Q1.3

Primitives: d, f

References: i, l, k, c Classes: Double, Tree, Computer, LinkedList

Objects: k, i, l

Q1.9

So we have a class called Test. The class has a public attribute x that is initialised when the class is instantiated. The class also has a public method called Test (it is NOT a constructor). This method is never called in main(). Therefore, x is never changed to 7 and remains 0 instead.

Q2.1

Using private state is always a good idea because then there is no chance that you, other people or other classes can accidentally change an attribute that you didn't mean to change. If you do need to change or read the attribute, then you will need to deliberately use the public getters and setters which requires you to know exactly what you are doing. The principle is called encapsulation.

Q2.2

Disadvantages: Python programmers get used to this and they do follow the convention because it's become a habit but it could still cause problems when other classes accidentally get access to the public attributes and change what is not meant to be changed (i.e. external code modifies internal attributes).Therefore, it is easier for a python object to end up in an inconsistent state. Also, java catches the illegal access at compile time while python will not warn you unless something crashes during run time. Therefore, there is more responsibility on the developer.

Advantages: no need to write getter and setter methods which makes the code shorter, cleaner, more comprehensible and less repetitive.

Q2.3

1. (a)

```
public class Vector 2{
  private float x, y;

  public Vector(_x, _y){
    x = _x;
    y = _y;

  }
  public Vector2 add(Vector2 other){
    return Vector2(x + other.x, y + other.y);
  }
  public Vector2 subtract(Vector2 other){
    return Vector2(x - other.x, y - other.y)
  }
  public float dot(Vector2 other){
    return x * other.x + y * other.y;
  }
  public Vector2 scale(float a){
    return Vector2(a*x, a*y);
  }
  public float magnitude(){
    return Math.sqrt(Math.pow(x,2) + Math.pow(y,2));
  }
  public Vector2 normalised(){
    float magnitude = this.magnitude();
    return Vector2(x/magnitude(), y/magnitude());
  }
}
```

(b) Immutable when the state of a class cannot be changed in any way. So the attributes would have to be private and final (final means they can be assigned only once). There should be no setter methods. All fields would need to be set in a constructor only.

(c) i. This is a mutable method. It returns void which means it modifies the internal state of the current vactor inside the procedure.

ii. This is an immutable method since it returns a new vector and does not modify the current vector in any way.

iii. This method is a bit odd, it does not use the current vector whatsoever...

iv. This method is immutable. It does not depend on an instance of Vector2 and returns a new vector.

(d) Make code changes as described in part (b) and we could also use comments.

Q2.4

NOT FINISHED

```
public class OOPLinkedList {

private OOPLinkedListElement head;


static class OOPLinkedElement {

    int data;
    OOPLinkedListElement next;

    // Constructor
    OOPLinkedListElement(int d)
    {
        data = d;
        next = null;
    }
}

// Method to insert a new node
public static LinkedList insert(OOPLinkedList list, int data)
{
    // Create a new node with given data
    OOPLinkedListElement new_e = new OOPLinkedListElement(data);
```

```
        // If the Linked List is empty,
        // then make the new node as head
        if (list.head == null) {
            list.head = new_e;
        }
        else {
            for each element, shuffle them up by one element.


        }

        // Return the list by head
        return list;
    }


    }
```

Q2.5

The class violates SRP because it does not have a single purpose i.e. there is a method for fetching, parsing and printing data. This could be split into 3 separate classes each of which is assigned one of the responsibilities. And then the 4th class would combine them all by taking the other 3 classes as constructor parameters.

Q2.6

1.
```
public class StudentRecord{
  private final String name, college;
  private final int age;

  public StudentRecord(String _name, int _age, String _college){
    name = _name;
    age = _age;
    college = _college;
  }
  public void print(){
    System.out.println(name);
    System.out.println(age);
    System.out.println(college);
  }
}
```

2.
```
public record StudentRecord(String name, int age, String college){
  public void print(){
    System.out.println(name);
    System.out.println(age);
    System.out.println(college);
  }
}
```

3. I am not sure..

Q3.1

References cannot do arbitrary memory access unlike pointers. This makes the chance of crashing a program smaller. We cannot make a reference point to an unallocated memory address or a memory address that is in another process. So the reference is always either a valid object or null and nothing else. A reference is verified at run-time, is it a valid memory, does the object exist, is the type correct? This prevenets silent memory corruption and moving on into invalid state. Pointers do not have a run-time protection which makes them dangerous.

Q3.2

Every time I try to upload an image, it gets corrupted for some reason so would it be okay, if I explained it in words?

So Person p is a local variable, no person object is created, and p holds the null value. It is not a reference and it does not point to anywhere in memory.

p2 is a reference and points to a newly created object which is allocated some memory location. p = p2 means p becomes a reference and points to the same memory location as p2 does. When p2 is assigned a new object, it points to a different object now while p still points to the old person. p = null means that p doesn't point anywhere now, the old person can be considered for garbage collection because nothing points to it, and p2 still points to the second person.

Q3.3

The first add method has 3 parameters: an array xy, integer dx and integer dy. The first element (i.e. element 0) of the array is added to the integer dx and the second element (i.e. element 1) of the array is added to the integer dy.

The second add method takes 4 parameters: integer x, integer y, integer dx and integer dy. dx is then added to x and dy is added to y.

In main we have an array xypair, the 0th and 1st element of which is 1. So the first and second arguments of the add method are both 1. So we pass the integers that are in the array which then become local copies inside the method. This means the array itself is not modified. So the first print line would output 1 1. While the second print line would output 2 2 because the array itself is modified.

Q3.4

Would the code from Q3.3 be a good example?

Q4.1

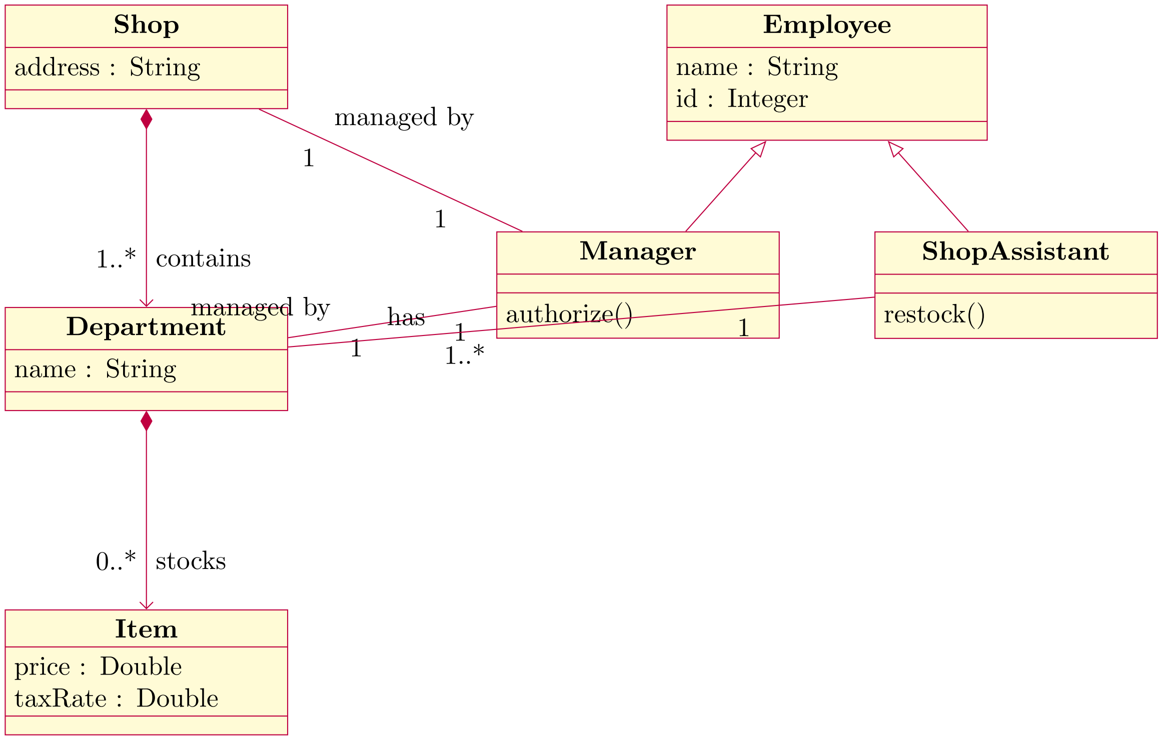
Inheritance represents an ”is-a” relationship. 3D vector cannot inherit from 2D because they behave differently. 3D vector needs a third component z for every method defined in 2D.

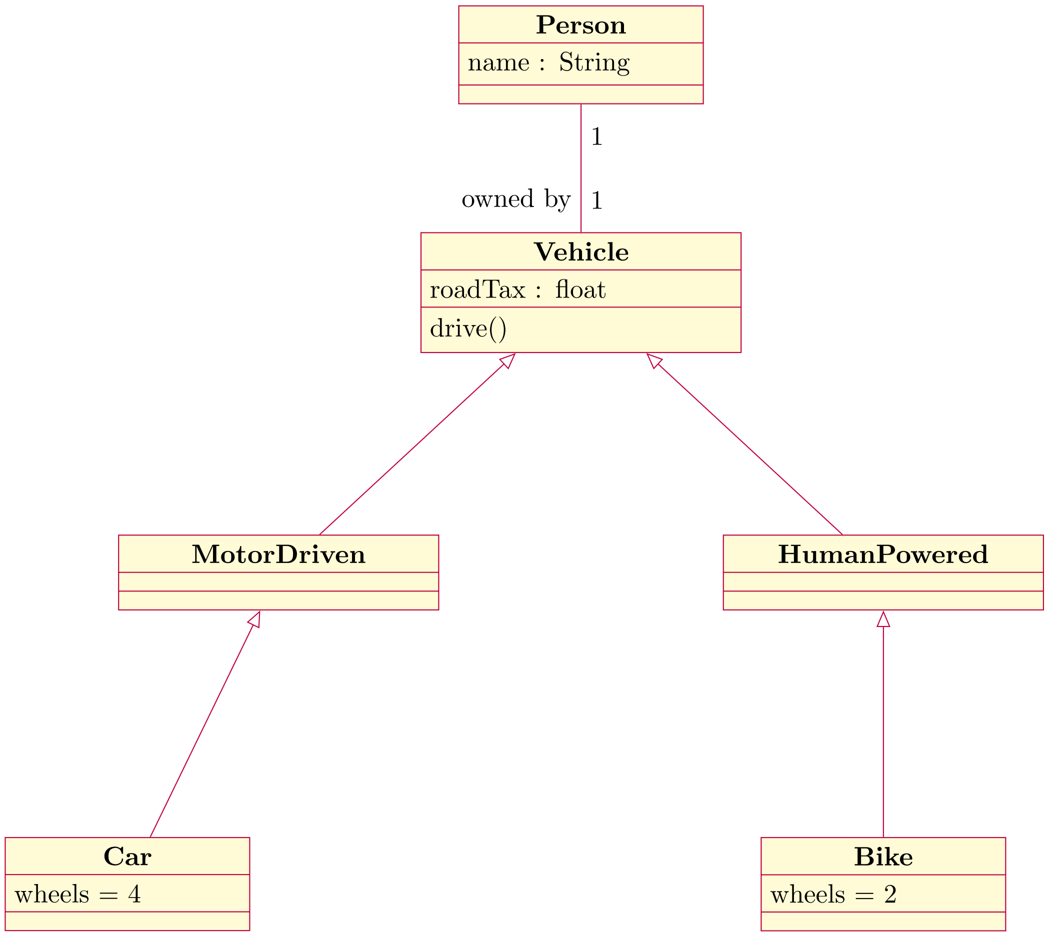
Q4.2

When there is no access modifier then the field can be accessed by the class and the package but not by everyone else or any subclasses.

```
public class Person{
int age;
public Person(age){this.age = age} //this will compile }
public class Student extends Person{
public Student (age){ this.age = age}
//this will compile only if class Student is in the same package as class Person, otherwise it will
}
```

Q4.4





I am aware that there are 3 more questions left, I ran out of time to do them, but I will attempt them again during holidays.