Programación orientada a objetos. Examen febrero 2010.

3 de febrero de 2010.

1. Sean las siguientes clases e interfaces:

```
interface ICiencia {
    void GenerarConocimiento();
}
interface ITerapiaAlternativa {
    void TratarSerVivo();
}
class Reiki : ITerapiaAlternativa {
  // implementacion correcta
}
abstract class CienciaExacta : ICiencia {
    // implementacion completa y correcta
    abstract void Comprobar();
}
interface IPseudoCiencia: ICiencia, ITerapiaAlternativa {
    // implementacion completa y correcta
    void Refutar();
}
class Matematicas: CienciaExacta {
    // implementacion completa y correcta
}
class Homeopatia: IPseudoCiencia {
    // implementacion completa y correcta
     List<String> Substancias();
}
class FengShui : IPseudoCiencia {
    // implementacion completa y correcta
    public Int16 NivelEnergia { get { return 10; } }
    void EnergizarUniverso() { }
}
class Program {
    void Main() {
        /*1*/ ICiencia algebra = new Matematicas();
        /*2*/ IPseudoCiencia floresBach = algebra;
        /*3*/ ITerapiaAlternativa osteopatia = new FengShui();
        /*4*/ FengShui feng = osteopatia;
        /*5*/ CienciaExacta fisica = new CienciaExacta();
        /*6*/ Homeopatia yuyosVerdes = feng;
        /*7*/ IPseudoCiencia pseudo = yuyosVerdes;
    }
```

1.1 **Indica** las líneas de código incorrectas. Si una asignación es incorrecta, de todas formas asume que la variable fue definida.

1.2 **Indica** los tipos de la *variable* **feng** de la línea 4 y los tipos del *objeto* creados en la línea **1**.

1.3 ¿Qué mensajes puedes recibir un objeto como el creado en la línea 3? ¿Cómo lo sabes? ¿Qué mensajes pueden ser enviados al mismo objeto bajo los términos de la variable que lo referencia en la misma línea?

1.4 Un objeto referenciado por una variable de tipo **IPseudoCiencia** ¿Puede poseer una implementación de **void EnergizarUniverso ()**? En caso afirmativo, provee un ejemplo de esto, en caso negativo, explica por qué.

2. Sea el siguiente código

```
class Clase {
	private Persona profesor;
	Persona Profesor {
		get { return profesor; }
		set { profesor = value; }
	}
	private Int32 comienzo;
	Int32 Comienzo { get { return comienzo; } }
	private Int32 fin;
	Int32 Fin { get { return fin; } }
	private Int32 inscriptos;
	Int32 Inscriptos {
		get { return inscriptos; }
		set { inscriptos = value; }
	}
	public Clase(Int32 comienzo,  Int32 fin) {
	  this. inscriptos = 0;
	  this.comienzo = comienzo;
	  this.fin = fin;
	 }
}
class Persona {}
class Gimnasio {
	void AsignarAClase(Profesor profesor, Clase clase) { }
	void Inscribir(Persona p) { }
	void AsistirAClase(Persona p, Clase clase) { }
	void Borrar(Persona p) { }
	Int32 CantidadPersonas { get { } }
}
```

Y las siguientes condiciones:

a) No se puede asignar el mismo profesor a clases diferentes en el mismo horario.

b) Cuando una persona se inscribe en el gimnasio, lo hace por un año. Una vez que asiste X veces a clases, se la borra del gimnasio y debe volver a inscribirse.

c) Para que una persona pueda asistir a clase debe estar inscripta en el gimnasio.

d) No pueden asistir más personas a una clase que 15.

e) El gimnasio tiene capacidad para 150 personas. Por lo que no se pueden inscribir más de 150 personas.

f) Luego que una persona se inscribe al gimnasio, la cantidad de personas inscriptas aumenta en uno.

g) La hora de comienzo de una clase debe ser anterior a la hora de fin de la clase.

2.1 **Indica** que condiciones corresponden a precondiciones, postcondiciones e invariantes.

2.2 Modifica las clases necesarias para cumplir todas las condiciones mencionadas. Además, declara las precondiciones, postcondiciones e invariantes usando Debug.Assert(bool Condition).

3. Sean las siguientes interface y clases:

```
interface IEnumerable {
  IEnumerador GetEnumerador();
}
class Lista: IEnumerable {
  void Add(Object o) { /* código completo y correcto */ }
  void Insert(Object o, int index) { /* código completo y correcta */ }
  IEnumerador GetEnumerador() { /* código completa y correcta */ }
  Object Get(int index) { /* código completa y correcta */ }
  Boolean Contains(Object o) { /* código completo y correcto */ }
}
interface IConjunto: IEnumerable {
  void Add(Object o);
}
```

Te piden que proveas una implementación de `IConjunto` reutilizando código de la clase lista. Un conjunto es enumerable y no posee elementos repetidos, por lo cual al agregar un elemento, si existe, el elemento no se agrega.

3.1 Implementa `IConjunto` usando composición y delegación con `Lista`.

3.2 Implementa `IConjunto` usando herencia con `Lista`.

3.3 Para un programa real, ¿cuál de los dos mecanismo de reutilización utilizarías y por qué?

4. Sea el siguiente código:

```
public class Expendedor {
       IList contenidos = new ArrayList();
       void Cargar(Object contenido){
              contenidos.Add(contenido);
       }
       Object Expulsar(Object contenido) { return contenidos.get(contenido);}
}
```

4.1 Te piden que apliques algún mecanismo de reutilización para poder utilizar Expendedores que **solo** carguen un tipo de líquidos, por ejemplo Expendedores que **solo** carguen Refrescos o Expendedores que **solo** carguen Yogures. Debes utilizar aquél en el que puedas escribir lo menos posible para reutilizar el código. ¿Cuál utilizarías? Puedes hacer los cambios que creas necesario para poder utilizarlo.

5. Sea el siguiente código, que representa los proyectos de un Banco. Para cada proyecto existen gastos de consultoría y de viajes (misiones):

```
class Proyecto {
        private IList<Consultor> consultores;
        private IList<Mision> misiones;
        Double CalcularPresupuesto(){
            Double monto;
            foreach (Consultor c in consultores) {
                if (c.localidad == "EEUU") {
                    monto = 5350; /* costo de un consultor de ese país */
                } else {
                    monto = 2100; /* costo de un consultor de ese país */
                }
            }
            foreach (Mision m in misiones) {
                if (m.paisDestino == "EEUU") {
                    monto += 350; /* valor del viático en el país */
                } else {
                    monto += 150; /* valor del viático en el país */
                }
                monto += m.valorTicket;
            }
            return monto;
        }
    }
    class Mision {
        public String paisDestino;
        public Double valorTicket;
    }
    class Consultor {
        public String localidad;
    }
```

5.1 Critica el código en base a:
- Encapsulación
- OCP
- SRP
- DIP
- LSP

** Evalúa la posibilidad de tener por proyecto nuevos ítems(además de consultores y misiones) que repercutan en el presupuesto total.*

6. Mejora esta solución y asegúrate que el código que entregues cumpla todos los principios evaluados anteriormente.

7. El siguiente programa fue realizado por un programador para enviar los emails recibidos de `ucu.edu.uy` a una carpeta llamada UCU:

```
interface IMail {
  String De { get; } String A { get; } String Texto { get; }
}
class Carpeta {
  private String nombre;
  public Carpeta(String nombre) { this.nombre = nombre; }
  public void Guardar(String texto) {
     /* guarda en la carpeta de nombre 'nombre' */ }
  }
  class ClienteMail {
    void RecibirMails(String servidor) {
       Carpeta carpeta = new Carpeta("UCU");
       foreach (IMail mail in LeerMails(servidor)) {
         if (mail.De.Contains("ucu.edu.uy")) {
           carpeta.Guardar(mail.Texto);
         }
       }
    }
    private Ilist<IMail> LeerMails(String servidor) {
       /* baja los mails de servidor y los retorna */
    }
  }
```

Ahora necesita que si el texto del email dice “SPAM” el email sea enviado a la carpeta “SPAM”, y que si es para “jose@trabajo.com” lo envie a “TRABAJO”. Y que si viene de 'loteria.com.uy', y dice “Ganó!” lo mande a imprimir y le avise por SMS.

Programa los cambios necesarios en el programa para permitir realizar estas tareas y otras sin necesidad de cambiar `ClienteMail`, `Carpeta` ni `IMail` de ahí en más. Puedes modificar lo que quieras para hacer éstos cambios, pero respeta la idea general.

Usa las clases o interfaces creadas para proveer la funcionalidad existente (guardar en “UCU” cuando el origen es `ucu.edu.uy`).