1. There are two ways in Java to define a "generic" `Pair` class that can hold references to two objects; there is an old way and a new way. Briefly sketch a definition for the `Pair` class using the old generic way, and then briefly sketch a definition for the `Pair` class using the new generic way. Your sketches only need to contain a default constructor, a two-parameter constructor, two appropriate get methods and two appropriate set methods.

2. The following two lines both generate compiler warnings. What is wrong with them?

```
Stack<String> stack1 = new Stack();
Stack stack2 = new Stack<String>();
```

3. While `String` is a sub-type of `Object` (that is, every `String` object "is a" `Object` object), it is **not** true that `Stack<String>` is a sub-type of `Stack<Object>`. Explain why. (**Hint:** If `A` "is a" `B`, then everything you can do with `B` you can also do with `A`. What can you do with a `Stack<Object>` that you cannot do with a `Stack<String>`?)

4. Why might the first group of statements print true, but the second print false?

```
Integer a1 = 100;
Integer a2 = 100;
System.out.println(a1 == a2);   // true

Integer b1 = 150;
Integer b2 = 150;
System.out.println(b1 == b2);   // false
```

5. What is wrong with the following code fragment?

```
List<String> listOfStrings = new ArrayList<String>();
String s;
for (s : listOfStrings)
   System.out.println(s);
```

6. (a) Suppose that `it` is an `Iterator` object. Write a small piece of Java code that prints all the objects of `it` to `System.out`.

   (b) Suppose that `it` is an `Iterator` object that returns `MyClass` objects. Write a small piece of Java code that inserts all the objects of `it` into a newly created `Queue` object.

7. Suppose we perform the following series of stack operations on a single, initially empty stack:

   push(5), *push*(3), *pop*(), *push*(2), *push*(8), *pop*(), *pop*(), *push*(9), *push*(1), *pop*(), *push*(7), *push*(6), *pop*(), *pop*(), *push*(4), *pop*(), *pop*().

   Draw a picture of the stack at the point where it contains the maximum number of elements (be sure to indicate the top and bottom of the stack). How many of the above operations had been performed at that point?

8. Suppose you have a stack `s` that contains `(1 2 3)`, with `1` being the top-of-stack, and a queue `q` that is empty. Using no other variables and only the `push()` and `pop()` stack operations and the `add()` and `remove()` queue operations, show a sequence of operations that leave the queue `q` empty and the stack `s` with each of the following contents.

   (a) Leave the stack `s` with the contents `(1 3 2)` with `1` as top-of-stack.

   (b) Leave the stack `s` with the contents `(3 1 2)` with `3` as top-of-stack.

9. Suppose we implement a stack using a partially-filled array. What is wrong with storing the top-of-stack at location `[0]` and the bottom of the stack at the last used position of the array?

10. If we use a partially-filled array to implement a queue, which is better and why? Having the front of the queue at location `[0]` (and the rear at the last used position of the array), or having the rear of the queue at location `[0]` (and the front at the last used position of the array).

11. If we use a linked list with a head and a tail reference to implement a stack, which is better and why? Having the top-of-stack at the head of the linked list, or having the top-of-stack at the tail of the linked list?

12. If we use a linked list with a head and a tail reference to implement a queue, which is better and why? Having the front of the queue at the head of the linked list, or having the front of the queue at the tail of the linked list?

13. Here is an incorrect pseudo code for an algorithm which is supposed to determine whether a String of parentheses is balanced:

```
boolean isBlanced( String input )
{
   declare a character stack
   while ( input has more characters )
   {
      read a character from input
      if ( the character is a '(' )
         push it on the stack
      else if ( the character is a ')' and the stack is not empty )
         pop a character off the stack
      else
         return false
    }
    return true
}
```

   Give an example of an input string that is made up of only the characters `'('` and `')'`, is unbalanced, but for which this algorithm will return true. Explain what is wrong with the algorithm. Can this algorithm ever incorrectly return false when its input string is a balanced string?

14. Convert the following expression from postfix to infix notation. Use the minimum number of parentheses needed.

```
6 3 2 4 + - *
```

15. Convert the following expressions from infix to postfix notation.

```
1 + 2 + 3 + 4
1 + (2 + (3 + 4))
1 + (2 + 3) + 4
2 * 3  * (9 + (3 - 1) + 4) * (5 - 1)
```

16. Using the two stack algorithm for evaluating fully parenthesized infix expressions, draw the contents of the two stacks just after the '4' token has been read from the following input String and processed by the algorithm.

```
"( ( ( 2 * 3 ) * ( 9 + ( ( 3 - 1 ) + 4 ) ) ) * ( 5 - 1 ) )"
```

17. Write a recursive method with two integer parameters, m and n, and the precondition that $0 \leq m \leq n$. The method should print a line of m asterisks, then a line of $m + 1$ asterisks, and so on up to a line of n asterisks. Then the pattern is repeated backward: a line of $n - 1$ asterisks, then $n - 2$, and so on down to m. The only loop allowed in your implementation is a loop to print a line of m asterisks. You may have two copies of this loop in different places of the implementation. If $m = 5$ and $n = 8$, then the output should look like this.

```
*****
******
*******
********
*******
******
*****
```

18. A `Queue` object can be implemented using two `Stack` objects. Below is an outline of how to do this. Complete the implementations of the `add()` and `remove()` methods.

The key idea is that if you push all the elements from one stack onto an empty stack, then the items get reversed and what was the top-of-stack on the original stack becomes the bottom of the new stack.

In the implementation below, one of the stacks is used to implement the `add()` method and the other stack is for the `remove()` method. It should always be the case that one of the two stacks is empty and the contents of the queue are in the other stack. By shifting the elements from one stack to the other, you can move the front of the queue to the top of a stack, or you can move the rear of the queue to the top of a stack.

```
import java.util.Stack;
public class Queue<T>
{
   private Stack<T> forAdding   = new Stack<T>();
   private Stack<T> forRemoving = new Stack<T>();

   public void add(T item)
   {








   }


   public T remove( )
   {
      T result = null;








      return result;
   }
}
```

19. A `Stack` object can be implemented using one `Queue` object. Below is an outline of how to do this. Complete the implementations of the `push()` and `pop()` methods.

    Hint: To pop an item from the stack, get all of the items from the queue one at a time and put them at the rear, except for the last one which you should remove and return.

```
import java.util.Queue;
public class Stack<T>
{
   private Queue<T> queue = new Queue<T>();

   public void push(T item)
   {




   }


   public T pop( )
   {
      T result = null;










      return result;
   }
}
```

12-03-2012 at 02:37 h