Herencia

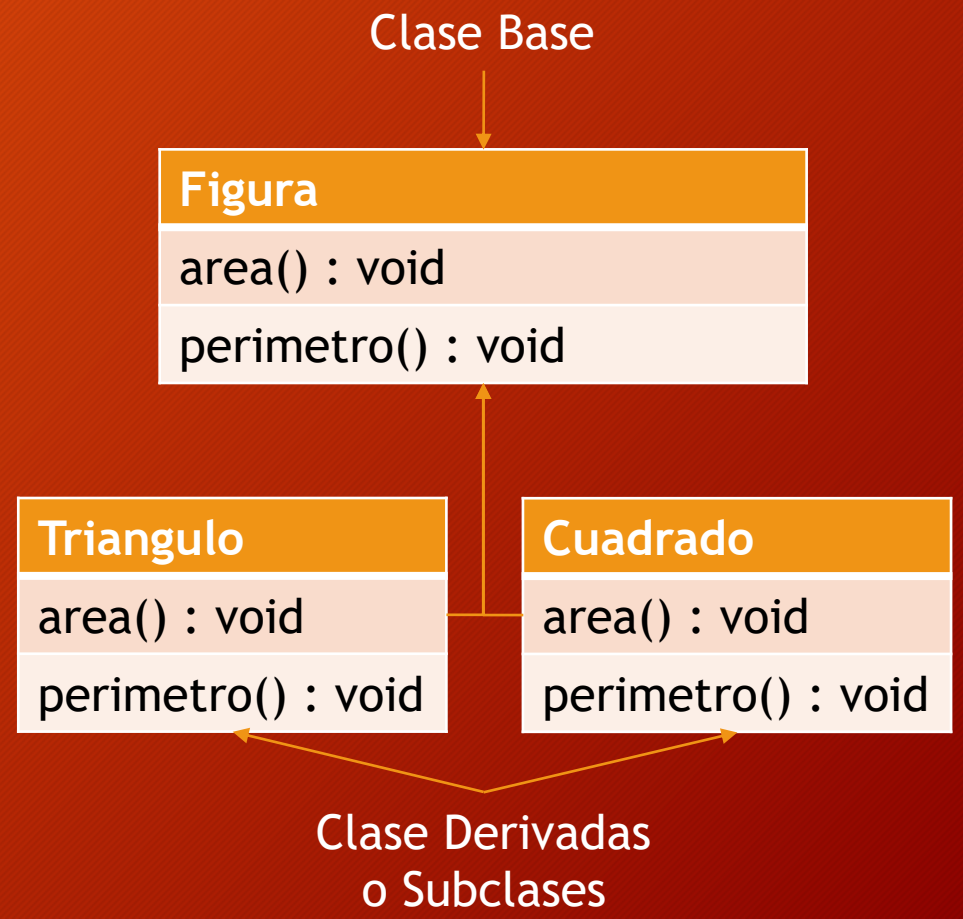
9

Programación II y Laboratorio de Computación II

Edición 2018

Concepto de Herencia

- Es una relación entre clases en la cual una clase comparte la estructura y comportamiento definido en otra clase.
- Cada clase que hereda de otra posee:
  - Los atributos de la clase base además de los propios.
  - Soporta todos o algunos de los métodos de la clase base.
- Una subclase hereda de una clase base.



Concepto de Herencia

- El propósito principal de la herencia es el de organizar mejor las clases que componen una determinada realidad, y poder agruparlas en función de atributos y comportamientos comunes. A la vez que cada una se especializa según sus particularidades.
- La herencia permite crear nuevas clases a partir de otras ya existentes (en lugar de crearlas partiendo de cero).
- La clase en la que está basada la nueva clase se la conoce como *clase base o* padre, mientras que la clase hija se conoce como *clase derivada*.

Tipos de Herencia

- **Herencia Simple:**
  - Una clase derivada puede heredar sólo de una clase base (los lenguajes .NET soportan este tipo de herencia)
- **Herencia Múltiple:**
  - Una clase derivada puede heredar de una o más clases base (C++ es un ejemplo de lenguaje que soporta este tipo de herencia).

Herencia C#

```
[modificadores] class NombreSubclase : NombreClaseBase
{ }
```

- **Modificadores**: son modificadores de visibilidad y/o de clase.
- **Class**: Le indica al compilador que el bloque de código es una declaración de clase.
- **Operador (:)** Le indica al compilador que es una subclase de la clase que precede al operador.
- **NombreClaseBase**: Es el nombre de la clase padre.

Herencia de la clase derivada

- Una clase derivada hereda todo de su clase base, **excepto los constructores.**
- Los miembros públicos de la clase base se convierten implícitamente en miembros públicos de la clase derivada.
- Sólo los miembros de la clase base tienen acceso a los miembros privados de esta clase, aunque la clase derivada también los hereda.
- Una clase derivada no puede ser más accesible que su clase base. Por ejemplo, no es posible derivar una clase pública de una clase privada.

Modificador Protected

- El significado del modificador de acceso **protected** depende de la relación entre la clase que tiene el modificador y la clase que intenta acceder a los miembros que usan el modificador.
- Para una clase derivada, la palabra reservada **protected** es equivalente a la palabra **public**.
- Entre dos clases que no tengan una relación base-derivada, por el contrario, los miembros protegidos de una clase se comportan como miembros **privados** para la otra clase.

Miembros Heredados

- Cuando una clase derivada hereda un miembro **protected**, ese miembro también es implícitamente un miembro protegido de la clase derivada.
- Esto significa que todas clases que deriven directa o indirectamente de la clase base pueden acceder a los miembros protegidos.
- Los métodos de una clase derivada sólo tienen acceso a sus propios miembros heredados con protección. No pueden acceder a los miembros protegidos de la clase base a través de referencias a ésta.

Ejemplo

```
class ClaseBase
{
    protected int edad;
}
class ClaseDerivada : ClaseBase
{
}
class Clase : ClaseDerivada
{
    string Compila()
    {
        return edad;
    }
}
```

```
class ClaseError : ClaseBase
{
    void Falla(ClaseBase t)
    {
        // No se puede acceder
        // protegido!
        return t.edad;
    }
}
```

Constructores: Base

- Para hacer una llamada a un constructor de la clase base desde un constructor de la clase derivada se usa la palabra reservada **base**:

```
[modificadores] Constructor( [Args]) : base([Args])
{ }
```

Ejemplo

```
class ClaseBase
{
    protected int edad;
    public ClaseBase(int edad)
    {
        this.edad = edad;
    }
}
class ClaseDerivada : ClaseBase
{
    public ClaseDerivada(int edad)
        : base(edad)
    { }
}
```

Constructores

- Si la clase derivada no hace una llamada explícita a un constructor de la clase base, el compilador de C# usará implícitamente un constructor de la forma :base().

```
public MiConstructor()
{}

//Equivale a...
public MiConstructor()
    : base()
{}
```

Constructores

- El comportamiento implícito anterior es válido en muchos casos porque:
  - Una clase sin clases base explícitas extiende implícitamente la clase **System.Object**, que contiene un constructor público sin parámetros (por defecto).
  - Si una clase no contiene ningún constructor, el compilador utilizará inmediatamente el constructor por “defecto”.
  - El compilador no creará un constructor por defecto si una clase tiene su propio constructor explícito.
  - No obstante, el compilador generará un mensaje de error si el constructor indicado no coincide con ningún constructor de la clase base.

Clases Selladas

- La mayor parte de las clases son autónomas y no están diseñadas para que otras clases deriven de ellas.
- Para que el programador pueda comunicar mejor sus intenciones al compilador y a otros programadores, C# permite declarar una clase como ***sealed*** (sellada).
- La derivación de una clase sellada no está permitida (no se puede heredar de ella).

Sealed

```
[modificadores] sealed class ClaseBase
{}
```

- Microsoft® .NET Framework contiene muchos ejemplos de clases selladas.
- Por ejemplo la clase **System.String**
- Esta clase está sellada y, por tanto, ninguna otra clase puede derivar de ella.

Resumen / Puntos Clave

- La herencia permite crear nuevas clases más especializadas a partir de otras ya existentes más generales.
- Las clases derivadas son versiones especializadas de las clases base. (Son del tipo de la clase base).
- En .NET sólo se admite HERENCIA SIMPLE (Sólo se puede heredar de una clase).
- La herencia es transitiva: Si C hereda de B, y B hereda de A, entonces C también hereda de A.

Resumen / Puntos Clave

- Se hereda TODO menos los constructores y finalizadores.
- Los miembros **private** no son visibles en las clases derivadas (PERO SÍ SE HEREDAN).
- Los miembros **protected** son accesibles desde todas las clases derivadas directa o indirectamente de la clase base, pero no desde otras clases que no hereden de ella.
- Una clase derivada no puede ser más accesible que su clase base.
- Si no se realiza una llamada explícita a un constructor de clase base [:base()], el compilador de C# proporciona automáticamente una llamada al constructor sin parámetros o predeterminado de la clase base.

Tipos de Clases y Herencia

Class Type		Can inherit from others	Can be inherited	Can be instantiated
normal	:	YES	YES	YES
abstract	:	YES	YES	NO
sealed	:	YES	NO	YES
static	:	NO	NO	NO