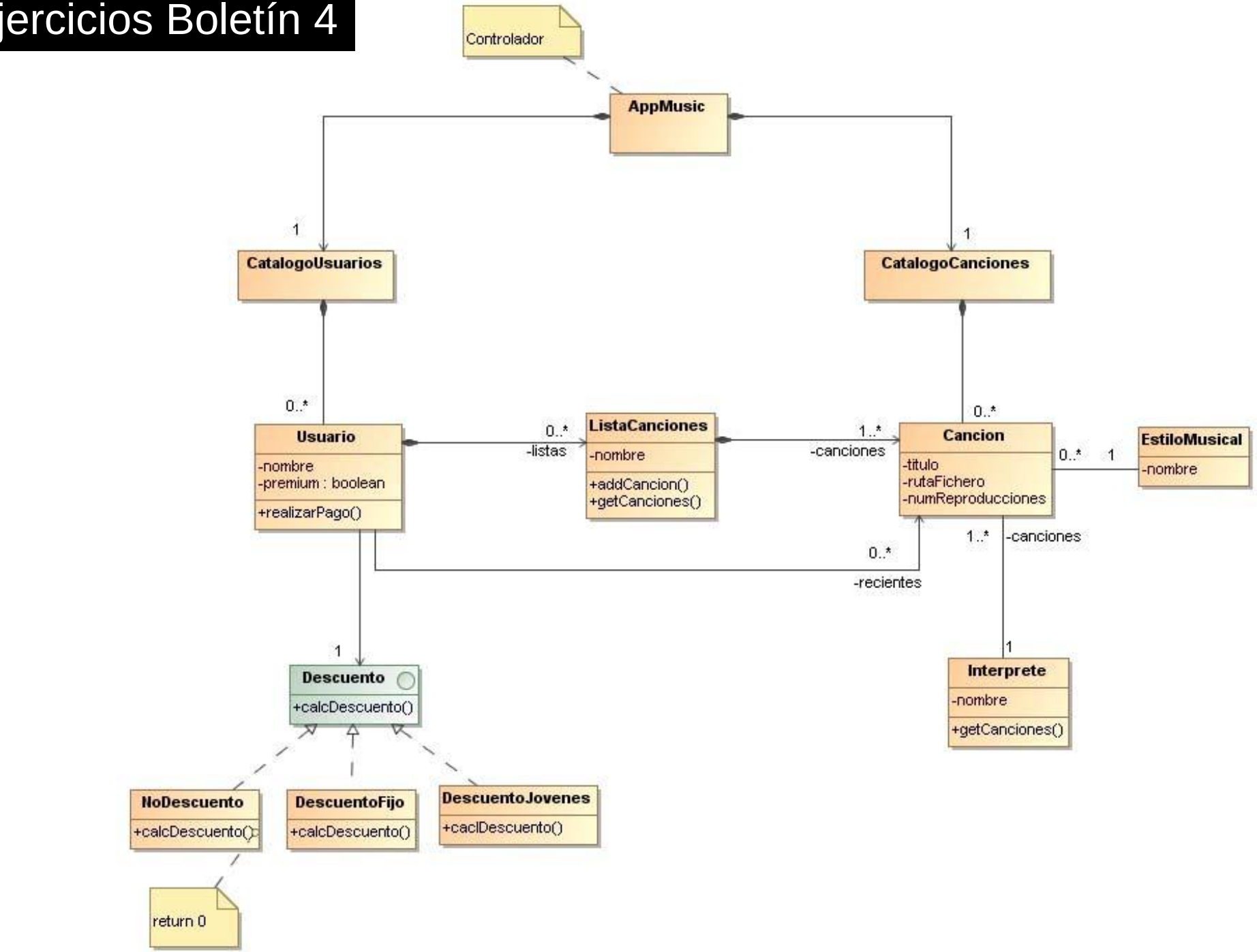
Ejercicios Boletín 4



1. Dada una lista de intérpretes, encontrar una canción cuyo título contenga “I love you” o “te quiero”

```
Cancion cancion = artistas.stream()
            .flatMap(a -> a.getCanciones().stream())
            .filter(c -> c.getTitulo().contains("I love you") ||
                    c.getTitulo().contains("te quiero"))
            .findAny()
            .orElse(null);
```

2. Dada una lista de intérpretes, comprobar si el título de alguna canción contiene “love” o “te quiero”,

```
boolean existe = artistas.stream()
            .flatMap(i -> i.getCanciones().stream())
            .anyMatch(c -> c.getTitulo().contains("love") ||
                    c.getTitulo().contains("te quiero"));
```

3. Dada una lista de intérpretes, contar el número de canciones cuyo título contiene “love” o “amor”.

```
long num = artistas.stream()
                .flatMap(a -> a.getCanciones().stream())
                .filter(c -> c.getTitulo().contains("love") ||
                             c.getTitulo().contains("amor"))
                .count();
```

4. Dada una lista de intérpretes obtener una lista con todas las canciones de un determinado estilo.

```
List<Cancion> listacanciones = artistas.stream()
          .flatMap(a -> a.getCanciones().stream())
          .filter(c -> c.getEstilo().contentEquals("rock"))
          .collect(toList());
```

5. Dada una lista de canciones mostrar por consola los títulos y nombres de interprete de las 10 canciones más veces reproducidas.

```
canciones.stream()

  .sorted(Comparator.comparing(Cancion::getNumReproducciones)
                          .reversed())
      .limit(10)
      .forEach(c -> System.out.println(c.getTitulo() + " - " +
                c.getInterprete().getNombre()
                + "-->"+ c.getNumReproducciones()));
```

6. Dado un repositorio de canciones, escribe un método que, dado el título de una canción, el nombre del intérprete y el nombre de un estilo obtenga la lista de canciones que satisfacen ese filtro. Se debe considerar que los argumentos pueden ser `null`.

```
enum RepositorioCanciones {
    INSTANCE;
    private final List<Cancion> canciones;

    public List<Cancion> buscarCanciones(String titulo,
                                         String interprete, String estilo) {
       return canciones.stream()
          .filter(c -> titulo == null ||
                       c.getTitulo().equalsIgnoreCase(titulo))
          .filter(c -> interprete == null ||
                       c.getInterprete().equalsIgnoreCase(interprete))
          .filter(c -> estilo == null ||
                   c.getEstilo().equalsIgnoreCase(estilo))
          .collect(Collectors.toList());
}
```

7. Dada una lista de usuarios, obtener una lista ordenada de los nombres de usuarios premium que han incluido un determinado título de canción en sus playlists.

```
List<String> nombresUsuarios = usuarios.stream()
    .filter(u -> u.isPremium())
    .filter(u -> u.getListas().stream()
           .flatMap(l -> l.getCanciones().stream())
                        .anyMatch(c ->
                             c.getTitulo().contentEquals(titulo))
    .map(Usuario::getNombre)
    .sorted()
    .collect(toSet());
```

8. Dada una lista de usuarios, obtener una lista con todos los nombres de los intérpretes de canciones incluidas en las playlists. La colección no debe contener elementos repetidos.

```
List<String> nombresArtistas =
          usuarios.stream()
              .flatMap(u -> u.getListas().stream())
              .flatMap(l -> l.getCanciones().stream())
              .map(c -> c.getInterprete().getNombre())
              .distinct()
              .collect(toList());
```

9. Dado el nombre de un intérprete, obtener una lista, ordenada de menor a mayor según la longitud del título, de los títulos de canciones de ese intérprete que forman parte de las playlists de **una lista de usuarios.**

```
List<String> nombresAristas2 = usuarios.stream()
        .flatMap(u -> u.getListas().stream())
        .flatMap(l -> l.getCanciones().stream())
        .filter(c -> c.getInterprete().getNombre().equals(nombre))
        .map(c -> c.getTitulo())
        .distinct()
        .sorted((t1, t2) -> Integer.compare(t1.length(),
                                            t2.length()))
        .collect(toList());
```

10. Dado un usuario obtener la suma de las duraciones de todas las canciones de un determinado interprete que están en las playlists de ese usuario.

Opción A: `sum()`

```
int suma1 = tamara.getListas().stream()
      .flatMap(l -> l.getCanciones().stream())
      .filter(c -> c.getInterprete().getNombre().equals(nombre))
      .distinct()
      .mapToInt(Cancion::getDuracion)
      .sum();
```

Opción B: `Collectors.summingInt()`

```
int suma1 = tamara.getListas().stream()
      .flatMap(l -> l.getCanciones().stream())
      .filter(c -> c.getInterprete().getNombre().equals(nombre))
      .distinct()
      .collect(Collectors.summingInt(Cancion::getDuracion))
```

Opción C: `reduce()`

```
int suma1 = tamara.getListas().stream()
   //igual que en A: flatMap + filter+distinct+mapToInt
   .reduce(0, (ac, d) -> ac + d);
```

11. Dada una lista de canciones obtener una tabla que registre para cada estilo musical el número de canciones de esa lista.

```
Map<String, Long> estiloscounter = pl2.getCanciones().stream()
    .collect(Collectors.groupingBy(
          c -> c.getEstilo().getNombre(), Collectors.counting())
    );


estiloscounter.forEach((estilo,count) ->
       System.out.println(estilo + " " + count.intValue()));
```

12. Dada la declaración **Optional<Descuento>** descuento escribe un código que muestre el valor del descuento o un mensaje indicando que no hay descuento. La clase Descuento tiene el método `getValorDescuento()`.

```
Optional<Descuento> descuento;

descuento.ifPresentOrElse(
    (d) ->
         {System.out.println("El descuento es: " +
                        d.getValorDescuento() + "%"},
    () ->
        System.out.println("No hay descuento disponible");
```

```
String mensaje = descuento
                    .map(d -> "El valor del descuento es: "
                              + d.getValorDescuento() + "%")
                    .orElse("No hay descuento disponible.");
```

13. Supuesto que el estilo musical puede no ser establecido para una canción (cardinalidad 0..1 entre `Cancion` y `Estilo`), explica cómo se utilizaría el tipo `Optional` para a) mostrar por consola el nombre del estilo de una determinada canción, b) recorrer una lista de canciones para retornar el estilo más numeroso. Un estilo registra una lista con las canciones de dicho estilo.

```
class Cancion {..
    private String estilo;
    public Optional<String> getEstilo {
          Optional.ofNullable(estilo);
    }
}
```

a)

```
cancion.getEstilo()
       .ifPresentOrElse(
          estilo -> System.out.println("Estilo: " + estilo),
          () -> System.out.println("Estilo no definido.")
       );
```

13. Supuesto que el estilo musical puede no ser establecido para una canción (cardinalidad 0..1 entre `Cancion` y `Estilo`), explica cómo se utilizaría el tipo `Optional` para a) mostrar por consola el nombre del estilo de una determinada canción, b) recorrer una lista de canciones para retornar el estilo más numeroso. Un estilo registra una lista con las canciones de dicho estilo.

b)

```
// Encontrar el estilo con más canciones
Optional<Estilo> estiloMasNumeroso = canciones.stream()
    .filter(c -> c.getEstilo().isPresent())
    .collect(Collectors.groupingBy(c -> c.getEstilo().get()))
    .entrySet().stream()
    .max(Comparator.comparingInt(entry -> entry.getValue().size()))
    .map(Map.Entry::getKey);


// Mostrar el resultado
estiloMasNumeroso.ifPresentOrElse(
    estilo -> System.out.println("El estilo con más canciones es: "
                                 + estilo.getNombre()),
    () -> System.out.println("No hay estilos con canciones.") ); }
```