Programación Orientada a Objetos

U1: Introducción al paradigma de la programación orientada a objetos

LI. Jaime Jesús Delgado Meraz
j2deme+poo@gmail.com

Instituto Tecnológico de Ciudad Valles

Enero – Junio 2013

Introducción

- La programación orientada a objetos (POO), representa un intento de hacer programas que modelen más cercanamente la manera en que las personas piensan y se comportan en el mundo.
- En los antigüos lenguajes de programación, para resolver un problema, el programador debía identificar una tarea computacional que debía ser ejecutada para dar solución a dicho problema.
- Programar, entonces, consistía en encontrar una secuencia de instrucciones que completaran la dicha tarea.
- Pero en la programación orientada a objetos, en lugar de tareas, encontramos **objetos**:
  - Entidades que tienen comportamientos, que almacenan información y que pueden interactuar con otros objetos.

- Los objetos en el software pueden representar entidades reales o abstractas en el dominio del problema.
- Esto supone hacer el diseño del programa más natural y por lo tanto más fácil de hacer bien y de hacerlo entendible.
- Los lenguajes orientados a objetos incluyen un conjunto de caracterśticas que los hacen diferentes de un lenguaje estándar.
- Para hacer uso de estas características, debemos “orientar” nuestro pensamiento correctamente.

Orientación a Objetos

- La orientación a objetos es un conjunto de herramientas y métodos, que permiten a los ingenieros de software, construir sistemas de software confiables, amigables para el usuario, mantenibles, bien documentados y reusables que satisfagan los requerimientos de sus usuarios.
- Se dice, que la orientación a objetos, provee a los desarrolladores de software con nuevas herramientas mentales, de gran uso en la solución de una amplia variedad de problemas.

- La orientación a objetos, provee de una nueva vista de la computación. Un sistema de software es visto como una comunidad de *objetos* que cooperan entre sí, mediante el paso de *mensajes* para resolver un problema.
- Un lenguaje de programación orientado a objetos, debe proveer de soporte para los siguientes conceptos orientados a objetos:
  1. Objetos
  2. Clases
  3. Herencia
  4. Polimorfismo

Paradigmas de programación

- La programación orientada a objetos, es uno de múltiples *paradigmas* de programación.
- Otros paradigmas de programación comunes incluyen la programación imperativa (Pascal, C), la programación lógica (Prolog) y la programación funcional (ML, Haskell, Lisp).
- Los lenguajes lógico y funcional, se dice que son **declarativos**.

Definición

La palabra paradigma se define como un modelo, ejemplo, forma o estilo de hacer algo.

- Por lo tanto, podemos decir que un paradigma de programación es una manera de conceptualizar las maneras de realizar computo y como las tareas a realizar deben estructurse y organizarse.

- Podemos distinguir entre dos tipos de lenguajes de programación:
  - Lenguajes imperativos, que describen el conocimiento del "como",
  - Lenguajes declarativos, que describen el conocimiento del "que es".
- Un lenguaje es *declarativo* si describe como es algo, más que como crearlo. Este es un enfoque diferente de los lenguajes de programación imperativos tradicionales tales como Fortran y C, los cuales requieren que el programador especifique un algoritmo a ejecutar.
- En resumen, se podría decir que los programas imperativos hacen el algoritmo explícito y dejan la solución implícita, mientras que los programas declarativos hacen la solución explícita y dejan el algoritmo implícito.

- Los lenguajes imperativos requieren que escribamos paso a paso especificando como se debe hacer algo.
- Por ejemplo para calcular la función **factorial** en un lenguaje imperativo, se podría escribir algo como:

```
1 public int factorial(int n) {
2     int ans = 1;
3     for (int i = 2; i <= n; i++){
4         ans = ans * i;
5     }
6     return ans;
7 }
```

- En este caso, lo que se tiene es un procedimiento (serie de pasos), que al seguirse producen una resultado.

Programación Funcional

- La programación funcional es un paradigma de programación que trata la computación como la evaluación de funciones matemáticas.
- Este tipo de programación enfatiza la definición de funciones, en contraste a la programación procedural, que enfatiza la ejecución de instrucciones secuenciales.
- La siguiente es la función **factorial** en el lenguaje *Lisp*:

```
1 (defun factorial (n)
2     (if (<= n 1) 1 (* n (factorial (- n 1))))
3 )
```

- En este caso, se define la función factorial y no se desglosan los pasos para calcularla. El factorial de n esta definido como 1 si $n <= 1$, en cualquier otro caso $n * factorial(n - 1)$.

Programación Lógica

- Prolog (PROgramming in LOGic) es el lenguaje más utilizado en programación lógica. Esta basado en las ideas matemáticas de relaciones e inferencias lógicas.
- Prolog es un lenguaje declarativo, lo que significa que más que describir como computar una solución, un programa consiste en una base de datos de hechos y relaciones lógicas (reglas), que describen las relaciones que existen para la aplicación dada.
- La función **factorial** se escribe en Prolog mediante dos reglas simples:

```
1 fac(0,1).
2 fac(N,F) :- N > 0,
3     M is N - 1,
4     fac(M,Fm),
5     F is N * Fm.
```

- En los lenguajes *procedurales*, todo es un procedimiento.
- En los lenguajes *funcionales*, todo es una función.
- En los lenguajes *lógicos*, todo es una expresión lógica (predicado).
- En los lenguajes *orientados a objetos*, todo es un objeto.

El nuevo paradigma

- Se dice que las técnicas para resolver problemas en la programación orientada a obejtos modela de manera más cercada la manera en que los humanos resolvemos nuestros problemas del día a día.
- Consideremos un ejemplo: Supongamos que queremos enviar un arreglo de flores a una amiga llamada Robin que vive en otra ciudad. Para resolver el problema solo tendríamos que caminar a la florería más cercana, atendida por, digamos, Fred. Decirsle a Fred el tipo de flores a enviar y la dirección a donde deben entregarse. Y tendríamos por seguro que las flores se entregarán.

- Analizando los mecanismos usados para resolver el problema:
  - Primero encontramos un **agente** apropiado (Fred, en este caso) y le pasamos un **mensaje** con una petición.
  - Es entonces, la **responsabilidad** de Fred satisfacer la petición.
  - Existe alguna especie de **método** (un algoritmo o conjunto de operaciones) usado por Fred para satisfacer la petición.
  - No necesitamos conocer los métodos específicos usados para satisfacer la petición, tal información esta oculta a la vista.
- Por supuesto, no necesitamos saber los detalles del proceso, pero poniendo más atención, observamos que Fred entrega un mensaje diferente a otro florista en la ciudad donde vive Robin. Ese florista pasa otro mensaje a su subordinado, quien hace el arreglo, las flores, junto con otro mensaje son pasadas al repartidor y así sucesivamente.

- Esto nos demuestra una primera imagen de la programación orientada a objetos:

Un programa orientado a objetos esta estructurado como una **comunidad** de agentes que interactuan llamados **objetos**. Cada objeto tiene un rol que jugar. Cada objeto provee un **servicio** o realiza una acción que es usada por otros miembros de la comunidad.

Objetos y Clases

- Es difícil hablar sobre objetos sin mencionar clases; es igualmente difícil hablar sobre clases sin traer a colación los objetos.
- Un **objeto** es cualquier *cosa*. Una **clase** consiste en una *categoría* de cosas.
- Un objeto, es un elemento específico que pertenece a una clase, a esto se le llama **instancia** de una clase.
- Una clase define las características de sus objetos y los métodos que pueden aplicarse a dichos objetos.

Objetos y Clases

- Es difícil hablar sobre objetos sin mencionar clases; es igualmente difícil hablar sobre clases sin traer a colación los objetos.
- Un **objeto** es cualquier *cosa*. Una **clase** consiste en una *categoría* de cosas.
- Un objeto, es un elemento específico que pertenece a una clase, a esto se le llama **instancia** de una clase.
- Una clase define las características de sus objetos y los métodos que pueden aplicarse a dichos objetos.

Nota

El tipo de dato de una variable (`float`, `int`, `char`) define que acciones se pueden realizar con la variable. Una clase es similar, es un *tipo de dato* para objetos más complejos que define lo que se puede hacer con ellos.

- Supongamos la clase `Plato`, sabemos que podemos sostener un `Plato` en la mano, podemos comer de un `Plato` y que podemos básicamente, interactuar con el.
- Un `Plato` tiene **atributos** como tamaño y color, y **métodos** como `llenar` y `lavar`.
- `miPlato` es un objeto y un miembro (o instancia) de la clase `Plato`, a esta situación se le denomina una relación **es un**, puesto que podemos decir "`miPlato` es un `Plato`".
- Por ejemplo, `tuPlato` es otra instancia de `Plato`, así como `platoParaEnsalada`.
- Puesto que `miPlato`, `tuPlato` y `platoParaEnsalada` son instancias de `Plato`, comparten características y métodos, todos tienen tamaño y color, y pueden ser llenados y lavados.

- Si alguien dijera "Le voy a comprar a mi abuela un `windsorEscarlata`", es muy probable que no pudieramos imaginar las características de dicho objeto.
- ¿Es algo que se come?¿Es algo que se viste?

[1] Clase `Boton`, Clase `CuartoHotel`

- Si alguien dijera "Le voy a comprar a mi abuela un `windsorEscarlata`", es muy probable que no pudieramos imaginar las características de dicho objeto.
- ¿Es algo que se come?¿Es algo que se viste?
- Si dijera por otro lado que un `windsorEscarlata` es un `Plato`, es claro que tendríamos una idea inicial, puesto que nuestro conocimiento de la clase `Plato` es general.
- Si el objeto es un `Plato`, sabemos que tiene un tamaño y color, y que puede ser llenado y lavado.
- Organizar los componentes de un programa en clases y objetos refleja una manera natural de pensar.[1]

[1] Clase `Boton`, Clase `CuartoHotel`

Herencia

- El concepto de utilizar clases provee de una manera útil de organizar objetos, esto es especialmenteútil puesto que las clases son reusables.
- Esto es, que podemos **extenderlas** (heredarlas). Podemos crear clases nuevas que extiendan o que sean **descendientes** de clases existentes.
- Las clases descendientes, pueden **heredar** todos los atributos de la clase original (o **padre**), o pueden sobreescribir atributos inapropiados.
- Utilizar herencia ahorra una gran cantidad de trabajo. Cuando creamos una clase que hereda de otra, solo necesitamos crear las nuevas características.

- Por ejemplo, cuando una compañia automotriz diseña un nuevo modelo de auto, no construye cada componente desde cero. El auto tal vez incluya alguna nueva característica, pero la muchas de las "nuevas" características serán simples modificaciones de lo ya existente.
- De manera similar, podemos crear poderosos programas computacionales más fácil, si muchos de sus componentes son usados "como tal" o con mínimas modificaciones.
- La herencia no permite escribir programas que no podiamos escribir si la herencia no existiera, es decir, *podríamos* crear cada parte del programa desde cero, pero reusar clases existentes hace el trabajo más fácil.

- En geometría, un `Cubo` es un descendiente de un `Cuadrado`.
- Un `Cubo` tiene todos los atributos de un `Cuadrado`, más una característica extra: profundidad.
- Un `Cubo`, sin embargo, tiene un método diferente de calcular el `areaTotal` (y `volumen`) al que tiene un `Cuadrado`.
- La clase `PlatoDesechable` tiene todas las características de `Plato`, más algunas extras.
- En los negocios, un `EmpleadoMedioTiempo` contiene todos los atributos de un `Empleado`, más algunos atributos especiales.
- Puesto que los lenguajes de programación orientados a objetos permiten herencia, podemos construir clases que sean extensiones de clases existentes; y con esto evitamos comenzar desde cero cada vez que queramos crear una clase.

Polimorfismo

- Los módulos de programación podrían ocasionalmente verse en la necesidad de cambiar la forma en que operan dependiendo de su contexto.
- Los programas orientados a objetos utilizan el **polimorfismo** para llevar a cabo la misma operación de una manera personalizada al objeto.
- Esto no se permite en lenguajes de programación que no son orientados a objetos.
- Sin el polimorfismo, tendríamos que usar módulos separados o diferentes nombres para un método que multiplica dos números y otro que multiplica tres números.
- Sin el polimorfismo, habría que generar nombres de módulos separados para un método para limpiar un `Plato`, otro para limpiar un `Carro` y otro para un `Bebé`

```
1  class Calculadora {
2      int multiplica2Enteros(int a, int b){
3      . . .
4      }
5      int multiplica3Enteros(int a, int b,int c){
6      . . .
7      }
8      float multiplica2Decimales(float a, float b){
9      . . .
10     }
11     . . .
12     float multiplica(float arr[]){
13     . . .
14     }
15 }
```

- Tal como una licuadora produce jugo, sin importar si ponemos 2 frutas o 3 verduras, al usar una función de multiplicación, polimórfica y orientada a objetos, produciría el resultado correcto sin importar si proporcionamos 2 enteros o 3 números decimales.
- Por lo tanto, usar un método polimórfico orientado a objetos, para limpiar, funcionaría adecuada y correctamente en un `Plato`, `Carro` o `Bebé`.
- Cuando logramos entender el concepto de polimorfismo en el mundo real, entendemos que usamos diferentes metodos para manejar un carro, manejar una situación o manejar una empresa. Esto debido a como funciona nuestro lenguaje.
- Cuando sabemos usar el polimorfismo en un lenguaje orientado a objetos, damos un gran paso hacia la construcción de objetos que funcionan como sus contrapartes del mundo real.

Modularidad

- Programar en los antiguos lenguajes procedurales, tenía dos grandes desventajas:
  1. El proceso de programación requería de tanto detalle, que el programador (y cualquiera que leyera el código), llegaba a olvidar el problema general.
  2. Enunciados similares requeridos en diversas partes del programa, tenían que ser reescritos en más de un lugar.
- Escribir programas, se volvió más fácil, cuando los lenguajes de programación comenzaron a permitir la escritura de **métodos** (grupos de instrucciones que pueden ser ejectuados como una unidad).

- El uso de métodos, permite a los programadores, agrupar instrucciones juntas, las cuales son conocidas entre los lenguajes de programación como: funciones, procedimientos, métodos, subprogramas, subrutinas o simplemente rutinas.
- Los programas modulares, son más fáciles de leer, que aquellos no-modulares, puesto que el nombre descriptivo de un grupo, representa una serie de pasos detallados.

Abstracción

- **Abstracción**, es el proceso de poner atención a propiedades o características importantes, ignorando los detalles.
- Usamos la abstracción en la vida real, al hacer una lista de pendientes, que contiene cosas como "Hacer las compras", "Lavar carro", "Pagar el agua", entre otras.
- Cada una de esas tareas, requiere de múltiples pasos y decisiones, pero nosotros no escribimos cada detalle involucrado en estas tareas.
- Por supuesto, se debe poner atención a los detalles en un determinado punto, y en un programa modularizado, cada uno de los módulos deberá ser escrito paso a paso.

- Cuando trabajamos con objetos de la vida real, damos por sentado la abstracción.
- Cuando hablamos por teléfono, no tenemos que preocuparnos por como se transmiten las señales, como se forman las palabras en nuestra boca, como se realizan los cobros de teléfono, etc. Si tuvieramos que hacerlo, probablemente nunca llegaríamos a completar una llamada.
- Programar en lenguajes de alto nivel, nos permite hacer uso de la abstraccón.
- Si programamos una función para imprimir, no tenemos que darle las instrucciones a la impresora sobre como hacerlo, podemos cambiar la función para imprimir en pantalla, y esto no debería cambiar el flujo de nuestro programa.
- Además de las ventajas de la abstracción, los programas modulares, pueden ser escritos más rápido, asignado un programador a cada módulo, donde cada uno puede trabajar simultánea e independientemente.

Encapsulamiento

- Los módulos o procedimientos actuan relativamente como mini programas autónomos. Estos procedimientos no solo puede contener sus conjuntos de instrucciones, sino también sus propias variables.
- Las variables e instrucciones dentro de un módulo estan ocultas y contenidas, es decir, **encapsuladas**, lo que hace el módulo independiente de los demás, y por lo tanto reusable.
- En la vida real, podemos observar muchos ejemplos de encapsulamiento. Cuando se construye una casa, no se inventan las instalaciones de plomería y aire acondicionado. Más bien, se hace uso de sistemas previamente diseñados y probados.

- No necesitamos conocer los detalles más finos de como funciona el sistema; son unidades contenidas dentro de sí que se pueden incorporar en la casa conectandolas mediante una interfaz estándar, o métodos de entrada, como un conector eléctrico.
- Este tipo de encapsulación, reduce el tiempo y esfuerzo necesarios para construir una casa. Asumiendo que los sistemas de plomería y aire acondicionado ya se esten usando en otras casas, se mejora la confiabilidad de la casa, mediante la experiencia previa.
- Además, al no necesitar saber como funciona el sistema de aire acondicionado, si se cambia por un modelo diferente, no importa cuando difieran en su funcionamiento interno. El resultado, una casa fresca, es lo que importa.

- De manera similar, el software reusable, ahorra tiempo y dinero y mejora la confiabilidad. Si una función o método ha sido probado con anterioridad, podemos tener la confianza de que producira el resultado correcto.
- Si otro programador creara una nueva función para sustituir a la anterior, no importará como funcione, mientras que obtengamos el resultado correcto.
- En los lenguajes procedurales, estamos limitados en la programación. Debemos saber los nombres y que módulos usar, y no podemos reusar los nombres para otros módulos en el mismo programa.
- Si necesitamos algo similar, pero ligeramente diferente, debemos crear un nuevo módulo, con un nombre diferente y usar el nuevo nombre para llamar al módulo.
- En la programación orientada a objetos, se reducen drásticamente este tipo de limitaciones.