Regis University CC&IS
CS310 Data Structures
Programming Assignment 4: Stacks and Queues

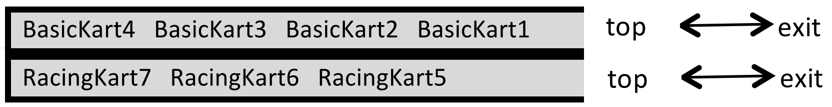
Problem Scenario

The brokerage office has an annual event to reward the brokers. This year the office decided to treat the brokers to go-karting. (You might be wondering about the reward for the software developers. Your reward is getting to write this program. Maybe this isn't such a great place to work after all.)

The go-kart track maintains two different types of go-karts. They have a basic go-kart, which does not go very fast, and another go-kart that is for racing. The brokerage office decided that any broker with more than $5,000,000.00 or more dollars in their portfolio would be eligible to use the racing go-karts. These brokers will be referred to as top brokers. Any broker with less than $5,000,000.00 (standard broker) will only be allowed to use the basic go-kart.

The track has two lines of go-karts, one for each type. Basic go-karts are parked in one single line and the racing go-karts are parked in a second single line. There is only one entrance/exit for each line. Go-karts are numbered, starting from 1 in the basic line, and continuing the numbering in the racing line. The lines can currently hold 4 basic go-karts and 3 racing go-karts.

So the basic go-karts will be numbered 1 to 4, and the racing go-karts will be numbered 5 to 7. (Notice that the karts are inserted initially such that they are descending order, meaning the lowest numbered kart in each stack will be the first to be removed.)


BasicKart4 BasicKart3 BasicKart2 BasicKart1
top
exit
RacingKart7 RacingKart6 RacingKart5
top
exit

This program will use stacks and queues to control the use of go-karts and brokers in the lines.

If all go-karts from the racing line are gone, a top broker may be assigned a go-kart from the basic go-kart line. If all go-karts from both lines are out, or the broker is not a top broker and the basic go-kart line is empty, the broker must wait until a go-kart is returned. The requesting broker is placed into a queue. There will be a queue for top brokers and a queue for the standard brokers. Brokers in the top broker queue will be assigned any returned go-karts ***before*** the brokers in the other queue.

Obtaining and Unzipping the NetBeans Assignment Template Project Files

A NetBeans template project has been provided for you to complete Assignment 4. The first step is to locate this file in the Resources for Students online content area and download it to your computer. Make sure to save the zip file to a convenient location so that you can locate it later.

After you have downloaded the project zip file, you need to unzip the file so that you can work with the files correctly. On Windows computers, double-clicking on the file will probably NOT unzip the file. To unzip the file, right-click on it and select an option like "Extract All…", which will bring up a dialog box to help you unzip the file correctly. If you need help doing this, an Internet search will provide many resources.

You may also want to move your unzipped project folder to another location, such as where you save all of your other course work.

Note: If you encounter this warning during compile it is safe to ignore it and it will not impact your ability to complete the assignment: "warning: [options] bootstrap class path not set in conjunction with -source 8". This will probably only be displayed if you are using an alternate version of NetBeans than what is described in the course content.

Renaming Your Project Name and Project Folder

The first thing you want to do is rename your project name from *CS310Assignment4Template* to *CS310Assignment4<firstName><lastName>*, where <firstName> and <lastName> are replaced with your first and last name. So, someone with a first name of "John" and a last name of "Smith" would rename the project as "CS310Assignment4JohnSmith". To do this, follow these general steps:

1. In the Project window, right-click on the current project name of *CS310Assignment4Template*
2. From the menu, select "Rename…"
3. In the Project Name box, enter the correct new name
4. Check the box for the "Also Rename Project Folder" option
5. Finally, click the "Rename" button

Using the Assignment Template Project Files

It is important to know exactly what the provided template files give you and what you need to actually complete for this assignment.

Provided For You:

- A project that will compile and run. All necessary code files and directories are created.
- Fully functional and documented Broker and StockTrade classes. (They have something that will need to be completed for Assignment 5, which can be ignored for now.)
- Fully functional and documented BrokerLog and StockTradeLog classes. The offer the minimum functionality needed to complete this assignment, so features have been removed.
- A partially functional and documented CS310Assignment4 class.
- A fully functional BrokerNode class. There is nothing to change in this class.
- Templates for the GoKartStack, GoKartUsage, BrokerQueue, and Report classes. These are minimally functional.
- A **go-kart-testing-data.txt** file in the input folder of the project. This contains the same go-kart request/return data in the PDF that shows the expected output results. You can copy and paste from here into the gokartinfo.txt file for easier testing.

To Complete:

- CS310Assignment4 class – methods pertaining to the go-kart scenario (processGoKartRequest and processGoKartReturn) and documentation for these methods
- BrokerQueue class – method code and documentation
- GoKartStack class – method code and documentation
- GoKartUsage class – method code and documentation (just add your name)
- Report class – method code and documentation

Each of the areas in the Brokerlog, StockTradeLog, and Report class that need to be completed have a comment inside the method body like this:

```
// TODO - complete this method
```

Once you complete a method, remove these comments (i.e., the TODO comments).

DO NOT CHANGE the method signatures, data member names (attributes), constant names, or access specifiers (public / private / static).

You are also responsible for completing any of the missing required Javadoc comments that are part of the course coding standards. You can use the Javadoc commenting in other classes of the project as an example for writing your own comments.

Program Requirements

You will be using the CS310Assignment4.java file to read an **`assn#input.txt`** input data file (as used for previous assignments), since you will still need to access the brokers, and will need to be able to determine which of them are top brokers. In this assignment, Brokers and StockTrades will not be removed and the lists will no longer be cleaned.

A second input data file will be used to simulate brokers requesting go-karts and returning go-karts. The filename is **`gokartInfo.txt`** and will be located in the **input** folder. Each line of data will be formatted like this (note that this file is NOT comma delimited, like the broker data file):

```
REQUEST brokerLicenseNumber
```

or

```
RETURN brokerLicenseNumber
```

The go-karts for each line in the lot will be stored in a **stack**. When a broker requests a go-kart, the next available go-kart is provided (by popping a go-kart off the stack).

> You will create your own **GoKartStack** using a standard array. The data stored in each stack will be an integer, to represent the go-kart number. Go-karts are numbered, starting from 1 in the basic line, and continuing the numbering in the racing line (so the basic go-kart stack will hold go-karts numbered 1 to 4. The racing go-kart stack will hold go-karts numbered 5 to 7). Use constants to define the go-kart number ranges, so the program will work if the company decides to expand the available number of go-karts later. The constants are already created for you. You might want to make sure your program works with differing amounts of go-karts and numbering. There will not be a gap in the numbers though, so the go-karts will always be numbered 1 to N, with some amount of basic and some racing. For instance, there could be two basic go-karts numbered 1 and 2 and two racing go-karts numbered 3 and 4. Be sure to put the constants back to the default starting values if you change them when testing.
>
> Each stack should be initialized so that all the go-karts are in the stack, with the highest go-kart number at the bottom of the stack and lowest go-kart number at the top of the stack. Note that this means the stack will initially be ***full***. When the stack is ***empty***, there are no more available go-karts available of that type.
>
> This implementation will be used to create two separate stacks of go-karts in the CS310 main class. The same implementation will be used to instantiate each stack (representing lines of go-karts) – one stack for basic go-karts and one stack for racing go-karts. (This is taken care of at the top of the CS310 main class.)
>
> The **GoKartStack** class should implement the ability to **push** and **pop** go-karts from the stack. You also need to include methods to determine if the stack is **empty** or **full** – and use these methods. (Method signatures are provided in the templates.)

As brokers request and are assigned a go-kart from the lot, you will need to be able to specify which broker has which go-kart. You will keep track of this information using the **GoKartUsage** class. As brokers request and return go-karts, you will need to update the go-kart usage information. Documentation was left in the template to help you understand what needs to happen here.

If a broker requests a go-kart, but there are not any go-karts available, the broker will be assigned to a queue to wait for the next available go-kart. If the broker is a top broker, that broker is assigned to the top broker queue. If the broker is not, then the broker is assigned to standard broker queue. A new waiting broker will be added to the rear of the queue. When a broker is assigned a returning go-kart, and is removed from the queue, that broker will be taken from the front of the queue.

> You will create your own **BrokerQueue** using a singly linked list, with data field references to both the **front** and **back** nodes in the queues (you will use the **BrokerNode** class from to build this linked list).
>
> This implementation will be used to create **two separate queues** of **BrokerNodes** in the CS310 main class. The same implementation will be used to instantiate each queue (representing lines waiting for go-karts) – one queue for waiting standard brokers, and one for waiting top selling brokers. (This is taken care of at the top of the CS310 main class.)
>
> The **BrokerQueue** code should be able to **add (enqueue)** and **remove (dequeue)** brokers from a queue (i.e. the add method should have a **Broker** as an input parameter, and the remove method should return a **Broker**).
>
> You will also need to build methods to check if a queue is **empty** and use this method when needed. Be sure to check if a queue is empty, before trying to remove a broker from it.

The brokerage office also wants the program to check for other office workers trying to use the go-karts. If someone requests a go-kart, but his/her broker license number is not in your **Broker** linked list (built using the code from Assn 3), then issue an error message, ignore the request, and move on to the next request/return. For example (both indented the same as the other audit trail items):

```
ERROR: Unknown broker ABC00788! Request ignored!

ERROR: Unknown broker CBA00788! Return ignored!
```

Note: Do not add methods to the stack or queue that give access to the underlying array or singly linked list structures. The CS310 main class and Report class must use the stack and queue methods you are required to create in order to accomplish the assignment goals. When implementing the stack and queue, you can of course access the array or linked list to make them perform as expected.

Note: UML diagrams are at the end of this document for those that like them.

As each request/return action occurs, you will provide an audit trail. Sample audit trails are provided in the Assignment 4 testing data provided in the Resources for Students section of the online course content.

After processing the entire **`gokartInfo.txt`** input file, your application will provide a report with the existing go-kart assignments (which broker is currently using which go-kart), which go-karts are available in each of the stacks (the top of the stack is to be listed first, followed by the other go-karts, in order as they are popped from the stack), and which brokers are still sitting in each of the queues (in the order as they are removed from the queue). To generate this report, you will complete the generateGoKartReport method in the Report class.

The report should be placed into a **`gokartUsageReport.txt`** file to be located in the **output** folder.

The program must follow all **CS310 Coding Standards** from Content section 1.9.

Hints

You will need to create and call three new methods within the **CS310Assignment4** class:

processGoKartInfo – read and process **gokartInfo.txt** data file. (Created for you in template.)

processGoKartRequest – handle Brokers requesting a go-kart (produces request audit trail)

processGoKartReturn – handle Brokers returning a go-kart (produces return audit trail)

The new classes for this program are:

GoKartStack – implementation of a stack of go-karts with go-kart numbers. Note that two different stacks will be instantiated (basic and racing) from the same **GoKartStack** class.

GoKartUsage – keeps track of which Brokers are currently using which go-karts

BrokerQueue – implementation of queue of Brokers waiting for go-karts. Note that two different queues will instantiated (standard and top broker) from the same **BrokerQueue** class.

Additional Requirements

- Create **Javadoc headers** for the GoKartStack, GoKartUsage, BrokerQueue, and Report classes. You are responsible for completing the comment headers created.
- Your input data files will still be read from the **input** folder in your project. A filled in assn4input.txt and gokartinfo.txt are already provided. You can create your own test data or copy/paste the provided test data here when you are working with your project.
- The report your program creates will once again be written to the **output** folder.
- Make sure that your program has no compilation errors.
  - Remember, if you have compile errors, a red error symbol is placed on the file and folder name tabs. Make sure to clear all compile errors before submitting your project.

  WARNING: Programs that **do not compile** will not be accepted!

Program Submission

This programming assignment is due by 11:59 PM on the date listed in the **Course Assignments by Week / Due Dates Listing** posted in the Faculty Course Content area.

- Export your project from NetBeans (make sure that you renamed everything correctly first):
  - Highlight the project name.
  - Click on **File** from the top menu, and select **Export Project**.
  - Select **To ZIP**
  - Name your export file in the following format:
    `CS310<lastname>Assn<x>.zip`

    For example:
    `CS310SmithAssn4.zip`

NOTE: Save this zip file to some other directory, not your project directory.

- Submit your **.zip** file to the **Assignment 4** Submission Folder (located under **Assignments** tab in online course).
  Only NetBeans export files will be accepted.
  Do not use any other kind of archive or zip utility.

Grading

This program will be graded using the **rubric** that is linked under **Student Resources** page.

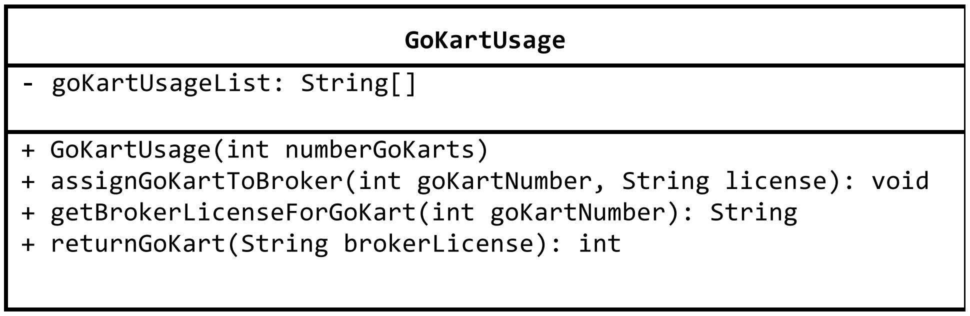
WARNING:
*Programs submitted late will **not** be accepted,*
and will receive a grade of 0.

See Resources for Students in the online course content for testing data and expected outputs, as well as the NetBeans template for completing this assignment.

New Class UML Diagrams

BrokerQueue
- queueFront: BrokerNode - queueBack: BrokerNode - queueSize: int
+ BrokerQueue() + isQueueEmpty(): boolean + enqueue(Broker brokerObj): void + dequeue(): Broker + peek(): Broker

GoKartStack
- goKartStack: int[] - topIndex: int
+ GoKartStack(int numberGoKarts, int startingGoKartNumber) + isStackEmpty(): boolean + isStackFull(): boolean + push(int goKartNumber): void + pop(): int + peek(): int



```
GoKartUsage
- goKartUsageList: String[]
+ GoKartUsage(int numberGoKarts)
+ assignGoKartToBroker(int goKartNumber, String license): void
+ getBrokerLicenseForGoKart(int goKartNumber): String
+ returnGoKart(String brokerLicense): int
```