Programación Orientada a Objetos. Segundo Parcial 1er semestre 2010.
Viernes 11 de junio de 2010

1 Sea el siguiente código:

```
class MundialFutbol {
    private IList cuadros = new ArrayList();
    private IList partidos = new ArrayList();
    void AgregarCuadro(Cuadro cuadro) {}
    void AgregarPartido(Partido cuadro) {}
    Cuadro Ganador {}
}
class Cuadro {
    private IList jugadores = new ArrayList();
    void AgregarJugador(Jugador jugador) {
    }
    void BorrarJugador(Jugador jugador) {
    }
    void Jugar(Cuadro cuadro) { }
}
class Jugador {}
```

Y las siguientes condiciones:

a) Luego de jugar un partido, el cuadro ganador gana 3 puntos y el perdedor 0, o ambos ganan 1 punto (puedes definir como quieras quien gana y quien pierde).
b) No se puede borrar un jugador de un cuadro luego de que el cuadro jugo al menos un partido.
c) El mejor cuadro es el que sumó mayor puntaje hasta el momento. Si hay más de uno, es cualquiera de ellos.
d) Un jugador no puede estar jugar en más de un cuadro.
e) Una vez inscrito un cuadro en el mundial, el mismo queda registrado.
f) Solo se puede registrar en el mundial un partido de cuadros están registrados en el mundial.

1.1. Identifica qué condiciones corresponden a precondiciones, postcondiciones o invariantes.
1.2. Utilizando Debug.Assert(), agrega el código necesario para verificar esas precondiciones, postcondiciones e invariantes.

2.1 Compara el uso de herencia frente a composición y delegación respecto a acoplamiento.
2.2 ¿Por qué se dice que la composición y delegación permite reutilización dinámica, mientras que herencia reutilización estática?.

Sea el siguiente código:

```
class Monstruo {}
class Cazador { void Cazar(Monstruo monstruo) { } }
```

La cultura de la especialización ha llevado a que los Cazadores se especialicen en cierto tipo de monstruos.
3.1 Programa Cazadores que solo puedan cazar cierto tipo de Monstruos. Asegurate que no haya cazadores que cacen cosas que no son monstruos.
3.2 Se agrega el siguiente código:

```
class MonstruoAtrapado {
      private Monstruo monstruo;
      Monstruo Atrapado {
            get { return monstruo; }
            set { monstruo = value; }
      }
}
class Orco : Monstruo {}
class Cazador {
      MonstruoAtrapado Atrapar(Monstruo monstruo) { }
}
```

Se desea que cuando se atrape un Orco, se retoner un MonstruoAtrapado cuyo Monstruo Atrapado sea un Orco, sin necesidad de "castear" el Monstruo a Orco luego de atraparlo. Realiza los cambios necesarios para permitir ésto.

4.1 Discute si el tener una clase abstracta con únicamente métodos concretos puede tener algún escenario de uso. Si contestas afirmativamente, da un ejemplo de cómo lo aplicarías.
4.2 Compara una clase abstracta donde todo los métodos son abstractos con una interfaz con las mismas operaciones. ¿Cuáles son sus diferencias? ¿Cuál utilizarías?.
4.3 Indica si el siguiente código es correcto. En caso de no serlo, indica brevemente **todos** sus errores.

```
abstract class NombreClase {
      abstract void M1();
}
interface NombreInterface : NombreClase {
      void M3();
}
class NombreHija: NombreInterface {
      override void M1() { }
      abstract void M3();
}
```

5. El siguiente código se utiliza en una editorial para generar poemas en base a ciertos parámetros. Según una palabras clave utiliza un algoritmo complicado que estudia las rimas por medio de reglas de razonamiento en base a las características de las palabras ingresadas y la base de datos de palabras que puede utilizar. En este momento están en proceso de ampliación y necesitan tener escritores que escriban otros tipos de textos además de Poemas.

```
class Poema {
    public string Titulo { get; set; }
    private IList parrafo = new ArrayList();
    public IList Parrafos {get { return parrafo; } }
}
class Escritor {
    Poema GenerarPoema(String titulo,
                       String clave1, String clave2, String clave3) {
        Poema nuevo = new Poema();
        nuevo.Titulo = titulo;
        for(int i = 0; i < 3; i++) {
            nuevo.Parrafos.Add(ProcesarParrafo(
                  clave1, clave2, clave3, i));
        }
        return nuevo;
    }
```

```
    String ProcesarTexto(String c1, String c2, String c3, int i) {
        if (i == 0) {
            return Rimar(c1);
        } else if (i == 1) {
            return Rimar(c2);
        } else {
            return Rimar(c3);
        }
    }
    String Rimar(String c) { /* genera el texto */ }
}
```

5.1
Critica el código en base a ISP, SRP y DIP.
5.2
Programa el código nuevamente para que cumpla los principios mencionados, si es que no los cumplía.

6.
En una empresa se desea implementar un sistema para flujos de trabajo. Los trabajos se deben poder combinar de varias formas, insertando puntos de control donde la calidad del resultado de los mismos es evaluada. Hay trabajos como: escribir un reporte, enviar un mail al gerente avisando de un problema, realizar una compra, etc.

```
interface ITrabajo {
      Resultado Realizar();
}
interface IEvaluador {
      bool Evaluar(Resultado r);
}
```

Se quiere poder combinar trabajos y evaluadores permitiendo cosas como:

- Si un evaluador evalua negativamente el resultado de un trabajo A, se debe realizar un trabajo B, en otro caso, se debe realizar un trabajo C.
- Realizar un trabajo A, y luego realizar otro trabajo B.
- Evaluar un trabajo A usando varios evaluadores.

Programa un modelo que te permita realizar este tipo de procesos. Intenta programarlo de forma de maximizar las chances de reutilización y combinación. Implementa clases para permitir los tres casos anteriores.