LAB #10 - STACKS AND QUEUES

PART 1: Exercises

1. What is the output after the following Java program is executed? Assume the class `Lab10_Ex1` defines the missing methods to print.

```
//Client for user-defined classes
//LinkedStackClass and QueueClass
//Methods (missing here): printStack, printQueue
import java.util.*;
public class Lab10_Ex1 {
    static Scanner input = new Scanner(System.in);
    public static void main(String[] args) {
        LinkedStackClass<Character> s = new LinkedStackClass<Character>();
        QueueClass<Character> q = new QueueClass<Character>(10);
        Character ch;
        for(ch = 'A'; ch <= 'H'; ch++)
            s.push(ch);
        System.out.println("Initial stack is: ");
        printStack(s);
        for(int i = 1; i <= 4; i++) {
            s.pop();
            q.enqueue(s.peek());
            s.pop();
            q.enqueue(s.peek());
            s.push(q.front());
            q.dequeue( );
        }
        System.out.println("\nFinal stack is: ");
        printStack(s);
        System.out.println("\nThe queue is: ");
        printQueue(q);
    }
    //method definition for printStack
    //method definition for printQueue
}
```

---

2. What does the following `mystery` method do? Assume that the method call is done for a queue holding the values: `1 2 3 4 5`

```
public static void mystery(QueueClass<Integer> q) {
     LinkedStackClass<Integer> s = new LinkedStackClass<Integer>();
     while(!q.isEmptyQueue()) {
         s.push(q.front());
         q.dequeue();
     }
     while(!s.isEmptyStack()) {
         q.enqueue(2 * s.peek());
         s.pop();
```

```
    }
}
```

---

3. What is the output generated by the following Java code?

```
//...
//INPUT: 15 28 14 22 64 35 19 32 7 11 13 30 -999
//ignore missing code like declarations for stack, queue, x, input...
//...
stack.initializeStack();
queue.initializeQueue();
stack.push(0);
queue.enqueue(0);
x = input.nextInt();
while(x != -999){
  switch(x % 4) {
    case 0:
      stack.push(x);
      break;
    case 1:
      if(!stack.isEmptyStack()) {
         System.out.println("Stack elem. = " + stack.peek());
         stack.pop();
      }
      else
         System.out.println("Stack is empty!");
      break;
    case 2:
      queue.enqueue(x);
      break;
    case 3:
      if(!queue.isEmptyQueue()) {
         System.out.println("Queue elem. = " + queue.front());
         queue.dequeue();
      }
      else
         System.out.println("Queue is empty!");
      break;
  }
  x = input.nextInt();
}
System.out.print("Stack elements: ");
while(!stack.isEmptyStack()) {
  System.out.print(stack.peek() + " ");
  stack.pop();
}
System.out.println();
System.out.print("Queue elements: ");
while(!queue.isEmptyQueue()) {
  System.out.print(queue.front() + " ");
  queue.dequeue();
```

```
}
System.out.println();
//...
```

PART 2: Programming

For the next problems write a client to test the following methods. Check out the **lecture notes** for the necessary files (`StackADT.java, LinkedStackClass.java, QueueADT.java, QueueClass.java, QueueException.java, QueueOverflowException.java,` and `QueueUnderflowException.java.`)

1. (Method: `printBackStack`) Task: Display the contents of a stack in reverse order (top to bottom), leaving the stack unchanged.

SOLUTION:

```
public static void printBackStack(LinkedStackClass<Integer> intStack) {
    ...
}
```

---

2. (Method: `printStack`) Task: Display the contents of a stack in direct order (bottom to top), leaving the stack unchanged.

SOLUTION:

```
public static void printStack(LinkedStackClass<Integer> intStack) {
    ...
}
```

---

3. (Method: `getSecond`) Task: Find the second item in a stack, leaving the stack unchanged.

SOLUTION:

```
public static Integer getSecond(LinkedStackClass<Integer> intStack) {
    ...
}
```

---

4. (Method: `countItems`) Task: Count the number of items in a stack, leaving the stack unchanged.

SOLUTION:

```
public static int countItems(LinkedStackClass<Integer> intStack) {
   ...
}
```

---

5. (Method: `removeItem`) Task: Delete every occurrence of a specified item from a stack, leaving the order of the remaining items unchanged.

SOLUTION:

```
public static void removeItem(LinkedStackClass<Integer> intStack, Integer n) {
   ...
}
```

---

6. (Method: `reverseStack`) Task: Reverse a stack, using a queue. The original stack is lost.

SOLUTION:

```
public static void reverseStack(LinkedStackClass<Integer> s) {
    QueueClass<Integer> q = new QueueClass<Integer>();
    ...
}
```

---

7. (Method: `reverseQueue`) Task: Reverse a queue, using a stack. The original queue is lost.

SOLUTION:

```
public static void reverseQueue(QueueClass<Integer> q){
      LinkedStackClass<Integer> s = new LinkedStackClass<Integer>();
      ...
}
```

---

8. (Method: `printQueue`) Task: Display the contents of a queue in direct order, leaving the queue unchanged.

SOLUTION:

```
public static void printQueue(QueueClass<Integer> q) {
       ...
}
```

SAMPLE OUTPUT:

```
Enter integers (999 to stop): 25 10 15 10 20 37 10 59 21 10 27 20 4 2 10 53 47 37 59
77 27 20 10 999
The original stack printed in direct order (bottom to top) is:
25 10 15 10 20 37 10 59 21 10 27 20 4 2 10 53 47 37 59 77 27 20 10
The stack printed in reverse order (top to bottom) is:
10 20 27 77 59 37 47 53 10 2 4 20 27 10 21 59 10 37 20 10 15 10 25
The stack stores 23 items.
The top is: 10
The second item (below top) is: 20
Enter value to be removed from stack: 10
The stack after removing every occurrence of 10 is:
25 15 20 37 59 21 27 20 4 2 53 47 37 59 77 27 20
Reversed the stack. The new stack printed in direct order is:
20 27 77 59 37 47 53 2 4 20 27 21 59 37 20 15 25

The queue is:
3 6 9 12 15 18 21 24 27 30
The reversed queue is:
30 27 24 21 18 15 12 9 6 3
```
