# *Programación orientada a objetos. Segundo parcial 2005.*

*Viernes 17 de junio de 2005.*

*Al igual que las demás instancias de evaluación este parcial 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 preguntas 1, 4, 5, 6 y 7 valen la mitad que las demás. 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.*

1. Dos de las tres formas más comunes de reutilización en programación orientada a objetos son **herencia** y **composición-delegación**. Ignorando la semántica de las relaciones de generalización-especialización y exclusivamente desde el punto de vista de la reutilización de código ¿puedo hacer lo mismo con composición-delegación que con herencia? Justifiquen la respuesta de forma breve y concreta.

2. 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 `MétodoReutilizable()`.

```
public class Base
{
    public void MétodoReutilizable()
    {
        Console.WriteLine("Esto es algo útil");
    }
}

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

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

3. 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()
    {
        Tipo s = new Sucesora();
        s.MétodoReutilizable();

        Tipo c = new Compuesta();
        c.MétodoReutilizable();
    }
}
```

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**.
7. ¿Puedo prescindir del **encadenamiento estático** en un lenguaje de programación orientada a objetos? ¿Y del **encadenamiento dinámico**? ¿Qué se pierde en cada caso? Respondan en forma breve y concreta.
8. 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. ¿Qué está mal en el siguiente programa? ¿Cómo lo resolverían? Justifiquen en forma breve y concreta la respuesta.

```
public class Cuadrado
{
    private Int32 lado;
    public Int32 Lado { get { return lado; } set { lado = value; }}
    public Cuadrado(Int32 lado)
    {
        this.lado = lado;
    }
    public virtual Int32 Area()
    {
        return lado * lado;
    }
}

public class Cubo:Cuadrado
{
    public Cubo(Int32 lado):base(lado) {
    }
    public override Int32 Area()
    {
        return Lado * Lado * 6;
    }
}

class Programa
{
    static void Main()
    {
        Cuadrado cuadrado = new Cuadrado(10);
        Console.WriteLine(cuadrado.Area());

        Cuadrado cubo = new Cubo(10);
        Console.WriteLine(cubo.Area());
    }
}
```

10. 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?