Programación Orientada a Objetos. Primer Parcial 1er semestre 2012.

Miércoles 25 de abril de 2012

1.- Sean las siguientes clases e interfaces, y las siguientes asignaciones:

```
public interface IPersonaje {
    String getNombre();
}
public interface ICienciaFiccion {
}
public interface ISuperHeroe:ICienciaFiccion, IPersonaje{
    String getPoder();
    void generarPoder();
    void combatirAlEnemigo();
}
public interface ISuperVillano:ICienciaFiccion, IPersonaje{
    String getPoder();
    void generarPoder();
}
public interface IPelicula {
    void Reproducir();
    void Nombre();
    List Actores();
}
public class PeliculaCF: IPelicula, ICienciaFiccion {}
public class SHVolador: ISuperHeroes {
    void Volar();
}
public class SHTerrestre: ISuperHeroes {
    void Correr();
    void Trepar();
}
public class PersonajeHistorieta: IPersonaje, ICienciaFiccion {
    void ContarChiste();
}
public class SVVolador: ISuperVillano {
    void Volar();
}
public class SVTerrestre: ISuperVillano {
    void Correr();
}
public class Mafalda: IPeersonaje, IPelicula {}

class Program
{
    static void Main(string[] args)
    {
        /1/ ISuperHeroe b = new SHVolador();
        /2/ ISuperVillano v = b;
        /3/ IPersonaje m = new Mafalda();
        /4/ IPelicula p = m;
        /5/ PersonajeHistorieta ph = new IPersonaje();
        /6/ ISuperVillano sv = new SHTerrestre();
        /7/ ICienciaFiccion c = new PeliculaCF();
        /8/ PersonajeHistorieta h = c;
    }
}
```

1.1 Indique las líneas de código incorrectas y justifique muy brevemente.

2.- Responde las siguientes preguntas basándote en el ejercicio anterior:
2.1 - ¿De qué tipos son los objetos creados en las líneas 3 y 6?
2.2 - ¿Que mensajes puede recibir un objeto cómo el creado en la línea 7? ¿Cómo lo sabes?
2.3 - ¿Que mensajes puede recibir un objeto bajo los términos de la variables declaradas en la líneas 1 y 8? ¿Cómo lo sabes?

3.- Considerando las clases e interfaces del ejercicio 1, te pedimos que realices lo siguiente:
3.1 - Elige las dos clases que más quieras y agrega a cada una 2 variables de instancia.
3.2 - Encapsula ambas variables de instancia, creando propiedades.
3.3 - Realiza los cambios necesarios sobre una de las clases, para que sus instancias sean objetos inmutables.
3.4 - Utilizando las clases que modificaste, programa un fragmento de código en donde se tenga al final del mismo:
a) 2 objetos iguales. Indica el nombre de las variables que los referencian.
b) 3 variables que referencien a objetos idénticos. Indica el nombre de las variables que los referencian.

4.- Observa el siguiente código:

```
//Interfaz que representa una instrucción de procesador
interface IInstruccion
{
    String Nombre { get; }
    Object Ejecutar(Object paramA, Object paramB);
}

//Interfaz que representa el resultado de ejecutar una instrucción
interface IResultado
{
    String Mensaje();
}

class Procesador
{
    public IResultado Procesar(IInstruccion instruccion)
    {
        /* Procesa la instrucción y devuelve el resultado de la ejecución */
    }
}
```

4.1 - Imagina una implementación de IResultado que representa siempre una ejecución correcta y devuelve el mensaje "Ejecución correcta". ¿Consideras que esta clase deba ser inmutable? ¿Por qué?
4.2 - Imagina una implementación de IInstruccion que representa una instrucción que suma dos enteros. ¿Consideras que esta clase deba ser inmutable? ¿Por qué?
4.3 – Implementa ambas clases (una clase que implemente IResultado y otra clase que implemente IInstruccion).
4.4 – La clase que implementa IInstruccion que creaste, ¿cumple con LSP? Justifica.

5.- Sea el siguiente esqueleto de código:
5.1 - Complétalo para que un ciclista haga funcionar su bicicleta y que un skater haga andar su patineta. Considera que el único modo de hacer funcionar a ambos transportes es que hagan girar sus ruedas. Todas las ruedas giran de la misma manera. *Es irrelevante para el ejercicio la manera de girar de las ruedas, lo que importa es que sepan girar.*

```
//Completar
interface IRueda
{
    void Girar();
}

class Rueda
{
    //Completar
}
```

```
class Skate: ITransporte
{
  // Completar
   public void Avanzar()
   {
       // Completar
   }
}
class Bicicleta: ITransporte
{
   // Completar
   public void Avanzar()
   {
     // Completar
   }
}
```

```
class Ciclista
{
    // Completar
}
class Skater
{
    //Completar
}
```

5.2 – Agrega las líneas necesarias al Main del program para que efectivamente un ciclista y un skater, anden en sus respectivos vehículos. Fomenta la colaboración por medio del envió de mensajes entre los objetos.

```
class Program {
     static void Main(string[] args)
     {
         // Completar
     }
}
```

5.3 – Agrega al modelo un deportista, que sea ciclista y skater.

6.- Sea el siguiente código:

```
class Alumno {
    private String nombreCompleto;
    public String NombreCompleto {get;set;}
    private String cI; { get; set; }
    public String CI {get;set;}
}
class Clase {
    private ArrayList alumnos;
    private String carrera;
    public String Carrera { get; set; }
    private String nombreAsignatura
    public String NombreAsignatura { get; set; }
    private String semestre
    public String Semestre { get; set; }
    void RegistrarAlumno(Alumno a) {
        //Agrega un alumno a la materia
    }
    bool BorrarAlumno(Alumno a) {
        //Si existe, borra al alumno a la materia
        return true;
    }
}
class Pregunta
{
    private id;
    private String texto;
    public String Texto{ get; set; }
}
```

```
class GeneradorPrueba {
    private ArrayList cienPreguntas;
    private String GenerarEncabezado(Clase c) {
        return c.Carrera + "" + c.NombreAsignatura + " " + c.Semestre;
    }
    private String GenerarPreguntas(Int16 cantidad) {
        String cuestionario= "";
        Random rnd = new Random();
        for (int i = 0; i > cantidad ;i++)
        {
            cuestionario += ((Pregunta)cienPreguntas[(rnd.Next(100))]).Texto;
        }
        return cuestionario;
    }
    private String GenerarPiePrueba() {
        return "* Numere las hojas" + "Universidad Católica del Uruguay";
    }
    public String GenerarPrueba(Clase c, Int16 cantidad) {
        return GenerarEncabezado(c)+ GenerarPreguntas(cantidad)+ GenerarPie();
    }
}
```

6.1 – Explica qué entiendes por patrón experto. Indica si el mismo se cumple. En caso de no cumplimiento brinda una breve explicación y referencia el lugar en el código donde se provoque la violación.
6.2 – Explica qué entiendes por el principio Abierto-Cerrado (OCP). En caso de no cumplimiento brinda una breve explicación y referencia el lugar en el código donde se provoque la violación.
6.3 – Explica qué entiendes pro el principio de responsabilidad única (SRP). En caso de no cumplimiento brinda una breve explicación y referencia el lugar en el código donde se provoque la violación.

7.- Según lo que respondiste en el ejercicio anterior, te pedimos que realices lo siguiente:
7.1 - Para los patrones o principios que encontraste que se violaban, desarrolla una solución para que se cumplan.
7.2 - Para los patrones o principios que encontraste que se cumplían, programa en alguna clase, los cambios necesarios para que se violen. Debes programar los cambios.
* En ambos casos debes programar la respuesta.

Consideraciones Generales:

- Pon nombre en todas las hojas
- Numera las hojas
- Puedes escribir de los dos lados de la hoja
- Puedes escribir en la letra del parcial. En ese caso entrégala y ponle nombre.
- ¡SUERTE!