Programación orientada a objetos. Examen diciembre 2007.

Jueves 20 de diciembre de 2007.

Sean las siguientes clases:

```
interface Control {
   void Paint();
}
interface Container {
   IList<Control> Controls { get; }
}
class SimpleContainer : Container {  /* implementacion completa y correcta */    }
class ContainerControl: Control, Container {
   // implementacion completa y correcta
   void Clean() { /* .... */ }
}
class Form: ContainerControl {
   private Boolean open = false;
   Boolean IsOpen { get { return open; } }
   void Open() { open = true; }
}
class Dialog: ContainerControl {
   Boolean IsOk { get { /* .... */ } }
   void Show() { /* .... */ }
}
class Program {
   void Main() {
      /*1*/ Control a = new Form();
      /*2*/ Container b = new Dialog();
      /*3*/ Form c = new ContainerControl();
      /*4*/ Dialog d = c;
      /*5*/ SimpleContainer e = new ContainerControl();
      /*6*/ ContainerControl f = new Form();
      /*7*/ SimpleContainer g = new Container();
    }
}
```

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 **c** y los tipos del objeto creado en la línea **5**.

1.3 ¿Qué cambios deberías realizar para que la siguiente asignación sea válida? No puedes cambiar la asignación.

```
Form h = new Dialog();
```

1.4 La siguiente asignación ocurre dentro de un método declarado en la clase de un objeto **O.**

```
ContainerControl i = new Dialog();
```

¿Qué mensajes puedes enviar al objeto referenciado por la variable **i** desde el objeto **O**? ¿Cómo lo sabes?

El siguiente código se utiliza para controlar el uso de un cine.

```
class Pelicula { }
class Entrada {
   private Pelicula pelicula;
   Pelicula Pelicula { get { return pelicula; } set { pelicula = value; } }
}
class Persona {
   private Entrada entrada;
   Entrada Entrada { get { return entrada; } set { entrada = value; } }
}
class Cine {
   /* modificar */
   void Proyectar(Pelicula pelicula)      { /* modificar */ }
   void TerminarProyeccion()              { /* modificar */ }
   void Entrar(Persona p)                 { /* modificar */ }
   void Salir(Persona p)                  { /* modificar */ }
   Pelicula Proyectando                   { get { /* modificar */ } }
   Int32 Personas                         { get { /* modificar */ } }
}
```

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 retorna 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.

2.1 Indica que condiciones corresponden a precondiciones, postcondiciones e invariantes.

2.2 Modifica la clase **Cine** para cumplir todas las condiciones mencionadas. Declara las precondiciones, postcondiciones e invariantes usando **Debug.Assert(bool Condition)**.

Sea el siguiente código

```
class Cuadrado: Rectangulo {
   public Cuadrado(Int32 lado) : base(lado, lado) { }
   public override Int32 Ancho { set { base.Ancho = value; base.Largo = value; } }
   public override Int32 Largo { set { base.Largo = value;  base.Ancho = value; } }
}
class Rectangulo {
   private Int32 ancho;
   public virtual Int32 Ancho { get { return ancho; } set { ancho = value; } }
   private Int32 largo;
   public virtual Int32 Largo { get { return largo; } set { largo = value; } }
   public Rectangulo(Int32 ancho, Int32 largo) {
      Largo = largo;
      Ancho = ancho;
   }
}
class Cliente {
   public void Metodo(Rectangulo rectangulo) {
      rectangulo.Ancho = 4;
      rectangulo.Largo = 5;
      Debug.Assert(rectangulo.Ancho * rectangulo.Largo == 20);
   }
}
```

3 ¿Qué principio se está violando? ¿Por qué? ¿Qué clase causa que se viole?

Sea el siguiente código

```
interface ILista {
   void Agregar(Object o);
   Boolean Existe(Object o);
}
class Lista: ILista {
   private IList lista = new ArrayList();
   public virtual void Agregar(Object o) { lista.Add(o); }
   public virtual Boolean Existe(Object o) { return lista.Contains(o); }
}
```

4.1 Programa dos nuevas clases llamadas **ListaSinRepetidos** de tipo **ILista** que no agreguen un objeto si el mismo ya está en la lista. Utiliza un mecanismo de reutilización diferente en cada caso e indica cuál estás usando. Debes reutilizar el código provisto, pero no puedes modificarlo.

4.2 ¿Cuál de los dos mecanismos eligirías si programaras un sistema real? ¿Por qué? Si **Lista** tuviera un método **Int32 Cantidad** que retorna la cantidad de elementos de la lista y una postcondición en **void Agregar** que dice que una vez agregado un elemento la cantidad de elementos aumenta en uno ¿Impactaría ésto en tu decisión? ¿Por qué?.

5 Modifica el código de **Lista** e **ILista** del ejercicio 4 para utilizar genéricos.

6.1 Provee un ejemplo de código donde el resultado del programa es distinto si se utiliza encadenamiento estático que si se utiliza dinámico. Indica cual es la salida del programa para cada caso.

6.2 ¿En base a qué se determina el método a encadenar en encadenamiento estático? ¿Y en dinámico?.

Sea el siguiente código:

```
class Facultad {
   IDictionary<String, Materia> materias = new Dictionary<String, Materia>();
   public IDictionary<String, Materia> Materias { get { return materias; } }

   IList<Materia> materiasLlenas = new List<Materia>();
   public IList<Materia> MateriasLlenas { get { return materiasLlenas; } }

   IDictionary<Alumno, IList<Materia>> materiasAlumno =
                  new Dictionary<Alumno, IList<Materia>>();
   public IDictionary<Alumno, IList<Materia>> MateriasAlumno { get { return materiasAlumno; } }
}
class Alumno {
}
class Materia {
   private Int32 inscriptos;
   public Int32 Inscriptos { get { return inscriptos; } set { inscriptos = value; } }
}
class Program {
   private Facultad facultad = new Facultad();
   void Main() {
      Alumno a = new Alumno();
      Materia m = facultad.Materias["POO"];
      m.Inscriptos += 1;
      if (m.Inscriptos >= 50) {
         facultad.MateriasLlenas.Add(m);
      }
      facultad.MateriasAlumno[a].Add(m);
   }
}
```

7.1 Crítica el código en base a criterios de cohesión y acoplamiento. Justifica tus críticas.

7.2 Explica claramente que harías para mejorar el acoplamiento de la clase **Program** y la cohesión de la clase **Facultad**. Si crees que la mejor forma que tienes para explicarlo es programandolo, hazlo.