Programación orientada a objetos. Examen diciembre 2012.

Viernes 21 de diciembre de 2012.

1.- Sean las siguientes clases e interfaces:

```
interface IComponent {
    Int32 X { get; } Int32 Y { get; }
}
interface IDisposable {
    void Dispose();
}
class Control : IComponent, IDisposable {
    public Boolean Visible { get { /* ... */ } set { /* ... */ } }
}
interface IHtmlElement {
    String Name { get; }
}
abstract class HtmlControl : Control, IHtmlElement {

    public abstract void Render();
}
interface InputControl { }
class HtmlInput : HtmlControl, InputControl {
    public void Clear() { /* ... */ }
}
class HtmlLink : HtmlControl {
    public void GoToURL() { /* ... */ }
}
// Nota: asume que todas las clases están implementadas completa y correctamente

class Program {
    public static void Main(String[] args)
    {
        /*1*/ IComponent a = new InputControl();
        /*2*/ Control b = new HtmlControl();
        /*3*/ HtmlControl c = b;
        /*4*/ IHtmlElement d = new HtmlLink();
        /*5*/ IDisposable e = d;
        /*6*/ HtmlInput f = d;
        /*7*/ HtmlInput g = new HtmlLink();
    }
}
```

1.1.- Indica las líneas de código **incorrectas**. Si una asignación es incorrecta, de todas formas asume que la variable fue definida.

1.2.- Indica los tipos de la variable **d** y los tipos del objeto creado en la línea **7**.

1.3.- Sea el siguiente código:

```
        public static void Main(String[] args)
        {
            Control a = new Control();
            Control b = a;
            a.Visible = true;
            b.Visible = !a.Visible;

        }
```

¿Cuál es el valor de **b.Visible** al final del código? ¿Cuál es el valor de **a.Visible** al final del código?

1.4.- Un objeto referenciado por una variable de tipo HtmlElement ¿Puede poseer una implementación de void GoToUrl()? En caso afirmativo, provee un ejemplo de ésto, en caso negativo, explica por qué.

2.- El siguiente extracto de código fue obtenido de un sistema de planta de una reconocida industria Uruguaya. El código se ha traducido a C# y se ocultaron los strings de conexión reales por temas de seguridad.

```
class Conexion
{
    public string planta;
    public void EjecutarConsulta(string consulta)
    {
        string conexionString;
        //se asigna el string de conexión adecuado para la planta
        switch (planta)
        {
            case "planta1":
                conexionString = "..conexion planta 1...";
                break;
            case "planta2":
                conexionString = "..conexion planta 2...";
                break;
            case "planta3":
                conexionString = "..conexion planta 3   ...";
                break;
            default:
                conexionString = "..conexion casa matriz...";
                break;
        }
        //Se ejecuta la consulta sobre el servidor de base de datos
        SqlConnection cnx = new SqlConnection(conexionString);
        SqlCommand cmd = new SqlCommand(consulta, cnx);
        cnx.Open();
        cmd.ExecuteNonQuery();
        cnx.Close();
    }
}
```

Este programa fue escrito en un lenguaje que no es orientado a objetos. Teniendo esto en cuenta, responde a las siguientes preguntas:

2.1.- ¿Qué problema encuentras en el código? Debes basarte en los principios del curso para justificar tu respuesta.

2.2.- Si pudieras re programar esta clase con tus conocimientos actuales, ¿qué cambios realizarías? Programa estas modificaciones.

3.- Propone un fragmento de código donde sea provechoso hacer uso de por lo menos una clase abstracta. El código provisto debe dejar en claro el objetivo de las clases abstractas, contra el uso de interfaces o clases concretas, por lo que te sugerimos que te esmeres en el ejemplo para demostrarlo. Agrega una breve descripción al código que proveas para explicar tu solución.

4.- Sea el siguiente código:

```
class Pelicula { }
class Entrada
{
    private Pelicula pelicula;
    public Pelicula Pelicula { get { return pelicula; } set { pelicula = value; }
}
class Persona
{
    private Entrada entrada;
    public Entrada Entrada { get { return entrada; } set { entrada = value; } }
}

public class Cine
{
    private Pelicula proyectando;
    Pelicula Proyectando { get { return proyectando; } }
    private IList<Persona> personas = new List<Persona>();
    Int32 Personas { get { return personas.Count; } }
    public Cine() {
    }
    void Proyectar(Pelicula pelicula) {


        proyectando = pelicula;



    }
    void TerminarProyeccion() {


        proyectando = null;



    }
    void Entrar(Persona p) {


        Int32 aux = Personas;
        personas.Add(p);
        p.Entrada.Pelicula = null;



    }
    void Salir(Persona p) {


        personas.Remove(p);



    }
}
```

La clase **Cine** debe cumplir las siguientes condiciones:

a) No se puede proyectar una película mientras otra se está proyectando.
b) Luego de llamar a proyectar, **Proyectando** debe retornar la película proyectada.
c) Solo se puede terminar la proyección cuando se está proyectando una película.
d) Luego de terminar la proyección, **Proyectando** debe retornar nulo.
e) Una persona solo puede entrar al cine cuando se está proyectando una película.
f) Luego que una persona entra al cine, la cantidad de personas adentro aumenta en uno.
g) Una persona solo puede salir del cine si antes había entrado.
h) Una persona solo puede entrar al cine si tiene una entrada para la película que se está proyectando.
i) Una vez que la persona entra al cine, pierde su entrada.
j) En un cine, se está proyectando una película o la cantidad de personas es cero.

4.1.- Indica que condiciones corresponden a precondiciones, postcondiciones e invariantes.
4.2.- Completa **en la letra**, las condiciones definidas anteriormente utilizando Debug.Assert(bool condition). El código entregado ya resuelve la lógica del problema, tu tarea es únicamente agregar las pre condiciones, post condiciones e invariantes donde corresponda. No es necesario, pero si lo prefieres puedes agregar algo adicional.

5.- Sea el siguiente código:

```
public class Conductor
{
    public void Conducir()
    {
    /*…*/
    }
}
public class Trabajador
{
    public void Trabajar()
    {
    /*…*/
    }
}
public class Programa
{
    public static void Main()
    {
         ConductorProfesional o = new ConductorProfesional();
         o.Conducir();
         o.Trabajar();
         o.LlevarPasajeros();
    }
}
```

Escribe la clase ConductorProfesional que falta, asumiendo que el lenguaje tiene o no herencia simple o múltiple, según las siguientes opciones:

	Tiene herencia simple	Tiene herencia múltiple
5.1.-	Si	Si
5.2.-	Sí	No
5.3.-	No	Sí
5.4.-	No	No

Si dos opciones tienen la misma respuesta, no escribas nuevamente la clase, sólo indícalo.

6.- Observa el siguiente código:

```
class Maya
{
    public DateTime PredecirFinDelMundo(IPredicion prediccion)
    {
        return prediccion.Predecir();
    }
}
interface IPredicion
{
    DateTime Predecir();
}
interface IConocimiento
{
    void UtilizarConocimiento();
}
class Astronomia:IConocimiento
{
    public void UtilizarConocimiento()
    { /* Implementación correcta */ }
}
class Calendario
{
    public DateTime ObtenerFecha(IConocimiento conocimiento)
    { /* Implementación correcta */ }
}
class Program
{
    static void Main(string[] args)
    {
        Calendario cal = new Calendario();
        Astronomia astro = new Astronomia();
        Maya maya = new Maya();
    }
}
```

Sin modificar la clase `Maya`, debes lograr que el mismo pueda predecir la fecha del fin del mundo. Considera que el uso de un `Calendario` y conocimientos de `Astronomía` son requeridos para dar una predicción.

<u>Consideraciones generales:</u>

- Todos los ejercicios valen lo mismo.
- Pon nombre en todas las hojas.
- Numera todas las hojas.
- Puedes escribir en ambas caras de las hojas.
- Contesta en la letra los ejercicios que así lo indican.
- El trabajo es individual

- ¡Mucha suerte!