Programación orientada a objetos. Primer parcial 1º semestre 2008.

Viernes 18 de abril 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 Igualable {
        Boolean Igual(Igualable i);
}
public interface Comparable: Igualable {
        Boolean Mayor( Comparable i);
        Boolean Menor( Comparable i);
}
public interface Numero: Comparable {
        Numero Sumar (Numero n);
}
public class Cedula: Comparable {
        Natural GetNumero() { /*...*/ }
        Natural GetDigitoControl() { /*...*/ }
        /*...*/
}
public class Irracional: Numero {
        Natural GetNumero() { /*...*/ }
        Natural GetDecimales() { /*...*/ }
        /*...*/
}
public class Natural: Numero { /*...*/ }
class Program {
   static void Main(string[] args) {
      /*1*/ Cedula c1 = new Numero();
      /*2*/ Cedula c2 = new Natural();
      /*3*/ Numero n1 = new Cedula();
      /*4*/ Natural n2 = n1;
      /*5*/ Numero n3 = n2;
      /*6*/ Numero n4 = new Irracional();
      /*7*/ Comparable n5 = n4;
      /*8*/ Irracional n6 = n4;
      /*9*/ Igualable i1 = new Natural();
   }
}
```

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 ¿Qué mensajes puedo enviar a las variables definidas en las líneas correctas? ¿Cómo lo sabes?

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

2.1 ¿Qué tipos tiene una variable declarada como Numero?

2.3 ¿Qué tipos puede tener un objeto referenciado por una variable declarada como Numero? Considera solo los tipos definidos en la pregunta.

Se la siguiente implementación de Natural:

```
public class Natural: Numero {
        private Int32 v;
        public Natural(Int32 v) {
                this.v = v;
        }
        public Boolean Igual(Igualable i) {
                return ((Natural)i).v == v;
        }
        public Boolean Mayor(Comparable i) {
                return ((Natural)i).v > v;
        }
        public Boolean Menor(Comparable i) {
                return ((Natural)i).v < v;
        }
        public Numero Sumar(Numero n) {
                v = v + ((Natural)n).v;
                 return this;
        }
}
```

3.1. ¿Que imprime el siguiente fragmento de código?

```
Natural n1 = new Natural(5);
Natural n2 = n1;
n2.Sumar(new Natural(5));
Console.WriteLine(n2.Igual (new Natural(10)));
Console.WriteLine(n1.Igual (new Natural(10)));
Console.WriteLine(n2.Igual (new Natural(5)));
```

3.2. ¿Son los objetos instancias de Natural inmutables? ¿Por qué? En caso negativo, programa Natural nuevamente para que sean inmutables. **Solo escribe nuevamente el código que cambia**.

4. Supon la existencia de un lenguaje orientado a objetos que no requiere definir el tipo de las variables y no existe el concepto de interfaces. ¿Qué problemas podrían existir desde el punto de vista de un emisor de mensajes? ¿Y de un receptor?

Sea el siguiente código:

```
foo.bar(baz);
```

5.1 ¿Qué determina cuál método se ejecuta si se está aprovechando polimorfismo?

5.2 En el lenguaje descrito en el ejercicio 4, ¿podría aprovechar polimorfismo?. Justifica tu afirmación.

Lee el siguiente código

```
public class Alfajor {
	public Double PrecioDulce;
	public Double PrecioMasa;
}
public class Kiosko {
	public Boolean PuedeComprar(Alfajor a, Double dinero, String moneda) {
		Double pesos = ConvertirAPesos(dinero, moneda);
		return pesos >= a.PrecioDulce + a.PrecioMasa;
	}
	private Double ConvertirAPesos(Double dinero, String moneda) {
		if (moneda.Equals("U$S")) {
			return dinero / 20;
		} else if (moneda.Equals("$")) {
			return dinero;
		} else {
			return -1;
		}
	}
}
```

Analiza muy brevemente el código anterior de acuerdo a:

6.1 Encapsulación

6.2 Patrón Experto

6.3 SRP

6.4 LSP

7. Vuelve a programar el programa del ejercicio anterior basándote en el patrón Experto y SRP.

Sea el siguiente programa:

```
public class Pintor {
	public void Pintar(Caja c) {
		PintarForma(c.GetForma());
	}
	private void PintarForma(Forma forma) {
		// implementación altamente compleja!
	}
}
public class Caja {
	public Forma getForma() {
		// implementación monstruosamente compleja!
	}
}
public class Robot {
	private Forma formaRobot;
	public Robot() {
		formaRobot = ConstruirForma();
	}
	private Forma ConstruirForma() {/* implementación O(!n^!n) */ }
}
public class Forma { /* código y más código */ }
```

8. Programa lo necesario para que Pintor pueda imprimir instancias de Robot guiándote por el OCP (Principio Abierto/Cerrado). Justifica por qué tu programa aplica correctamente el OCP.