# Faculty of Computer Science

## CSCI 5308 – Advanced Topics in Software Development

## Assignment 02

Submitted By:

Rishi Vasa

Banner ID: B00902815

CSID: vasa

Email ID: rishi.vasa@dal.ca

30th October 2022 Rishi Vimal Vasa rishi.vasa@dal.ca
B00902815

## GITLAB LINK TO THE PROJECT:
## https://git.cs.dal.ca/courses/2022-fall/csci-5308/vasa/-/tree/main/A2

## DESCRIPTION:

### 1) INTERFACE SEGREGATION PRINCIPLE:

Description of violation of the principle in the Bad Package:
The inversion segregation principle has been violated in the program executed by running the bad package's "MainBadInterfaceSegregation" class. I have created a simple program that fetches the hardcoded details for Spotify tracks and podcasts.

Thus, the "SpotifyTrack" and "SpotifyPodcast" both implement a single common interface "ISpotifyPlayer", which contains a contract to getTitle, getArtists, getAlbum, getNarrator, getSeason and getEpisode.

Tracks and Podcasts have different attributes. Tracks are not organized by season or episode number, and they do not have a narrator. Podcasts are not categorized in albums either. Thus, implementing a common interface which contains too many methods will not apply to all Spotify content. Thus, the interfaces need to be segregated.

This program also inadvertently violates the Liskov's Substitution Principle as the return nothing methods in the SpotifyTrack and SpotifyPodcast classes will refuse bequests when called directly.

Description of the fix applied for the principle in the Good Package:
The interface structure has been altered to fix the violation. The common methods that can be implemented by both – Tracks and Podcasts – are getTitle and getArtists, as each Podcast and Song will have a title and can have multiple artists creating it. So, a common interface called "ISpotifyPlayer" has been created which contains a contract for just these two methods.

Further, two more interfaces "ISpotifyTracks" and SpotifyPodcast" have been created and implemented by the SpotifyTrack and SpotifyPodcast classes respectively. These interfaces only contain the methods that are required by the Tracks and Podcast classes.

Now, in the future, we can add more functionality or attributes which are specific to podcasts or songs. If Spotify decides to add videos to its library soon, we can create another interface which would implement methods related to videos as well, making our code easy to change.

## 2) DEPENDENCY INVERSION PRINCIPLE:

Description of violation of the principle in the Bad Package:
The dependency inversion principle is violated in the program executed by running the bad package's "MainBadDependencyInversion" class. It's a simple program that fetches co-ordinates of the uber ride's source and destination addresses from Google/Apple Maps API.

The UberRide class fetches the coordinates from two separate APIs, for which it is dependant on two concrete classes. Depending on changing business requirements, API's can be switched. When this happens, every class dependant on the concrete API classes will need to be changed. Thus, instead of depending on concrete classes, we need to program to interfaces and be well prepared for inevitable change.

Description of the fix applied for the principle in the Good Package:
Instead of depending on concrete classes, dependencies have been injected to the UberRide class. To do so, an interface has been created "ICoordinates" which guarantees that coordinates will be faced based on the input address.

Now, AppleMapsAPI and GoogleMapsAPI implement the ICoordinates interface and fetch the required data from their APIs. The UberRide class now depends on abstractions implemented by the ICoordinates interface as the methods getSourceCoordinates and getDestinationCoordinates now accept an instance of the implementation of the interface.

This means in the future, if the requirements change and the API changes to bing maps, a new API class "BingMapsAPI" can be created which implements the ICoordinates interface, and it can be instantiated via the interface. UberRide will continue functioning as it is since it isn't concerned with *who* is implementing the logic to fetch the coordinates. It now depends on abstractions rather than concretions.

## 3) LISKOV'S SUBSTITUTION PRINCIPLE:

Description of violation of the principle in the Bad Package:

The Liskov's Substitution principle has been violated in the program executed by running the bad package's "MainBadLiskovSubstitution" class. It's a Facebook Post program which fetches the statistics of different types of Facebook posts such as a Text post and a Reel Post.

The abstract class "FacebookPost" is extended by "ReelsPost" and "TextPost". The FacebookPost class does have some methods such as getPostID and getLikeCount which can be utilized by both ReelsPost as well as TextPost. However, FacebookPost also contains methods like getAudioUsed which is not applicable to TextPosts and getViewsCount for which the calculation behaviour of TextPost and ReelsPost is entirely different.

This means, TextPost has to return a dummy value or throw an exception if getAudioUsed is called from its object. Moreover, ReelsPost is altering the expected behaviour of the getViewsCount method by adding Replay Count. These are a few ways in which the LSP is being violated in this program.

In the future, if more types of posts would have been added to this program, it would perhaps also require certain guard clauses to differentiate between the type of object during runtime. This means we are losing the ability to program generically.

Description of the fix applied for the principle in the Good Package:

Resolving the Liskov's Substitution Principle would require altering the inheritance structure of the program. Instead of ReelsPost and TextPost directly extending FacebookPost, 2 more abstract classes have been created – MultimediaPost and SimplePost – which extends the common to all functionalities of FacebookPost.

Moreover, an extra type of Post – ImagePost has also been created which extends SimplePost alongwith TextPost. ReelsPost now extends MultimediaPost and is now free to utilize its getAudioUsed method and overwrite its logic to getTotalViewsCount.

Due to this rearrangement of the inheritance structure, in the future if we require to add more post types, we can simply create one more abstract class which would extend the base FacebookPost class and add its own functionality on top of it. This makes sure we even adhere to the Open/Closed principle making our base classes closed for modification, but open to extension.

## REFERENCES:

[1]. Rob’s Brightspace Course content for SOLID principles.
https://dal.brightspace.com/d2l/le/content/230468/Home?itemIdentifier=D2L.LE.Content.ContentObject.ModuleCO-3095831