Programación orientada a objetos. Segundo parcial 2º semestre 2006.

Viernes 10 de noviembre de 2006.

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.

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:

```
public class Char //...
{
    public Boolean IsCrLf()
    {
        //...
    }
}

public class KeyPressEventArgs : EventArgs
{
    public Char KeyChar { get { //... } set { //... } }
}

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

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

    protected override void KeyPressed(KeyPressEventArgs e)
    {
        //...
    }
}

public class Calculator
{
    public Double Calculate(String expression)
    {
        //...
    }
}
```

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 en el control 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`

`KeyPressed(KeyPressEventArgs e)` es usado por el framework cada vez que el usuario oprime una tecla en el control. La tecla oprimida está disponible en la propiedad `Char KeyChar` del argumento `e` de tipo `KeyPressEventArgs`. Un método interesante de la clase `Char` para lo que viene a continuación es `Boolean IsCrLf()` que permite determinar si el carácter correspondiente a una tecla oprimida es el retorno de carro.

La clase `Calculator` tiene la responsabilidad de calcular el resultado de una expresión aritmética simple. Por ejemplo `Calculate("2+2")` retornaría 4. Si la expresión aritmética no es válida el método levanta una excepción del tipo `ArgumentException`.

1. 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 `CalculatorEdit` debe tener por lo menos los mismos métodos que la clase `Edit`. El método `void Calculate()` debe remplazar el contenido del control por el resultado de la expresión en él contenida si la expresión es válida y no hacer nada en caso contrario:

```
public class CalculatorEdit
{
    public void Calculate()
    {
        //...
    }
}
```

2. Terminen de programar 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. En los controles `AutoCalculatorEdit` el usuario puede oprimir el retorno de carro para reemplazar el contenido del control por el resultado de la expresión en él contenida si la expresión es válida y no hacer nada en caso contrario. Usen la opción que les dé menos trabajo, es decir, la que requiera escribir la menor cantidad de código. La clase `AutoCalculatorEdit` debe tener por lo menos los mismos métodos que la clase `CalculatorEdit`:

```
public class AutoCalculatorEdit
{
    // ...
}
```

3. Supongan que controles de la clase `Edit` de la primera pregunta fueron usados para construir la interfaz de usuario de una aplicación. En esa aplicación hay desperdigadas por todo el código sentencias de la forma `Edit e = new Edit()`. Ahora se desea reemplazar todos los controles de la clase `Edit` por controles de la clase `AutoCalculatorEdit`. ¿Cuál de las tres clases anteriores -la de herencia simple y composición y delegación, la de herencia múltiple, o la de composición y delegación- puede ser usada cambiando solamente `new Edit()` por `new AutoCalculatorEdit()`? Justifiquen la respuesta.
4. Un objeto se manda un mensaje a sí mismo. ¿El método que se ejecuta como consecuencia tiene que estar implementado en la clase de ese objeto? Justifiquen la respuesta de forma breve y concreta.
5. En el **encadenamiento estático** el tipo de un objeto se determina:
    a. A partir la **variable o referencia** al objeto en **tiempo de compilación**.
    b. A partir de la **clase** del objeto en **tiempo de compilación**.
    c. A partir la **variable o referencia** al objeto en **tiempo de ejecución**.
    d. A partir de la **clase** del objeto en **tiempo de ejecución**.

6. En el **encadenamiento dinámico** el tipo de un objeto se determina:
    a. A partir la **variable o referencia** al objeto en **tiempo de compilación**.
    b. A partir de la **clase** del objeto en **tiempo de compilación**.
    c. A partir la **variable o referencia** al objeto en **tiempo de ejecución**.
    d. A partir de la **clase** del objeto en **tiempo de ejecución**.

Sean las siguientes clases:

```
public class Ancestro
{
    public void Método()
    {
        Console.WriteLine("Yo soy el ancestro");
    }
}

public class Sucesor : Ancestro
{
    public void Método()
    {
        Console.WriteLine("El sucesor soy yo");
    }
}

class Programa
{
    static void Main()
    {
        Ancestro b = new Sucesor();
        b.Método();
    }
}
```

7. El programa compila y es correcto. ¿Qué aparece en la consola si el lenguaje usara **encadenamiento estático**? ¿Y si usara **encadenamiento dinámico**? Justifiquen la respuesta.
8. Enuncie el **principio de sustitución de Liskov**. Busque la aplicación del principio en alguno de los ejercicios anteriores e identifique en ese código los términos que aparecen en el principio.