Monsters of Rock - comentarios

Antes de empezar

Este programa es grande. Es más grande que cualquier cosa que hayan hecho hasta ahora.
Yo implementé hasta el punto 8, y voy por 18 clases y una interfaz.
Cada punto tiene sus bemoles, hay que agregar un atributo acá, una clase allá, varios métodos que no son triviales.

Para poder tener éxito, se recomiendan ampliamente cuatro cosas.

Primero, no hacerlo de una. Yo lo hice punto por punto. Hice el punto 1, le hice los tests, después el punto 2, los tests, y así. De hecho, el punto 4 lo hice en varias veces.
Vale hacer entregas parciales. Este ejercicio lo vamos a ir completando con el tiempo.
Hasta el punto 3 ya es una entrega interesante. Si no llego, el punto 1 solo, pero **andando y con tests**, ya es algo que se puede mandar.

Segundo, hacer diagramas. Es muy fácil perderse cuando tenés muchas clases. Tal vez les convenga hacer un diagrama separado por cada punto, que tenga solamente lo que sirve para resolver ese punto. Fíjense lo que envié de Minions, hay muchos diagramas, y no es que cada uno incluye todo lo que tienen los otros. Cada uno ataca a un punto en particular.
Va un apéndice con un programa para hacer diagramas.

Tercero, hacer tests. Tranquilo, un punto, un test, un punto, un test. Mejor menos puntos con tests, que más puntos sin tests.
Abajo van algunas recomendaciones para evitar un poco el copy-paste entre test y test.

Cuarto, pregunten. A Ariel, a mí, a Román.

Punto 1 - total de copias vendidas por una banda.

El total de copias de una banda es la suma del total de copias de cada disco.
El total de copias de un disco es la suma de las copias vendidas en cada país.
Cada disco se tiene que acordar de la cantidad de copias que se vendieron en cada país.
P.ej.: eso describe a un disco

```
// los paises
Pais argentina = new Pais("Argentina");
Pais mexico = new Pais("Mexico");
Pais chile = new Pais("Chile");
// el disco y la cantidad de copias vendidas en cada país
Disco dynamo = new Disco(1992);
dynamo.setCopiasEnPais(argentina, 150000);
dynamo.setCopiasEnPais(chile, 50000);
dynamo.setCopiasEnPais(mexico, 30000);
```

Este disco vendió 150000 copias en Argentina, 50000 en Chile, y 30000 en México.

¿Cómo hace un disco para recordar esta información?
La recomendación es crear una clase que se puede llamar `CopiasVendidas`, que simplemente se acuerde de un país y de una cantidad.
Entonces cada disco tiene como atributo una colección de `CopiasVendidas`. La definición del atributo queda así

```
private Collection<CopiasVendidas> copiasPorPais
                                = new HashSet<CopiasVendidas>();
```

Para calcular las copias vendidas de un disco, alcanza con hacer el mapToInt de la cantidad de cada elemento de esta colección, y sobre eso hacer la suma.
Esto es análogo al item de pedido que vimos en el ejercicio de mercadería. Los que se acuerden de Base de Datos, es como las tablas que se agregan para una relación.

Punto 2 - último disco editado por una banda.

Este se resuelve usando max sobre la colección de discos de la banda.
Ahora, el closure que le pasan al map como parámetro tiene dos parámetros, no uno. Este closure es un ***comparador***, esto está explicado en el apunte de colecciones en Wollok que usamos el año pasado.
La más directa es hacerlo como lo indica Román en su apunte mágico.
Para quien tenga ganas de entender qué pasa ahí, meto un apéndice al final sobre comparadores.

Punto 3 - bandas de los eventos en una sede

Para esto hay que lograr que cada sede conozca qué eventos se hicieron.
Pero OJO que también cada evento necesita conocer en qué sede se hizo.
Queda un “doble enganche”: la sede tiene un atributo que es una colección de eventos, el evento tiene un atributo que es una sede.
Por ahora, vale que en el test diga algo como

```
megaFestival.setSede(lunaPark);
lunaPark.agregarEvento(megaFestival);
```

Después vemos cómo manejar esto de forma elegante.

También hay que registrar las bandas que van a participar en cada evento.
En este punto alcanza con una colección de bandas. Pero ya en el punto siguiente, para algunas validaciones van a necesitar también el disco y la duración.
Para manejar eso, creo que conviene crear una clase, p.ej. BandaEnEvento, que “tenga” una banda, un disco y un int que es la duración.

Puntos 4 y 5 - saber si se puede agregar una banda a un evento, y agregarla (o lanzar error).

Hay muchas condiciones:

1. Una que no dice pero es obvia: el disco que presenta una banda debe ser de esa banda. P.ej. Soda Stereo no puede agregarse a un evento presentando un disco de Virus.
2. Si el evento lo organiza una discográfica, entonces el disco que se presenta tiene que haber sido producido por la misma discográfica.
3. Si el evento lo organiza una ciudad, entonces la banda tiene que ser del mismo país de esa ciudad.
4. Un festival no puede durar más de 720 minutos = 12 horas. O sea, si sumando la duración de las bandas inscriptas más la que se quiere agregar se superan las 12 horas, no vale agregarla.
5. Un festival define una colección de géneros, y la banda tiene que ser de uno de esos géneros.
6. Para un festival, si ya tiene asignada la banda de cierre (que es lo mismo que “principal”) y se quiere agregar otra, la que se agregue tiene que tener un total de copias vendidas menor que la principal.
7. Parecido para recitales, pero la banda principal tiene que tener por lo menos el triple de copias vendidas que las otras.
8. Las bandas soporte de un recital tienen que ser del mismo género que la principal.
9. Un recital no puede tener más de cuatro bandas: la principal y tres soportes.
10. Una cena-show no puede tener más de una banda.

Aclaración: no es necesaria una clase Genero, se pueden manejar con Strings.

En un festival y en un recital, la banda principal es la primera que se agrega. O sea, si a un festival o recital se le pregunta si puede agregarse una banda, y no se le agregó antes ninguna banda, entonces la banda por la que se pregunta es para ponerla como principal. Si el festival o recital ya tiene bandas, entonces ya tiene una principal, entonces hay que tener en cuenta las condiciones 5 y 6.

Pero ¡OJO! el mensaje es uno solo, no hay un mensaje p.ej.

```
puedeAgregarBandaPrincipal(Banda banda, Disco disco, int duracion)
```

y otro distinto

```
puedeAgregarBandaNoPrincipal(Banda banda, Disco disco, int duracion)
```

Tiene que haber un único mensaje p.ej.

```
puedeAgregarBanda(Banda banda, Disco disco, int duracion)
```

y dentro del método, darse cuenta si la banda sería la principal o no.
Los tres tipos de evento tienen que ser polimórficos para este mensaje.

Lo mismo para agregar una banda, no vale que haya dos métodos
`agregarBandaPrincipal`
y `aregarBandaPrincipal`, tiene que haber uno solo:

```
agregarBanda(Banda banda, Disco disco, int duracion)
```

que también tiene que manejarse en forma polimórfica entre los tres tipos de evento. Si es festival o recital y no tiene ninguna banda, es la principal; si no, no.

Una aclaración para los items 2 y 3. Cada evento tiene un organizador. El organizador puede ser una discográfica o una ciudad. Por lo tanto, las discográficas y las ciudades tienen que ser polimórficas para los eventos. Si esto no sale, dejen estas validaciones para más adelante.

Creo que conviene pensar estos dos ítems en conjunto. Les tiro una idea de cómo manejarse

- Primero el punto 5, o sea agregar banda, sin validaciones. Con esto resuelven lo de agregar como principal o no. Testeen p.ej. agregar tres bandas a un festival, y validen que la primera que agregaron es la principal, y que tiene 3 bandas, y p.ej. la duración total del festival.
  OJO acá, el punto 3 tiene que seguir andando. Ahí hay que considerar todas las bandas, la que es principal y las que no. Consejo: hacer que todos los eventos entiendan
  ```
  getBandas()
  ```
  La CenaShow devuelve una colección con una sola banda.
  A propósito, para esto chusmeen los mensajes `Collections.singletonList` y `Collections.singleton`.
- Después el punto 4, pero tomando las validaciones de a una, sobre todo si estoy mareado. Hago una, la pruebo. Hago otra, la pruebo.
- En algún momento, agregar la validación al punto 5. Para lanzar un error, la más fácil es
  ```
  throw new RuntimeException("mensaje");
  ```
  El miércoles vemos cómo validar que hay una excepción en un test.
  Lamentablemente es más difícil que en Wollok (como casi todo).

Yo dejé las validaciones por organizador para más adelante, las hice junto con los puntos 7 y 8. O sea, también vale hacer algunas validaciones, una entrega parcial, y después las otras.

Punto 6 - saber si un evento incluye un género

Este no es difícil. Conviene preguntarle a un evento si incluye o no un género, no que te devuelva la colección de géneros.

Punto 7 - costo de alquiler de una sede

Este creo que tampoco es muy difícil. Conviene tener antes el 6, para resolver el descuento del 20%. La condición es que el evento tiene que incluir alguno de los eventos en promoción del anfiteatro, o sea, un any sobre los eventos en promoción.

Punto 8 - ingresos y gastos

Acá va un Template Method de cabeza. La condición es la misma para los tres tipos de evento: ingresos > gastos. Lo que cambia de acuerdo al tipo de evento es cómo se calculan los ingresos y los gastos.

Algunos comentarios particulares

- Para los festivales, otra vez clase aparte, p.ej. `AporteAFestival`, que "tiene" un auspiciante y el monto.
- Las empresas auspiciantes vale manejarlas como String.

- Por más que el monto máximo por entradas lo usan solamente para las cena-show, creo que conviene ponerlo para todos los eventos, la cuenta es la misma.

Puntos 9, 10, 11

Próximamente (si sigo escribiendo, no corrijo).

Tests - algunas ideas

Creo que cuanto más se complica un enunciado, más conveniente es ir llevando tests. A mí al menos, me ayuda a aclararme ideas de diseño (en qué objeto tengo que poner qué cosa).

Para armar cada test, puede que necesiten muchos objetos. Para no tener que poner el código de la creación y confiugración de estos objetos en cada test, usé el @Before que tiene JUnit 4. Si le ponés @Before a un método, se va a ejecutar antes de cada test.
Queda algo así:

```
public class BandaTest {
	protected Pais argentina = new Pais("Argentina");
	protected Pais mexico = new Pais("Mexico");
	protected Ciudad mendoza = new Ciudad("Mendoza", argentina);
	protected Ciudad acapulco = new Ciudad("Acapulco", mexico);

	protected Banda soda = new Banda(argentina, "pop", 150000);
	protected Disco signos = new Disco(1986);
	// etc.

	@Before
	public void configure() {
		soda.addToDiscos(signos);
		signos.setCopiasEnPais(argentina, 120000);
		signos.setCopiasEnPais(chile, 25000);
		// etc.
	}

	@Test
	public void testDiscosVendidos() {
		assertEquals(145000, soda.getTotalCopiasVendidas());
	}
}
```

Fíjense que los objetos que se crean quedan como atributos del objeto. Después se pueden usar en los tests.

Si quiero tener muchas clases de tests, y tener una inicialización compartida entre todas ellas, una opción fácil es definir un MonstersTest, poner la inicialización (atributos y método @Before) en esa clase, y que las clases de test hereden de MonstersTest.
Haciendo así, todo el código del ejemplo anterior salvo el test quedaría en MonstersTest, y después tendríamos

```
public class BandaTest extends MonstersTest {
	@Test
	public void testDiscosVendidos() {
		assertEquals(145000, soda.getTotalCopiasVendidas());
	}
}
```

Apéndice: comparadores en Java

Va una pequeña explicación sobre los comparadores en Java.
Un comparador recibe dos elementos de la colección (en este caso, dos discos) y devuelve un número que indica cuál es el más grande. Hay que devolver:

- -1 si el primero es más chico que el segundo.
- 0 si son iguales.
- 1 si el primero es más grande que el segundo.

P.ej. para obtener la persona de más edad en una colección:

```
Persona elMasViejo = personas.stream().max((p1,p2) ->
        if (p1.getEdad() < p2.getEdad()) {
                return -1;
        } else if (p1.getEdad() > p2.getEdad()) {
                return 1;
        } else {
                return 0;
        });
}
```

Apéndice: para hacer diagramas de clase

Si se sienten cómodos haciéndolos a mano, excelente.
Yo me cansé y busqué alguna herramienta piola. Apareció un editor rarísimo, que no hay que instalarlo, se usa desde el browser. Miren en

http://www.umlet.com/umletino/umletino.html

Lo más raro que tiene es que los detalles de una caja o una flecha se editan abajo a la derecha. Pero una vez que me acostumbré, es lo más ágil que haya usado.
También es raro cómo se graba. Si le das “save”, lo graba en un espacio administrado por el mismo browser. Para grabarlo en disco, están los “file export”. Se puede grabar a un archivo XML, que después levanta perfectamente con el “file import”. Pero OJO en el “file export”, cuando te da las opciones de diagrama (el archivo XML) o imagen (un .PNG), lo que hay que hacer no es clickear, sino botón derecho y “guardar enlace como …” (o “save as…” si tenés el browser en inglés).

También tiene una versión que se instala. A mí me alcanzó con esta.