Programación orientada a objetos. Primer parcial 2005.

Viernes 22 de abril de 2005.

Al igual que las demás instancias de evaluación este parcial 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.

1. ¿Está bien que una clase tenga **estado** y no **comportamiento**? ¿Y que tenga **comportamiento** y no **estado**? Consideren la clase `Actor` que aparece a continuación. Los actores tienen la responsabilidad de conocer su nombre ¿Hay algo mal en esta clase `Actor` tal como está programada? Si lo hay escriban la versión corregida.

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

2. 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;
  }
}
```

3. Los objetos de la clase `Película` tienen ahora también la responsabilidad de conocer los actores de la película y la de agregar actores a la película. Modifiquen la clase `Película` según sea necesario. Usen la clase `Actor` de la primera pregunta.
4. ¿Puede un objeto tener un tipo que tenga más operaciones que el tipo definido por la clase de ese objeto? Justifiquen la respuesta. Programen un pequeño ejemplo en C# si fuera posible.
5. ¿Puede tener más de un tipo un objeto de una clase que no implementa una interfaz? Justifiquen la respuesta. Programen un pequeño ejemplo en C# si fuera posible.
6. Cuando un objeto cliente necesita la colaboración de un objeto servidor con cierto tipo… ¿Cuáles son las obligaciones del cliente respecto del tipo? ¿Y las del servidor? ¿Cuáles son los derechos del cliente? ¿Y los del servidor? Formulen las respuestas usando los términos mensaje, operación, método.
7. 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();
  }
}
```

8. 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.

9. 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;
    }
  }
```