Programación orientada a objetos. Segundo parcial 1º semestre 2008.

Miércoles 11 de junio de 2007.

Al igual que las demás instancias de evaluación este examen será calificado de acuerdo a la escala de letras vigente: D, R, B, MB, BMB, S o D, R, B, MB, S según la generación de cada alumno. Por eso las preguntas no tienen puntos.

Las respuestas deben aparecer en el orden en que están formuladas las preguntas.
Numeren las hojas.
Gracias y mucha suerte.

1.

```
class PilaEnteros {
   private IList elementos = new ArrayList();
   Int32 Tamaño() {
      return elementos.Count;
   }
   Int32 Tope() {
      return (Int32) elementos[0];
   }
   void Push(Int32 o) {
      elementos.Insert(0, o);
   }
   Int32 Pop() {
      Int32 i = (Int32) elementos[0];
      elementos.RemoveAt(0);
      return i;
   }
}
```

Y las siguientes condiciones:

a) La pila contiene cero o más elementos.
b) No se pueden agregar a la pila elementos nulos.
c) No se pueden quitar elementos cuando la pila está vacía.
d) No se puede consultar el tope de la pila cuando está vacía.
e) Cuando se agrega un elemento, el tamaño de la pila aumenta en uno y el tope de la pila corresponde al elemento agregado..

1.1. Identifica que condiciones corresponden a precondiciones, postcondiciones o invariantes.
1.2. Utilizando Debug.Assert(), agrega el código necesario para verificar esas precondiciones, postcondiciones e invariantes.
1.3 ¿Quién es el culpable de que se viole una precondición? ¿Y una postcondición?.

2.
Utilizando la clase del ejercicio anterior, programa una PilaEnterosSinRepetidos que no permita agregar elementos repetidos. Para hacerlo, utiliza:

2.1. Composición y delegación
2.2. Herencia

3.
Supone que quieres programar pilas para varios tipos de datos. ¿Qué mecanismo de reutilización utilizarías? ¿Por qué? Programa la clase que reutilizarías y provee un ejemplo donde la reutilices. Puedes utilizar como base la clase del ejercicio 1, si explicas claramente qué modificaciones le harías

4.

El principio de inversión de dependencias (DIP), entre otras virtudes, aumenta el grado de reutilización de una clase. Explica brevemente por qué sucede ésto mediante un ejemplo donde apliques el DIP.

5.

```
interface IForma { /* mucho código */ }
interface ICosa { /* mucho más código */ }
interface IAccion {
   void Actuar();
}
interface IEjecutor {
   void Ejecutar(IAccion a);
}
interface IMundo {
   void PonerCosa(Int32 x, Int32 y, ICosa cosa);
   IForma GetForma();
   void ActuarUnCiclo();
}
class PasarCiclo : IAccion {
   private IMundo mundo;
   public PasarCiclo(IMundo mundo) {
      this.mundo = mundo;
   }
   public void Actuar() {
      mundo.ActuarUnCiclo();
   }
}
class PintorMundo {
   void PintarMundo(IMundo mundo) {
      PintarForma(mundo.GetForma());
   }
   void PintarForma(IForma forma) {
      /* insuperable cantidad de código */
   }
}
```

El código anterior no cumple el principio de segregación de interfaces.
5.1. ¿Por qué? Programa el código nuevamente explicando que cambios harías para que lo cumpla.
5.2. Compara ambas versiones respecto a reutilización.

6.

6.1 Provee un ejemplo donde el resultado de la ejecución del código es distinto si se utiliza encadenamiento estático que dinámico.
6.2 Dado un lenguaje en el que no se definen los tipos de las variables, ¿qué tipo de encadenamientos puedo usar en ese lenguaje? ¿Por qué?

7.

Sea el siguiente código

```
class SensorTemperatura {
   public void Medir() {
        /* complejo y extenso código de inicialización del medidor */
        while (true) {
                Double temperatura = MedirTemperatura();
                Console.WriteLine("La temperatura es: " + temperatura);
        }
   }
   private Double MedirTemperatura() { /* codigo correcto, completo y muuuuuy largo */ }
}
```

Te piden que **modifiques** la clase SensorTemperatura para que puedas **reutilizarla** para realizar distintas tareas cuándo cambia la temperatura. Además, te piden que **programes** las siguientes tareas específicas:

a) Imprimir por consola un mensaje cuando la temperatura supera un cierto valor.

b) Imprimir por consola un mensaje cuando la temperatura baja de cierto valor.

Pueden existir programas que realicen la tarea a) pero no la b), que realicen la b) pero no la a) o que realicen la a) y la b). Se quiere dejar al programa la opción de decidirlo en **tiempo de ejecución**.

A tu jefe se le ocurrieron cientos de tareas interesantes además de éstas, como guardar en una base de datos todos los cambios de temperatura, mantener un promedio, mandar un mail cuando éste supera cierto límite, etc. No te pide que las programes, pero si que tu programa esté abierto a poder hacerlo en el futuro, sin necesidad de modificar nada de lo que programaste.

Tu programa debe incluir: la versión modificada de SensorTemperatura, las clases que tienen el código de las tareas a) y b) y cualquier otra cosa que necesites para que todo compile y funcione.