**Desarrollo de software** → proceso de abstracción que permite construir un modelo para la solución de un problema.

P.O.O → paradigma que guía el proceso de desarrollo de software.

**Principio fundamental:** desarrollar un sistema de software en base a las entidades relevantes del problema que le da origen.

→ Objetos del problema.

→ Cada objeto del problema es una ENTIDAD que puede caracterizarse a través de sus atributos y su comportamiento.

Atributos y comportamiento agrupan a los objetos en **CLASES.** → Define los atributos que caracterizan a un conjunto de objetos.

# OBJETOS Y CLASES

Modelo: representación que permite describir/explicar/analizar un sistema o proceso a partir de sus entidades relevantes y el modo en que se relacionan.

- Mecanismo fundamental: ABSTRACCIÓN. →
  Identificación de entidades y su caracterización de acuerdo a sus propiedades y al modo en que interactúan con las demás.
- Mecanismo fundamental: CLASIFICACIÓN. →
  Agrupar entidades a partir de sus semejanzas y diferencias en base a diferentes criterios.

El principio fundamental del paradigma de la P.O.O es construir un sistema de software con base en las entidades de un modelo elaborado a partir de un proceso de abstracción y clasificación.

**OBJETO** → entidad (física o conceptual) caracterizada a través de atributos y comportamiento. Este último queda determinado por un conjunto de servicios que el objeto puede brindar y de sus responsabilidades.

Cuando el sistema de software está en ejecución se crean objetos de software.

- Modelo/representación de un objeto del problema.
- Tiene identidad/estado interno y recibe mensajes.

Los objetos se comunican a través de mensajes para solicitar y brindar **servicios.**

Su ejecución puede modificar E.I. del objeto. ←

**El modelo computacional propuesto por la P.O.O. es un mundo poblado de objetos comunicándose a través de mensajes.**

Objetos del problema — entidades identificadas durante el desarrollo de requerimientos del diseño.

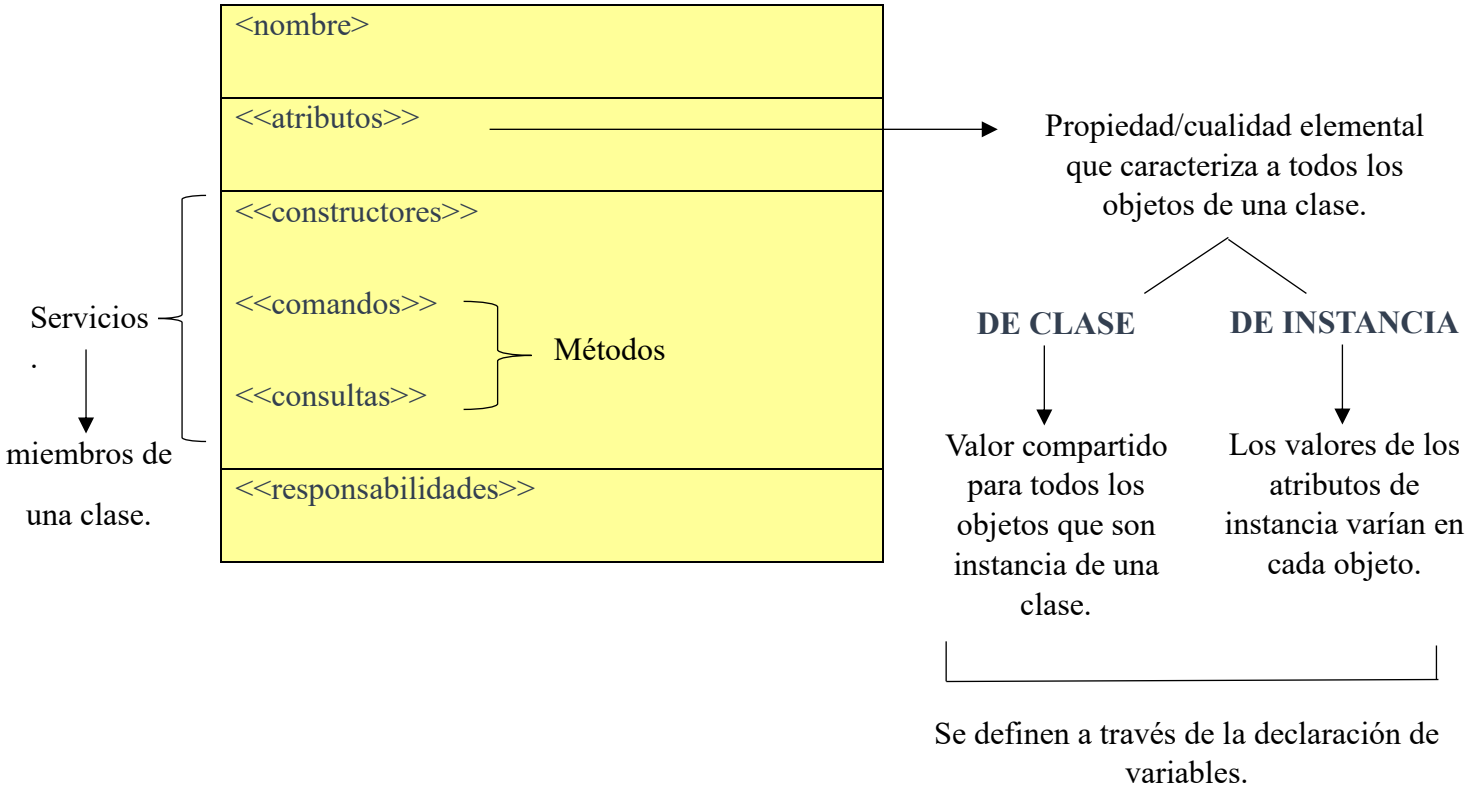
Objetos de software — cada representación que modela en ejecución a una entidad del problema.

**CLASES**

- Todos los objetos que son instancia de una clase van a estar caracterizados por los mismos atributos.
- Patrón que establece los atributos y el comportamiento de un conjunto de objetos.
- Modulo de software que puede construirse, verificarse y depurarse con cierta independencia a los demás.

Desde el punto de vista **estático**, un sistema de software orientado a objetos es una colección de clases relacionadas, mientras que, desde el punto de vista **dinámico**, un sistema de software orientado a objetos es un conjunto de objetos comunicándose.

Diagrama de clases: representación visual que permite modelar la estructura de un sistema de software a partir cd las clases que lo componen.




**Constructores:** se invocan para crear un objeto y poseen el mismo nombre que la clase.

↳ Inicializan el valor de los atributos de instancia.

- Si una clase no define explícitamente un constructor, el compilador crea automáticamente uno.
- Puede haber varios constructores (diferente numero o tipo de parámetros).

**Comandos:** al ejecutarse modifica el valor de 1 o mas atributos del objeto que recibió el mensaje.

**Consultas:** al ejecutarse no modifica el valor de ningún atributo.

↳ Generalmente retornan un valor.

Un lenguaje orientado a objetos permite retener, durante la implementación, la organización de las clases modeladas durante el desarrollo de requerimientos y la etapa del diseño.

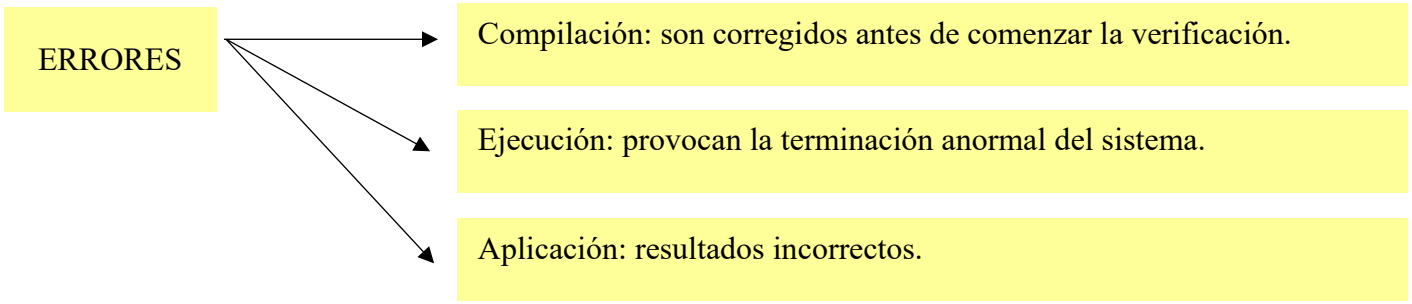
**Obtener:** consultas que retornan el valor de un atributo.

**Establecer:** comando que modifica el valor de un atributo.

- PRIVATE: modificador que indica que las variables solo son visibles y pueden ser usadas dentro de la clase.
- STATIC: modificador que indica que todos los objetos de clase comparten el mismo valor en cada umbral.
- FINAL: modificador que indica que las variables poseen valores constantes.

En java la ejecución comienza cuando se invoca el método "**main**" de una clase especifica.

La clase que define al método "main" no modela a objetos del problema.

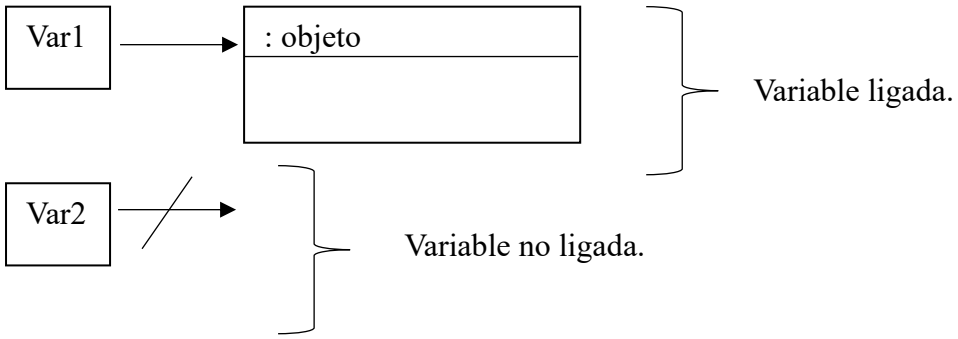



| Error en compilación | Error en ejecución |
|---|---|
| Invocar a un método que no esta declarado en la clase del tipo estático de la variable. | Error de casteo. |
| Asignación polimórfica mal hecha.<br>Ej: HIJA aux = new PADRE; | Acceder a una componente del arreglo que no existe. |
| Crear una instancia de una clase abstracta.<br>Ej: ABSTRACTA aux = new ABSTRACTA; | Mandar un mensaje a una variable no ligada. |
| | Dividir por cero. |

**Clase tester**: verifica los servicios de una o mas clases para un conjunto de clases de prueba.

En java los objetos son referenciados a través de variables.

Una variable ligada mantiene una referencia del E.I. de un objeto.




Una clase define un **tipo de dato** a partir del cual se pueden declarar variables.

Establece un conjunto de valores y operaciones que se aplican sobre estos valores.

La declaración de una variable establece su nombre, tipo y alcance.

Visibilidad ← ↓

El segmento de código en el cual puede ser usada.

En java una variable declarada local a un bloque se crea en el momento que se ejecuta la instrucción de declaración y se destruye cuando termina el bloque que corresponde a la declaración.

Cada bloque crea un nuevo **ambiente de referenciamiento**, formado por todos los identificadores que pueden ser usados.

**MENSAJES Y MÉTODOS**

Cuando un objeto recibe un mensaje, el flujo de control se interrumpe y el control pasa al método que se liga al mensaje.

- TOSTRING: retorna la concatenación de los valores de los atributos del objeto que recibe el mensaje.

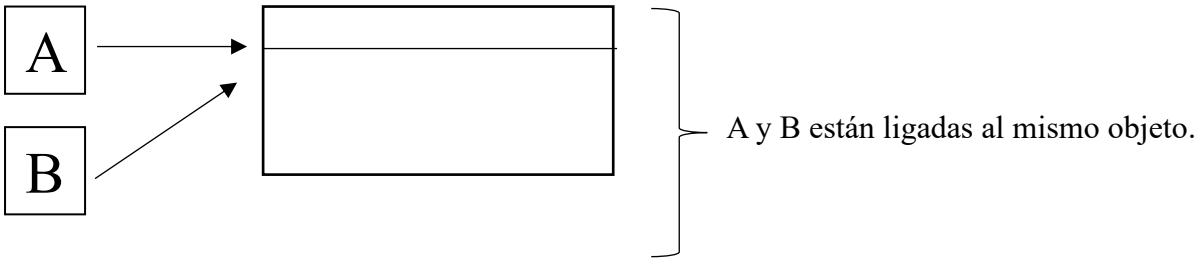
Parámetros y resultados de tipo de clase

Un método puede recibir como parámetro o retornar como resultado a un objeto. En ambos casos la variable es de tipo clase.

- Tipo elemental: int/char/boolean/float.
- Tipo clase.

Una variable de tipo elemental almacena un valor de su tipo mientras que una variable de tipo clase almacena una **referencia** a un objeto de software de su clase.

¡PASAJE DE PARÁMETROS POR VALOR!




## Identidad, igualdad y equivalencia

La referencia a un objeto puede ser usada como propiedad para identificarlo.

A= =B ——→ el "= =" compara el estado interno.

Para comparar el E.I. de los objetos se comparan los valores de los atributos de instancia.

EQUALS —— compara E.I. del objeto que recibe el mensaje con el E.I. del objeto que pasa como parámetro.

COPY ——— modifica el E.I. del objeto que recibe el mensaje con el E.I. del que pasa como parámetro.

CLONE —— crea un nuevo objeto con el mismo E.I. del objeto que recibe el mensaje.

Objeto que recibe mensaje: se accede a su E.I. directo a través de los atributos de instancia.

Objeto que pasa por parámetro: se accede a su E.I. a través de las operaciones provistas por la clase.

- "=": dos objetos tienen la misma identidad.
- "equals": dos objetos son equivalentes.

# ASOCIACION Y DEPENDENCIA ENTRE CLASES

Las entidades de una clase semejantes de acuerdo al criterio elegido y además se relacionan de la misma manera con las entidades de otras clases.

La ASOCIACIÓN es una relación entre clases que se produce cuando el modelo de un objeto del problema contiene o puede contener al modelo de otro objeto del problema.

↳ Relación TIENE_UN (PE: un objeto de clase A tiene un atributo de clase B).

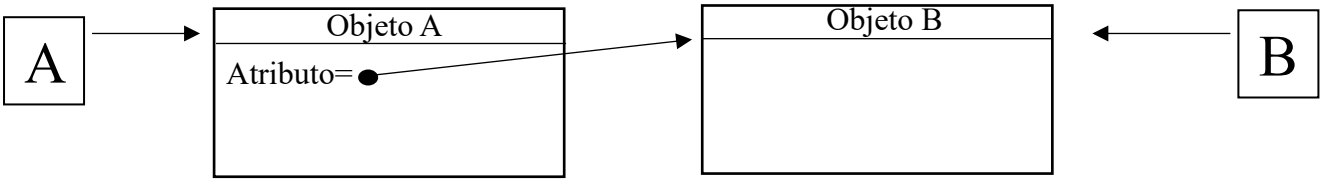
La relación de DEPENDENCIA se produce cuando el modelo de un objeto del problema usa al modelo de otro objeto.

↳ Relación USA_UN (PE: un objeto de la clase A usa un objeto de la clase B).

| Dependencia entre clases | Asociación entre clases |
|---|---|
| Cuando una clase declara una variable local o un parámetro de otra clase, decimos que existe una dependencia entre la primera y la segunda y surge una relación del tipo "usa un" entre los objetos. | Cuando una clase tiene un atributo de otra clase, ambas clases están asociadas y la relación es de tipo "usa un". |

Una relación "tiene un" con frecuencia provoca una relación de tipo "usa un".

Representación por referencia: el E.I. de un objeto contiene referencias a los objetos de las clases asociadas.




Igualdad y equivalencia entre objetos de clases asociadas.

Cuando una clase está asociada a otra, la igualdad, copia y clonación se pueden hacer en forma **superficial** o en **profundidad**.

SUPERFICIAL

- **Equals**: se evalúa si están ligados a una misma instancia de la clase asociada
- **Copy**: modifica E.I. del objeto que recibe el mensaje con los valores almacenados en el objeto que pasa como parámetro.
- **Clone**: retorna un objeto que tiene exactamente el mismo E.I. que el objeto que recibió el mensaje

PROFUNDIDAD

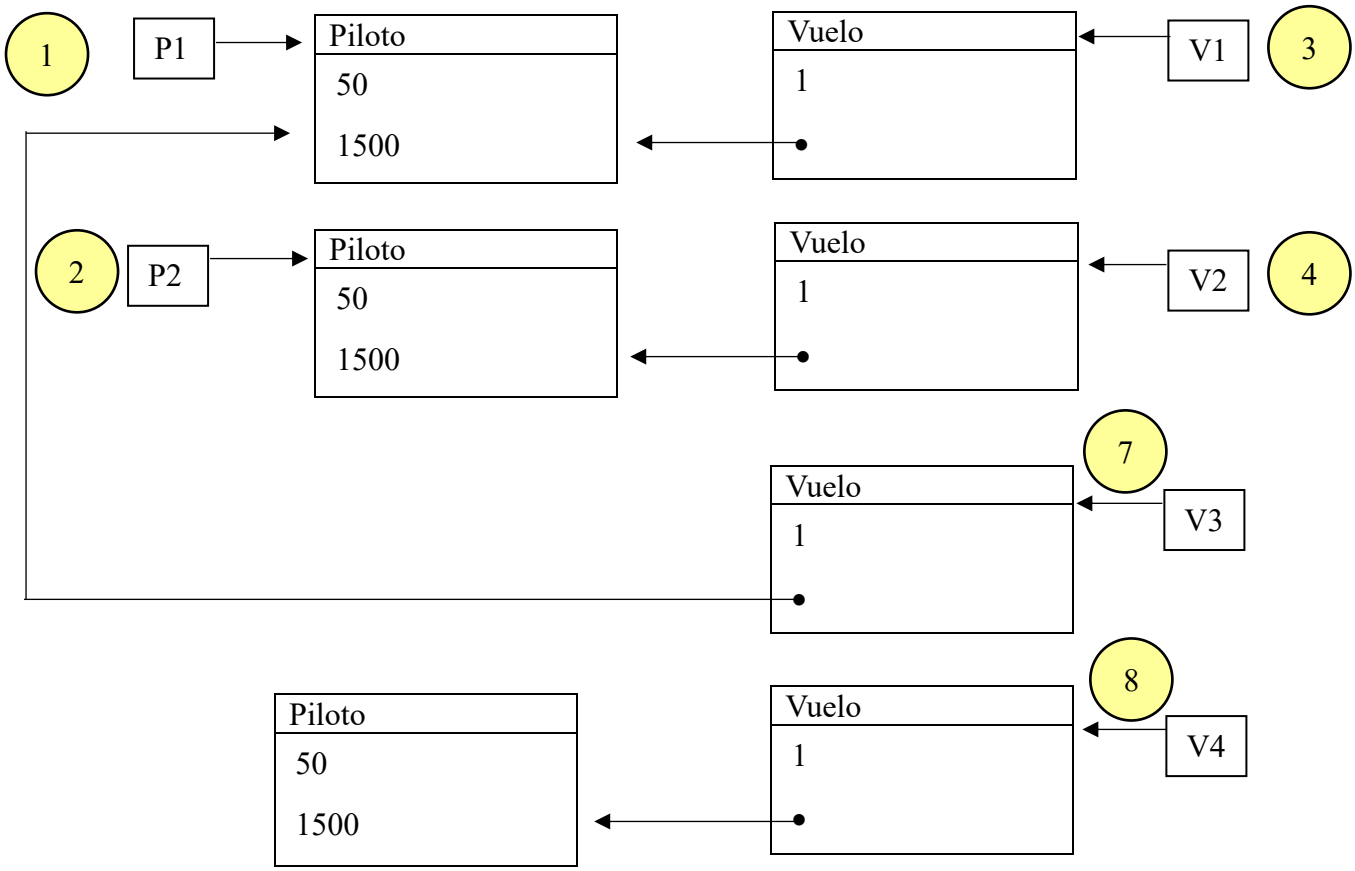
**Equals**: requiere que los E.I. de los objetos asociados sean equivalentes, aunque las referencias sean diferencias.

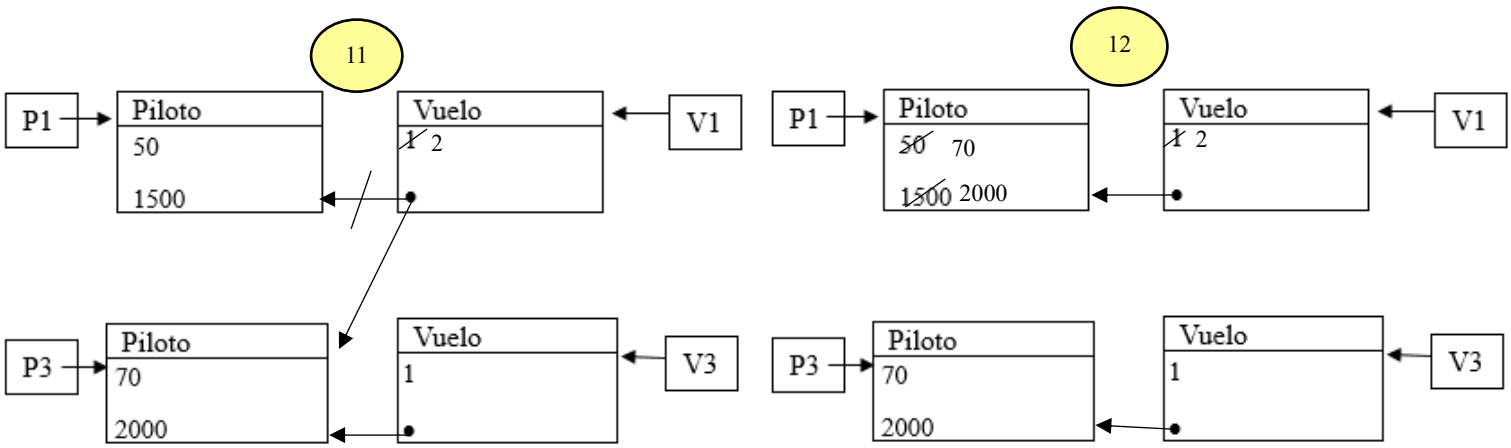
**Copy**: el E.I. del objeto que recibe el mensaje se modifica con los valores de los atributos del objeto que se recibe como parámetro, excepto las referencias que no se modifican, pero si se modifica a el E.I. de los objetos asociados.

**Clone**: retorna un objeto que tiene exactamente el mismo E.I. que el objeto que recibió el mensaje excepto la referencias, que estarán ligadas a clones de los objetos asociados.

COPY-CLONE-EQUALS EN PROFUNDIDAD Y SUPERFICIAL

1. Piloto P1 = new Piloto (50, 1500);
2. Piloto P2 = new Piloto (50, 1500);
3. Vuelo V1 = new Vuelo (1, P1);
4. Vuelo V2 = new Vuelo (1, P2);
5. V1. equals(V2) ⟶ false (diferentes referencias). ⟶ EQUALS SUPERFICIAL.
6. V1. equals(V2) ⟶ true (mismo E.I.). ⟶ EQUALS PROFUNDIDAD.
7. Vuelo V3 = V1. clone() ⟶ CLONE SUPERFICIAL
8. Vuelo V4 = V1. clone () ⟶ CLONE PROFUNDIDAD.
9. Piloto P3 = new Piloto (70, 2000);
10. Vuelo V3 = new Vuelo (2, P2);
11. V1.copy(V3) ⟶ COPY SUPERFICIAL.
12. V1.copy(V3) ⟶ COPY PROFUNDIDAD.



| | Superficial | Profundidad |
|---|---|---|
| EQUALS | A=B.obtenerA(); | A.copy(B.obtenerA()); |
| COPY | A==B; | A.equals(B.obtenerA()); |
| CLONE | New A(a, b, c); | New A(a, b, c.clone()); |

**Clientes y proveedores de servicios**

Algunas clases son clientes de los servicios provistos por otras clases proveedoras (una clase puede ser cliente y proveedora en simultaneo).

Una clase que usa los servicios provistos por otra clase, es cliente de la clase que provee dichos servicios. Una clase que tiene atributos de otra clase, es cliente de las clases a las que corresponden esos atributos. Estas ultimas son proveedoras de servicios.

↓

Clase cliente chequea las responsabilidades.

La clase cliente accede a la clase proveedora a través de si interfaz. La P.O.O. propone minimizar la interfaz a través de la cual se comunican una clase cliente y una proveedora, de modo que se **minimice** también el **impacto de los cambios**.

## Encapsulamiento y abstracción → La construcción de cada módulo de software puede realizarse sin conocer el sistema al cual va a integrase.

- Brinda independencia.
- Mecanismo fundamental que permite utilizar cada componente como una "caja negra".
  - → Saber qué hace sin saber cómo lo hace.

  Cada componente es un modulo que, al analizarlo individualmente, es posible hacer una abstracción de los detalles del contexto en el que va a ser utilizado para enfocarse en como lograr su propósito.

**Encapsulamiento en la P.O.O.**: cada objeto de software interactúa con otros objetos del entorno a través de su interfaz y oculta su E.I. y los detalles que describen como actúa cuando recibe un mensaje

→ Si la estructura del E.I. se modifica, el cambio no afecta a los otros miembros del entorno.

Para que un objeto de software oculte su E.I. la clase que define sus atributos y comportamiento debe esconder la implementación.

El encapsulamiento es el mecanismo que permite agrupar en una clase los atributos y el comportamiento que caracteriza a sus instancias y esconder la representación de los datos y los algoritmos que modelan el comportamiento, de modo que solo sean visibles dentro de la clase.

P.O.O. → encapsula la representación de los datos, de modo que no sean visibles desde el exterior de la clase. →

- Los modificadores de acceso permiten establecer la visibilidad de los miembros de una clase.
- Se declaran privados. → Las clases cliente no pueden acceder directamente al E.I.

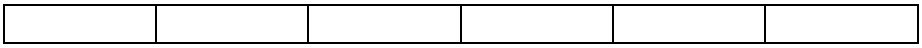
Se utilizan arreglos para modelar **estructuras lineales, homogéneas y con acceso directo.**

Cada elemento tiene un predecesor (excepto el primero) y un sucesor (excepto el ultimo).

Todos los elementos son del mismo tipo.

Cada una de sus componentes puede accederse a través de su posición.

Los arreglos arrancan en 0.



0 1 2 3 4 5

Debemos acceder a las posiciones desde N-1.

Recorrido exhaustivo ⟶ es necesario recorrer todos los componentes.

Recorrido no exhaustivo ⟶ no es necesario recorrer todos los componentes.

```
Public boolean alMenosN(Punto p, int n, int f){
        Int cont=0;
        For (int j=0;  j<cantColumnas() && con<N;  j++){
                If (mat[f][j]!=null && mat[f][j].inverso(p)){
                        Cont++;
                }
        }
        Return cont==n;
}
```

```
Public boolean ExactamenteN(Punto p, int n, int f){
        Int cont=0;
        For (int j=0;  j<cantColumnas() && con<=N;  j++){
                If (mat[f][j]!=null && mat[f][j].inverso(p)){
                        Cont++;
                }
        }
        Return cont==n;
}
```

```
Public boolean aLoSumoN(Punto p, int n, int f){
        Int cont=0;
        For (int j=0;  j<cantColumnas() && con<=N;  j++){
                If (mat[f][j]!=null && mat[f][j].inverso(p)){
                        Cont++;
                }
        }
        Return cont<=n;
}
```

En la implementación, una clase que establece atributos y servicios define un **tipo de dato** a partir del cual es posible crear instancias.

**Colecciones y Tablas** ⟶

Estructura con N elementos de un tipo base, cada uno de los cuales ocupa una posición que es significativa en la estructura.

NULOS Y LIGADOS SE INTERCALAN.

↓

Estructura con capacidad para mantener max. elem. de un tipo base.

LIGADOS AL PRINCIPIO.

Búsqueda binaria ——— colección de elementos esta ordenada de acuerdo a un atributo y es necesario decidir si un elemento en particular pertenece a la colección.

Partir la estructura en mitades, considerando que el elemento buscado puede ser:

- Igual al que está en el medio.
- Menor que el que está en el medio.
- Mayor que el que está en el medio.

Algoritmo:

```
DE: c.
Ini ◄——— 0.
Fin ◄——— cant posiciones ligadas -1.
Existe◄—— falso.
Mientras no existe y Ini<=fin
        Mitad◄—— (Ini+Fin)/2.
        Si Tmitad es igual a c
                Existe◄——verdadero.
        Sino
                Si Tmitad es menor a c
                        Ini◄——mitad+1.
                Sino
                        Fin◄—— mitad-1.
```

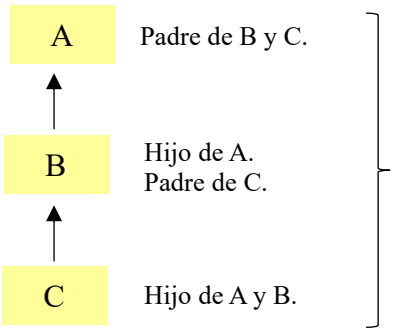
En un método basado en clases, las entidades se agrupan a partir de sus semejanzas y diferencias, establecidas en función de algún criterio. Una vez establecido el criterio de clasificación, es posible decidir si una entidad pertenece a una clase.

# Herencia y polimorfismo

→ Relación esUn. → Las instancias de una clase derivada son también instancias de las clases de las cuales hereda.

Herencia entre clases → herencia simple → clasificación jerárquica (en niveles).

→ Una clase especifica hereda las propiedades de las clases más generales en las cuales está incluida.




Heredan atributos y comportamientos de clases generales y agregan atributos y comportamientos específicos (también pueden redefinir comportamiento).

La **herencia** permite crear clasificaciones que modelan una relación de generalización-especialización entre clases.

El **polimorfismo** permite que las diferentes entidades de una misma clase puedan exhibir distintas formas de un mismo comportamiento.

→ En la P.O.O: mecanismo que permite que un mismo nombre pueda quedar ligado a objetos de diferentes clases y que objetos de distintas clases puedan recibir un mismo mensaje y cada uno actúe de acuerdo al comportamiento establecido por su clase.

La herencia favorece la reusabilidad y extensibilidad.

El polimorfismo favorece la productividad (mismo nombre puede asociarse a abstracciones diferentes dependiendo del contexto).

Utilizamos modificadores de acceso "protected" ——— permite que las clases derivadas accedan a los atributos de sus clases "padre" directamente.

Una clase derivada hereda de la clase base todos sus atributos y métodos, pero NO LOS CONSTRUCTORES. →

→ Pueden invocar a los constructores de las clases más generales.

Palabra reservada SUPER. ←

Polimorfismo → capacidad de asociar diferentes definiciones a un mismo nombre de modo que el contexto determine cual corresponde usar.

Mecanismo que permite que objetos de distintas clases puedan recibir un mismo mensaje y cada uno actúe de acuerdo a su comportamiento establecido por SU clase.

**Variable polimórfica:** puede quedar asociada a objetos de diferentes clases.

**Asignación polimórfica:** liga un objeto de una clase a una variable declarada de otra clase.

**Método polimórfico:** al menos uno de los parámetros formales es una variable polimórfica.

## TIPO ESTÁTICO Y DINÁMICO

- Tipo que aparece en la declaración.
- Clase a la que pertenece el objeto referenciado.

Redefinición de métodos

En la clase derivada se define un método con el mismo nombre, numero y tipo de parámetros que un método definido en la clase base.

La definición del método en la clase derivada REDEFINE el método de la clase base.

### LIGADURA DINÁMICA DE CÓDIGO

Vinculación en ejecución de un mensaje con un método.

Cuando un método definido en una clase queda redefinido en una clase derivada, el tipo dinámico de la variable determina que método va a ejecutarse en respuesta a un mensaje.

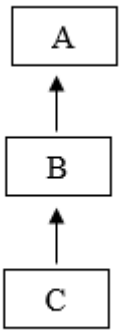
Chequeo de tipos

Establece restricciones sobre:

- Las asignaciones polimórficas.
- Los mensajes que un objeto puede recibir.

**En una asignación polimórfica la clase del objeto que aparece a la derecha del operador de asignación debe ser de la misma clase o de una clase descendente de la clase de la variable que aparece a la izquierda del operador.**

TIPO ESTÁTICO DE UNA VARIABLE DTEERMINA EL CONJUNTO DE TIPOS DINÁMICOS.




- A a1 = new A;
- A a2 = new B; **ERROR.**
- B b1 = new A;
- B b2 = new C; **ERROR.**
- C c1 = new A;
- C c2 = new B;

Compilador chequea tipo estático y se ejecuta el tipo dinámico.

El tipo estático de la variable determina los mensajes que un objeto puede recibir, pero el tipo dinámico determina la implementación específica del comportamiento que se uta en respuesta a los mensajes.

**CASTING** ⟶ relaja el control del compilador.

EJEMPLO

En la clase empleado

```
Public boolean equals (Emplado e){
        Boolean igual=false;
        If(this==e)
                Igual=true;
        Else
                If(this.getClass()==e.getClass())
                        Igual=legajo==e.obtenerLegajo() &&
                        sueldoBasico==e.obtenerSueldoBasico() &&
                        fechaIngreso.equals(e.obtenerFechaIngreso());
Return igual;
}
```

En la clase gerente

```
Public boolean equals (Emplado e){
        Boolean igual=false;
        If(this==e)
                Igual=true;
        Else
                If(this.getClass()==e.getClass()){
                        Gerente g = (Gerente) e;
                        Igual=super.equals(e) &&
                        productividad==g.obtenerProductividad();
                }
Return igual;
}
```

Los mecanismos de herencia, polimorfismo y vinculación dinámica favorecen la extensibilidad porque reducen el impacto de los cambios.

**Clase abstracta *** ⟶ puede incluir uno, varios, todos o ningún método abstracto.

No puede implementarse de manera general para todas las instancias de la clase ⟵

La clase abstracta define solo su signatura, sin bloque ejecutable.

Todos los objetos de la clase van a 0ofrecer una misma funcionalidad, pero la implementación correcta no puede generalizarse.

Modificador “abstract”.

# Genericidad

Consiste en abstraer lo que es común a un conjunto de entidades para definir un concepto que las incluya a todas.

Código no depende del tipo de los datos.

Favorece la extensibilidad y la reusabilidad. ⟶ Evita escribir el mismo código repetidamente, acelerando el proceso de desarrollo.

Reduce impacto en los cambios.

La abstracción de datos y el encapsulamiento permiten que una o mas clases usen los servicios provistos por otra clase considerando solo su comportamiento, sin tener en cuenta como la implementa.

La herencia permite aumentar el nivel de abstracción mediante un proceso de clasificación en niveles. Al mismo tiempo puede usarse para modelar genericidad y alternativamente la genericidad puede modelarse usando polimorfismo paramétrico.

Una clase genérica es aquella que encapsula a una estructura cuyo comportamiento es independiente del tipo de sus elementos.

# Algunos métodos relevantes

METODO DE LA BURBUJA

```
Public void ordenar(int []a){
        int aux;
        for(int i=0; i<a.length-1; i++){
                for(int j=0; j<a.length-i-1; j++){
                        if(a[j+1]<a[j]){
                                aux=a[j+1];
                                a[j+1]=a[j];
                                a[j]=aux;
                        }
                }
        }
}
```

METODO DOBLE PUNTERO PARA CONVERTIR LISTA EN COLECCIÓN

```
Public void agruparSurtidores(){
        int puntero1=0;
        int puntero2=0;
        surtidor aux;
        while(puntero1<cantPosiciones())
                if(tabla[puntero1]!=null){
                        aux=tabla[puntero1];
                        tabla[puntero2]=aux;
                        puntero2++;
                        puntero1++;
                else
                        puntero1++;
```