Programación Orientada a Objetos - *Examen 25 de Julio de 2011.*

Ejercicio 1

Basándote en el siguiente código, haz que un `Programador` genere un programa. No puedes modificar el código pero sí agregar nuevas líneas de código:

```
public class Programa{}
public interface ICompilador
{
    Programa Compilar(String programa, ILenguaje lenguaje);
}
public interface ILenguaje
{
    String Name { get; }
}
public class Programador{ }
```

Ejercicio 2

Sea el siguiente código:

```
public class Foto
{
    public double MegaPixels;
    public double MP()
    { /* Devuelve los MegaPixels de la foto */}
    public void Render()
    { /* Código que muestra la foto */}
}
public class PortaRetratosDigital
{
    IList fotos = new ArrayList();
    public void MostrarFotos() {
        foreach (Foto f in fotos){
            if (f.MP() > 3)
            {
                f.Render();
            }
        }
    }
    public void ImprimirEnImpresora(Impresora sonix) {
        foreach (Foto f in fotos){
            if (f.MP() > 3)
            {
                sonix.Imprimir(f);
            }
        }
    }
}
public class Impresora {
       public void Imprimir(Object objeto) {/*impl*/}
}
```

Critique el código en base a:

- Encapsulacion
- LSP
- DIP
- SRP

Ejercicio 3

El Gerente de Sistemas le comunica que el cliente quiere modificar el `PortaRetratosDigital`. Se le solicita que además de mostrar fotos, pueda mostrar videos y en un futuro otros medios como pueden ser videos 3D. También se desea poder setear diferentes condiciones para mostrar algo en la pantalla. Actualmente solo verifica que se desplieguen las fotos de mas de 3 megapixles, y se quieren poder agregar otras reglas.

Se quiere modificar el código del ejercicio 2 para cumplir con estos requisitos además de solucionar los principios que no se cumplían en el código original (ejercicio 2).

Ejercicio 4

Sean las siguientes clases e interfaces:

```
interface IDeporte { }
interface IAcuatico : IDeporte { }
interface ITerrestre : IDeporte { }
class Futbol : ITerrestre { }
class BabyFutbol : Futbol { }
class Rugby : ITerrestre { }
class Hockey : ITerrestre { }
abstract class Elemento { }
class Surf : IAcuatico, Elemento { }
class Natacion : IAcuatico { }
class WaterPolo : IAcuatico, Elemento { }
class Programa {
    void Principal()
    {
        /*1*/ ITerrestre f = new Futbol();
        /*2*/ BabyFutbol bf = f;
        /*3*/ IDeporte r = new Natacion();
        /*4*/ Elemento e = new Elemento();
        /*5*/ Natacion s = r;
        /*6*/ IAcuatico w = new WaterPolo();
        /*7*/ Surf s = w;
    }
}
```

a) Indica las líneas de código incorrectas. (Aclaración: Si una asignación es incorrecta, de todas formas asume que la variable fue definida).

b) Indica los tipos de la variable **bf** de la **línea 2** y los tipos del objeto creado en la **línea 6**

c) Realiza los cambios necesarios en el código anterior para que:

i- el objeto creado en la línea 1 pueda recibir los siguientes mensajes: jugar(), agregarJugador()

ii- la variable declarada en la línea 7 pueda invocar los siguientes mensajes: subirTabla(), romperOla()

Ejercicio 5

El siguiente código permite determinar si alguno de los jugadores de un país fue elegido el mejor jugador de la Copa América:

```
class Jugador {
    private String nombre;
    public String Nombre { get; set; }
}
class Evaluador
{
    IList jugadores = new ArrayList();
    bool resultado = false;
    bool MejorJugador()
    {
        foreach (Jugador j in jugadores)
        {
            if (j.Nombre == "Luis Suarez")
                resultado = true;
        }
        return resultado;
    }
}
```

a) Modifica el código anterior para que la política que define el mejor jugador no dependa de la clase Evaluador.

b) Indica en el código invocaciones a métodos que se encadenan dinamicamente e invocaciones que se desencadenen estaticamente. Si no tienes encuentras esto en el código que generaste, agrega el código adicinal que precises para ejemplificarlo. Justifica.

Ejercicio 6

a) Explica que entiendes por “Diseño por Contratos”.

b) Programa una clase cualquiera, y ciertas operaciones para las cuales definas una invariante de clase, una pre-condición y una post-condición. Solo debes implementar lo referente a las condiciones que crees.

c) Explica como puede influir negativamente en el principio de sustitución de Liskov(LSP) una mala definición de pre-condiciones, post-condiciones e invariantes.

Ejercicio 7

Sean las siguientes clases e interfaces:

```
interface CopaAmerica {
    int Goles { get; set; }
}

class Uruguay : CopaAmerica {
    private int goles;
    public int Goles { get { return goles; } set { goles = value; } }
}

class Paraguay : CopaAmerica {
    private int goles;
    public int Goles { get { return goles; } set { goles = value; } }
}
```

Y es siguiente código:

```
    Uruguay celeste = new Uruguay();
    celeste.Goles = 6;
    CopaAmerica charrua = new Uruguay();
    charrua = celeste;
    charrua.Goles = 3;
```

```
CopaAmerica rojo = new Paraguay();
rojo.Goles = 0;
Console.WriteLine(charrua.Goles + celeste.Goles + rojo.Goles);
```

a) ¿Qué imprime por consola?

b) ¿Cuántos objetos iguales hay al final del código y en que líneas se crean? Si no encuentras objetos iguales realiza las modificaciones necesarias en el código para que los haya.

c) ¿Cuántas variables referencian a objetos idénticos al final del código? Si no encuentras variables que referencien a objetos idénticos realiza las modificaciones necesarias en el código para que las haya.

d) ¿Son los objetos instancia de Paraguay inmutables? Realiza los cambios necesarios para que los objetos instancia de Paraguay sean inmutables, si no lo son, o mutables, si ya son inmutables.