```
public class Ejemplo64
{

  public static void Main()
  {
    Animal[] misAnimales = new Animal[8];

    misAnimales[0] = new Perro();
    misAnimales[1] = new Gato();
    misAnimales[2] = new GatoSiames();

    for (byte i=3; i<7; i++)
      misAnimales[i] = new Perro();

    misAnimales[7] = new Animal();
  }

}
```

La salida de este programa sería:

```
Ha nacido un animal
Ha nacido un perro
Ha nacido un animal
Ha nacido un gato
Ha nacido un animal
Ha nacido un gato
Ha nacido un gato siamés
Ha nacido un animal
Ha nacido un perro
Ha nacido un animal
Ha nacido un perro
Ha nacido un animal
Ha nacido un perro
Ha nacido un animal
Ha nacido un perro
Ha nacido un animal
```

7.3. Funciones virtuales. La palabra "override"

En el ejemplo anterior hemos visto cómo crear un array de objetos, usando sólo la clase base, pero insertando realmente objetos de cada una de las clases derivadas que nos interesaba, y hemos visto que los constructores se llaman correctamente... pero con los métodos puede haber problemas.

Vamos a verlo con un ejemplo, que en vez de tener constructores va a tener un único método "Hablar", que se redefine en cada una de las clases hijas, y después comentaremos qué ocurre al ejecutarlo:

```
/*--------------------------*/
/*  Ejemplo en C# nº 65:    */
/*  ejemplo65.cs            */
```

```
/*                          */
/*  Ejemplo de clases       */
/*  Array de objetos de     */
/*    varias subclases con  */
/*    metodos               */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Animal
{

  public void Hablar()
  {
    Console.WriteLine("Estoy comunicándome...");
  }
}

// ------------------

public class Perro: Animal
{

  public new void Hablar()
  {
    Console.WriteLine("Guau!");
  }
}

// ------------------

public class Gato: Animal
{

  public new void Hablar()
  {
    Console.WriteLine("Miauuu");
  }
}

// ------------------

public class Ejemplo65
{

  public static void Main()
  {

    // Primero creamos un animal de cada tipo
    Perro miPerro = new Perro();
    Gato miGato = new Gato();
    Animal miAnimal = new Animal();

    miPerro.Hablar();
    miGato.Hablar();
    miAnimal.Hablar();
```

```
    // Linea en blanco, por legibilidad
    Console.WriteLine();

    // Ahora los creamos desde un array
    Animal[] misAnimales = new Animal[3];

    misAnimales[0] = new Perro();
    misAnimales[1] = new Gato();
    misAnimales[2] = new Animal();

    misAnimales[0].Hablar();
    misAnimales[1].Hablar();
    misAnimales[2].Hablar();
  }

}
```

La salida de este programa es:

```
Guau!
Miauuu
Estoy comunicándome...

Estoy comunicándome...
Estoy comunicándome...
Estoy comunicándome...
```

La primera parte era de esperar: si creamos un perro, debería decir "Guau", un gato debería decir "Miau" y un animal genérico debería comunicarse. Eso es lo que se consigue con este fragmento:

```
Perro miPerro = new Perro();
Gato miGato = new Gato();
Animal miAnimal = new Animal();

miPerro.Hablar();
miGato.Hablar();
miAnimal.Hablar();
```

En cambio, si creamos un array de animales, no se comporta correctamente, a pesar de que después digamos que el primer elemento del array es un perro:

```
Animal[] misAnimales = new Animal[3];

misAnimales[0] = new Perro();
misAnimales[1] = new Gato();
misAnimales[2] = new Animal();

misAnimales[0].Hablar();
misAnimales[1].Hablar();
misAnimales[2].Hablar();
```

Es decir, como la clase base es "Animal", el primer elemento hace lo que corresponde a un Animal genérico (decir "Estoy comunicándome"), a pesar de que hayamos dicho que se trata de un Perro.

Generalmente, no será esto lo que queramos. Sería interesante no necesitar crear un array de perros y otros de gatos, sino poder crear un array de animales, y que contuviera animales de distintos tipos.

Para conseguir este comportamiento, debemos indicar a nuestro compilador que el método "Hablar" que se usa en la clase Animal puede que sea redefinido por otras clases hijas, y que en ese caso debe prevalecer lo que indiquen las clases hijas.

La forma de hacerlo es declarando ese método "Hablar" como "virtual", y empleando en las clases hijas la palabra "override" en vez de "new", así:

```
/*--------------------------*/
/*  Ejemplo en C# nº 66:    */
/*  ejemplo66.cs            */
/*                          */
/*  Ejemplo de clases       */
/*  Array de objetos de     */
/*    varias subclases con  */
/*    metodos virtuales     */
/*                          */
/*  Introduccion a C#,      */
/*    Nacho Cabanes         */
/*--------------------------*/

using System;

public class Animal
{

  public virtual void Hablar()
  {
    Console.WriteLine("Estoy comunicándome...");
  }
}

// -----------------

public class Perro: Animal
{

  public override void Hablar()
  {
    Console.WriteLine("Guau!");
  }
}

// -----------------

public class Gato: Animal
{
```

```
  public override void Hablar()
  {
    Console.WriteLine("Miauuu");
  }
}

// ------------------

public class Ejemplo66
{

  public static void Main()
  {

    // Primero creamos un animal de cada tipo
    Perro miPerro = new Perro();
    Gato miGato = new Gato();
    Animal miAnimal = new Animal();

    miPerro.Hablar();
    miGato.Hablar();
    miAnimal.Hablar();

    // Linea en blanco, por legibilidad
    Console.WriteLine();

    // Ahora los creamos desde un array
    Animal[] misAnimales = new Animal[3];

    misAnimales[0] = new Perro();
    misAnimales[1] = new Gato();
    misAnimales[2] = new Animal();

    misAnimales[0].Hablar();
    misAnimales[1].Hablar();
    misAnimales[2].Hablar();
  }

}
```

El resultado de este programa ya sí es el que posiblemente deseábamos: tenemos un array de animales, pero cada uno "Habla" como corresponde a su especie:

```
Guau!
Miauuu
Estoy comunicándome...

Guau!
Miauuu
Estoy comunicándome...
```

Ejercicio propuesto:

- **(7.3.1)** Crea una versión ampliada del ejercicio 7.2.1, en la que no se cree un único objeto de cada clase, sino un array de tres objetos.