Code is available here

1 Intro to Java

1.2 Functional languages are declaritive, so you specify what to do, but not necessarily how to do it. At their core they rely on functions without side effects, whereas imperative languages rely on procedures which execute side effects, where the implementation is specified.

1.3 `double`, `int`, `float` and `null` are primitives.
`LinkedList`, `Double`, `Tree` and `Computer` are classes
`k` and `l` are objects.

1.4 Consistent naming standards encourage readable and well formatted code, which makes collaboration with others easier, and also means when using other peoples libraries or sharing your own code as libraries, everything is named as you would expect.

1.5 (b) Since the argument have the same type, there is no way for the compiler to infer which version of the function to use.

1.9 The constructor returns void, so the compiler treats it as a method and not a constructor. The default constructor is called, which does nothing and hence x is never changed.

2 Class Design and Encapsulation

2.1 Using private ensures that you can fully control how an attribute is modified. Other classes cannot directly modify the values of attributes, and you can control with setters and getters how the values are accessed or read.

2.2 In very specific use cases, it may be useful to modify the values of private attributes. However relying on convention means that there actually isn't anything directly stopping you from ignoring the understore, and modifying the attribute. This means classes will be unsafe.

2.3 (b) No setters, attributes declared with `final`

(c) The first method could be used for the mutable version, as it could add the vector directly to the vector which calls it.
The second method could be used by either mutable or immutable version, as it returns the sum of the vectors without modifying the objects.
Similarly the third function can work for either, but is poorly designed as it does not use the object calling it at all.
Unsure on the 4th prototype.

(d) Make the class final.

2.4 (b)

OOPLinkedList
head : OOPLinkedListElement
+ getHead() : OOPLinkedListElement + valAt(i : int) : OOPLinkedListElement + removeHead() : void + add(val : int) : void + length() : int

OOPLinkedListElement
val : int next: OOPLinkedListElement
+ getVal() : int + getNext() : OOPLinkedListElement

2.5 SRP states that each class should only have one function, while Report has three. It should be broken up into three seperate classes.

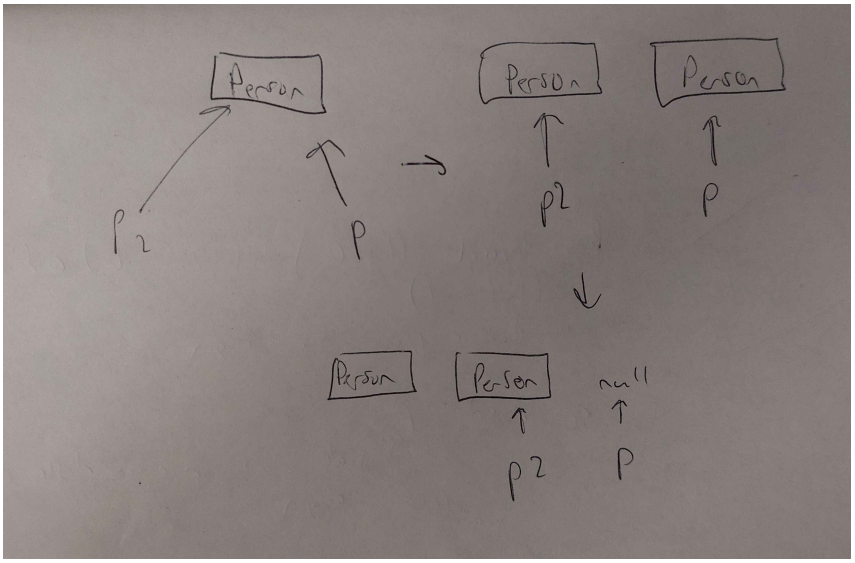
2.6 (c) Having a defined `equals()` function means that comparisons can be made on the class, expanding the functionality of the record. Similarly, having a `hashCode()` function means that the record can be used as a key in a HashMap or HashSet, so it does not limit the functionality of the record and means it can be used in cases like these without risking an error.

2.7

3 Pointers, References and Memory

3.1 References are more controlled than pointers and are more restrictive over what you can access. Pointers give you unrestrained access to the data at that memory address, whereas the compiler ensures that references can only let you access the relevant data — references cannot be arbitrarily assigned.

3.2



3.3 It will print `"1 1"` and then `"2 2"`. This is because the first call of add matches the signature of the function with 4 integer arguments. This then does not change anything

as x and y in the function are contained to the scope of the function. When add is called again, it is called with the other signature, and as arrays are stored in variables as a reference, the reference to the array is passed into the function so modifying xy changes xy outside of the function.

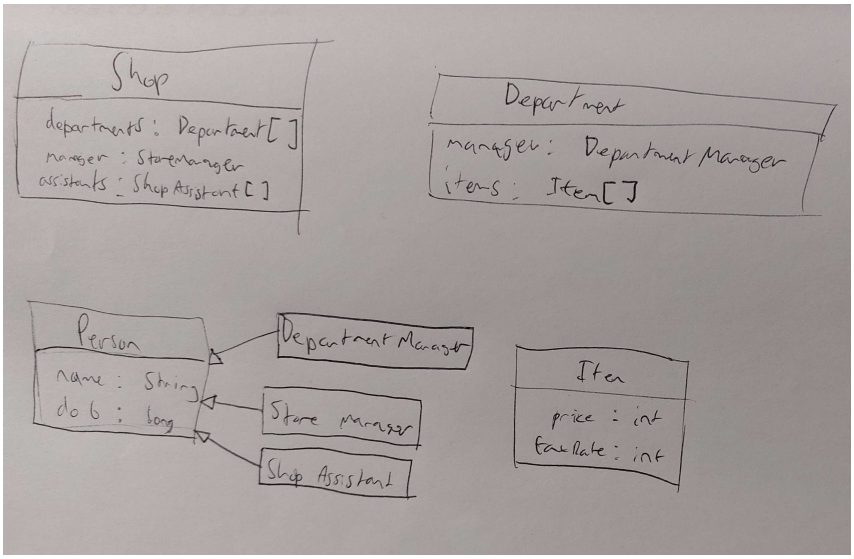
3.5 Only using pass by reference means lots of memory will be saved, as if any two variables store the same data, they can just point to the same data instead. Less copying can increase performance when copying large data structures. Functions can also directly modify the data, without having to reassign to the variable. However, not being able to pass by value can be wasteful, as references would be needed to point to very small simple primitives, wasting space, when it could simply just store the value directly. Also, functions may modify variables directly even when it may not be desirable.

4 Inheritance

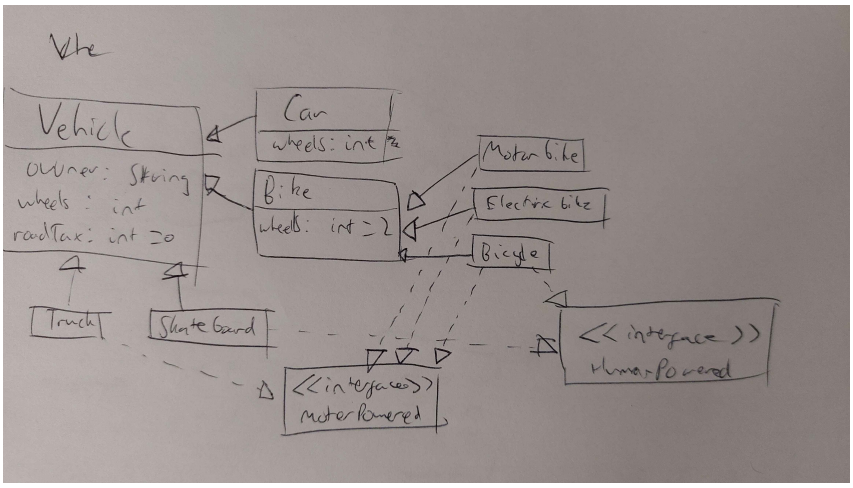
4.1 For a Vector3D, all behaviour will need to be overridden for the class to be fully functional (as there are 3 variables not 2). This is an inefficient use of inheritance, as it should only really be used for when additional behaviour needs to be added to existing behaviour.

4.2 The default access modifier is private, as attributes cannot be modified after the constructor is called.

4.4 (a)



(b)



4.5 (a) If the modifier was public, protected or unspecified the code would compile fine, but if the modifier was private compliation would fail.

(b) If the modifier was public or protected the code would compile fine, otherwise if unspecified or private it would fail.

(c) If the modifier was public or protected the code would compile fine, otherwise if unspecified or private it would fail.

(d) Compilation would fail unless the modifier was public, as only then can the attributes be accessed.

4.6 This will print `"a.value = Super"` and `"Sub.printValue: sub"`. `a.value` stores the value from Super, which it has been generalised to in the initialisation. `printValue` has a different value because the compiler recognises that a is actually a Sub in memory, and calls the correct function accordingly.