho de escribir programas, la programación orientada a funciones.

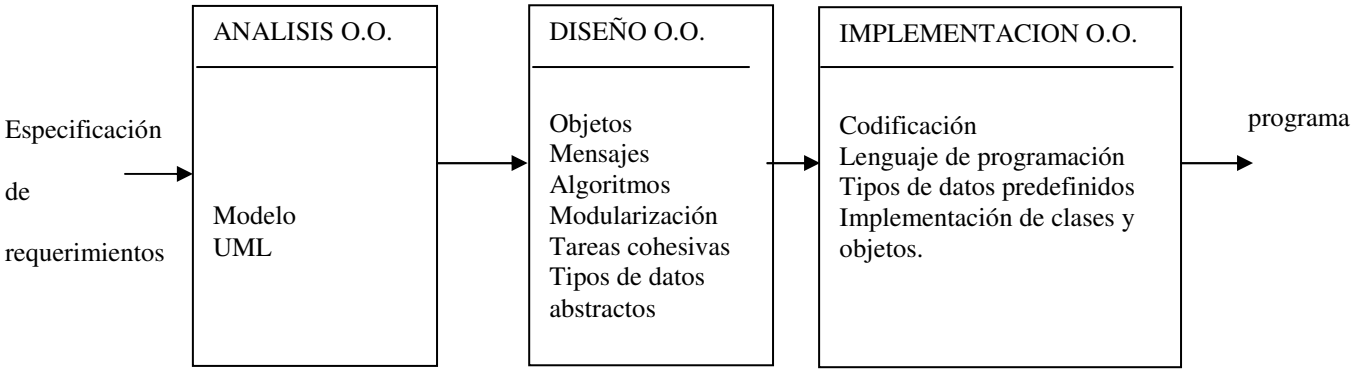
Así que la pregunta es, si la programación orientada a objetos disminuye la complejidad del software, es decir, si disminuye la dificultad de construir programas de computadora. Veamos que podemos decir acerca de dicha complejidad.

- En cualquier rama de la ciencia, la complejidad es atacada –disminuida- mediante la construcción de modelos. Cuando aplicamos la orientación a objetos para resolver un problema usando un programa de computadora, nuestros principales esfuerzos son dirigidos hacia la abstracción –limitar el universo- de objetos, las clases a las que pertenecen, identificación de sus atributos y métodos. Este reconocimiento de objetos es requerido para poder obtener los objetos de salida, luego de procesar los objetos de entrada. Al proceso de abstraer los objetos, clases, atributos y métodos tanto de objetos de entrada como de salida, se le conoce como *análisis orientado a objetos*. Existen algunas metodologías para efectuar el análisis orientado a objetos de un problema. Una de ellas muy popular en estos días, es el uso de diagramas del UML (lenguaje de modelado unificado).
- El hecho de efectuar un buen análisis orientado a objetos, permite un menor tiempo en la implementación de programas orientados a objetos. La orientación a objetos permite una mejor adaptibilidad a los cambios en los requerimientos de un programa, debido a que las clases encapsulan tanto atributos como métodos –operaciones-., de manera que si existiera un cambio en un objeto, como sus atributos y métodos están encapsulados en una clase, los cambios se realizarían sólo sobre atributos y métodos de ese objeto de forma que los efectos colaterales son más fáciles de manejar.

1.4 Conceptos del ciclo de vida del software.

Hagamos una restricción en las etapas del ciclo de vida del software y revisemos las 4 primeras :

- Especificación de requerimientos.
- Análisis orientado a objetos.
- Diseño orientado a objetos.
- Programación orientada a objetos.



Recordemos que software significa programas, de manera que estamos viendo las etapas que debemos seguir en la construcción de un programa –grandote o chiquito, complejo o fácil-. En este curso de fundamentos de programación, tendremos como principal objetivo aprender a programar, es decir, aprender a construir programas. Por lo tanto, es obvio que debemos de seguir estas etapas cuando construyamos nuestros programas de computadora. Así que no las olvidemos, debemos "talachearlas".

1.4.1. Especificación de requerimientos.

Es la primera etapa en el desarrollo de un programa. generalmente se consideran los siguientes aspectos :

- Ambito del problema.
- ¿Qué es necesario?.
- Contexto de la aplicación.
- ¿Qué se asume por cierto?.
- Necesidades de funcionamiento.

La especificación de requerimientos establece qué es lo vamos a hacer, no cómo lo vamos a hacer. Es un documento escrito en lenguaje natural donde se detallan las necesidades que requerimos cubrir en el programa. No es un documento donde se propone la solución del problema. Los requerimientos son las características que deberá cumplir el programa que construyamos siguiendo las etapas de análisis, diseño e implementación –codificación-.
La especificación de requerimientos es un documento muy importante al desarrollar un programa, ya que al terminar de realizar un programa una medida de la ***calidad*** de éste, es comprobar si el programa cumple con cada uno de los requisitos funcionales descritos en la especificación. Un programa tiene calidad si cumple con todos los requerimientos en el documento de especificación.

1.4.2 Análisis orientado a objetos.

Su propósito es modelar el problema a resolver de tal manera que pueda ser entendido. Si no construimos un modelo, entonces el análisis sería una especificación en lenguaje natural con inconsistencias, ambiguedades e incompleto. El modelo en nuestro caso sería un modelo orientado a objetos. Pero, ¿qué es un modelo?.
Modelo. es una abstracción de un problema con el propósito de entenderlo antes de construirlo. Para el caso del desarrollo de un programa orientado a objetos, el modelo especifica gráficamente a los objetos que intervienen en el ámbito de nuestra aplicación, de forma que también indica las clases en las que se agrupan los objetos, los atributos y los métodos para dichos objetos.
Características del análisis orientado a objetos :

- Su salida es un modelo orientado a objetos, incluyendo las clases, atributos y métodos.
- Clarifica los requerimientos definidos en la etapa previa de especificación.

- Proporciona una base formal para el acuerdo (acerca de lo que se requiere), entre el demandante del programa y el desarrollador del programa.
- Su resultado representa la comprensión del sistema que se requiere programar, de manera que sirva como una buena preparación para el diseño del programa.
- El modelo que se construye utiliza notaciones precisas.
- Permite tratar con la complejidad de un problema, de forma que pueda ser programado. En otras palabras reduce la complejidad al escribir un programa.
- El proceso de construir un modelo dentro del análisis, obliga al desarrollador a confrontar lo que no entiende acerca del dominio de una aplicación, al inicio del desarrollo de un programa.

Para construir modelos orientados a objetos existe un conjunto de diagramas que forman parte del *Lenguaje para Modelado Unificado* (UML en inglés). El UML consiste de un grupo de diagramas que nos ayudan en las diferentes etapas del desarrollo de un programa, no solamente en el análisis.

Los diagramas del UML son los siguientes :
- Diagramas de casos de uso.
- Diagramas de clase.
- Diagramas de actividad.
- Diagramas de secuencia.
- Diagramas de estado.
- Diagramas de componentes y de puesta en uso.

1.4.3 Diseño orientado a objetos.

Una vez que hemos analizado un problema con el fin de construir un programa que lo soluciones, seguimos con la etapa del diseño. Durante la etapa del diseño se especifica el CÓMO se va a construir el programa, a diferencia del análisis en donde se especifica el QUÉ es necesario hacer para construirlo.

En la etapa del diseño se decide la ARQUITECTUIRA DEL PROGRAMA, es decir su estructura y estilo. Esta arquitectura proporciona el contexto que servirá de base para efectuar decisiones mas detalladas en posteriores estados del diseño. La arquitectura del programa consiste en organizar el programa en módulos que contienen objetos. A este proceso se le denomina *modularización*.

Cada módulo contiene mensajes -*identidicadorDeObjeto.Metodo()*- entre objetos que pertenecen o no al módulo. Este conjunto de mensajes en que consiste el módulo, efectúan tareas simples, lógicas y completas. A estas tareas se les denominan tareas cohesivas.

Los diagramas de casos de uso del UML son utilizados en la etapa del diseño, representando una parte fundamental de base para conseguir la modularización del programa. En los diagramas de casos de uso se especifican las tareas que se deben realizar en cada módulo del programa. Los objetos que intervienen ya han sido previamente modelados en el análisis mediante la utilización de los diagramas de clase.

Las transformaciones de los objetos se detallan en los diagramas de estado, las cuales son descritas de manera íntegra en la etapa del diseño.

Otras fases en el diseño de objetos son las definiciones de las *interfases* y *algoritmos* utilizados para la implementación de los diferentes métodos y tareas previamente analizadas y descritas. Los algoritmos son escritos en lenguaje natural en neustro caso el español. Los tipos de datos empleados en la definición de las variables que intervienen en los algoritmos, se les llama tipos de datos abstractos. La etapa del diseño orientado a objetos también añade objetos internos para completar la implementación del programa, además de emplear refinamientos sucesivos de manera que se eficienticen las estrucuras de los objetos –atributos y métodos- y los algorimtos en cada módulo en los que se ha descompuesto el programa.

Los diagramas del UML que son fundamentales en la etapa de diseño para construcción de algoritmos y refinación de las clases, son los diagramas de actividad y diagramas de secuencia. estos 2 diagramas se relacionan fuertemente con las acciones que producen los cambios en los objetos definidos en los diagramas de estados del UML utilizados en la etapa del análisis.

1.4.4 Programación orientada a objetos.

Esta etapa es llamada de varias maneras : Implementación ó Codificación. En la etapa de diseño los algoritmos que realizan las tareas de cada módulo en que se dividió el programa, se escriben en lenguaje natural –para nosotros el español-. En cambio, en la etapa de implementación o codificación los algoritmos –programas- que cumplen con las tareas de cada módulo, son escritos –codificados- en el lenguaje de programación seleccionado. El lenguaje de programación que nosotros usaremos es el lenguaje **C#**.

Aquí los algoritmos de la etapa de diseño son codificados en sentencias propias del lenguaje de programación. Un lenguaje de programación tiene tipos predefinidos de datos, tiene sus sentencias y clases propias, de manera que es relativamente fácil definir clases y objetos que previamente habíamos abstraido en las etapas de análisis y diseño.

En el lenguaje C# una clase se define utilizando la palabra reservada **class**. Los atributos y métodos se encierran entre los caracteres { y }. Después de la palabra reservada **class** se agrega el nombre de la clase.

```
class Alumno
{
   // definición de atributos

  //  definición de métodos
}
```

Los atributos de un objeto que pertenece a una clase tienen siempre un tipo. Por ejemplo, la calificación de un alumno es un entero de 0 a 100, así que el tipo del atributo `calificacion` es de tipo entero –**int** en C#-. El nombre del alumno es una cadena de caracteres de manera que el tipo del atributo `nombre` es tipo cadena –**string** en C#-. Los tipos **int** y **string** son tipos de datos predefinidos en C#. Los tipos predefinidos son los tipos de datos básicos que un lenguaje tiene integrados en su ambiente de desarrollo, para que el programador los utilice sin restricción alguna.

1.5 Elementos primordiales en el modelo de objetos.

En las secciones anteriores ya hemos visto los conceptos de :

- *Abstración*, la cual consiste de una examinación colectiva de ciertos aspectos de un problema, con el objetivo de aislar aquellos aspectos que son importantes para algún propósito y suprimir aquellos que no lo son. Se utiliza para abstraer objetos, sus atributos y métodos, además de las clases a las que pertenecen los objetos.
- Encapsulamiento, que consiste en la separación de los aspectos externos de un objeto, los cuales son accesibles por otros objetos, de los detalles de implementación interna del objeto, los cuales se ocultan a otros objetos.
- Modularidad, consiste en organizar el programa en módulos que contienen objetos pertenecientes a determinadas clases, cuya interacción entre ellos desarrollan una tarea específica, lógica y completa. La suma de estos módulos y su correspondiente tarea que realizan, son la resolución del problema.
- Jerarquía y herencia, se refieren a compartir atributos y métodos entre clases, que se relacionan de manera jerárquica.
- Polimorfismo, es una característica de la orientación a objetos, que significa que un mismo método puede tener diferente manera de realizarse, en las diferentes clases que tengamos bajo estudio

Existe otro concepto que es muy importante : el **ocultamiento** de atributos. El ocultamiento de atributos está relacionado al concepto de encapsulamiento. Dado que en la clase es donde se concentran en una sóla entidad a los atributos y a los métodos, podríamos pensar ¿para qué se hace esto?.

Una regla de oro en la programación orientada a objetos es ocultar a los atributos del mundo externo a la clase, es decir, ocultar los atributos a otros objetos que no pertenecen a la clase, de manera que para accederlos ya sea para lectura, ya sea para escritura, el acceso sólo puede efectuarse mediante los métodos de la clase. A los métodos de la clase a menudo se les denomina interfase de la clase –comportamiento-. Un programador deberá saber los métodos definidos en la clase para poder acceder a los atributos de los objetos pertenecientes a esa clase. Por ejemplo, si queremos modificar la calificación de un alumno, deberíamos utilizar el método `AsignarCalificacion(califNueva)` que toma al parámetro `califNueva` y lo asigna al atributo `calificacion` del objeto involucrado en el mensaje. Digamos que el objeto le hemos llamado `oAlumno1`, entonces el mensaje sería :

`oAlumno1.AsignarCalificacion(100)` ó

`oAlumno1.AsignarCalificacion(califNueva)`

En el primer mensaje, se asigna un 100 al alumno, mientras que en el segundo mensaje se asigna el valor del parámetro `califNueva` al alumno.

1.6 Historia de los paradigmas en el desarrollo de software.

Sólo me referiré al paradigma de la *orientación a funciones*, ya que antecede a la orientación a objetos en uso y popularidad. Antes de escribir sobre la orientación a funciones, veremos el concepto de paradigma.

Paradigma.- representa una forma de abstraer los elementos que se encuentran en el contexto de un problema y que enfocándonos a ellos –analizándolos-, nos ayudan a construir el software –programa- que requerimos.

Desde 1973 en que Edgser Dijkstra contribuye con sus principios de programación estructurada, las metodologías propuestas en el mundo de la computación para desarrollo de programas que incluyen análisis y diseño, todas se basaron en la orientación a funciones.

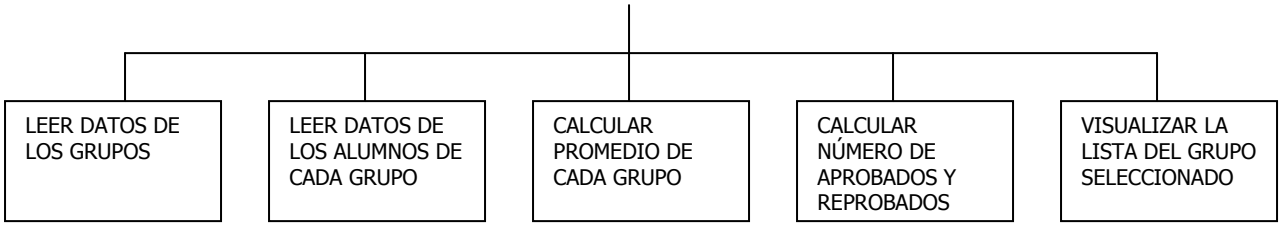
Orientación a funciones. indica que debemos descomponer funcionalmente un problema, de manera que obtengamos módulos que efectúen una función que puede ser desde compleja hasta simple. La modularización que conlleva la descomposición en funciones del problema a resolver, se debe llevar a cabo hasta obtener funciones simples, lógicas y completas. Una función con estas características es llamada función cohesiva.

Por ejemplo, un profesor requiere de obtener una lista de cada uno de sus grupos donde se indique la clave del grupo, el horario, el periodo, y el número de control, el nombre, y la calificación final de cada uno de los alumnos del grupo que se trate, además requiere del promedio del grupo, del número de alumnos aprobados y del número de alumnos reprobados..

La orientación a funciones nos indica que debemos descomponer en funciones este problema, que sumadas lo resuelvan. Entonces debemos efectuar el proceso de abstracción de dichas funciones, ordenando la ejecución de cada función de manera que se eviten indeterminaciones en los datos. La indeterminación de los datos es otro concepto manejado en la programación estructurada, consiste de evitar utilizar datos que previamente no hayan sido asignados o leídos. Para nuestro ejemplo, una función sería la visualizar los datos de los alumnos. Digamos que los datos de los alumnos del grupo FUNDAMENTOS DE PROGRAMACION no han sido leidos aún, ¿que salida en la visualización podríamos esperar?. Discútelo con tu profesor y tus compañeros. Es claro que estaríamos en un caso de indeterminación de datos, ¿de cuáles datos? , pues del número de control, nombre y calificación final de los alumnos del grupo FUNDAMENTOS DE PROGRAMACION. Una abstracción de las funciones que podrían resolver el problema del profesor son :

- Leer datos de los grupos.
- Leer datos de los alumnos de cada grupo
- Calcular promedio de cada grupo
- Calcular número de aprobados y de reprobados.
- Visualizar la lista del grupo seleccionado.

Generalmente en la orientación a funciones, la descomposición funcional es mostrada con rectángulos que denotan a las funciones de manera jerárquica.

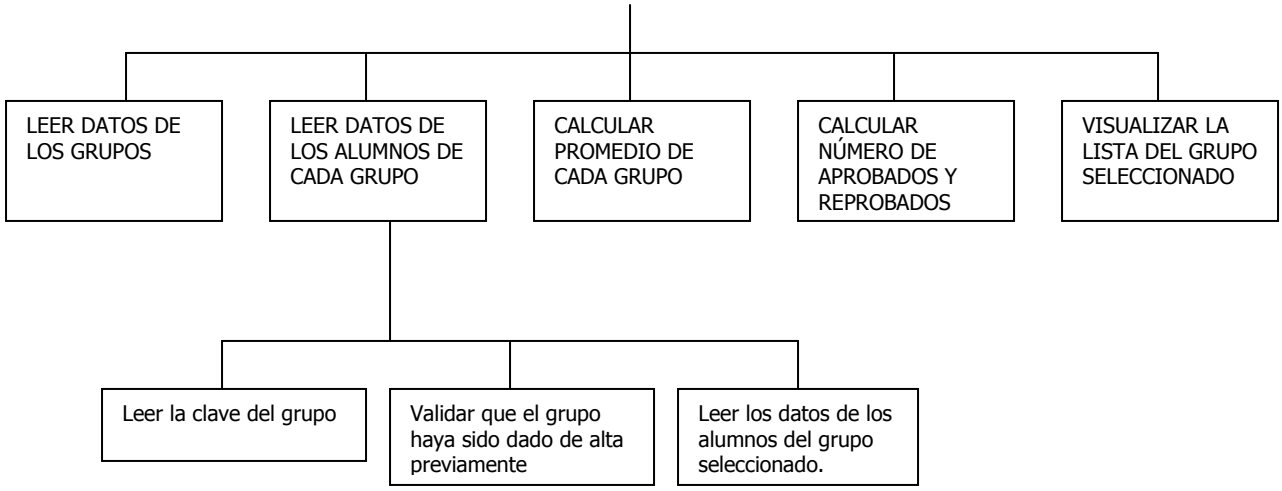


Para el punto de vista de un programador esta descomposición funcional –modularización- podría estar compuesta de sólo funciones cohesivas. Es decir, los módulos están tan obvios para él que ya no necesita seguir descomponiendo a ninguna de ellas. Otro programador podría no coincidir con este punto de vista y tratará de seguir descomponiendo funcionalmente, al módulo que el crea que la tarea que realiza se compone de otras tantas funciones.

Por ejemplo, digamos que el módulo LEER DATOS DE LOS ALUMNOS DE CADA GRUPO requiere de una mayor modularización, según lo indicamos enseguida :

- Leer la clave del grupo
- Validar que el grupo haya sido dado de alta previamente.
- Leer los datos de los alumnos del grupo seleccionado.

El diagrama de la descomposición funcional cambiaría a :



Estos refinamientos se realizan hasta que el analista o programador quede satisfecho con su modularización del problema.

1.7 Beneficios del modelo de objetos y de la P.O.O. sobre otros paradigmas.

Los beneficios del desarrollo orientado a objetos son sus características que hemos ya abordado, tales como :

- Clasificación.
- Herencia.
- Polimorfismo.
- Encapsulamiento.
- Ocultamiento.

En cuanto a la programación orientada a objetos podríamos enumerar :

- Sobrecarga de operadores y métodos.
- Constructores y destructores.
- Manejo de excepciones.
- Atributos y métodos estáticos.
- Recolectores de basura - Liberadores de memoria -.

Los beneficios en cuanto a desarrollo de sistemas :

- Facilidad en la traducción de requerimiento de negocios a módulos de código. Lo anterior es debido a que la orientación a objetos nos permite modelar nuestras aplicaciones basados en ideas de objetos del mundo real, de manera que frecuentemente identificamos una relación directa entre gente, cosas, conceptos, y las clases equivalentes que las modelan –representan-. Estas clases tienen las mismas propiedades y comportamientos tal como los conceptos en el mundo real que ellas representan, lo cual ayuda a identificar mas fácilmente –rápidamente- el código necesario a escribir, asi como las diferentes partes de la aplicación que requieren de interactuar entre ellas.
- Otro beneficio es el resuso de código. Frecuentemente necesitamos de los mismos tipos de dato en diferentes sitios de la misma aplicación. Por ejemplo, una aplicación referente a un hospital que requiera administrar los registros de sus pacientes, definitivamente necesita de una clase `Persona`. Existe además determinada gente involucrada en el cuidado de un paciente, como son los doctores, las enfermeras, administradores del hospital, gente de atención de asegurados, entre otros. Si definimos una clase genérica llamada `Persona`, que contenga todos las propiedades y métodos comunes a toda esta gente, podríamos tener una buena cantidad de código reusable que no siempre es posible usando la orientación

a objetos. Ahora imaginemos que podemos usar esta clase `Persona` en otros proyectos, el beneficio del reuso de código se extiende a otras dimensiones.

- La modularidad de las clases es otro beneficio. Si descubriéramos un error en la clase `Persona` o si desearamos agregar o cambiar los métodos en la clase, nuestro esfuerzo lo dirigimos claramente a la clase y objetos de la clase `Persona`. Toda la clase generalmente está en un solo módulo : un archivo. Cualquier proceso de la aplicación que tenga relación con la clase `Persona` será afectado inmediatamente por los cambios en ella. La identificación de estos procesos es fácil, de manera que los cambios necesarios los localizamos de forma simple, así que la búsqueda del código que debamos actualizar se reduce a una tarea sin muchas penas.