MIDTERM 1 REVIEW SOLUTIONS

INTRODUCTION:

The goal of this section will be to mix in the explanatory elements of a regular discussion with the more personalized format of a guerilla section. When checking a group's solutions, try to ask each member of the group a question. You can break down the problem into multiple parts and spread your questions among the group.

EXCEPTIONS

Q1 Except Me for Who I am

Consider the following:

```
1  public class IntList {
2      private int head;
3      private IntList tail;
4
5      /* Returns the index of an element in the list */
6      public int getIndex(int item) {
7          int index = 0;
8          IntList temp = this;
9          while(temp.head != item) {
10             temp = temp.tail;
11             index++;
12         }
13         return index;
14     }
15
16     public int getIndexThrowException(int item) throws IllegalArgumentException {
17         // YOUR CODE HERE
18     }
19
20     public int getIndexDefaultNegative(int item) {
21         // YOUR CODE HERE
22     }
23 }
```

(a) What happens when you call `getIndex(int item)` on an element that is not in the list?
You will run into a NullPointerException.

(b) Write `getIndexThrowException`, which attempts to get the index of an item, but throws an `IllegalArgumentException` with a useful message if no such item exists in the list. Do not use if statements, while loops, for loops, or recursion. (Hint: you can use `get(int item)`)

(c) Write `getIndexDefaultNegative`, which attempts to get the index of an item, but returns -1 if no such item exists in the list. Again, do not use if statements, while loops, for loops, or recursion.

```java
public int getIndexThrowException(int item) throws IllegalArgumentException {
    try {
        int index;
        index = getIndex(item);
        return index;
    } catch (NullPointerException e) {
        throw new IllegalArgumentException("Tried to access a null object");
    }
}

public int getIndexDefaultNegative(int item) {
    try {
        int index;
        index = getIndex(item);
        return index;
    } catch (NullPointerException e) {
        return -1;
    }
}
```

Q2 Volskaya Industries

a. Baddice Einer wants to update his `SuperArray` data structure to add the `trim()` method, which makes null values at the end of the array disappear, similar to trimming an `ArrayList` down to size. For example, `{1, 2, null, null}` would trim to `{1, 2}`.

However, there's one issue. If the array is fragmented, which means there are null values in between non-null values rather than just at the end (e.g. `{1, null, 2, 3}`), trimming the array will not remove all the null values. Help him `throw` a `FragmentationException` with an error message when this happens. This exception should be handled by the user of `SuperArray` and should not directly cause the program to exit. (You may not need all lines.)

```
public class SuperArray {
    private Object[] arr;

    public SuperArray(int size) { arr = new Object[size]; }
    public int length() { return arr.length; }
    public Object get(int i) { return arr[i]; }
    public void set(Object o, int i) { arr[i] = o; }

    public class FragmentationException extends Exception {
        public FragmentationException (String msg) {
            super(msg);
        }
    }
    public void trim() throws FragmentationException {
        boolean fragmented = false;
        int trim_to = -1;
        // Finds if the array is fragmented. Assume this works properly.
        for (int i = arr.length - 1; i >= 0; i--) {
            if (arr[i] != null && trim_to == -1) trim_to = i;
            if (trim_to != -1 && arr[i] == null) fragmented = true;
        }
        if (fragmented) {
            String messageToPrint = "OMGZOR Fragmentation!!!!";
            throw new FragmentationException(messageToPrint);
        }
        arr = Arrays.copyOfRange(arr, 0, trim_to);
    }
}
```

TESTING

Q1 IntList Testing (Sp 15)

```
public class IntList {
    public int first;
    public IntList rest;

    public static IntList list(int… args) {...}

    @Override
    public boolean equals(Object o) {...}

    /** A method that destructively modifies the IntList to keep only
     * to keep only every other element. For example, (1,2,3,4,5) would
     * become (1,3,5)
     */
    public void skippify() {
        ...
    }
}
```

Write a JUnit test that verifies that skippify is working correctly on these two lists:

```
IntList A = IntList.list(1,2,3,4,5,6);
IntList B = IntList.list(3,4,3,6,3);
```

Solutions:

```
@Test
Public testSkippify() {
    IntList A = IntList.list(1,2,3,4,5,6);
    IntList B = IntList.list(3,4,3,6,3);
    IntList expectedA = IntList.list(1,3,5);
    IntList expectedB = IntList.list(3,3,3);
    A.skippify();
    B.skippify();
    assertEquals(expectedA, A);
    assertEquals(expectedB, B);
}
```

ASYMPTOTICS

Q1 Code-Analysis Runtime Q's (from Quiz 7)

Determine the tightest runtime bounds for the following functions on the input N.

```
void mystery1(int N) {
    for (int i = 1; i <= N * N; i *= 2) {
        for (int j = 0; j < i; j++) {
            System.out.println("moo");
        }
    }
}
```

Answer:

1.5 points: $\Theta(N^2)$

```
int mystery2(int N) {
    if (N == 0)
        return 0;
    return mystery2(N/3) + mystery2(N/3) + mystery2(N/3);
}
```

Answer:

1.5 points: $\Theta(N)$

Q2 Conceptual Asymptotics Q's

Order the following big-O runtimes from most to least efficient:

$O(\log n),\ O(1),\ O(n^n),\ O(n^3),\ O(n\log n),\ O(n),\ O(n!),\ O(2^n),\ O(n^2\log n)$

Solutions:

$O(1) \subset O(\log n) \subset O(n) \subset O(n\log n) \subset O(n^2\log n) \subset O(n^3) \subset O(2^n) \subset O(n!) \subset O(n^n)$

For each example below, there are two algorithms solving the same problem. Given the asymptotic runtimes for each, is one of the algorithms **guaranteed** to be faster? If so, which? And if neither is always faster, explain why. Assume the algorithms have very large input (so *N* is very large).

(a) Algorithm 1: $\Theta(N)$, Algorithm 2: $\Theta(N^2)$

(b) Algorithm 1: $\Omega(N)$, Algorithm 2: $\Omega(N^2)$

(c) Algorithm 1: $O(N)$, Algorithm 2: $O(N^2)$

(d) Algorithm 1: $\Theta(N^2)$, Algorithm 2: $O(\log N)$

(e) Algorithm 1: $O(N\log N)$, Algorithm 2: $\Omega(N\log N)$

(a) Algorithm 1: $\Theta(N)$ - Θ gives tightest bounds therefore the slowest algorithm 1 could run is relative to N while the fastest algorithm 2 could run is relative to N^2.

(b) Neither, $\Omega(N)$ means that algorithm 1's running time is lower bounded by N, but does not provide an upper bound. Hence the bound on algorithm 1 could not be tight and it could also be in $\Omega(N^2)$ or lower bounded by N^2.

(c) Neither, same reasoning for part (b) but now with upper bounds. $O(N^2)$ could also be in $O(1)$.

(d) Algorithm 2: $O(\log N)$ - Algorithm 2 cannot run SLOWER than $O(\log N)$ while Algorithm 1 is constrained on to run FASTEST and SLOWEST by $\Theta(N^2)$.

(e) Neither, Algorithm 1 CAN be faster, but it is not guaranteed - it is guaranteed to be "as fast as or faster" than Algorithm 2.

Guerilla Section 2 T/F

(a) True or false: if $f(N) \in O(N)$ and $g(N) \in O(N^2)$, and both functions are non-negative, then $|g(N) - f(N)| \in \Omega(N)$. if true, explain why; otherwise, give a counterexample.
False: Consider $f(N) = g(N) = N$

(b) True or false: if $f(N) \in \theta(N)$ and $g(N) \in \theta(N^2)$, and both functions are non-negative, then $|g(N) - f(N)| \in \Omega(N)$. If true, explain why; otherwise give a counterexample.
True: Since $g(N) \in \theta(N^2)$, it is also in $\Omega(N)$, while $f(N) \in O(N)$. Since g grows at least as N^2, and O(N) quantity is eventually negligible in comparison.

INTLISTS

Q1 hasCycle

Given an *encapsulated* IntList **a,** write a method that checks if it is circular.

```
/**
 *
 * @param a is an IntList
 * @return true iff a is circular, which means it has an IntListNode
 * whose next pointer points to a previous IntListNode
 */
public static boolean hasCycle(IntList a) {
    if (a == null || a.head == null  ) {
        return false;
    }
    IntListNode slow = a.head        ;
    IntListNode fast = a.head        ;
    while (fast != null && fast.next != null) {
        slow = slow.next             ;
        fast = fast.next.next        ;
        if (slow == fast             ) {
            return true;
        }
    }
    return false                     ;
}
```

Q2 Double in place

Without looking at your lab solutions, write a method that duplicates each node in this linked list destructively. This list is implemented as a **doubly linked list.** As an extra exercise, implement this for an encapsulated IntList.

```
/** DLList version */
public void doubleInPlace() {
    DLNode curr = sentinel.next;
    while (curr != sentinel) {
        DLNode dup = new DLNode(curr.item, curr, curr.next);
        dup.prev.next = dup;
        dup.next.prev = dup;
        curr = dup.next;
    }
}
```

```
/** Encapsulated IntList version */
public void doubleInPlace() {
    IntListNode curr = head;
    while (curr != null) {
        IntListNode dup = new IntListNode(curr.item, curr.next);
        curr.next = dup;
        curr = dup.next;
    }
}
```

Arrays

a) Consider the code below. Assume N is odd and positive.

```
public static int[][] genCoolGrid(int N) {
    int[][] grid = new int[N][N];// Everything defaults to 0 in int arrays
    for (int i = 0; i <= N / 2; i += 1) {
        for (int j = (N / 2) - i; j <= (N / 2) + i; j += 1) {
            grid[i][j] = 8;
            grid[N - 1 - i][j] = 8;
        }
    } return grid; /* sorry, low on space, bad style, but works! */
}
```

Show the return value of `genCoolGrid(5)` in the boxes below. Note: top left is `grid[0][0]`.

grid[0][0]	0	0	8	0	0	grid[0][4]
	0	8	8	8	0	
	8	8	8	8	8	
	0	8	8	8	0	
grid[4][0]	0	0	8	0	0	grid[4][4]

Solutions that left the 0's blank were given almost-full points.

JAVA GRAB-BAG

Q1. What would Java Print?

Code	Output

`public class Fastest {`
`  String map;`
`  public Fastest copy() {`
`  map = "Possible";`
`  return this;`
`  }`
 	
`  public static void main(String[] args) {`
`  Fastest map1 = new Fastest();`
`  map1.map = "BGH";`
`  Fastest map2 = map1.copy();`
`  System.out.println(map1.map);`
`  map2.map = "Lost Temple";`
`  System.out.println(map2.map);`
`  }`
`}`
	`Possible`
	`Lost Temple`

`public class Ketchup {`
`  public void friend(int ketchup) {`
`  int tomato = ((ketchup + 4) / 2);`
`  System.out.println(tomato);`
`  }`
 	
`  public static void main(String[] args) {`
`  Ketchup ash = new Ketchup();`
`  friend(4);`
`  }`
`}`
	`Compile-Time error: Cannot access a non-static method from a static context`

`public class Thyme {`
`  public static String zoo = "zoo";`
 	
`  public void gul(String dan) {`
`  dan = dan + this.zoo;`
`  zoo = dan;`
`  System.out.println(dan);`
`  }`
 	
`  public static void main(String[] args) {`
`  Thyme herb = new Thyme();`
`  herb.gul("hand");`
`  herb.gul("hand");`
`  }`
`}`
	`handzoo`
	`handhandzoo`

Login: _____

```
public class Gateway {
   int id;
   double location;
   Object[] productionQ = new Object[3];

   public void processQueue() {
      for (int i = 1; i <= productionQ.length; i++) {
         produce(productionQ[i]);
      }
   }

   public void produce(Object unit) {
      if (unit != null)
         System.out.println("Produced " +
               unit.toString());
   }

   public static void main(String[] args) {
      Gateway gw = new Gateway();
      gw.productionQ[0] = "zealot";
      gw.productionQ[1] = "stalker";
      gw.processQueue();
   }
}
```

```
Produced stalker
Runtime error:
ArrayIndexOutOfBoundsExceptio
n
```

Q2. Flibrocon

4. Flirbocon (12 points). Consider the declarations below. Assume that `Falcon` extends `Bird`.

```
Bird bird = new Falcon();
Falcon falcon = (Falcon) bird;
```

Consider the following possible features for the `Bird` and `Falcon` classes. Assume that all methods are **instance methods** (not static!).

F1. The `Bird::gulgate(Bird)` method exists.[1]
F2. The `Bird::gulgate(Falcon)` method exists.
F3. The `Falcon::gulgate(Bird)` method exists.
F4. The `Falcon::gulgate(Falcon)` method exists.

> The notation `Bird::gulgate(Bird)` specifies a method called `gulgate` with parameter of type `Bird` from the `Bird` class.

a) Suppose we make a call to `bird.gulgate(bird);`

Which features are sufficient **ALONE** for this call to compile? For example if feature F3 or feature F4 alone will allow this call to compile, fill in the F3 and F4 boxes.

☒ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method. For example, if having features F2 and F4 only (and not F1 or F3) would result in `Bird::gulgate(Bird)` being executed, check boxes F2 and F4 only.

☒ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

☒ F1 ☐ F2 ☒ F3 ☐ F4 ☐ Impossible

b) Suppose we make a call to `falcon.gulgate(falcon);`

Which features are sufficient **ALONE** for this call to compile?

☒ F1 ☒ F2 ☒ F3 ☒ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method.

☒ F1 ☐ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Bird::gulgate(Falcon)` method.

☐ F1 ☒ F2 ☐ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

☐ F1 ☐ F2 ☒ F3 ☐ F4 ☐ Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Falcon)` method.

☐ F1 ☐ F2 ☐ F3 ☒ F4 ☐ Impossible

---

[1] In other words, the `Bird` class has a method with the signature `gulgate(Bird)`

Q3. What Would Java Display?

```
1  System.out.println("Part a:");
2  // Part a:
3
4  Monster m = new Monster();
5  /* Static type of m: Monster
6     Dynamic type of m: Monster */
7  //Muahahahaha!!!
8
9  m.mash(m);
10 /* mash is a static method, look at m's static type to determine method called. */
11 //Monster:
12 //I go blargh
13 //I am 5 spooky
14
15 System.out.println("Part b:");
16 //Part b:
17
18 Monster g = new Ghoul();
19 /* Static type of g: Monster
20    Dynamic type of g: Ghoul */
21 /* Always an implicit call to super() in subclass's constructor. */
22 //Muahahaha
23 //I am a ghoul.
24
25 g.mash(g);
26 /* Again, mash is a static method. Look at g's static type.
27    Chooses line 36 from original code. */
28 //Monster:
29 /* On line 38 from original code, m's dynamic type is ghoul,
30    so will choose Ghoul's spook. */
31 //I'm so ghoul: blargh
32 //I am 5 spooky.
33
34 System.out.println("Part c:");
35 //Part c:
36
37 g.spookFactor = 10;
38 /* spookFactor is a declared as a static int.
39    Test yourself: if you change this to a nonstatic int,
40    how is the output of this program different?
41    Note: When looking for methods, java looks at the dynamic type.
42    When looking for fields, java looks at the static type. */
43
44 m.mash(m);
45 /* Again, looks at m's static type (Monster), and selects line 36
46    from the original code.
47    m's dynamic type is Monster so line 38 from the original code
48    will call Monster's spook. */
49 //Monster:
50 //I go blargh
51 //I am 10 spooky.
```

```
53 System.out.println("Part d:");
54 //Part d:
55
56 Ghoul ghastly = new Ghoul();
57 /* ghastly static: Ghoul,
58    ghastly dynamic: Ghoul */
59 //Muahahaha!!!
60 //I am a ghoul.
61
62 m = ghastly;
63 /* Now, m static: Monster,
64    m dynamic: Ghoul*/
65
66 ghastly = (Ghoul) m;
67 /* This does not change the static type of m! */
68
69 ghastly.haunt();
70 /* This is a dynamic method lookup, since haunt is not a static method.
71    Line 10 from the original code calls ghastly's static type's mash,
72    which is Ghoul's mash method, since mash is a static method. */
73 //ERRERERRRRERRR
74 //boogity boo:
75 //I'm so ghoul: blargh
76 //I am 10 spooky.
77
78 m.mash(ghastly);
79 /* m's static type is Monster, so it calls Monster's mash.
80    m's dynamic type is Ghoul so line 38 in the original code
81    calls Ghoul's spook.
82 */
83 //I'm so ghoul: blargh
84 //I am 10 spooky.
```