CS61B Week 5: Java Esoterica

List one or more reasons you might encounter each of the following exceptions. Also, specify whether each exception is checked or unchecked.

IOException Checked, means something went wrong with file input or output (FileNotFoundException is a subclass).

ArrayIndexOutOfBoundsException Unchecked, means you tried to access an array with either a negative index or an index greater than the length of the array.

NoClassDefFoundError Unchecked, probably means you are trying to run your program from the wrong directory (e.g. you are typing "java Main" from inside the enigma directory rather than "java enigma/Main" from outside it).

OutOfMemoryError Unchecked, means you have taken up too much space with your variables. This is almost always the result of an infinite loop.

NullPointerException Unchecked, means you tried to access a method or variable of a variable that is set to null. In other words, you tried to do some_variable "dot" something (some_variable.something), where some_variable is null.

StackOverflowError Unchecked, means you called too many functions that never resolved. This is almost always the result of infinite recursion (for example, if the function foo() has a call to foo() in its body and calls itself indefinitely).

IllegalArgumentException Unchecked, means you called a function with an argument that it cannot except for some reason. For example, you may have called a divide method with a 0 as the divisor, which should be illegal. Note that this does NOT mean you provided the wrong number or type of arguments to your function.

ClassCastException Unchecked, means you tried to cast something to the wrong type. For example, if you run "Object x = 5; String y = (String) x;" you will get this error. Note that the above code WILL compile. The compiler only looks at the static types and concludes you have some Object that may be able to be cast to a String, it does not know that the dynamic type of your object makes the cast invalid.

What do these compiler messages mean (besides that you did something wrong)?

```
MyClass.java:23: cannot find symbol
symbol  : method foo(java.lang.String,int)
location: class MyClass
    foo("blah", 8);
    ^
```

This means your program cannot find the method foo. You will want to make sure you've written the method, and it's actually in the Class you think it's in.

```
MyOtherClass.java:47: bar(int) in MyOtherClass cannot be applied to (java.lang.String,int)
    bar("cat", -100);
    ^
```

This means your program can find the method, but you're applying it with the wrong arguments. In this case, you're calling it with a String and an int while the method only accepts an int.

```
Note: MyThirdClass.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

This almost always means you're missing a generic type somewhere. In other words, you wrote something like "ArrayList<Integer> a = new ArrayList();" where you should have written "ArrayList<Integer> a = new ArrayList<Integer>();"

The United States Congress, in its infinite wisdom, passes a law that all nickels are worth 7 cents. Calculate the least number of coins you need to make change for exactly n cents. Assume you can only use combinations of pennies, dimes, quarters, and these new nickels (use brute force!).

```
public static int makeChange(int n) {
    int min = Integer.MAX_VALUE;
    for (int a = 0; a <= n; a++) {
        for (int b = 0; b <= n/7; b++) {
            for (int c = 0; c <= n/10; c++) {
                for (int d = 0; d <= n/25; d++) {
                    if (a + 7 * b + 10 * c + 25 * d == n) {
                        min = Math.min(a + b + c + d, min);
                    }
                }
            }
        }
    }
    return min;
}
```

Sample Midterm Question of the Week:

```
/** Slice the list L into a list of N lists such that list #k contains
 *  all the items in L that are equal to k modulo N, in their original
 *  order. For example, if N is 3 and L contains [9, 2, 7, 12, 8, 1, 6],
 *  then the result is [ [9, 12, 6], [7, 1], [2, 8] ]. The operation
 *  is destructive (it may destroy the original list) and creates no new
 *  IntList objects (it will, of course, create new IntList2 objects).
 */
static IntList2 dslice (IntList L, int n) {
    IntList[] slices = new IntList[N]; // NOTE: Creates no IntLists!
    while (L != null) {
        IntList next = L.tail;
        int k = L.head % N;
        L.tail = slices[k];
        slices[k] = L;
        L = next;
    }
    IntList2 result;
    result = null;
    for (int k = N-1; k >= 0; k -= 1)
        result = new IntList2 (dreverse (slices[k], null), result);
    return result;
}
static IntList dreverse (IntList L, IntList rest) {
    if (L == null)
        return rest;
    IntList next = L.tail;
    L.tail = rest;
    return dreverse (next, L);
}
```

Sample Interview Question of the Week:
The makeChange function we wrote above starts to get really slow as we try larger numbers (even just $20, or 2000 coins, takes a few seconds to calculate). Write a more efficient implementation. Your program, running on an instructional machine, should easily be able to make change for numbers of coins in the $10,000 range.
This is a classic example of dynamic programming, a method for breaking a large problem into many smaller subproblems that are easier to solve. The basic idea is to build an array with n elements where the ith index of the array is the least number of coins it takes to make i cents. We start our array from one cent and build towards n. We can see that the number of coins it takes to make i cents using a v-cent coin is the minimum number of coins it takes to make i-v cents, plus one for the coin we just used to make i cents. Therefore, to fill the ith value of our array we iterate through our avaiable coins with values $v_1, v_2, ..., v_k$ and choose the one where the (i-v_x)th element of our array is least. If we have k coins and we're trying to make n cents, this algorithm will take around n * k operations, which is way better than what we were doing before.

CODE:

```
public static int dynamicMakeChange(int n) {
    int[] coins = {1, 7, 10, 25};
    int[] min = new int[n+1];
    for (int i = 1; i <= n; i++) {
        int localMin = Integer.MAX_VALUE;
        for (int k = 0; k < coins.length; k++) {
            if (i - coins[k] >= 0 && min[i - coins[k]] + 1 < localMin) {
                localMin = min[i - coins[k]] + 1;
            }
        }
        min[i] = localMin;
    }
    return min[n];
}
```