Structural Design Patterns

Social Media and Restaurants

Flyweight

Assume you are developing a text editor application that allows users to create, edit, and save documents. Each document may contain a large number of characters, and each character can have their own font, size, and color.

3 examples of each:

FONTS	COLOR	SIZE
Arial	Red	12
Calibri	Blue	14
Verdana	Black	16

You realize that many characters will share the same properties and you will optimize the memory usage of the text editor by using the Flyweight design pattern. Specifically, you need to create a set of flyweight objects that represent the character properties (font, size, and color), and share these objects among all characters that have the same properties.

Your implementation should allow the user to create and edit documents with different character properties and provide the ability to save and load documents from disk.(save and load from File)

HINT: It may be difficult to delete character properties. Therefore, do NOT add the ability to delete character properties once it has been created.

For the driver program save “HelloWorldCS5800” with at least 4 variations in character properties.

Proxy

Assume you are developing a music streaming application that allows users to listen to their favorite songs. The application has a large library of songs, and each song has its own metadata which are: (String)title, (String)artist, (String)album, and (int)duration.

Your task is to implement a proxy object that acts as a cache for the song metadata.

NOTE: *You do NOT need to play music.* You are only concerned about the metadata!

The proxy object should fetch the metadata from the server on-demand and store it in memory for faster access. To help you get started you have been provided the SongService Interface:

```
import java.util.List;

public interface SongService {
    Song searchById(Integer songID);
    List<Song> searchByTitle(String title);
    List<Song> searchByAlbum(String album);
}
```

Your implementation should use the Proxy design pattern to encapsulate the caching and use the interface provided for the application to access the song metadata.

For the driver, create 5 – 10 songs and attempt to search for them using the methods from the SongService shown above. For a more realistic representation of this application I would recommend adding a 1 second delay on the real server calls to understand the advantage of using a proxy. Essentially add this code block below at the beginning of the three methods for the real server only, NOT the proxy.

```
try {
    Thread.sleep(1000);
} catch (Exception e) {}
```