EECS 1012 Programming Assignment/Project 2
Due: Aug 3 (Wed) 23:00

This assignment/project can be done by yourself, or, with a team of two. Only one member of each team submits the answers.
Note that, however, **you can neither share your solution with other teams nor get help from them**. Please be aware that your code will be checked by both a plagiarism detection tool and TAs to make sure your solution to this problem is unique.

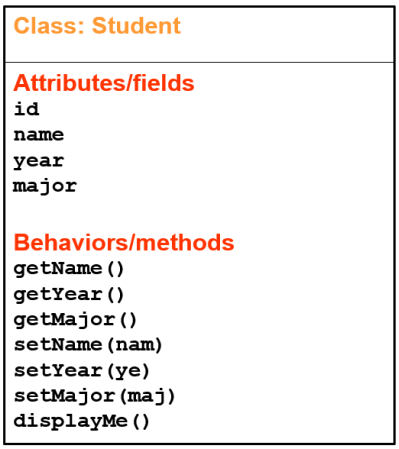
This programming assignment is intended to give you some insights into the OOP features of JavaScript, giving you preparedness for future courses such as EECS1012, EECS2030, EECS2011, EECS3101 etc. This assignment gives you exposure to the following skills:

- basic OOP concepts, including classes, objects, attributes, constructors, getter methods, setter methods and other methods.
- basic data structures such as Stacks, Queues, and their implementation using OOP.

You have 5 problems to solve.

Problem 1

In this problem, implement a simple Student class.
Recall a class contains data (attributes) and behavior (methods)



The Student class has the following attributes/fields:
`name`
`id`
`year`
`major`

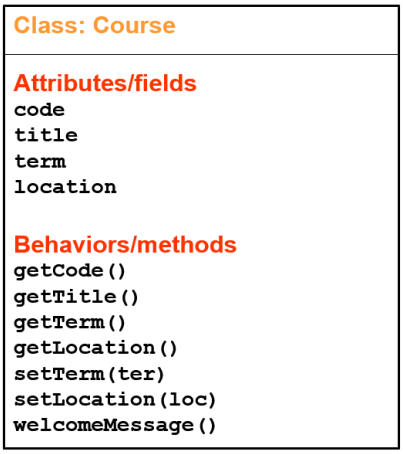
and has the following methods:
`getName()` return the name
`getYear()` return the year
`getMajor()` return the major
`setName(nam)` set/change the name to `nam`
`setYear(ye)` set/change the year to `ye`
`setMajor(maj)` set/change the major to `maj`
`displayMe()` return a string that contains the id, name, year and major separated by a space

Implement these functions in file **A2.js**. Also complete the `useStudent()` function, which creates an instance (object) of the Student class, and calls methods of the object. You can add variables if you need.
When you finish, open **A2.html** with browser, and also open the console on the browser. Then click button "Run the program (Student)".
If implemented correctly, the following output will be displayed in the browser's console:

```
year: 2
major: Math
00173 Jon Lee 2 Math
-------------
year: 4
major: EECS
00173 Jon Lee 4 EECS
```

Problem 2

In this problem, implement a simple Course class.



The Course class has the following attributes/fields:

`code` code of the course, e.g., "EECS1012"
`title` title of the course, e.g, "Computational Thinking"
`term` semester code of the course, e.g., "21W", "22S"
`location` location of the course, e.g. "LAS-C", "CLH-B"

and has the following methods:

`getCode()` return the course code
`getTitle()` return the course title
`getTerm()` return the semester code
`getLocaton ()` get the location for the course
`setTerm (ter)` set/change the semester code for the course to `ter`
`setLocation(loc)` set/change the location for the course to `loc`
`welcomeMessage()` return a string that contains a welcome message "Welcome to *code title (xx), held in yy* " where *code* is the course code, *title* is the course title, *xx* is the term code, and *yy* is the location

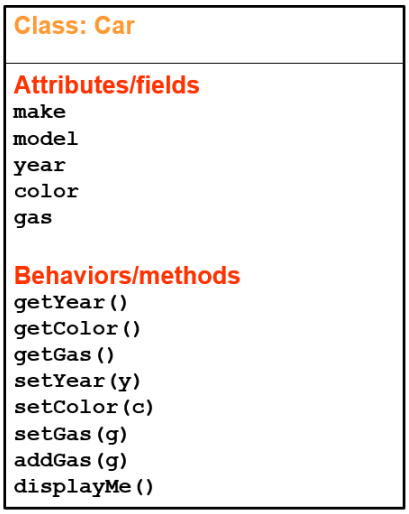
Implement these functions in file **A2.js**. Also complete the `useCourse()` function, which creates an instance (object) of the Course class, and calls methods of the object. You can add variables if you need.
When you finish, open **A2.html** with browser, and also open the console on the browser. Then click button "Run the program (Course)".
If implemented correctly, the following output will be displayed in the browser's console:

```
term: 21F
location: VH-B
Welcome to EECS1012 Computational Thinking (21F), held in VH-B
-------------
term: 22S
location: LAS-C
Welcome to EECS1012 Computational Thinking (22S), held in LAS-C
```

Problem 3

In this problem, implement a simple Car class.
The Car class has the following attributes/fields:

`make`
`model`
`year`
`color`
`gas` which has default value of 20 (Liter)



and has the following methods

`getYear()` return the year
`getColor()` return the color
`getGas()` returns the current gas level
`setYear(ye)` set/change the year to `ye`
`setColor(col)` set/change the color to `col`
`setGas(amount)` set/change the current gas level to `amount`
`addGas(amount)` add `amount` of gas to the current gas level

`displayMe()` return a string that contains make, model, year, color and gas level, separated by a space.

Implement these functions in file **A2.js**. Also complete the `useCar()` function, which creates instances (objects) of the Car class, and call methods of the objects. You can add variables if you need.
When you finish, open **A2.html** with browser, and also open the console on the browser. Then click button "Run the program (Car)". If implemented correctly, the following output will be displayed in the console

```
year: 2019
color: black
gas: 20
Honda Civic 2019 black 20
-------------
year: 2020
color: silver
gas: 50
Honda Civic 2020 silver 50
-------------
year: 2021
color: blue
gas: 20
Hyundai Elantra 2021 blue 20
-------------
gas: 90
Hyundai Elantra 2021 blue 90
```

Problem 4 Stacks and Queues

In this exercise, we use array to implement Stack and Queue data structure.

Problem 4A Stacks

Study the provided code for class Stack in **A2StackQ.js**, and complete this class definition.
Note that in this implementation, when popping an element out of stack, the element is not removed from the internal array. Instead, the algorithm just uses attribute *top* to indicate the current range of the stack.

Also complete the function useStack(). Try to understand the existing code. You can add variables if you need.
When you finish, open **A2StackQ.html** with browser, and also open the console on the browser. Then click button "Run the Stack".
If implemented correctly, you should get the following output on console:

```
size: 0
isEmpty: true
-------------
4
6
2
7
size: 4
isEmpty: false
peek: 4
-------------
pop: 4
```

```
size: 3
peek: 6
6
2
7
-------------
pop: 6
size: 2
peek: 2
2
7
---  push 10 ---------
size: 3
peek: 10
10
2
7
---  push 100 ---------
size: 4
peek: 100
100
10
2
7
-------------
pop: 100
size: 3
peek: 10
10
2
7
```

Problem 4B Queues

Study the provided code for class Queue in the same file, and complete this class definition.
Note that in this implementation, when dequeue an element, the element is not removed from the internal array. Instead the algorithm just uses attribute *frontIndex* and *rearIndex* to signify the current range of the queue.
Also complete the function useQueue(). Try to understand the code. You can add variables if you need.
When you finish, open **StackQueue.html** with browser, and also open the console on the browser. Then click button "Run the Queue".
If implemented correctly, you should get the following output on console:

```
size: 0
isEmpty: true
-------------
size: 4
isEmpty: false
front: 7
rear: 4
7 2 6 4
```

```
-------------
dequeue: 7
size: 3
front: 2
rear: 4
2 6 4
-------------
dequeue: 2
size: 2
front: 6
rear: 4
6 4
--- enqueue 10 ----------
size: 3
front: 6
rear: 10
6 4 10
--- enqueue 100----------
size: 4
front: 6
rear: 100
6 4 10 100
-------------
dequeue: 6
size: 3
front: 4
rear: 100
4 10 100
```

Problem 4C Sorting using Queues

Now that you have some ideas of how Stacks and Queues work, here you develop an algorithm to sort a queue of numbers in ascending order, using an auxiliary stack.

The idea is to maintain the sorted elements of the queue in the auxiliary stack.

The steps to sort a queue using a stack are as follows:

0. Create an auxiliary stack.
1. Dequeue an element E from the queue
2. If the stack is empty, push the element E in the stack.
3. Else, if the stack is not empty, but E is smaller than the top element of the stack, push E onto the stack.
4. Else (E is larger than the top element of the stack), pop the elements out of the stack until a smaller or equal element is found at the top of the stack or the stack becomes empty.
    - Each popped element in step 4 is put back into the queue (via enqueue).
    - When a smaller or equal element is found at the top of the stack or the stack becomes empty, push E onto the stack
5. Continue by repeating step 1-4 until the input queue becomes empty.
6. Now the stack contains all the elements of the queue, with the smallest element on top. All we need to do next is to transfer all the elements from the stack to the queue. The resulting queue will have the elements in ascending order.

front

As an example, suppose the input queue contains 3 1 2 where left-most is the queue front (same for all queue content listed below).

- Retrieve 3 from the queue. Since the stack is empty, push 3 into the stack. Now queue contains 1 2, stack contains 3
- Next, retrieve 1 from the queue. Since 1 is smaller than the top stack element 3, push 1 into the stack. Now queue contains 2 and stack contains 1 3 ⇐ top
- Next, retrieve 2 from the queue. Since 2 is larger than the top stack element 1, we pop 1 from the stack and enqueue it into the queue. Now queue contains 1, stack contains 3. Since 2 is smaller than the top stack element 3 now, push 2 into the stack. Now queue contains 1, stack contains 2 3
- Next, retrieve 1 from the queue. Since 1 is smaller than the top stack element 2, push 1 into the stack. Now queue is empty, stack contains 1 2 3
- Since the queue is empty now, we transfer the stack elements into the queue, so the queue will contain 1 2 3

As another example, suppose the input queue contains 7 3 5 6

- Retrieve 7 from queue. Since the stack is empty, push 7 into the stack. Now queue contains 3 5 6, stack is 7
- Next, retrieve 3 from the queue. Since 3 is smaller than the top stack element 7, push 3 into the stack. Now queue contains 5 6, stack contains 3 7
- Next, retrieve 5 from the queue. Since 5 is larger than the top stack element 3, we pop 3 from the stack and enqueue it into the queue. Now queue contains 6 3, stack contains 7 Since 5 is smaller than the top stack element 7 now, push 5 into the stack. Now queue contains 6 3, stack contains 5 7
- Next, retrieve 6 from the queue. Since 6 is larger than the top stack element 5, we pop 5 from the stack and enqueue into the queue. Now queue contains 3 5, stack contains 7 Since 6 is smaller than top stack element 7 now, push 6 into the stack. Now queue contains 3 5, stack contains 6 7
- Next, retrieve 3 from the queue. Since 3 is smaller than the top stack element 6, push 3 into the stack. Now queue contains 5, stack contains 3 6 7
- Next, retrieve 5 from the queue. Since 5 is larger than the top stack element 3, we pop 3 from the stack and enqueue it into the queue. Now queue contains 3 and stack contains 6 7 Since 5 is smaller than top stack element 6 now, push 5 into the stack. Now queue contains 3 stack contains 5 6 7
- Next, retrieve 3 from the queue. Since 3 is smaller than the top stack element 5, we push 3 into the stack. Now the queue is empty, stack contains 3 5 6 7
- Since the queue is empty now, we transfer the stack elements into the queue, so queue will contain 3 5 6 7

Study the code of function sortQueue(). The function creates a queue of 6 random elements and then calls function sortQUsingStack() to sort the queue.
Complete the function sortQUsingStack(q), which takes as argument a queue q, and sorts the queue using a stack.

When you finish, open **StackQueue.html** with browser, and also open the console on the browser. Then click button "Run the sorting algorithm" several times.
If implemented correctly, then each time you click the button, a queue of six random elements will be generated and then sorted, similar to the outputs below.

```
input:  37 64 48 90 65 91
sorted: 37 48 64 65 90 91
---------------
input:  24 5 51 32 90 45
sorted: 5 24 32 45 51 90
---------------
input:  10 26 29 53 55 70
sorted: 10 26 29 53 55 70
---------------
input:  62 28 94 33 12 81
sorted: 12 28 33 62 81 94
---------------
input:  70 71 55 76 2 77
sorted: 2 55 70 71 76 77
```

SUBMISSION

Submit the following files:

`team.txt` (only need to submit this file if you work in a team of up to two. List names and student numbers of both the team members)

```
A2.html
A2.js

A2StackQ.html
A2StackQ.js
```

Compress the files (.zip or **.tar** or **.gz), and then submit the (single) compressed file on eClass.**

- **Note that if you work on a team, only one team member submits (including** team.txt**). The other member does not need to submit anything.**