Programación orientada a objetos. Examen julio 2009.
Lunes 13 de julio de 2009.

Sean las siguientes clases e interfaces:

```
interface Infusion {
    void Calentar();
}
interface Liquido {
     void Burbujear();
}
interface Adictivo { }
interface Italiano {
     void AndarEnGondola();
}
abstract class Colombiano {
     abstract public void PlantarCafe();
}
class Te: Liquido, Infusion, Adictivo {
    /* Implementacion completa */
}
class Cafe: Colombiano, Liquido, Infusion {
    /* Implementacion completa */
    void AgregarAzucar(Int32 cuantas) {
        /* ... */
    }
}
class Tiramisu: Cafe, Italiano { }
class Program {
     static void Main() {
        /*1*/ Italiano espresso = new Cafe();
        /*2*/ Cafe postre = new Tiramisu();
        /*3*/ Italiano panaCotta = postre;
        /*4*/ Infusion negra = new Te();
        /*5*/ Liquido negro = negra;
        /*6*/ Colombiano Higuita = new Tiramisu();
        /*7*/ Liquido zabaglione = postre;
    }
}
```

1.1 Indica las líneas de código incorrectas. Si una asignación es incorrecta, asume que la variable fue definida.
1.2 Indica los tipos del objeto creado en la línea 2.
1.3 Sea el siguiente código:

```
/*1*/ Colombiano c = new Cafe();
/*2*/ ...
```

¿Qué mensajes puedes enviar a un objeto referenciado por la variable c en la línea 2?

El siguiente código se utiliza para una importante multinacional:

```
class Trabajador {
    Int32 Sueldo { get { /* ... */ } }
    void TomarCafe() { }
    void Trabajar() { }
    void Cobrar(Int32 ingreso) { }
```

```
    }
    class Empresa {
        void Contratar(Trabajador t) { }
        void Despedir(Trabajador t) { }
        void PagarSueldos() { }
        void Facturar(Int32 ingreso) { }
    }
```

El código debe cumplir con las siguientes condiciones:.
a) Un trabajador no puede trabajar sin tener su dosis de café.
b) Una vez que trabajó, el trabajador agota su dosis de café.
c) Un trabajador no puede cobrar su sueldo sin haber trabajado.
d) Una empresa no puede contratar dos veces al mismo trabajador.
e) Una empresa nunca tiene más trabajadores que los que puede pagar.
g) Luego de pagar el sueldo a un trabajador, la empresa lo despide (¡son contratados por proyecto!).

2.1 Indica qué condiciones corresponden a precondiciones, postcondiciones e invariantes.
2.2 Modifica el código provisto para cumplir todas las condiciones mencionadas. Declara las precondiciones, postcondiciones e invariantes usando `Debug.Assert(bool Condition)`. Puedes completar en los lugares indicados y agregar clases adicionales si lo necesitas.

3.1 Supón que tienes un lenguaje con soporte para herencia pero no para encadenamiento dinámico. ¿Qué pierdes respecto a herencia en comparación a un lenguaje que si lo soporte?
3.2 Supón que tienes un lenguaje con soporte para composición y delegación pero no para encadenamiento dinámico. ¿Qué pierdes respecto a composición y delegación en comparación a un lenguaje que si lo soporte?

Sea el siguiente código:

```
    interface Producto { }
    class Manzana: Producto {
        private readonly Double precioKg;
        public Double PrecioKg { get { return precioKg; } }
        private readonly Double peso;
        public Double Peso { get { return peso; } }
        public Manzana(Double precioKg, Double peso) {
            this.precioKg = precioKg;
            this.peso = peso;
        }
    }
    class Aceite: Producto {
        private readonly Double precioLt;
        public Double PrecioLt { get { return precioLt; } }
        private readonly Double volumen;
        public Double Volumen { get { return volumen; } }
        public Aceite(Double precioLt, Double volumen) {
            this.precioLt = precioKg;
            this.volumen = volumen;
        }
    }
    class CalculadorPrecio {
        Double CalcularPrecio(Producto prod) {
            Double p = 0;
            if (prod is Manzana) {
                p = ((Manzana)prod).PrecioKg *
                        ((Manzana)prod).Peso * IVA * COFIS
* IMPUESTO_A_MANZANAS - OFERTA_DE_JULIO;
```

```
            } else if (prod is Aceite) {
                p = ((Aceite)prod).PrecioLt *
((Aceite)prod).Volumen * IVA * COFIS * IMPUESTO_AL_AGUA;
            } else {
                return -1;
            }
        }
    }
```

4.1 El código anterior compila y es correcto (una vez definidas las constantes declaradas en mayúsculas). ¿Qué principios no se cumplen y por qué?.
4.2 Programa el código nuevamente para que los cumpla.

Sea el siguiente código:

```
    public class Buscador {
        Object Buscar(Object aguja, Object[] pajar) {
            /* Complejo algoritmo de busqueda */
        }
    }
    public class Conjunto {
        private Object[] items;
        private Buscador buscador = new Buscador();
        public void Agregar(Object o) {
            if (buscador.Buscar(o, items) == null) {
                /* agregar item */
            }
        }
    }
```

5.1 Te piden que programes un Conjunto que solo guarde cadenas de caracteres (String). Sabes que no será fácil reutilizar el código como está propuesto. Modificalo para que pueda ser reutilizado para programar este nuevo Conjunto y muestra como reutilizarías esas clases para definir este nuevo tiepo de Conjunto.

Sea el siguiente código:

```
    public class Socket {
        public virtual IEscritor GetEscritor() {
            return EscritorSocket(this);
        }
    }
    class EscritorSocket : IEscritor {
        private readonly Socket socket;
        public EscritorSocket(Socket s) {
            this.socket = socket;
        }
        public virtual void write(Byte b) {
            /* escribe el byte a través de un Socket*/
        }
    }
    public interface IEscritor {
        void write(Byte b);
    }
```

6.1 Describe las ventajas de usar composición y delegación frente a herencia.
6.2 Te piden que proveas un nuevo tipo de `Socket` cuyo escritor solo escriba los bytes 1 usando las clases anteriores. No puedes modificar ninguna de las clases dadas. Puedes usar cualquiera de los mecanismos de reutilización vistos.