



Ejercicio 1

```
public class CanalTwitch implements Comparable<CanalTwitch> {

    public Double getPorcentajeRetencion() {
        Double res = 0.;
        if (getNumEspectadoresGanados() > 0) {
            res = 100.0 * getNumFollowersGanados() / getNumEspectadoresGanados();
        }
        return res;
    }

    public Double getRatioEfectividad() {
        //NOTA: También es válido usar getSeconds()
        return getTiempoVisionado().toSeconds() *
            1. / getDuracionContenido().toSeconds();
    }

    public Integer getNumEspectadoresOcasionales() {
        return getNumEspectadoresPicoMax() - getMediaEspectadores();
    }
}
```

Ejercicio 2

```
public String getCanalMasPorcentajeRetencion(String idioma) {

    Predicate<CanalTwitch> pred =
        canal -> idioma == null || canal.getIdioma().equals(idioma);

    Comparator<CanalTwitch> c =
        Comparator.comparing(CanalTwitch::getPorcentajeRetencion);

    return canales.stream()
        .filter(pred)
        .max(c)
        .map(CanalTwitch::getNombre)
        .orElse("");
}
```

Ejercicio 3

Solución 1: usando reduce

```
public Duration getDuracionTotalContenido(Boolean esSocio) {

    return canales.stream()
        .filter(canal->canal.getSocioTwitch().equals(esSocio))
        .map(CanalTwitch::getDuracionContenido)
        .reduce(Duration.ZERO, Duration::plus);
}
```

Solución 2: haciendo conversión a segundos

```
public Duration getDuracionTotalContenido2(Boolean esSocio) {  
  
    //NOTA: también es válido usar getSeconds en la función  
    Long totalDuracion = canales.stream()  
        .filter(canal -> canal.getSocioTwitch().equals(esSocio))  
        .mapToLong(canal -> canal.getDuracionContenido().toSeconds())  
        .sum();  
  
    return Duration.ofSeconds(totalDuracion);  
}
```

Ejercicio 4

```
public IntegergetNumCanalesPorEncimaMediaEspectadoresOcasionales() {  
  
    Double mediaHits = canales.stream()  
        .mapToInt(CanalTwitch::getNumeroEspectadoresOcasionales)  
        .average()  
        .getAsDouble();  
  
    Long res = canales.stream()  
        .filter(canal -> canal.getNumEspectadoresOcasionales() > mediaHits)  
        .count();  
  
    return res.intValue();  
}
```

Ejercicio 5

Solución 1: más eficiente, con un solo recorrido

```
public Double getPorcentajeCanalesAdultosEntreNMasVisionados(Integer n) {  
  
    Checkers.check("n debe ser mayor que cero", n > 0);  
  
    Comparator<CanalTwitch> c = Comparator.comparing(CanalTwitch::getTiempoVisionado)  
        .reversed();  
  
    Map<Boolean, List<CanalTwitch>> mapMasVisionados = canales.stream()  
        .sorted(c)  
        .limit(n)  
        .collect(Collectors.partitioningBy(CanalTwitch::getContenidoAdulito));  
  
    Integer numCanalesAdultos = mapMasVisionados.get(true).size();  
    Integer numCanalesNoAdultos = mapMasVisionados.get(false).size();  
  
    return 100.0 * numCanalesAdultos / (numCanalesAdultos + numCanalesNoAdultos);  
}
```

Solución 2: con dos recorridos

```
public Double getPorcentajeCanalesAdultosEntreNMasVisionados(Integer n) {  
  
    Checkers.check("n debe ser mayor que cero", n > 0);  
    Comparator<CanalTwitch> c = Comparator.comparing(CanalTwitch::getTiempoVisionado)  
        .reversed();  
    List<CanalTwitch> masVisionados = canales.stream()  
        .sorted(c)  
        .limit(n)  
        .collect(Collectors.toList());  
  
    Long numCanalesAdultos = masVisionados.stream()  
        .filter(CanalTwitch::getContenidoAdulto)  
        .count();  
  
    return 100.0 * numCanalesAdultos/masVisionados.size();  
}
```

Ejercicio 6

Solución 1: con Collectors.toMap

```
public SortedMap<String, CanalTwitch> getCanalMasEfectivoPorIdioma() {  
  
    Comparator<CanalTwitch> c =  
        Comparator.comparing(CanalTwitch::getRatioEfectividad);  
  
    return canales.stream()  
        .collect(Collectors.toMap(CanalTwitch::getIdioma,  
            Function.identity(),  
            BinaryOperator.maxBy(c),  
            TreeMap::new));  
}
```

Solución 2: con Collectors.collectingAndThen

```
public SortedMap<String, CanalTwitch> getCanalMasEfectivoPorIdioma() {  
  
    Comparator<CanalTwitch> c =  
        Comparator.comparing(CanalTwitch::getRatioEfectividad);  
  
    return canales.stream()  
        .collect(Collectors.groupingBy(CanalTwitch::getIdioma,  
            TreeMap::new,  
            Collectors.collectingAndThen(  
                Collectors.maxBy(c),  
                Optional::get)));  
}
```

Ejercicio 7

```
public List<String> getRankingIdiomasOrdenadoPorPorcentajePresencia() {  
  
    Map<String, Double> mPorcentajes = getPorcentajePresenciaPorIdioma();  
  
    Comparator<Entry<String, Double>> c = Entry.comparingByValue();  
    // También es válido  
    // Comparator<Entry<String, Double>> c = Comparator.comparing(Entry::getValue);  
  
    return mPorcentajes.entrySet().stream()  
        .sorted(c.reversed())  
        .map(Entry::getKey)  
        .collect(Collectors.toList());  
}
```

El método getPorcentajePresenciaPorIdioma se puede resolver de varias formas.

Solución 1: con dos recorridos y Collectors.summingInt

```
private Map<String, Double> getPorcentajePresenciaPorIdioma() {  
  
    Long duracionTotal = canales.stream()  
        .mapToLong(canal->canal.getDuracionContenido().toSeconds())  
        .sum();  
  
    return canales.stream()  
        .collect(Collectors.groupingBy(CanalTwitch::getIdioma,  
            Collectors.summingDouble(  
                canal -> canal.getDuracionContenido().toSeconds() * 100.0 /  
                duracionTotal)));  
}
```

Solución 2: con un recorrido de todos los canales y dos recorridos sobre el Map, el segundo recorrido con Collectors.toMap

```
private Map<String, Double> getPorcentajePresenciaPorIdioma() {  
  
    // Duración total por idioma  
    Map<String, Long> m = canales.stream()  
        .collect(Collectors.groupingBy(CanalTwitch::getIdioma,  
            Collectors.summingLong(  
                canal -> canal.getDuracionContenido().toSeconds())))  
  
    // Duración total de todo el contenido  
    Long duracionTotal = m.values().stream()  
        .mapToLong(duracion- > duracion)  
        .sum();  
  
    // Porcentaje de presencialidad por idioma  
    return m.entrySet().stream()  
        .collect(Collectors.toMap(  
            Entry::getKey,  
            e -> e.getValue() * 100.0 / duracionTotal));  
}
```

Solución 3: con dos recorridos de todos los canales y una función auxiliar para el cálculo de porcentajes.

```
private Map<String, Double> getPorcentajePresenciaPorIdioma() {  
  
    Long duracionTotal = canales.stream()  
        .mapToLong(canal -> canal.getDuracionContenido().toSeconds())  
        .sum();  
  
    return canales.stream()  
        .collect(Collectors.groupingBy(  
            CanalTwitch::getIdioma,  
            Collectors.collectingAndThen(  
                Collectors.toList(),  
                lista -> calcularPorcentaje(lista, duracionTotal))));  
}  
  
private Double calcularPorcentaje(List<CanalTwitch> canales, Long duracionTotal) {  
  
    Long suma = canales.stream()  
        .mapToLong(canal -> canal.getDuracionContenido().toSeconds())  
        .sum();  
  
    return suma * 100.0 / duracionTotal;  
}
```