Object-oriented Programming in Java
M Tech Preparatory Term, 2023
Week 1

This set of programming problems are intended to help you exercise basic features of Java including designing classes that use other classes.

1 Garbage Collection

Run the following piece of code:

```
public class Test1 {
    public static void main () {
        int N = 10;
        int M = 100000;
        for(int i =0; i< N; i++) {
            int[] box = new int[M];
        }
    }
}
```

Are there large values of N (within the range of “int”) for which this program does not work?

If you modify the program as follows:

```
public class Test1 {
    public static void main () {

        int N = 10;
        int M = 100000;
        int[][] boxes = new int[N][];
        for(int i =0; i< N; i++) {
            int[] box = new int[M];
            boxes[i] = box;
        }
    }
}
```

How large can you make N and still be able to run on your machine? Why is this different from the first version?
Can you re-configure Java on your machine to allow larger amounts of memory (heap)?

2 Matching Parenthesis

Write a Java program to read in a file containing a sequence of parentheses of various kinds - (){}[] - in any order and combination, and check if the parentheses are “balanced”.
In the case of failure, the program should specify the position in the input stream where the first mismatch is detected (the first input character is at location 0)

For example, for the input,
([]{()})
the output should be **success**
and for the input
([]{())}
the output should be **failed at 6**

Implement this without using explicit arrays/lists/stacks or other heap memory.
You could first implement a version that takes a string (hardwired in **main**) as the input, and then modify it to read from a file.

3 Unique IDs

A website assigns a unique id to each registered user. The ID is of the form yyyy-xxxxxx, where yyyy is the year and xxxxxx is a unique integer (always shows as 6 digits, with leading 0’s if needed) that are sequential for each year. That is, the first user for a given year ‘abcd’ is assigned the ID abcd-000001, the next abcd-000002 and so on. We can assume that the 6 digits are sufficient to accommodate all visitors for a given year. The integer part of the id gets reset to 1 the next year.

Write a Java class IdGenerator that encapsulates the above logic. This should have a method getNextId() which should return the next unique id, taking into account the year. Write a main that invokes this repeatedly. To test the program, the main class can have a method getYear() which returns a random year each time. You can try out the Java LocalDate class to get the actual date and year.

Assume this program runs for years without having to restart. All intermediate data should be in memory - do not use files or databases to store any data.

4 A homegrown Stack

As we will see later, Java provides a number of standard data structures. However, as a practice, we will implement our own Stack object.

Implement a Stack class that can handle String data. The class should provide the basic interfaces for any stack - push, pop, and isEmpty.

To implement this, first implement a LinkedList class, which also manages String data in its nodes. The LinkedList class should provide methods to add and remove Strings from the head of the list, check if the list is empty, etc.

Now, implement the Stack class using this implementation of LinkedList.

Test this Stack by using it to reverse the order of words in a sentence.