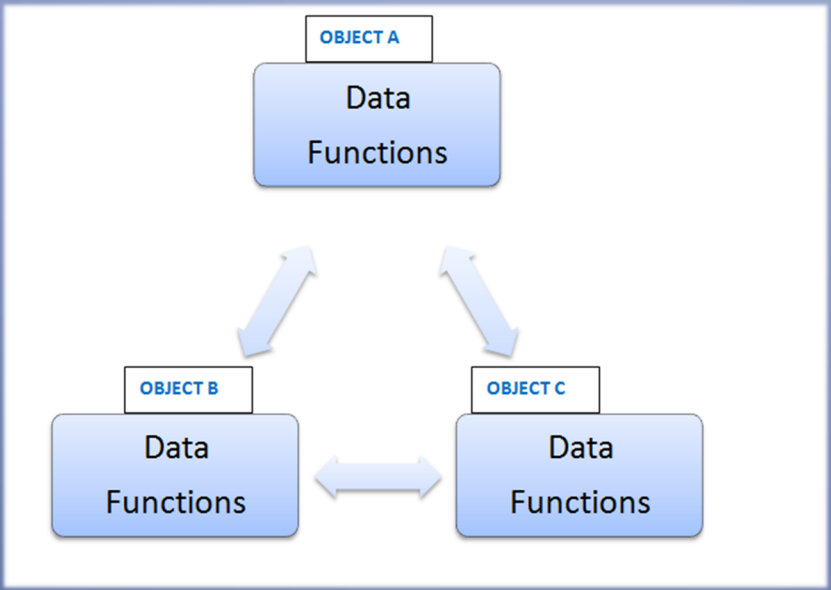
Tema 1



Introducción a la Programación Orientada a Objetos

1. Programación: elementos básicos

1.1. Algoritmos y programas

Se puede definir un *algoritmo* como un conjunto preescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad (*definición extraída de Wikipedia*).

Un algoritmo ha de ser preciso, porque debe dejar muy claro el orden el que se han de ejecutar cada una de las instrucciones que lo conforman.

Asimismo ha de estar bien definido, de cara a que si se ejecuta n veces, en todas llegue al mismo resultado.

Por último la finitud viene dada por el hecho de que el algoritmo tiene que finalizar en algún momento, y por tanto el número de instrucciones que lo componen ha de ser finito.

Un programa se asemeja a un algoritmo, dado que es un conjunto de instrucciones que se introducen en una máquina empleando un determinado lenguaje para que esta nos devuelva un resultado determinado.

Como consecuencia, para llegar a desarrollar un programa es imprescindible haber implementado antes el algoritmo. Los lenguajes de programación no son más que una manera de expresar los algoritmos.

1.2. Lenguaje de programación

La programación es la rama de la informática que engloba al conjunto de operaciones que permiten expresar un algoritmo en forma de programa. Los lenguajes que se emplean para escribir esos programas se llaman lenguajes de programación. Los encargados de esta tarea son los programadores.

Podemos clasificar los lenguajes de programación en cuatro clases:

- Lenguajes de bajo nivel.

Los algoritmos no solo se emplean en programación, sino en cualquier área de la actividad humana donde sea necesario plantear secuencias de instrucciones para llevar a cabo un objetivo.

El trabajo de programador es una tarea compleja que requiere amplios conocimientos de informática, matemáticas, electrónica, etc... así como una gran capacidad de adaptación a los diferentes campos de la industria y el comercio, y un alto nivel de empatía para poder comprender las necesidades del cliente y plasmarlas en el programa.

- Lenguajes de alto nivel.
- Lenguajes de cuarta generación.
- Lenguajes orientados a objetos.

1.2.1. Lenguajes de bajo nivel

Los principales lenguajes de bajo nivel son el lenguaje máquina y el lenguaje ensamblador.

El lenguaje máquina

Como hemos visto en el tema 4, este lenguaje está escrito en el único código que comprende el ordenador, el binario. Todas sus instrucciones son cadenas de 0s y 1s, que especifican las operaciones a realizar y los datos con los que se trabaja, así como las posiciones de memoria donde se pueden encontrar unos y otros.

Entre las ventajas de este lenguaje está la mayor velocidad de ejecución al no tener que traducir el programa a un formato comprensible por la máquina.

Entre sus inconvenientes está que es difícil y lento de codificar, y aún más de depurar –corregir los errores-, además de que no es portable –solo se puede ejecutar en la CPU que disponga del mismo juego de instrucciones con el que se ha implementado el programa-.

Ensamblador	Formato de la instrucción
MOVL Rd, Inm8	00 100 Rd Inm8
MOVH Rd, Inm8	00 101 Rd, Inm8
MOV [Ri], Rs	00 010 Ri Rs 00000
CLR Rds	01 01 100 Rds 000000
ADD Rs Rs1, Rs2	01 00000 Rs Rs1 Rs2
INC Rds	01 01 001 Rds 000000
DEC Rds	01 01 010 Rds 000000
COMP Rs1 Rs2	01 00 111 Rs1 Rs2 000
Br cond desp	11 cond desp
STOP	00 111 00000000000

La tabla anterior nos muestra la correspondencia entre instrucciones de ensamblador y la cadena binaria asociada.

ejemplo ensamblador: Hola Mundo!

```
DOSSEG
.model small
.stack 100h
.data
msgHello DB "Hola mundo!",13,10,"$"
.code
  mov ax,@data              ; Strichpunkt leitet Kommentar ein:
  mov ds,ax                 ; Datensegment initialisiert

  mov dx,offset msgHello    ;
  mov ah,9                  ; Ausgabe eines String
  int 21h                   ;

  mov ax,4C00h              ; Programm beenden
  int 21h                   ;
END
```

Ejemplo de código ensamblador

El lenguaje ensamblador

El lenguaje ensamblador fue el primer intento de sustituir el lenguaje máquina por otro más parecido a los que utilizan las personas. En este lenguaje, las instrucciones consta de una serie de instrucciones conocida como nemónicos, que son más fáciles de recordar que las secuencias binarias.

La mayor ventaja del ensamblador está en la facilidad y mayor rapidez de codificación frente al lenguaje máquina.

Sin embargo, aún se trata de código no portable y obliga a los programadores a dominar, no solo el lenguaje, sino también el funcionamiento interno de la máquina –posiciones de memoria, registros de CPU, etc…-.

Por último, este lenguaje requiere de un compilador, esto es, un programa traductor que convierta los nemónicos en 0s y 1s de cara a que puedan ser ejecutados por la máquina.

En el tema 4 hemos visto un ejemplo sencillo de ciclo de máquina con un lenguaje ensamblador creado ad-hoc.

1.2.2. Lenguajes de alto nivel

Estos lenguajes son posteriores a los de bajo nivel, y se crearon con los siguientes objetivos:

- Conseguir que el programa sea independiente de la plataforma o máquina sobre la que se ejecuta, esto es lo que se llama portabilidad.

- Que los programadores sean capaces de codificar y comprender los programas mucho más fácilmente que si empleasen lenguaje máquina o ensamblador, para ello emplean palabras inteligibles para nosotros, habitualmente en inglés, para que de esta manera la semántica facilite la memorización de estas. Esto redunda en una reducción de la curva de aprendizaje de estos lenguajes, y por tanto en el tiempo necesario para implementar, probar y corregir el programa.

- Disponer de librerías o almacenes donde poder disponer de programas ya implementados, de cara a ser reutilizados para hacer más rápido el trabajo de los desarrolladores –"no inventar la rueda"-. Se trata sobre todo de programas que se ocupan de E/S, manejo de ficheros de datos, funciones matemáticas o del sistema, etc…

Los lenguajes de alto nivel no son inteligibles directamente para el ordenador, sino que, al igual que el ensamblador, necesitan ser compilados para obtener el programa en código máquina.

Logos de varios lenguajes de alto nivel.

```c
#include<stdio.h>
void main()
{
    int opcion;
    printf("1. Capital de Argentina\n");
    printf("2. Capital de España\n");
    printf("3. 10000+58000 = ?\n");
    printf("4. Capital de Uruguay\n");
    scanf("%i",&opcion);
    switch(opcion)
    {
    case 1:
        printf("\n\nBuenos Aires");
        break;
    case 2:
        printf("\n\nMadrid");
        break;
    case 3:
        printf("\n\n68000");
        break;
    case 4:
        printf("\n\nMontevideo");
        break;
    default:
        printf("\n\nOpcion erronea. Intenta de nuevo.");
    }
}
```

Ejemplo de programa codificado en lenguaje C

Logo de PL-SQL de Oracle

Entre los problemas que presentan los lenguajes de alto nivel está el bajo nivel de aprovechamiento de la máquina ya que no se programa directamente sobre los registros de memoria y de la CPU, así como que el tiempo de ejecución es mayor que en código máquina. Por otro lado se emplea mucha más memoria.

Asimismo, la gran cantidad de lenguajes de alto nivel existentes hace imposible manejar a nivel de experto todos ellos.

Algunos de los lenguajes de alto nivel que destacan o han destacado a lo largo de la historia informática son: Fortran, Cobol, Basic, Pascal y C.

1.2.3. Lenguajes de 4º Generación

Los lenguajes de cuarta generación se desarrollaron con el objeto de generar un lenguaje de programación de alto nivel que se pareciera mucho más al lenguaje natural que los lenguajes de 3º generación o alto nivel.

Entre sus ventajas está el poder elaborar programas en menos tiempo, de manera intuitiva y en muy poco tiempo, requiriendo menos conocimientos específicos del lenguaje y de la máquina. En la mayoría de los casos suelen estar formados por una serie de módulos precompilados, que después se pasan a código máquina para obtener el fichero ejecutable.

Entre sus inconvenientes están que los programas desarrollados son menos flexibles que los implementados con lenguajes de alto nivel.

Entre los lenguajes de cuarta generación podemos destacar:

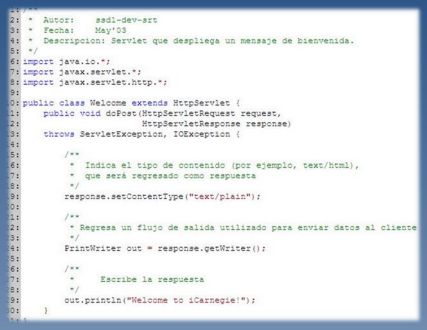
- *SQL*: Se trata de un lenguaje que permite realizar consultas en una base de datos, como vimos en el tema 11. La variante embebida en otros lenguajes es PL-SQL

- *4GL*: Permite generar aplicaciones con ayuda de una gran cantidad de módulos programados en lenguaje C.

1.2.4. Lenguajes orientados a objetos

Los lenguajes orientados a objetos son aquellos que siguen los principios de la Programación Orientada a Objetos –POO-, que estudiaremos a lo largo de este tema. A diferencia de estos, los lenguajes de alto nivel trabajaban según el paradigma de la programación estructurada. Este paradigma es englobado en la POO junto con otros como el de objeto, clase, encapsulación, etc…, que hace a los lenguajes orientados a objetos mucho más capaces de representar el mundo real

Existen varios lenguajes que soportan programación orientada a objetos: Visual Basic, Object Pascal, Smalltalk, C++, Visual C++, Visual J, etc.



```
/**
 *  Autor:    ssdl-dev-srt
 *  Fecha:    May'03
 *  Descripcion: Servlet que despliega un mensaje de bienvenida.
 */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Welcome extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
    throws ServletException, IOException {

        /**
         *  Indica el tipo de contenido (por ejemplo, text/html),
         *  que será regresado como respuesta
         */
        response.setContentType("text/plain");

        /**
         * Regresa un flujo de salida utilizado para enviar datos al cliente
         */
        PrintWriter out = response.getWriter();

        /**
         *     Escribe la respuesta
         */
        out.println("Welcome to iCarnegie!");
    }
```

Imagen que muestra un fragmento de codigo en lenguaje Java.

1.3. Creación de un programa

Todo programa sigue una serie de fases, de manera secuencial y ordenada, desde el planteamiento del problema hasta la puesta en producción de la aplicación: análisis del problema, búsqueda del algoritmo, codificación en el lenguaje de programación elegido, compilación del programa fuente, montaje o linkado del programa, ejecución para comprobar su funcionamiento y, por último, explotación y mantenimiento del programa.

1.3.1. Análisis del problema

El análisis del problema nos permite estudiar la necesidad planteada desde todos los puntos de vista, de cara a tener toda la información necesaria para definir el el proceso que se deberá seguir para lograr la automatización de la solución.

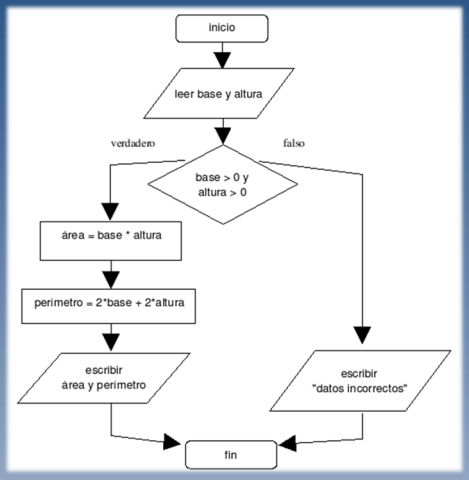
Veamos el ejemplo de pseudocódigo de un programa que, dados dos números introducidos desde el teclado, nos permite visualizar el mayor de los dos.

```
Leer de teclado NUM 1 y NUM 2
SI NUM 1 > NUM 2
Visualizar "Mayor =" NUM 1
SI NO
Visualizar "Mayor =" NUM 2
Fin
```

1.3.2. Búsqueda del algoritmo

El desarrollo del algoritmo que va a permitirnos resolver el problema planteado se realiza en esta fase, y es fundamental para que el programa funcione

correctamente. Para ello se emplean diferentes técnicas, tales como los ordinogramas o el pseudocódigo.



Ejemplo de un pseudocódigo de una operación de cálculo del área y perímetro de un polígono rectangular.

1.3.3. Codificación del programa

La codificación del programa no es más que la traducción del algoritmo a un lenguaje de programación en concreto, lo cual nos da como resultado un conjunto de instrucciones almacenadas en un fichero llamado código fuente. Es una fase relativamente sencilla, dado que el núcleo del trabajo, la obtención del algoritmo, ya está hecho.

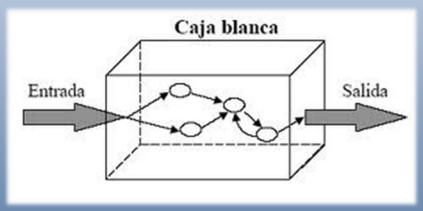
1.3.4. Compilación del programa

Una vez escrito el programa fuente, éste se traduce mediante el compilador a código máquina, apareciendo uno nuevo, denominado programa objeto.

En esta fase es cuando se corrigen los errores producidos en la fase de codificación de las instrucciones del algoritmo.

1.3.5. Montaje o linkado

El linkado permite enlazar el programa con las librerías necesarias para que las instrucciones del código funcionen correctamente, y crear con ello el fichero ejecutable.



Las pruebas de caja blanca nos permiten evaluar todas las posibles secuencias de instrucciones que se pueden ejecutar en un programa.

1.3.6. Fase de pruebas

EN la fase de pruebas de ejecuta el programa empleando diferentes baterías de datos de entrada con el fin de encontrar fallos de funcionamiento, de cara a corregirlos.

Caso de que no se haga se corre el riesgo de que el programa falle durante su paso a producción, lo cual podría provocar serios problemas en la organización en la que fuera implantado.

1.3.7. Fase de explotación y mantenimiento

En esta fase se tiene el programa en producción, y en función de las nuevas necesidades de la organización, se llevan a cabo las modificaciones oportunas de cara a optimizar su rendimiento.

2. Programación Orientada a Objetos (POO)

2.1. Definición de POO

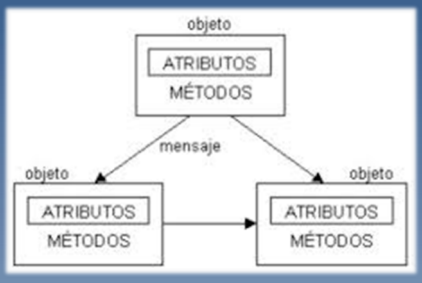
Como hemos explicado anteriormente, la POO, o Programación Orientada a Objetos se basa en la representación de los elementos de un programa como objetos de diferentes clases.

Hemos comentado en un punto anterior que los programas constan de dos elementos básicos, el código y los datos. La programación puede organizarse en torno a cualquiera de esos dos elementos.

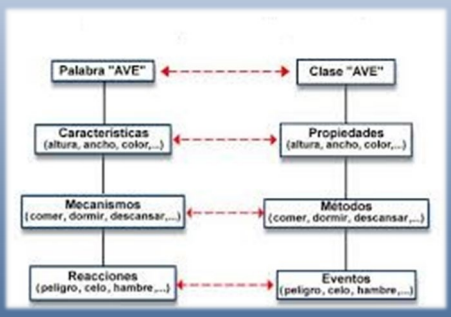
En el caso de que la programación se organice en torno al código, hablamos de programación orientada a procesos o programación estructurada. En este caso se plantea el programa como código que actúa sobre los datos. Es el esquema que emplean los lenguajes de alto nivel o 3º generación.

Si la programación se organiza en torno a los datos, hablamos de programación orientada a objetos –POO-. En este caso son los datos los que controlan el código.

En la POO la programación se basa en definir objetos, sus características y los métodos que modifican estas características así como la comunicación con otros objetos.



La esencia de la POO es la representación de la realidad mediante un conjunto de objetos que interactuan entre ellos a través de mensajes.



Una clase se puede considerar como un equivalente a definir un elemento del mundo real.

2.2. Clase

2.2.1. Definición

La base de la POO son las clases. Las clases son elementos que representan las habilidades y propiedades de los objetos de la vida real que el programa tiene que emplear.

Podríamos considerar una clase como un molde de un objeto –por ejemplo una llave- de la que podemos obtener infinitas instancias de este.

Una clase se representa como código mediante un bloque de sentencias encerradas entre delimitadores de inicio y fin, que contiene las propiedades y métodos que deberán tener todos los objetos que se instancien a partir de esa clase.

2.2.2. Variables

Para definir las propiedades de una clase debemos definir los campos donde se almacenarán las características de los objetos de esa clase.
Los nombres de esos campos se denominan *identificadores* o variables. En Java pueden estar formados por letras, números y símbolos de subrayado, empezando siempre por una letra. Java es sensitivo a mayúsculas, como C –en cuya sintaxis se ha inspirado-.
Al declarar cada campo estamos reservando espacio de memoria, que dependerá del tipo de dato que se vaya a almacenar ahí.
Las variables que se definen pueden ser *de instancia*, que almacenan información que puede ser diferente para cada objeto, o *estáticas*, que tendrán el mismo valor para todos los objetos de esa clase.

2.2.3. Métodos

En POO los métodos representan los comportamientos y habilidades de una clase. Un método está compuesto por dos elementos:

- *Una cabecera*: Aquí se definen el tipo de datos que devuelve, el nombre, el ámbito de acceso –desde donde se le puede invocar- y una lista de parámetros de entrada.
- *Un cuerpo:* Consta de una secuencia de instrucciones englobadas entre un delimitador de inicio y otro de fin, a las que se asigna un nombre.

Las instrucciones en un método pueden ser de diferentes tipos:

- *Llamada:* sirven para ejecutar un método
- *Asignación*: asignan a una variable un valor determinado, bien almacenado en otra, o bien definido explícitamente
- *Repetición*: repiten n veces una misma instrucción o grupo de instrucciones
- *Selección*: permiten elegir la ejecución de una u otra instrucción en función del cumplimiento o incumplimiento de una condicion
- *Excepción*: Se ejecutan en casos especiales, como interrupciones del sistema, gestión de errores, etc…

El nombre de un método sigue las reglas de los identificadores de variables, aunque se sigue la norma de que la primera letra será minúscula.

Cuando un método es invocado de cara a su ejecución, se sustituyen los parámetros formales de la lista de parámetros por datos reales. Para invocar a un método se escribe en primer lugar el nombre del objeto instanciado de la clase en cuestión y luego, separado de un punto, el nombre del método.

2.2.4. Constructores

Los constructores son métodos cuya función es instanciar un objeto de la clase a la que pertenecen. Los constructores pueden inicializar objetos asignando unos valores por defecto a sus atributos.

Tienen el mismo nombre de la clase de la que forman parte.

2.3. Objeto

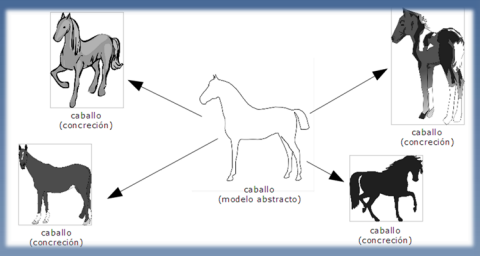
2.3.1. Definición

Un objeto es la instanciación de una clase, o sea un ejemplar de esta: por ejemplo, el roble que hay delante de mi casa es una instanciación de la clase Roble.
Los objetos tienen dos características básicas, que son el estado y el comportamiento. El primero nos lo definen los valores que toman sus atributos o propiedades, mientras que su comportamiento lo definen los métodos.

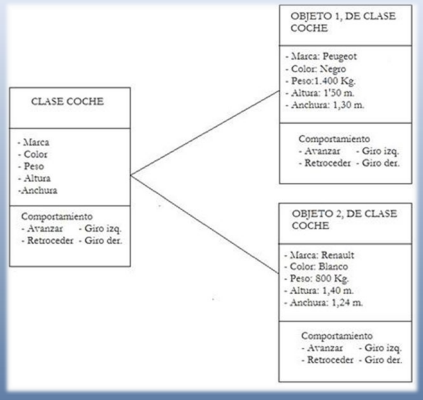
2.3.2. Creación

En POO la creación de un objeto se hace, como se ha comentado, instanciando una clase. Para ello necesitamos una variable a la que asignaremos el objeto instanciado de esa clase.
El proceso de creación de un objeto constará de dos fases:

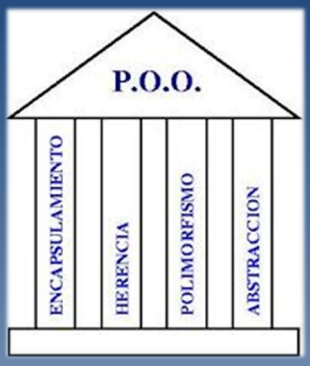
- *La declaración del objeto*, donde escribiremos el nombre de la clase y detrás la variable que representará el objeto



En la imagen se puede observar como la clase –Caballo- se instancia cuatro veces en otros tantos objetos –razas- con diferentes valores para sus atributos.

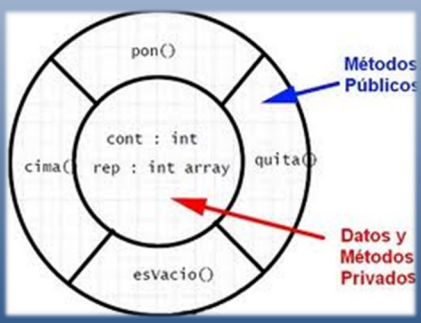


A partir de una clase –*Coche*- se pueden crear diferentes objetos, con valores distintos para sus atributos –*Objeto 1* y *Objeto 2*-



Los cuatro elementos básicos de la POO son la encapsulación, la herencia, la abstracción y el polimorfismo.

La abstracción permite a cada persona comprender el universo que le rodea en función de lo que de él considera esencial.



La encapsulación emplea métodos publicos como interfaz para modificar de manera ordenada los datos privados. Actua, por tanto, como un elemento de filtrado y control de acceso al contenido del objeto.

- *La creación del objeto propiamente dicha,* donde emplearemos el constructor par reservar el espacio de memoria para los atributos y métodos del objeto, así como los valores por defecto de los primeros.

2.3.3. Referenciación

Una vez creado el objeto, para usar los atributos o los métodos de este debemos referenciarlos poniendo el nombre del objeto delante y, separado por un punto, el nombre del atributo o del método con el que deseamos trabajar.

2.4. Principios de la orientación a objetos

2.4.1. Abstracción

La abstracción es el procedimiento por el cual se toman los elementos fundamentales de un objeto que lo distinguen de los demás, dejando de lado los detalles accesorios. Es esencial para comprender la globalidad de una situación, sin perderse en lo irrelevante.

La abstracción como mecanismo que implica la descomposición de un sistema complejo en subsistemas más sencillos, empleando para ello clasificaciones jerárquicas.

2.4.2. Encapsulación

La encapsulación consiste en ocultar al exterior todos aquellos elementos del objeto que no se considere imprescindible mostrar.

Permite disponer de una interfaz –que toma la forma de métodos- a través de la cual el objeto se comunica con el resto de los objetos del sistema, y por otro lado, mantiene ocultos los detalles de la implementación del código –métodos, atributos, etc…- permitiendo la manipulación de estos únicamente a través de la interfaz.

Para conseguir esto debemos de definir todos aquellos elementos que no queremos que se vean desde el exterior como privados –private- indicando como públicos –public- los métodos que conformen la interfaz del objeto.

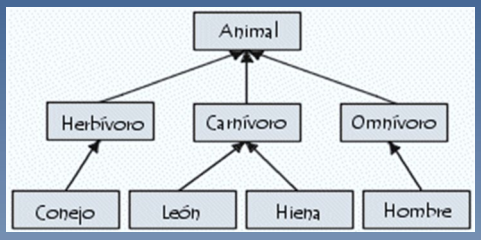
Los métodos públicos han de definirse de tal manera que no pongan en riesgo la implementación interna de una clase.

2.4.3. Herencia

La herencia es el proceso por el cual definimos una clase, que a su vez puede ser padre de otras clases, que heredarán de ella atributos y métodos, independientemente de los métodos y atributos propios que se definan en ellas.

De no existir la herencia, tendríamos que redefinir en cada clase todas las características que deseásemos incluir. Con la herencia, solo hay que añadir las características propias -métodos y atributos- de esa clase que la hacen diferente de las demás, además de poder redefinir las características heredadas si fuera necesario.

En Java, la clase de la que se hereda se denomina superclase, y la que hereda se llama subclase.

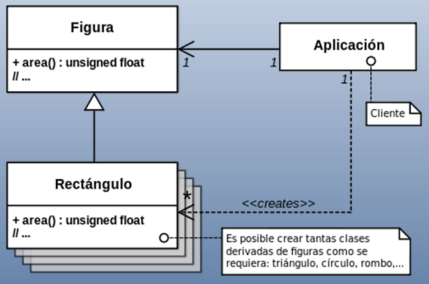


En el esquema de la figura podemos ver un esquema de herencia en el que los hijos –herviboro, carnívoro, omnívoro- heredan de su padre –animal- las caracteristicas propias de los mamíferos, mientras que las propias –el tipo de alimentación- se definen en cada uno de ellos.

2.4.4. Polimorfismo

El polimorfismo permite emplear un mismo mensaje a objetos de clases distintas, de tal manera que ese mensaje tendrá efectos diferentes, dará lugar a la ejecución de distintos métodos y modificará distintos atributos.

En POO esto nos permite que un método, que tiene una implementación determinada –habitualmente es abstracto, o sea vacío- en la clase padre, tenga diferentes implementaciones en cada una de sus clases hijas.



En este caso, el método área, que no tiene ningún contenido en la clase padre Figura, posee diferentes implementaciones en las clases hijas –Rectángulo, Triángulo, Círculo, etc…-

2.5. Mensajes

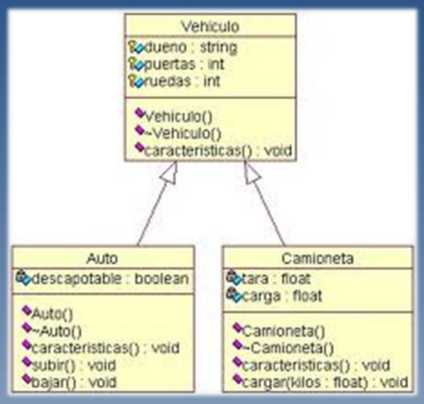
Una de las claves de la POO es la representación de la realidad mediante objetos que interactúan entre sí. Para ello tienen que enviarse entre ellos mensajes y como respuesta a

ellos, tenga lugar la ejecución de los métodos de estos objetos.

Los mensajes han de incluir el nombre del objeto al que se dirigen, el método que se desea ejecutar y los parámetros de entrada para dicho método.

2.6. UML

El desarrollo de un programa incluye, como hemos visto, las fases de análisis y diseño del software. Para poder crear la documentación necesaria para estas fases se desarrollaron diferentes tipos de notaciones, siendo la más exitosa UML –*Unified Modeling Language*-, que es un lenguaje de modelado que permite, a partir del diseño de la aplicación, generar código. En el cuadro de la izquierda podemos ver algunos ejemplos de modelado de UML en los que se representan objetos, clases, herencia, etc….



En este ejemplo podemos ver un diagrama de clases en el que se representa una herencia desde una entidad padre –Vehículo- a dos entidades hijas –Auto y Camioneta-.

Se puede observar como los nombres de las clases empiezan por mayúscula, los atributos se indican con su nombre y tipo de dastos, y los métodos se indican con paréntesis –donde estarán los parámetros en caso de haberlos- y después el tpo de datos que devuelven –void quiere decir que no devuelven nada-.