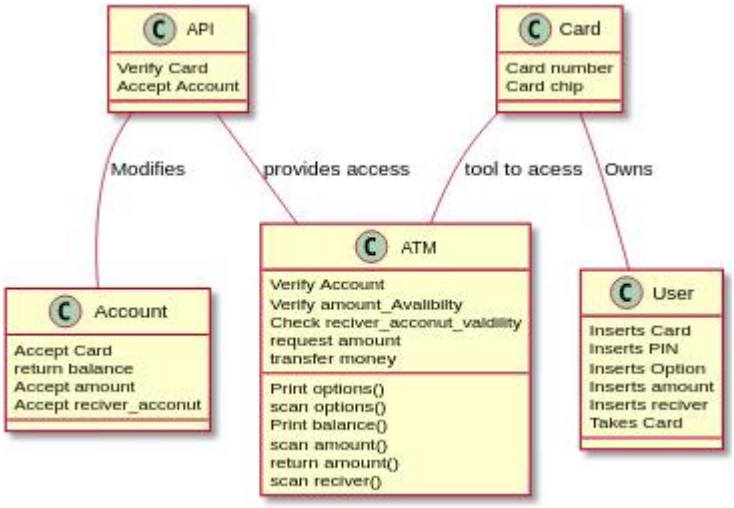
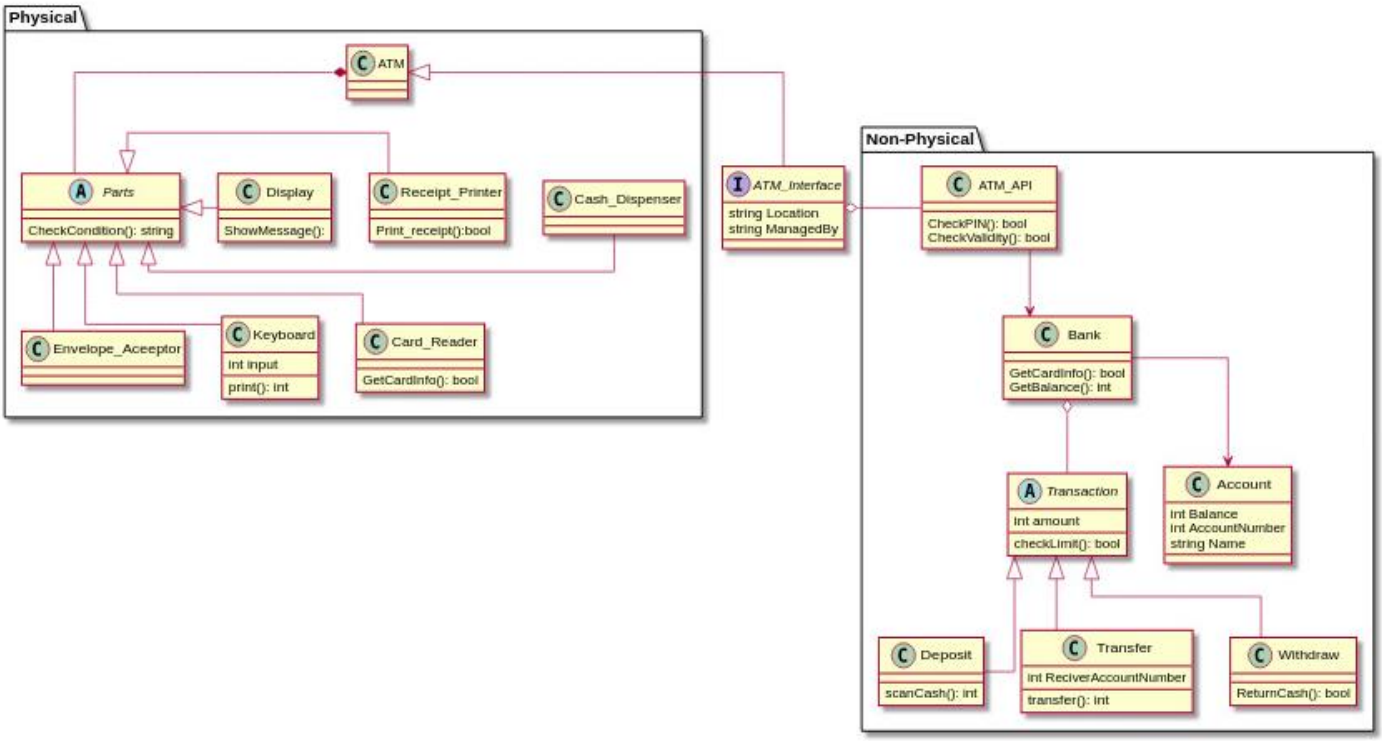
Analyzing lab 2 ATM



To start with, just by looking at the diagram you can see it is a very bad design. The diagram is very unclear were the couplings are just the classes know each other (association) which is wrong in many ways. That a class A knows class B I not enough in this diagram, a class A might own class B (aggregation, composition), a class A might be class B. Therefore, this diagram is not valid and not acceptable. The other problem is rigidity; all classes in my diagram have so many functions inside which means it will be allot harder to reuse the code, or to rewrite the code if the programmer detected a bug. In my class ATM I have so many functions and all the functions inside has too much responsibility, for example *transfer money* or in other words transactions can be a class itself because there are deposit, transfer and withdraw transaction. Therefore, in other words the diagram strongly violates *Single Responsibility Principle*. Let us say I solved this problem by making a class deposit, transfer and withdraw, where class ATM has all this classes (Aggregation). The problem is this does not follow the open-closed principle and these classes can be designed better. The solution is we create an abstract class and call it transaction were the classes deposit, transfer and withdraw inherit from it. Instead of modifying a class that does not quite fit our needs, we can declare a new class that inherits the features of the original class and adds new features. The new class can even redefine some of the inherited features. Another problem is that the classes depends on methods it does not use, for instance class account is dependent on methods it does not use. However, this is not what the Interface segregations principle is. The Interface segregations principle says a class should not be forced on methods it does not use, were in my diagram it is just a very not smart programmer that put methods in the wrong classes. Lastly, the diagram also violates *Dependency Inversion Principle* because high-level methods are depending on low-level modules. In other words, abstraction is not used at all in my ATM diagram. My diagram violates all five-design principles in SOLID design principle, which means this is not even close to how a real diagram of ATM.

Analyzing Lab 6 ATM



Unlike the diagram from lab 2, this diagram is better because it follows SOLID design principle. To start with, just by looking at the diagram we can see it is allot more clear than the one from Lab 2. All of my classes have one responsibility, which means it does not violate *Single Responsibility Principle*. The design of this diagram makes it allot easier for the programmer to add new features, which means it does not violate open-closed principle. In addition, none of my classes depends on methods it does not use for example class Deposit, Transfer and withdraw inherit from class transaction, were all three have their own methods that they use but also a methods that all three uses which they inherit form transaction. In other words, this diagram does not violate *Interface Segregation principle*. In addition, high-level modules in my diagram depend on abstraction and does not depend on low-level modules and vice versa. This diagram does not violate Dependency Inversion principle either. Lastly, even if the diagram is not perfect I think it is much better than the one from lab 2. I tried to follow SOLID design principle, which I think I did. I really want a feedback on this so I can improve.

Mohammed Darouich