**Before Interview**

1. Set up Java workspace with Stack and StackTest and Junit pre-configured.

**During Interview**

1. Explain What is Pairing Programming and Why communication is Important
   1. Evaluating communication
2. Explain what is TDD
   1. Evaluating testing ability
3. Explain Stack, Node, and No Collection Library

**Exercise:**

Empty (Warmup):

1. Explain Behavior of isEmpty method
2. Interviewer types out empty test with only empty Stack
3. Interviewer types out stub for isEmpty and return false
4. Run Test and see that it fails
5. **Question 1**: Ask Candidate what is the simplest thing to do.
   1. Default answer is return true
6. Run Test and see that it passes
7. Explain that empty is a warm up exercise and tell candidate to relax

Push:

1. Explain to candidate that this is a static implementation and we need to improve on the test
2. Interviewer types out test for one and many Stacks
3. Interviewer types out stub for Push
4. Push is a void method that takes an object “o” as a parameter
5. Run Test and see that it fails
6. **Question 2**: Ask candidate what is the simplest thing to pass all the test
   1. Default Answer is using a Boolean variable
7. Run Test and see that it passes

Peek:

1. Interviewer explains Peek: look at top element, but don’t remove it
2. Interviewer types out test for empty and one
3. Interviewer types out stub for Peek
4. **Question 3:** Ask candidate how to test for many for Peek
   1. assertEquals(“b”, manyStack.peek())
5. Run test and see that it fails
6. **Question 4:** Ask candidate what the simplest way to implement peek is
   1. Default answer is an Object class variable that stores the last element added
7. Interviewer should prompt candidate to refactor Push to update the new variable. IsEmpty Boolean can also be factored out.
8. Run Test and see that it passes

Pop:

1. Interviewer explains Pop: remove and return the top element
2. **Question 5:** Ask the candidate to write some tests for Pop
   1. Should (at least) write a test for an untouched stack, a stack that was already popped once, and an empty stack
3. Interviewer types out stub for Pop
4. **Question 6:** Ask the candidate to implement Pop
   1. Default answer should be to start using a Node variable (“head” of the stack)
5. Interviewer should prompt candidate to refactor Peek, Push, and isEmpty to use new Node variable.
6. Run tests and see that it passes
7. **Question 7:**  Ask the candidate to write tests for Pop and Peek together
   1. Should write a test to pop then peek the single stack
   2. Should write a test to pop then peek the many stack

Contains:

1. Interviewer explains Contains method
2. Method should return a Boolean and take an object “o” as a parameter
3. Interviewer types out stub for contains
4. **Question 8:** Ask the candidate to write test cases for Contains
   1. Should write assertTrue and assertFalse tests for single stack
   2. Should write an assertFalse test for the empty stack
   3. Should write assertTrue and assertFalse tests for many stack
   4. Should write test of testing contain after pop
5. Run test and see that it fails
6. **Question 9:** Ask the candidate to implement contains
   1. One answer is using a for-loop or while-loop, iterating until element is found at head, otherwise false.
   2. Other option is to recursively call a helper method on head, until element is found, otherwise false.
7. Run tests and see that is passes

Size:

1. Interviewer explains the Size method
2. Method returns an Integer and takes no parameters
3. Interviewer types out the stub for Size
4. **Question 10:** Ask the candidate to write test cases for Size
   1. Should write a test for each stack to test their starting sizes
   2. Then, for each stack (empty, single, many), they should Pop, then test the Stack’s size.
5. Run test and see that if fails
6. **Question 11:** Ask the candidate to implement Size
   1. Default answer should be an Integer size variable
   2. It should be incremented during Push, and decremented during Pop, and left alone during Peek.
7. If the candidate did not write a test to check the empty set after a pop, then show them how this could cause size to become negative.
8. Run tests and see that it passes

PeekN (Bonus):

1. If the candidate has made it this far, done very well, and there is time remaining, you can also ask them this question for some bonus points.
2. Interviewer explains PeekN to the candidate
3. PeekN returns an object and takes an Integer as a parameter. It finds the value at that index in the stack, returns it, but does not remove it.
4. Interviewer types stub for PeekN
5. **Question 12**: Ask the candidate to write some tests for PeekN
   1. Should write test for the empty stack, where peeking at any index will return null
   2. Test for the single stack at index 0
   3. Test for the many stack for at least index 1 and 2
   4. Test for the many stack at index 3 or greater, where it should be null.
6. Run test and see that it fails
7. **Question 13:** Ask the candidate to implement PeekN. Whatever solution the candidate used for Contains, prompt them to use the other (i.e. if they used a loop for contains, they should use recursion here)
   1. Default answer should be to iterate over the stack, much like Contains, until the desired index is found, and that value returned.
8. Run tests and see that it passes