PROG 1400 - Assignment # 2

Refactoring for OOP Principles

Outcomes:

#1. Research the principles of object-oriented programming, including encapsulation, inheritance and polymorphism
#2. Implement object-oriented design principles in OOP applications

This assignment is going to challenge you in the following areas:

- Understanding existing code that you didn't write
- Identifying violations of object-oriented principles
- Refactor existing code to better adhere to object-oriented principles of **classes, encapsulation, inheritance, composition**

Instructions

1. Clone the Assignment #2 repository to your local machine
2. Download the Assignment #2 source code from Brightspace
    a. Create a branch in your repository
    b. Add the source code from Brightspace
3. Familiarize yourself with the source code and try to identify violations of object-oriented principles
4. Refactor the code so that it adheres to object-oriented principles of **classes, encapsulation, inheritance, composition**
5. Extend the program by adding your own music player class (iPhone, CD Player, Radio etc.)
    a. Add unique **state** and **behavior** for your music player class that make it different from the MP3Player and RecordPlayer classes
6. Add an enumeration called "**MusicGenres**"
    a. It should have at least 5 entries
    b. Add an instance variable of the type "**MusicGenres**" to the **Song** class. When you instantiate a song, it you will need to assign it a value from your **MusicGenres** enumeration
7. Commit your changes to a branch and push to your remote repository on GitHub
8. Create a Pull Request for your branch and submit the Pull Request URL on Brightspace

Hints:

- Identify the duplicated behavior of the MP3Player and RecordPlayer classes, extract that behavior into a superclass
- Remember how important **encapsulation** is, objects should not be directly manipulating or directly accessing the **state** (instance variables) of other objects

- Sub-classes **inherit state** and **behavior** from super-classes. Sometimes they may wish to modify the **state** or **behavior** they inherit from their super-classes.
- You should have a constructor for your **Song** class.
- **Constructors** aren't necessary for the **MP3Player or RecordPlayer** classes.