EVALUACION NO CONTINUA
Examen final ADSOF 2018/19 – Convocatoria Ordinaria
Contesta cada ejercicio en una hoja separada

Ejercicio 1. (2.5 puntos)

Vamos a desarrollar una aplicación informática que gestione y publique *noticias* y *reportajes* en un periódico digital. Cada noticia se crea con fechas de inicio y de fin del evento que cubre la noticia (p.ej., "*Peor ola de calor en este siglo*" con inicio 21/7/2018 y fin 11/8/2018). Ambas fechas pueden coincidir (p.ej., "*Nadal gana en Roma*" con inicio y fin 19/5/2019). Además cada noticia tiene un titular y un texto ampliado. Las noticias solo pueden ser de 3 tipos distintos (deportivas, sucesos o políticas), y cada tipo tiene características adicionales propias. Las noticias deportivas tienen un deporte y un nombre del equipo o deportista implicado. Los sucesos incluyen un tipo de suceso y la localidad donde ha ocurrido. Las noticias políticas incluyen la lista de partidos políticos involucrados en la noticia.

También se gestionarán *reportajes* formados por otras noticias o reportajes. Cada reportaje también tiene titular, texto ampliado y dos fechas: la de inicio será la menor de todas las fechas de inicio de sus contenidos y la fecha final será la mayor de todas las fechas de final de sus contenidos. Ambas fechas podrán variar cuando se añadan contenidos al reportaje. Un reportaje podrá ser eliminado, sin que ello signifique la eliminación de sus contenidos.

Otro componente importante de la aplicación son los *presentadores* que reaccionan a cambios en la popularidad de las noticias para *seleccionarlas y presentarlas* de distintas maneras a los lectores del periódico. Por ejemplo, un presentador de novedades incluye la noticia en la sección de Última Hora al detectar un pico fuerte en sus descargas recientes. Otra posibilidad es un presentador de tendencias que muestra la noticia en portada cuando detecta un nivel de descargas alto sostenido durante más de 6 horas. Debes hacer un diseño extensible que facilite añadir otros tipos de presentadores en versiones futuras de la aplicación

Las noticias y reportajes pueden ser descargadas por el público para leerlas en su terminal. Descargar un reportaje implica descargar todos sus contenidos. Cada descarga de una noticia o reportaje será contabilizada actualizando sus estadísticas de acceso. Dichas estadísticas incluirán las descargas en cada minuto de las últimas 24 horas y el tiempo medio transcurrido entre descargas realizadas durante los últimos 10 minutos. Todas las descargas deben activar a los presentadores encargados de vigilar la popularidad de la noticia o reportaje descargado. Cada presentador se configura con la noticia o reportaje cuya popularidad se va a encargar de vigilar. Una misma noticia o reportaje puede tener asociados varios presentadores para vigilar su popularidad.

Se pide:

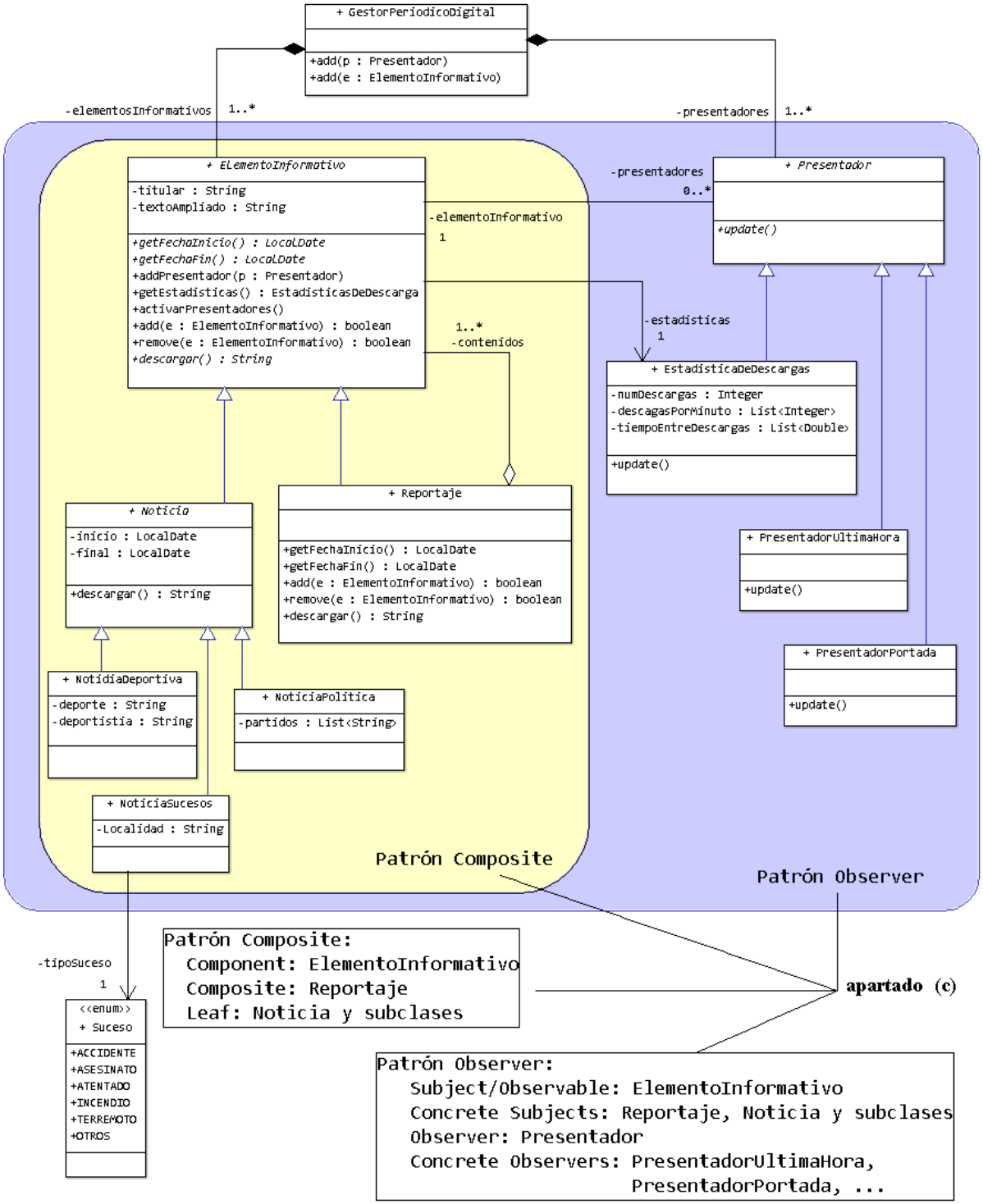
(a) Representar el diagrama de clases en UML para el fragmento de aplicación descrito arriba, con todos los métodos necesarios excepto *getters* sencillos, que simplemente devuelvan el valor del atributo correspondiente. Tampoco hace falta incluir los constructores. **(1.5 puntos)**

(b) Añade los siguientes métodos a la(s) clase(s) adecuada(s) de tu diagrama de clases, indicando sus **parámetros** de entrada, valor de **retorno** y **seudocódigo (**no hace falta todo el detalle de código Java) **(0.6 puntos)**

1. Un método para obtenga la fecha de inicio de un reportaje.
2. Un método que realice la descarga de un reportaje.
3. Un método que active los presentadores asociados a una noticia o reportaje cuando se descarga.

(c) Indica qué patrón o patrones de diseño has utilizado en tu diagrama, identificando las clases de tu diagrama que se corresponden con los elementos de dicho(s) patrón(es). **(0.4 puntos)**

Solución: (a)



(b) Método en clase Reportaje, getFechaInicio(): LocalDate

```
LocalDate inicio = this.contenidos.get(0).getFechaInicio();
for (ElementoInformativo e : this.contenidos
  if ( e.getInicio().compareTo(inicio)<0 )  inicio = e.getInicio();
return inicio;
```

Método en clase Reportaje, descargar() : String

```
String descarga = this.titular + "\n" + this.textoAmpliado + "\n";
for (ElementoInformativo e : this.contenidos)
  descarga = descarga + e.descargar() + "\n";
this.activarPresentadores();
return descarga;
```

Método en clase ElementoInformativo, activarPresentadores(): {

```
for (Presentador p : this.presentadores)
   p.update();
```

Ejercicio 2 (2.75 puntos)

Se quiere construir una aplicación para la creación de cursos educativos on-line. Como parte de esta aplicación, debes crear clases Java para la gestión de temas y sus contenidos. Un tema tiene un nombre, y opcionalmente un tipo de actividad. Estas pueden ser de tipo teoría, prácticas, y de refuerzo. Cada tipo de actividad lleva consigo un grado de dificultad: 1 para las de refuerzo, 2 para las prácticas y 3 para la teoría. Un tema puede contener otros temas o pruebas (pero no puede contener directa o indirectamente elementos repetidos). Una Prueba tiene un nombre, un tipo de actividad y un entero que indica un grado extra de dificultad. El sistema debe generar un identificador único tanto para las pruebas, como para los temas.

Uno de los objetivos de la aplicación es poder calcular el *volumen* de un tema, así como la *dificultad* estimada de pruebas y temas. El volumen de un tema se calcula como el número de elementos que contiene, directa o indirectamente, incluido él mismo. La dificultad de un tema es la suma de la dificultad de todos sus elementos contenidos, directa o indirectamente (incluido él mismo); y la de una prueba es la suma de la dificultad extra más la del tipo de actividad.

Se pide:

(a) **[2.25p]** Utilizando patrones de diseño completar el siguiente programa Java para obtener la salida de más abajo. Se valorará el uso apropiado de conceptos de orientación a objetos, la extensibilidad y la concisión del código.

(b) **[0.5p]** ¿Qué patrones has utilizado?, identifica las clases que se corresponden con los elementos de dicho(s) patrón(es).

```
public class Main {
  public static void main(String[] args) {
    Tema t1    = new Tema("Repaso de programacion");        // sin tipo actividad
    Tema t12   = new Tema("Bucles", TipoActividad.TEORIA); // tipo teoría (dificultad 3)
    // prueba práctica (dificultad 2) con grado extra de dificultad 2
    Prueba p1  = new Prueba("For, while y do", TipoActividad.PRACTICAS, 2);
    Tema t13   = new Tema("Recursion", TipoActividad.REFUERZO); // tema de refuerzo (dif. 1)
    // prueba de refuerzo (dificultad 1), extra 4 dificultad
    Prueba p2  = new Prueba("Pasar de recursivo a iterativo", TipoActividad.REFUERZO, 4);

    t1.anyade(t12, p1, t13);
    t13.anyade(p2);

    System.out.println("Podemos anyadir p2 a t1: "+t1.anyade(p2));
    System.out.println(t1);
    System.out.println("Volumen: "+t1.volumen()+" elementos");
    System.out.println("Dificultad: "+t1.dificultad());

    System.out.println("Dificultad: "+p2.dificultad());
  }
}
```

Salida esperada: ***(donde el identificador n del tema o prueba se escribe como \|n\|)***

```
Podemos anyadir p2 a t1: false
|0|:Repaso de programacion : [|1|:Bucles (TEORIA), Prueba: |2|:For, while y do (PRACTICAS),
|3|:Recursion (REFUERZO): [Prueba: |4|:Pasar de recursivo a iterativo (REFUERZO)]]
Volumen: 5 elementos
Dificultad: 13
Dificultad: 5
```

```
enum TipoActividad {
    REFUERZO(1), TEORIA(3), PRACTICAS(2);

    private int dificultad;
    TipoActividad(int dificultad) { this.dificultad = dificultad; }
    public int dificultad() { return this.dificultad; }
}

abstract class Actividad {
    private final String titulo;
    private final TipoActividad ta;
    private final int id;
    private static int contador = 0;

    public Actividad(String titulo, TipoActividad ta) {
        this.titulo = titulo;
        this.ta = ta;
        this.id = contador++;
    }

    public boolean contiene(Actividad a) { //en principio solo si son la misma
        return this==a || a.id == id;
    }
    public int volumen() { return 1; }
    public int dificultad() { return ta!=null? ta.dificultad():0; }
    @Override
    public String toString() { return "|"+this.id+"|:"+this.titulo+ (ta!=null ? " ("+ta+")":"") ; }
}

class Tema extends Actividad {
    private final List<Actividad> temas = new ArrayList<>();

    public Tema(String titulo, TipoActividad ta) {super(titulo, ta);     }
    public Tema(String titulo) { this(titulo, null); }

    @Override public int volumen() {
        int acum = super.volumen();
        for (Actividad u : temas)
            acum += u.volumen();
        return acum;
    }
    @Override public int dificultad() {
        int acum = super.dificultad();
        for (Actividad u : temas)
            acum += u.dificultad();
        return acum;
    }

    public boolean anyade(Actividad...unidades) {
        for (Actividad u : unidades) {
            if (contiene(u) || u.contiene(this)) return false;
            temas.add(u);
        }
        return true;
    }

    @Override
    public boolean contiene(Actividad u) {
        if (super.contiene(u)) return true; //el mimso
        for (Actividad t : temas)
            if (t.contiene(u)) return true;
        return false;
    }
    @Override
    public String toString() { return super.toString()+(temas.isEmpty() ? "" : " : "+temas);}
}

class Prueba extends Actividad {
    private int gradoDificultad;
    public Prueba(String titulo, TipoActividad te, int gd) {
        super(titulo, te);
        this.gradoDificultad = gd;
    }
    @Override public int dificultad() { return super.dificultad()+gradoDificultad; }
    @Override public String toString() { return "Prueba: "+super.toString(); }
}
```

(b) Composite, donde Actividad es *Component*, Prueba es *Leaf*, y Tema es *Leaf* y *Component*.

Ejercicio 3 (3 puntos)

Se quiere construir una clase genérica `MapaValorable` que se parametriza con un tipo, y asocia una colección de valoraciones a distintos objetos del tipo paramétrico. Las valoraciones deben ser compatibles con `IValoracion`, una interfaz que permite obtener la puntuación de la valoración, que es un entero. Todos los objetos compatibles con `IValoracion` deben forzosamente definir un orden natural, que por defecto será el que viene dado por la puntuación, y que se usará para ordenar las valoraciones asociadas a los objetos.

Adicionalmente, deseamos poder presentar los objetos de acuerdo al resultado de hacer la media de la puntuación de sus valoraciones (truncada a entero), de menor a mayor. Esto es tarea del método `porOrden` de `MapaValorable` que debe lanzar una excepción si la colección de valoraciones es vacía. Este método debe estar sobrecargado, para permitir configurarse con otros órdenes (por ejemplo, de mayor a menor).

El siguiente listado ilustra el uso de la clase `MapaValorable` para almacenar críticas sobre películas. Las críticas de cada película vienen ordenadas por puntuación, y en caso de empate por orden alfabético.

Se pide: Todo el código Java necesario para que el siguiente programa imprima la salida de más abajo. Se valorará el uso apropiado de conceptos de orientación a objetos, la extensibilidad y la concisión del código.

```
public class Main {
  public static void main(String[] args) {
    MapaValorable<Pelicula> cartelera = new MapaValorable<>();
    List<IValoracion> buenasCriticas  = Arrays.asList(    new Critica("excelente!", 5),
                                                          new Critica("divertida", 4),
                                                          new Critica("buena", 4));
    Collection<Critica> malasCriticas = Arrays.asList(    new Critica("infumable", 1),
                                                          new Critica("incomprensible", 2));
    cartelera.anyade(new Pelicula("Vengadores: endgame", 2019), buenasCriticas);
    cartelera.anyade(new Pelicula("Vengadores: endgame", 2019), malasCriticas);
    cartelera.anyade(new Pelicula("Lady bird", 2017),           buenasCriticas);
    cartelera.anyade(new Pelicula("Sharknado", 2013),           buenasCriticas);

    System.out.println("La puntuación de '"+buenasCriticas.get(0)+"' es "+
                    buenasCriticas.get(0).puntuacion());

    System.out.println("Cartelera :\n"+cartelera);
    ________ {                    // Completar [1]
       // Presentar películas por media de puntuaciones, de mayor a menor
      System.out.println("Cartelera por media de puntuaciones: \n"+
                         cartelera.porOrden((a, b)->b-a));
      cartelera.anyade(new Pelicula("Kung Fury", 2015), Collections.emptyList());
       // Presentar películas por media de puntuaciones, de menor a mayor (orden por defecto)
      System.out.println("Cartelera por media de puntuaciones: "+cartelera.porOrden());
    } _____________________________ { // Completar [2]
      System.out.println(v);
    }
  }
}
```

Salida esperada

```
La puntuación de '5: excelente!' es 5
Cartelera :
{Vengadores: endgame (2019)=[1: infumable, 2: incomprensible, 4: buena, 4: divertida, 5:
excelente!], Lady bird (2017)=[4: buena, 4: divertida, 5: excelente!], Sharknado (2013)=[4:
buena, 4: divertida, 5: excelente!]}
Cartelera por media de puntuaciones:
{4=[Lady bird (2017), Sharknado (2013)], 3=[Vengadores: endgame (2019)]}
Valores vacios para la clave Kung Fury (2015)
```

Solución

```
class ExcepcionValoresVacios extends Exception {
        private String clave;
        public ExcepcionValoresVacios(String cl) { this.clave = cl; }
        public String toString() { return "Valores vacios para la clave "+this.clave;}
}

interface IValoracion extends Comparable<IValoracion>{
        int puntuacion();
        @Override default int compareTo(IValoracion o) {        return this.puntuacion()-o.puntuacion();}
}

class Critica implements IValoracion {
        private int puntuacion;
        private String texto;

        public Critica(String t, int p) {
                this.puntuacion = p;
                this.texto = t;
        }
        @Override public int compareTo(IValoracion o) {
                int com = IValoracion.super.compareTo(o);
                return (com==0) ? this.texto.compareTo(((Critica)o).texto) : com;
        }
        @Override public int puntuacion() { return this.puntuacion; }
        @Override public String toString() { return this.puntuacion+": "+this.texto; }
}

class Pelicula {
        private String titulo;
        private int anyo;

        public Pelicula (String t, int a) { this.titulo = t; this.anyo = a; }

        @Override public boolean equals(Object p) {
                if (this==p) return true;
                if (!(p instanceof Pelicula)) return false;
                Pelicula peli = (Pelicula) p;
                return this.titulo.equals(peli.titulo) && this.anyo == peli.anyo;
        }
        @Override public int hashCode ()   { return this.titulo.hashCode()+this.anyo; }
        @Override public String toString() { return this.titulo+" ("+this.anyo+")"; }
}

class MapaValorable<T>  {
        private Map<T, TreeSet<IValoracion>> valoraciones = new LinkedHashMap<>();

        public void anyade(T clave, Collection<? extends IValoracion> valoraciones) {
                if (!(this.valoraciones.containsKey(clave))) this.valoraciones.put(clave, new TreeSet<>());
                this.valoraciones.get(clave).addAll(valoraciones);
        }
        public SortedMap<Integer, List<T>> porOrden(Comparator<Integer> c) throws ExcepcionValoresVacios{
                SortedMap<Integer, List<T>> ordered = new TreeMap<>(c);
                for (T t : this.valoraciones.keySet()) {
                        if (this.valoraciones.get(t).isEmpty()) throw new ExcepcionValoresVacios(t.toString());
                        int valor = this.valoraciones.get(t).stream().mapToInt(IValoracion::puntuacion).sum()/
                                    this.valoraciones.get(t).size();
                        if (! ordered.containsKey(valor) ) ordered.put(valor, new ArrayList<T>());
                        ordered.get(valor).add(t);
                }

                return ordered;
        }
        public SortedMap<Integer, List<T>> porOrden() throws ExcepcionValoresVacios{ return this.porOrden((a,b)->a-b);}
        @Override public String toString() { return this.valoraciones.toString(); }
}

// [1] try
// [2] catch (ExcepcionValoresVacios v)
```

Ejercicio 4 (1.75 puntos)

Se quiere diseñar una clase `MapTools`, con métodos útiles para potenciar el uso del método `map` de `Stream`. Debes añadir el método `enRango` que permitirá usar `map` con una función *base* y dos funciones *limitadoras*, una *limitadora superior* y otra *limitadora inferior*. Sin considerar las funciones limitadoras, el resultado final sería el mismo que utilizando el método `map` directamente con la función base. En cambio, cada función limitadora impone un límite (uno superior y otro inferior) al valor resultante de la función base antes de que sea tratado por el método `map` en su forma habitual. Es decir, si el valor de la función base supera al de la limitadora superior, `map` deberá recibir el valor de la limitadora superior para operar con él, y si es inferior al de la limitadora inferior deberá recibir el de esta limitadora. En cualquier otro caso, `map` debe recibir el valor de la función base. Ver ejemplos de uso abajo. Además, si la función limitadora inferior produjese un valor superior al producido por la función limitadora superior se debe generar una excepción como se muestra al final de la salida esperada.

Se pide: Implementar el método `enRango` para la clase `MapTools` de la forma más genérica y flexible que sea razonable para que el siguiente programa ejemplo produzca la salida indicada abajo.

```
import java.util.*;
import java.util.stream.*;

public class Ej4 {
    public static Integer doble(Number n) { return 2 * n.intValue(); }

    public static void  main(String[] args) {
     Collection<Integer> lista = Arrays.asList(2, -3, 5, 32, -10, -20);
     System.out.println( lista.stream()
                              .map( Ej4::doble )  // sin limitadoras
                              .collect( Collectors.toList() ));

     System.out.println( lista.stream()  // con dos limitadoras, inferior y superior
                              .map( MapTools.enRango(Ej4::doble, x -> x-3,  x -> x+3) )
                              .collect( Collectors.toList() ));

     Collection<String> palabras = Arrays.asList( "bata", "buey", "paz", "zoo" );
     System.out.println( palabras.stream()
                                 .map( MapTools.enRango(x->x,  x->"boy",  x->"que") )
                                 .collect( Collectors.toList() ));

     System.out.println( lista.stream()  // limitadoras provocan excepción
                              .map( MapTools.enRango(Ej4::doble, x -> x*x,  x -> 3*x) )
                              .collect( Collectors.toList() ));
   }
}
```

Salida esperada:

```
[4, -6, 10, 64, -20, -40]
[4, -6, 8, 35, -13, -23]
[boy, buey, paz, que]
Exception in thread "main" ExcepcionRango: Límite superior -9 menor que el inferior 9
```

SOLUCION:

```
class ExcepcionRango extends RuntimeException { public ExcepcionRango (String msg) { super(msg); } }

class MapTools {
   public static <T extends Comparable<? super T>>  Function<T, T>
                 enRango( Function<? super T, ? extends T>  base,
                          Function<? super T, ? extends T> bottom,
                          Function<? super T, ? extends T> top)
   {

        return x -> {
                T r = base.apply(x);
                T max = top.apply(x);
                T min = bottom.apply(x);
                if ( min.compareTo(max) > 0 )
                 throw new ExcepcionRango ("Limite superior " + max + " menor que el inferior "+ min);
                return (max.compareTo(r) > 0) ? max : (min.compareTo(r) < 0) ? min : r;
              };
    }
}
```