CS 284 A/B: Mid-Term (45 Minutes)
Spring 2016

March 8, 2016

Student Name:

Honor Pledge:

Grade sheet:

Problem 1 (15 points)	
Problem 2 (25 points)	
Problem 3 (25 points)	
Problem 4 (15 points)	
Problem 5 (20 points)	

Problems

Problem 1. For each of the following, indicate whether they are examples of white-box testing or black-box testing.

1. Testing the boundary conditions of a for loop inside a method

   White box

2. Testing the behavior of the program if the input file is not found

   Black box

3. Determining if the re-allocation of an array-based list is performed at the right time

   White box

4. Testing whether a method throws an exception on a specific input

   Black box

Problem 2. Indicate for each of the following code fragments:

1. the number of times that it prints out on the screen and
2. the complexity using $\mathcal{O}$ notation.

You may assume that $n > 2$. In case you might require it, here is the formula for the sum of the numbers from 1 to n: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$.

```
// a
for(i=0; i<n; i++) {
  for(j=0; j<i+2; j++) {
    System.out.println(i + " " + j);
  }
}
```

Number of times: $\frac{(n+1)(n+2)}{2} - 1$, since $\sum_{i=2}^{n+1} i = (\sum_{i=1}^{n+1} i) - 1 = \frac{(n+1)(n+2)}{2} - 1$
Complexity: $\mathcal{O}(n^2)$

```
// b
for(i=0; i<n; i++) {
  if (i % 3 == 0) {
    for(j=0; j<n; j++) {
          System.out.println(i + " " + j);
    }
  }
}
```

Number of times: $\lceil \frac{n}{3} \rceil \cdot n$
Complexity: $\mathcal{O}(n^2)$

Problem 3. Implement an operation to be included in the class SingleLinkedList

public boolean removeLast()

that removes the last element in a single linked list. If the list is empty, it should return false, otherwise it should return true. Recall that the class SingleLinkedList<E> has an inner class Node<E>, depicted below, and that it has two data fields, head (that refers to the first node in the list) and size. In your implementation, you may not use any other operations from this list, or from any other class.

```
private static class Node<E> {

        private E data;
        private Node<E> next = null;

        public Node(E data, Node<E> next) {
            this.data = data;
            this.next = next;
        }

        public Node(E data) {
            this(data, null);
        }
    }
```

```
public boolean removeLast() {
        if (size == 0) {
                return false;
        } else if (size == 1) {
                head = null;
                size = 0;
                return true;
        } else {
                Node<E> current = head;
                for (int i = 0; i < size - 2; i++) {
                        current = current.next;
                }
                current.next = null;
                size--;
                return true;
        }
}
```

This page is intentionally left blank. You may use it for writing your answers.

Problem 4. Suppose that you are given a class that implements a Stack and you execute the following lines of code:

```
public static void main(String[] args){
  Stack<Integer> myStack = new Stack<Integer>();
  myStack.push(4);
  myStack.push(7);
  myStack.push(2);
  x = myStack.pop();
  y = myStack.peek();
  myStack.push(x+y);
  z = myStack.peek();
  w = myStack.pop();
}
```

What are the values of:

- $x =$
- $y =$
- $z =$
- $w =$

- $x = 2$
- $y = 7$
- $z = 9$
- $w = 9$

Problem 5. Which of these two implementations of SingleLinkedList.toString(), both correct, is more efficient? Why?

```
// Implementation 1
public String toString() {
  StringBuilder sb = new StringBuilder("[");
  if (size>0) {
    for (int i=0; i<size-1; i++) {
      sb.append(this.get(i).toString());
      sb.append(" ==> ");
    }
    sb.append(this.get(size-1).toString());
  }
  sb.append("]");
  return sb.toString();
}

// Implementation 2
public String toString() {
  StringBuilder sb = new StringBuilder("[");
  Node<E> p = head;
  if (p != null) {
    while (p.next != null) {
      sb.append(p.data.toString());
      sb.append(" ==> ");
      p = p.next;
    }
    sb.append(p.data.toString());
  }
  sb.append("]");
  return sb.toString();
}
```

You may assume the following implementation of SingleLinkedList.get(**int** index):

```
  private Node<E> getNode(int index) {
    Node<E> node = head;
    for (int i = 0; i < index && node != null; i++) {
      node = node.next;
    }
    return node;
  }

  public E get(int index) {
    if (index < 0 || index >= size) {
      throw new IndexOutOfBoundsException(Integer.toString(index));
    }
    Node<E> node = getNode(index);
    return node.data;
}
```

This page is intentionally left blank. You may use it for writing your answers.

Implementation 2 is more efficient. The reason is that implementation 1 requires $\mathcal{O}(\text{size}^2)$ node accesses (size is the number of nodes in the list), whereas implementation 2 requires merely $\mathcal{O}(\text{size})$ node accesses.

Since implementation 1 calls `getNode(i)` for all $i \in \{0, 1, \ldots, \text{size}-1\}$ and `getNode(i)` accesses $i + 1$ nodes, implementation 1 accesses $\sum_{i=1}^{\text{size}} i = \frac{\text{size} \cdot (\text{size} + 1)}{2} = \mathcal{O}(\text{size}^2)$ nodes.

Implementation 2 goes through all nodes in the list starting from the head, thereby accessing each node only once. Hence, implementation 2 accesses $\mathcal{O}(\text{size})$ nodes.