Programación orientada a objetos. Examen julio 2006.

Viernes 12 de julio de 2006.

Al igual que las demás instancias de evaluación este examen será calificado con una nota conceptual de acuerdo a la escala de letras vigente según la generación de cada alumno. Por eso las preguntas no tienen puntos.

Pongan el nombre sólo en la primera hoja. Las respuestas deben aparecer en el orden en que están formuladas las preguntas. Escriban sólo del anverso de la hoja y eviten doblarlas. Gracias y mucha suerte.

Sean las siguientes clases:

```
class X
{
    virtual int M()
        requires A; // Precondicion de M en X
        ensures B; // Poscondición de M en X
    {
        // Código no mostrado
    }
}

class Y: X
{
    override int M()
        requires C; // Precondición de M en Y
        ensures D; // Poscondición de M en Y
    {
        // Código no mostrado
    }
}
```

`A` y `B` son precondición y poscondición, respectivamente, del método `M` de la clase `X`. `C` y `D` son precondición y poscondición, respectivamente, del método `M` sobrescrito en la clase `Y` sucesora de la clase `X`. Consideren el siguiente programa:

```
public class Program
{
    static void Main() {
        X x = new Y();
        x.M();
    }
}
```

Razonen sobre las consecuencias del principio de sustitución y el polimorfismo en relación con el diseño por contrato. El método `M` de la clase `Y` que se ejecuta espera que se cumpla `C` y asegura que se cumple `D` pero en el contrato definido por el tipo `X` se especifica que el método `M` espera que se cumpla `A` y garantiza que se cumple `B`.

Recuerde que una afirmación P es más fuerte que otra Q si P implica Q y P es diferente de Q; y si P es más fuerte que Q, Q es más débil que P.

1. ¿Cuáles de las siguientes afirmaciones son correctas?

    a. `C` puede ser más fuerte que `A`
    b. `C` puede ser igual que `A`
    c. `C` puede ser más débil que `A`

d. `D` puede ser más fuerte que `B`
e. `D` puede ser igual que `B`
f. `D` puede ser más débil que `B`

Un objeto de la clase Y debe poder aparecer en los lugares donde un cliente espere un objeto de la clase X.

El cliente está obligado a satisfacer las precondiciones establecidas por X y ninguna más. Por lo tanto las precondiciones de Y pueden ser iguales o más débiles, pero nunca más fuertes. Si fueran más fuertes, el cliente se encontraría con una sorpresa: a pesar de haber hecho todo lo necesario, el método no puede ejecutarse.

El servidor está obligado a satisfacer por lo menos las poscondiciones establecidas por X pero eventualmente puede satisfacer alguna más. Por lo tanto las poscondiciones de Y pueden ser iguales o más fuertes, pero nunca más débiles. Si fueran más débiles, el cliente se encontraría con una sorpresa: el método no hizo todo lo que debía hacer.

Respuesta: b, c, d, e.

Sean las siguientes clases:

```
public class Edit
{
    private string text;
    public string Text { get { return text; } set { SetText(value); }
}

    protected virtual void SetText(string value)
    {
        text = value;
    }

    public void Underline(string word)
    {
        // Código no mostrado
    }
}

public class SpellChecker
{
    public List<string> Check(string text)
    {
        // Código no mostrado
    }
}
```

La clase `Edit` representa un control de edición de un framework de interfaz de usuario. La propiedad `Text` representa el texto mostrado en el control. Cuando el usuario escribe el framework usa la propiedad `Text` para cambiar el contenido del control. También el programador puede usar la propiedad `Text` para cambiar el contenido del control por código. El método `void Underline(string word)` subraya todas las apariciones de una palabra en el texto mostrado en el control.

La clase `SpellChecker` representa un corrector ortográfico. El método `List<string> Check(string text)` retorna una lista con las palabras erróneas en el texto recibido como parámetro.

2. Terminen de programar la siguiente clase de tres formas: la primera usando herencia simple y composición y delegación, la segunda usando herencia múltiple -asuman por un instante que C# soportara herencia múltiple- y la tercera usando solo composición y delegación. La clase

`SpellCheckedEdit` debe tener por lo menos los mismos métodos que la clase `Edit`. El método `void Check()` debe encontrar las palabras que tienen errores ortográficos y subrayarlas:

```
public class SpellCheckedEdit
{
    public void Check()
    {
        // ...
    }
}
```

Con herencia simple y composición y delegación:

```
public class SpellCheckedEdit : Edit
{
    public void Check()
    {
        SpellChecker s = new SpellChecker();
        List<string> l = s.Check(Text);
        foreach (string w in l)
        {
            Underline(w);
        }
    }
}
```

Con herencia múltiple, si la hubiera:

```
public class SpellCheckedEdit : Edit, SpellChecker
{
    public void Check()
    {
        List<string> l = Check(Text);
        foreach (string w in l)
        {
            Underline(w);
        }
    }
}
```

Con composición y delegación solamente:

```
public class SpellCheckedEdit
{
    private Edit edit;

    public string Text {
        get { return edit.Text; } set { SetText(value); } }

    protected virtual void SetText(string value)
    {
        edit.Text = value;
    }

    public void Underline(string word)
    {
        edit.Underline(word);
    }
```

```
    public void Check()
    {
        SpellChecker s = new SpellChecker();
        List<string> l = s.Check(Text);
        foreach (string w in l)
        {
            Underline(w);
        }
    }
}
```

3. Terminen de programa la siguiente clase usado una de las siguientes tres formas: la primera es herencia simple y composición y delegación, la segunda es herencia múltiple -asuman nuevamente que C# soportara herencia múltiple- y la tercera es composición y delegación. Usen la opción que les dé menos trabajo, es decir, la que requiera escribir la menor cantidad de código. La clase `AutoSpellCheckedEdit` debe tener por lo menos los mismos métodos que la clase `SpellCheckedEdit`:

```
public class AutoSpellCheckedEdit
{
    // ...
}
```

4. Un tipo es un conjunto de operaciones. En C# esos conjuntos pueden ser declarados de dos formas. ¿Cuáles son?

5. ¿Los objetos de la clase `SpellCheckedEdit` tienen el mismo tipo que los de la clase `Edit`? Respondan esta pregunta para cada una de las formas diferentes en las que contestaron la pregunta 2.

6. Supongan que la clase `Edit` de la pregunta 2 anterior fue usada para construir la interfaz de usuario de un programa. En el código de ese programa hay innumerables sentencias de la forma `Edit x = new Edit()` e innumerables métodos de múltiples clases que reciben objetos de la clase `Edit` como parámetro de la forma `MetodoCualquiera(Edit x)`; al objeto contenido o referenciado en la variable o parámetro `x` se le envían todos los mensajes posibles.

   ¿Qué cambios serían necesarios para que en lugar de objetos de la clase `Edit` se usen en algunos casos objetos de la clase `SpellCheckedEdit`? Tengan en cuenta cada una de las formas diferentes en las que contestaron la pregunta 2 anterior.

7. A la luz de la respuesta anterior, examinen las ventajas y desventajas de usar herencia y composición y delegación como mecanismo de extensión.

8. Sean las siguientes clases:

```
public class Uno
{
    // Código no mostrado
}
public class Dos: Uno
{
    // Código no mostrado
}
public class Tres
{
   public void Imprimir(/*¿Parámetro?*/)
   {
    // Código no mostrado
   }
}
```

   Y el siguiente fragmento de código:

```
Uno uno = new Uno();
Dos dos = new Dos();
Tres tres = new Tres();
tres.Imprimir(uno); // 1
tres.Imprimir(dos); // 2
```

a. ¿Cómo se debe definir el parámetro en el método `Imprimir` para que el código anterior no dé error?
b. ¿Y para que dé error en línea marcada con 1 solamente? ¿Y para que dé error tanto en la línea marcada con 1 como la marcada con 2?