Apuntes A2.-Guía de Iniciación al Lenguaje JAVA de la Universidad de Burgos

I.1. INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

A. Programación Orientada a Objetos

La orientación a objetos es un paradigma de programación que facilita la creación de software de calidad por sus factores que potencian el mantenimiento, la extensión y la reutilización del software generado bajo este paradigma.

La programación orientada a objetos trata de amoldarse al modo de pensar del hombre y no al de la máquina. Esto es posible gracias a la forma racional con la que se manejan las abstracciones que representan las entidades del dominio del problema, y a propiedades como la jerarquía o el encapsulamiento.

El elemento básico de este paradigma no es la función (elemento básico de la programación estructurada), sino un ente denominado objeto. Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones que pueden modificar dicho estado, y determinan las capacidades del objeto.

Java incorpora el uso de la orientación a objetos como uno de los pilares básicos de su lenguaje.

B. Los objetos

Podemos definir objeto como el "encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios". **[Piattini et al., 1996]**.

Un objeto además de un estado interno, presenta una interfaz para poder interactuar con el exterior. Es por esto por lo que se dice que en la programación orientada a objetos "se unen datos y procesos", y no como en su predecesora, la programación estructurada, en la que estaban separados en forma de variables y funciones.

Un objeto consta de:

- Tiempo de vida: La duración de un objeto en un programa siempre está limitada en el tiempo. La mayoría de los objetos sólo existen durante una parte de la ejecución del programa. Los objetos son creados mediante un mecanismo denominado *instanciación*, y cuando dejan de existir se dice que son *destruidos.*
- Estado: Todo objeto posee un estado, definido por sus *atributos*. Con él se definen las propiedades del objeto, y el estado en que se encuentra en un momento determinado de su existencia.

- Comportamiento: Todo objeto ha de presentar una interfaz, definida por sus *métodos*, para que el resto de objetos que componen los programas puedan interactuar con él.

El equivalente de un *objeto* en el paradigma estructurado sería una *variable*. Así mismo la *instanciación de objetos* equivaldría a la *declaración de variables*, y el *tiempo de vida de un objeto* al *ámbito de una variable*.

C. Las clases

Las clases son abstracciones que representan a un conjunto de objetos con un comportamiento e interfaz común.

Podemos definir una clase como "un conjunto de cosas (físicas o abstractas) que tienen el mismo comportamiento y características... Es la implementación de un tipo de objeto (considerando los objetos como instancias de las clases)". **[Piattini et al., 1996]**.

Una clase no es más que una plantilla para la creación de objetos. Cuando se crea un objeto (*instanciación*) se ha de especificar de qué clase es el objeto instanciado, para que el compilador comprenda las características del objeto.

Las clases presentan el estado de los objetos a los que representan mediante variables denominadas *atributos*. Cuando se instancia un objeto el compilador crea en la memoria dinámica un espacio para tantas variables como atributos tenga la clase a la que pertenece el objeto.

Los *métodos* son las funciones mediante las que las clases representan el comportamiento de los objetos. En dichos métodos se modifican los valores de los atributos del objeto, y representan las capacidades del objeto (en muchos textos se les denomina *servicios*).

Desde el punto de vista de la programación estructurada, una clase se asemejaría a un módulo, los atributos a las variables globales de dicho módulo, y los métodos a las funciones del módulo.

D. Modelo de objetos

Existen una serie de principios fundamentales para comprender cómo se modeliza la realidad al crear un programa bajo el paradigma de la orientación a objetos. Estos principios son: la abstracción, el encapsulamiento, la modularidad, la jerarquía, el paso de mensajes y el poliforfismo.

a.) Principio de Abstracción

Mediante la abstracción la mente humana modeliza la realidad en forma de objetos. Para ello busca parecidos entre la realidad y la posible implementación de *objetos del programa* que simulen el funcionamiento de los *objetos reales*.

Los seres humanos no pensamos en las cosas como un conjunto de cosas menores; por ejemplo, no vemos un cuerpo humano como un conjunto de células. Los humanos entendemos la realidad como objetos con comportamientos bien definidos. No necesitamos conocer los detalles de porqué ni cómo funcionan las cosas; simplemente solicitamos determinadas acciones en espera de una respuesta; cuando una persona desea desplazarse, su cuerpo le responde comenzando a caminar.

Pero la abstracción humana se gestiona de una manera jerárquica, dividiendo sucesivamente sistemas complejos en conjuntos de subsistemas, para así entender más fácilmente la realidad. Esta es la forma de pensar que la orientación a objeto intenta cubrir.

b.) Principio de Encapsulamiento

El encapsulamiento permite a los objetos elegir qué información es publicada y qué información es ocultada al resto de los objetos. Para ello los objetos suelen presentar sus métodos como interfaces públicas y sus atributos como datos privados e inaccesibles desde otros objetos.

Para permitir que otros objetos consulten o modifiquen los atributos de los objetos, las clases suelen presentar métodos de acceso. De esta manera el acceso a los datos de los objetos es controlado por el programador, evitando efectos laterales no deseados.

Con el encapsulado de los datos se consigue que las personas que utilicen un objeto sólo tengan que comprender su interfaz, olvidándose de cómo está implementada, y en definitiva, reduciendo la complejidad de utilización.

c.) Principio de Modularidad

Mediante la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.

Esta fragmentación disminuye el grado de dificultad del problema al que da respuesta el programa, pues se afronta el problema como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

d.) Principio de Jerarquía

La mayoría de nosotros ve de manera natural nuestro mundo como objetos que se relacionan entre sí de una manera jerárquica. Por ejemplo, un perro es un mamífero, y los mamíferos son animales, y los animales seres vivos...

Del mismo modo, las distintas clases de un programa se organizan mediante la *jerarquía*. La representación de dicha organización da lugar a los denominados *árboles de herencia*:

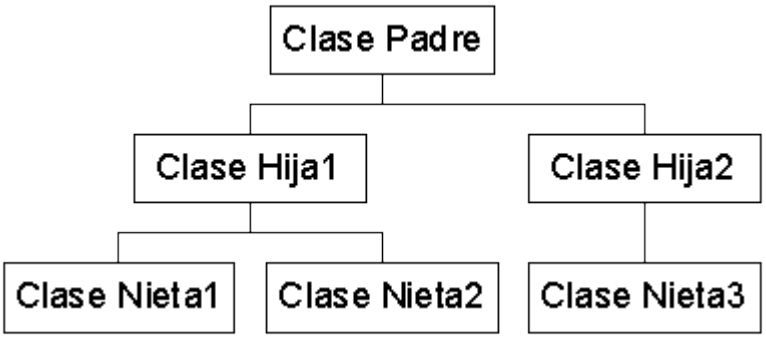

Imagen 1: Ejemplo de árbol de herencia

Mediante la *herencia* una *clase hija* puede tomar determinadas propiedades de una *clase padre*. Así se simplifican los diseños y se evita la duplicación de código al no tener que volver a codificar métodos ya implementados.

Al acto de tomar propiedades de una clase padre se denomina *heredar*.

e.) Principio del Paso de Mensajes

Mediante el denominado *paso de mensajes*, un objeto puede solicitar de otro objeto que realice una acción determinada o que modifique su estado. El paso de mensajes se suele implementar como llamadas a los métodos de otros objetos.

Desde el punto de vista de la programación estructurada, esto correspondería con la llamada a funciones.

f.) Principio de Polimorfismo

Polimorfismo quiere decir "un objeto y muchas formas". Esta propiedad permite que un objeto presente diferentes comportamientos en función del contexto en que se encuentre. Por ejemplo un método puede presentar diferentes implementaciones en función de los argumentos que recibe, recibir diferentes números de parámetros para realizar una misma operación, y realizar diferentes acciones dependiendo del nivel de abstracción en que sea llamado.

E. Relaciones entre objetos

Durante la ejecución de un programa, los diversos objetos que lo componen han de interactuar entre sí para lograr una serie de objetivos comunes.

Existen varios tipos de relaciones que pueden unir a los diferentes objetos, pero entre ellas destacan las relaciones de: asociación, todo/parte, y generalización/especialización.

a.) Relaciones de Asociación

Serían relaciones generales, en las que un objeto realiza llamadas a los servicios (métodos) de otro, interactuando de esta forma con él.

Representan las relaciones con menos riqueza semántica.

b.) Relaciones de Todo/Parte

Muchas veces una determinada entidad existe como conjunción de otras entidades, como un conglomerado de ellas. La orientación al objeto recoge este tipo de relaciones como dos conceptos; la agregación y la composición.

En este tipo de relaciones un *objeto componente* se integra en un *objeto compuesto*. La diferencia entre agregación y composición es que mientras que la composición se entiende que dura durante toda la vida del objeto componedor, en la agregación no tiene por qué ser así.

Esto se puede implementar como un objeto (*objeto compuesto*) que cuenta entre sus atributos con otro objeto distinto (*objeto componente*).

c.) Relaciones de Generalización/Especialización

A veces sucede que dos clases tiene muchas de sus partes en común, lo que normalmente se abstrae en la creación de una tercera clase (*padre* de las dos) que reúne todas sus características comunes.

El ejemplo más extendido de este tipo de relaciones es la herencia, propiedad por la que una clase (*clase hija*) recoge aquellos métodos y atributos que una segunda clase (*clase padre*) ha especificado como "heredables".

Este tipo de relaciones es característico de la programación orientada a objetos.

En realidad, la generalización y la especialización son diferentes perspectivas del mismo concepto, la generalización es una perspectiva ascendente (*bottom-up*), mientras que la especialización es una perspectiva descendente (*top-down*).

Para más información sobre el modelo de objetos en la programación avanzada, y las relaciones entre objetos véase **[García, 1998]** o para una información más detallada consulte **[Booch, 1996]**.