Programación orientada a objetos. Segundo ejercicio.

4 de abril de 2008.

La empresa que les pidió que programaran la simulación de los autos de carrera quedo tan impresionada con los resultados extremadamente realistas y precisos que generaron sus programas que los han llamado nuevamente.

Para esta ocasión, simplemente les piden que programen un simulador de mundos, donde ellos puedan evaluar el resultado de políticas económicas, el crecimiento de microorganismos, el vuelo de las gaviotas, y la expansión de epidemias, entre otros.

Claro que para ésto es necesario empezar con algunos componentes básicos sobre los cuales construir las simulaciones.

Parte 1

Como primer paso deberás programar un mundo de dos dimensiones.

El mundo se corresponderá con una grilla, donde en cada celda de la grilla puede haber un único elemento.

Para que puedas interactuar con tu mundo, este debe proveer una forma de poner elementos en una celda, quitar elementos de una celda, y saber en que celda está un cierto elemento

En particular, un mundo debe tener *por lo menos* el siguiente tipo:

```
public interface IWorld
{
    /// <summary>
    /// Retorna la cosa contenida en la posicion especificada.
    /// </summary>
    /// <param name="x">la coordenada x</param>
    /// <param name="y">la coordenada y</param>
    /// <returns>la cosa, o <code>null</code> si no hay nada en esa
    /// posicion</returns>
    IThing GetThing(Int32 x, Int32 y);
    /// <summary>
    /// Retorna si el mundo contiene la cosa.
    /// </summary>
    /// <param name="thing">la cosa</param>
    /// <returns><code>true</code> si lo contiene, <code>false</code> en
    /// otro caso.</returns>
    Boolean Contains(IThing thing);
    /// <summary>
    /// Retorna si el mundo contiene la cosa en la posicion especificada.
    /// </summary>
    /// <param name="thing">la cosa</param>
    /// <param name="x">la coordenada x de la posicion</param>
    /// <param name="y">la coordenada y de la posicion</param>
    /// <returns><code>true</code> si lo contiene, <code>false</code> en
    /// otro caso.</returns>
    Boolean Contains(IThing thing, Int32 x, Int32 y);
    /// <summary>
    /// Retorna si el mundo contiene la posicion especificada.
    /// </summary>
    /// <param name="x">la coordenada x</param>
    /// <param name="y">la coordenada y</param>
    /// <returns><code>true</code> si la coordenada esta contenida en el
    /// mundo, <code>false</code> en otro caso.</returns>
    Boolean IsInWorld(Int32 x, Int32 y);
    /// <summary>
    /// Guarda la cosa en la posicion x e y del mundo.
```

```
        /// </summary>
        /// <param name="thing">la cosa a guardar</param>
        /// <param name="x">la coordenada x</param>
        /// <param name="y">la coordenada y</param>
        void PutThing(IThing thing, Int32 x, Int32 y);
        /// <summary>
        /// Quita la cosa del mundo.
        /// </summary>
        /// <param name="thing">la cosa</param>
        void RemoveThing(IThing thing);
        /// <summary>
        /// Retorna la posicion de la cosa en el mundo.
        /// </summary>
        /// <param name="thing">la cosa</param>
        /// <returns>la posicion de la cosa en el mundo, o <code>null</code>
        /// si no esta contenida en el</returns>
        IPoint GetPosition(IThing thing);
    }
```

Donde IThing es simplemente una interfaz de marca que no tiene operaciones, e IPoint representa la posición de una celda en el mundo en base a sus coordenadas X e Y.

Este primer mundo que programes debe ser un cuadrado o un rectángulo, pero en algún momento no descartes que debas programar mundos con otras formas.

Si necesitas agregar métodos adicionales a tu clase, o incluso prefieres cambiar los propuestos, adelante, pero debes proveer por lo menos la misma funcionalidad que provee el tipo propuesto.

Recuerda programar una clase de prueba para la clase que implementes. Para hacerlo, necesitaras tener cosas para poner en tu mundo. No debería serte muy difícil tenerlas...

Parte 2

Para poder seguir una simulación, es importante tener alguna forma de ver gráficamente como suceden las cosas.

Para que no pierdas tiempo estudiando como dibujar por pantalla, te proveemos una biblioteca de clases que te permitirá dibujar figuras en la pantalla de tu monitor.

La biblioteca se compone de cinco tipos de objetos:

- IPoint: son un punto en coordenadas x,y.
- IShape: son figuras. Las figuras se componen de un conjunto de puntos.
- IPainter: son objetos capaces de pintar figuras.
- IBoard: son pizarra donde un pintor puede dibujar. La pizarra es simplemente una matriz de colores.
- IGui: son controles gráficos que puedes usar para mostrar una pizarra.

Además, se incluyen algunas clases concretas: una que representa un color, y otras que implementan estas interfaces.

Para poder pintar algo, debes usar las clases provistas adecuadamente, para lo que debes leer cuidadosamente su documentación. Pero podemos darte como pista que para hacerlo debes crear una pizarra, pedirle al pintor que pinte una figura sobre ella, y luego pedirle al control gráfico que muestre la pizarra pintada.

Para la parte dos, deberás lograr, de alguna forma, que el mundo se muestre por pantalla. Debes mostrar en la pantalla el contorno de tu mundo y pintar cada celda ocupada.

Para que tu programa pueda hacer uso de la interfaz gráfica, debes incluir como referencia de tu proyecto la biblioteca System.Windows.Forms, agregarla usando using a tu clase Program y al final de código del método void Main(), llamar a Application.Run().

El código final de la clase Program lucirá similar a:

```
using System;
using System.Windows.Forms;
using PainterLib;
namespace UcuLife {
        static class Program {
                [STAThread]
                static void Main() {
                        IWorld world = ....;
                        IBoard board = ....;
                        IPainter painter = ....;
                        IGui gui = ....;
                        // Hacer que el pintor pinte el mundo sobre la pizarra
                        .....
                        gui.Open();
                        // Le pedimos al IGui que muestre la pizarra pintada
                        gui.Redraw(board);
                        Application.Run();
                }
        }
}
```

Ten en cuenta que las clases provistas son únicamente para reforzar conceptos de POO. ¡Son muy malas respecto a performance! ¡Ni se te ocurra pensar en usarlas en el curso de programación gráfica o algo así!.

Parte 3

Por último, debes poder cargar el mundo y su contenido a partir de un archivo que contiene *tags*.

Para ésto, se define el siguiente formato para los tags del archivo:

```
<square-world id="mundo1" size="50"/>
```

Define un mundo cuadrado de 50 celdas de lado con nombre "mundo1".

```
<rectangular-world width="50" height="20"/>
```

Define un mundo rectangular de 50 celdas de ancho y 20 de alto con nombre "mundo2".

```
<block world="mundo1" x="10" y="20"/>
```

Define un bloque que debe colocarse en el mundo1 en la posicion (10, 20).

```
<block world="mundo2" x="15" y="25"/>
```

Define un bloque que debe colocarse en el mundo2 en la posicion (15, 25).

Si recibes un archivo con ese contenido, debes cargar los mundos, poner los bloques en el mundo, y desplegar ambos por pantalla. Los bloques son cosas que ocupan una celda y no hacen absolutamente nada más.

Al final de estas tres partes deberías tener el esqueleto mínimo para hacer tu simulación. Pero claro, no hay nada dinámico hasta el momento ni nada útil que simular... todo a su tiempo, el próximo ejercicio agregará algo de movimiento a tus fotos.

Importante

Casos de prueba `nunit` y documentación ndoc

Recuerda que un programa *debe* incluir casos de prueba y documentación ndoc. Los casos de prueba deben asegurarse de que cada método de cada clase que programaste funcione como es debido. Intenta utilizar ésto como una herramienta para trabajar menos y más seguro, y no como un requisito "burocrático".

Reglas de colaboración

Los ejercicios son individuales. Puedes diseñar una solución en conjunto con otros compañeros y aprovechar comentarios o correcciones de los profesores, pero debes entregar un código que tú comprendiste, escribiste, compilaste, ejecutaste y probaste. Si entregas una clase que a nuestro criterio es idéntica a la de un compañero, puedes estar en problemas.

Entregas tardías

No se aceptarán entregas fuera de fecha. Los trabajos se entregan para que sigas constantemente el curso y no con objetivo de evaluarte, pese a que influyen en tu evaluación final en caso de que estés en riesgo de perder el curso. Un trabajo a las apuradas no tiene valor en este contexto.

Entregas por correo electrónico

No se aceptarán entregas por correo electrónico excepto que Web Asignatura no esté disponible seis horas antes a la fecha final de entrega. La entrega por correo electrónico debe enviarse a todos los profesores y pedirles confirmación de entrega. Es tu responsabilidad asegurarte de que el trabajo haya sido recibido.