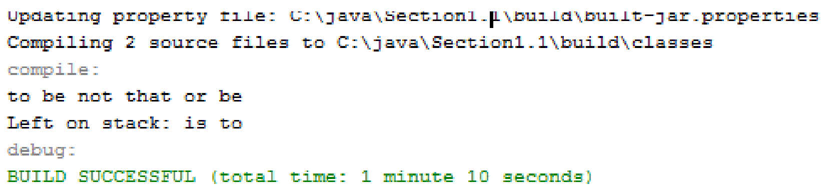
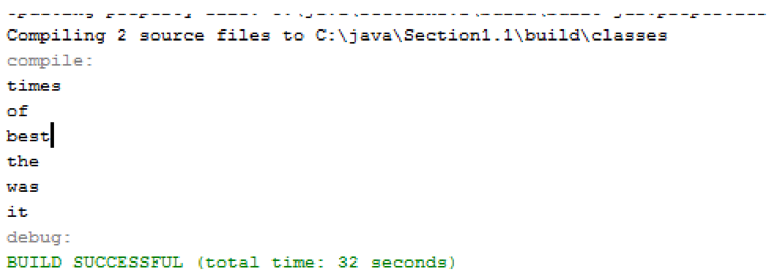
Activity 9

1. Add a method `isFull()` to ArrayStackOfStrings.java.



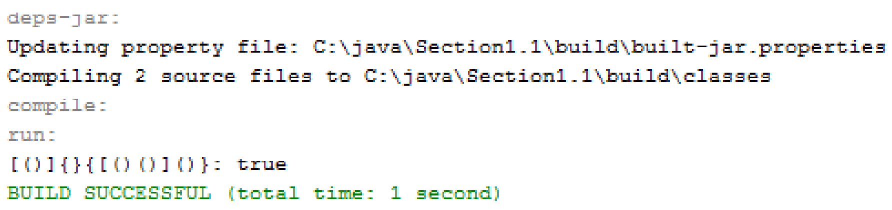
```
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
to be not that or be
Left on stack: is to
debug:
BUILD SUCCESSFUL (total time: 1 minute 10 seconds)
```

2. Write a stack client Reverse.java that reds in strings from standard input and prints them in reverse order.



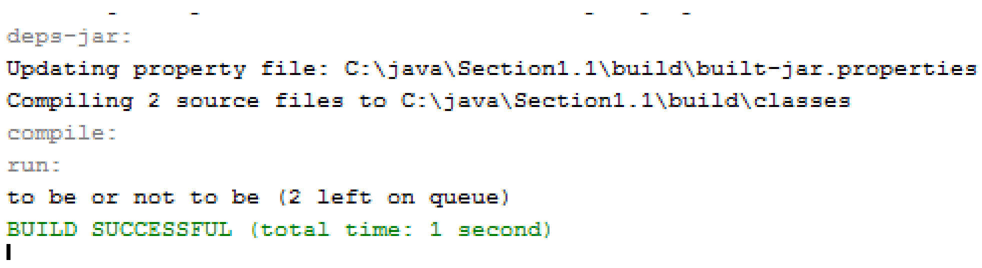
```
[illegible]
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
times
of
best
the
was
it
debug:
BUILD SUCCESSFUL (total time: 32 seconds)
```

3. Write a stack client Parentheses.java that reads in a text stream from standard input and uses a stack to determine whether its parentheses are properly balanced. For example, your program should print `true` for`[()]{}{[()()]()}` and false for `[(])`. Hint : Use a stack.



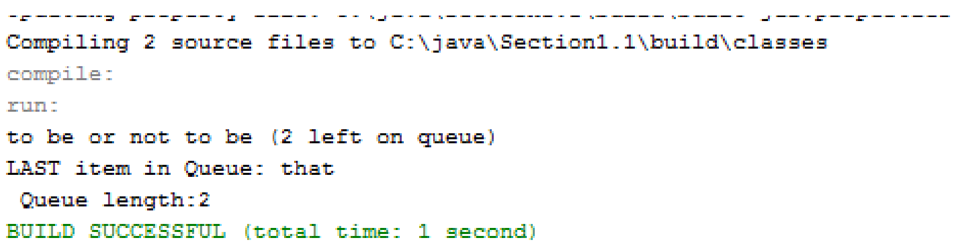
```
deps-jar:
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
run:
[()]{}{[()()]()}: true
BUILD SUCCESSFUL (total time: 1 second)
```

4. Add a method `length()` to Queue.java that returns the number of elements on the queue. Hint: Make sure that your method takes constant time by maintaining an instance variable `N` that you initialize to 0, increment in `enqueue()`, decrement in `dequeue()`, and return in `length()`.



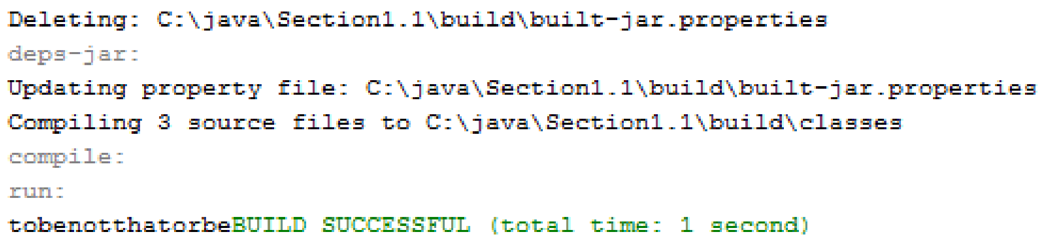
```
deps-jar:
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
run:
to be or not to be (2 left on queue)
BUILD SUCCESSFUL (total time: 1 second)
```

5. Develop a class DoublingQueueOfStrings.java that implements the queue abstraction with a fixed-size array, and then extend your implementation to use array doubling to remove the size restriction.
6. Write a Queue.java client that takes a command-line argument `k` and prints the `k`th from the last string found on standard input.



```
[illegible]
Compiling 2 source files to C:\java\Section1.1\build\classes
compile:
run:
to be or not to be (2 left on queue)
LAST item in Queue: that
 Queue length:2
BUILD SUCCESSFUL (total time: 1 second)
```

7. (For the mathematically inclined.) Prove that the array in DoublingStackOfStrings.java is never less than one-quarter full. Then prove that, for any `DoublingStackOfStrings` client, the total cost of all of the stack operations divided by the number of operations is a constant.



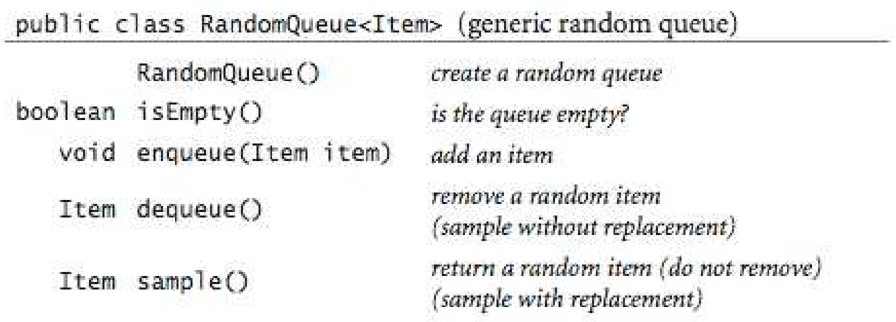
```
Deleting: C:\java\Section1.1\build\built-jar.properties
deps-jar:
Updating property file: C:\java\Section1.1\build\built-jar.properties
Compiling 3 source files to C:\java\Section1.1\build\classes
compile:
run:
tobenotthatorbeBUILD SUCCESSFUL (total time: 1 second)
```

8. Write a method `delete()` that takes an `int` argument `k` and deletes the `k`th element in a linked list, if it exists.

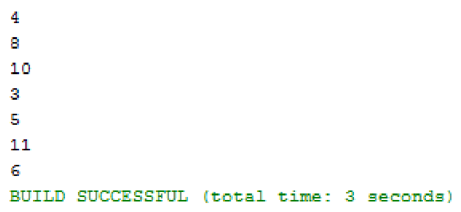
   Solution.

```
// we assume that first is a reference to the first Node in the
list
public void delete(int k) {
    if (k <= 0) throw new RuntimeException("Invalid value of k");

    // degenerate case - empty linked list
    if (first == null) return;

    // special case - removing the first node
    if (k == 1) {
        first = first.next;
        return;
    }

    // general case, make temp point to the (k-1)st node
    Node temp = first;
    for (int i = 2; i < k; i++) {
        temp = temp.next;
        if (temp == null) return;   // list has < k nodes
    }

    if (temp.next == null) return;  // list has < k nodes

    // change temp.next to skip kth node
    temp.next = temp.next.next;
}
```

9. **Random queue.** Create an abstract data type `RandomizedQueue.java` that supports the following operations: `isEmpty()`, `insert()`, `random()`, and `removeRandom()`, where the deletion operation deletes and returns a random object. *Hint:* maintain an array of objects. To delete an object, swap a random object (indexed 0 through N-1) with the last object (index N-1). Then, delete and return the last object.

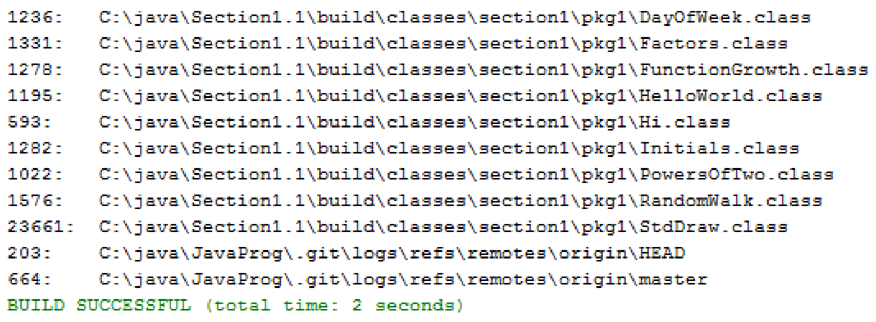


public class RandomQueue<Item>		(generic random queue)
	`RandomQueue()`	*create a random queue*
`boolean`	`isEmpty()`	*is the queue empty?*
`void`	`enqueue(Item item)`	*add an item*
`Item`	`dequeue()`	*remove a random item (sample without replacement)*
`Item`	`sample()`	*return a random item (do not remove) (sample with replacement)*



```
4
8
10
3
5
11
6
BUILD SUCCESSFUL (total time: 3 seconds)
```

10. **Listing files.** A Unix directory is a list of files and directories. Program Directory.java takes the name of a directory as a command line parameter and prints out all of the files contained in that directory (and any subdirectories) in level-order. It uses a queue.



```
1236:   C:\java\Section1.1\build\classes\section1\pkg1\DayOfWeek.class
1331:   C:\java\Section1.1\build\classes\section1\pkg1\Factors.class
1278:   C:\java\Section1.1\build\classes\section1\pkg1\FunctionGrowth.class
1195:   C:\java\Section1.1\build\classes\section1\pkg1\HelloWorld.class
593:    C:\java\Section1.1\build\classes\section1\pkg1\Hi.class
1282:   C:\java\Section1.1\build\classes\section1\pkg1\Initials.class
1022:   C:\java\Section1.1\build\classes\section1\pkg1\PowersOfTwo.class
1576:   C:\java\Section1.1\build\classes\section1\pkg1\RandomWalk.class
23661:  C:\java\Section1.1\build\classes\section1\pkg1\StdDraw.class
203:    C:\java\JavaProg\.git\logs\refs\remotes\origin\HEAD
664:    C:\java\JavaProg\.git\logs\refs\remotes\origin\master
BUILD SUCCESSFUL (total time: 2 seconds)
```