CS 367 Homework 1

Alex Pizzuto

---

1 Question One

We desire code that will remove every other element, beginning with the first, from a `List<String>` object. We have been given the following code:

```
for (int i = 1; i <= words.size(); i += 2) {
    words.remove(i);
}
```

A Suppose words initially contains:

```
[ "washington", "j. adams", "jefferson", "madison", "monroe", "j.q. adams",
    "jackson", "van buren" ]
```

Then after the code executes, we are left with

```
[ "washington", "jefferson", "madison", "j.q. adams", "jackson" ]
```

B This code does not work correctly. Java is zero-indexed, so the initialization condition should be `int i = 0` if we want to remove the first element in the list. Additionally, we should only be skipping one index at a time in the increment, so we should have `i +=1`, because once we remove an element, the elements higher in the list are shifted down an index. In other words, if at first our list is a list of ints, `myList = [1,2,3]`, then once we remove `myList[0]`, our new list is `myList = [2,3]`. Next we want to remove the element 3, which is now at index 1 instead of 0, so we need only increment by 1.

C Yes, the code is broken. Beyond just not functioning properly, as described in part (B), if a list with zero or one elements is passed to the code fragment, then we throw an `IndexOutOfBounds`. To fix the code, we should have:

```
for (int i = 0; i <= words.size(); i += 1) {
    words.remove(i);
}
```

2 Question Two

A Suppose we have called `ArrayList.add` 200 times on a single list, and have never removed any of the added elements. How many times has the array been resized? How many more elements can

we add (with no removals) without causing the array to resize again? If at first we have room to store 10 elements in the list, and each time we resize the list capacity triples, then after n resizings, we have allotted enough room to store 10×3^n elements. If we have called `ArrayList.add` 200 times on a single list, then we have resized 3 times ($10 \to 30 \to 90 \to 270$). Thus, we have room to store another 70 elements before resizing again.

B

$$i \times 3^d$$

C If we need a capacity of at least c elements, and our initial capacity is i elements, then we must triple until $c < i \times 3^d$. In other words, $\log_3(\frac{c}{i}) < d$. This is the same as saying we need $\log_3(\frac{c}{i})/1+1$ where division here is integer division, and the plus one is to ensure that our new capacity is larger than the needed capacity.

3 Question Three

A If `v1` is true and all other variables are false, then output would be

```
main enter
a enter
b enter
b exit
a exit
c enter
d enter
main caught Exc1
main exit
```

B If `v2` is true and all other variables are false, then output would be

```
main enter
a enter
b enter
Program terminated due to Exception Exc2
```

C If `v3` is true and all other variables are false, then output would be

```
main enter
a enter
b enter
b exit
a exit
c enter
d enter
c caught Exc3
main caught Exc4
main exit
```

D If `v4` is true and all other variables are false, then output would be

```
main enter
a enter
b enter
b exit
a exit
c enter
d enter
d exit
c caught Exc3
c exit
main caught Exc4
main exit
```

E If `v1` and `v4` are true and all other variables are false, then output would be

```
main enter
a enter
b enter
b exit
a exit
c enter
d enter
main caught Exc1
main exit
```

F If exception type `Exc2` were a checked exception, then you must include a throws clause in any method header where that Exception might be passed, in this case, in method a, b, and main.

G If exception type `Exc5` were a checked exception, then you must include a throws clause in any method header where that Exception might be passed, in this case, in method c, d, and main.