0. (4 points) Write your name and TA’s name above and read and sign the box below.

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
</tr>
</tbody>
</table>

READ AND SIGN THIS:

I certify that the answers on this exam are all my own work, and that I have not discussed the exam questions or answers with anyone in the class who has already taken the exam. I also will not (directly or indirectly) discuss the exam questions or answers with anyone in the class who has not yet taken the exam.
1. (5 points) In some methods, you wrote code to check if a certain precondition was held. If the precondition did not hold, then you threw an exception. This leads to robust code by catching client code misusing your methods. This seems like a great idea, so when would you **NOT** want to check for the precondition in a method? Why wouldn’t you? (You may use no more than 30 words.)

Answers (or close derivatives thereof) we accepted:
- It would be too inefficient to do so.
- The precondition is ensured by the other methods in the class.

2. (5 points) Recall the definition of the ListNode class:

   ```java
   public class ListNode {
       int data;
       ListNode next;
   }
   ```

   We can print a linked list using the following method:

   ```java
   public static void print(ListNode front) {
       ListNode current = front;
       while (current != null) {
           System.out.println(current.data + " ");
           current = current.next;
           if (current == front) {
               break;
           }
       }
   }
   ```

   Starting from the node pointed to by `front`, draw one or more nodes and link them together in such a way that the call `print(front)` will not “behave” properly.
3. (8 points) Consider the following method:

```java
public void mystery(int n) {
    if (n >= 7) {
        System.out.println(n);
    } else {
        System.out.print(n);
        System.out.print(n);
        mystery(n + 7);
    }
}
```

For each call below, indicate what output is produced by the method. If the call results in infinite recursion, write out the first 5 characters followed by “…”.

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Output Produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery(-1)</td>
<td>-1-6613</td>
</tr>
<tr>
<td>mystery(711)</td>
<td>711</td>
</tr>
<tr>
<td>mystery(0)</td>
<td>007</td>
</tr>
<tr>
<td>mystery(-5)</td>
<td>-5-229</td>
</tr>
</tbody>
</table>

4. (8 points) Consider the following method:

```java
public void mystery(int n) {
    if (n == 0) {
        System.out.print(n);
    } else {
        System.out.print("(");
        mystery(n - 1);
        System.out.print(n);
        mystery(n - 1);
        System.out.print(")");
    }
}
```

For each call below, indicate what output is produced by the method. If the call results in infinite recursion, write out the first 5 characters followed by “…”.

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Output Produced</th>
</tr>
</thead>
<tbody>
<tr>
<td>mystery(2)</td>
<td>((010)2(010))</td>
</tr>
<tr>
<td>mystery(0)</td>
<td>0</td>
</tr>
<tr>
<td>mystery(-1)</td>
<td>((((((...</td>
</tr>
<tr>
<td>mystery(1)</td>
<td>(010)</td>
</tr>
</tbody>
</table>
5. (20 points) A palindrome reads the same backward or forward. The number 12321 is a palindrome, because if you start from the left side or the right side, it reads exactly the same way. Write a method `isPalindrome` that takes an integer array `nums` and returns `true` or `false` if the array of numbers constitute a palindrome in the sense that the first element matches the last element; the second element matches the second-to-last element, and so on. Do **NOT** use recursion.

Examples where `isPalindrome` would return `true`:

```
123  2  24  2  123
```

```
123  2  214  99  99  214  2  123
```

Examples where `isPalindrome` would return `false`:

```
123  2  24  1  123
```

```
123  2  214  99  98  214  2  125
```

```java
public boolean isPalindrome(int[] nums) {
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] != nums[nums.length - 1 - i]) {
            return false;
        }
    }
    return true;
}
```

If your solution to this problem is blank, you will receive ¼ credit (5 points).
6. (10 points) What is the running time of this method? Circle one: O(1) O(n)

Public int add100(int[] array) {
    if (array.length < 100) {
        return 0;
    }

    int sum = 0;
    for (int i = 0; i < 100; i++) {
        sum += array[i];
    }
    return sum;
}

Explain your answer in 30 words or less.

We are interested in the running time as the array gets larger and larger. After the array exceeds a size of 100, the size of the array is irrelevant, and a constant number of steps is needed to solve the problem.
7. (20 points) Recall the definition of the ListNode:

```java
public class ListNode {
    int data;
    ListNode next;
}
```

Assume that we define a class SortedLinkedIntList that is similar to LinkedIntList, except that the data stored in the nodes are in sorted order:

```java
public class SortedLinkedIntList {
    private ListNode front;

    <methods>
}
```

Write a method mode(void) for the SortedLinkedIntList class that returns the mode of the numbers in the list. The mode of a set of numbers is the number that appears the most frequently. If there is a tie, return any of the numbers that share the highest frequency. If the list is empty, throw an IllegalStateException.

For example, if the list contained [1,1,1,4,4,4,4,4,4,5,6,6,70,99,99], then a call to mode() would return 4, since it appears the most frequently (six times). If the list contained [1,4,4,7,7,11,25,25,99], then your method can return either 4, 7, or 25, because they all share the highest frequency (two times).

Answer on next page.

If your solution to this problem is blank, you will receive ¼ credit (5 points).
public int mode(void) {
    if (front == null) {
        throw new IllegalStateException();
    }

    int value = front.data;
    int maxRun = 1;
    int currentRun = 1;

    ListNode current = front.next;
    ListNode last = front;
    while (current != null) {
        if (current.data == last.data) {
            currentRun++;
            if (currentRun > maxRun) {
                maxRun = currentRun;
                value = current.data;
            } else {
                currentRun = 1;
            }
        }
        last = current;
        current = current.next;
    }
    return value;
}
Stack Interface
---------------
// Interface Stack defines a set of operations for manipulating a
// LIFO (Last In First Out) structure that can be used to store
// objects.

public interface Stack {
    // post: given value is pushed onto the top of the stack
    public void push(Object value);

    // pre : !isEmpty()
    // post: removes and returns the value at the top of the stack
    public Object pop();

    // post: returns true if the stack is empty, false otherwise
    public boolean isEmpty();

    // post: returns the current number of element in the stack
    public int size();
}

Queue Interface
----------------
// Interface Queue defines a set of operations for manipulating a
// FIFO (First In First Out) structure that can be used to store
// objects.

public interface Queue {
    // post: given value inserted at the end of the queue
    public void enqueue(Object value);

    // pre : !isEmpty()
    // post: removes and returns the value at the front of the queue
    public Object dequeue();

    // post: returns true if the queue is empty, false otherwise
    public boolean isEmpty();

    // post: returns the current number of element in the queue
    public int size();
}
8. (30 points) Write a method `pushNumTimes` that takes a `Queue q` as an argument and returns a `Stack`. Assume that `q` is storing `Integer` objects. Each integer value `i` at position `n` (where the object at the front of the queue has position 1) in the queue will be replaced by an `Integer` object with an integer value of `(i * n)` on the stack. The contents of the queue do **not** have to be preserved.

For illustration purposes, let a queue’s contents be represented as a list of numbers, where the leftmost number represents the front of the queue; let a stack’s contents be represented as a list of numbers where the leftmost number represents the top of the stack. Suppose `q` initially contained `[6, 4, 5, 3]`, then the stack returned from a call to `pushNumTimes(q)` will have `[6, 8, 15, 12]`.

For your convenience, the `Queue` and `Stack` interfaces are on the previous page. The names of the classes that implement those interfaces are `LinkedQueue` and `ArrayStack`, respectively. Both classes have constructors that take no arguments.

If `q` initially contained:

<table>
<thead>
<tr>
<th>position:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>value:</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Then the returned stack would contain:

| 6     |
| 8     |
| 15    |
| 12    |

```java
go public Stack pushNumTimes(Queue q) {
    Stack s = new ArrayStack();
    int counter = 1;
    while (!queue.isEmpty()) {
        int n = ((Integer)queue.dequeue()).intValue();
        s.push(new Integer(n * counter));
        counter++;
    }
    while (!s.isEmpty()) {
        q.enqueue(s.pop());
    }
    while (!q.isEmpty()) {
        s.push(q.dequeue());
    }
    return s;
}
```
9. (20 points) Consider the following definitions:

```java
public class Apple extends Date {
    public void method2() {
        System.out.println("Apple2");
    }
}

public class Banana {
    public void method2() {
        System.out.println("Banana2");
    }
}

public class Cherry extends Date {
    public void method2() {
        System.out.println("Cherry2");
    }
    public void method1() {
        super.method1();
        System.out.println("Cherry1");
    }
}

public class Date extends Banana {
    public void method1() {
        System.out.println("Date1");
    }
    public void method2() {
        System.out.println("Date2");
        method1();
    }
}
```

And assuming the following variables have been defined:

```java
Object var1 = new Cherry();
Banana var2 = new Date();
Banana var3 = new Cherry();
Apple var4 = new Apple();
```
In the table below, indicate in the right-hand column the output produced by the statement in the left-hand column. If the statement produces more than one line of output, indicate the line breaks with slashes as in "a/b/c" to indicate three lines of output with "a" followed by "b" followed by "c". If the statement causes an error, fill in the right-hand column with either the phrase "compiler error" or "runtime error" to indicate when the error would be detected.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>var1.method1();</td>
<td>compiler error</td>
</tr>
<tr>
<td>var2.method1();</td>
<td>compiler error</td>
</tr>
<tr>
<td>var3.method1();</td>
<td>compiler error</td>
</tr>
<tr>
<td>var4.method1();</td>
<td>Date1</td>
</tr>
<tr>
<td>((Cherry)var2).method1();</td>
<td>runtime error</td>
</tr>
<tr>
<td>((Date)var2).method2();</td>
<td>Date2 / Date1</td>
</tr>
<tr>
<td>((Date)var1).method2();</td>
<td>Cherry2</td>
</tr>
<tr>
<td>((Cherry)var1).method1();</td>
<td>Date1 / Cherry1</td>
</tr>
<tr>
<td>((Cherry)var4).method1();</td>
<td>compiler error</td>
</tr>
<tr>
<td>((Date)var3).method2();</td>
<td>Cherry2</td>
</tr>
</tbody>
</table>
10. (20 points) **Using recursion**, write a method `times` that takes two integers `a` and `b` as parameters and returns their product, *i.e.*, `a * b`. You are to multiply them by using a series of additions. You may use `+`, `-`, comparison operators (`>`, `>=`, `<`, `<=`, `==`, `!=`), and Boolean operators (`&&`, `||`). You may NOT use `*`, `/`, any loops (*i.e.*, `for` or `while`) or any method from the Java library.

You may find the following equality useful: \( a \times b = b + (a - 1) \times b \)

For at most \( \frac{3}{4} \) credit (*i.e.*, 15 points), you may assume that both arguments are non-negative. If you would like that option, sign here: ____________________________

```java
public int times(int a, int b) {
    if (a < 0) {
        return -times(-a, b);
    }
    if (a == 0) {
        return 0;
    }
    return b + times(a-1, b);
}
```

If your solution to this problem is blank, you will receive \( \frac{1}{4} \) credit (5 points).
11. (BONUS: 5 points) Tell us a funny story or joke. If your submission cracks everyone on the course staff up, you’ll get an extra smiley face. If you’re not a funny person, you may draw your TA. If it looks like you spent more than a minute, you get full-credit.

If you didn’t leave this page blank, we gave you 5 extra points.