Programación orientada a objetos. Primer parcial 2º semestre 2006.

Viernes 15 de septiembre de 2006.

Al igual que las demás instancias de evaluación este examen será calificado de acuerdo a la escala de letras vigente: D, R, B, MB, BMB, S o D, R, B, MB, S según la generación de cada alumno. Por eso las preguntas no tienen puntos.

Pongan el nombre sólo en la primera hoja. Las respuestas deben aparecer en el orden en que están formuladas las preguntas. Escriban sólo del anverso de la hoja y eviten doblarlas. Gracias y mucha suerte.

Sea el siguiente programa:

```
using System;

interface InterfazUno
{
    String Propiedad { get; set; }
    String Metodo(String parámetro);
}

class ClaseUno
{
    private String propiedad;
    public String Propiedad { get { return propiedad; }
        set { propiedad = value; } }

    Int32 Metodo(Int32 parámetro)
    {
        /* Código no mostrado */
    }
}

class ClaseDos : InterfazUno
{
    private String propiedad;
    public String Propiedad { get { return propiedad; }
        set { propiedad = value; } }

    public String Metodo(String parámetro)
    {
        /* Código no mostrado */
    }
}

class Program
{
    static void Main(string[] args)
    {
        /*1*/ ClaseUno a = new ClaseUno();
        /*2*/ a.Propiedad = "a";
        /*3*/ ClaseUno b = new ClaseUno();
        /*4*/ b.Propiedad = "a";
        /*5*/ ClaseDos c = new ClaseDos();
        /*6*/ c.Propiedad = "a";
```

```
        /*7*/ ClaseUno d = a;
        /*8*/ ClaseDos e = c;
    }
}
```

1. ¿Cuándo dos objetos son iguales?
2. En el código anterior ¿qué variables contienen o referencian objetos iguales? Pongan el nombre de las variables y la línea donde son creados los objetos.
3. En el código anterior ¿qué variables contienen o referencian al mismo objeto? Pongan el nombre de las variables y la línea donde son creados los objetos.

Sean las clases e interfaces del programa anterior pero el siguiente programa:

```
class Program
{
    static void Main(string[] args)
    {
        /*1*/ ClaseUno a = new ClaseUno();
        /*2*/ InterfazUno b = a;
        /*3*/ ClaseDos c = b;
        /*4*/ ClaseDos d = new ClaseDos();
        /*5*/ InterfazUno e = d;
        /*6*/ ClaseUno f = e;
        /*7*/ InterfazUno g = new ClaseDos();
    }
}
```

4. Indiquen las líneas de código incorrectas y justifiquen la decisión.
5. ¿Cuáles tipos tiene cada uno de los objetos contenidos o referenciados en las variables definidas en las líneas correctas? Indiquen los nombres de los tipos en cada caso. Consideren solo los tipos definidos en esta pregunta.
6. ¿Qué mensajes pueden ser enviados a los objetos contenidos o referenciados en las variables `a` y `e` asumiendo que esas líneas fueran correctas? ¿Cómo lo saben? Escriban los mensajes. Consideren solo los tipos definidos en esta pregunta.
7. ¿Los objetos contenidos o referenciados en qué variables tienen el mismo tipo? Para cada tipo en esta situación indiquen las variables. Consideren solo los tipos definidos en esta pregunta.
8. ¿Los objetos contenidos o referenciados en qué variables tienen más de un tipo? Para cada variable en esta situación indiquen los tipos. Consideren solo los tipos definidos en esta pregunta.

El siguiente código compila, ejecuta y funciona bien. Los métodos y clases que no aparecen aquí están implementados correctamente pero se omiten para preservar el poco espacio disponible.

```
class LectorTags
{
    public LectorTags()
    {
        String archivo = "miarchivo.html";
        String tags = LeerTagsDeArchivo(archivo);

        ListaTag listaTags = ObtenerListaTag(tags);
        if (listaTags == null)
        {
            Console.WriteLine("Error en el programa");
```

```
            return;
        }
        BuscarTagsRepetidos buscador = new BuscarTagsRepetidos();
        if (buscador.HayTagsRepetidos(listaTags, archivo))
        {
            Console.WriteLine("Error hay tags repetidos");
            return;
        }
        foreach(Tags tag in listaTags) {
            Console.WriteLine("Encontre tag:" + tag);
        }
    }
    .....
}
```

9. Critiquen en forma breve tres puntos concretos del código anterior. Elijan aquellos puntos que consideren más graves.

Sean las siguientes interfaces. Las interfaces están definidas en `System` y están copiadas aquí con la documentación del .NET Framework para vuestra comodidad.

```
namespace System
{
    /// <summary>
    /// Admite la clonación, que crea una nueva instancia de una clase
    /// con el mismo estado que una instancia existente.
    /// </summary>
    public interface ICloneable
    {
        /// <summary>
        /// Crea un nuevo objeto copiando el estado de la instancia
        /// actual.
        /// </summary>
        /// <returns>Nuevo objeto que es una copia de esta
        /// instancia.</returns>
        object Clone();
    }
}
```

Un `Stack` es una colección de objetos que sólo me permite acceder al último objeto agregado:

```
using System;
using System.Collections;
using System.Diagnostics;

class Stack
{
    private ArrayList elements = new ArrayList();

    private Int32 limit;
    public Int32 Limit { get { return limit;  } }

    /// <summary>
    /// Crea un nuevo Stack con el límite indicado.
    /// </summary>
    public Stack(Int32 limit)
    {
        this.limit = limit;
    }
```

```
    /// <summary>
    /// Agrega un objeto en la cima del Stack.
    /// </summary>
    public void Push(Object element)
    {
        …
    }

    /// <summary>
    /// Remueve y retorna el objeto en la cima del Stack.
    /// </summary>
    public Object Pop()
    {
        …
    }

    /// <summary>
    /// Retorna el objeto en la cima del Stack sin removerlo.
    /// </summary>
    public Object Peek()
    {
        …
    }
}

class Program
{
    static void Main()
    {
        Stack b = new Stack(2);
        String s = "abc";
        b.Push(s);
        Console.WriteLine(b.Count);
        b.Push(b.Pop());
        Console.WriteLine(b.Count);
        b.Push("xyz");
        Console.WriteLine(b.Count);
    }
}
```

10. Hagan las modificaciones necesarias en la clase `Stack` para que las instancias de esta clase tengan el tipo `IClonable`. Programen también los métodos que están incompletos.

11. Agreguen usando `Debug.Assert(Boolean)` o `requires` y `ensures` lo que haga falta para especificar las siguientes afirmaciones:

    - La cantidad de elementos en el `Stack` es siempre menor que el límite.
    - La cantidad de elementos en el `Stack` es siempre mayor o igual que cero.
    - Al agregar un elemento, ese elemento es el último agregado.
    - Al agregar un elemento, la cantidad de elementos aumenta en uno.
    - Para remover un elemento, el `Stack` no puede estar vacío.
    - Al remover un elemento, el `Stack` ya no contiene ese elemento.
    - Al remover un elemento, la cantidad de elementos en el `Stack` disminuye en uno.

    No es necesario que escriban el programa nuevamente, mientras quede bien claro dónde harían los cambios.