Programación Orientada a Objetos – Examen 01 de Marzo 2013

Ejercicio 1.- Sean las siguientes clases e interfaces:

```
interface IFigura {
        public double Area();
    }
    interface IRectangulo:IFigura { }
    class Tecla : IRectangulo
    {   //implementación completa y correcta
        public void Presionar() { /*imp*/ }
    }
    interface IDisplay: IRectangulo{}
    class Celular : Tecla,IDisplay {
        //implementación completa y correcta
        public void Llamar() { /*imp*/ }
    }
    class Nokia : Celular {
        //implementación completa y correcta
    }
    class Rombo : IFigura {
        //implementación completa y correcta
    }
    class Cometa : Rombo {
        //implementación completa y correcta
        public void Volar() { /*imp*/ }
    }
    interface ICirculo : IFigura { }
    abstract class Balon : ICirculo {
        //implementación completa y correcta
        abstract void Rodar();
    }
    class Bochin : Balon { }

    class Program
    {
        static void Main(string[] args)
        {
            /*1*/ IFigura d = new Tecla();
            /*2*/ IRectangulo r = d;
            /*3*/ Nokia n = new Celular();
            /*4*/ ICirculo b = new Balon();
            /*5*/ Balon ch = new Bochin();
            /*6*/ IFigura f = new Rombo();
            /*7*/ Cometa c = f;
            /*8*/ IRectangulo g = new IDisplay();
        }
    }
```

1.1- Indica las líneas de código incorrecto. En caso de asignaciones inválidas, asume que las variables fueron definidas.
1.2- Define los tipos de las variables creadas en las líneas **3** y **8**.
1.3- Define los tipos del objeto creado en la línea **6**.
1.4- Indica qué mensajes puede recibir un objeto, bajo los términos de la variable declarada en la línea **5**.
1.5- Indica que mensajes puede recibir un objeto como el creado en la línea **5**.

Ejercicio 2.- Observa el siguiente código de una aplicación para recomendar ejercicios, de un conocido gimnasio:

```
class VirtualGym
{
    public void ElegirEjercicio(Persona persona)
    {
        if (persona.Peso > 80)
        {
            EjercicioBici b = new EjercicioBici();
            b.HacerBici(persona);
        }
        else
        {
            EjercicioCorrer c = new EjercicioCorrer();
            c.Correr(persona);
        }
    }
}

class Persona
{
    public int Peso { get; }
}

class EjercicioCorrer
{
    public void Correr(Persona p)
    { /* Implementación correcta */ }
}
class EjercicioBici
{
    public void HacerBici(Persona p)
    { /* Implementación correcta */ }
}
```

Se desea mejorar esta aplicación de forma tal que pueda recomendar otros ejercicios que no sean solamente correr y bicicleta, y que además pueda tomar en cuenta otros atributos de la persona(no solo el peso), para seleccionar el ejercicio más adecuado. Por ejemplo, su altura, ritmo cardíaco, etc.

2.1.- Programa la mejor solución para cumplir con estos requisitos.

Ejecricio 3.- Se quiere modelar una clase que represente cocktails de modo genérico. Listamos algunos cocktails a modo de ejemplo: "Jack on the rocks" – licor Jack Daniel´s y hielo, "Mohito cubano" – Ron, hierba buena, azúcar, jugo lima y hielo, "Apple Martini" – vodka, cereza y licor de manzana. Para lograr esto deberás:

3.1- Programar una clase llamada **Cocktail**, que represente tragos de manera genérica, permitiendo representar tragos diversos. Esta clase debe poseer al menos un constructor.
3.2- Programar otras interfaces o clases que puedan colaborar con **Cocktail**, para lograr que esta sea general.
3.3- Programar desde el Program, la creación de 3 cocktails diferentes (3 objetos instanciados de la clase Cocktail) para ejemplificar el uso de tu solución.
Nota: Se evaluará lo genérico que sea el diseño de la solución. Intenta pensar cómo influiría la creación de nuevos cocktails en tu solución para auto-evaluar tu diseño.

Ejercicio 4.- En la siguiente tabla encontrarán dos maneras diferentes de reutilizar los saltos que bien sabe aplicar la clase **Hop**:

<table>
<tr><th>Mecanismo A</th><th>Mecanismo B</th></tr>
<tr><td colspan="2">Clase a reutilizar:

```
class Hop {
    public void Hop()
    {
        /*Complejo código que genera un salto*/
    }
}
```

</td></tr>
<tr><td>

```
class Bunny {
    private Hop hop;
    public void Hop()
    {
        hop.Hop();
    }
}
```

</td><td>

```
class Bunny:Hop
{
}
```

</td></tr>
<tr><td colspan="2">Cliente de Bunny:

```
class PascuaMelo
{
    private Bunny conejoBlanco;
    /*Constructor apropiado*/
    public void EntregarHuevosPascua()
    {
        /*código que utiliza el conejo y sus movimientos para entregar los huevos*/
    }
}
```

</td></tr>
</table>

4.1- Indica qué mecanismo de reutilización se utiliza en cada caso (A y B).
4.2- Indica cuál de las implementaciones utilizarías y porqué. Ayúdanos a entender tu decisión escribiendo las ventajas y desventajas de cada solución. Enfoca tu respuesta en el impacto que tienen ambas soluciones sobre la clase **PascuaMelo** ante posibles cambios.

Ejercicio 5.- Observa el siguiente código:

```
class Radio
{
    public double frecuencia1 = 90.3;
    public double frecuencia2 = 91.9;
    public double frecuencia3 = 93.9;
    public double frecuencia4 = 91.1;
    public double frecuenciaActual;

    public Radio(double frecuencia)
    {
        frecuenciaActual = 90.3;
    }
    public void SintonizarRadio(int frecuencia)
    {
        switch (frecuencia)
        {
            case(1):
                frecuenciaActual = frecuencia1;
                break;
            case (2):
                frecuenciaActual = frecuencia2;
                break;
            case (3):
                frecuenciaActual = frecuencia3;
                break;
            default:
                frecuenciaActual = frecuencia4;
                break;
        }
    }
    public void Desplegar()
    {/*Complejo código que despliega información de la frecuencia actual en un display*/}

    public void ModificarVolumen(bool subir)
    {/* Complejo código que sube o baja el volúmen de los parlantes */}
}
```

5.1- Este programa contiene errores conceptuales. Encuentra todos los que puedas e indica para cada error cual es el concepto/principio que se viola.

Ejercicio 6.- Basándote en el programa del **ejercicio 5**, te pedimos que:

6.1- Programes una solución a todos los problemas que encontraste.
6.2- Agregues a la solución una precondición, una post condición y una invariante utilizando **Debug.Assert(bool condition)**.
6.3- Agregues lo que precises, tal que la ejecución de las sentencias que escribas provoquen la generación de: dos variables de diferente tipo que referencien al mismo objeto y dos objetos iguales entre sí.