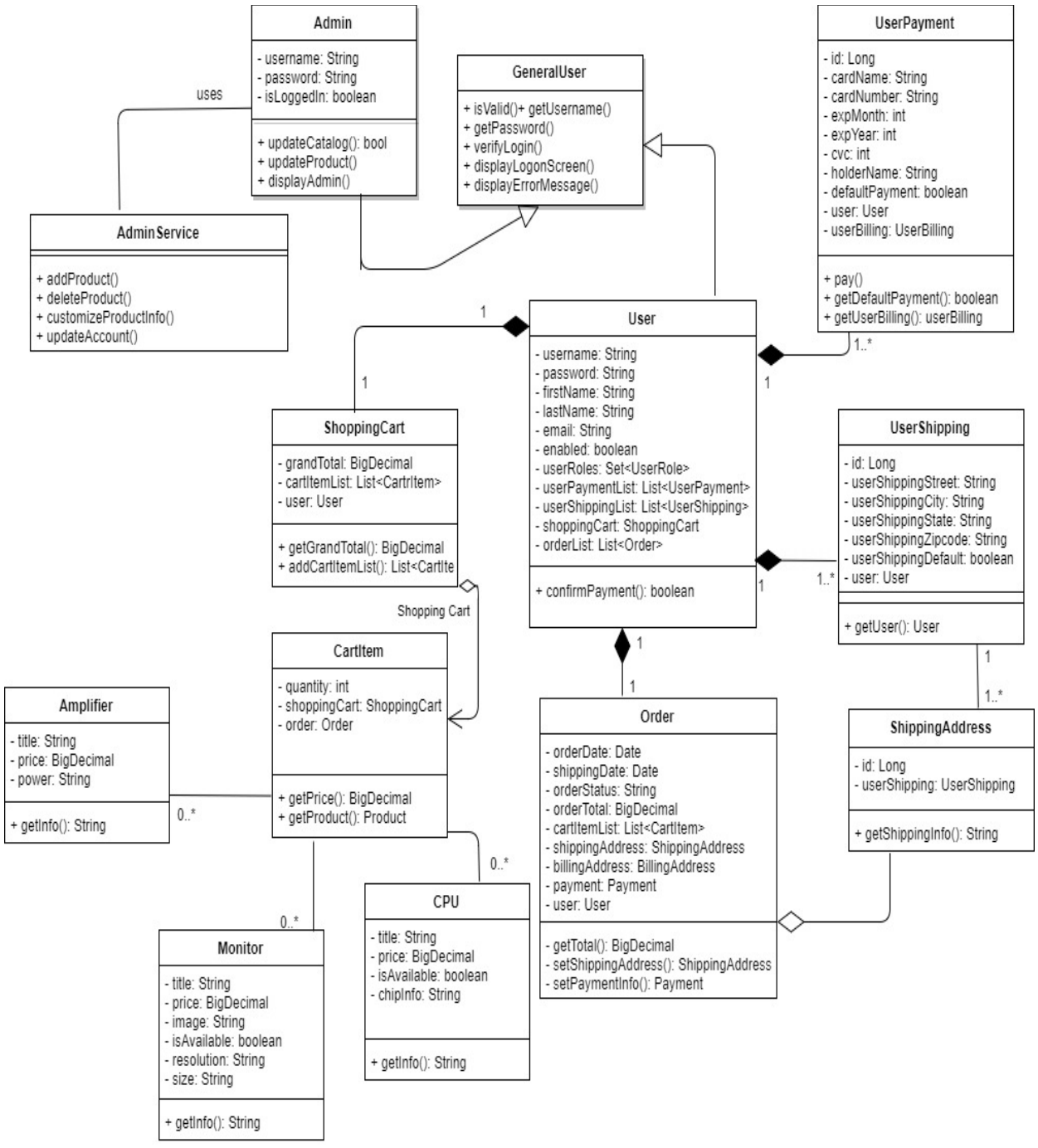
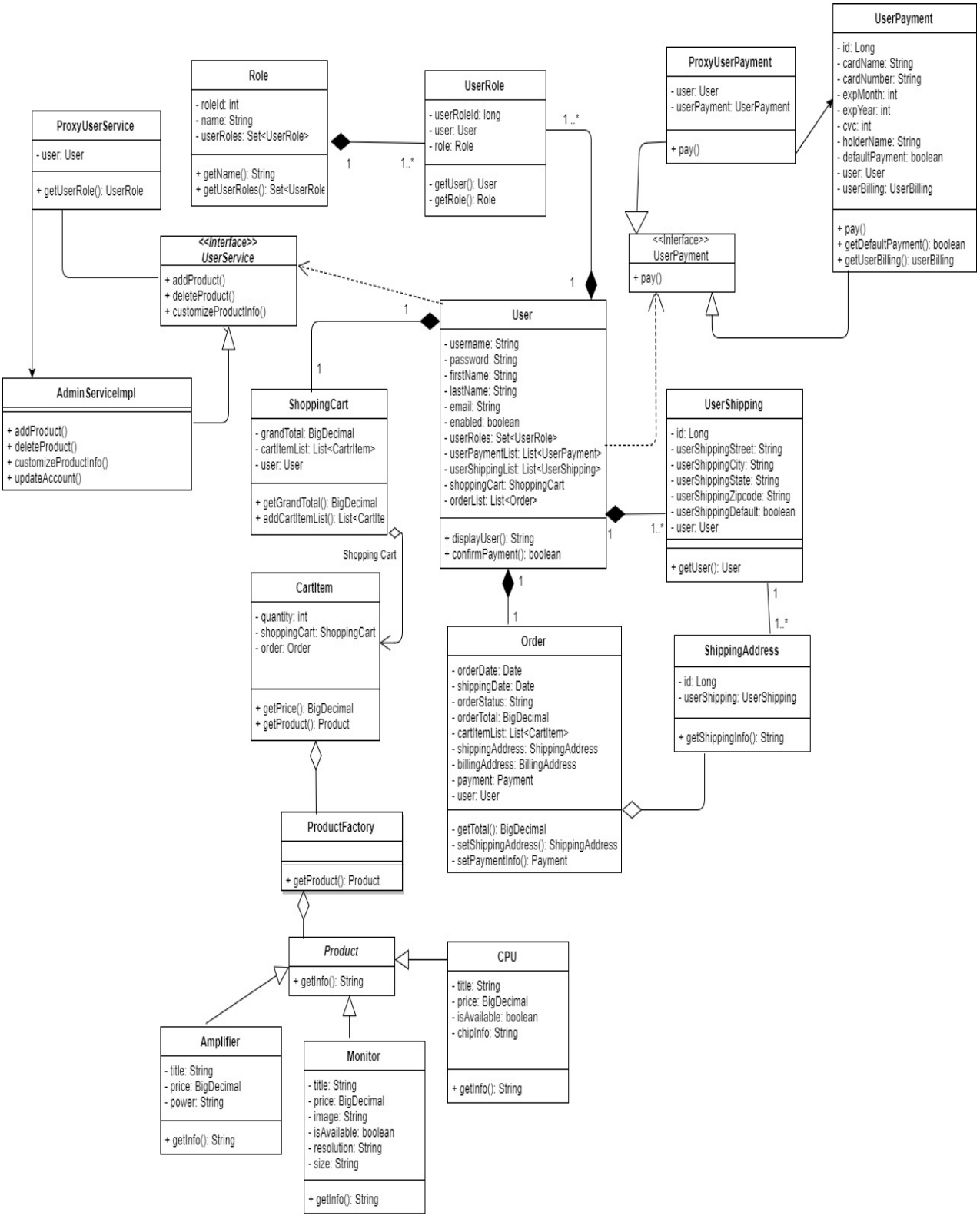
Electronics Store – Class Diagram ver1



Electronics Store – Class Diagram ver2



Refactoring Design Pattern

We realized some initial setbacks of our original architecture and applied some design patterns covered in class. The ***proxy design pattern*** is used for two sub features in our web-app, including authorize the functionalities of user/admin roles and user payment processing.

1> **Authorize user/admin actions** *(Protection Proxy)***:** Admins can add/view/edit product inventories while users can just perform transactions of available products. As we want to provide controlled access of specific functionalities to specific type of users, a proxy is needed to clarify user role of whoever uses the system. The real operation method is queried when the input parameter from getUserRole() is accepted as "admin".
2> **Payment processing** needs a certain level of protection and indirection from Proxy Smart Reference. As long as the payment is confirmed, the real UserPayment class is called to implement its function, otherwise, the real operation methods are hidden from client side. Also, for the reason that the object is not necessarily created immediately, only needed when the customer finishes ordering and confirmPayment() is true.

We also use the ***simple Factory design pattern*** to replace class constructors (if the system scales up and there are a wide range of categories of products sold) by creating an abstraction through which classes are returned determined at run-time.

Also, we have read and decided to use the **Simple Data Access Layer Pattern** with JDBC which simplifies database access operations through some method calls like CRUD operations rather than making connection and executing some queries. This additional layer instead handles all database related calls and queries inside it. On the front-end side, we use AngularJS which utilizes ***Observable pattern*** for component to communicate with the root app component via a message service *subscribe()*