Programación orientada a objetos. Examen febrero 2009.

Lunes 9 de febrero de 2009.

Sean las siguientes clases e interfaces:

```
interface Cosmetico {
    public string GetColor { /* implementacion */ }
    void Vencer();
}
abstract class Esmalte: Cosmetico {
    /* Implementacion completa y correcta */
    void AplicarCapa(){}
}
class Delineador : Cosmetico {
    /* Implementacion completa y correcta */
    void Delinear(){}
}
interface Labial : Cosmetico {
    /* Implementacion completa y correcta */
    void Pintar();
}
class AcrilicoFijador: Esmalte {
    /* Implementacion completa y correcta */
    void Proteger(){}
}
class Secante : AcrilicoFijador {
    /* Implementacion completa y correcta */
    void Secar(){}
}
class BrilloReparador : Labial {
    /* Implementacion completa y correcta */
    void Reparar();
}
interface Accesorio { }
class EspumaConAplicador : Cosmetico, Accesorio {
    /* Implementacion completa y correcta */
    void Aplicar() {}
}
class GelParaAfeitar : EspumaConAplicador { /* implementación */ }
class EspumaBlanca : EspumaConAplicador { /* implementación */ }
class Program {
     static void Main() {
       /*1*/ Accesorio maxShave = new EspumaConAplicador();
       /*2*/ Cosmetico lorealStrong = new Delineador ();
       /*3*/ Cosmetico sallyHansenPink = new Esmalte();
       /*4*/ Labial maybellineBeige = lorealStrong;
       /*5*/ Secante oilSellant = new AcrilicoFijador ();
       /*6*/ EspumaBlanca whiteFresh = new EspumaConAplicador();
       /*7*/ Delineador coverGirlFine = lorealStrong;
       /*8*/ Accesorio ColoramaShine = oilSellant;
     }
}
```

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

1.2 Indica los tipos de la variable **oilSellant**.

1.3 Sea el siguiente código:

```
/*1*/ Cosmetico c = new GelParaAfeitar ();
/*2*/ ...
```

¿Qué mensajes puedes enviar a los objetos referenciados por la variable c en la línea 2? ¿Cómo lo sabes?

El siguiente código se utiliza para monitorear una competencia de natación:

```
class Competencia {
      /* completar */
      void AgregarNadador(Nadador n);
      Boolean EstaTerminada();
      Nadador ObtenerGanador();
}
class Nadador {
      /* completar */
      void Clasificar(Competencia c) {/* completar */};
      Boolean Clasifico(Competencia c) {/* completar */};
      void Nadar(Competencia c, Double tiempoNado) {/* completar */};
      Double TiempoNado(Competencia c) {/* completar */};
}
```

El código debe cumplir con las siguientes condiciones:
a) Una competencia esta terminada cuando todos los nadadores han nadado.
b) Para poder competir en una competencia, los nadadores deben haber clasificado.
c) Solo se puede obtener el ganador cuando la competencia ha terminado.
d) No se puede registrar un nuevo nadador una vez que la competencia ha terminado.
e) Para poder nadar en una competencia, el nadador debe haber clasificado.
f) El ganador es el nadador que registro un menor tiempo de nado para esa competencia.
g) Solo se puede obtener el tiempo de nado para una competencia una vez que el nadador haya nadado.

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. Sea el siguiente ejemplo modificado desde el ejercicio uno:

```
    public class Cosmetico {
        public void Vencer() {
            Console.WriteLine("Cosmetico vencido");
        }
    }
    public class Esmalte : Cosmetico {
        public void Vencer(){
            Console.WriteLine("Esmalte vencido");
        }
    }
    public class Program {
        public static void Main() {
            Esmalte c1 = new Esmalte();
            Cosmetico c2 = c1;
            c2.vencer();
        }
    }
```

3.1 Describe esquemáticamente las diferencias entre encadenamiénto estático y encadenamiento dinámico.
3.2 Indica cuál es la salida del programa si se utiliza encadenamiento estático y cuál es si se utiliza encadenamiento dinámico. ¿Cómo lo sabes?
3.3 Estoy programando un método en una clase y no sé si alguna subclase de ésta necesitará sobrescribir ese método ¿lo declaro para que use encadenamiento estático o dinámico? Justifiquen la respuesta en forma breve y concreta.

Sea el siguiente código:

```
    interface IHorno {
        void Prender();
        void Apagar();
        void CocinarPollo(Pollo pollo);
        void CocinarRes(Res res);
    }
```

```
class Horno: IHorno {
    public void Prender() {}
    public void Apagar() {}
    public void CocinarPollo(Pollo pollo) {}
    public void CocinarRes(Res res) {}
}
class ChefAves {
    private IHorno horno;
    public ChefAves(IHorno horno) { this.horno = horno; }
    public void PrepararYProcesarPollo(Pollo pollo) {
        horno.CocinarPollo(pollo);
    }
    public void Prender() {}
    public void Apagar() {}
}
class ChefCarnes {
    private IHorno horno;
    public ChefCarnes(IHorno horno) { this.horno = horno; }
    public void SalarYPrepararRes(Res res) {
        horno.CocinarRes(res);
    }
    public void Prender() {}
    public void Apagar() {}
}
public class Pollo { }
public class Res { }
```

4.1 El código anterior compila y es correcto. ¿Qué principios no se cumplen y por qué?.
4.2 Explica **claramente** que cambios realizarías para que el código cumpla estos principios. No tienes por que volver a programar el código.
Sea el siguiente código:

```
public class Socket {
    public virtual Socket GetFlujoDatos() {
  /* retorna un flujo de datos para escribir en el socket */ }
}
public class FlujoDatos {
    public virtual void write(Byte b) { /* escribe el byte */ }
}
```

5.1 ¿Cuándo utilizarías herencia y cuando utilizarías composición y delegación?
5.2 Te piden que implementes una clase que filtre todos los bytes con valor 0 que son escritos en un socket. Argumenta brevemente si sería mejor usar herencia o composición y delegación para hacerlo, y e implementa la solución con la opción que te parezca más adecuada.

6.1 Modifica el código del ejercicio 4 para que sea posible definir nuevos chefs especializados en un tipo de comida sin necesidad de declarar nuevas clases. Con el nuevo código, debe ser posible definir chefs de pollos, carne, vegetales, etc.

Sea el siguiente código:

```
public class Dodecaedro {
    Double Area() { /*compleja fórmula matemágica*/}
}
public interface Figura {
    Double ObtenerArea();
}
public class ImpresorArea {
    public void Imprimir(Figura f) {
        /*complejisimo código de inicialización de la terminal */
        Console.WriteLine("El area de esta figura es {0}",
                          f.ObtenerArea());
    }
}
```

7.1 Imprime el área de un Dodecaedro sin modificar el código anterior.