Due: Oct. 16 (10/16), 13:00

Overview

This assignment consists of four parts: implementing a stack, implementing a queue, evaluation of prefix notation expressions, and implementing a basic matchmaking system.

General Notes

- *Read this homework guideline carefully.* If you do not follow the guidelines, you may receive a 0 regardless of whether your code works or not.
- Do not use any IDEs (Eclipse, IntelliJ IDEA, etc.)
  - We recommend Sublime Text (Linux/Mac/Windows), Notepad++ (Windows), or TextWrangler (Mac).
  - IDEs often create a "package" of your code, which breaks the autograder.
  - **If you know how to fix the package problem**, you can use any IDE you want. However, we will not answer any questions related to this problem since we have already recommended a solution.
- Do not change any method or class signatures. If your code changes any class or method names or signatures, you will receive an automatic 0.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If your code does not compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- To ensure that your code will be accepted by the autograder, you should submit your code on LearnUS, download it again, recompile it and check the provided test suite. This way, you know the file you are submitting is correct.

Introduction

In this assignment, you will implement a generic stack and queue based on a linked list. We have provided a fully functoning linked list implementation in the form of a class file. The linked list is singularly linked, linear, has a head and tail pointer, and implements the following methods:

- `LinkedList()`
- `void insert(int index, T data)`
- `void remove(int index)`
- `void clear()`
- `T getData(int index)`
- `int getSize()`
- `String toString()`

You should not edit any code in the `LinkedList.java` file.

Stack

Stacks are a commonly used *LIFO* (Last In First Out) data structure. In this homework, you will implement a generic stack using the provided linked list implementation. The methods to be implemented are:

- `Stack()`
- `T peek()`
- `T pop()`
- `void push(T data)`
- `void clear()`
- `int getSize()`
- `boolean isEmpty()`

The descriptions of these methods can be found in the `Stack.java` file. Note that **all** of these methods should run in $O(1)$ time.

Queue

Queues are another common data structure, but are *FIFO* (First In First Out), unlike stacks. Again, using the provided linked list implementation, you will implement a generic queue. The methods to be implemented are:

- `Queue()`
- `T peek()`
- `T dequeue()`
- `void enqueue(T data)`
- `void clear()`
- `int getSize()`
- `boolean isEmpty()`

The descriptions of these methods can be found in the `Queue.java` file. Note that **all** of these methods should run in $O(1)$ time.

Postfix Calculator

As you learned in the lecture, *postfix notation* is a representation for mathematical expressions. For example, the expressions $5+10\times4$ would be written as $5\ 10\ 4\ \times\ +$. Stacks and queues can be used ot evaluate expressions written in postfix notation. Here you will use your stack and queue implementations to evaluate postfix expressions. You will implement only two methods:

- `PostfixCalculator()`
- `int evaluate(String exp)`

The descriptions of these methods can be found in the PostfixCalculator.java file. For this assignment, we will guarantee several things:

- We will only use the operators $+\ -\ \times$.
- We will only consider non-negative integers as terms.
- All inputs to solve are guaranteed to be valid postfix notation expressions of length at least 1.
- Terms and operators will be separated by a single space.
- The expression evaluation will never overflow Java's `int`.

League of Legen

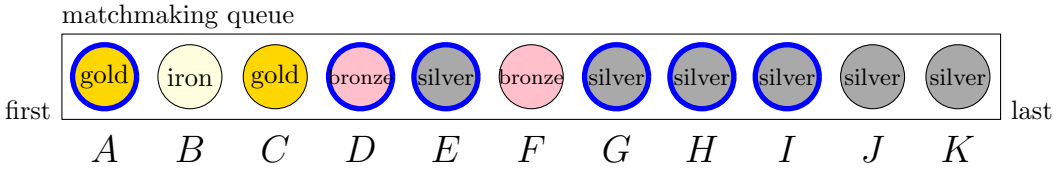
League of Legen is a famous 3-on-3 online game. Before playing the game, players enter a matchmaking queue to form a team of three and find an opponent team of three. Now, one cannot simply group any six people into one game because players' skill levels vary a lot. Therefore, the designers of League of Legen decides to make a tier system and categorizes players into one of 7 tiers according to their skill level.

$$\text{Iron} < \text{Bronze} < \text{Silver} < \text{Gold} < \text{Platinum} < \text{Emerald} < \text{Diamond}$$

Based on the players' tiers, you must implement a matchmaking system that is reasonable in waiting time and skill level.

Matchmaking policies

Since a single game requires six players, one can think of grouping six players of the same tier whenever they are available. However, some tiers like Emerald or higher have very few players and would take too long for them to play a game. Therefore, League of Legen tries to mix players from nearby tiers. Matchmaking takes place every time when six players of the same tier are waiting in the matchmaking queue. If this condition is satisfied, then the six players with the same tier, say T, are selected. Then, the matching system considers the neighboring tiers. For each neighboring tier, the matching system searches the player that has waited the most among the players in the neighboring tier. If that player has waited longer than all the players with tier T, the matching system selects that player from the other tier and removes the player that has waited the shortest time period among the six selected players. Therefore, the matching system may replace at most one player from its neighboring tier; in other words, at most two players among the initial six players will be replaced.



For example, let's say that 11 players are waiting for matchmaking in the order described by the figure above (Gold-Iron-Gold-Bronze-Silver-Bronze-Silver-Silver-Silver-Silver-Silver). Since six players in the Silver tier are waiting, the matchmaking starts. By default, we first select the six Silver tier players E, G, H, I, J, and K. Now, we look for the players that have waited the longest in the Gold tier and the Bronze tier, which are neighboring tiers of Silver. Player A, the player who has waited the longest from the Gold tier, has indeed waited longer than the player that has waited the longest in the Silver tier. Therefore, the matching system removes the last Silver player, K, and select player A. Likewise, the Bronze player D that has waited the most entered the queue before all Silver players. Therefore, the matching system removes J and selects D. Note that the system does not choose B, the Iron player, because Iron is two tiers away from Silver. Moreover, even though C is a Gold tier player who has waited longer than all Silver players in the matchmaking queue, the policy only cares for the player in the Gold tier that has waited the most, so C remains in the matchmaking queue. The players that are selected by the matchmaker are outlined in blue.

Matchmaking implementation

The following are the methods that you will implement in `MatchMaker.java`.

1. `public Player[] newPlayer(Player player)`: given a new player, keep track of the players that are waiting in the matchmaking queue using a queue that you implemented. If the set of waiting players meet the matchmaking criteria, return an array of 6 selected players listed in the order of their arrival. Each call should take $O(1)$ time.

General Directions

- Write your name and student ID number in the comment at the top of the files in src/main directory.
- Implement all of the required methods.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return types and edge cases.
- All the codes we provided can be found in src/base directory. If you are unsure what a class/method exactly does, please refer to the code.

Submission Procedure

Submit the files in the src/main directory, excluding Main.java, as a zip file. You *must* make a zip file for submission using the Gradle build tool (refer to Compiling section).

For this assignment, you should submit only the following five files:

- Stack.java
- Queue.java
- PostfixCalculator.java
- MatchMaker.java
- your_student_id_number.txt

You must rename 2023xxxxxx.txt to your actual student ID number. Inside that text file, you must include the following text. Please be sure to write all the following text including the last period.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2023123456.txt* is correct while something like *2023123456_pa2.txt* will receive 0.

Compiling

You can test your Java code using the following command:

```
% ./gradlew -q runTestRunner
```

You can also make a zip file for submission using the following command. The zip file named with your student id (the name of the .txt file) will lie in the "build" directory:

```
% ./gradlew -q zipSubmission
```

We provide an empty Main class for testing using standard input/output:

```
% ./gradlew -q --console=plain runMain
```

Since this file (src/main/Main.java) is not for submission, you can use any package in the file.

On Windows, use `gradlew.bat` instead of `./gradlew`.

Testing

We have provided a small test suite (src/test) for you to check your code. You can test your code by compiling and running the tester program.

Note that the test suite we will use to grade your code will be much more rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.