

SOLUTIONS

1. (8 pts) Consider the code fragment

```
String st = "[ (a) [ { ] } ]";  
balanced(st);
```

For the `balanced` method below (which is the same as we discussed in class),

- Write out the value of the stack as each character `c` is processed during the `balanced(st)` method call, until the method returns.
- What is the return value for the method call `balanced(st)`?

```
public static boolean balanced (String s){  
    Stack<Character> stack = new Stack<Character>();  
    for (int i=0; i < s.length(); i++){  
        Character c = s.charAt(i);  
        if (c == '{' || c == '(' || c == '[')  
            stack.push(c);  
        else if (c == '}'){  
            if (stack.empty() || stack.pop() != '{') return false;  
        }  
        else if (c == ')'){  
            if (stack.empty() || stack.pop() != '(') return false;  
        }  
        else if (c == ']'){  
            if (stack.empty() || stack.pop() != '[') return false;  
        } // end else  
    } // end for  
    return stack.empty();  
} // end balanced
```

Solution:

```
                '('  '('  '{'  
stack: '['  '['  '['  '['  '['  '['  
c:    '['  '('  'a'  ')'  ']'  '['  '{'  ']'
```

At this point, the method returns `false`, since the new character `']'` does not match the top of the stack `'{'`.

2. (a) (5 pts) Fully parenthesize the following Java expression (using the standard Java rules of precedence of operation and left-right associativity).

$a + 3 * b - 2 / (6 - c)$

Solution:

$((a + (3 * b)) - (2 / (6 - c)))$

(b) (5 pts) Write out the postfix version of the expression from part (a).

Solution:

$a 3 b * + 2 6 c - / -$

3. (6 pts) What is the output to the screen of the following code fragment?

```
Stack<Integer> s = new Stack<Integer>();
s.push(30);
for (int i=0; i<4; i++)
    s.push(i);
s.pop();
while (!s.empty())
    System.out.println(s.pop());
```

Solution:

2
1
0
30

4. (6 pts) What is the output to the screen of the following code fragment (assuming Queue is a class implementing QueueInterface with Java generics)?

```
Queue<String> q = new Queue<String>();
q.enqueue("John");
q.enqueue("Karl");
String st = q.peek();
q.enqueue("Beth");
q.enqueue(st);
st = q.dequeue();
q.enqueue(st);
while (! q.empty())
    System.out.println(q.dequeue());
```

Solution:

Karl
Beth
John
John

5. (6 pts) What is the output to the screen?

```
Stack<Integer> s = new Stack<Integer>();
Queue<Integer> q = new Queue<Integer>();
q.enqueue(5);
s.push(3);
s.push(q.peek());
q.enqueue(s.peek());
q.enqueue(7);
while (! q.empty())
    System.out.println(q.dequeue());
while (! s.empty())
    System.out.println(s.pop());
```

Solution:

```
5
5
7
5
3
```

6. (8 pts) Consider an implementation of `QueueReferenceBased` which implements `QueueInterface`, uses Java generics, and throws a `QueueException`. The implementation uses a linked list structure with both `head` and `tail` references (the `head` for the front of the queue, and the `tail` for the back). Recall the `Node<E>` class has accessor methods `getItem` and `getNext`.

Write the code for the method

```
public E dequeue() throws QueueException
```

Be sure to correctly account for the cases of dequeue-ing from queues containing only 0 or 1 element.

Solution:

```
public E dequeue() throws QueueException{
    if (head == null)
        throw new QueueException ("attempt to dequeue from "
            + "an empty queue");
    else{
        E ret = head.getItem();
        head = head.getNext();
        if (head == null)
            tail = null;
    }
}
```

```
        return ret;
    }
}
```

7. (8 pts) Write a method

```
public void printStack (Stack<E> s)
```

which prints out to the screen all the elements (from the top down) of the stack **s**, and then restores the stack to contain the same elements in the original order. (Hint: use an auxiliary stack to store the elements as they are printed out.)

Solution:

```
public void printStack (Stack<E> s){
    Stack<E> temp = new Stack<E>();
    while (! s.empty()){
        E curr = s.pop();
        System.out.println(curr);
        temp.push(curr);
    }
    while (! temp.empty())
        s.push( temp.pop() );
}
```