Supervision Work 1

Github Repo: https://github.com/JamieLittle16/OOP-supervision-1

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

- In functional programming all variables are immutable while imperative languages generally allow mutable variables
- Functional languages tend to use recursion for iterations while imperative languages generally use for and while loops.
- Functions never modify state and only return values in functional languages. Imperative languages can modify state as well as returning values.

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

a) Primitives are the most fundamental types. A primitive variable value is stored directly in stack memory.

```
int num = 10;
```

The variable `num` holds the value 10.

b) References refer to a variable in memory that does not contain the data itself but rather a memory address of the data itself.

```
String test_str = "Hello World";
```

The variable `test_str` is stored on the stack and contains a memory address (a reference) to an area of heap memory containing "Hello World".

c) A class outlines the structure for an object (it does not actually occupy memory yet - unless it is static), it just describes what the object will look like.

```
class Car {
   int miles;
   void drive() {
      System.out.println("Driving");
   }
}
```

It describes what pieces of state and function an object will have. An object of type Car will have a number of miles and the ability to drive.

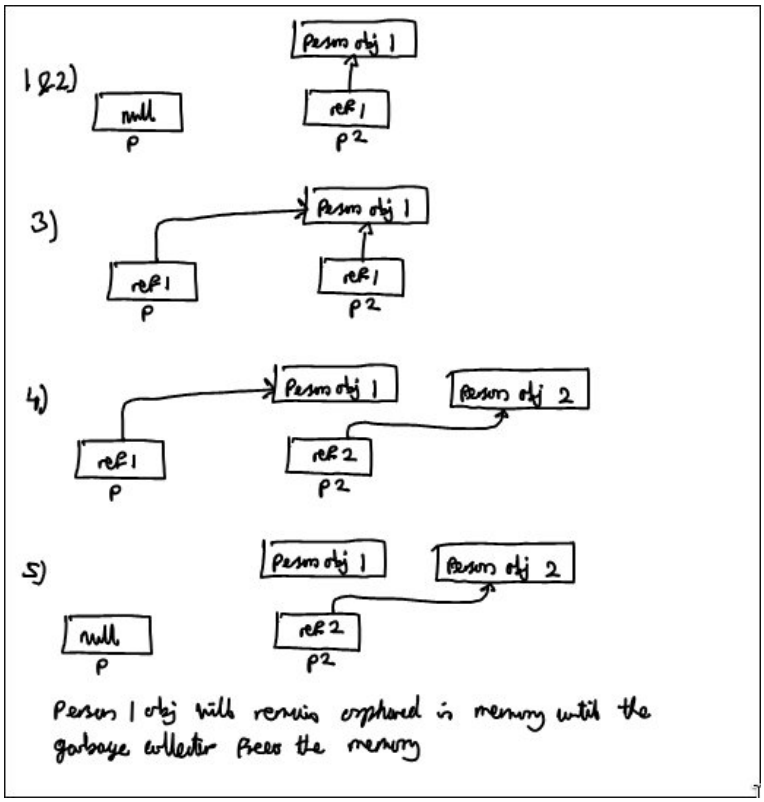
d) An object is an instance of a class that actually exists in memory. It bundles together variables and functions.

```
Car newcar = new Car();
```

Separate instances of a class can be modified independently from each other (unless they have static attributes).

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

```
1|  Person p = null;
2|  Person p2 = new Person();
3|  p = p2;
4|  p2 = new Person();
5|  p = null;
```



Note: Person 1 obj will remain orphaned in memory until the garbage collector frees the memory.

4. Demonstrate the use of all of Java's different control flow statements 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).

Refer to Q4 in the github repo.

5. 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)
}
```

In Java function parameters are passed by value. This means when performing `vectorAdd`, `dx` and `dy` are added to copies of `x` and `y`, while the original values remain unchanged.

Refer to Q5 in the github repo.

6. 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

Refer to Q6 in the github repo.

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

See Q7 in github repo.

8. Describe the stack data structure.

A stack data structure is LIFO (Last In First Out).

- The last element added to the stack is the first element to be taken out.
- Conversely the first element added will be the last element to be taken out.

Unlike an array, a stack has a single access point. Only the element at the top of the stack can be accessed. To get the third element the first and second must first be removed.

9. Implement a stack data structure 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.

See Q9 in the github repo.