Programación orientada a objetos. Examen febrero 2006.

Jueves 2 de febrero 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 e interfases en C#:

```
interface Persona
{
    string Nombre { get; set; }
}

interface Empresa
{
    int RUC { get; set; }
}

class Cliente
{
    private int numero;
    public int Numero { get { return numero; } set { numero = value; } }
}

class ClienteEmpresa : Cliente, Empresa
{
    private int ruc;
    public int RUC { get { return ruc; } set { ruc = value; } }
}

class ClientePersona : Cliente, Persona
{
    private string nombre;
    public string Nombre { get { return nombre; } set { nombre = value; } }
}

class Programa
{
    static void Main(string[] args)
    {
        Cliente unaEmpresa = new ClienteEmpresa();
        Empresa otraEmpresa = new ClienteEmpresa();
        Cliente unaPersona = new ClientePersona();
        Persona otraPersona = new ClientePersona();
    }
}
```

1. ¿Cuáles son los tipos de los objetos referenciados por las variables `unaEmpresa`, `otraEmpresa`, `unaPersona` y `otraPersona`? Ignoren el tipo `Object` en la enumeración.

   Cambien la declaración de `unaEmpresa` en el método `Main` de la clase `Programa` para que el objeto referenciado por esa variable tenga menos tipos de los que tiene en la declaración original.

2. ¿Cuál es el error en la siguiente línea de código?

   ```
   Cliente unaPersona = new Persona();
   ```

3. ¿Cuál es el error en las siguientes línea de código?

   ```
   Cliente unCliente = new ClientePersona();
   unCliente.RUC = 213011840017;
   ```

4. Una clase cualquiera puede heredar de una clase abstracta o implementar una interfaz. Desde el punto de vista de los tipos ¿qué similitudes y diferencias existen en ambos casos? Programen un pequeño ejemplo en C#, usando las clases provistas anteriormente y modificándolas según sea necesario, para mostrar al menos una similitud y una diferencia.

5. Dadas las siguientes clases, agreguen las que haga falta para que el programa compile. Usen para una de ellas herencia y para la otra composición-delegación. No pueden escribir de nuevo `UnMétodo()`.

```
public class Base
{
    public void UnMétodo()
    {
        Console.WriteLine("Esto se imprimirá en la consola…");
    }
}

public class Programa
{
    static void Main()
    {
        Sucesora s = new Sucesora();
        s.UnMétodo();

        Compuesta c = new Compuesta();
        c.UnMétodo();
    }
}
```

6. Usando las clases del ejercicio anterior hagan la menor cantidad de cambios necesarios para que `Programa` imprima lo mismo en la consola.

```
class Programa
{
    static void Main()
    {
        UnaClase s = new Sucesora();
        s.UnMétodo();

        UnaClase c = new Compuesta();
        c.UnMétodo();
    }
}
```

7. 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();
    }
}
```

El programa compila y es correcto. Al ejecutar la clase `Programa` aparece "Yo soy el ancestro" en la consola. Hagan la menor cantidad de cambios en la menor cantidad de clases para que aparezca "El sucesor soy yo". ¿Qué cambiaría si no pudieran modificar la clase `Programa`?

9. 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. ¿Hay alguna relación entre el principio de sustitución de Liskov y el **diseño por contrato**?

10. Sean las siguientes clases:

```
public class CompactDisc {
    public void playAll() {
        /* … */
    }
}

public class VideoDisc {
    public void playMovie() {
        /* … */
    }
}
```

Los métodos `CompactDisc.playAll()` y `VideoDisc.playMovie()` son semánticamente iguales pero tienen nombres diferentes porque han sido programados en distinto momento y por diferentes personas.

Un fragmento del programa que usa estas clases aparece a continuación:

```
IList s = new ArrayList();
s.Add(new CompactDisc());
/* Otros discos compactos son creados y agregados aquí… */

s.Add(new VideoDisc());
/* Otros videodiscos son creados y agregados aquí… */

foreach (Object o in s) {
    if (o is CompactDisc) {
        ((CompactDisc)o).playAll();
    }
    else if (o is VideoDisc) {
        ((VideoDisc)o).playMovie();
    }
}
```

Ahora su jefe le pone a usted a cargo de este programa y le pregunta si el fragmento está escrito usando buenas prácticas de programación orientada a objetos; en caso de que no sea así le pedirá que lo escriba de nuevo -el jefe no sabe programación orientada a objetos y confía en su respuesta o se hace el que no sabe para evaluar si usted sabe-.

a. ¿Las clases y el fragmento están bien escritos o no? Justifique su respuesta.

b. En caso que no lo esté escriba la versión correcta. Justifique sus decisiones.