FUNDAMENTOS Y LOGICA DE PROGRAMACION

Licenciatura en Ingeniería en Sistemas computacionales

Tercer Cuatrimestre

Mtro. Andrés Alejandro Reyes Molina

Mayo - Agosto

PROGRAMACIÓN LOGICA

Objetivo de la materia:

Analizar y solucionar problemas informáticos y representar su solución mediante herramientas de software orientado a objetos.

UNIDAD I

FUNDAMENTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS

1.2.- Conceptos fundamentales de la Programación Orientada a Objetos.

1.3.- Lenguajes orientados a objetos

1.4.- Relaciones entre clases y objetos.

1.5.- Papel de clases y objetos en el análisis y el diseño.

1.2.- Conceptos fundamentales de la Programación Orientada a Objetos.

El concepto de Sistema de Programación Orientado a Objetos - Object Oriented Programming System (OOPS), agrupa un conjunto de técnicas que nos permiten desarrollar y mantener mucho más fácilmente programas de una gran complejidad.

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 sola entidad, los datos que lo identifican y su comportamiento.

Características del paradigma orientado a objetos (clases, objetos, herencia, polimorfismo, generalización):

Objeto: Un objeto se puede definir desde el punto de vista conceptual como una entidad individual de un sistema y que se caracteriza por un estado y un comportamiento.

Clase: Las clases son declaraciones o abstracciones de objetos. Una clase es un contenedor de uno o más datos (variables o propiedades miembro) junto a las operaciones de manipulación de dichos datos (funciones/métodos).

Sintaxis de una clase:

```
1    Class nombreClase
2    {
3        miembros
4    }
```

Generalización: La generalización es la propiedad que permite compartir información entre dos entidades evitando la redundancia.

Electrodoméstico es una superclase de todas las otras clases (máquinas lavadoras, frigoríficos, hornos de microondas, tostadoras, lavavajillas…).

Especialización: El proceso inverso de la generalización por el cual se definen nuevas clases a partir de otras ya existentes se denomina especialización.

Herencia: La herencia permite definir nuevas clases a partir de otras clases ya existentes, de modo que presentan las mismas características y comportamiento de éstas, así como otras adicionales. Se refiere a compartir atributos y métodos entre clases, que se relacionan de manera jerárquica.

Polimorfismo:

La propiedad de polimorfismo es aquella en que una operación tiene el mismo nombre en diferentes clases, pero se ejecuta de diferentes formas en cada clase.

Abstracción:

El término abstracción que se suele utilizar en programación se refiere al hecho de diferenciar entre las propiedades externas de una entidad y los detalles de la composición interna de dicha entidad.

Encapsulación:

El encapsulado o encapsulación de datos es el proceso de agrupar datos y operaciones relacionadas bajo la misma unidad de programación.

- **Clase**: 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**: (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.
- **Objeto**: Instancia de una clase. 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**: 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.

- **Atributos**: características que tiene la clase.

- **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.

1.3.- Lenguajes orientados a objetos

Un entorno de programación puede estar concebido y organizado de maneras muy diferentes. A continuación, se mencionan algunas de ellas.

En estas condiciones no puede hablarse propiamente de un entorno de programación.

- El editor es un editor de texto simple
- El compilador traduce cada fichero de código fuente a código objeto
- El montador (linker / builder / loader) combina varios ficheros objeto para generar un fichero ejecutable
- El depurador maneja información en términos de lenguaje de máquina

Un entorno de programación propiamente dicho combina herramientas como éstas, mejoradas y mejor integradas. A veces se nombra con las siglas IDE (Integrated Development Environment).

Los componentes cuya evolución ha sido más aparente son los que realizan la interacción con el usuario:

- El editor ya no es un simple editor de texto, sino que tiene una clara orientación al lenguaje de programación usado (reconoce y maneja determinados elementos sintácticos)
- El depurador no presenta información en términos del lenguaje de máquina, sino del lenguaje fuente.
- El editor está bien integrado con las demás herramientas (se posiciona directamente en los puntos del código fuente en los que hay errores de compilación, o que se están ejecutando con el depurador en un momento dado.

Lenguajes de bajo nivel

- Código máquina
- Ensamblador

Lenguajes de medio nivel

- BCPL
- C

Lenguajes de alto nivel

- ADA
- ALGOL
- BASIC
- Clipper
- Cobol

- C++
- FORTH
- Fortran
- Haskell
- Informix 4gl
- Java
- Lexico (con códigos en castellano o sinónimos en otros idiomas)
- Lisp
- Logo
- Modula
- PASCAL
- Prolog
- RPG
- Visual Basic

Lenguajes orientados a objetos

Se le llama así a cualquier lenguaje de programación que implemente los conceptos definidos por la programación orientada a objetos.

Ejemplos de lenguajes orientados a objeto

- C++
- Objective C
- Java
- Smalltalk
- Eiffel
- Lexico (en castellano)
- Ruby
- Python
- OCAML

- Object Pascal
- CLIPS
- Visual .net
- Java
- Actionscript
- COBOL
- Perl
- C#
- Visual Basic.NET
- PHP

1.4.- Relaciones entre clases y objetos.

Objetos

- Un objeto es una cosa tangible, algo a que se puede aprehender intelectualmente o algo hacia lo que se puede dirigir una acción o pensamiento.
- Un objeto representa un item individual e identificable, o una entidad real o abstracta, con un papel definido en el dominio del problema
- Un objeto tiene:
    1. Estado
    2. Comportamiento
    3. Identidad

    La estructura y el comportamiento de objetos similares se definen en sus clases comunes. El término objeto y ejemplo (instance) de una clase son intercambiables.

Estado de un objeto

El estado de un objeto abarca todas las propiedades del objeto, y los valores actuales de cada una de esas propiedades. Las propiedades de los objetos suelen ser estáticas, mientras los valores que toman estas propiedades cambian con el tiempo.

- El hecho de que los objetos tengan estado implica que ocupan un espacio, ya en el mundo físico, ya en la memoria del ordenador.
- El estado de un objeto está influido por la historia del objeto.
- No deben confundirse los objetos, que existen en el tiempo, son mutables, tienen estado, pueden ser creados, destruidos y compartidos..., con los valores (los asignados a una variable, por ejemplo) que son cantidades con las propiedades de ser atemporales, inmutables.
- El estado de un objeto representa el efecto acumulado de su comportamiento.

Identidad de un objeto

Identidad es la propiedad de un objeto que lo lleva a distinguirse de otros.

Comportamiento de un objeto

Comportamiento es como un objeto actúa y reacciona, en términos de sus cambios de estado y de los mensajes que intercambia.

El comportamiento de un objeto representa su actividad externamente visible y testable. Son las operaciones que una clase realiza (llamadas también mensajes) las que dan cuenta de cómo se comporta la clase. Por operación se denota el servicio que una clase ofrece a sus clientes. Un objeto puede realizar cinco tipos de operaciones sobre otro, con el propósito de provocar una reacción:

1. Modificador: altera el estado de un objeto.
2. Selector: accede al estado de un objeto, sin alterarlo.
3. Iterador: permite a todas las partes de un objeto ser accedidas en un orden.
4. Constructor: crea un objeto y/o inicializa su estado.

5. Destructor: libera el estado de un objeto y/o destruye el objeto.

C++ soporta, además de las operaciones, subprogramas libres. En la terminología de C++ las operaciones que un cliente puede realizar sobre un objeto se declaran como funciones miembros.

Relaciones entre objetos

Las relaciones entre objetos abarcan las operaciones, resultados y suposiciones que unos hacen sobre los otros.

1. Links Son conexiones físicas o conceptuales entre objetos. Denota la asociación específica por la que un objeto (cliente) usa o solicita el servicio de otro objeto (servidor). El paso de mensajes entre objetos los sincroniza.
2. Agregaciones Denota relaciones todo/parte, con capacidad para governar desde el todo el parte. Es equivalente a la relación "tener un". El todo puede contener a la parte.

Agregación es conveniente en las ocasiones en que el encapsulamiento de las partes es prioritario. Si se requiere que las relaciones entre objetos estén vagamente acopladas, se utilizan links.

Clases

Una clase es un conjunto de objetos que comparten una estructura y comportamiento comunes.

- Clase representa una abstracción, la esencia que comparten los objetos.
- Un objeto es un ejemplo de una clase.
- Un objeto no es una clase, y una clase no es un objeto (aunque puede serlo, p.e. en Smalltalk).
- Las clases actúan como intermediarias entre una abstracción y los clientes que pretenden utilizar la abstracción. De esta forma, la clase muestra:

1. visión externa de comportamiento (interface), que enfatiza la abstracción escondiendo su estructura y secretos de comportamiento.
2. visión interna (implementación), que abarca el código que se ofrece en la interface de la clase.

Relaciones entre clases

Representan tipos de compartición entre clases, o relaciones semánticas.

1. **Asociación**. Indica relaciones de mandato bidireccionales (Punteros ocultos en **C++**). Conlleva dependencia semántica y no establece una dirección de dependencia. Tienen cardinalidad.
2. **Herencia**. Por esta relación una clase (subclase) comparte la estructura y/o comportamiento definidos en una (herencia simple) o más (herencia múltiple) clases, llamadas superclases.
    - Representa una relación del tipo "es un" entre clases.
    - Una subclase aumenta o restringe el comportamiento o estructura de la superclase (o ambas cosas).
    - Una clase de la que no existen ejemplos se denomina {\it abstracta}.
    - **C++** declara como *virtuales* todas aquellas funciones que quiere modificar en sus subclases.
3. **Agregación**. Representa una relación del tipo "tener un" entre clases. Cuando la clase contenida no existe independientemente de la clase que la contiene se denomina agregación **por valor** y además implica contenido físico, mientras que si existe independientemente y se accede a ella indirectamente, es agregación **por referencia**.
4. **Uso**. Es un refinamiento de la asociación donde se especifica cual es el cliente y cual el servidor de ciertos servicios, permitiendo a los clientes acceder sólo a las interfaces públicas de los servidores, ofreciendo mayor encapsulación de la información.
5. **Ejemplificación** Se usa en lenguajes que soportan genericidad (declaración de clases parametrizadas y argumentos tipo *template*). Representa las relaciones entre las clases parametrizadas, que admiten parámetros formales, y las clases obtenidas cuando se concretan estos parámetros formales, ejemplificados o inicializados con un ejemplo.

6. **Metaclases** Son clases cuyos ejemplos son a su vez clases. No se admiten en **C++**.

Relaciones entre clases y objetos

- Todo objeto es el ejemplo de una clase, y toda clase tiene 0 ó más objetos.
- Mientras las clases son estáticas, con semántica, relaciones y existencia fijas previamente a la ejecución de un programa, los objetos se crean y destruyen rápidamente durante la actividad de una aplicación.

El diseño de clases y objetos es un proceso incremental e iterativo. Debe asegurar la optimización en los parámetros:

1. Acoplamiento: Grado de acoplamiento entre módulos.
2. Cohesión: Mide el grado de conectividad entre elementos de un módulo, y entre objetos de una clase.
3. Suficiencia: Indica que las clases capturan suficientes características de la abstracción para conseguir un comportamiento e interacción eficiente y con sentido.
4. Completitud: Indica que la interface de la clase captura todo el significado característico de una abstracción, escrito en el mínimo espacio.
5. Primitividad: Las operaciones deben implementarse si dan acceso a una representación fundamental de la abstracción. Cuales son operaciones primitivas y cuales no (se pueden realizar a partir de otras) es un asunto subjetivo y afecta a la eficiencia en la implementación.