Programación orientada a objetos. Primer parcial 2006.

Viernes 28 de abril 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 punto: todas valen lo mismos

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;

namespace PreguntaUno
{
    class ClaseUno
    {
        public String VariableInstanciaUno;
        public Int32 VariableInstanciaDos;
    }

    class ClaseDos
    {
        public String VariableInstanciaUno;
        public Int32 VariableInstanciaDos;
    }

    class Program
    {
        static void Main()
        {
            /*1*/ ClaseUno a = new ClaseUno();
            /*2*/ a.VariableInstanciaUno = "a";
            /*3*/ a.VariableInstanciaDos = 1;
            /*4*/ClaseUno b = new ClaseUno();
            /*5*/ b.VariableInstanciaUno = "a";
            /*6*/ b.VariableInstanciaDos = 1;
            /*7*/ ClaseDos c = new ClaseDos();
            /*8*/ c.VariableInstanciaUno = "a";
            /*9*/ c.VariableInstanciaDos = 1;
            /*10*/ ClaseUno d = a;
            /*11*/ ClaseDos e = c;
        }
}
```

1.1. Identifiquen que objetos son iguales poniendo el nombre de las variables que los referencian y en qué línea son creados.

1.2. Identifiquen que objetos son el mismo poniendo el nombre de las variable que los referencian y en qué línea son creados.

Sea el siguiente programa:

```
using System;

namespace PreguntaDos
{
    interface InterfazUno
    {
        void ComportamientoUno();
    }

    interface InterfazDos
    {
        void ComportamientoDos();
    }

    interface InterfazTres
    {
        void ComportamientoTres();
    }

    class UnaClase : InterfazUno, InterfazDos
    {
        public void ComportamientoUno()
        {
        }

        public void ComportamientoDos()
        {
        }

        public void ComportamientoTres()
        {
        }
    }

    class OtraClase : InterfazUno
    {
        public void ComportamientoUno()
        {
        }
    }

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

2.1 Indiquen las líneas de código incorrectas.

2.2 ¿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.

2.3 ¿Qué mensajes pueden ser enviados a los objetos contenidos o referenciados en las variables `a` y `e`? ¿Cómo lo saben? Escriban los mensajes. Consideren solo los tipos definidos en esta pregunta.

2.4 ¿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.

2.5 ¿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.

Sea el siguiente programa:

```
using System;

namespace PreguntaTres
{
    class ClaseUno
    {
        public String VariableInstancia;
        public void MetodoUno()
        {
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ClaseUno a = new ClaseUno();
            a.VariableInstancia = "a";
            a.MetodoUno();
        }
    }
}
```

3.1 ¿Cuál es el estado del objeto contenido o referenciado por la variable `a`?

3.2 ¿Y el comportamiento?

Sea el siguiente programa:

```
using System;

namespace PreguntaCuatro
{
    class ClaseUno
    {
        public String VariableInstancia;
        public void Metodo(ClaseDos dos)
        {
            Console.WriteLine(dos.VariableInstancia);
        }
    }

    class ClaseDos
    {
        public String VariableInstancia;
        public void Metodo(ClaseUno uno)
        {
            Console.WriteLine(uno.VariableInstancia);
        }
    }
```

```
    class Program
    {
        static void Main()
        {
            ClaseUno uno = new ClaseUno();
            uno.VariableInstancia = "Uno";
            ClaseDos dos = new ClaseDos();
            dos.VariableInstancia = "Dos";
            uno.Metodo(dos);
            dos.Metodo(uno);
        }
    }
}
```

4.1 El programa compila y funciona. ¿Qué sale impreso en la consola al ejecutar este programa?

4.2 ¿Cuáles son las responsabilidades de los objetos de la clase `ClaseUno`? ¿Y las de los objetos de la clase `ClaseDos`?

4.3 ¿Qué comentarios tendrían respecto a la distribución de responsabilidades?

Sean las siguientes interfaces. Las interfaces están definidas en `System.Collections` y están copiadas aquí con la documentación del .NET Framework para vuestra comodidad. La interfaz `IEnumerable` ya la conocen.

```
/// <summary>
/// Exposes the enumerator, which supports a simple iteration over a
/// non-generic collection.
/// </summary>
interface IEnumerable
{
    /// <summary>
    /// Returns an enumerator that iterates through a collection.
    /// </summary>
    /// <returns>An IEnumerator object that can be used to iterate
    /// through the collection.</returns>
    IEnumerator GetEnumerator();
}

/// <summary>
/// Defines size, enumerators, and synchronization methods for all
/// nongeneric collections.
/// </summary>
interface ICollection : IEnumerable
{
    /// <summary>
    /// Copies the elements of the ICollection to an Array, starting
    /// at a particular Array index.
    /// </summary>
    /// <param name="array">The one-dimensional Array that is the
    /// destination of the elements copied from ICollection. The Array
    /// must have zero-based indexing. </param>
    /// <param name="index">The zero-based index in array at which
    /// copying begins.</param>
    void CopyTo(Array array, int index);

    /// <summary>
    /// Gets the number of elements contained in the ICollection.
    /// </summary>
    Int32 Count { get; }
```

```
    /// <summary>
    /// Gets a value indicating whether access to the ICollection is
    /// synchronized (thread safe).
    /// </summary>
    Boolean IsSynchronized { get; }

    /// <summary>
    /// Gets an object that can be used to synchronize access to the
    /// ICollection.
    /// </summary>
    Object SyncRoot { get; }
}
```

Un `Bag` es una colección de objetos que no tiene repetidos: si agrego el mismo objeto por segunda vez no ocurre nada. Sea el siguiente programa:

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

namespace PreguntaCinco
{
    class Bag
    {
        private ArrayList elements = new ArrayList();

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

        public Bag(Int32 limit)
        {
            this.limit = limit;
        }

        public Boolean Contains(Object element)
        {
            return (elements.IndexOf(element) != -1);
        }

        public void Add(Object element)
        {
            …
        }

        public void Remove(Object element)
        {
            …
        }
    }
```

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

5.1 Sabiendo que las colecciones son objetos del tipo ICollection hagan las modificaciones necesarias en la clase `Bag` para que las instancias de esta clase tengan el tipo `ICollection`. Tengan en cuenta que las instancias de la clase `ArrayList` también son del tipo `ICollection`.

5.2 Agreguen usando `Debug.Assert(Boolean)` lo que haga falta para especificar las siguientes afirmaciones:

- La cantidad de elementos en el `Bag` es siempre menor que el límite.
- La cantidad de elementos en el `Bag` es siempre mayor o igual que cero.
- Al agregar un elemento, si ese elemento ya fue agregado antes, no cambia la cantidad de elementos en el `Bag`; de lo contrario, la cantidad de elementos aumenta en uno.
- Al agregar un elemento, el `Bag` contiene el elemento.
- Para remover un elemento, el `Bag` debe contener el elemento.
- Al remover un elemento, el `Bag` ya no contiene el elemento.
- Al remover un elemento, la cantidad de elementos en el `Bag` disminuye en uno.

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