Homework 4 - Proxy Design Pattern

The Proxy Design Pattern works exactly as the name implies: we employ a middleman between the user and the actual function that we want them to interface with. This middleman implements the same abstract class as the actual function does, but it makes some additional checks before using the actual function as intended.

As an example, take my program. The “actual” function we want the user to use is FileFetcher, but we want to ensure that the user can only fetch files that are appropriate for their standing; Guest users can only fetch files that don’t have “classified” in the title, and Agent users can only fetch files that don’t have “entertainment” in the title. If we simply let both of these users use a class UnrestrictedFileFetcher, they would access things they should not be seeing, so we create proxy classes ProxyCivillianFetcher and ProxyOfficialFetcher, which check if the file being requested is appropriate. If it’s found to be appropriate, then these proxies simply call UnrestrictedFileFetcher on the requested file, but if it was inappropriate an error message is given. This way we have separated the user from UnrestrictedFileFetcher, despite the fact that it is still called under the hood.

The benefit of this design pattern is we can make our system more secure by limiting the way in which a user interacts with our underlying objects. With unrestricted access to the underlying class a user can run into behavior that was never intended to happen and might crash. This however leads into the drawback, which is that if the user can control when to use our proxy, they might be picking and choosing when to use it and when not to, leading to confusing and contradictory behavior because they occasionally choose to overlook our proxy setup.