CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 3

Selections and Methods in Java

Due: Wednesday, April 26, 11:59 PM

Learning Goals:

- Write Java code that includes:
  - Selections
  - Methods
- Write your own test cases to test the correctness of your methods

NOTE: This assignment should be completed INDIVIDUALLY.

Coding Style (10 points)

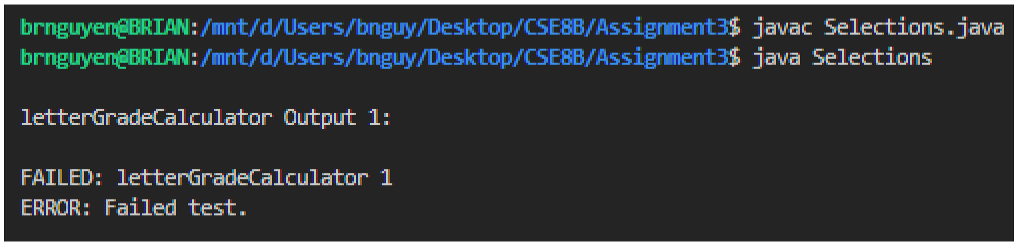
For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Get Started with Starter Code (0 points)

1. If using a personal computer, then ensure your Java software development environment does not have any issues. If there are any issues, then review Assignment 1, or come to the office/lab hours before you start Assignment 3.
2. First, navigate to the `cse8b` folder that you have created in Assignment 1 and create a new folder titled `assignment3`
3. **Download the starter code.**
   You can download the starter code from Piazza → Resources → Homework → `Assignment3.zip` and **unzip** it.
   a. Once you have downloaded the zip file, double click on it from the Finder (for Mac) or File Explorer (for Windows).

   b. A folder should be created in the same location as the zip file. You can find the files with the starter code within that folder.

4. The starter code should only be two files called `Selections.java` and `ZooManager.java`. Place the starter code within the `assignment3` folder that you have just created.
5. Compile and run the starter code, and you should expect the following output:
   For `Selections.java`,

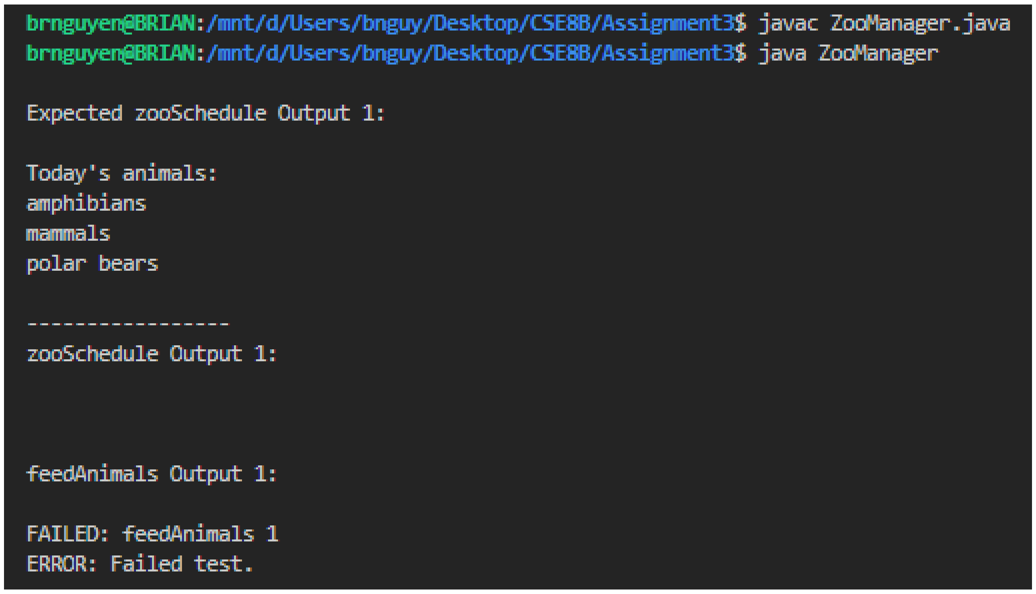


```
brnguyen@BRIAN:/mnt/d/Users/bnguy/Desktop/CSE8B/Assignment3$ javac Selections.java
brnguyen@BRIAN:/mnt/d/Users/bnguy/Desktop/CSE8B/Assignment3$ java Selections

letterGradeCalculator Output 1:

FAILED: letterGradeCalculator 1
ERROR: Failed test.
```

   For `ZooManager.java`,



```
brnguyen@BRIAN:/mnt/d/Users/bnguy/Desktop/CSE8B/Assignment3$ javac ZooManager.java
brnguyen@BRIAN:/mnt/d/Users/bnguy/Desktop/CSE8B/Assignment3$ java ZooManager

Expected zooSchedule Output 1:

Today's animals:
amphibians
mammals
polar bears

-----------------
zooSchedule Output 1:



feedAnimals Output 1:

FAILED: feedAnimals 1
ERROR: Failed test.
```

Although we have not covered Java classes in lecture yet, you can think of Java classes as an isolated entity that contains methods. The starter code is set up so you don't need to know anything about the details of creating a class from scratch. Your job is to just fill in the starter code with methods that have been explained below. *(We have mentioned the places where you need to fill in the code)*

Part 1: Practice Selections (40 points)

In `Selections.java` starter code, there are three methods declared for you that will help you get familiar with selections. You will need to fill in the logic to complete both methods: `letterGradeCalculator`, `accountCreation`, and `sortThreeStrings`

Before you implement your methods, please take a look at the constants at the top of the class. All of the error messages as well as method-specific variables are already defined for you with the keywords `private final static`. **IMPORTANT: You should use these constants when developing your methods to ensure that your code will always give the correct output. You technically do not have to but using them will help you remain consistent.**

1. `letterGradeCalculator` (8 points):

This method calculates a grade for an imaginary course. It takes in five `double` variables: `score1, score2, score3, score4, and finalScore`, which represent the scores received on four assignments and the final exam, respectively. This method will compute the letter grade associated with the **average** of the **top three assignment scores and the final exam score** (note that one assignment score is dropped). All scores have equal weight. Follow the rules of our letter grade calculator below, and assign your result to the variable `letterGrade`.

- All scores must be a **valid** score: greater than or equal to 0 and less than or equal to 100. If any of the scores violate this condition, return `"Invalid set of scores"`.
- **Drop** the **lowest assignment score** before calculating the average of the four remaining scores (three highest assignment scores + final exam score).
- The grading scheme is simple and `letterGrade` should contain the grade associated with the following cutoffs when the method returns:

Grade	**Range**
A	90 to 100
B	80 to 90
C	70 to 80
D	60 to 70
F	Below 60

- **Note:** the minimum value of a range is **inclusive**, while the maximum value of a range is exclusive (meaning that a score of 80 is considered a `"B"`, but a score of 90 is an `"A"`
- Lastly, this imaginary course is strict and if you fail the final **(getting strictly below a score of 60)** you fail the course, so return "F".😞

Parameters: `double score1, double score2, double score3, double score4, double finalScore`
Return type: `String`
Example: `letterGradeCalculator(95,50,87,79,75)` → `B`
Explanation:
Here, we have four assignment scores (95, 50, 87, 79) and one final exam score (75). We drop the lowest assignment score (50), and then compute the average of the four remaining scores (the three assignments and the one final exam). That is, (95 + 87 + 79 + 75) / 400 = 84%. This gives us the letter grade, B.

2. `accountCreation` (12 points):

This method is a simple validator for creating fake accounts. It takes in a `String email and a String` password, and it will return a `String` message, letting us know if the account was created successfully or if there is a problem with our email or password. **Note:** it is safest to use the constants provided to guarantee you pass the test cases. You can assume that both strings will not be empty. Both the email and password **must abide** by the following rules **in this order of precedence** (meaning that the email is more important than the password so if both strings are wrong, output the email error message or if the password has multiple violations output the error message for the violation that is **higher on this list**):

Email

- This method will only accept emails that end with the following domains:
  - `"@ucsd.edu"`, `"@gmail.com"`, `"@yahoo.com"`, `"@outlook.com"`.

Password

- Password must be **at least 9** characters
- The password must **start with** a capital letter
- The password must **end with** a digit
- The password must **not contain** the name of the email address.
  - If the email address was `"spiderman@gmail.com"`, then `"spiderman"` should not be part of the password. This check is **case-insensitive**, meaning that if

`"Spiderman@gmail.com"` was the password, `"spiderman"` or `"Spiderman"` would be an error.

  - ***Hint***: the **contains()** method may be useful here. Learn more about it from [Java documentation about String methods]

If the email address and password do not break any of these rules, then the method should return the String: `"Account created successfully"`

Parameters: `String, String`
Returns: `String`
Example: `accountCreation("spiderman@ucsd.edu", "Spidermanrox1")` → `"Password must not contain the username"`

3. `sortThreeStrings` (12 points):

This method will take in three `String` variables (assume the strings will not be empty) and return a comma separated `String (there is a space after each comma)` with the three variables **lexicographically sorted**. This means that strings should be sorted based on the order of their characters, similar to how words are arranged in a dictionary.

For example, let's consider a list of strings: ["Apple", "banana", "apple"]. If we sort this list lexicographically, the resulting sorted list would be: ["Apple", "apple", "banana"]. In lexicographical order, we start from the leftmost character to the rightmost character. Lexicographical sorting considers the ASCII or Unicode values of characters, so uppercase letters come before lowercase letters, and if two words are identical up to the length of the shorter word, the shorter word comes first. (***Hint***: we have learned a method that already takes into account lexicographical order when *comparing* strings)

(**Note:** you should only use the concepts we've covered so far in lecture to create this method - as mentioned in the starter code: **there should be no added imports**)

Parameters: `String, String, String`
Returns: `String`
Example: `sortThreeStrings("lol", "bye", "hi")` → `"bye, hi, lol"`

Part 1.1: Test Correctness of Selections Methods (8 points)

Similar to the previous assignment, you will be testing your code. **In this part of the assignment, you need to implement your own test cases in the method called `unitTests`.**

In the starter code, several test cases are already implemented for you. You can regard it as an example to implement other cases. The general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method should return `true` only when all the test cases are passed. Otherwise, it should return `false`. **To get full credit for this section, for each method, you should have at least three total test cases that cover different situations (including the ones we have provided).** In other words, you will need to create two more tests for `letterGradeCalculator`, two more tests for `accountCreation`, and two more tests for `sortThreeStrings.`

Part 2: Implement ZooManager.java (50 points)

The local San Diego Zoo has asked our CSE8B class for help. They need some methods to help them manage different aspects of the zoo. Being the great programmers we are, we created a ZooManager class to hold all our zoo-related methods. You need to implement three methods that can help the zoo function, each of these methods have been explained below.

All of the methods below will follow the same numbering system for the days of the week:

1	2	3	4	5	6	7
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday

1. `void zooSchedule(int dayOfWeek, boolean isPremium, boolean emergency)` (14 points):

This method takes in an `int dayOfWeek`, which represents what day it is, `boolean isPremium`, which is `true` if the person has a premium membership to the zoo, and `boolean emergency`, which is `true` if there is an emergency at the zoo. With the following inputs, the method should **print out** (note the return type of the method!) the schedule of animals on display for the specific day. The zoo has four main types of animals: `amphibians`, `mammals`, `birds`, and `reptiles`. There are two popular animals reserved for premium members: `polar bears` and `penguins`. Follow the guidelines below to display the correct schedule. Make sure to follow the format of the example.

- If an invalid day of the week is passed in, **print** `"Invalid day of the week"`
- On even days of the week, the zoo displays `amphibians` and `mammals` in that order.
- On odd days of the week, the zoo displays `birds` and `reptiles` in that order.
- If the person is a premium member of the zoo, they can see `polar bears` on even days and `penguins` on odd days.
- If the zoo is undergoing an emergency, print `"Zoo is closed due to emergency"`

Parameters: `int dayOfWeek, boolean isPremium, boolean emergency`
Return type: `void`
Example: `zooSchedule(2,true,false)` →

```
Today's animals:
amphibians
mammals
polar bears
```

2. `String feedAnimals(int dayOfWeek, int hourOfDay)` (14 points):

This method takes in two `int` variables: `dayOfWeek`, which represents what day it is, and `hourOfDay`, which represents what hour of the day it is **(24-hour military time)**. Based on the day and time, return a `String` informing the zookeeper if the animals are fed or not. Follow the guidelines below:

- If an invalid day of the week **or** an invalid hour of the day is passed in, **return** `"Invalid input provided"`. (Notice the different return types between this method and the previous one?)
  - Hour of the day must be greater than or equal to 0 and less than or equal to 23
- Animals are **not fed** on Sundays, so if it is Sunday **return** `"No feeding scheduled on Sundays"`
- Animals are fed once:
  - `amphibians` and `mammals` are fed between 8:00 - 10:00.
    - return `"Feeding the amphibians and mammals"`
  - `birds` and `reptiles` are fed between 12:00 - 14:00.
    - return `"Feeding the birds and reptiles"`
- At any other time of the day **return** `"No feeding scheduled at this time"`

Parameters: `int dayOfWeek, int hourOfDay`

Return type: `String`
Example: `feedAnimals(4, 8)` → `Feeding the amphibians and mammals`

3. `double calcTicketPrice(int dayOfWeek, int age, boolean isPremium)` (14 points):

This method calculates the price of a ticket based on several factors and returns the price as a `double`. It takes in two `int` variables: dayOfWeek and age. Once again, it also takes a `boolean` variable, isPremium. The return type is a double

- If an invalid day of the week is passed in, **return** `0.0`.
- If the person is strictly under the age of 5, they get free admittance, so return `0.0`
- If the person is between the age of 5 and 12 (inclusive range), the child price is `14.0`
- If the person is between the age of 13 and 64 (inclusive range), the adult price is `20.0`
- If the person is greater than or equal to the age of 65, the senior price is `12.0`
- Premium members get a discount of 40%
- If it's the weekend, simply add an additional 3 dollars surcharge (**post premium discount**). Note**:** if the person is under 5, they still get free admittance on weekends; the 3 dollars surcharge is not added.

Parameters: `int dayOfWeek, int age, boolean isPremium`
Return type: `double`
Example: `calcTicketPrice(6, 23, true)` → `15.0`

Part 2.1: Test Correctness of ZooManager Methods (8 points)

Similar to the previous part of the assignment, you will be testing your code. **In this part of the assignment, you need to implement your own test cases in the method called `unitTests`.**

In the starter code, several test cases are already implemented for you. You can regard it as an example to implement other cases. The general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method should return `true` only when all the test cases are passed. Otherwise, it should return `false`. **To get full credit for this section, for each**

method, you should have at least three total test cases that cover different situations (including the ones we have provided). In other words, you will need to create two more tests for `zooSchedule`, two more tests for `feedAnimals`, and two more tests for `calcTicketPrice`.

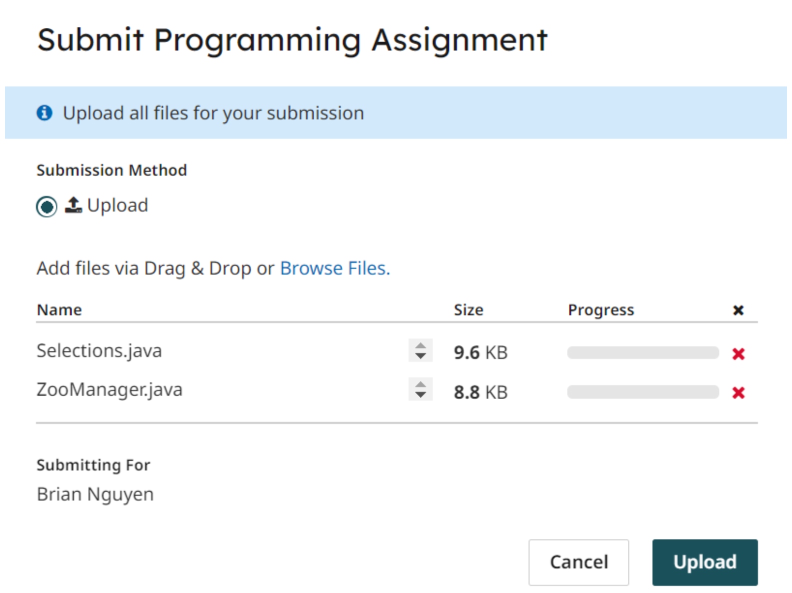
Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on Assignment 2.
2. Click the DRAG & DROP section and directly select the required files (`Selections.java` **and** `ZooManager.java`). Drag & drop is fine. **Please make sure you don't submit a zip. Make sure the filename is correct.**
3. **You can resubmit an unlimited number of times before the due date.** Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of basic tests. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

5. Your submission should look like the screenshot below. **If you have any questions, then feel free to post on Piazza!**


Submit Programming Assignment

Upload all files for your submission

Submission Method

● Upload

Add files via Drag & Drop or Browse Files.

Name	Size	Progress
Selections.java	9.6 KB	
ZooManager.java	8.8 KB	

Submitting For

Brian Nguyen