Burdell and the Buzz

Problem Description

As an aspiring band manager, you need to promote your band and expand publicity. Being hip with the times, you know that social media can make or break a band's popularity. So, you decide to bring some 'lucky' Georgia Tech students to spread the good word!

Solution Description

Write the Concert, Musician, and Fan classes to guide your band on their rock star journey. You will design these classes from scratch following the instructions below, so no files are provided.

Note: When creating the specified getter and setter methods for each class, use the naming convention taught in class, e.g. getTicketPrice() and setDate().

Concert.java

Fields

This class has the following private fields, and **associated getter methods**:

- `double ticketPrice`. General Admission ticket price.
- `int capacity`. Venue capacity.
- `int ticketsSold`. Number of tickets sold.
- `String location`. Location of the concert.
- `String date`. The date of the concert in the format MM/DD/YYYY. For example, "02/14/2019".

Constructor

This class has the following constructors:

- There are two constructors (hint: use constructor chaining)
- The first constructor takes the following four arguments and assigns them to instance variables **in the order listed**:
  - Ticket price
  - Capacity
  - Location
  - Date
  - Do not include `ticketsSold` as a parameter; rather, set its field to **0** within the constructor.
- The second constructor takes the following three arguments and assigns them to instance variables **in the order listed**:
  - Capacity
  - Location
  - Date
  - Do not include `ticketsSold` as a parameter; rather, set its field to **0** within the constructor. Additionally, do not include `ticketPrice` as a parameter; rather, set its field to **30** within the constructor.

Methods

This class has the following public methods:

- `boolean isSoldOut()`. Using the capacity and ticketsSold fields, return whether or not this concert is sold out.
- `void sellTicket()`. Increment the counter for the number of tickets that have been sold. If the concert is already sold out, you should not change the number of tickets that have been sold.

- `String toString()`. Returns a String in this format:

```
A concert on <date> at <location>
```

- Setters for both `location` and `ticketPrice`.

Musician.java

Fields

This class has the following private fields, and **associated getter methods**:

- `String name`. Name of the musician.
- `String instrument`. Name of the musician's weapon of choice.
- `int yearsPlaying`. Number of years the musician has been playing the instrument.
- `double skillLevel`. Related to `yearsPlaying`, more to explain shortly.

Constructor

This class has the following constructors:

- There are two constructors (hint: use constructor chaining)
- The first constructor takes the following three arguments and assigns them to instance variables **in the order listed**:
  - Name
  - Instrument
  - Years Playing
- The other constructor takes two parameters **in the order listed** and sets 0 as the value for `yearsPlaying`:
  - Name
  - Instrument
- The `skillLevel` instance variable is determined inside the constructors by the value for `yearsPlaying`. At 0 years played, `skillLevel` is **1.0**. Add **0.5** to `skillLevel` for every additional year played.

Methods

This class has the following public methods:

- `void rehearse()`.
  - Increases `skillLevel` by 0.5 and `yearsPlaying` by 1 when called.
- `void perform()`.
  - Increases `skillLevel` by 1 when called.
- `String toString()`.
  - Returns a string that prints out the musician's name, what instrument they play, and how long they've been playing for. (No need to worry about "year" vs. "years" here)
  - For example:

```
My name is Taylor Swift. I have been playing guitar for 10 years.
```

Fan.java

Fields

This class has the following private fields, and **associated getter methods**:

- `int yearsAsFan`. How many years this person has been a fan of the band.
- `int albumsBought`. Number of albums that this fan has bought.
- `int concertsAttended`. Number of concerts this fan has attended.
- `boolean buzzcard`. Boolean representing whether or not this fan has their Buzzcard.

- `Musician favoriteMusician`. This fan's favorite musician.

Constructor

This class has the following constructor:

- The first constructor takes the following five arguments and assigns them to instance variables **in the order listed**:
  - Number of years as a fan
  - Number of albums bought
  - Number of concerts attended
  - Whether or not they have a Buzzcard
  - Their favorite musician

Methods

This class has the following public methods:

- `boolean winGiveaway()`.
  - This method returns whether or not the fan wins the 'giveaway'. However, since we want fans to attend, this will always return true! (Even though they win, they still need to pay for a ticket–your band needs the revenue..)
- `String liveTweet(Concert concert)`. Livetweeting signifies that the Fan has attended a concert. This method will return a String based on the passion level of the fan. Specifically, if the corresponding conditional is true, add to the tweet each time *on a new line and in the order presented*:
  - If the fan has been following the band for over 5 years, the tweet will include: "Best band ever!"
  - If the fan paid more than $100 for a ticket, the tweet will include: "Totally worth my entire bank account!"
    - (Hint: you created a getter method for ticket price within the Concert class, and you passed in a Concert parameter to this method)
  - If the fan has bought at least one album, the tweet will include: "Even better in person!"
  - At the end of every live tweet, it will always include: "I've been to <number of concerts> concerts!" The fan assumes that *this* concert does count towards the total number of concerts they have attended in this statement. Update any necessary variables accordingly.
    - Use a ternary statement to indicate the word "concert" if this is the fan's first concert, and the word "concert**s**" otherwise. Proper grammar is essential for any successful manager.
  - For example, if we have a Fan named Joe Schmo who has been a fan for 10 years, paid $200 for a ticket, bought 5 albums, and has been to 4 concerts (including this one), the tweet should read:

```
Best band ever!
Totally worth my entire bank account!
Even better in person!
I've been to 4 concerts!
```

- `void lostBuzzcard()`. Every Georgia Tech student needs to keep careful track of their Buzzcard! But, sometimes we get a little reckless.
  - If the fan has been following the ban for over 3 years, set the boolean `buzzcard` to false.
- `void announceFavoriteMusician()`. This method should print:

```
My favorite musician is <name of Musician>!
```

ConcertSimulator.java

This class is purely an **example** of what an output could look like. Its purpose is for you to test your code to produce the correct provided output. **You do not have to submit this file.** - Create 2 Musicians - Christopher W. Klaus has been playing violin for 30 years - UGA has been playing students for 234 years - Create 1 Fan - Gerald Clough - 14 years as a fan, 8 albums bought, 52 concerts attended, has his Buzzcard, and his favorite musician is Christopher W. Klaus - Announce Gerald Clough's favorite Musician - Create 1 Concert - Ticket price is $500 - Venue capacity is 60 seats - Location is Mercedes-Benz Stadium - Date is 02/03/2019 - Sell 60 tickets then see whether or not the concert is booked - If it is, print "Sorry! [details of concert] is fully booked!" - Is there a way to print the details of the concert without hardcoding? - Print out Gerald Clough livetweeting (Note: This increases his total number of concerts attended by one) - The sample output is as follows:

```
My name is Christopher W. Klaus. I have been playing violin for 30 years.
My name is UGA. I have been playing students for 234 years.
My favorite musician is Christopher W. Klaus!
Sorry! A concert on 02/03/2019 at Mercedes-Benz Stadium is fully booked!
Best band ever!
Totally worth my entire bank account!
Even better in person!
I've been to 53 concerts!
```

Allowed Imports

To prevent trivialization of the assignment, you are *not* allowed to import any classes or packages.

If you would like to import anything, ask on Piazza.

Feature Restrictions

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

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

Rubric

[35] **Concert**

- [5] `Concert` has correct fields.
  - [1] per field
- [5] `Concert` has getters for each field.
  - [1] per getter
  - Each getter must have correct name and must work to get the point.
- [10] `Concert` has correct constructors
  - [2.5] `ticketPrice`, `capacity`, `location`, and `date` constructor exists and functions
  - [2.5] `capacity`, `location`, and `date` constructor exists and functions
  - [5] Proper use of constructor chaining
- [15] Other methods
  - [3] `isSoldOut` exists and functions properly
  - [3] `sellTicket` exists and functions properly
  - [3] `toString` exists and functions properly
  - [3] `setLocation` exists and functions properly
  - [3] `setTicketPrice` exists and functions properly

[30] **Musician**

- [4] `Musician` has correct fields.
  - [1] per field
- [4] `Musician` has getters for each field.
  - [1] per getter
  - Each getter must have correct name and must work to get the point.
- [10] `Musician` has correct constructors
  - [2.5] per constructor for existing and functioning
  - [5] Proper use of constructor chaining
- [12] Other methods
  - [3] `rehearse` exists and functions properly
  - [3] `perform` exists and functions properly
  - [6] `toString` exists and functions properly

[35] **Fan**

- [5] `Fan` has correct fields.
  - [1] per field
- [5] `Fan` has getters for each field.
  - [1] per getter
  - Each getter must have correct name and must work to get the point.
- [5] `Fan` has correct constructor
  - [5] Constructor exists and functions
- [20] Other methods
  - [2] `winGiveaway` exists and functions properly
  - [14] `liveTweet` exists and functions properly
    - [3] Correct logic and format for printing "Best band ever!"
    - [3] Correct logic and format for printing "Totally worth my entire bank account!"
    - [3] Correct logic and format for printing "Even better in person!"
    - [1] Correctly uses "concert" v/s "concerts"
    - [2] Uses ternary statement to determine whether or not to use "concert" or "concerts"
    - [2] liveTweet prints statements in the correct order
  - [2] `lostBuzzcard` exists and functions properly
  - [2] `announceFavoriteMusician` exists and functions properly

Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points. Review the style guide and download the checkstyle jar. Run checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar *.java
Audit done. Errors (potential points off):
0
```

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!

Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips page for more information.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Turn-In Procedure

Submission

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

- Concert.java
- Musician.java
- Fan.java

Make sure you see the message stating "HW## submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1) Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2) Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment are also autograded once the submission deadline has passed. The autograders used are often dependent on specific output formats, so **make sure that your submission passes all test cases marked "FORMAT:"**.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Ensure you pass all "FORMAT:" tests
  - Be sure to follow the **exact** formatting as specified in the assignment

  - This includes case sensitivity, newline characters, and word for word Strings
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications