1. Write a java function that creates an array-of-arrays to represent an n×n identity matrix of floats.

2. Write another Java functin that transposes an n×n matrix represented using an array-of-arrays. Your function should be in-place, i.e. use only O(1) space.

   Implemented in "nnarray.java". The class is pasted below.

```
public class nnarray {

    private final float[][] matrix;
    private final int n;

    nnarray(int n){
        this.n = n;
        matrix = new float[n][n];
        for (int i = 0; i < n; i++){
            matrix[i][i] = 1;
        }
    }

    void transpose(){
        float t;
        for (int i = 0; i < n; i++){
            for (int j = i; j < n; j++){
                t = matrix[i][j];
                matrix[i][j] = matrix[j][i];
                matrix[j][i] = t;
            }
        }
    }
}
```

3. **4.2**

   Draw some simple diagrams to illustrate what happens with each step of the following Java code in memory.

Before Line 1:

location	value
0	
1	
2	
3	
4	

p is assigned. However, since the value of p is null, it does not point to anything (it has a null pointer).

After Line 1:

location	value
0	p
1	
2	
3	
4	

p2 is assigned to a new person. So p2 references the memory address of this new person.

After Line 2:

location	value
0	p
1	p2
2	Person instance
3	
4	

p is then set to reference whatever p2 is referencing. So p references the memory address of the person (2).

After Line 3:

location	value
0	p
1	p2
2	Person instance
3	
4	

p2 is then reassigned to a new person. So the p2 now references a new person. However: p is unnaffected.

After Line 4:

location	value
0	p
1	p2
2	Person instance
3	Person instance
4	

p is set to reference null. So p's pointer is removed. Since there is nothing referencing the old person (2), it is marked as unused and will (hopefully) be garbage-collected.

After Line 5:

location	value
0	p
1	p2
2	Person instance
3	Person instance
4	

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

3.3

(a) Implemented in "Vector2D.java". The class is pasted below:

```
public class Vector2D {

    private float[] xy;

    Vector2D(){
        xy = new float[] {0, 0};
    }

    Vector2D(float[] vect){
        set(vect);
    }

    Vector2D(float x, float y){
```

```
        set(x, y);
    }

    void set(float[] vect){
        if (vect.length == 2){
            xy = new float[] {vect[0], vect[1]};
        }
        else{
            throw new IllegalArgumentException();
        }
    }

    void set(float x, float y){
        xy = new float[] {x, y};
    }

    float getx(){return xy[0];}

    float gety(){return xy[1];}

    float[] getxy(){return new float[] {xy[0], xy[1]};}

    void add(Vector2D v){
        xy[0] += v.getx();
        xy[1] += v.gety();
    }

    void subtract(Vector2D v){
        xy[0] -= v.getx();
        xy[1] -= v.gety();
    }

    float dot(Vector2D v){
        return xy[0] * v.getx() + xy[1] * v.gety();
    }

    float magnitude(){
        return (float) Math.sqrt(Math.pow(xy[0], 2) + Math.pow(xy[1], 2));
    }

    void normalise(){
        float magnitude = magnitude();
        xy[0] /= magnitude;
        xy[1] /= magnitude;
    }

    void scale(float n){
        xy[0] *= n;
        xy[1] *= n;
    }
}
```

(b) What changes would be needed to make it immutable?

To make it immutable you'd have to remove all the "set" methods, and change the add, subtract, scale and normalise methods to return the result rather than set the current vector to be the result.

(c) Contrast the following prototypes for the addition method for both the (i) mutable and (ii) immutable versions.

- `public void add(Vector2D v)`
- `public Vector2D add(Vector2D v)`
- `public Vector2D add(Vector2D v1, Vector2D v2)`
- `public static Vector2D add(Vector2D v1, Vector2D)`

(i) mutable:

The first signature is suited to a mutable Vector2D. It would suggest that the second vector is being added to the second and the result is being stored in the vector.

The second signature is less suitable for a mutable vector. The best mutable interpretation of this would be to add the second vector to the first and then return a new vector with the value of the old vector (so that the previous value is not lost). In practical terms there are few cases where you would want the old vector (and so this interpretation would be niche) – conversely it could be argued that this implementation would be more flexible than the first implementation.

The third signature is not suitable to a mutable vector. This implies adding two vectors together and then returning the sum – which is not a helpful implementation for an instance of vector.

The fourth signature is generally unsuitable to a mutable vector. However, it is static so not specific to a vector and since it takes different arguments to more conventional add methods, it could overload add – such an implementation could be useful for a mutable vector.

(ii) immutable:

The first signature is totally unsuitable to an immutable vector. Since there is no return type and the vector is immutable; this function can neither change variables or return variables and so can do nothing.

The second signature is suitable for an immutable vector. It allows the second vector to be added to the first and the result to be returned.

The third signature is not suited to an immutable vector. Adding two vectors together in an instance of a vector doesn't make sense.

The fourth signature is suited to an immutable vector. This is a static method and so specific to the class rather than an object. This means you would add together two vectors in the class rather than add one vector to another.

(d) How can you convey to a user of your class that it is immutable?

- You can make every function return a value – to make it clear that none of them make any persistent changes.
- You could include immutable in the name of the class.
- Don't implement any methods which are ambigious – ie which could be percieved to be mutable (ie "update").
- You can include comments and documentation saying it is immutable.

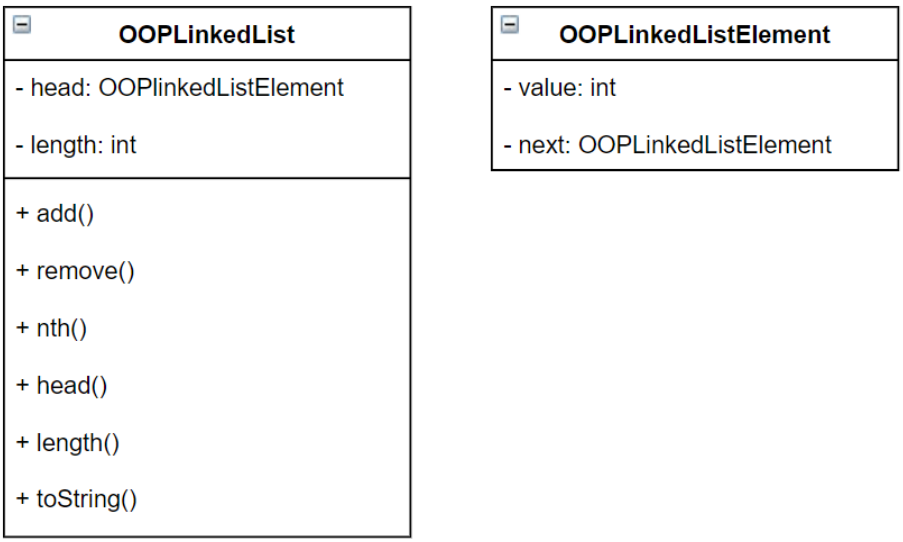
3.4

(a) Implemented in "OOPLinkedList.java". The class is pasted below.

```
package uk.ac.cam.hjel2.OOPsv1.LinkedList;
public class OOPLinkedList {

    OOPLinkedListElement head;
    int length;

    static class OOPLinkedListElement{

        final int value;
        OOPLinkedListElement next;

        OOPLinkedListElement(int v, OOPLinkedListElement pointer){
            value = v;
            next = pointer;
        }
    }

    public void add(int n){
        head = new OOPLinkedListElement(n, head);
        length += 1;
    }

    public void remove(){
        if (length != 0) {
            head = head.next;
            length -= 1;
        }
    }

    public int nth(int n){
        if (n >= length){
            throw new IndexOutOfBoundsException();
        }
        OOPLinkedListElement ith = head;
        for (int i = 0; i < n; i++){
            ith = head.next;
        }
        return ith.value;
    }

    public int head(){
        if (head != null){
            return head.value;
        }
        throw new IndexOutOfBoundsException("OOPLinkedList is empty");
    }

    public int length(){return length;}

    @Override
    public String toString(){
        OOPLinkedListElement current = head;
        if (current == null){return "[]";}
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append("[");
        while (current.next != null){
```

```
                stringBuilder.append(current.value);
                stringBuilder.append(", ");
                current = current.next;
            }
            stringBuilder.append(current.value);
            stringBuilder.append("]");
            return stringBuilder.toString();
        }
    }
```

(b)



5.5

Implemented in "OOPSortedLinkedList.java". The code is pasted below.

```
package uk.ac.cam.hjel2.OOPsv1.LinkedList;

public class OOPSortedLinkedList extends OOPLinkedList{

    @Override
    public void add(int n){
        OOPLinkedListElement current = head;
        if (head == null || n < head.value){
            head = new OOPLinkedListElement(n, head);
        }
        else{
            insert(n, current, current.next);
        }
    }

    private void insert(int n, OOPLinkedListElement previous, OOPLinkedListElement current){
        if (current == null){
            previous.next = new OOPLinkedListElement(n, null);
            length++;
        }
```

```
            else if (n < current.value){
                previous.next = new OOPLinkedListElement(n, current);
                length++;
            }
            else{
                insert(n, current, current.next);
            }
        }
    }
```