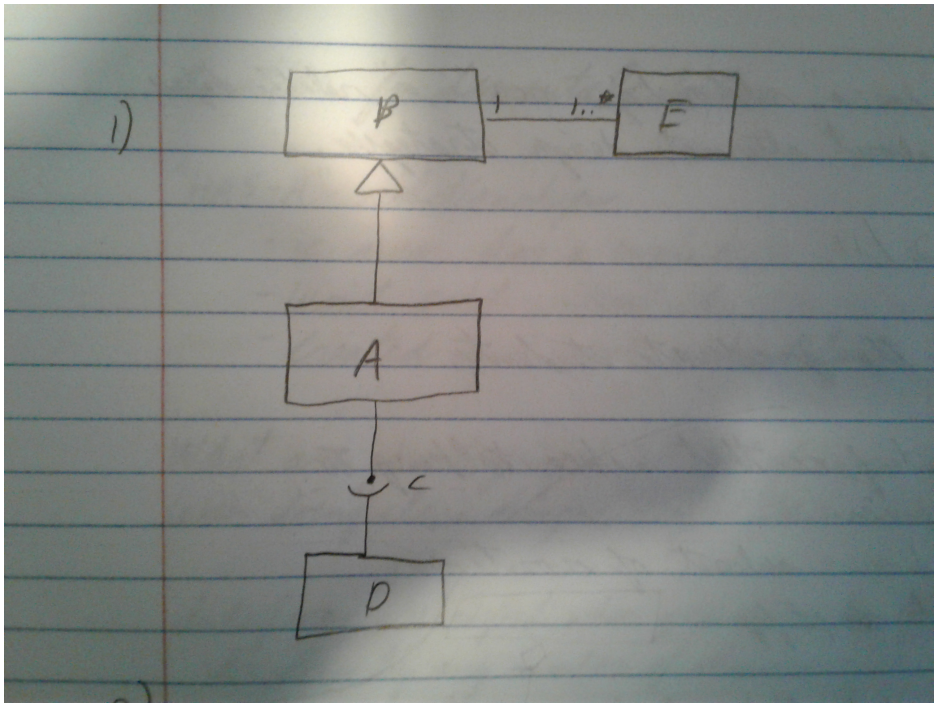
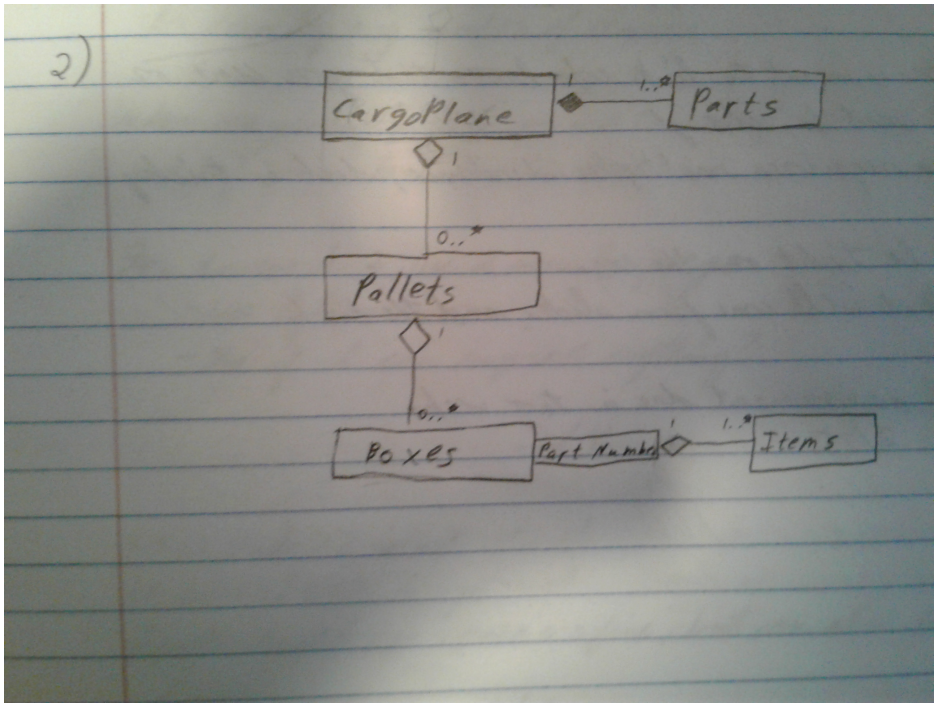
1 Solutions

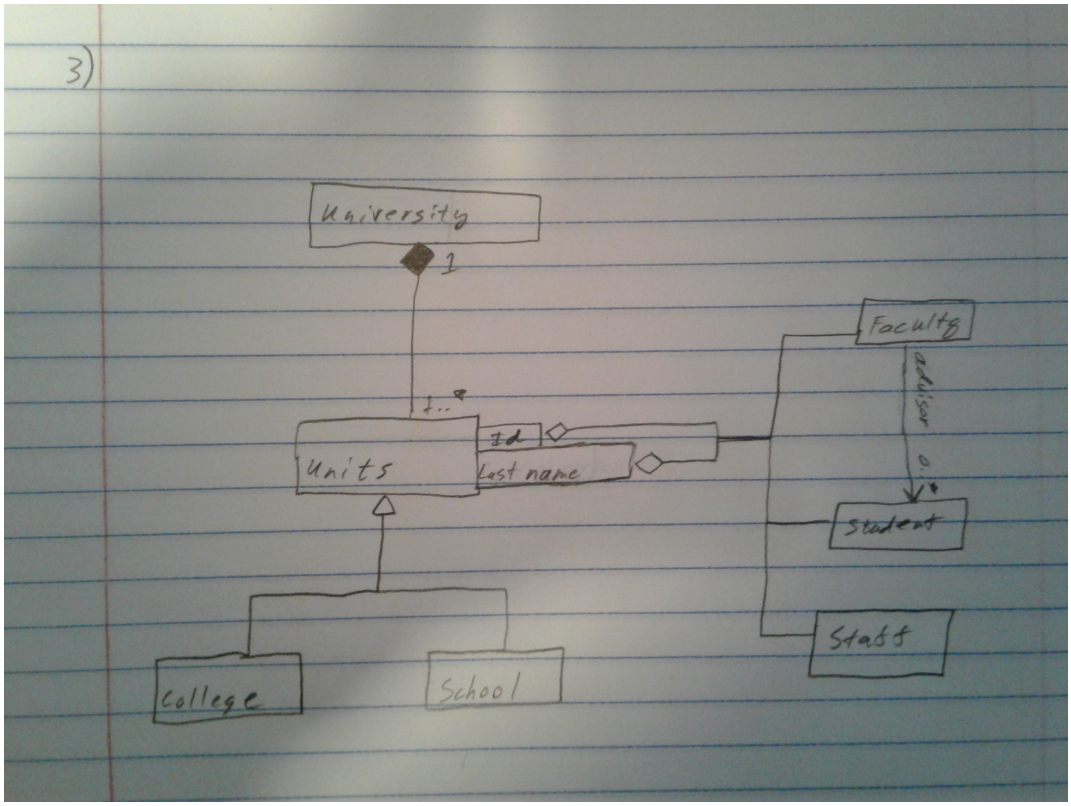
1. See picture below



2. See picture below



3. See picture below



4. Certainly not. B is a superclass of A, which means that it has no idea of A, or the implementation of A, including any interfaces that it implements. This makes sense from an OOP concept because A is inheriting the features of B, and not the other way around. We also are likely to want A to contain the implementation, including any interfaces. Also, if D could access an instance of B, then it could also access any other subclasses of B that might get created that don't have the interface implemented, just as B doesn't have the interface implemented.

5. The first and most important principle that the moron software engineer violated is that of having working code. The program will call `getArea()` and instead receive the perimiter, and since different engineers were working on different parts, unless he collaborated this with his equally stupid friends, the program will break.
   Now that that's out, he also violated the substitutability of polymorphic code, since now the main routine cannot simply call `getArea()` for all shapes, but instead must identify the type of shape, then if it is a Square, then call the actual method to get the area, whatever that may be. This also ties in with the fact that the engineer didn't do

a good job of coding to an interface, but rather just made code that worked and called it something arbitrary at best. This, in turn, creates poor abstraction.

The engineer also violated the maintainability of his code. The next engineer who comes along may or may not know the formula for the area and perimiter of a square off the top of his or her head, and it will waste lots of valuable time trying to figure out why the code works the way it does, rather than simply being able to read the function name.

This also in a different way violates cohesion. Cohesion is technically how closely the routines in an object are related, but it also includes the clarity of how much the routines make sense in a given object. And, in this case, it is super "unclear" what this method does.

So, in addition to violating some OOP principles, this is just bad software engineering, and should never be done, even unto thy worst enemies. In the words of DevHumor, "Write software like the person maintaining it is a psychopathic murderer who knows where you live."