# *Programación orientada a objetos. Examen diciembre 2011*

*16 de diciembre de 2011.*

**1.** Sean las siguientes clases e interfaces:

```
interface IFiesta {
    void Celebrar();
}
interface IComidaTipica {
    DateTime Vencer();
}
class PanDulce : IComidaTipica {
       // implementacion correcta
}
abstract class AnioNuevo : IFiesta {
    // implementacion completa y correcta
    abstract String AnioActual();
}
class PeleaGallos: IFiesta, IComidaTipica {
    // implementacion completa y correcta
    void Pelear();
}
class Navidad: IFiesta {
    // implementacion completa y correcta
       void ComerUvas();
}
class AnioNuevoChino: AnioNuevo {
    // implementacion completa y correcta
       void ComerArroz();
}
class PanNuez : PanDulce {
    // implementacion completa y correcta
    private Int32 gramosNueces { get; set; }
}
class Program {
    static void Main(string[] args)
    {
        /*1*/ PeleaGallos gallo = new PanNuez();
        /*2*/ IComidaTipica comida = gallo;
        /*3*/ AnioNuevo anio = new AnioNuevo();
        /*4*/ AnioNuevoChino anioChino = anio;
        /*5*/ IFiesta fiesta = gallo;
        /*6*/ PanDulce pan = new PanDulce();
        /*7*/ IFiesta fiestaPanera = pan;
        /*8*/ PanNuez panNuez = pan;
    }
}
```

1.1 **Indica** las líneas de código incorrectas. Si una asignación es incorrecta, de todas formas asume que la variable fue definida.

1.2 **Indica** los tipos de la variable **pan** de la línea 6 y los tipos del *objeto* creado en la línea **1**.

1.3 ¿Qué mensajes puedes recibir un objeto como el creado en la línea 3? ¿Cómo lo sabes?

1.4 ¿Qué mensajes pueden ser enviados al mismo objeto bajo los términos de la variable que lo referencia en la misma línea (3)?

**2.** Sea el siguiente código

```
class Blockster {
    private ArrayList peliculas;
    private ArrayList socios;
    private String encargado;
    public Blockster(ArrayList peliculas, ArrayList socios) {/*implementar*/}
    void AgregarPelicula(Pelicula pelicula) {/*implementar*/}
    void BorrarPelicla(Pelicula pelicul) {/*implementar*/}
    void InscribirSocio(Persona persona) {/*implementar*/}
    void BorrarSocio(Persona persona) {/*implementar*/}
    void AlquilarPeliculaASocio(Pelicula pelicula, Persona socio)
    {/*implementar*/}
}
class Pelicula {
       private String titulo;
       private String genero;
       //completar
}
class Persona {
    private Int32 edad;
    private String cedulaIdentidad;
    //completar
}
```

Y las siguientes condiciones:
a) En Blockster no se puede alquilar una película apta para mayores, a un socio menor de 18.
b) El video “Blockster“ no puede operar sin encargado.
c) Cuando se agrega un socio, el número de socios crece.
d) Un socio no puede alquilar una película si no está disponible
e) Cuando se agrega una película, esta pasa a formar parte del catálogo.
f) Para borrar un socio, este debe estar registrado.
g) Al alquilar una película, la facturación total del video sube en $80

2.1 **Indica** qué condiciones corresponden a precondiciones, postcondiciones e invariantes.
2.2 Modifica las clases necesarias para cumplir todas las condiciones mencionadas, usando Debug.Assert(bool Condition). Debes agregar toda la lógica que sea necesaria, para que las condiciones que agregues estén cubiertas por tu código.

**3.** Sea el siguiente código:

```
interface IExplosivo
{
    Chispas Explotar();
}
class Bomba : IExplosivo
{
    public Int32 numeroExplosiones;
    private String color;
    void CargarPolvora(Int32 gramos) { /* código completo y correcto */ }
    Chispas Explotar() {
        Chispas chispas;
        /* código completo y correcto */
        return chispas;
    }
}
class Chispas {
    /* código completo y correcto */
}
```

Agrega lo siguiente al código anterior (debes programar lo requerido):
3.1 Define un constructor para la clase Bomba.
3.2 Encapsula sus atributos, si es que no están debidamente encapsulados.
3.3 Agrega líneas de código para que se generen dos variables que apunten a objetos idénticos y tres variables que referencien a objetos iguales (no identicos).

3.4 ¿Es posible conocer si dos objetos son iguales sin conocer cuál es la clase concreta que se utilizó para crearlos (conociendo simplemente las interfaces implementadas por sus clases)? ¿Es posible bajo estas circunstancias decir que son idénticos?
3.5 Agrega una línea de código que se encadene dinámicamente y otra que se encadene estáticamente. Puedes utilizar lo que creaste en la parte 3.3.

**4.** Sea el siguiente código:

```
class Nuera {
        private Hashtable memoria;
        String UbicacionMantel () {
            return memoria["mantel"].ToString();
        }
        String UbicacionCopas () {
            return memoria["copas"].ToString();
        }
        String UbicacionPlatos () {
            return memoria["platos"].ToString();
        }
        String UbicacionCubiertos () {
            return memoria["cubiertos"].ToString();
        }
        String UbicacionMesaFestiva () {
            return memoria["mesa"].ToString();
        }
        void LavarVajilla() {//implementacion completa y correcta}
}
class Suegra {
       void PonerMesa(Nuera nuera){
              Colocar(nuera.UbicacionMantel(), nuera.UbicacionMesaFestiva());
              Colocar(nuera.UbicacionCopas(), nuera.UbicacionMesaFestiva());
              Colocar(nuera.UbicacionPlatos(), nuera.UbicacionMesaFestiva());
              Colocar(nuera.UbicacionCubiertos(), nuera.UbicacionMesaFestiva());
       }
       void Colocar(String ubicacionOrigen, String ubicacionDestino)
       {/*Mueve el objeto desde posición origen a destino*/}
}
```

4.1 Indica un principio de los dados en el curso, que este fragmento de código esté violando. Señala en el código dónde se provoca la violación.
4.2 Soluciona los problemas causados por la violación qué encontraste.

**5.** Un fabricante de heladeras utiliza el siguiente código para controlar el funcionamiento de las mismas:

```
class Heladera
{
    private bool enfriar;
    public void Funcionar() {
        if (this.Temp() > 10)
        {
            this.enfriar = true;
        }
        else
        {
            this.enfriar = false;
        }
    }
    public double Temp()
    { /* Código que calcula la temperatura de la heladera */ }
}
```

Actualmente esta heladera según el valor de la temperatura, decide si enfriar o nó.
El fabricante desea desarrollar una heladera inteligente que permita hacer más cosas, por ejemplo: mandar un mensaje de texto cuando queda la puerta abierta, realizar una compra online si no hay más leche, descongelar freezer si hay mucha escarcha y muchas otras labores.

Propone una solución para este problema intentando que la heladera inteligente no se vea muy afectada, al aparecer nuevos componentes (condicion/función). Debes programar y ejemplificar en la heladera al menos dos de los componentes requeridos, dejando claro cómo se integrarían nuevos casos a futuro. Puedes alterar el código cuanto quieras. Cuanto más elegante sea la solución, mejor calificada será. A su vez, no debe violar ninguno de los principios vistos en el curso (Experto, SRP, OCP, LSP e ISP).

**6.** Sea el siguiente código:

```
class Calentador {
        ICalentable sustancia;
        void Calentar () {
        //Calienta de forma muy compleja una sustancia ICalentable}
        }
}
```

6.1 Implementa una `JarraAgua` utilizando los dos mecanismos de reutilización vistos en el curso: a) Herencia b) Composición y Delegación.

6.2 Explica 3 diferencias entre estos 2 mecanismos de reutilización.

**7.** El código presentado a continuación representa un ejemplo del patrón *Template Method Pattern*. Independientemente de si conoces o no este patrón, se te pide que observando el código provisto, describas un posible escenario de uso que se ajuste estrictamente a esta definición. No puedes modificar el código provisto.

```
abstract class Base {
    void MetodoUno(){
        //lineas de código generales a Base y sus Derivadas
        if (MetodoDos()){
            // líneas de código si MetodoDos devuelve True
        }
        else {
            // líneas de código si MetodoDos devuelve False
        }
        //líneas de código generales a Base y sus Derivadas
    }
    public abstract Boolean MetodoDos();
}
class DerivadaUno : Base {
    public override void MetodoDos() {
        //implementación particular a DerivadaUno
    }
}
class DerivadaDos : Base {
    public override void MetodoDos() {
        //implementación particular a DerivadaDos
    }
}
```

La validez de la respuesta estará dada por la real aplicabilidad del ejemplo (no adaptar un problema a la solución, sino encontrar un escenario que sea resuelto por este fragmento).

Consideraciones generales:

- Puedes contestar parte de las preguntas en las hojas del enunciado (**siempre que digas en las hojas de examen que parte de la respuesta está allí**)
- Puedes usar ambos lados de las hoja de examen para responder.
- Debes numerar las hojas.
- No es necesario que copies los fragmentos completos de código para responder una pregunta (simplemente puedes aclarar diciendo algo como "...código desde la línea 3 a la 6...", en ese caso numera las líneas.