Programación orientada a objetos. Examen julio 2005.

Viernes 7 de julio de 2005.

1. Las clases `Película` y `Video` que aparecen a continuación son algunas de las clases de una aplicación para un club de video. En este club las películas que son estreno tienen un precio y las que no son estrenos tienen otro precio. ¿Las clases son adecuadas? Si no lo son escríbanlas nuevamente como es debido.

```
…
public class Película
{
  private String nombre;
  public String Nombre { get { return nombre; } }

  private Int32 año;
  public Int32 Año { get { return año; } }

  public Película(String nombre, Int32 año) {
    this.nombre = nombre;
    this.año = año;
  }
}

…
public class Video
{
  private Película película;
  public Película Película { get { return película; } }

  private Int32 númeroCopia;
  public Int32 NúmeroCopia { get { return númeroCopia; } }

  private Boolean estreno = true;
  public Boolean Estreno {
    get { return estreno; } set { estreno = value; } }

  private Double precio;
  public Double Precio { get { return precio; } set { precio = value; } }

  public Video(Película película, Int32 númeroCopia, Double precio)
  {
    this.película = película;
    this.númeroCopia = númeroCopia;
    this.precio = precio;
  }
}
```

2. Sean las clases `Perro` y `Cruza` para una aplicación que permite registrar las cruzas entre perros de raza. Completen en las clases el código que dice "…". Tengan en cuenta que:

   - Un perro no puede cambiar de raza.
   - Un perro no puede cambiar de sexo.
   - No se puede cambiar el perro y la perra de una cruza.
   - No se puede quitar cachorros de una cruza.

```
…
public enum Sexo { Macho, Hembra }
```

```
…
public class Perro {
  private String nombre;
  public String Nombre { get { return nombre; } }

  private Sexo sexo;
  public Sexo Sexo { get { return sexo; } set { sexo = value; } }

  public Perro(…) {…}
  public Cruza Cruzar(…) {…}
}

…
public class Cruza {
  private Perro perro;
  public Perro Perro {…}

  private Perro perra;
  public Perro Perra {…}

  private IList prole = new ArrayList();

  public Cruza(…) {…}

  public void AddCachorro(Perro cachorro) {
    prole.Add(cachorro);
  }

  public IEnumerator GetProle() {
    return prole.GetEnumerator();
  }
}
```

3. Definan en forma breve y precisa **precondición** y **poscondición**. Agreguen a las clases `Perro` y `Cruza` del ejercicio anterior las precondiciones y poscondiciones que consideren necesarias para asegurar que:
   - Un perro no se puede cruzar consigo mismo.
   - Un perro se debe cruzar con otro de su misma raza.
   - Un pero no se puede cruzar con otro del mismo sexo.
   - Un perro no se puede cruzar con sus padres ni con ninguno de sus hermanos.
   - Una cruza no puede cambiar los perros que intervinieron en ella.

   Agreguen estas afirmaciones al código del ejercicio anterior usando `Debug.Assert()`. No tienen porqué escribir nuevamente toda la clase, mientras indiquen claramente dónde se inserta cada modificación. Pueden agregar otros métodos además de los que ya existen.

4. Consideren las clases `Alumno` y `Materia` para la inscripción a cursos y exámenes.

   La clase Alumno tiene:
   - Nombre: Es el nombre del alumno.
   - Materias: La lista de materias que debe cursar durante toda la carrera.

   La clase `Materia` tiene:
   - `Nombre`: Es el nombre de la materia.
   - `AnotadoCurso`: Inicialmente es `false`. Pasa a `true` cuando el alumno se inscribe al curso.
   - `AnotadoExamen`: Inicialmente es `false`. Pasa a `true` cuando el alumno se inscribe al examen.
   - `AproboCurso`: Inicialmente es `false`. Pasa a `true` cuando el alumno aprueba el curso.
   - `AproboExamen`: Inicialmente es `false`. Pasa a `true` cuando el alumno aprueba el examen.

Usando **precondiciones** y **poscondiciones** implementen en la clase `Alumno` los métodos `AnotarseCurso(String nombre)` y `AnotarseExamen(String nombre)` teniendo en cuenta las siguientes consideraciones:

- Si el alumno no está anotado a un curso, no puede tener ni el curso ni el examen aprobado, ni puede estar anotado al examen.
- Si el alumno no tiene aprobado un curso, no puede estar anotado al examen, ni tener el examen aprobado.
- Si el alumno no está anotado al examen, no puede tener el examen aprobado.
- El alumno se puede inscribir a un curso si tiene aprobados los cursos de las materias previas.
- El alumno se puede inscribir a un examen si tiene aprobado el curso y los exámenes de las materias previas.

```
…
public class Alumno
{
  private String nombre;
  public String Nombre { get { return nombre; } }

  private IList materias = new ArrayList();
  public IList Materias { get { return materias; } }

  public Alumno(String nombre) {
    this.nombre = nombre;
  }
}

…
public class Materia {

  private String nombre;
  public String Nombre { get { return nombre; } }

  private Boolean anotadoCurso = false;
  public Boolean AnotadoCurso { get { return anotadoCurso; }
    set { anotadoCurso = value; } }

  private Boolean anotadoExamen = false;
  public Boolean AnotadoExamen { get { return anotadoExamen; }
    set { anotadoExamen = value; }}

  private Boolean aprobadoCurso = false;
  public Boolean AprobadoCurso { get { return aprobadoCurso; }
    set { aprobadoCurso = value; } }

  private Boolean aprobadoExamen = false;
  public Boolean AprobadoExamen { get { return aprobadoExamen; }
    set { aprobadoExamen = value; } }

  private IList previas = new ArrayList();

  public Materia(String nombre) {
    this.nombre = nombre;
  }
}
```

5. Dos de las tres formas más comunes de reutilización en programación orientada a objetos son **herencia** y **composición-delegación**. Ignorando la semántica de las relaciones de generalización-especialización y exclusivamente desde el punto de vista de la reutilización de código ¿puedo hacer lo mismo con composición-delegación que con herencia? Justifiquen la respuesta de forma breve y concreta.

6. Dadas las siguientes clases agreguen las que haga falta para que el programa compile. Usen para una de ellas herencia y para la otra composición-delegación. No pueden escribir de nuevo `MétodoReutilizable()`.

```
public class Base
{
    public void MétodoReutilizable()
    {
        Console.WriteLine("Esto es algo útil");
    }
}

public class Programa
{
    static void Main()
    {
        Sucesora s = new Sucesora();
        s.MétodoReutilizable();

        Compuesta c = new Compuesta();
        c.MétodoReutilizable();
    }
}
```

7. Usando las clases del ejercicio anterior hagan la menor cantidad de cambios necesarios para que `Programa` imprima lo mismo en la consola.

```
class Programa
{
    static void Main()
    {
        Tipo s = new Sucesora();
        s.MétodoReutilizable();

        Tipo c = new Compuesta();
        c.MétodoReutilizable();
    }
}
```

8. ¿Qué está mal en el siguiente programa? ¿Cómo lo resolverían? Justifiquen en forma breve y concreta la respuesta.

```
public class Cuadrado
{
    private Int32 lado;
    public Int32 Lado { get { return lado; } set { lado = value; }}
    public Cuadrado(Int32 lado)
    {
        this.lado = lado;
    }
    public virtual Int32 Area()
    {
        return lado * lado;
    }
}

public class Cubo:Cuadrado
{
    public Cubo(Int32 lado):base(lado) {
    }
    public override Int32 Area()
    {
        return Lado * Lado * 6;
    }
}
```

```
class Programa
{
    static void Main()
    {
        Cuadrado cuadrado = new Cuadrado(10);
        Console.WriteLine(cuadrado.Area());

        Cuadrado cubo = new Cubo(10);
        Console.WriteLine(cubo.Area());
    }
}
```

9. Enuncie el **principio de sustitución de Liskov**. Busque la aplicación del principio en alguno de los ejercicios anteriores e identifique en ese código los términos que aparecen en el principio. ¿Hay alguna relación entre el principio de sustitución de Liskov y el **diseño por contrato**?

10. ¿Puede haber **polimorfismo** en un lenguaje en el que no hay **herencia**? Justifiquen su respuesta. Expliquen en forma clara y precisa cómo funciona el **encadenamiento estático** y el **encadenamiento dinámico** en esos lenguajes. ¿El concepto es aplicable a los lenguajes en los que no se declara el tipo de las variables y argumentos? Justifiquen su respuesta.