OOP supervision 2

See the Github repo at https://github.com/MarcellPerger1/CST-1A-OOP-Supervision2 for the full code.

Section 4: Inheritance

4.1

Inheritance should only be used when the subclass has the same methods and attributes as the base class with some extra (i.e. the subclass extends the functionality). The classes should also follow the Liskov substitution principle: anywhere where the superclass can be used, the subclass should also be allowed. However, a `Vector3` cannot be used everywhere where a `Vector2` can be used (e.g. you can add a `Vector2` to a given `Vector2` but you can add a `Vector3` to that `Vector2`). The misunderstanding is that it's not enough for the data to be the same with some stuff added, the methods/behaviour also has to be the same but with some stuff added.

4.2

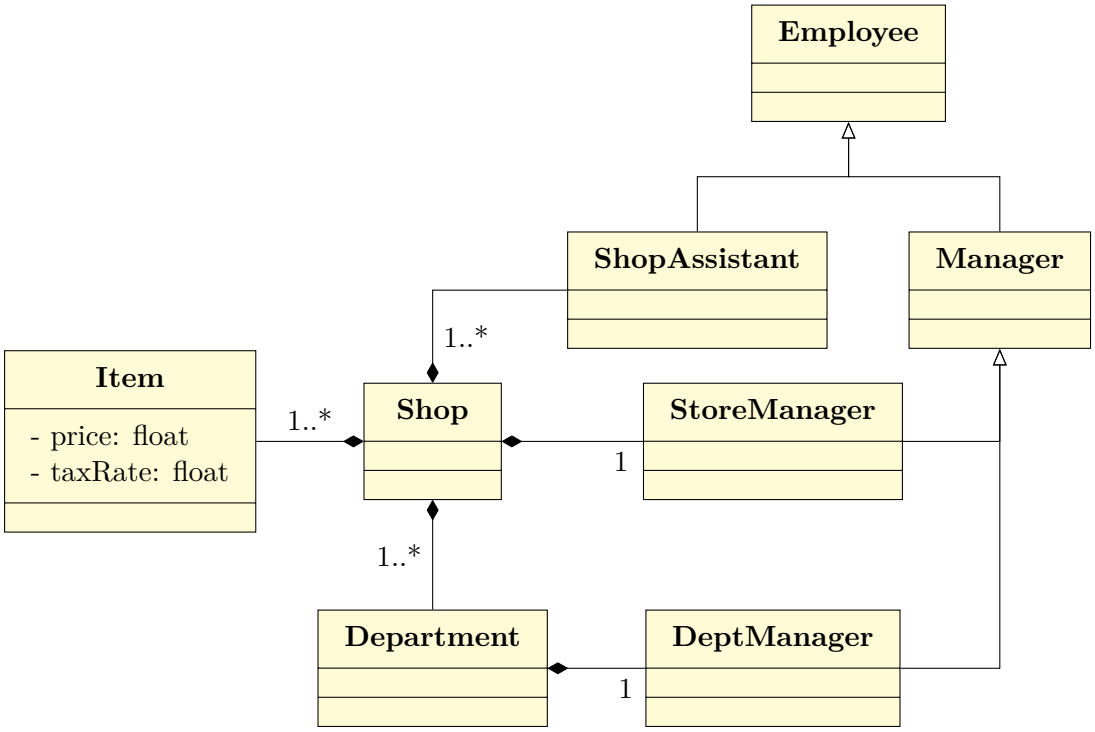
If you don't specify and access modifier, the default is package-private - it can be accessed by anything in its class and anything in the same package as it. See `Q4P2/**` for demo (commented-out code will make the compilation fail if it is uncommented)

4.3

See `Q4P3/NoConstructor/*.java` for classes without any constructor was added.
See `Q4P3/UnaryConstructor/*.java` for how it had to be modified when `A`'s constructor now has to take one argument (I had to add a constructor in all the subclasses to correctly call `A`'s constructor with the passed argument).

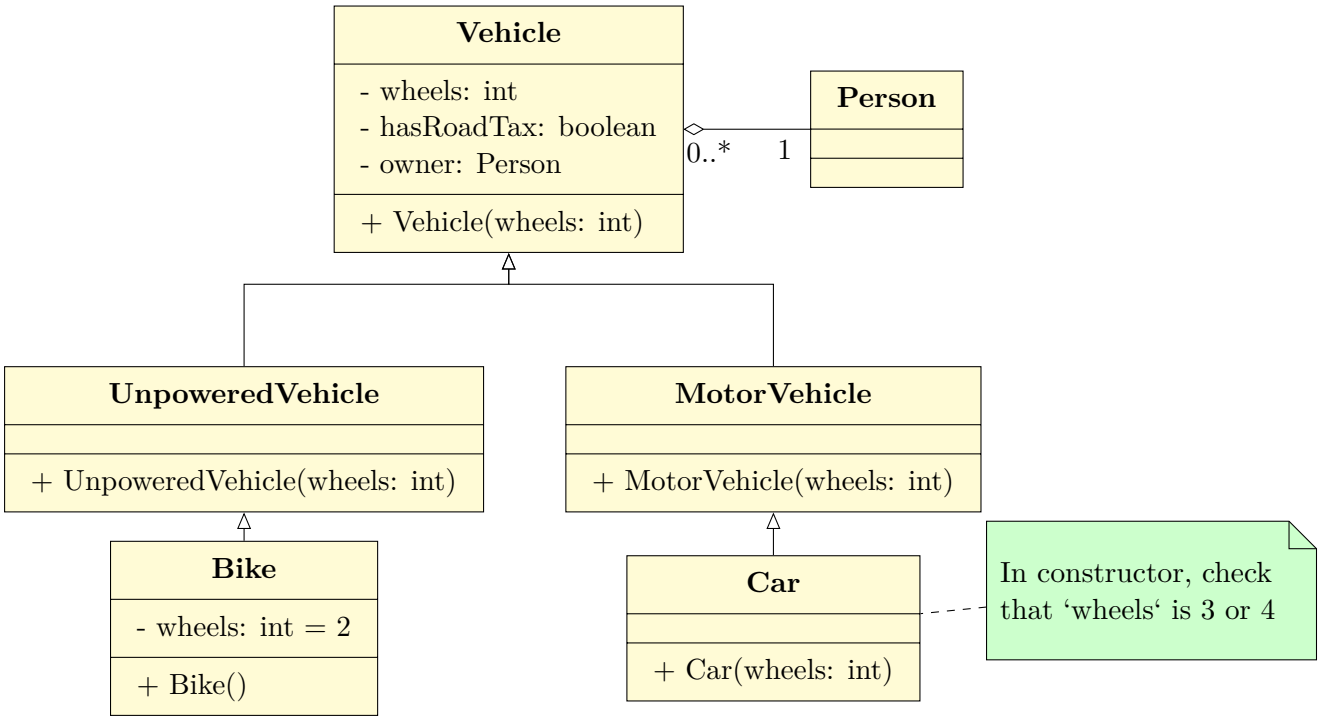
4.4

a



Getters, setters and any attributes for the relationships have been omitted for conciseness. Though this architecture means that an employee cannot be promoted without having to delete and recreate the object.

b



4.5

There will never be any runtime errors, only compile-time errors (as Java is smart enough to figure out at compile-time if you're not allowed to be accessing something)

	`public`	`protected`	unspecified	`private`
a	OK	OK	OK	Error
b	OK	OK	Error	Error
c	OK	OK	OK	Error
d	OK	Error	Error	Error

4.6

It will print `a.value = Super` and then `Sub.printValue: Sub`. This is because attributes exhibit static polymorphism so it will look at the declared type of `a` (`Super`) and access the attributes of that object. However, when methods exhibit dynamic polymorphism so it will look at the runtime type of `a` (`Sub`) and call `Sub.printValue()` and in that method, the compiler knows that the type of `this` will be `Sub` so will use `Sub.value`. This is because method dispatch is implemented using vtable, a hidden attribute that hold information about the actual type about the actual runtime type of an object and the method call uses this table to lookup the address of the correct `printValue` function to call.

4.7