3.Metodología orientada al objeto

3.1 Introducción

El diseño orientado al objeto, al igual que otras metodologías de diseño orientadas a la información, crea una representación del campo del problema del mundo real y lo hace corresponder con el ámbito de la solución, que es el software.

El diseño orientado al objeto produce un diseño que interconecta objetos de datos (elementos dato) y operaciones de una forma que modulariza la información y el procesamiento; por el contrario, otros métodos dejan aparte el procesamiento.

La naturaleza única del diseño orientado al objeto queda reflejada en su capacidad de construir sobre tres pilares conceptuales importantes del diseño de software:

- Abstracción
- Ocultamiento de información
- Modularidad

Estos conceptos se explicarán en los siguientes puntos.

El análisis orientado al objeto (AOO), el diseño orientado al objeto (DOO) y la producción orientada al objeto comprenden un conjunto de actividades de Ingeniería del Software para la construcción del sistema orientado a objetos.

Utilizando el diseño orientado al objeto el diseñador puede crear sus propios tipos abstractos de datos y abstracciones funcionales y hacer corresponder el campo del mundo real con esas abstracciones creadas por el propio programador. Esta correspondencia será la mayoría de las veces mucho más natural, ya que el rango de tipos abstractos de datos que puede inventar el diseñador es virtualmente ilimitado. Más aún, el diseño del software se desliga de los detalles de representación, sin que ello afecte al sistema de software global.

Los objetivos clave del diseño orientado al objeto son:

A. *Aumentar la productividad*: Según algunos estudios, el diseño orientado al objeto logra aumentar la productividad de un desarrollo en un 20 %, lo cual no es mucho. Sin embargo, sabemos que entre el 75% y el 80% del coste de un sistema se produce después del desarrollo inicial. Pues bien, es precisamente en esta fase donde el diseño orientado al objeto puede ayudarnos a aumentar de forma espectacular la productividad.

B. *Incrementar calidad*: Cuando hacemos referencia al término calidad no solo nos estamos refiriendo a la ausencia de errores, sino también a otros aspectos quizás no tan fáciles de medir como son la facilidad de uso, la portabilidad o la facilidad de modificación.

C. *Facilidad de mantenimiento*: Debemos partir de la base que es imposible prever cambios que se producirán en meses o quizás años posteriores, pero lo que si podemos hacer es separar las partes del sistema que son intrínsecamente volátiles de aquellas que pueden ser estables. Los aspectos más volátiles podrían ser el interface externo, los atributos que describen ítems en el dominio del problema;... Mientras que los más estables deben ser las clases.

D. *Continuidad en la representación*: Uno de los problemas más importantes que presentan las metodologías estructuradas clásicas es el problema de comunicación existente entre las fases de Análisis y Diseño, e incluso dentro de la primera , entre los DFD (Diagramas de Flujo de Datos) y los DER (Diagramas Entidad-Relación).

   Las técnicas orientadas a objetos resuelven este problema, de manera que :

   - ♦ No hay diferencias entre la notación empleada en el análisis y la que se usa en el diseño.
   - ♦ No hay etapa de transición al diseño.
   - ♦ Es posible intercalar tareas del Análisis Orientado a Objetos y del Diseño Orientado a Objetos en el ciclo de desarrollo de la aplicación.
   - ♦ Sin embargo, análisis y diseño usan técnicas diferentes. Mientras que el AOO utiliza técnicas que ayudan a identificar y definir clases y objetos del dominio del problema, el diseño orientado al objeto emplea técnicas que ayudan a identificar y definir clases y objetos que reflejen la implementación de requerimientos.
   - ♦ La representación es uniforme desde el AOO hasta el DOO y la POO, constituyendo un marco de trabajo para la comprensibilidad, reusabilidad y extensibilidad.

3.1.1 El paradigma objetual

Orientado a Objetos (OO) se ha convertido en un sinónimo de algo bueno, moderno, etc., se aplica a múltiples cosas hoy en día, entre ellas a la que a nosotros nos interesa.

Los beneficios prometidos por la OO son básicamente dos:

- **Reusabilidad:** Los nuevos sistemas OO pueden ser creados utilizando otros Sistemas Orientados a Objetos ya existentes.

- **Extensibilidad:** Los nuevos Sistemas Orientados a Objetos así obtenidos son fácilmente ampliables sin tener que *retocar* los módulos, empleados en su construcción. El término OO debe entenderse como algo general, y que está íntimamente ligado a las etapas de Análisis y Diseño.

3.1.1.1 HISTORIA Y MITOS

En los Sistemas Orientados a Objetos tradicionalmente, primero, se hablaba sólo del tema de Programación y posteriormente fueron tomando importancia los de Análisis y Diseño.

La *Programación orientada a objetos* (POO) comienza en 1967 con Simula'67, y continua en los 70's con el desarrollo de SmallTalk.

La clave del comienzo de la POO la tuvo el hecho de querer simular modelos de la realidad, lo cual era, y es, bastante complicado utilizando lenguajes típicos de 3ª generación, básicamente porque no permite un fácil y buen modelado de los objetos del mundo real, así como de la manera de interactuar entre ellos.

En POO los mensajes reemplazan al concepto típico de función, y constituyen el diálogo entre objetos, lo que, a fin de cuentas, es la ejecución de un programa.

Los mensajes constituyen la comunicación entre los objetos y el término OO lo introdujo SmallTalk, el cual estaba influido por Simula y el trabajo de tesis doctoral de Alan Kay (máquina Flex). En Xerox esta máquina se llamó DynaBook y estaba influida por los conceptos de clase y herencia de Simula además de por determinadas características del lenguaje funcional Lisp.

Los primeros *Lenguajes orientados a objeto* (LOO) se caracterizan por tener el problema típico de la eficiencia, es por eso, que muchos de los LOO de hoy en día son evoluciones Orientadas a Objetos de lenguajes de tercera generación, p.e.: Object-Pascal, Objetive-C, C-Talk, C++, Turbo-Pascal, etc., o simplemente nuevos lenguajes, pero muy parecidos a los "normales" de 3ª generación.

Es interesante destacar, que conforme pasa el tiempo, y más se asienta el uso de la POO el foco de atención sobre temas relacionados con lo "Orientado a Objetos" se ha ido trasladando del concepto de Programación a los de Diseño (DOO) y a los de Análisis (AOO).

3.1.1.2 MITOS DE LA POO. ETIQUETAS QUE NO SON CIERTAS

Acompañando al concepto de POO siempre vamos a encontrar asociados lo que vamos a denominar una serie de *mitos* o etiquetas que no son ciertas.

Vamos a ver a continuación algunos de ellos:

- El primero de estos mitos es el de que "*todos los programas construidos utilizando técnicas de POO son automáticamente reutilizables y extensibles*". Esto es falso, si no programamos correctamente caeremos en los mismos errores que utilizando programación procedural normal. Sí que es cierto que un buen Análisis y Diseño previos facilitan enormemente la labor del programador, ya

que la POO ofrece una forma más natural de expresar o modelar los problemas del mundo real en un computador.

- Otro mito que tampoco es cierto completamente es el de que la POO rompe completamente con la programación procedural, no sólo no es cierto, sino que además de valerse de ella la extiende con los nuevos conceptos que añade.
- Otra idea a eliminar es la de que todo LOO necesita de una Librería de Clases a partir de la cual derivar cualquier clase nueva para trabajar con él, SmallTalk.
- También es falso lo que alguna vez se ha dicho como que un LOO no puede obtenerse como evolución de uno no OO, sólo puede ser LOO uno nuevo diseñado para ello. La mayoría de LOO de hoy en día son evoluciones de lenguajes de 3ª generación corrientes, tales como Pascal, C, etc.
- Otro aspecto a desmitificar en programas realizados con técnicas de POO es el de que estos no presentan una ejecución ordenada y clara. La ejecución de un programa realizado utilizando POO consiste en el diálogo que se establece entre los objetos que conforman el problema que se trata de resolver. Este diálogo se realiza mediante el paso de mensajes entre ellos.

3.1.2 Definiciones Previas

3.1.2.1 CLASE

Consiste en la descripción de uno o más objetos del mundo real en base a una serie de atributos y servicios.

A estos atributos se les llama también *variables de instancia*, mientras que a los servicios se les llama *métodos*.

Se puede decir que representa al *conjunto de objetos que comparten una estructura y comportamiento comunes*.

Una clase debe tener una *parte no visible* desde el exterior y una *parte visible* que es la encargada de acceder a esta parte no visible. En la parte no visible se suelen situar las variables de instancia mientras que en la visible se colocan los métodos de instancia.

A las clases cuyas instancias son a su vez clases se les llama metaclases.

No se debe confundir el concepto de *clase* con el de Tipo Abstracto de Datos. La diferencia está, según autores, en que los TAD no soportan herencia, mientras que el concepto de clase sí.

3.1.2.2 OBJETOS

Para las *personas*, un objeto suele ser algo de lo siguiente:

- Algo tangible y/o visible.
- Algo que puede ser comprendido mentalmente.

- Algo a lo que va dirigido el pensamiento o la acción.

Podemos utilizar como definición formal del mismo la siguiente:

> *Un objeto es una entidad real o abstracta con un papel bien definido en el dominio del problema.*

Además, un objeto se caracteriza porque presenta un estado, un comportamiento y una identidad propia.

El estado de un objeto representa todas las propiedades, normalmente estáticas, del objeto además de los valores actuales, normalmente dinámicos, de cada una de estas propiedades.

Podemos decir que el comportamiento de un objeto es el modo en que éste actúa y reacciona, hablando en términos de cambios en su estado y paso de mensajes.

3.1.2.3 MENSAJES Y MÉTODOS

El término utilizado para nombrar a la acción que un objeto realiza sobre otro es el de mensaje y también el de operación.

Podemos decir que lo que un objeto le pide que realice a otro es el mensaje, mientras que cómo hace el segundo objeto eso que le han pedido sería el método.

Según diversos autores puede haber hasta cinco tipos distintos de operaciones sobre un objeto:

- Modificadoras.
- Selectoras.
- Iteradoras.
- Constructoras.
- Destructoras.

En cuanto a la identidad, ésta se define como la propiedad que distingue a un objeto de otros, y normalmente viene designada por un nombre único de variable.

3.1.3 Características de los LOO

3.1.3.1 ENCAPSULACIÓN

Muchos autores intercambian los términos de encapsulación y abstracción.

Normalmente, se admite como encapsulación a la posibilidad de agrupar bajo una misma entidad los datos y las funciones, o métodos, que trabajan con esos datos. Aunque otros autores llaman a esto abstracción.

Los mecanismos clásicos de abstracción utilizados han sido, y por este orden:

- Procedimientos.
- Módulos.
- TAD's.
- Objetos junto con el paso de mensajes, herencia y polimorfismo.

Según este criterio, al agrupar las funciones que trabajan con una serie de datos en la misma entidad que estos, se consigue independizar la implementación interna de la misma, de manera que posibles cambios en su interior afectan mínimamente a sus usuarios.

Trabajando de este modo se consigue minimizar el tiempo empleado en desarrollar un nuevo sistema.

3.1.3.2 HERENCIA

Consiste en un mecanismo para expresar la similitud entre clases.

Se utiliza en la construcción de nuevas clases a partir de otras clases ya existentes, de manera que las nuevas clases así creadas incorporan la estructura y el comportamiento de la(s) clase(s) de la(s) que heredan.

Cuando una clase *hereda* de otra u otras, se dice que es subclase o clase derivada, de ella(s), que, a su vez, son su(s) superclase(s) o clase(s) base.

Lo que estamos diciendo es que *comparte* las variables de clase y de instancia, así como los métodos de clase y de instancia de su(s) superclase(s).

La herencia reduce el número de cosas que hemos de *decir* sobre una nueva clase que creamos, para ello debemos tener la precaución de hacer que herede de una clase parecida a ella.

No todos los LOO admiten la llamada herencia selectiva, sino que se limitan a heredar todo lo que les proporciona su(s) superclase(s).

La herencia tiene una serie de beneficios para el programador:

- Reusabilidad.
- Compartir Código.
- Consistencia de interfaz.
- Posibilidad de construir Software.
- Prototipado rápido.
- Polimorfismo.
- Ocultación de información.

Pero también tiene una serie de pegas:

- Velocidad de ejecución menor.
- Tamaño de los programas mayor.
- Sobrecarga debida al paso de mensajes.
- Aumento de la complejidad de los programas.

3.1.3.3 PASO DE MENSAJES

Como ya se ha visto, un objeto consta de una serie de datos privados que se encargan de mantener su estado actual, además de una serie de operaciones que pueden acceder a ellos y modificarlos o consultarlos. Estas operaciones se caracterizan porque son las únicas que pueden *llegar* hasta ellos.

Dependiendo de lo que pidamos al objeto, éste invocará una u otra operación, y una vez finalizada devolverá el control al objeto que le envió ese mensaje, si asumimos que no existe la concurrencia.

Suele existir la tendencia a confundir los conceptos de mensaje y función invocada en respuesta a un mensaje o método, sin embargo es posible diferenciarlos teniendo en cuenta que:

- Una función o subprograma puede tener 0 ó más argumentos, mientras que un mensaje tendrá siempre 1 ó más. Este primer argumento suele ser el objeto al cual se le envía el mensaje, y se pasa como parámetro de forma oculta para nosotros.
- Un nombre de función se identifica mediante una relación 1:1 con el código a ejecutar cuando se llame, mientras que un mensaje no, ya que el código a ejecutar depende del objeto al que se le envió el mensaje.

Una vez llegados a este punto, debemos destacar la aparición de dos conceptos nuevos, Consumidor y Suministrador de clases y/o objetos en un sistema software. El consumidor de clases sólo ve, y le preocupan, las operaciones que puede realizar con los objetos de determinadas clases que le proporciona un suministrador, y no cómo están realizadas esas operaciones.

3.1.3.4 ENLACE DINÁMICO

Vamos a considerarlo como un instante de tiempo en el que se determina o identifica el trozo de código.

Referido a programación vamos a considerar dos tipos de enlace distintos:

- Compilación: El tipo de una variable o identificador, el código a ejecutar en respuesta a la llamada de una función se conoce en tiempo de compilación.
- Ejecución: El tipo de una variable o identificador, o el código a ejecutar en respuesta a la llamada de una función no se conoce hasta que nos encontramos en tiempo de ejecución.

Cuando se habla de tipo de un identificador, se debe distinguir entre:

- Identificador: Es sólo un nombre.
- Valor: El contenido de la memoria asociada a un identificador.
- Tipo: Depende del lenguaje con el que trabajemos. En unos casos irá asociado a una variable y significará una cosa, mientras que en otros irá asociado a un valor y significará otra.

3.1.3.5 POLIMORFISMO

El término proviene del griego polimorphos (poly = muchas, morphos = formas).

En lenguaje de programación el término se aplica a:

- Objetos: Objetos polimórficos, vistos estos como variables o como argumentos de funciones, los cuales pueden contener valores de distinto tipo a lo largo de la ejecución de un programa.

3.1.4 Ejemplo para comprender el punto de vista OO

Silla es una instancia o miembro de una clase de objetos mucho mayor que denominamos mueble. Esta clase tiene asociados un conjunto de atributos genéricos, por ejemplo:

- Precio.
- Dimensiones
- Peso.
- Situación.
- Color.

entre muchos atributos posibles.

Estos atributos se aplican estemos hablando de una mesa, de un sofá, una silla, etc., y dado que una silla es una instancia ó miembro de la clase mueble, hereda todos los atributos definidos en la clase. La figura 1 muestra la correspondencia entre clase y objeto donde vemos cómo el objeto hereda todos los atributos definidos de la clase:

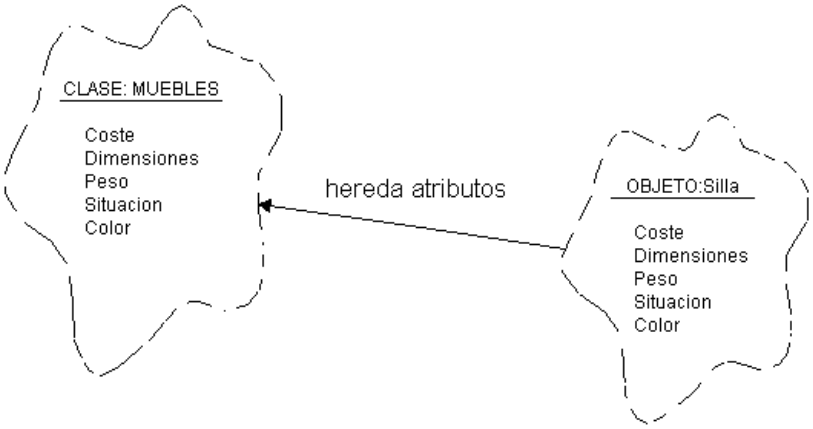

figura 1: Correspondencia entre clase y objeto

Una vez que se ha definido la clase, se pueden crear instancias de esa clase, como por ejemplo, el nuevo objeto Mesa (aparece en la figura 2), que es miembro de la clase Mueble y hereda todos los atributos del mueble.

Cada objeto de la clase mueble puede ser manipulado de muchas formas. Puede ser vendido y comprado, modificado (quitar una pata o pintarlo de rojo) o movido de un sitio a otro. Cada uno de esos métodos u operaciones modifican uno ó más atributos del objeto. Por ejemplo, si el atributo situación es realmente un elemento de datos compuesto que está definido como:

situación = Edificio + piso + habitación.

entonces, una operación denominada Mover modificará uno o más de esos elementos de datos que componen el atributo situación. Más operaciones válidas serán las siguientes, comprar, vender, pesar, etc.

Todas la operaciones para la clase Mueble son heredadas por todas las instancias de la clase.

La figura 2 muestra el esquema que resultaría:

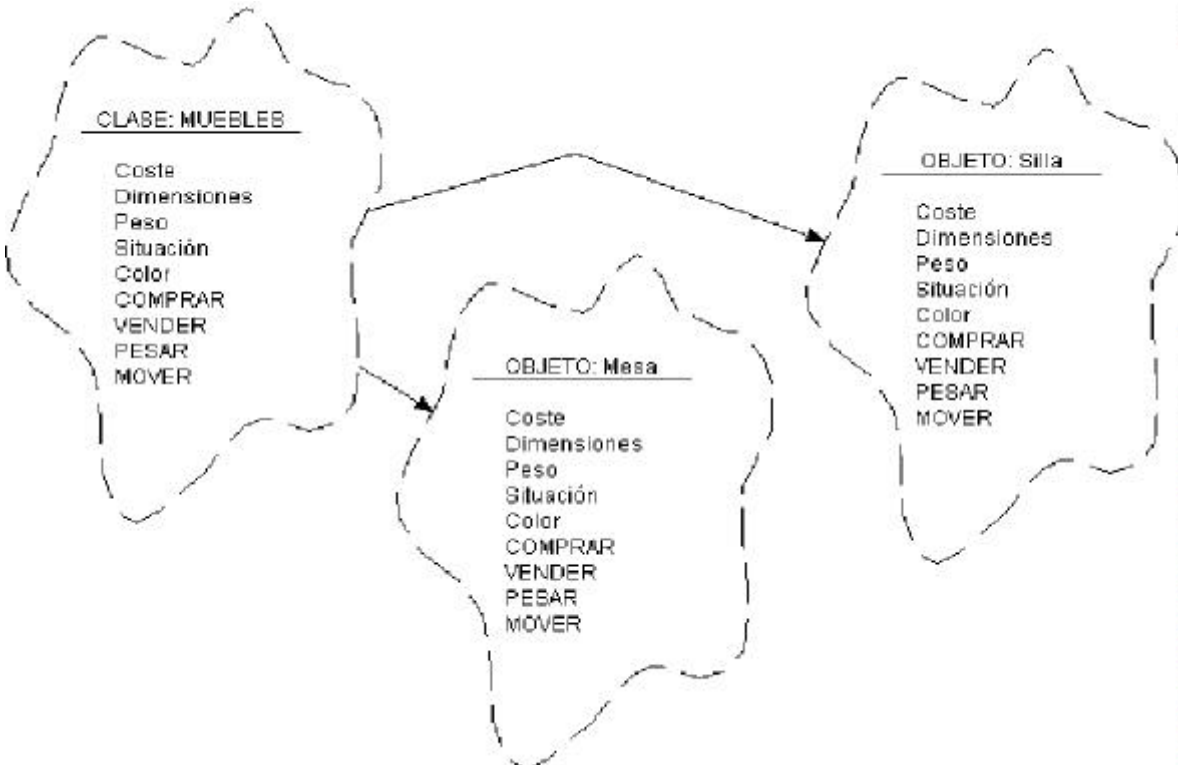

figura 2: Herencia de atributos y operaciones

Los objetos heredan todos los atributos y operaciones de la clase.

3.1.5 Análisis Orientado a objetos

3.1.5.1 IDENTIFICACIÓN DE OBJETOS

Si miramos alrededor de una habitación, veremos objetos físicos que pueden ser identificados fácilmente pero cuando observamos una aplicación de Software resulta más complicado detectar objetos.

Para identificar objetos debemos hacer lo siguiente:

- Examinar la descripción del problema llevando a cabo un *análisis gramatical* de la descripción del procesamiento del sistema a construir.
- Determinar los objetos subrayando cada nombre y añadiéndolo en una tabla.
- Una vez aislados todos los nombres los objetos pueden ser:

  - Entidades Externas.- (gente, dispositivos), que producen o consumen información a ser utilizada en el sistema.

- ◊ Cosas.- (Informes, cartas, señales) que son parte del dominio de información del problema.
- ◊ Ocurrencias o sucesos.- (Transferencia de una propiedad) que ocurren en el contexto de operación del sistema.
- ◊ Papeles.- que juegan las personas que interactúan con el sistema (vendedor, ingeniero).
- ◊ Unidades Organizativas.- (división, equipo, grupo) que son relevantes para la aplicación.
- ◊ Lugares.- (Muelle descarga) que establecen el contexto del problema y del funcionamiento general del sistema.
- ◊ Estructuras.- (sensores, ordenadores) que definen clases de objetos.

La estructura de la tabla sería de la siguiente manera:

Tabla

Objeto potencial	Clasificación
Sensor	Entidad Externa

3.1.5.2 ESPECIFICACIÓN DE ATRIBUTOS

Los atributos describen un objeto. Para desarrollar un conjunto significativo de atributos para un objeto, el analista debe estudiar de nuevo la narrativa de procesamiento y seleccionar aquello que pertenezca al objeto, y siempre se debe responder esta pregunta:

¿Qué elementos de datos (elementales o compuestos) definen completamente ese objeto en el contexto del problema actual?

Ejemplo: Sensor
Sus atributos serían:
Tipo sensor + nº sensor + unidad alarma

3.1.5.3 DEFINICIÓN DE LAS OPERACIONES (MÉTODOS)

Una operación cambia un objeto de alguna forma. Por tanto una operación debe tener "conocimiento" de la naturaleza de los atributos del objeto y debe ser implementada de forma que se le permita manipular las estructuras de las que hayan sido derivadas de los atributos.

Las operaciones se pueden dividir en 3 categorías:

- ♦ Manipular los datos (adiciones, borrados)
- ♦ Realizar algún Cálculo
- ♦ Monitorizar un objeto frente a la ocurrencia de algún suceso de control.

Para obtener un Conjunto de operaciones el analista lleva a cabo de nuevo un análisis gramatical y se aíslan los verbos.

Ejemplo: "cada sensor tiene asignado un tipo y un número".

Operación - ASIGNACION

3.1.5.4 COMUNICACIÓN ENTRE OBJETOS

Para construir el sistema, los objetos definidos deben de comunicarse mediante mensajes. El mensaje estará formado por:

- Mensaje: (destino, operación, argumentos)

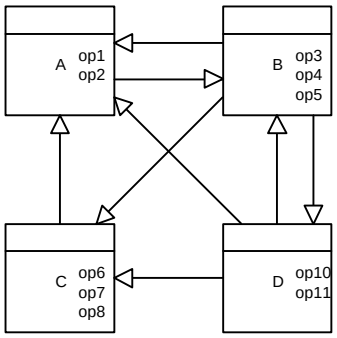

figura 3: Paso de mensajes

Mensaje: (C, op7, < datos>)

Resumiendo: A un objeto se le pide que realice una de sus operaciones enviándole un mensaje que le indique qué es lo que tiene que hacer.

El objeto receptor responde al mensaje escogiendo, primero la operación que implementa el nombre del mensaje, ejecutando esa operación y devolviendo el control al emisor.

3.1.5.5 EJEMPLO

Como ejemplo de aplicación del método recién descrito, consideremos una herramienta CAD para visualizar y manipular una serie de primitivas bidimensionales.

La herramienta permite a los usuarios crear y manipular polígonos, cintas y cónicas bidimensionales en un dispositivo de gráficos en color. Usando un dispositivo gráfico de entrada como puede ser un ratón, el usuario puede mover, rotar, escalar y colorear las primitivas.

Los pasos a dar son:

- *Identificar las abstracciones de datos para cada subsistema*

  Los requisitos indican que la herramienta va a tener tres primitivas diferentes, pero primero buscamos cosas comunes a todas las primitivas.

  Clase superior se llama objeto geométrico y realiza funciones que son adecuadas para todas las clases. La clase primitiva es una clase abstracta, actuando como contenedor de los datos y los métodos que usan las clases inferiores de la jerarquía.

- *Identificar los atributos de cada abstracción*

  Al menos un atributo común a todas las primitivas resulta obvio, el color. Otros atributos como la posición, la orientación, la textura, la escala, la transparencia, etc. Por los requisitos sabemos que el usuario va a mover, a escalar y a rotar las primitivas, por lo que escogemos la posición, los factores de escala y la orientación como variables de instancia adecuadas.

- *Identificar las operaciones de cada abstracción*

  Aquí, los requisitos de nuestra herramienta especifican las operaciones **crear, mover, escalar y rotar.**

- *Identificar la comunicación entre objetos*

  Ahora podemos especificar el protocolo entre objetos asociando los mensajes con los métodos anteriores:

```
nueva!              crear_primitiva
posición=           poner_posición
posición?           obtener_posición
posición+           sumar_posición
orientación=        poner_orientación
orientación?        obtener_orientación
orientación+        sumar_orientación
escala=             poner_escala
escala?             obtener_escala
escala+             sumar_escala
color=              poner_color
color?              obtener_color
```

  Como hemos visto en el apartado 3.1.3.3 existen una serie de operaciones que son las únicas que pueden acceder a los datos privados de los objetos. También hemos visto que la comunicación entre objetos se realiza mediante el pase de mensajes. En los mensajes anteriores observamos que se invocan a cuatro tipos diferentes de operaciones :

  ! que invoca a la función de crear primitiva.

  = que invoca a la función de asignar valor

  + que invoca a la función que modifica el valor

? que invoca a la función que devuelve el valor actual del atributo en cuestión.

- *Probar el diseño con guiones*

  Hacemos corresponder los requisitos con guiones sencillos que muestren cómo conseguir cada requisito:

```
Crear primitiva:primitiva nueva!nombre=unaPrimitiva
Mover           :    unaPrimitiva posición = (1, 2)
Rotar           :    unaPrimitiva orientación = 30
Escalar         :    unaPrimitiva escala = (10, 1)
Colorear        :    unaPrimitiva color = (1, 0, 0)
```

  Ahora repetimos los cinco pasos para añadir nuevas abstracciones.

- *Identificar las abstracciones de datos para cada subsistema*

  La figura 4 muestra este nivel de abstracción, Polígono, Cónica, Cinta

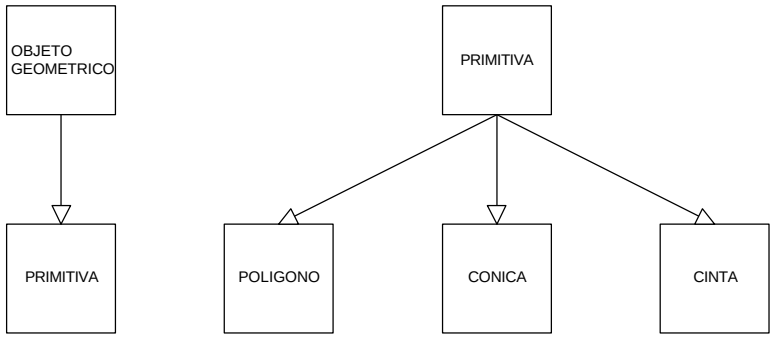

figura 4: Jerarquía inicial de clases y segundo nivel de abstracción

- *Identificar los atributos de cada abstracción*

  Aquí sólo seguiremos los siguientes pasos para las cónicas. Para las cónicas hay seis coeficientes que las especifican y que pasan a ser variables de instancia que son a, b, c, d, e, f.

  Nuevas operaciones para establecer y recuperar los seis coeficientes:

```
Crear_cónica         crear una cónica
poner_coeficientes   establece coeficientes
obtener_coeficientes obtiene coeficientes
```

- *Identificar la comunicación entre objetos*

```
nueva!          crear_cónica
coeficientes=   poner_coeficientes
coeficientes?   obtener_coeficientes
```

Una vuelta más en el diseño dará por terminado el proceso por ahora.

- Identificar las abstracciones de datos para cada subsistema

Seis son los coeficientes que especifican una cónica. Observamos, sin embargo, que los círculos y las elipses tienen parámetros que puede especificar fácilmente un usuario, cónica - {parábola, elipse, círculo, hipérbola}.

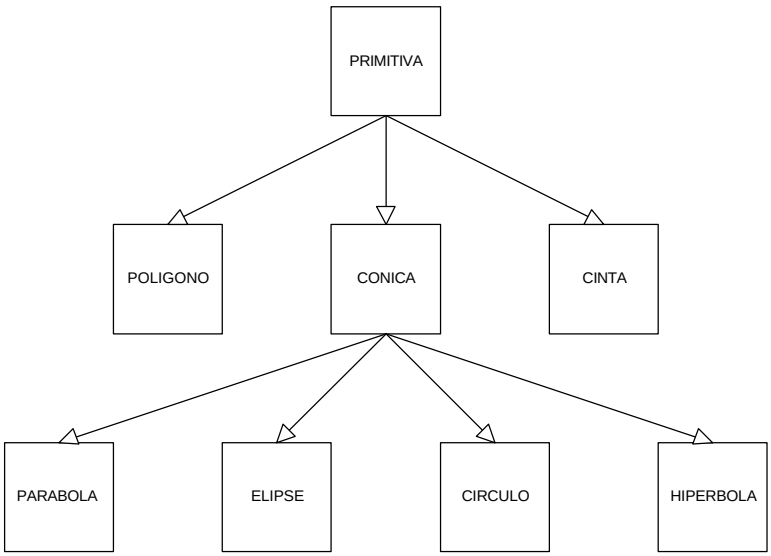

figura 5: Jerarquía de clases para primitiva

- Como círculo hereda los métodos de cónica, debemos inspeccionar los métodos de las cónicas para ver si hay alguna necesidad de redefinición. Por ejemplo, en la ecuación de las cónicas, los círculos tienen algunos coeficientes a cero. Si dejamos que **círculo** herede el método poner_coeficiente de **cónica**, podríamos crear lo que pensemos que sea un círculo, pero que en realidad sea una cónica genérica. Es necesario redefinir el método.
- *Probar el diseño con guiones*

```
Crear un círculo:Círculo nuevo! nombre=unCírculo
Modif. círculo  :unCírculo radio= 10
Mover           :unCírculo posición=  (1, 2)
Rotar           :unCírculo orientación= 30
Escalar         :unCírculo escala=  (10, 1)
Colorear        :unCírculo color=  (1, 0, 0)
```

Tras repetir los pasos para la elipse, nos damos cuenta de que un **círculo** es una especialización de una **elipse**. Aunque esto resultaba obvio en este ejemplo, en muchos casos de diseño sólo nos damos cuenta de estos aspectos una vez que hemos diseñado ya algunas clases.