Concepto de Objeto y Clase

Concepto de Objeto

Si miramos a nuestro alrededor casi todo puede ser considerado un objeto. El dinero, un helicóptero, una bicicleta, un automóvil. Los objetos representan cosas, simples o complejas, reales o imaginarias. Una antena parabólica es un objeto complejo y real. Un objeto Profesor, representa los detalles y actividades de una persona, no es esa persona en sí misma, es pues, imaginario. Una frase, un número complejo, una receta y una cuenta bancaria también son representaciones de cosas intangibles. Todas son objetos.

Algunas cosas no son objetos, sino atributos (características del objeto), como el color, el tamaño y la velocidad, que son los que lo definen. Los atributos, como la velocidad del objeto avión, o el tamaño de un objeto edificio, reflejan el estado de un objeto. El estado es el valor concreto que toma cada atributo, por ejemplo, el atributo color toma el valor "rojo".

En términos mas generales, un objeto es una abstracción conceptual del mundo real que se puede traducir a la representación de un modelo de análisis y diseño OO o a un lenguaje de programación OO.

Los objetos tienen una funcionalidad o comportamiento definido, y se espera que siempre se comportarán de acuerdo a las especificaciones que están definidas para ellos. Por ejemplo, el cambiador de CD nunca provocará que un automóvil acelere, y pisar el pedal del freno no colgará el teléfono. Esta es la forma de gestionar la complejidad en la vida real, cada cosa haciendo lo que debe, aquello para lo cual fue concebida. Trasladada al campo del software, funciona igual de bien. Es así que se traduce el concepto de objeto en el mundo real al de la informática como una entidad provista de propiedades (datos, atributos) y comportamiento (funcionalidad, métodos).

Propiedades de los Objetos

Desde un punto de vista más formal, puede decirse que un objeto queda definido por las siguientes propiedades:

- Estado (atributos)
- Comportamiento (operaciones)
- Identidad (propiedad que lo distingue de los demás objetos)

El **estado** de un objeto define la situación del mismo en un momento específico en el tiempo. Los valores que toma un objeto se almacenan en variables internas llamadas **atributos**.

El **comportamiento** describe las acciones y reacciones de un objeto. Es la manera en que responde a mensajes o estímulos recibidos. Lo hace básicamente enviando mensajes a otros objetos, respondiendo al que le envía el mensaje o cambiando de estado. Las distintas formas de respuesta se denominan **métodos**, y toman la forma de procedimientos o funciones. Se dice que el comportamiento indica lo que el objeto *sabe hacer*.

La **identidad** es lo que diferencia a un objeto de todos los otros del mismo tipo. Hablando en términos computacionales, la identidad del objeto se puede interpretar como la referencia, es decir el lugar específico en que se almacena el objeto. Dos objetos son distintos incluso aún en el caso de que los valores de todos sus atributos (p. ej. nombre y tamaño) coincidan. Dos manzanas pueden ser totalmente idénticas pero no por eso pierden su identidad: nos podemos comer una u otra.

Los atributos y los métodos son las propiedades que dan estructura al objeto.

- **Atributos**: son las características que describen al objeto. En POO los atributos corresponden a las clásicas "variables" de la programación estructurada. Los atributos de un objeto pueden ser datos simples o puede ser una estructura más compleja de datos (matrices, vectores, listas, etc.). Pueden ser de tipo primitivo (entero, carácter, etc.), o pueden a su vez ser objetos.
  Sin embargo, existe una diferencia con las "variables", y es que los atributos se pueden heredar de unos objetos a otros. En consecuencia, un objeto puede tener un atributo de maneras diferentes:
  - *Atributo propio:* creado dentro de la cápsula del objeto.
  - *Atributo heredado:* definido en un objeto diferente, antepasado de éste (padre, abuelo, etc.).

  También se los denomina atributos o variables miembro, porque el objeto las posee por el mero hecho de ser miembro de una clase.
- **Métodos:** operaciones de un objeto que realizan el acceso a sus propios datos, y los procesa para obtener los resultados esperados. Se puede definir un método como el código asociado a un objeto determinado, y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste.
  Los métodos especifican la forma en que actúa el objeto, y permiten manipular y controlar sus propios datos. No deben tener acceso directo a las estructuras de datos de otros objetos. Para utilizar la estructura de datos de otro objeto, deben enviar un mensaje a éste.
  Son sinónimos de 'método' todos aquellos términos que se han aplicado tradicionalmente a los programas, como procedimiento, función, rutina, etc. Sin embargo, es conveniente utilizar el término 'método' para que se distingan claramente las propiedades especiales que adquiere un programa en el entorno POO, que afectan fundamentalmente a la forma de invocarlo (únicamente a través de un mensaje) y a su campo de acción, limitado a un objeto y a sus descendientes.

Objeto y encapsulación

Un objeto es una unidad que combina datos (atributos) y las operaciones (métodos) que operan sobre esos datos. Los objetos *encapsulan* (agrupan y ocultan) sus operaciones y atributos.

Encapsulación es la propiedad del modelo de objetos que describe la vinculación entre un conjunto de operaciones y el estado a un objeto particular. Está íntimamente relacionada con la **ocultación de la información**, definiendo qué partes de un objeto son visibles y qué partes están ocultas.

Esta propiedad permite asegurar que los datos y los detalles de implementación de un objeto están ocultos al exterior. Si un objeto (obj-1) desea modificar los datos de otro objeto (obj-2), sólo puede hacerlo a través de los métodos definidos en obj-2 para acceder a esos datos.

Un objeto es un TAD (Tipo Abstracto de Dato) al que se añaden características propias de la POO como la herencia, el paso de mensajes, el polimorfismo y la ligadura dinámica.

Recordemos que un TAD es un tipo de dato caracterizado por las operaciones que se pueden llevar a cabo sobre los elementos del mismo. Es decir, los TAD engloban dos conceptos: la representación de los datos y las operaciones permitidas sobre ellos. Por ejemplo, los tipos de datos ***int*** tienen una forma particular de representación (por ej. Complemento a 2, sujeto a su rango de representación), y las operaciones que soporta son por ejemplo la suma, resta, división, etc. Mientras que el tipo de datos ***char***

tiene otro modo de representación (por ej. el ASCII de 7 bits → 128 caracteres), y una operación por ejemplo puede ser concatenar.

Los objetos tienen una interfaz pública, que es la que indica su funcionalidad (*qué están dispuestos a hacer*), y una representación privada, que es la implementación (*cómo lo hacen*). Esto permite ocultar la información que se desea que no esté accesible desde el exterior.

Haciendo una analogía con la industria automotriz, el que provee a una fábrica de automóviles los equipos de audio, no suele especificar cómo están construidos los mismos. Sólo deben asegurar una serie de conectores o interfaces que permiten su instalación y una serie de características y funcionalidades a prestar. De este modo, el fabricante del equipo de audio puede realizar cambios en los circuitos internos sin necesidad de que el fabricante de autos deba modificar nada si se respetan las interfaces y las funcionalidades a prestar. Esto se logra porque el equipo de audio, desde el punto de vista de la fábrica de automóviles, es una caja negra, que se testea cerrada, teniendo en cuenta solamente que cumpla con las prestaciones establecidas por un **contrato**, y que tenga las interfaces necesarias para conectarlo.

Trasladando este concepto a la industria del software, el cliente (otro programa, otro objeto), no necesita manipular los datos y las implementaciones de los métodos de los objetos. Utiliza el objeto de acuerdo a lo que indica su interfaz. Esto implica que se pueden usar las interfaces de los tipos de datos definidos por el programador (UDT), pero no la implementación. De este modo, la interfaz se convierte en una especie de **contrato** entre las partes (objeto y cliente), que debe ser respetado.

El ocultamiento implica separar el **qué** del **cómo**. Es decir, el cliente debe conocer **qué** hace el objeto, pero no **cómo** lo hace.

Las ventajas del ocultamiento son:

- permite cambios de implementación
- impide violaciones de restricciones a los datos internos

Interfaz pública

La encapsulación incluye la ocultación de la información:

- Algunas partes son visibles (interfaz pública)
- Otras partes son ocultas (privadas)

El volante representa un interfaz público hacia el mecanismo de giro de un coche

La implementación del volante es privada y sobre ella sólo puede actuar el propio volante

La parte pública o interfaz de una clase describe qué es lo que pueden hacer los objetos de esa clase.
Las partes privadas de una clase describen la implementación de los métodos, es decir, **cómo** hacen lo que saben hacer.

En sistemas orientados a objeto puros, todo el estado es privado y sólo se puede acceder a través de operaciones de la interfaz pública. Por ejemplo, un método *frenar* puede cambiar el estado del atributo *velocidad*.

La ventaja de la ocultación de los detalles de implementación es que el objeto puede cambiar internamente, y la interfaz pública proporcionada continúa siendo compatible con el original. Como consecuencia, los programas que utilizaban el objeto pueden seguir funcionando sin alteración alguna.

Esto es extremadamente útil al modificar código, ya que se restringe la propagación de cambios. También se fomenta la reusabilidad, ya que el código puede ser utilizado como una tecnología de caja negra (como los circuitos integrados en la industria electrónica). Desde el punto de vista económico, esto representa una ventaja de indudable valor.

Interacción entre Objetos: mensajes

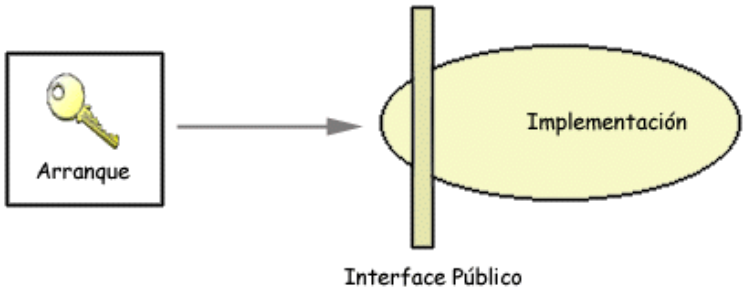
Los objetos contribuyen al comportamiento del sistema colaborando con otros objetos. Por ejemplo:

- El volante interacciona con el objeto intermitente
- El objeto acelerador interacciona con el objeto motor

El modelado de objetos no sólo modela los objetos en un sistema, sino también sus interrelaciones. Para realizar su tarea, un objeto puede delegar trabajos en otro. Este otro puede ser parte integrante de él o ser cualquier objeto del sistema.

Los objetos interaccionan enviándose mensajes unos a otros. El modo de envío de mensajes depende de la naturaleza de los objetos modelados.

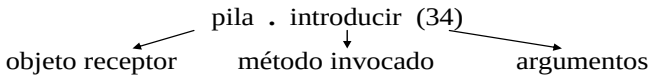
Los mensajes son tratados por los métodos de la interfaz pública del objeto que los recibe. Debido a las normas de encapsulación y de ocultación de la información, no hay otro modo de influir o de comunicarse con un objeto.



Tras la recepción de un mensaje el objeto actuará. La acción puede ser el envío de otros mensajes, cambiar el estado, o hacer cualquier otra cosa apropiada para el objeto en su estado.

Para que un objeto haga algo, se le envía una petición. Esta hace que se active el método apropiado y se ejecute, llevando a cabo la tarea prevista.

El mensaje que constituye la petición contiene el nombre del objeto, el nombre de una operación y, opcionalmente, un grupo de argumentos.



Los objetos pueden ser muy complejos, puesto que pueden contener muchos otros objetos, y éstos a su vez pueden contener otros, etc. Sin embargo, gracias a la propiedad de encapsulación, la persona que utilice un objeto no necesita conocer su complejidad interna, sino sólo la forma de comunicarse con él (a través de mensajes) y la forma en que responde (retorno).

Concepto de Clase

Una clase es una descripción de un conjunto de objetos (o tipo de objeto) con características comunes. Es una abstracción que describe un grupo de instancias con propiedades (atributos) comunes, comportamiento (operaciones) común, relaciones comunes con otros objetos y (lo que es más importante) una semántica común. Así un caballo y un establo tienen los dos un costo y una edad, pero no es probable que pertenezcan a la misma clase. La agrupación o clasificación tiene que tener sentido para el problema en cuestión. Por ejemplo, una clase Transporte puede incluir objetos tren, objetos coche o incluso, objetos caballo. El hecho de que esos objetos funcionen de forma diferente es irrelevante para el problema que se intenta resolver. Si Transporte sólo se preocupa del **traslado de un sitio a otro**, la implementación subyacente no importa, es decir, no importa cómo se traslada el tren o el caballo, si ambos logran el objetivo.

La definición de una clase no crea ningún objeto. Las clases son sólo modelos o plantillas que describen cómo se construyen los objetos.

También se puede ver una clase como un molde, esquema, o patrón que define la forma de sus objetos. Es una estructura estática que define qué estado y comportamiento van a tener los objetos y, a partir de ese esquema, dinámicamente durante la ejecución de un programa, se van a ir creando objetos que pertenezcan a esa clase.

Las Clases son entidades conceptuales que sirven para abstraer y modelar un Sistema. Los problemas se pueden modelar en los términos específicos del dominio. Precisamente, una de las ventajas de la orientación a objetos es que propicia que el problema sea representado en términos específicos del dominio. Esto permite a los diseñadores construir modelos que se proyectan directamente en el problema de la aplicación.
Ejemplos:

Sistema escuela

- Clase Libro
- Clase Pizarra

- Clase Tiza
- Clase Diploma
- Clase Borrador

Sistema Administración de hotel

- Clase Conserje
- Clase Habitación
- Clase Cliente
- Clase Botones
- Clase Mucama

En cada sistema, se modelan los objetos que interactúan en él. La ventaja de crear modelos cercanos al dominio es que son fáciles de comprender, hay menor probabilidad de cometer errores y descuidos, y los modelos pueden ser verificados por los usuarios que definen los requisitos.

Propiedades de una clase: atributos y métodos

Una clase define los atributos y comportamiento de un grupo de objetos con características similares.

Atributos: datos internos de los objetos pertenecientes a una clase, sobre los cuales se realizan las operaciones.

Un atributo en una clase representa un dato contenido en cada una de las instancias de la clase. Cada atributo toma un valor particular en cada uno de los objetos que se crean a partir de dicha clase, conformando el "estado" del objeto.

Distintas clases pueden tener atributos con el mismo nombre (p. ej. nombre, en las clases Persona y Calle) pero cada atributo debe ser único dentro de una clase.

También pueden existir atributos propios de la clase, que son aquellos que tienen un único valor para toda la clase, es decir, el mismo valor para cada uno de los objetos creados a partir de esa clase. Generalmente se los llama **atributos estáticos**. En Java se implementa anteponiéndoles la palabra *static*.

Métodos: también conocidos como operaciones miembro. Son los procedimientos que determinan el comportamiento de los objetos de la clase.

Con el mismo criterio que para atributos, existen los métodos de clase, a los que se llama **métodos estáticos**. En Java se implementa anteponiéndoles la palabra *static*.

Instancias de una clase

Se pueden crear muchos objetos de la misma clase. Cada vez que se construye un objeto a partir de una clase, se crea lo que se llama una **instancia** de esa clase: Objeto = instancia de una clase.

El proceso de instanciación consiste en crear un nuevo objeto (concreto) a partir de un molde o especificación de referencia (abstracto, conceptual). La clase es justamente quién juega este papel de molde para crear el nuevo objeto o instancia. Podemos ver a la clase, en este sentido, como a una fábrica de objetos.

Todos los objetos creados mediante este proceso de instanciación a partir de una misma clase (molde), tendrán las mismas características estructurales y entenderán el mismo protocolo.

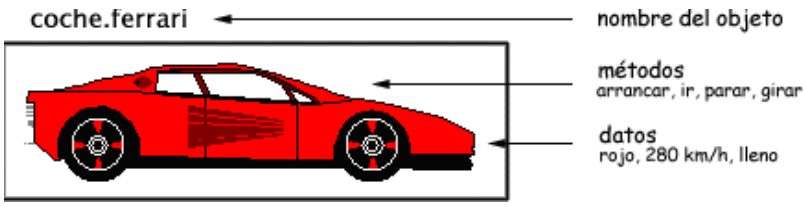
La diferencia entre instancia y clase está en el grado de abstracción:

- un objeto es una abstracción de un objeto del mundo real
- una clase es una abstracción **de un grupo de objetos** del mundo real.

La abstracción a nivel de clase permite la generalización y evita la redefinición de las características (atributos, comportamiento o relaciones) comunes, de forma que se produce una reutilización de estas

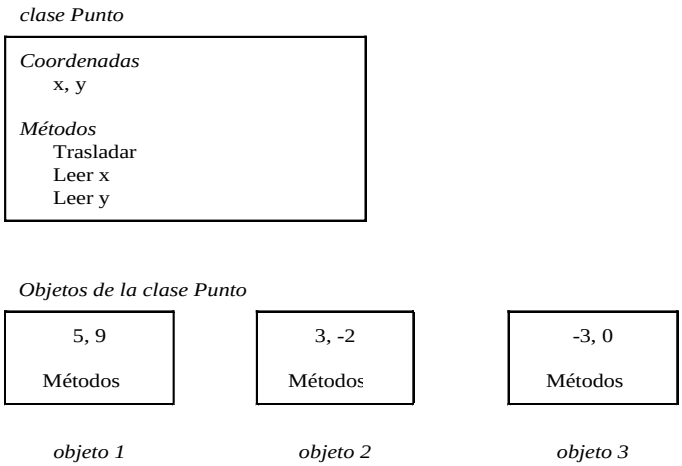
definiciones comunes por parte de cada uno de los objetos. Por ejemplo todas las elipses (instancias) comparten las mismas operaciones para dibujarlas o calcular su área.

Se dice que un objeto **pertenece** a una clase. Existe una diferencia entre los atributos y los métodos en relación a los objetos pertenecientes a una clase:



- hay una copia de los atributos en cada instancia
- hay una **única** copia de los métodos, que es compartida por todas las instancias, que se encuentra en la clase que los agrupa.

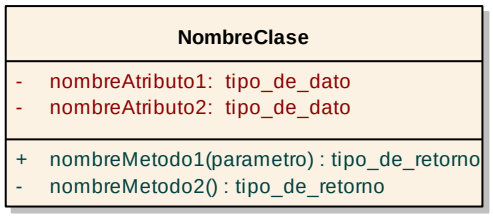
Ejemplo:



El símbolo gráfico para representar clases es un rectángulo. Las clases se representan en los diagramas de clases, que son plantillas que describen un sistema.

El Lenguaje de Modelado Unificado (UML) es en la actualidad el esquema de representación gráfica más utilizado para modelar sistemas orientados a objetos. Fue desarrollado en forma conjunta por Booch, Jacobson y Rumbaugh, tres de los metodologistas más reconocidos en el desarrollo del A/DOO. El UML es el estándar propuesto por el OMG (Object Management Group), organización internacional sin fines de lucro que promueve el uso de la TOO.

La notación UML para clases es la siguiente:



En la parte superior del rectángulo se coloca el nombre de la clase, que comienza con mayúscula.

En el medio se representan los atributos, cuyos nombres se inician con minúscula. Llevan además la visibilidad del atributo (+/-, publico/privado), y el tipo de dato (int, char, float, etc).

En la parte inferior se detallan los métodos, con minúscula, indicando también la visibilidad, el tipo de retorno (int, char, float, etc), y si tuvieran parámetros.

Encapsulamiento y Visibilidad

Un objeto sólo muestra lo necesario para poder interactuar con otros objetos. La interfaz de cada componente se define de forma que revele tan poco como sea posible de sus particularidades interiores.

El encapsulamiento u ocultación de la información se implementa en los lenguajes de POO a través de la visibilidad o acceso a las variables miembro o atributos y a los métodos. La visibilidad es precisamente la propiedad que define si un atributo u operación de un objeto es accesible desde fuera del propio objeto.

Los lenguajes de programación generalmente proporcionan tres niveles de visibilidad: public, protected y private.

Las características públicas (public) son accesibles a cualquier usuario de la clase. Constituyen la interfaz de la clase. Pueden ser tanto métodos como atributos. Sin embargo, no es recomendable que los atributos sean públicos. Para respetar los principios del paradigma de objetos, y aprovechar sus beneficios, se deben ocultar (encapsular) todos sus atributos, **y proveer los métodos de acceso apropiados**.

Las características protegidas (protected) están ocultas al mundo exterior pero son accesibles a la clase que los contiene y a cualquier clase derivada de la misma. Es un status mixto entre public y private.

Las características privadas (private) sólo son accesibles desde los métodos de la propia clase. Esto es lo que se conoce como encapsulamiento. Es el ocultamiento de información de un objeto hacia los demás. El objeto esconde sus datos de los demás objetos y sólo permite el acceso a los mismos mediante sus propios métodos. La gran ventaja del encapsulamiento consiste en que si un módulo u objeto cambia internamente sin modificar su interfaz, el cambio no desencadenará ninguna otra modificación en el sistema.

En los lenguajes OO la encapsulación garantiza que los usuarios de un objeto sólo pueden interactuar con él y modificarlo a través de su interface (conjunto de métodos públicos).

El encapsulamiento evita la corrupción de los datos de un objeto. Si los objetos pudieran ser accedidos de cualquier forma, los datos se podrían corromper o ser utilizados de mala manera. El encapsulamiento protege los datos del uso arbitrario y no pretendido.

El encapsulamiento también oculta los detalles de la implementación de sus métodos. Los usuarios de un objeto conocen las operaciones que pueden solicitar de un objeto, pero desconocen los detalles de cómo se lleva a cabo la operación. Es decir, desconocen la implementación. El encapsulamiento al separar el comportamiento del objeto de su implementación, permite la modificación de ésta sin que se tengan que modificar las aplicaciones que lo utilizan.

Implementación de una clase en Java

```
public class NombreClase {
   //atributos
   private tipo_de_dato nombreAtributo1;
   private tipo_de_dato nombreAtributo2;

   //operaciones
   public tipo_devuelto nombreMetodo1 (parametros){
              cuerpo del método1 (código)
   } // fin del método

   public tipo_devuelto nombreMetodo2 (){
              cuerpo del método2 (código)
   }  // fin del método
} // fin de la clase
```

Ejemplo:

```
public class Punto {
   //atributos
   private int x;
   private int y;

   //operaciones
   public void trasladar(int a, int b){. . . implementación ...}

   public int leerX(){
         return x;
   }
```

```
    public int leerY(){
          return y;
    }
}
```

Métodos Constructores

Los objetos son entidades que existen en el tiempo; por ello, deben ser creados o instanciados. Un objeto (entidad real) se crea a partir de la definición de la clase (molde) a la que pertenece. Esta operación se hace a través de métodos especiales llamados constructores o inicializadores. Pueden existir varios constructores (varias formas diferentes de crear objetos a partir de una clase). En java los constructores son métodos especiales que deben tener siempre el mismo nombre de la clase, y no tienen ningún tipo de retorno.

La instanciación la ejecutará implícitamente el compilador o explícitamente el programador, mediante la invocación a los constructores. Cuando se crea una instancia de una clase, el constructor de la misma es llamado automáticamente.

El propósito de un constructor es inicializar los atributos del objeto de la clase cuando se crea el objeto. Se le asigna un espacio de memoria y un estado consistente.

```
public class Punto {
   //atributos
   private int x;
   private int y;

//método constructor
   Punto(int p_x, int p_y){
         x = p_x;
         y = p_y;
    }

   //operaciones
   public void trasladar(int a, int b){. . . implementación ...}
   public int leerX(){
         return x;
   }

   public int leerY(){
         return y;
   }
}
```

En Java la instanciación se realiza mediante el operador **new**, el cual invoca al método constructor, proporcionándole los parámetros apropiados. Para crear o instanciar un objeto de la clase Punto se utiliza la siguiente instrucción:

```
Punto unPunto = new Punto(2, 5);
```

La Programación Orientada a Objetos

Booch la define como *"Método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son miembros de una jerarquía de clases unidas mediante relaciones de herencia"*.

De esta definición se deriva que la programación sin herencia es explícitamente no orientada a objetos. Es programación con tipos de datos abstractos. Los lenguajes que no ofrece soporte directo para la herencia se conocen como lenguajes **basados** en objetos. Por ejemplo Ada.

Según Cardelli y Wegner (1985), los requisitos que debe satisfacer un lenguaje para ser OO son:

- Soporta objetos que son abstracciones de datos con una interfaz de operaciones con nombre y un estado local oculto
- Los objetos tienen un tipo asociado (clase)
- Los tipos pueden heredar atributos de los supertipos

El elemento fundamental de la POO es el objeto. Se puede definir un objeto como un conjunto complejo de datos y código (algoritmos) que poseen estructura y forman parte de una organización. Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser utilizado en distintos puntos en una organización, o incluso en organizaciones totalmente diferentes que precisen de él. Si el objeto ha sido bien diseñado, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la POO sea muy apta para la reutilización de código.

Los humanos piensan en términos de objetos. Poseen la capacidad de ver imágenes digitales de personas, aviones, árboles, etc, como objetos, en vez de verlos como puntos de colores individuales. Pueden pensar en términos de playas en vez de granos de arena, en bosques en vez de árboles y en casas en vez de ladrillos.

El AOO es un método de análisis que examina los requisitos desde la perspectiva de clases y objetos que se encuentran en el vocabulario del dominio del problema. El propósito del análisis orientado a objetos (AOO) es convertir un problema del mundo real en un sistema informático. En esta etapa crucial se debe decidir qué objetos del mundo real están implicados, y al encontrar los objetos que se necesitan también se les debe agregar atributos comportamiento.

El DOO es un método de diseño que permite describir los modelos lógico y físico, así como los modelos estático y dinámico del sistema que se diseña. En esta etapa se deben buscar los objetos reales que se pueden representar en objetos informáticos, y especificar sus atributos y comportamiento.

Tanto el análisis como el diseño orientado a objetos modela el software en términos similares a los que utilizan las personas para describir objetos del mundo real: aprovechan las relaciones entre las clases, en donde los objetos de cierta clase (como un conjunto de vehículos) tienen las mismas características. También aprovechan las relaciones de herencia, en donde las nuevas clases de objetos se derivan absorbiendo las características de las clases existentes y agregando sus propias características únicas. Un objeto de la clase “Convertible” tiene las características de la clase más general “Automóvil”, además de que el techo del convertible puede ponerse y quitarse.

Estos principios básicos y corrientes son aprovechados por el paradigma de programación orientada a objetos.