Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2014/2015)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Ejercicio 1: *Diseño y* Programación Orientada a Objetos (3,5 puntos)

En una aplicación orientada a objetos para un negocio de numismática (monedas y sellos), todos los objetos numismáticos tiene un valor nominal (impreso o grabado en el sello o moneda) y un factor de antigüedad; ambos participan en el cálculo del precio de mercado del objeto. En general, este precio es la suma del valor nominal más un valor de antigüedad que varía según se trate de un sello o una moneda. En el caso de una moneda el valor de antigüedad es el producto del factor de antigüedad por la mitad del valor nominal; y en el caso de un sello el valor de antigüedad es el producto del factor de antigüedad por el doble del valor nominal. En el caso de los sellos su factor de antigüedad es siempre 0.125.

Se pide diseñar adecuadamente las clases Java necesarias para modelar los objetos descritos arriba de forma que al ejecutar la siguiente clase de prueba se produzca la salida mostrada abajo. No escribas métodos innecesarios para esta ejecución ni repitas código innecesariamente, pero aplica correctamente los fundamentales de diseño a las clases necesarias así como a las relaciones entre ellas y entre sus métodos.

```
public class Ej1 {
  public static void main(String[] args) {
            Numismatico  x1 = new Moneda(8.0, 0.75); // nominal 8, coeficiente antigüedad 0.75
            Numismatico  x2 = new Sello(2.0);       // nominal 2, sin coeficiente conservación

            System.out.println( "        Precio\tNominal\tValorAntigüedad");
            System.out.println( "moneda1: "+x1.precio()+"\t"+ x1.nominal()+"\t"+ x1.valorAntigüedad());
            System.out.println( "sello 2: "+x2.precio()+"\t"+ x2.nominal()+"\t"+ x2.valorAntigüedad());
  }
}
```

Salida esperada:

```
      Precio        Nominal ValorAntigüedad
moneda1: 11.0       8.0     3.0
sello 2: 2.5        2.0     0.5
```

```
public abstract class Numismatico {
            private double valorNominal;
            private double factorAntigüedad;

            public Numismatico(double valorNominal, double coefAntigüedad){
                        this.valorNominal = valorNominal;
                        this.factorAntigüedad = coefAntigüedad;
            }

            public double nominal() { return this.valorNominal; }

            public double factorAntigüedad(){ return this.factorAntigüedad; }

            public double precio(){ return valorNominal+valorAntigüedad(); } // es igual en monedas y en sellos

            public abstract double valorAntigüedad(); // se calcula distinto en monedas y en sellos
}

public class Moneda extends Numismatico {
            public Moneda(double valorNominal, double coefAntigüedad) { super(valorNominal, coefAntigüedad); }

            public double valorAntigüedad() { return this.nominal() * 0.5 * this.factorAntigüedad(); }
}

public class Sello extends Numismatico {
            private static final double COEF_ANTIGUEDAD_SELLOS = 0.125;

            public Sello(double valorNominal) { super(valorNominal, COEF_ANTIGUEDAD_SELLOS);  }

            public double valorAntigüedad() { return this.nominal() * 2 * this.factorAntigüedad(); }
}
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2014/2015)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Ejercicio 2: Colecciones (3,5 puntos)

Se quiere realizar una aplicación para un estudio sobre la evolución de los censos municipales a lo largo de los años. Para ello, se han creado una clase Municipio. Completa dicha clase, si fuese necesario, y la clase CensoHistorico además de añadir la clase HabitantesPorAños para que el programa de prueba dado abajo produzca la salida indicada. Nótese que el orden interno de almacenamiento de los municipios no es relevante, aunque al obtener la serie de números de habitantes de un municipio dado deben obtenerse en orden ascendente de año.

```
public class Municipio {
        private String nombre;
        private String provincia;
        public Municipio(String n, String p) { nombre = n; provincia = p; }
        public String getNombre() { return nombre; }
        public String getProvincia() { return provincia; }
        public String toString() { return nombre+"("+provincia+")"; }
// completar si fuese necesario
        @Override
1       public boolean equals(Object obj) {
                if (! (obj instanceof Municipio)) return false;
                Municipio m = (Municipio) obj;
                return this.nombre.equals(m.nombre) && this.provincia.equals(m.provincia);
        }
        @Override
        public int hashCode() { return this.toString().hashCode(); }
}
```

```
// añadir clase HabitantesPorAños
class HabitantesPorAños {
        private SortedMap<Integer,Double> mapa = new TreeMap<Integer,Double>();
2
        public Double get(Integer i) { return mapa.get(i); }
        public Double put(Integer i, Double d) { return mapa.put(i,d); }

        public String toString() { return mapa.toString(); }
}
```

```
public class CensoHistorico {
        private 3 Map<Municipio,HabitantesPorAños> censosMunicipales = 4new HashMap<>();

// completar con métodos necesarios
5       public CensoHistorico añadeNumHabitantes(Municipio m, Integer a, Double h) {
                if (! censosMunicipales.containsKey(m)) censosMunicipales.put(m, new HabitantesPorAños());
                censosMunicipales.get(m).put(a,h);
                return this;
        }

        public HabitantesPorAños getHabitantes(Municipio m) {
                if (! censosMunicipales.containsKey(m)) return new HabitantesPorAños();
                return censosMunicipales.get(m) ; // ojo será modificable
        }

        public String toString() { return censosMunicipales.toString(); }
}
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2014/2015)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

```
public class Ej21 {
	public static void main(String[] args) {
		CensoHistorico ch = new CensoHistorico();
		Municipio coslada = new Municipio("Coslada", "Madrid");
		Municipio sanFernando = new Municipio("San Fernando", "Madrid");
		Municipio otroSanFernando = new Municipio("San Fernando", "Cadiz");

		ch.añadeNumHabitantes( coslada, 2014, 25.0 )
		  .añadeNumHabitantes( new Municipio("Coslada", "Madrid"), 1988, 18.0 )
		  .añadeNumHabitantes( coslada, 2000, 29.0 );

		HabitantesPorAños sn = ch.getHabitantes(coslada);
		System.out.println( sn );	// años orden ascendente

		ch.añadeNumHabitantes( sanFernando, 2000, 23.0 );
		ch.añadeNumHabitantes( otroSanFernando, 2002, 8.0 );
		System.out.println( ch );
	}
}
```

Salida (con saltos de línea añadidos a mano para mejorar legibilidad):

```
{1988=18.0, 2000=29.0, 2014=25.0}
{San Fernando(Cadiz)={2002=8.0}, San Fernando(Madrid)={2000=23.0},
 Coslada(Madrid)={1988=18.0, 2000=29.0, 2014=25.0}}
```

Prueba 2 de Evaluación Continua

Análisis y Diseño de Software (2014/2015)

Contesta AL DORSO o en las áreas sombreadas, usa hojas adicionales si fuese necesario

Apellidos:
Nombre:

Ejercicio 3: Interfaces y Genericidad (3 puntos)

Se quiere construir una clase genérica Secuencia, que permita concatenar operaciones. Una secuencia tendrá obligatoriamente dos operaciones que se ejecutaran en primer y último lugar, así como cero o más operaciones intermedias. Las operaciones intermedias consumen y producen objetos del mismo tipo, mientras que la primera/última operación produce/consume un objeto *compatible* (aunque no necesariamente del mismo tipo) con las intermedias.

Completa el siguiente programa Java para que produzca la salida de más abajo:

```
class Longitud implements Operacion<String, Integer> {
        public Integer ejecutar(String s) { return s.length(); }
}

class Cuadrado implements Operacion<Integer, Integer> {
        public Integer ejecutar(Integer s) { return s*s; }
}

class Mayor10 implements Operacion<Integer, Boolean> {
        public Boolean ejecutar(Integer s) { return s>10; }
}

public class Genericidad {
        public static void main(String[] args) {
                Secuencia<String, Integer, Boolean> sec =
            new Secuencia<String, Integer, Boolean>(new Longitud(), new Mayor10());  // 1ª y ultima operacion
                sec.anyadeIntermedias(Arrays.asList(new Cuadrado(), new Cuadrado())); // operaciones intermedias
                Operacion<String, Boolean> s = sec;
                System.out.println(sec.ejecutar("Hola"));
                System.out.println(s.ejecutar("J"));
        }
}
```

Salida:

```
true
false
```

```
//Completar

interface Operacion<E, S>{
        public S ejecutar(E entrada);
}

class Secuencia<E, I, S> implements Operacion<E, S>{
        private final Operacion<? super E, ? extends I> primera;
        private final Operacion<? super I, ? extends S> ultima;
        private final List<Operacion<? super I, ? extends I>> intermedias;

        public Secuencia(Operacion<? super E, ? extends I> primera, Operacion<? super I, ? extends S> ultima
        {
                this.primera=primera;
                this.ultima=ultima;
                intermedias= new ArrayList<>();
        }

        public void anyadeIntermedias(List<Operacion<? super I, ? extends I>> nuevas) {
                intermedias.addAll(nuevas);
        }

        @Override
        public S ejecutar(E entrada) {
                I tmp= primera.ejecutar(entrada);
                for (Operacion<? super I, ? extends I> op: intermedias)
                        tmp=op.ejecutar(tmp);
                return ultima.ejecutar(tmp);
        }
}
```