
Programación II

Agregando métodos

Los métodos se declaran dentro de la clase de la siguiente manera:

```
public string ConvertirAString(string separador)
        {
            return string.Format("La posicion es {0}{1}{2}", x, separador, y);
        }
```

Sobrecarga (overloading)

- Capacidad para tener dos o más métodos con el mismo nombre
- Los métodos se tienen que diferenciar por su firma (cantidad y tipo de parametros)

```
public string ConvertirAString()
{
    return ConvertirAString(",");
}

public string ConvertirAString(string separador)
{
    return string.Format("La posicion es {0}{1}{2}", x, separador, y);
}
```

Agregando constructores y destructores

- **Un constructor es un método que se ejecuta ni bien termina la creación de un objeto. Sirve para inicializar la instancia en cuestión.**
  - Tienen el mismo nombre que la clase en la que están definidos
  - No deben especificar un valor de retorno
  - Se pueden definir varios constructores sobrecargados. El constructor que no acepta parametros es el constructor por default.
- **Un destructor es un método que se ejecuta cuando el objeto es recolectado por el garbage collector.**
  - Tienen el mismo nombre que la clase en la que están definidos, precedido por un **~**
  - No debe especificar un valor de retorno ni tomar parámetros.

```
public Posicion()
{
    _x = _y = 0;
}

public Posicion(int x, int y)
{
    this.x = x;
    this.y = y;
}

~Posicion()
{
    //liberar cosas antes de la destruccion
}
```

Agregando miembros de clase

Todos los miembros mencionados hasta el momento son de instancia. Estos pertenecen a una instancia y deben ser accedidos por medio de ella.

Una clase puede contener miembros de clase, los cuales son accedidos sin la necesidad de crear una instancia.

Para definir los miembros de clase hacemos uso de la palabra clave **static**

Ejemplo

¿Qué pasa si tenemos 2 instancias de posición con las mismas coordenadas y las comparamos?

```
Sokoban.Posicion p1, p2;
p1 = new Sokoban.Posicion(0, 0);
p2 = new Sokoban.Posicion(0, 0);
Console.WriteLine("Los valores son {0}", p1 == p2 ? "iguales" : "distintos");
```

¿Cómo podriamos remediar esto?

- Usando una variante del patrón *singleton*
- Cambiar la forma de comparar las intancias

El patrón singleton asegura que para una clase haya solo una instancia. En este caso vamos a aplicar el patrón con la salvedad que hay varias instancias dependiendo del estado interno del objeto.

Implementando el patrón Singleton en C#

```
 1         private Posicion(int x, int y) { this.x = x; this.y = y; }
 2
 3         private static List<Posicion> posiciones = new List<Posicion>();
 4
 5         public static Posicion ObtenerPosicion(int x, int y)
 6         {
 7             foreach (Posicion p in posiciones)
 8             {
 9                 if (p.x == x && p.y == y) return p;
10             }
11             Posicion res = new Posicion(x, y);
12             posiciones.Add(res);
13             return res;
14         }
15
16
```

1. Hacemos inaccesible el constructor desde fuera de la instancia
3. Agregamos un campo de clase para almacenar las instancias que se van creando
5. Agregamos un método para obtener una instancia, el cual revisa si la instancia buscada ya fue creada para no volverla a crear.

```
statics.Posicion p1, p2;
p1 = statics.Posicion.ObtenerPosicion(0, 0);
p2 = statics.Posicion.ObtenerPosicion(0, 0);
Console.WriteLine("Los valores son {0}", p1 == p2 ? "iguales" : "distintos");
```