COSC2391/2401 Software Architecture: Design and Implementation

TuteLab9 – Java Streams, Serialization and Files

For this tutelab you are going to write some file handling functionality that you can use in your assignment!

Your task is to implement the following interface:

```
public interface GameLoader
{
  // load all players from a file and return them as a Collection<Player>
  public abstract Collection<Player> loadAllPlayers(String path)
       throws GameLoaderException;

  // save all players from a Collection<Player> to a file
  public abstract void saveAllPlayers(String path, Collection<Player>
      players) throws GameLoaderException;

  // append a single player to an existing file
  public abstract void appendPlayer(String path, Player player)
      throws GameLoaderException;
}
```

For a simple text file implementation you could implement the following class:

```
public class GameLoaderText implements GameLoader
```

Note that in addition to implementing the class you will also need to create the class `GameLoaderException` which extends the `java.lang.Exception` class thereby being a *checked* exception. You can throw this in your implementation with an appropriate message (“File not found” etc.)

A text file for this implementation will consist of a single player per line whereby the id, name and points are comma separated e.g.

```
1,Player1,100
2,Player2,200
3,Player3,300
...
```

GETTING STARTED: You can refer to the Topic8 lecture notes and source code examples if necessary but the exercise will be much more beneficial if you try to do it from first principles by referring to the API docs for `FileInputStream/FileOuputStream` as well as the *decorating* classes such as `PrintStream` for writing or the `Scanner` for reading. Specifically, doing it from first principles will help you understand the individual classes and methods leading to more robust code rather than trying to cut/paste/hack it together which leads to fragile code and thus is always strongly discouraged!

ADVANCED

If you finish the text based implementation and successfully incorporate it into your assignment then you can produce an additional binary implementation called `GameLoaderBinary` which uses `DataInputStream` and `DataOutputStream` to read/write from/to a binary file using the same interface so that it can be substituted directly into your assignment to replace the text based implementation.

If you do this you will likely find that you have some common functionality since opening/closing the files and certain exception handling is common to both. To solve this problem you should create an abstract superclass called `AbstractGameLoader` which factors out the commonality.