Programación orientada a objetos. Examen febrero 2008.

Lunes 11 de febrero de 2008.

Sean las siguientes clases e interfaces:

```
interface Component {
   Int32 X { get; }      Int32 Y { get; }
}
interface Disposable {
   void Dispose();
}
class Control: Component, Disposable {
   /* implementacion completa y correcta */
   public Boolean Visible { get { /* ... */ } set { /* ... */ } }
}
interface HtmlElement {
   String Name { get; }
}
abstract class HtmlControl: Control, HtmlElement {
   /* implementacion completa y correcta */
   public abstract void Render();
}
interface InputControl { }
class HtmlInput: HtmlControl, InputControl {
   /* implementacion completa y correcta */
   public void Clear() { /* ... */ }
}
class HtmlLink: HtmlControl {
   /* implementacion completa y correcta */
   public void GoToURL() { /* ... */ }
}
class Program {
   public static void Main(String[] args) {
      /*1*/ Component a = new InputControl();
      /*2*/ Control b = new HtmlControl();
      /*3*/ HtmlControl c = b;
      /*4*/ HtmlElement d = new HtmlLink();
      /*5*/ Disposable e = d;
      /*6*/ HtmlInput f = d;
      /*7*/ HtmlInput g = new HtmlLink();
   }
}
```

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 **d** y los tipos del objeto creado en la línea **7**.

1.3 Sea el siguiente código:

```
Control a = new Control();
Control b = a;
a.Visible = true;
b.Visible = ! a.Visible;
```

¿Cuál es el valor de **b.Visible** al final del código? ¿Cuál es el valor de **a.Visible** al final del código?.

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

El siguiente código se utiliza para controlar un cajero automático (¡por suerte solo en este ejemplo!).

```
interface Cuenta {
   Double Saldo { get; }
}
interface Tarjeta {
   Cuenta Cuenta { get; }
}
class Cajero {
   /* completar */
   Boolean TarjetaIngresada { get { { /* completar */ } }
   Double DineroEnCajero() { get { { /* completar */ } }
   void InsertarTarjeta(Tarjeta tarjeta) { /* completar */ }
   void Depositar(Double importe) { /* completar */ }
   void Retirar(Double importe) { /* completar */ }
   Double SaldoTarjeta() { /* completar */ }
   void QuitarTarjeta() { /* completar */ }
}
```

La clase **Cajero** debe cumplir las siguientes condiciones:

a) Se puede insertar una tarjeta solo cuando no hay otra insertada.
b) Luego de insertar la tarjeta se puede depositar o retirar dinero, consultar el saldo o quitar la tarjeta; antes de insertarla no.
c) Si se realiza un deposito, el saldo de la tarjeta y el dinero en el cajero aumentan tanto como el importe depositado.
d) Si se realiza un retiro, el saldo de la tarjeta y el dinero en el cajero se decrementa tanto como el importe retirado.
e) No se puede retirar más dinero que el disponible en la cuenta de la tarjeta o en el cajero.
f) El dinero en el cajero nunca es negativo.

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

2.2 Modifica y programa correctamente la clase **Cajero** para cumplir todas las condiciones mencionadas. Declara las precondiciones, postcondiciones e invariantes usando **Debug.Assert(bool Condition)**.

Sea el siguiente código

```
class Lamparita {
   public void Prender() { /* implementacion correcta */}
   public void Apagar() { /* implementacion correcta */ }
}
class Boton {
   private Lamparita l;
   private Boolean presionado;
   public Boton(Lamparita l) {
      this.l = l;
   }
   void Apretar() {
      if (presionado) {
         l.Apagar();
         presionado = false;
      } else {
         l.Prender();
         presionado = true;
      }
   }
}
```

3.1 Crítica el código de acuerdo a criterios de reutilización.

3.2 Programa una nueva implementación tomando en cuenta las críticas realizadas.

Provee un ejemplo de código donde, con objetivos de reutilizar:

4.1 Es más adecuado utilizar composición y delegación que herencia.

4.2 Es más adecuado utilizar herencia que composición y delegación.

En cada caso, justifica porqué es más adecuado uno que el otro. El ejemplo debe incluir la/s clase/s reutilizada/s y la clase que reutiliza.

Sea el siguiente código:

```
class Camion
{
   IList contenidos = new ArrayList();
   void Cargar(Object contenido) { contenidos.Add(contenido); }
   IList Descargar() { return new ArrayList(contenidos); }
}
```

5 Te piden que apliques algun mecanismo de reutilización para poder utilizar Camiones que **solo** carguen un tipo de cosa, por ejemplo Camiones que **solo** carguen Paises o Camiones que **solo** carguen Colores. Debes utilizar aquel en el que puedas escribir lo menos posible para reutilizar el código. ¿Cuál usarías? Haz los cambios necesarios para poder usarlo.

6.1 Existen lenguajes de programación orientados a objetos donde **no** es necesario declarar el tipo de las variables. ¿Es posible utilizar encadenamiento estático en estos lenguajes? ¿Y dinámico?. Justifica.

6.2 ¿Qué pierdes si no puedes utilizar encadenamiento dinámico? ¿Y si no puedes utilizar estático?.

Sea el siguiente código

```
class SensorTemperatura {
   public void Medir() {
      while (true) {
         Double temperatura = MedirTemperatura();
      }
   }
   private Double MedirTemperatura() { /* codigo correcto y completo */ }
}
```

7 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.