Thomas Hart
CS 329
Midterm 1

Problem 1

```
/**
 * Returns the max capacity of the stack
 *
 * @ensures capacity returned = max capacity specified
 * @ensures capacity >= 1
 *
 * @return (capacity : int) the max capacity of the stack
 */
int capacity();

/**
 *  Add an item on the top
 *
 * @requires x != null
 * @requires x type = Integer
 * @requires current size < max capacity
 *
 * @ensures new(size) == old(size) + 1
 * @ensures new(size) <= max capacity
 * @ensures new(stack) == old(stack) + x
 * @ensures pop(new(stack)) == x
 *
 * @param x the element to push to the stack
 */
void push(Integer x);

/**
 * Remove the item at the top
 *
 * @requires current size > 0
 *
 * @ensures new(size) == old(size) - 1
 * @ensures new(size) >= 0
 * @ensures new(stack) == old(stack) - top of stack
 */
void pop();

/**
```

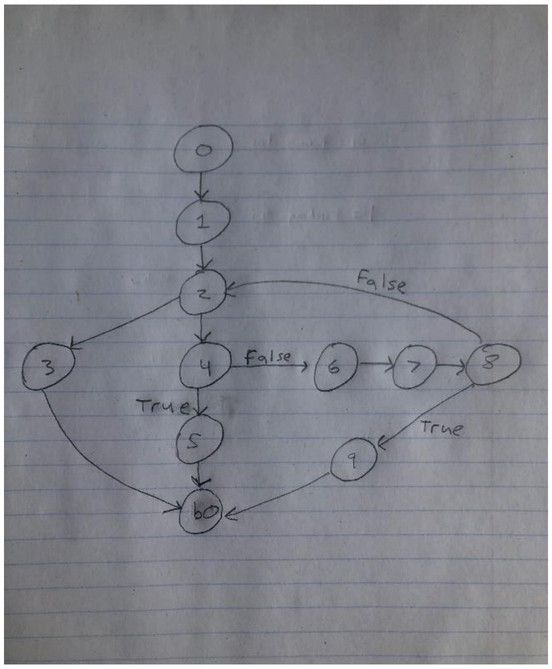
```
 * Return the item at the top (without removing it) -- assume peek can't be called if stack is empty
 *
 * @requires current size > 0
 *
 * @ensures element returned != null
 * @ensures element returned == top of stack
 * @ensures size(old) == size(new)
 *
 * @return (top : Integer) the element at the top of the stack
 */
Integer peek();

/**
 * Return the number of items in the stack
 *
 * @ensures 0 <= size < max capacity
 * @ensures size == number of items on the stack
 *
 * @return (size : int) number of items on the stack
 */
int size();
```

Problem 5

This visitor looks through an abstract syntax tree for IfStatement, WhileStatement, ForStatement, and SwitchStatement nodes. As it visits each node, it adds 1 to the "value" parameter, which would indicate how many nodes are currently being visited. It also adds the node to the "nodes" ArrayList if "value" is greater than 3. The "endVisit" functions for each node decrement the value to indicate that one fewer node is being visited. Based on all these things, getResultingNodes() returns the "nodes" ArrayList, which seems to carry all the blocks that are nested within 4 or more parent blocks. My guess for the application of this visitor is that a program might become too complicated if a block is nested within 4 or more other blocks. This visitor would detect and return any of those blocks.

Problem 6

CFG for the java method:



Problem 7

A)

Block	**Kill Set**	**Gen Set**
0	(i, *)	(i, 0)
1	∅	∅
2	(i, *)	(i, 2)
3	∅	∅
4	∅	∅
5	(i, *)	(i, 5)
6	(i, *)	(i, 6)
b0	∅	∅

A)

Block	**Entry Set**	**Exit Set**
0	∅	(i, 0)
1	(i, 0)	(i, 0)
2	(i, 0)	(i, 2)
3	(i, 2), (i, 6)	(i, 2), (i, 6)
4	(i, 2), (i, 6)	(i, 2), (i, 6)
5	(i, 2), (i, 6)	(i, 5)
6	(i, 2), (i, 6)	(i, 2), (i, 6)
b0	(i, 0), (i, 2), (i, 5), (i, 6)	(i, 0), (i, 2), (i, 5), (i, 6)

Problem 8

A)

Some interesting inputs would include strings that are supposed to fail such as too many operators, too many numbers, an order of numbers and operators that don't make sense, and "divide by zero" situations. We would also test each operator to make sure they function properly with negative and non-negative numbers. We could also try using extreme numbers in positive and negative directions to see if there are boundary limitations.

B)

With the given methods, Mockito would be useful. If we already knew the expected output for a given string, for example, we could use the "when-thenReturn" on the peek() method to return that value to any class that depends on the computation engine. We could also use the "verify" method to make sure the class using the computation engine produced the correct calls to pushOperand() and applyOperator(), and called them in the correct order.

In a test, you could create a mock of the computation engine and pass it to the class that uses it. You could then pass a predetermined string to that class so that it would generate the sequence of calls from the mocked object. You could call Mockito.verify(ce.pushOperand(15)), for example, along with other verify calls to make sure all functions were called the appropriate amount of times. We could also call Mockito.when(ce.peek()).thenReturn(5) to return the expected value for use within the class using the computation engine. This way, we don't need to make an actual computation engine to test the class using it.