ADSOF: Segundo examen parcial - 06/05/2020

Ejercicio 2 (3.5 puntos)

Necesitamos una aplicación para obtener el tiempo estimado necesario para grabar *vídeos* para clases de varias categorías de *contenidos*. El *tiempo estimado* (en minutos) para grabar cada vídeo resulta de aplicar un porcentaje de *incremento por complejidad de la materia* a la suma de su *duración de contenido* más su *duración de revisión*. Los porcentajes de incremento por complejidad de la materia son 0%, 12.5%, 12.5% y 25% respectivamente para las complejidades básica, media, avanzada y extrema, respectivamente. La duración de revisión será 0 por defecto, a menos que se haya especificado otra para ese vídeo en concreto. La duración de contenido se debe calcular de forma distinta según el tipo de vídeo. Por ahora consideraremos solo los vídeos sobre *teoría* y resolución de *ejercicio*. La duración de contenido de los vídeos sobre un ejercicio se indica individualmente, en horas o fracción, junto a la descripción del vídeo y la complejidad de la materia asignadas para cada ejercicio. La duración de contenido de los vídeos sobre teoría depende del número de diapositivas a tratar que se indica junto a la descripción y a la complejidad de la materia (asumiendo complejidad baja si no especifica ninguna): serán de 10 minutos por diapositiva, con un mínimo de 100 minutos y un máximo de 400 minutos por vídeo de teoría.

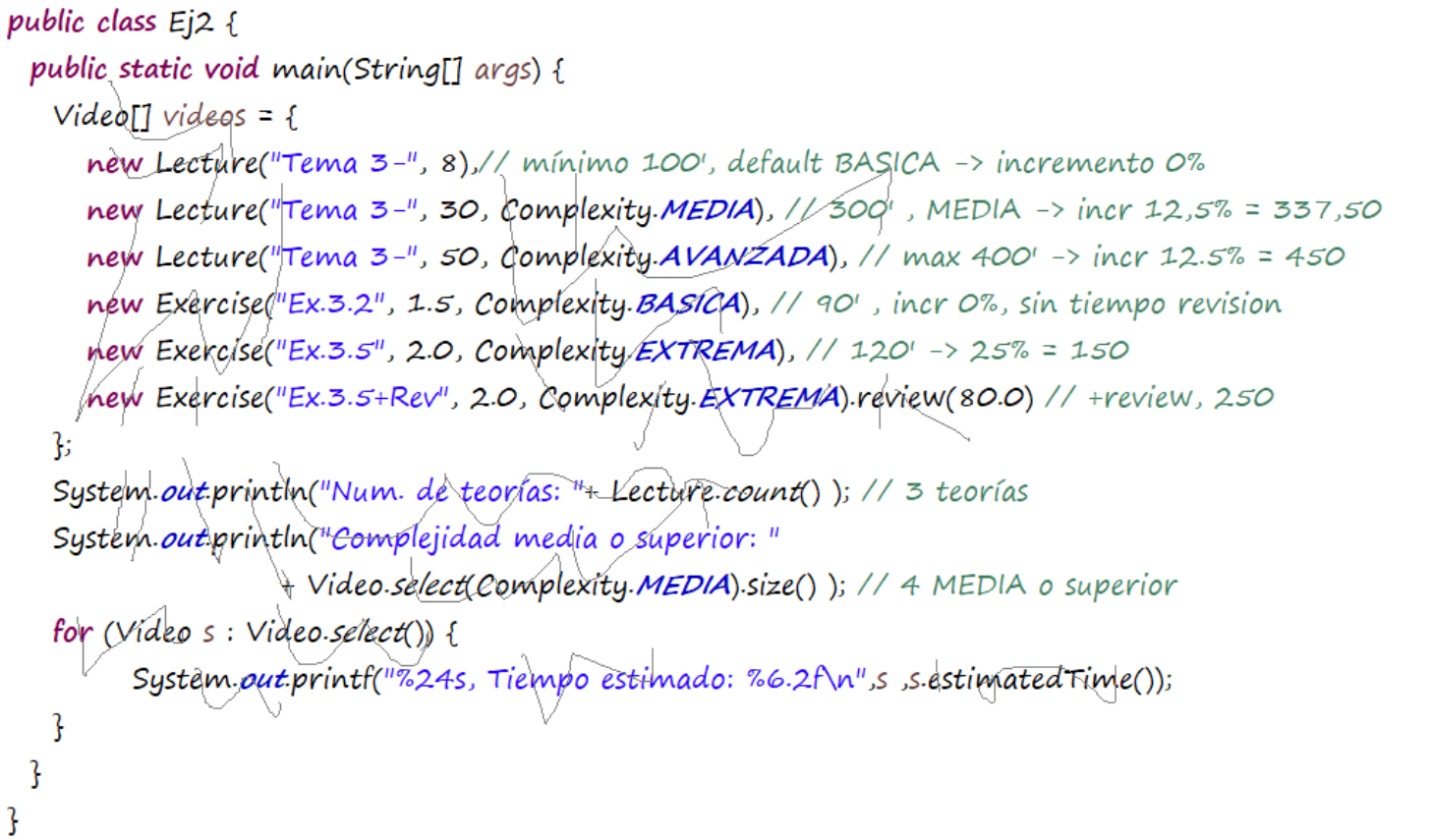
Además, queremos poder acceder fácilmente a todos los vídeos de complejidad igual o superior a una complejidad indicada, o todos los vídeos si no se indica ninguna. También deseamos contabilizar el número total de vídeos de teoría, haciendo que la aplicación añada automáticamente dicha numeración a la descripción de cada vídeo de teoría.

Nota: por simplicidad, puedes ignorar el control de errores derivados de cantidades negativas y strings vacíos o nulos.

Se pide diseñar y codificar en Java, siguiendo buenos principios de orientación a objetos, las clases necesarias para resolver los requisitos anteriores haciendo que el programa dado más abajo **produzca la siguiente salida:**

```
Num. de teorías: 3
Complejidad media o superior: 4
       Tema 3-#1, BASICA, Tiempo estimado: 100,00
        Tema 3-#2, MEDIA, Tiempo estimado: 337,50
     Tema 3-#3, AVANZADA, Tiempo estimado: 450,00
          Ex.3.2, BASICA, Tiempo estimado:  90,00
         Ex.3.5, EXTREMA, Tiempo estimado: 150,00
     Ex.3.5+Rev, EXTREMA, Tiempo estimado: 250,00
```

Tester en forma de imagen para evitar cortar y pegar de texto del enunciado a Eclipse. Más abajo va en forma de texto.



```
public class Ej2 {
  public static void main(String[] args) {
    Video[] videos = {
      new Lecture("Tema 3-", 8),// mínimo 100', default BASICA -> incremento 0%
      new Lecture("Tema 3-", 30, Complexity.MEDIA), // 300' , MEDIA -> incr 12,5% = 337,50
      new Lecture("Tema 3-", 50, Complexity.AVANZADA), // max 400' -> incr 12.5% = 450
      new Exercise("Ex.3.2", 1.5, Complexity.BASICA), // 90' , incr 0%, sin tiempo revision
      new Exercise("Ex.3.5", 2.0, Complexity.EXTREMA), // 120' -> 25% = 150
      new Exercise("Ex.3.5+Rev", 2.0, Complexity.EXTREMA).review(80.0) // +review, 250
    };
    System.out.println("Num. de teorías: "+ Lecture.count() ); // 3 teorías
    System.out.println("Complejidad media o superior: "
                       + Video.select(Complexity.MEDIA).size() ); // 4 MEDIA o superior
    for (Video s : Video.select()) {
        System.out.printf("%24s, Tiempo estimado: %6.2f\n",s ,s.estimatedTime());
    }
  }
}
```

Solución: Indica asignación orientativa de n décimas de puntos en forma de [n] sobre total de 35. Otros fallos penalizados aparte

```
public enum Complexity {
	// BASIC(0.0), MEDIUM(0.125), ADVANCED(0.125), EXTREME(0.25);
	BASICA(0.0), MEDIA(0.125), AVANZADA(0.125), EXTREMA(0.25);   // [2]
	private double value;
	private Complexity(double value) { this.value = value; }    // [2]
	public double getValue() { return this.value; }
}
import java.util.*;
public abstract class Video {     //  abstract  [1]
	private static List<Video> videos = new ArrayList<>();
	public static List<Video> select(Complexity en) {     // [3]
		List<Video> result = new ArrayList<>();
		for (Video a : Video.videos)
			if (a.priority.compareTo(en)>=0) result.add(a);
		return result;
	}
	public static List<Video> select() {           // [2]
		// return select( Complexity.values()[0]);
		return Collections.unmodifiableList(Video.videos);
	}
	private Complexity priority;
	private String description;
	public Video(String d, Complexity e) {     // [3]
		description = d; priority = e; videos.add(this); }

	public double estimatedTime()  { return (this.content()   // [4]
					+ this.review() )
					* (1.0 + this.priority.getValue()); }
	public abstract double content();              // [1]
	public  double review() { return 0.0; }       // [1]
	public String toString() {return description + ", "   + this.priority;}   // [1]
}
public class Lecture extends  Video {    // extends [1]
	public static final double MAX_CONTENT = 400.0;
	public static final double MIN_CONTENT = 100.0;
	public static final double CONTENT_COST_PER_SLIDE = 10.0;
	private static int lastId = 0;                    // [1]
	public static int count() { return lastId; }     // [1]

	private int numSlides;
	public Lecture(String descripcion, int c1, Complexity enumera) {
		super(descripcion + "#"+  ++lastId, enumera);     // [2]
		this.numSlides = c1;
	}
	public Lecture(String descripcion, int c1) {
		this(descripcion, c1, Complexity.BASICA); // BASIC);     // [2]
	}
	@Override public double  content() {    // [1]
		return Math.min(MAX_CONTENT,
				Math.max(MIN_CONTENT,
					CONTENT_COST_PER_SLIDE * this.numSlides));
	}
}
public class Exercise extends Video {     // extends [1]
	private double content;
	private double extra = 0.0;
	public Exercise(String descripcion, double hours, Complexity enumera) {
		super(descripcion, enumera);   // [2]
		content = 60.0 * hours;
	}
	@Override public double content() { return content; }         // [1]
	@Override public double review()  { return this.extra;}    // [1]
	public Exercise review(double extra)  { this.extra = extra; return this;}   // [2]
}
```