Supervision 1 – Object-Oriented Programming

Adapted from Dr Robert Harle's exercises by Dr Stephen Cummins

Types, objects classes & Pointers, references and memory

1. Give three differences between typical functional and typical imperative programming languages.

2. Provide an example, in code, of the following java constructs:
    a. Primitives,
    b. References
    c. Classes
    d. Objects

3. Draw a simple diagram to illustrate what happens with each step of the following java code in memory.

```
Person p = null;
Person p2 = new Person();
p=p2;
p2 = new Person();
p=null;
```

4. Demonstrate the use of all of Java's control flow statements (as given in the lectures) by writing the following functions:
    - int sum(int[] a), which produces the sum of the elements in a.
    - int[] cumsum(int[] a), which produces the cumulative sum of a e.g. [1,2,3] becomes [1,3,6].
    - int[] positives(int[] a) which produces the array of positives in a (note your returned array should not have any empty slots).

5. Write a java function that creates an array-of-arrays to represent an nxn matrix of floats.

6. The following code is a failed attempt to write a function that can be used to add increments to a 2D vector. Explain why it doesn't work and adapt the code to work (without using a custom class for those that know how to do this).

```
public static void vectorAdd(int x,int y,int dx, int dy) {
      x=x+dx;
      y=y+dy;
}
public static void main(String[] args) {
      int a=0;
      int b=2;
      vectorAdd(a,b,1,1);
      System.out.println(a+'' ''+b);
      // (a,b) is still (0,2)
}
```

7. Design and implement (from scratch – i.e. not using java lists) a singly linked list. Your class(es) should support the addition and removal of elements, querying of the head element, obtaining the nth element and computing the length of the list. Your list only needs to work with integers. Do not use Generics.

8. Write a method to detect cycles in your linked list.

9. Describe the stack data structure.

10. Implement a stack data structure, from scratch, in Java. Design the interface to your stack to be complete, consistent, and easy to use. Be prepared to justify your implementation in the supervision session.

Please submit your work by e-mail (as one pdf document that includes copies of your code). Anything that you write should be runnable and checked into a git repository (somewhere that I can access).