

Task One

Hypothetically we have a code review checklist in our team

| | Metrics | Descriptions | principles |
|--|--|--|----------------------|
| | Test coverage | Is there a need to test more cases? | |
| | Abstraction | | OOP basic principles |
| | Encapsulation | | OOP basic principles |
| | Polymorphism | | OOP basic principles |
| | Inheritance | | OOP basic principles |
| | Association, Aggregation and Composition | | OOP |
| | Composition over inheritance | | OOP |
| | Single responsibility | | Solid Principal |
| | Open-closed | | Solid Principal |
| | Liskov substitution | | Solid Principal |
| | Interface segregation | | Solid Principal |
| | Dependency inversion | | Solid Principal |
| | Creator | | GRASP Principles |
| | Information Expert | | GRASP Principles |
| | Low Coupling | | GRASP Principles |
| | High Cohesion | | GRASP Principles |
| | Controller | | GRASP Principles |
| | Pure Fabrication | | GRASP Principles |
| | Polymorphism | | GRASP Principles |
| | Readability | Are there any redundant code and comments | |
| | Security | Does the code expose the system to a cyber attack | |
| | Architecture | Does the code use encapsulation and modularization to achieve separation of concerns | |
| | Reusability | Does the code use reusable components, functions, and services | |
| | Long Method | | Code Smell |
| | Large Class | | Code Smell |
| | Long Parameter List | | Code Smell |
| | Hidden dependency | | Code Smell |
| | Gang of four Design pattern | | |
| | Team Convention | Code Convention, Code Preferences | |
| | | | |
| | | | |
| | | | |
| | | | |

- 1- Regarding avoiding **Hidden Dependency** I suggest Constructor dependency injection for all of your classes instead of Resolving Dependency by IOC container in the class constructor
- 2- Regarding **Single Responsibility** you can delegate the responsibility of `AssignValuesFromArguments()` to `AppSettings` object because it has already read the setting from `ConfigurationManager` in `ReadAllSettings()`
- 3- Exception Handling Issue
First Problem: `Environment.Exit(exitCode)` in `HandleError` in `ConsoleErrorHandler` suppresses and exits the app when get called, then if you have a `finally{}` in upper-level or below of catch that calls `HandleError()`, this `finally{}` has been never called. Because in catch you exit from the app.
Second Problem: redundant try-catch throughout the project that they don't do any specific business.
I would handle the general exception in `Main()` of `Programs.cs`.
- 4- `Backup.cs` class can implement `IDisposable` and in `Dispose()` method checks the business of File deletion. Consequentially we can delete Try Catch Finally in `Run()` from this file. Also we should call `backup.run()` in `using (){}` in `Program.cs`. that is guaranteed `Dispose()` get called for releasing the resource and memory
- 5- It doesn't need to keep `AppSettings` properties `static`, if your purpose was that keep them alive during the app life-cycle, you have already registered `AppSettings` as a singleton instance. From my perspective, I stay away from static classes or methods as much as possible unless in a rare situation.
- 6- `Get(string key)` in `AppSettings` needs to check if `ValuePairs.TryGetValue(key, out value);` cannot find the key. Then we can have a guard that if there isn't a key throw an exception.
- 7- In `Download()` method in `SalesForceWebDownloader` the `baseAddress` is built by some format and setting. And it isn't only here, there are a lot of places where we create some URLs and addresses for reaching our third-party provider. According to single responsibility, I prefer having another essence and nature that can be considered as a service (**Pure Fabrication**) to provide us with addresses and call it `AddressProvider`.
- 8- In `Login()` method in `SalesForceWebDownloader` if the login fails we should have a guard and handle it with the appropriate exception
The other point here is why `sfClient.Url` and `sfClient.SessionHeaderValue` is initialized, while only `sessionId` is returned. The scope of `sfClient` instance is just during the login method and it doesn't have any effect.
- 9- `DownloadExportFile()` method in `SalesForceWebDownloader` doesn't need to be a static method
- 10- Why `DownloadWebpage()` method in `SalesForceWebDownloader` isn't an async Method? Instead of taking the Result of `client.SendAsync()` we can await it, `Await` is an asynchronous wait but the result is a blocking wait.
Note: As you know we have a console app which seems for each time it gets run and take a backup then get closed at the end automatically. current code shows we don't have business requirements for parallel processing. Then asynchronous wait may not have any place for discussion. But I prefer to have a comprehensive approach and design through entire the app. For example instead of having `DownloadExportFile()` method as async and `DownloadWebpage()` method as sync, I prefer having both of them as Async (one signature), then finally `await backup.run()` in the `Main` method in `Program.cs`.
- 11- `Download()` method in `SalesForceWebDownloader` instead of returning `files.ToArray()` can return `files`; consequentially in the method return signature instead of `string[]` would be better return `Task<List<string>>`.
- 12- I extract two additional functionalities which can separate them into another method in `DownloadListOfExportFiles()` method in `SalesForceWebDownloader`

```
var page = await DownloadWebpage(_appSettings.Get(AppSettingKeys.DataExportPage), sessionId);
```

```
var matches = GetMatchingItems(page);
return GetExportFiles(matches);
```

- 13- In `IUploader` interface I would name the input of `Upload` as a `filePath` instead of `file`. Because the nature of it is path
14. In `S3Uploader` we have a try-catch and for better performance, I would use `when Syntax` in front of `catch` instead of checking the `if` in `Catch`
15. It doesn't need to keep the `ExitCode` enum inside `Enums` class. You can take `ExitCode` enum out of this class and delete the `Enums` class. Also edit the file name from `Enums.cs` to `Exitcode.cs`

////////////////////////////////////

16. Up to now we reviewed the code as it is, for avoiding off-topic discussion I assumed this code is based on our requirement and just reviewed the code. Maybe our servers are only able to execute the .net framework, not .net 6,7,8. But from now on as additional discussion, I want to take the step forward and discuss beyond the present topic.
 - 1) The code should have a test coverage. We consider the test as the first client which tests the business. We consider tests as online documents.
 - 2) If the commented explanation above each method or class is part of our team convention, I don't have a problem with it, but if not, I prefer not to have them, instead, I use self-explanatory naming for each element also using unit test coverage. Then my code review start point is reviewing the test scenario to understand the functionality.
 - 3) Now is September of 2024, and if we assume that we don't have any issues with using .net core or .net 7,8, then I prefer having a .net 8 one, it is a small project and it doesn't take too much effort to migrate it to the .net 8 one.
 - i. **Note:** My-dotNet8-Refactored-SalesForceBackup project is accessible in [my GitHub by this address](#) . it is WIP (work in progress) and I am working on it now.
 - 4) We assume knowing about the advantages of .net 8.
 - 5) We can Use the advantages of `IHostBuilder` and also can have a `BackgroundService` that names it as a worker. The worker can listen to the channel or TPL Dataflow.
 - 6) Then we can design entire of the flow as an async process. A user from the UI sends a request for making a new backup, then as soon as the request comes to the Controller, we dispatch `BackupCommand` to download the files from `SalesForceWebPage` and put them on the channel, then respond to the customer that "the backup is processing". Also, we can use `SignalR` as a push notification to show a progress bar to customers. Then Worker is listening to the channel and takes the downloaded files one by one from the channel and tries to process them and upload them as an Uploader to Azure or AWS. When the process is finished then push a notification to the customer that it gets finished. I will elaborate on it in the Third Task.
 - 7) Using `HttpClientFactory` instead of `HttpClient` for taking advantage of `httpClientPool` and having better management of HTTP request

Some Code Review Metric

| | | | |
|--|------------------------|--|----------------------------|
| | Inspection rate | The speed at which your team reviews a specific amount of code, calculated by dividing lines of code (LoC) by number of inspection hours | code review metrics |
| | Defect rate | The frequency with which you identify a defect, calculated by dividing the defect count by hours spent on inspection | code review metrics |
| | Defect density | The number of defects you identify in a specific amount of code, calculated by dividing the defect count by thousands of lines of code (kLOC). | code review metrics |