

**Asignatura:**

Programación 2

**Código de Asignatura:**

301-00070

**Facilitador:**

Leonardo Esqueda

**Investigación #1**

Tipos de clases.

**Estudiante:**

Samir Ramos

3-739-672

**Fecha de entrega:**

Jueves 5 de Julio de 2018

**Resumen**

Una parte muy importante de la programación orientada a objetos son las clases, si no fuera por ellas ni siquiera habría objetos. Al momento de crear una clase, java nos da la oportunidad de elegir entre distintos tipos de clase que queramos para la nuestra. Cada una con sus propias características y ventajas. Un programador que conoce estas características sabe cuando debe usar una y no otra, de manera que su desarrolla un buen software. Los tipos de clases son:

C. Public : Son muy comunes, accesibles desde cualquier otra clase enla misma librería (de otro modo hay que importarlas).

C. Abstract : Aquellas que tienen por lo menos un método abstracto. No implementan sus métodos, sino que dan las bases para que sean implementados en la herencia.

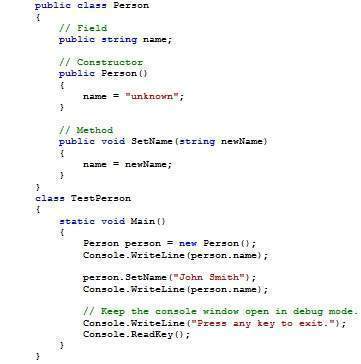
C. Final : Son las que terminan la cadena de herencia. Útiles por motivos de seguridad y eficiencia de un programa, ya que no permiten crear más sub-divisiones por debajo de esta clase.

C. Synchronizable :Especifica que sus métodos son sincronizados evitando problemas con los thread, de forma que estos no pueden empezar a correr un método si no a acabado el otro.

## Características fundamentales

* **Nombre de la clase**. Sirve para identificar a todos los objetos que tengan unas determinadas características.
* **Conjunto de atributos**. Datos miembros. El valor de los atributos representan el estado de cada objeto.
* **Conjunto de métodos**. [Funciones](https://www.ecured.cu/index.php?title=Funciones&action=edit&redlink=1) miembro. Permite que los objetos cambien de estado, dependiendo del estado anterior que tuviera el objeto.
* **Niveles de acceso** para proteger ciertos miembros de la clase. Normalmente, se definirán como ocultos (privados) los [atributos](https://www.ecured.cu/index.php?title=Atributos&action=edit&redlink=1) y visibles (públicos) los métodos.

**Ejemplo**



**Interfaz**

En [programación orientada a objetos](https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos), una **interfaz** (también llamada **protocolo**) es un medio común para que los [objetos](https://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n)) no relacionados se comuniquen entre sí. Estas son definiciones de [métodos](https://es.wikipedia.org/wiki/M%C3%A9todo_(inform%C3%A1tica)) y valores sobre los cuales los objetos están de acuerdo para cooperar.

Por ejemplo, en [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)) (donde se le llama [Interfaz](https://es.wikipedia.org/wiki/Interfaz_(Java)) *(Java interfaces)*, la interfaz Comparable especifica un método compararCon() que las clases deben implementar. Esto significa que un método aparte de ordenamiento, por ejemplo, puede ordenar cualquier objeto que implemente la interfaz Comparable, sin tener que saber nada sobre la naturaleza interna de la clase (excepto que dos de estos objetos pueden compararse mediante compararCon()).

El protocolo es una descripción de:

1. Los mensajes que el objeto entiende.
2. Los argumentos con los que estos mensajes pueden ser suministrados.
3. Los tipos de resultados que devuelven estos mensajes.
4. Los [invariantes](https://es.wikipedia.org/wiki/Invariantes_de_clase) que se conservan a pesar de las modificaciones en el estado de un objeto.
5. Las situaciones excepcionales que serán requeridas para ser manejadas por los clientes al objeto.

Veamos a continuación un ejemplo en lenguaje Java en el que creamos una interfaz Empleando que es implementada por las clases Carpintero y Fontanero, las cuales además heredan de Persona.

Main.java:

1. package EjemplosPOO.ejemplo03;
2. public class Main
3. {
4. public static void main(String[] args)
5. {
6. Carpintero c = new Carpintero();
7. Fontanero f = new Fontanero();
8. c.setNombreCompleto("Pedro Pérez Perera");
9. c.trabajar();
10. f.setNombreCompleto("Antonio Miranda Mena");
11. f.trabajar();
12. }
13. }

**Ámbito de las variables**

Una **variable de instancia**es una variable definida para las instancias de una clase  
(cada objeto tiene su propia copia de la variable de instancia).

Una **variable estática**es una variable definida para la clase  
(compartida entre todas las instancias de una clase).

Una **variable local**es una variable definida dentro del cuerpo de un método.

• El ámbito de una variable de instancia abarca todos los métodos no estáticos de una clase:

o Cuando es privada, todos los métodos pueden acceder al valor almacenado en la variable de instancia.

o Cuando es pública, se puede acceder a ella desde cualquier lugar en el que se disponga de una referencia a un objeto de la clase.

* El ámbito de una variable estática:  
  o Si es privada, cubre todos los métodos estáticos de la clase

en que está definida.  
o Si es pública, abarca todos los métodos estáticos de todas

las clases que formen parte de la aplicación.

* El ámbito de una variable local comienza en su declaración y termina donde termina el bloque de código ({}) que contiene la declaración.

El ámbito de una variable es la parte del programa en la que podemos hacer referencia a la variable

Modularización - 24 - © Fernando Berzal

**Uso de variables locales**

En Java, las declaraciones se pueden poner en cualquier parte del código de un método, no necesariamente al principio:

void method () {

int i=0;

while (i<10) { int j=0; ...

// Declara e inicializa **i**

// **i** está definido aquí // Declara **j**// **i** y **j** definidos  
// **j** ya no está definido

}  
System.out.print(i); // **i** todavía está definido

} // **i** deja de estar definido

De todas formas, nosotros declararemos siempre todas las variables locales al comienzo del cuerpo del método.

* Las variables locales han de declararse antes de utilizarse.
* Se pueden declarar variables locales con el mismo nombre en

diferentes bloques de código.

Incluso se podrían declarar en bloques de código no anidados dentro de un mismo método, aunque no es recomendable hacerlo.

IMPORTANTE

Las variables de instancia se inicializan automáticamente al crear un objeto (a 0 o null), mientras que las variables locales de un método tenemos que inicializarlas nosotros antes de usarlas.