# *Programación orientada a objetos. Primer parcial 2º semestre 2008.*

*Jueves 4 de setiembre de 2008.*

*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.*

*Las respuestas deben aparecer en el orden en que están formuladas las preguntas.*
*Numeren las hojas.*
*Gracias y mucha suerte.*

Sean las siguientes clases e interfaces y el código que las usa:

```
public interface IIdea {}
public interface ICientifico {
   IIdea Pensar();
}
public interface IIngles {
   String HinchaDe { get; set; };
   void TomarTe();
}
public interface IProgramador {
   void Programar();
}
public class Hooligan : IIngles {
   /* código necesario para que compile */
   public void RomperCosas() { /* código violento */ }
}
public class Programmer: ICientifico, IProgramador, IIngles {
   /* código necesario para que compile */
   public void RomperCodigoNazi() { /* complejo código */ }
}
public class Scientist : ICientifico, IIngles {
   /* código necesario para que compile */
   public void PublicarPaper() { /* trabajoso código */ }
}
public class EjercicioUno {
   public static void Main(String[] args) {
      /*1*/ IProgramador turing = new Programmer();
      /*2*/ ICientifico newton = new Scientist();
      /*3*/ IIngles henry = new IIngles();
      /*4*/ Programmer alan = turing;
      /*5*/ IProgramador bob = new Scientist();
      /*6*/ Hooligan tony = henry;
      /*7*/ IIngles isaac = newton;
      /*8*/ Scientist ringo = new Programmer();
   }
}
```

1.1. Indica que líneas **no** compilan y justifica por qué. Deben validar línea por línea. Si una asignación es incorrecta, deben asumir que la variable de todas formas fue definida.

1.2. La siguiente asignación ocurre dentro de un método declarado en la clase de un objeto **O**:

```
IIngles john = new Hooligan();
```

¿Qué mensajes puedo enviar al objeto referenciado por la variable **john** desde el objeto **O**? ¿Cómo lo sabes?

Tomando en cuenta las interfaces y clases definidas en el ejercicio 1

2.1. Indica los tipos de la variable **alan** y los del objeto creado en la línea **5**.

2.2. La siguiente asignación no es válida. ¿Qué cambios deberías realizar para que lo sea? No puedes cambiar la asignación.

```
IProgramador paul = new Hooligan();
```

Sea el siguiente código:

```
IIngles turing = new Programmer();
IIngles beckham = turing;
IIngles george = new Programmer();
turing.HinchaDe = "Arsenal";
beckham.HinchaDe = "Manchester";
george.HinchaDe = "Arsenal";
```

Asumiendo que el único estado de los objetos es de que cuadro son hinchas,

3.1. ¿Qué variables referencian a objetos iguales al final del código?

3.2. ¿Qué variables referencian al mismo objeto al final del código?

3.3. Si agrego la siguiente línea al final del código:

```
george.HinchaDe = "Manchester";
```

¿Cuántos objetos iguales existen al final del código?

3.4. ¿Son los objetos instancia de `Programmer` inmutables? ¿Por qué?

4.1. Provee un ejemplo de código de dos objetos de distinta clase pero con tipos en común.

4.2. Implementa un método que cumpla el LSP en una tercer clase utilizando el tipo en común. Justifica por qué tu código cumple el LSP.

5.1 ¿Qué te permiten hacer las interfaces que no te permiten hacer las clases? ¿Que te permiten hacer las clases que no te permiten hacer las interfaces?.

Sea el siguiente código:

```
public interface Competidor {
   public Int32 Velocidad { get; }
   public Int32 Posicion { get; set; }
}
public class Aquiles: Competidor {
   public Int32 Velocidad { get { return 1000; } } /* código necesario para que compile */
}
public class Tortuga: Competidor {
   public Int32 Velocidad { get { return 1; } } /* código necesario para que compile */
}
public class Relator {
   public void Relatar(Competidor c) {
      if (c.Velocidad == 1000) {
         Console.WriteLine("¡Aquiles se asoma por la punta!");
      } else {
         Console.WriteLine("¡La tortuga avanza paso cansino!");
      }
      Console.WriteLine("¡Que carrera señores que carrera!");
   }
}
public class Carrera {
   private Relator relator = new Relator();
   public void Competir(Competidor c1, Competidor c2) {
      while (true) {
         c1.Posicion = c1.Posicion + c1.Velocidad;
         relator.Relatar(c1);
         c2.Posicion = c2.Posicion + c2.Velocidad;
         relator.Relatar(c2);
      }
```

```
    }
  }
```

Crítica el código en base a:

6.1 Patrón experto
6.2 OCP
6.3 LSP

7 Programa los cambios necesarios en el código anterior para que se cumplan adecuadamente todos los criterios. **Ignora para este ejercicio la clase Relator.**

8 Sean las siguientes clases e interfaces:

```
public interface Nodo {
  /* Hijos retorna una lista de Nodos hijos */
  ArrayList Hijos { get; }
  String Nombre { get; }
}
public class Directorio {
  private ArrayList subdirectorios = new ArrayList();
  public ArrayList Subdirectorios { get { return subdirectorios; } }
  private String nombreRelativo;
  public Directorio NombreRelativo { get { return nombreRelativo; } }
  private Directorio padre;
  public Directorio Padre { get { return padre; } }
  public Directorio(Directorio padre, String nombreRelativo) {
    this.padre = padre;
    this.nombreRelativo = nombreRelativo;
  }
}
public class Arbol {
  public Arbol(Nodo raiz) {
    /* código de P2 */
  }
}
```

Programa los cambios necesarios para que Arbol pueda usar un Directorio como raíz. **Puedes únicamente agregar el código que falta e indicar claramente que cambios harías sobre el código existente.**