Homework 4 – Battle of the Bands

Authors: Allan Nguyen, Nathaniel Gregory, Melanie Su, Sumit Choudhury

Problem Description

This assignment will test your knowledge on abstract classes, inheritance, and polymorphism.

You are a big fan of musical bands, whether that be 1970s rock bands or 2010 K-pop bands. As an ultimate fan of many bands, you have given each a talent rating. You seek the answer to the question: who is the best band? To answer this question, you decide to use your handy 1331 programming skills to create a battle of the bands simulation to figure out the best band of all time!

The program will have a Band class representing an individual band and containing BandMember objects, each of which is a Singer or Guitarist. The driver class will run the simulation, add bands and band members (artists), modify their ratings, and have the bands battle. We will provide the driver class for you that has the skeleton code, but you need to fill in the rest!

Solution Description

BandMember.java

- Every BandMember should have the following attributes:
  - A string variable `'name'` that represents the name of the member.
  - An int variable `'talent'` that represents a score out of 5 on how much talent they have.
- Should have constructor for `BandMember` that takes in name and talent in that order. If provided a talent score below 1, assign 1. If provided a talent score above 5, assign 5.
- BandMembers are not concrete objects, meaning you should not be able to directly create an instance of one.
- A band member should have an abstract `perform()` method which returns an `int` representing a score out of 10 on their performance.
- When you print out a BandMember object, it should return "<name>: <talent>/5"
- Should have an `equals` method that overrides Object's equals. Two BandMembers are equal if they share the same name and talent.
- Should have a getter method for name, setter for talent.

Singer.java

- A singer has an is-a relationship with BandMember.
- They have a Range variable `'range'`. It can only be BASS, BARITONE, TENOR, ALTO, or SOPRANO. Think about what you can use to represent only these values!
- Should have constructor for `Singer` that takes in name, talent, and range in that order.
- When a singer performs, they will get a random int value from 1 to 10. Their talent rating is added to this random value to form their performance score. A singer has a 10% chance of hitting a high note, in which their score will double! The max score is a 10.
  - If their score is below 5, print out "<name>'s voice cracked in the middle of their performance! Score: <score>/10"
  - If their score is above 5, print out "<name> sang their heart out! Score: <score>/10"
- When you print out the Singer object, it should return "<name>: <talent>/5 <range>" For example, "John Lennon: 5/5 BARITONE"
- Should have an `equals` method that overrides Object's equals. Two singers are equal if their name, talent, and vocal ranges match (think about how you can reuse code). Otherwise, if a singer is compared with a non-singer band member, return false.

Guitarist.java

- A guitarist has an is-a relationship with BandMember.
- They have a Specialty variable 'specialization'. It can only take on the values ACOUSTIC or ELECTRIC
- Should have constructor for `Guitar` that takes in name, talent, and specialization in that order.
- When a guitarist performs, they will get a random int value from 1 to 10. Their talent rating is added to this random value to form their performance score. A guitarist has a 30% chance of a successful guitar solo, in which their talent rating will add to their score a second time. The max score is a 10.
    - If their score is below 5, print out "<name>'s fingers slipped and hit the wrong note! Score: <score>/10"
    - If their score is above 5, print out "<name> captured everyone's attention! Score: <score>/10"
- When you print out the member, it should return "<name>: <talent>/5 <range>" For example, "Jimi Hendrix: 5/5 ELECTRIC"
- Create an `equals` method that overrides Object's equals. Two guitarists are equal if their name, talent, and specialization match (think about how you can reuse code). Otherwise, if a guitarist is compared with a non-guitarist band member, return false.

Band.java

- A band should have a band name `'bandName'`.
- The band should have a String representing its `'genre'`.
- A band can hold band members in an ArrayList `'members'`.
- Should have constructor for `Band` that takes in name, genre, and members.
- Should have a method called `talentRating()`. This will compute and return the average talent rating of the band members as a double (do not worry about rounding). If there are no band members, it should return 1.
- Should have a method called `perform()`. This will call each band member's perform method then compute and print out the average performance score across all the band members in the format: "<Band Name>'s Performance Score: <score>/10". Finally, the method should return the average score as a double (do not worry about rounding).
- The band should have a `toString` method. Think about how you can utilize existing code! It should return in the format:
  "Queen (Classic Rock): 4.5/5
  Freddie Mercury: 5/5 BARITONE
  Brian May: 4/5 ELECTRIC"
- Should have a getter method for members and name

BattleSim.java

- This is the driver class for the program where we have the command-line interface. When the user runs the program, they have several options to interact with the program: add new artists/bands, change ratings, and perform simulation. Each of these options has its own method where the format and user prompt are done, but you need to achieve the functionality. We have provided the skeleton code, but you will need to fill in where it says "TODO".
- The BattleSim object holds an ArrayList of Bands. This is what you will need to use to add new bands and artists.
- You do not need to worry much about formatting in this class as we have done that. Simply fill in code to achieve the functionality!
- **NOTE: THIS CLASS WILL NOT COMPILE INITIALLY.** You'll need to create the other classes and methods before it will compile.

Advice / Important Notes

- Get your BandMember, Singer and Guitarist classes implemented, tested, and working first. Then work on the Band class and the driver. Incremental development and testing like this can make the program much more manageable.
- When running Checkstyle locally, you will get an error that says “Definition of 'equals()' without corresponding definition of 'hashCode()'“. Do not worry, we will not take points off for this error. Any other Checkstyle errors must be fixed.

Example Output

```
$ java BattleSimulator
Welcome to Battle of the Bands Simulator!

Would you like to:
1. View bands
2. Add new artist
3. Add new band
4. Change artist rating
5. Perform simulation
6. Exit program
Please enter one of the options above:
1

1. Queen
2. BTS
What band would you like to view: 2
BTS (K-Pop): 5.0/5
Jungkook: 5/5 TENOR

Would you like to:
1. View bands
2. Add new artist
3. Add new band
4. Change artist rating
5. Perform simulation
6. Exit program
Please enter one of the options above:
2

1. Queen
2. BTS
Which band would you like to add a member to: 1
What is the artist's name? Freddie Mercury
Talent rating from 1 to 5: 5
Singer or Guitarist: Singer
1. BASS
2. BARITONE
3. TENOR
4. ALTO
5. SOPRANO
Which range: 2
Added singer Freddie Mercury to Queen

Would you like to:
1. View bands
2. Add new artist
3. Add new band
4. Change artist rating
5. Perform simulation
6. Exit program
Please enter one of the options above:
5
```

```
Setting up simulation...
1. Queen
2. BTS
Select the first band: 1
Select the second band: 2

Beginning simulation...
Brian May captured everyone's attention! Score: 7/10
Freddie Mercury sang their heart out! Score 10/10
Queen's Performance Score: 8.500000/10
Jungkook sang their heart out! Score 10/10
BTS's Performance Score: 10.000000/10
The winner is BTS

Would you like to:
1. View bands
2. Add new artist
3. Add new band
4. Change artist rating
5. Perform simulation
6. Exit program
Please enter one of the options above:
6
$
```

Submitting

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `BandMember.java`
- `Singer.java`
- `Guitarist.java`
- `Band.java`
- `BattleSim.java`

Make sure you see the message stating, "Homework 4 submitted successfully." We will only grade your last submission be sure to **submit *every file* each time you resubmit.**

Import Restrictions

You may only import Scanner, Random, and ArrayList.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Checkstyle and Javadoc

You must run Checkstyle on your submission. (To learn more about Checkstyle, examine our course's external website Java Resources page under "CS 1331 Style Guide".) The Checkstyle cap for this assignment is **10 points**. If you don't have Checkstyle yet, download it from Canvas → files → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
        $java -jar checkstyle-8.28.jar yourFileName.java
```

```
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.
Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

Collaboration

No collaboration is allowed on this assignment. See syllabus for more details.

In addition, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due. Only post code on Piazza in a **private** post.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications
- Make sure to test your program manually based on the assignment instructions and expected outputs.

The Gradescope autograder visible to you is **NOT** comprehensive. A few test cases will be hidden when you submit. This is done intentionally so that you learn to write your own test cases for your assignments.