QUEENS COLLEGE Department of Computer Science
CSCI 313 First Midterm Exam Fall 2019 10.03.19

Instructor: Alex Ryba

07.45am – 09.00am, Thursday, October 03, 2019

Problem 1 *(10 points)*

(a) Write the Java interface for the ADT Stack. (Give only the 2 most important methods.)

```
public interfact Stack<T> {
   void push(T t);
   T pop();
}
```

(b) Write the Java interface for the ADT Queue. (Give only the 2 most important methods.)

```
public interfact Queue<T> {
   void enqueue(T t);
   T dequeue();
}
```

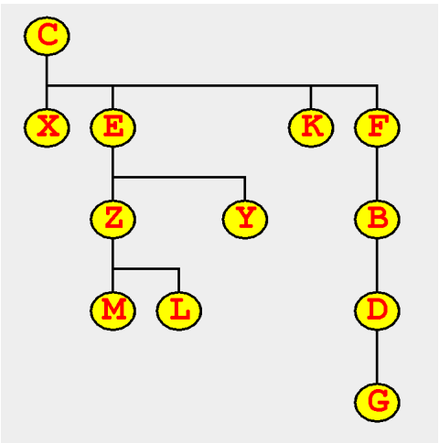
(c) Write the Java interface for the ADT Iterator. (Give only the 2 most important methods.)

```
public interfact Iterator<T> {
   boolean hasNext();
   T next();
}
```

(d) Write the Java interface for the ADT Iterable. (Give only the most important method.)

```
public interfact Iterable<T> {
   Iterator<T> iterator();
}
```

Consider the following tree:



(e) Write the level order traversal.

`CXEKFZYBMLDG`

(f) Write the preOrder traversal.

`CXEZMLYKFBDG`

(g) Write the postOrder traversal.

`XMLZYEKGDBFC`

(h) What is the height of the tree?

`4`

(i) What is the depth of the Node Z?

`2`

(j) What is the height of the Node Z?

`1`

Problem 2 *(10 points)* Consider the following `Java` program.

```
public class TwoStack<T> {
   private Stack<T> input;
   private Stack<T> output;

   public TwoStack() {
      input = new Stack<>();
      output = new Stack();
   }

   public boolean empty() {
      return input.empty() && output.empty();
   }

   public void add(T t) {
      input.push(t);
   }

   public T remove() throws Exception {
      if (empty()) throw new Exception("Empty");
      if (output.empty())
         while (!input.empty()) output.push(input.pop());
      return output.pop();
   }

   public static void main(String args[]) throws Exception {
      TwoStack<Integer> t = new TwoStack<>();
      int data[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
      t.add(data[0]);
      System.out.println(t.remove());                          // line a
      for (int i = 0; i < 10; i++) t.add(data[i]);
      for (int i = 0; i < 3; i++) System.out.print(t.remove());  // line b
      System.out.println();
      t.add(11);
      t.add(12);
      t.add(13);
      for (int i = 0; i < 8; i++) t.remove();
      while (!t.empty()) System.out.println(t.remove());         // line c
   }
}
```

(a) What is the ouput from the instruction on line (a)?

Answer: `0`

(b) What is the ouput from the instruction on line (b)?

Answer: `012`

(c) What is the ouput from the instruction on line (c)?

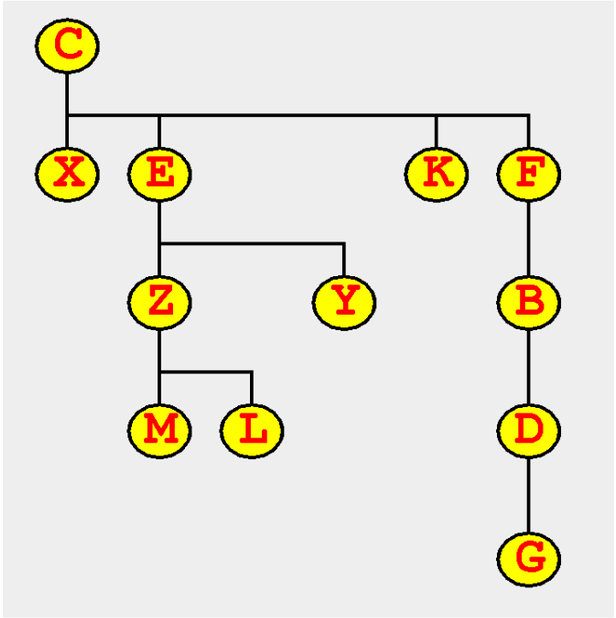
Answer:

```
12
13
```

(d) What are the standard names for the `class TwoStack` and its methods `add` and `remove`? (Hint: What iterface can it implement?)

Answer: A TwoStack object is a Queue. Its method add is enqueue. Its method remove is dequeue.

Problem 3 *(10 points)* This question asks you to fill in the missing parts from the code for a method that does reverse level order traversal of a tree. In reverse level order traversal the levels are visited from lowest to highest. Within each level the nodes are placed in a standard left to right order. For example, if the tree is as follows:



Its reverse level order traversal is `G M L D Z Y B X E K F C`.

Five small pieces of the code have been omitted and replaced by PART (a), PART (b) and so on in the following implementation. You should answer each part by supplying the omitted code. (In each case it is at most one line of code.)

The code makes use of the standard Java class `ArrayList`, a `class Tree` whose method `root()` returns its root node, and a `class TreeNode` whose method `getChildren()` returns an array list showing its children in order from left to right. You should not apply other classes or methods.

```
public static ArrayList<TreeNode> reverseLevelOrder(Tree t) {
   ArrayList<TreeNode> waiting = new ArrayList<>();
   ArrayList<TreeNode> answer = new ArrayList<>();
   if (PART (a)) {
      waiting.add(PART (b));
      int done = 0;
      while (done < waiting.size()) {
         TreeNode oldNode = waiting.get(PART (c));
         ArrayList<? extends TreeNode> nextBlock = oldNode.getChildren();
         for (PART (d))
            waiting.add(nextBlock.get(i));
         answer.add(PART (e));
      }
   }
   return answer;
}
```

(a) Give a replacement for PART (a). **Answer:** `t.root() != null`

(b) Give a replacement for PART (b). **Answer:** `t.root()`

(c) Give a replacement for PART (c). **Answer:** `done++`

(d) Give a replacement for PART (d). **Answer:** `i = nextBlock.size() - 1; i >= 0; i--`

(e) Give a replacement for PART (e). **Answer:** `0, oldNode`