Prueba 3 de Evaluación Continua

Análisis y Diseño de Software (2011/2012)

Contesta en esta misma hoja.

Apellidos:
Nombre:

Apartado 1. (4.5 puntos)

Completa el siguiente programa, para que su ejecución resulte en la siguiente salida:

```
1024: [1, 2, 4, 8, 16, 32]
PARES: [2, 4, 6, 8, 10, 12]
```

```
import  java.util.*;     // Complétalo

// Otra solucion sería que Series<K,V> extendiera de TreeMap<K,SortedSet<V>>
class Series<K,V> {  // Complétalo (a continuacion

   private Map<K,TreeSet<V>> mapa = new TreeMap<K,TreeSet<V>>();
   // Map<K,Set<V> no vale en este caso ¿por qué?

   public void crea(K n, TreeSet<V> serie) {
         mapa.put(n,serie);
   }

   public void añade(K n, V i) {
         if (mapa.containsKey(n)) mapa.get(n).add(i);
   }

   public void imprimeTodas() {
         for (K nombre : mapa.keySet()) {
               System.out.println(nombre +": "+ mapa.get(nombre));
         }
   }
} // end class Series
```

```
public class Main {                              // completa los espacios subrayados
       public static void main(String[] args) {
              Series<String,Integer> series = new Series<String,Integer>();
              Integer[] pot = {1,2,4,8,16,32};
              // Set<Integer> no vale en este caso ¿por qué?
              TreeSet<Integer>   seriePot = new TreeSet<Integer>(Arrays.asList(pot));
              Integer[] pares = {2,4,6,10,12};
              // aqui tampoco, por la misma razón
              TreeSet<Integer>  seriePares = new TreeSet<Integer>(Arrays.asList(pares));
              series.crea("PARES", seriePares);
              series.crea("1024" , seriePot);
              series.añade("PARES", 8);
              series.añade("1024", 4);
              series.imprimeTodas();
       }
}
```

Contesta en esta misma hoja.

Apellidos:
Nombre:

Apartado 2. (3.5 puntos)

Completa el siguiente programa para que produzca la siguiente salida:

```
t1 con duración =6, prioridad=2.5
Excepción : Duración negativa :-1
```

Como puedes observar, la clase *Cola* toma en el constructor una prioridad y un array de valores enteros. La duración de la *Cola* es la suma de dichos valores. El método *getDuracion()* señala un error debido a una duración negativa lanzando una excepción. Debes añadir las clases necesarias y completar las líneas de puntos del *main()*.

```
interface Tarea {
       int    getDuracion() throws DuracionNegativa;
       double getPrioridad();
}

class DuracionNegativa extends Exception {
       private int duracion;
       public DuracionNegativa(int d) {
              this.duracion = d;
       }
       public int getDuracion() {return duracion;}

}

class Cola implements Tarea {
       private double prioridad;
       private Integer[] duraciones;

       public Cola(double p, Integer[] nums) {
              prioridad = p;
              duraciones = nums;
       }
       public int getDuracion() throws DuracionNegativa {
              int suma = 0;
              for (Integer i : duraciones)
                     suma = suma + i;
              if (suma < 0) throw new DuracionNegativa(suma);
              return suma;
       }

       public double getPrioridad() {
              return prioridad;
       }
}


public class Main {
  public static void main(String[] args) {  // Completa las líneas punteadas
    Integer[] val1 = { 1, 2, 3 };
    Integer[] val2 = { -2, 1};
    Tarea t1 = new Cola (2.5, val1); // prioridad=2.5 y valores=val1
    Tarea t2 = new Cola (1.5, val2); // prioridad=1.5 y valores=val2

    try {  // completado

          System.out.println("t1 con duración ="+ t1.getDuracion() +
                             ", prioridad="+t1.getPrioridad());
          System.out.println("t2 con duración ="+ t2.getDuracion() +
                             ", prioridad="+t2.getPrioridad());
         }

        catch (DuracionNegativa d)  // completado
        {
            System.out.println("Excepción, duración negativa: "+d.getDuracion());;
        }
   }
}
```

Contesta en esta misma hoja.

Apellidos:
Nombre:

Apartado 3. (2 puntos)

Completa el siguiente programa para que produzca la siguiente salida:

```
5/15:one third
2/1:dos
8/4:dos
```

```
import java.util.*;

class Fraccion  implements Comparable<Fraccion>    {   // complétalo
	private Integer numerador;
	private Integer denominador;

	public Fraccion(int n, int m) { numerador = n; denominador = m; }

	public String toString() { return numerador + "/" + denominador; }

	public int compareTo(Fraccion f) {               // completa el método

		int dif= this.numerador * f.denominador - this.denominador * f.numerador;
		if (dif == 0) { //  distinguir las fracciones matematicamente iguales
			return this.numerador - f.numerador;
		} else {
			return dif;
		}

	}// end compareTo

	public int hashCode() {                          // completa el método

		return this.numerador.hashCode() * 10101 + this.denominador.hashCode();

	}// end hashCode

	public boolean equals(Object obj) {              // completa el método
		if (this == obj) return true;
		if (obj == null) return false;
		if (! (obj instanceof Fraccion)) return false;
		Fraccion f = (Fraccion) obj;

		return this.compareTo(f) == 0;

	}// end equals
}

public class Main {
	public static void main(String[] args) {
		HashMap<Fraccion,String> m = new HashMap<Fraccion,String>();

		m.put(new Fraccion(8,4),"dos");
		m.put(new Fraccion(5,15),"tercio");
		m.put(new Fraccion(5,15),"one third");
		m.put(new Fraccion(2,1),"dos");
		List<Fraccion> fracciones =  new ArrayList<Fraccion>(m.keySet());
		Collections.sort(fracciones);
		for (Fraccion e : fracciones ) {
			System.out.println(e + ":" + m.get(e));
		}
	}
}
```