T−1 PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

1.1 ¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?

La POO (Programación Orientada a Objetos) se puede definir como un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representan una instancia de alguna clase, y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de herencia.

Existen tres importantes partes en la definición: la programación orientada a objetos 1) utiliza objetos, no algoritmos, como bloques de construcción lógicos; 2) cada objeto es una instancia de una clase; 3) las clases se relacionan unas con otras por medio de relaciones de herencia.

Es importante remarcar que si alguna de estas características no se cumple no se trata de programación orientada a objetos. Por ejemplo, la programación si herencia es distinta de la POO y se le denomina *programación con tipos abstractos de datos* o *programación basada en objetos*. Por ejemplo, en las primeras versiones de PowerBuilder (I, II y III) no se trataba de un lenguaje totalmente orientado a objetos ya que no se podían heredar clases, así que fue denominado un LPBO (Lenguaje de Programación Basado en Objetos). No fue hasta la versión IV que paso a ser un LPOO (Lenguaje de programación Orientado a Objetos).

Los conceptos fundamentales de la POO son: *objetos, clases, herencia, mensajes y polimorfismo.*

- **OBJETOS.**

La idea fundamental en los lenguajes orientados a objetos es combinar en una sola unidad *datos y funciones que operan dentro de esos datos*. Por consiguiente, dentro de los objetos residen los datos de los lenguajes de programación tradicionales, tales como números, arrays, cadenas y registros, así como funciones o subrutinas que operan sobre ellos.

Las funciones dentro del objeto (*métodos*) son el único medio de acceder a los datos privados de un objeto. Si se desea leer un elemento datos de un objeto se llama a la función miembro del objeto, se lee el elemento y se devuelve el valor. No se puede acceder a los datos directamente. Los datos están ocultos, y eso asegura que no se produzca lo que en los lenguajes procedurales se llamaban *efectos colaterales*, es decir, que no se puedan modificar accidentalmente por funciones externas al objeto.

Funciones externas Procedimientos externos Datos

Público Privado

Los datos y las funciones asociados se dicen que están *encapsulados* en una única entidad o modulo. La *encapsulación* de datos y su ocultación son términos importantes en la descripción de lenguajes orientados a objetos.

1.2.1 Estructura interna de los objetos

La estructura interna de un objeto consta de dos componentes básicos:

- Atributos
- Métodos (operaciones o servicios)

1.2.1.1 Atributos

Los **atributos** describen el estado del objeto. Un atributo consta de dos partes, un nombre de atributo y un valor de atributo.

Los objetos simples pueden constar de tipos primitivos, tales como enteros, caracteres, boolen, etc. Los objetos complejos pueden constar de pilas, conjuntos, listas, arrays, etc, o incluso de estructuras recursivas de alguno o todos de sus elementos.

1.2.1.2 Métodos

Los **métodos** (operaciones o servicios) describen el comportamiento asociado a un objeto. La ejecución de un método puede conducir a cambiar el estado del objeto o dato local del objeto.

Cada método tiene un nombre y un cuerpo que realiza la acción o comportamiento asociado con el nombre del método. En un LPOO, el cuerpo de un método consta de un bloque de código procedimental que ejecuta la acción requerida. Todos los métodos que alteran o acceden a los datos de un objeto se definen dentro del objeto. No se pueden modificar los datos (atributos) de otros objetos directamente, sino que se ha de llamar a los métodos de dichos objetos para que los modifiquen.

Un método dentro de un objeto se activa por un mensaje que se envía por otro objeto al objeto que contiene el método. Del mismo modo, se puede llamar a un método de un objeto a través de otro método de ese mismo objeto.

Objeto A

Atributos

...

1.3 ORIENTACIÓN A OBJETOS

Las técnicas orientada a objetos proporcionan un nuevo enfoque para construir sistemas de software complejos a partir de unidades de software modularizado y reutilizable. 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. Para ello, la programación orientada a objetos se basa en cuatro elementos (propiedades) :

- Abstracción
- Encapsulamiento
- Modularidad
- Jerarquía

Si alguno de estos elementos no existe se dice que el modelo no es orientado a objetos.

1.3.1 Abstracción

La **abstracción** es uno de los medios más importantes mediante el cual nos enfrentamos con la complejidad inherente al software. La abstracción es la propiedad que permite representar las características esenciales de un objeto sin preocuparse de las restantes características (no esenciales). La abstracción se centra en la vista externa de un objeto, de modo que sirva para separar el comportamiento esencial de un objeto de su implementación.

1.3.2 Encapsulamiento

El **encapsulamiento** o **encapsulación** es la propiedad que permite asegurar que el contenido de la información de un objeto esta oculta del mundo exterior: el objeto A no conoce lo que hace el objeto B, y viceversa. De esta manera combinamos los datos y los métodos que manejan dichos datos en un único objeto.

1.3.3 Modularidad

La **modularidad** es la propiedad que permite dividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en si y de las restantes partes.

1.3.4 Jerarquía

La **jerarquía** es una propiedad que permite una ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son:

- Estructura de clases (jerarquía **es−un**(*is−a*):generalización/especialización)
- Estructura de objetos (jerarquía **parte de**(*part−of*):agregación)

1.3.4 Polimorfismo

Polimorfismo es la propiedad que indica, literalmente, la posibilidad de que una entidad tome muchas formas. En términos prácticos, el polimorfismo permite referirse a objetos de clases diferentes mediante el mismo elemento de programa y realizar la misma operación de diferentes formas, según sea el objeto que se referencia en ese momento.

T−2 COMUNICACIÓN ENTRE OBJETOS

Los objetos realizan acciones cuando ellos reciben mensajes. El mensaje es esencialmente una orden que se envía a un objeto para indicarle que realice alguna acción. Esta técnica de enviar mensajes a objetos se denomina *paso de mensajes*. Mensajes y métodos son dos caras de la misma moneda. Los métodos son los procedimientos que se invocan cuando un objeto recibe un mensaje. En terminología de programación tradicional, un mensaje es una *llamada a una función*.

2.1 ACTIVACIÓN DE UN OBJETO

A los objetos solo se puede acceder a través de su interfaz pública. ¿Cómo se permite el acceso a un objeto? Un objeto accede a otro objeto enviándole un mensaje.

2.2 MENSAJES

Un mensaje es una petición de un objeto a otro objeto al que le solicita ejecutar uno de sus métodos. Por convenio, el objeto que envía la petición se denomina *emisor* y el objeto que recibe la petición se denomina *receptor*.

Estructuralmente un mensaje consta de tres partes:

- *Identidad* del receptor.
- El *método* que ha de ejecutar.
- *Información especial* necesaria para realizar el método invocado (argumentos o parámetros requeridos)

Objeto Fecha Fecha sumar 3 meses

receptor método parámetros

Cuando un objeto está inactivo y recibe un mensaje se hace activo. El mensaje enviado por otros objetos tiene asociado un método que se activará cuando el receptor recibe dicho mensaje. La petición no especifica cómo se realiza la operación. Tal información se oculta siempre al emisor.

T−3 CLASES

La clase es la construcción del lenguaje utilizada más frecuentemente para definir los tipos abstractos de datos en lenguajes de programación orientados a objetos. Generalmente, una clase se puede definir como una descripción abstracta de un grupo de objetos, cada uno de los cuales se diferencia por un *estado* específico y es capaz de realizar una serie de operaciones.

En programación, una clase es una estructura que contiene datos y procedimientos (o funciones) que son capaces de operar sobre esos datos. Dentro de un programa, las clases tienen dos propósitos principales: definir abstracciones y favorecer la modularidad.

A partir de una clase se puede definir un número de objetos. Cada uno de estos objetos tendrá, generalmente, una serie de características propias, aunque compartirán operaciones comunes. Los objetos ocupan espacio en memoria, y en consecuencia deberán *crearse o instanciarse*, así como destruirse para liberar el espacio ocupado. Dos operaciones comunes típicas en cualquier clase son:

- *Constructor*: una operación que crea un objeto y/o inicia su estado.
- *Destructor*: una operación que libera el estado de un objeto y/o destruye el propio objeto.

Cuando se desea crear una nueva instancia de una clase, se llama a un método de la propia clase para realizar el proceso de construcción. Los métodos constructores se definen como métodos de la clase. De modo similar, los métodos empleados para destruir los objetos y liberar la memoria ocupada también se definen dentro de la clase.

3.1 CLASES ABSTRACTAS

Con frecuencia, cuando se diseña un modelo orientado a objetos es útil introducir clases a cierto nivel que pueden no existir en la realidad pero que son construcciones conceptuales útiles. Estas clases se conocen como *clases abstractas*.

Una clase abstracta normalmente ocupa una posición adecuada en la jerarquía de clases que le permite actuar como un depósito de métodos y atributos compartidos para las subclases de nivel inmediatamente inferior.

Las clases abstractas no tienen instancias directamente. Se utilizan para agrupar otras clases y capturar información que es común al grupo. Sin embargo, las subclases de clases abstractas que corresponden a objetos del mundo real si pueden tener instancias.

Una clase abstracta podría ser una impresora:

+ inyectores + agujas

Las clases derivadas de una clase base o abstracta se conocen como *clases concretas*, que ya pueden *instanciarse* (es decir, pueden tener *instancias*).

T−4 MODELACIÓN DE RELACIONES ENTRE CLASES

4.1 RELACIONES ENTRE CLASES

Las relaciones entre clases juegan un papel muy importante en el modelo de objetos. Las clases, al igual que los objetos, no existen de modo aislado. Por esta razón existirán relaciones entre clases y entre objetos.

Las relaciones entre clases, se deben a dos razones: 1) una relación de clases puede indicar algún tipo de compartición 2) una relación entre clases puede indicar algún tipo de conexión semántica.

Los tres grandes tipos de relaciones entre clases son:

- *Generalización / especialización* **(es−un)**
- *Agregación* **(todo−parte//tiene−un)**
- *Asociación.*

4.2 RELACIÓN DE GENERALIZACIÓN / ESPECIALIZACIÓN

Uno de los motivos por los cuales las clases se relacionan entre ellas es el hecho de poseer propiedades comunes. Las clases con propiedades comunes se organizan en superclases. Una superclase representa una generalización de las subclases. De igual modo, una sublase de una clase dada representa una *especialización* de la clase superior. La clase derivada *es−un* tipo de clase de la clase base o superclase.

Una superclase representa una *generalización* de las subclases. Una subclase de la clase representa una *especialización* de la clase ascendente.

Es−un Es−un

ESPECIALIZACIÓN GENERALIZACION

Es−un Es−un Es−un Es−un

Es−un Es−un Es−un

En el modelado orientado a objetos es útil introducir clases a un cierto nivel que puede no existir en la realidad, pero que son construcciones conceptuales útiles. Estas clases abstractas tienen como propiedad fundamental que no se pueden crear instancias de ellas. Por ejemplo nunca crearemos instancias de *vehículo sin motor*, pero sí de bicicleta y patinete.

4.3 RELACIÓN DE AGREGACIÓN

Una *agregación* es una relación que representa a los objetos compuestos. Un objeto es *compuesto* si se compone a su vez de otros objetos. La agregación de objetos permite describir modelos del mundo real que se componen de otros modelos, que a su vez se componen de otros modelos.

Este es un concepto que se utiliza para expresar tipos de relaciones entre objetos **parte−de** (*part−of)* o **tiene−un**(*has−a*). El objeto componente, también a veces denominado *continente* o *contenedor*, es un objeto agregado que se compone de múltiples objetos.

Tiene−un Tiene−un

Tiene−un Tiene−un

Tiene−un

4.4 ASOCIACIÓN

Una **asociación** es una conexión entre clases, una conexión (enlace) semántica entre objetos de las clases implicadas en la asociación. El establecimiento de una asociación define los roles (papeles) o dependencias entre objetos de dos clases y su cardinalidades (multiplicidad); es decir, cuantas instancias (ejemplares) de cada clase pueden estar implicadas en una asociación.

Una asociación es, normalmente, bidireccional, lo que significa que si un objeto se asocia con otros objetos, ambos objetos se conocen entre si. Una asociación representa que objetos de dos clases tienen un enlace entre ellos, lo que significa por ejemplo, que ellos *conocen sobre los otros, están conectados a, para cada x hay una y,* etc. La asociación se representa por una línea que une a las dos clases y el nombre de la asociación se escribe en la línea.

Trabaja para

Emplea a

T−5 HERENCIA

Una de las herramientas disponibles en lenguajes orientados a objetos más potente es la herencia. Esta propiedad permite a los objetos ser construidos a partir de otros objetos. El objetivo final es la **reutilizabilidad** o **reutilización**, es decir, reutilizar código anteriormente ya desarrollado.

La herencia supone una *clase base* y una *jerarquía de clases* que contiene las *clases derivadas* de la clase base. Las clases derivadas pueden heredar el código y los datos de su clase base, añadiendo su propio código especial y datos a ellas, incluso cambiar aquellos elementos de la clase base que necesita sean diferentes.

Clase Base

Clase derivada Clase derivada Clase derivada

5.1 HERENCIA SIMPLE (Herencia jerárquica)

En esta jerarquía cada clase tiene como máximo una sola superclase. La herencia simple permite que una clase herede las propiedades de su superclase en una cadena jerárquica.

5.2 HERENCIA MÚLTIPLE (Herencia en malla)

Una malla o retícula consta de clases, cada una de las cuales puede tener una o más superclases inmediatas. Una herencia múltiple es aquella en la que cada clase puede heredar métodos y variables de cualquier número de superclases.

La clase C tiene dos superclases, A y D. Por consiguiente, la clase C hereda las propiedades de las clases A y D. Evidentemente, esta acción puede producir un conflicto de nombres, donde la clase C hereda las mismas propiedades de A y D.

5.21 Herencia repetida

Otro de los problemas graves que produce la herencia múltiple es la *herencia repetida*. Este tipo de herencia se produce cuando una clase hereda de dos o más superclases que a su vez heredan de la misma superclase. La

mayoría de los lenguajes de programación no permiten la duplicación estática de la superclase, pero eso no se producirá siempre, y así se puede dar el caso de que el compilador duplique la clase que se hereda dos o más veces.

T−6 POLIMORFISMO

Otra propiedad importante de la programación orientada a objetos es el *polimorfismo*. Esta propiedad, en su concepción básica, se encuentra en casi todos los lenguajes de programación. El polimorfismo, en su expresión más simple, es el uso de un nombre o un símbolo para representar o significar mas de una acción.

Esta propiedad permite que un mismo método se comporte de forma distinta dependiendo de que objeto lo esta ejecutando. Por ejemplo, todos los mamíferos tienen el método *comer*, pero este método se efectuara de forma distinta si este mamífero es un ciervo comiendo hierba, un león comiendo carne, una ballena comiendo plancton o un niño comiéndose un caramelo.

La gran ventaja ofrecida por el polimorfismo es permitir que los nuevos tipos de datos sean manipulados de forma similar que los tipos de datos predefinidos, logrando así ampliar el lenguaje de programación de una forma más ortogonal.

Función

Función

Función miembro

Función miembro

Función miembro

Datos

Datos

Datos

Método_1

Método_2

Método_3

El *método_1* llama al *método_2*, enviándole un mensaje

El *método_3* se invoca por un mensaje de otro objeto

Sumar

Impresora

Impresora de Chorro de Tinta

Impresora matricial

Vehículo

Vehículo sin motor

Vehículo con motor

Bicicleta

Patinete

Ciclomotor

Coche

4 X 4

Deportivo

Familiar

Tejado

Habitación

Casa

Pared

Ventana

Puerta

Empleado

Empresa

Característica A

Característica B

Característica B

A

Característica X

Característica A

Característica B

Característica W

D

Característica Z

Característica Y

Característica B

Característica A

C

B

F

E

A

D

C

B

E

Persona

Trabajador

Estudiante

Estudiante trabajador

Característica A