Homework 2

Due date: January 30, 2023, 12pm (noon)

Learning Objectives

- Trace operations on stacks, queues, and deques.
- Implement algorithms using stacks and queues.
- Trace a search algorithm.

Exercises

1. (10 points)
   Trace the following pseudocode. What is the content of the stack, queue, and deque after the pseudocode has been executed?

```
Queue que = new empty Queue;
Stack stk = new empty Stack;
Deque deq = new empty Deque;

deq.addHead(A);
deq.addHead(B);
deq.addHead(C);
deq.addHead(D);

while (not deq.isEmpty()) {
   stk.push(deq.getTail());
   que.enqueue(deq.getTail());
   deq.removeTail();
}
while (not stk.isEmpty()) {
   deq.addHead(stk.peek());
   deq.addTail(que.peek());
   stk.pop();
   que.enqueue(que.peek());
   que.dequeue();
}
```

   Content of queue que from head to tail: ______________abcd________________________

   Content of stack stk from bottom to top: ______________empty_______________________

   Content of deque deq from head to tail:
   _______________abcdabcd______________________

2. (10 points)
   The Java class Queue shown on the next page is supposed to implement a queue with a maximum of 1000 elements using a circular array. However, the methods enqueue and dequeue contain logic errors. Correct the methods enqueue and dequeue.

```
public class Queue<T> {
   private T[] data;  // array with elements in the queue
   private int first; // index of the element at the head
   private int last;  // index at which the next element is inserted
   private int size;  // number elements in the queue
   private static final MAX_SIZE = 1000; //maximum queue size

   /**
    * Initialize newly created queue
    */
   public Queue<T> () {
      data = new T[MAX_SIZE];
      first = 0;
      last = 0;
      size = 0;
   }

   /**
    * Determines whether the queue is empty
    *
    * @return true if the queue is empty
    */
   public boolean isEmpty () {
      return (size == 0);
   }

   /**
    * Access the value at the head of the queue
    *
    * @return element at the head of the queue
    * @exception EmptyQueueException empty queue
    */
   public T peek () {
      if (size == 0) {
         throw new EmptyQueueException();
      }
      return data[first];
   }

   /**
    * Adds a new value at the tail of the queue
    *
    * @param item element to be inserted at the tail of the queue
    * @exception QueueOverflowException full queue
    */
   public void enqueue (T item) {
      if (size >= MAX_SIZE) {
         throw new QueueOverflowException();
      }
      if (last >= MAX_SIZE) {
         last = first;
      }
      data[last] = item;
      last++;
      size++;
   }
```

```
    /**
     * Removes the value at the head of the queue
     *
     * @return element at the head of the queue
     * @exception EmptyQueueException empty queue
     */
    public T dequeue () {
       if (size == 0) {
          throw new EmptyQueueException();
       }
       if (first < 0) {
          first = MAX_SIZE - 1;
       }
       T item = data[first];
       if (first == last) { //if there is only one item in queue
          data[first] = null;
          return item;
       }

       //deque removes the first element -> shift all elements “left” by one
       for (int i = 1; i < last; i++) {
          data[i] = data[i – 1];
       }
       first--;
       size--;
       return item;
    }
 }
```

3. (10 points)
   A food pantry, which collects and distributes only cans of vegetables and fruits, operates on a strictly first-in first-out policy. People can pick-up either the oldest can (based on the time the can was given to the food pantry) of all cans in the shelter, or they can select whether they would prefer a can of vegetables or fruits (and will receive the oldest can of that type). They cannot select which specific can they would like. Write Java-like pseudocode for the class FoodPantry below that contains the data structures to maintain this system and that implements operations enqueue, dequeueAny, dequeueVeggies, and dequeueFruits. Your class can use stacks, queues, and deques only. You can assume that there are the classes Stack, Queue, Deque, Can, FruitCan, and VeggieCan where FruitCan and VeggieCan are subclasses of class Can.

```
class FoodPantry {
   public int first;
   public int last;
   public Queue<Can> inventoryFruit;
   public Queue<Can> inventoryVeggie;

   public FoodPantry(){
         inventoryFruit = new Queue<Can>();
         inventoryVeggie = new Queue<Can>();
         firstFruit = inventoryFruit[0];
```

```
        firstVeggie = inventoryVeggie[0];
        lastFruit = inventoryFruit.length() – 1;
        lastVeggie = inventoryVeggie.length() – 1;
        }
        void enqueue(Can can) {
              if (can = Can.FruitCan) {
                    inventoryFruit.enqueue(can);
              } else {
                    inventoryVeggie.enqueue(can);
              }
        }

        Can dequeuAny()  {
              if (inventoryFruit.size == 0) {
                    return inventoryVeggie.dequeue();
              }
              if (inventoryVeggie.size == 0) {
                    return inventoryFruit.dequeue();
              }
              inventoryVeggie.dequeue();
              return inventoryVeggie[first];
        }

        Can dequeVeggies() {
              return inventoryVeggies.dequeue();
        }

        Can dequeueFruits() {
              Return inventoryFruits.dequeue();
        }
  }
```

4. (10 points)

   Apply BFS and DFS to determine the route in a network with the nodes A, B, C, D, E, F, G, and H. The nodes and neighbors of a node are given below. The source node is B, the destination node is E. The neighbors of a node are always retrieved in the order listed below.

Neighbors of A: B, C	Neighbors of E: A, F, G
Neighbors of B: A, C, D	Neighbors of F: B, C
Neighbors of C: B, D, F	Neighbors of G: C, E
Neighbors of D: B, F, H	Neighbors of H: D, G

   a) Use the generic search algorithm to list the order of the nodes in which they are removed from the collection nextNodes in case of BFS.
      Removed from nextNodes: B, A, C, D, F, H, G, E
   b) What is the solution path determined by BFS? B > C > D > H > G > E

Submission

Upload your solution as a PDF file to the course website.