
Programación II

La naturaleza de las clases

- Una clase representa solo la abstracción de un objeto, o sea, las características comunes a todos los objetos de esa clase
- Una clase es un conjunto de objetos que comparten la estructura, comportamiento y semántica

Interfaz e Implementación

- La programación es una cuestión de **contratos**
- Una clase sirve de vínculo entre una abstracción y todos sus clientes
- **Una clase se puede dividir en la vista externa e interna**
  - La vista externa es la interfaz, una declaración de todas las operaciones y propiedades disponibles en una clase
  - La vista interna es la implementación, contiene todos los detalles de implementación para llevar a cabo las operaciones definidas en la interfaz

Introducción a C# y .Net

Tipos de datos

- **Por valor:** contienen directamente los datos. Se almacenan en el stack. Ej: System.Int32, System.Boolean...
- **Por referencia:** contienen punteros a los datos almacenados en el heap. Ej: System.String, clases, colecciones...

Variables

En C# se definen de la siguiente manera:

```
int Edad;
string Nombre;
bool Casado;
```

Instrucciones condicionales

if

```
if (Nombre == "Bob")
{
    //Procesar Bob
}
else if (Nombre == "Pedro")
{
    //Procesar Pedro
}
else
{
    //Procesar el resto
}
```

switch

```
switch (Nombre)
{
    case "Bob":
        //Procesar Bob
        break;
    case "Pedro":
        //Procesar Pedro
        break;
    default:
        //Procesar el resto
        break;
}
```

Bucles

For

```
string[] Names = null;
//for
for (int i = 0; i < Names.Length; i++)
{
    Console.WriteLine(Names[i]);
}
//foreach
foreach (string name in Names)
{
    Console.WriteLine(name);
}
```

While

```
bool Condicion = true;
//while
while (Condicion)
{
    Condicion = ActualizarValor(1);
}
//do..while
do
{
    Condicion = ActualizarValor(2);
} while (Condicion);
```

Funciones

```
public bool ActualizarValor(int numero)
{
    return numero > 18;
}
```

Manejo de excepciones

```
public void BorrarArchivo(string Archivo)
{
    try
    {
        System.IO.File.Delete(Archivo);
    }
    catch (FileNotFoundException fnfe)
    { Console.WriteLine("No se encontro el archivo"); }
    catch (Exception e)
    {
        Console.WriteLine("Ocurrio una excepcion no controlada");
        throw e;
    }
    finally
    { Console.WriteLine("Esto ocurre si o si"); }
}
```

Tipos genéricos

```
//en el array puedo cargar cualquier cosa...
ArrayList nombres = new ArrayList();
nombres.Add("Pedro");
nombres.Add(23); // puede fallar en runtime si supongo que hay solo strings

//En este caso vamos a tener un error en tiempo de compilación
List<string> _nombres = new List<string>();
_nombres.Add("Pedro");
//_nombres.Add(23);
```

En el namespace **System.Collections.Generic** tengo disponibles:

- Lista
- Diccionario
- Stack, Queue...

Nullables

- Permiten representar la ausencia de valor en tipos por valor

```
public int? BuscarLibro(string libro) { return new Nullable<int>(); }
//....

public void otra_funcion()
{
    int? CodigoLibro = BuscarLibro("C# for dummies");
    if (CodigoLibro.HasValue)
    {
        Console.WriteLine("El código es {0}", CodigoLibro.Value);
    }
    else
    {
        throw new Exception("Libro no encontrado");
    }
}
```

Clases en C#

Algunos conceptos

- **Campos:** representan datos en un objeto
- **Propiedades:** proveen acceso a elementos de un objeto
- **Metodos:** definen las acciones que un objeto puede hacer
- **Eventos:** acciones que se pueden responder o manejar en el código.
- **Overloading:** capacidad de tener varios métodos con el mismo nombre.

El modelo de objetos

Para ver los conceptos de orientación a objetos vamos a implementar un juego llamado **Sokoban**

Sokoban es un clásico rompecabezas inventado en Japón, normalmente implementado como un videojuego.

Sokoban significa *encargado de almacén* en japonés. El objetivo del juego es empujar las cajas (o las bolas) hasta su lugar correcto dentro de un reducido almacén, con el número mínimo de empujes y de pasos. Las cajas se pueden empujar solamente, y no tirar de ellas, y sólo se puede empujar una caja a la vez.

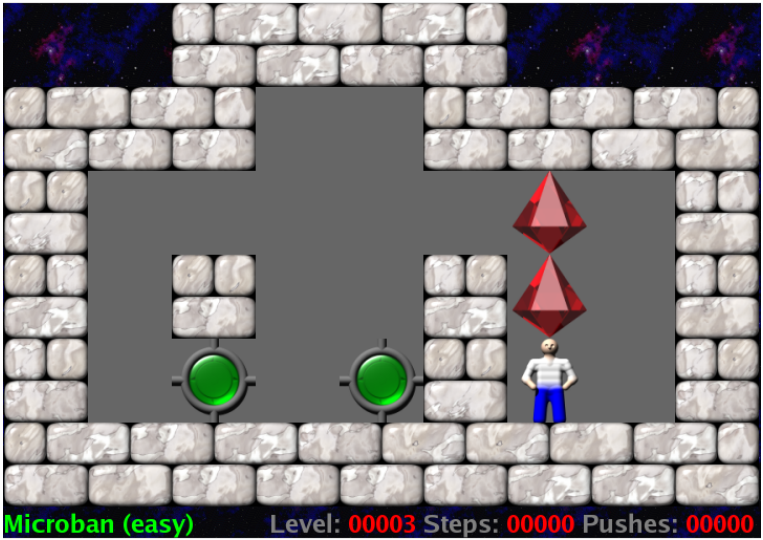
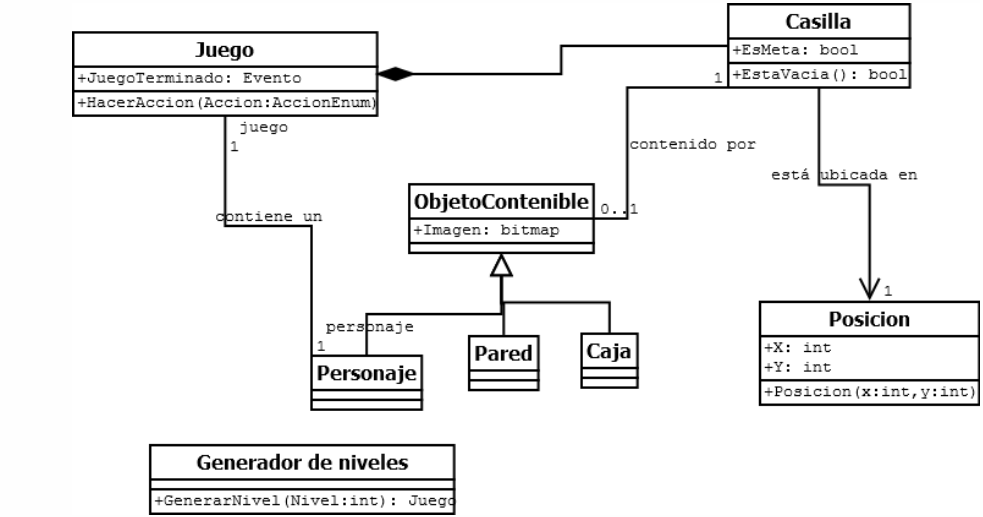

Microban (easy)
Level: 00003 Steps: 00000 Pushes: 00000

Diagrama de clases inicial



Definición de una clase. Clase Posición

Para definir una clase utilizamos la palabra clave **class**

```
1 using System;
2 namespace Sokoban
3 {
4     public class Posicion {    }
5 }
6
```

1. Importación de espacios de nombres
2. El espacio de nombre de la clase
4. La definición de la clase en si. Se utiliza la palabra clave **class** y opcionalmente un **modificador de acceso**.

Modificadores de acceso.

Permiten especificar desde donde son accesibles los miembros de una clase, o la clase misma.

Modificador	Nivel de acceso	Válido para
public	Acceso sin restricciones.	Clases y miembros
private	Accesible solo desde la clase	Miembros
internal	Accesible desde el assembly actual.	Clases y miembros
protected	Accesible desde la clase donde se declara o sus subclases	Miembros
protected internal	Accesible desde el assembly actual y desde las subclases	Miembros

Agregado de campos y propiedades.

- Los campos son variables a nivel de instancia.
- Las propiedades nos permiten encapsular el estado. Pueden ser:
  - **Solo lectura:** proveemos solo el método **get**
  - **Solo escritura:** proveemos solo el método **set**
  - **Lectura y escritura:** proveemos ambos métodos

```
private string _nombre;
public string nombre
{
    get { return _nombre; }   //método de lectura
    set { _nombre = value; }    //método de escritura
}
```

La clase posición con las propiedades y campos agregados:

```
public class Posicion
{
    private Int32 _x;
    public Int32 x
    {
        get { return _x; }
        set { _x = value; }
    }
    private Int32 _y;
    public Int32 y
    {
        get { return _y; }
        set { _y = value; }
    }
```

Creación de instancias

Para crear una instancia definimos una variable de ese tipo y utilizamos la palabra **new**

```
Posicion p1 = new Posicion();
p1.x = 10;
p1.y = -1;
Console.WriteLine("{0}  {1}  {2}",p1, p1.x, p1.y);
Console.ReadLine();
```

Ejercicio.

1. Modifique la clase posición para que las coordenadas se inicialicen en cero.
2. Agregue un control para que no se puedan guardar valores negativos en las coordenadas.
3. Cree una instancia de posición para validar los puntos anteriores.

Bibliografía y enlaces útiles.

- Booch, Grady et all: Object Oriented Analysis and Design with applications - Third Edition
- Enlace a Sokoban en Wikipedia: http://es.wikipedia.org/wiki/Sokoban