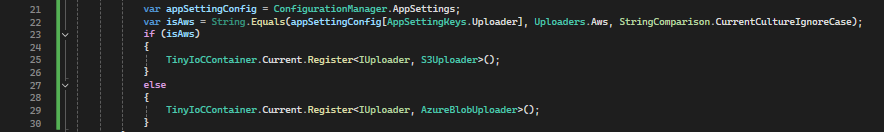
1. To avoid Hidden Dependency I suggest Constructor dependency injection for all of your class instead of Resolving Dependency by IOC container in class constructor

In Bootstrap.cs in Register(), according to AppSettingKeys.Uploader you decided register S3Uploader or AzureBlobUploader and you instantiated them Manually then for avoiding get stuck to instantiated the other dependency you decided to resolve dependency by IOC Container then it drive you to **Hidden Dependency** as one of the Code Smell. You can write it such as below:



1. Regarding **Single Responsibility** you can delegate the responsibility of AssignValuesFromArguments() to AppSettings object because it has already read setting from ConfigurationManager in ReadAllSettings() then it doesn’t need to have the below line codes in Program.cs

\_appSettings = TinyIoCContainer.Current.Resolve<IAppSettings>();

AssignValuesFromArguments(args);

1. It doesn’t need to keep AppSettings property static, if your purpose was that keeping them alive during the app life-cycle, you can consider AppSettings instance as a singleton object. From my perspective, I stay away from static classes or methods as much as possible unless in a rare situation.
2. I don’t have a problem using TinyIoCContainer until at least it provides our minimum and basic requirements.

If we consider the scenario that we have many projects in our team then I prefer using a comprehensive tools and solution entire projects in our team. Choose appropriate IOC container is one of the design decision which we can have in our team, there are a lot of mature, and simple IOC in the market that have a good community, Contribution and support some feature