CLASSROOK

Software design patterns.

1. The Database class uses the **singleton** design pattern. Database operations require atomicity in most cases. For example, when two downloads happen at the same time, the download counter of a file should not be incremented twice, which requires two separate update operations to happen sequentially. The problem becomes even more pressing when we keep multiple copies of the database across different servers when the service scales up horizontally. To address this concern, the database object offers a single point of access that is available globally.

2. The Course class uses the **facade** design pattern that hides details of underlying database operations from users. Users can interact with reviews and course materials in various ways, which require multiple database operations. For example, when a user intends to download a course material, the database is accessed three times to deduct the downloader's credit, to increase the uploader's credit, and to transmit the file object. Instead of exposing those operations to users, the course class only provides interfaces that they really care about. Additionally, the facade design pattern guarantees the low coupling principle. Change of database, for example, would not affect the logic of interacting with reviews and course materials.

3. The sorting strategy class uses the **strategy** design pattern which allows both the objects (reviews or files) and the sorting algorithms to be interchangeable. When displaying reviews and course materials to the users, they could be sorted according to different criterions specified by the user. The implementation of the sorting algorithm is independent of both the specific object that is being sorted, and the attribute of the object that is sorted by. What's more, depending on the size of data that needs to be processed, different variants of the sorting algorithms can be implemented. For example, when the total number of reviews become so large that even a O(nlogn) algorithm will take too much time to finish, we can create a variant of sorting strategy that only yields an approximate result.