

Herencia → para crear clases q' heredan, extienden y modifican las de otras clases

Clase base - Clases derivadas

↳ cada puede derivar de una.

Clase derivada → hereda todo lo de la clase base salvo constructores y finalizadores

Sobreescripción de métodos → clases derivadas pueden invalidar métodos heredados para implementar los propios

→ normal (en la clase base)

→ overriding (en la derivada)

p. acceder a miembros de clase base, uso palabra clave base (para o super clase derivada)

uso

Para invocar a un método (u otro miembro) de la clase base
En el encabezado de un constructor para indicar el constructor de la clase base que se debe invocar.

Miembros privados → solo con variables donde se definen, no se pueden acceder desde clases derivadas

En general es una buena práctica NO acceder directamente a campos definidos en una superclase

Sin embargo, si se necesita extender el acceso de un miembro a las clases derivadas (pero no a las demás), se debe marcar como protegido (protected)

internal → se acceden solo desde dentro = ensamblado

protected → solo desde clase o sus derivadas

```
class Automotor
{
    public string Marca { get; }
    private int _modelo;
    public int Modelo
    {
        get => _modelo;
        protected set => _modelo = (value < 2005) ? 2005 : value;
    }
    ...
}
```

El modificador de acceso afecta sólo al descriptor set.
La propiedad Modelo podrá ser leída por cualquier clase pero sólo podrá modificarse desde la clase Automotor y sus derivadas.

- Las clases no pueden ser más accesibles que su clase base
- Ningún miembro puede ser más accesible que su tipo o el tipo de sus parámetros, índices o valor de retorno

Clase abstracta → definición común

→ No se puede instanciar

→ Métodos, inicializadores y eventos abstractos.

→ Clase abstracta puede tener clases no abstractas.

→ No puede haber métodos abstractos en clases concretas.

→ Tiene q' existir todo lo abstracto implementado en la concreta.

- Los finalizadores (también conocidos como destructores) se usan para "hacer limpieza" cuando el recolector de elementos no utilizados (*garbage collector*) libera memoria.
- El *garbage collector* comprueba periódicamente si hay objetos no referenciados por ninguna aplicación. En tal caso llama al finalizador (si existe) y libera la memoria que ocupaba el objeto.
- Se puede forzar la recolección con `GC.Collect()` pero en general debe evitarse por razones de rendimiento.

p. borro objeto, q' ya no se usa

polimorfismo

Al involucrarnos en un programa no se a q' se llama, se define en ejecución
obj. polimórfico por adoptar tipo \neq al declarado

```
object o;  
o = 1;  
o = "ABC";  
Automotor a;  
a = new Auto("Ford", 2000, TipoAuto.Deportivo);  
a = new Colectivo("Mercedes", 2010, 20);
```

o y a son objetos polimórficos.
En tiempo de ejecución va a
ocurrir polimorfismo (van a
adoptar distintas formas de tipos)

a. imprimir \rightarrow si a es auto se ejecuta imprimir de auto, sino colectivo.
se define en s.