Scientific Computing and Object Oriented Programming - Homework 3

On demand (video) content in C#

Deadline 29-04-2018 23:59

Objective Design and implementation of *playlists* management (loading and saving) for on-demand video service.

Thanks to the excellent job you made in previous assignments, the *scoopflix* company decided to assign you the design of software for loading and saving of users' video playlists.

Playlist class

Define a `Playlist` class which behaves like a *view queue* of `IPlayable` objects (pretty much like Spotify playlists). Several `IPlayable`s can be added to a `Playlist`; The `Playlist` class keeps track of the *current played* content. Define and implement the following method of the class `Playlist`

`public void Add(IPlayable content)` Adds the `IPlayable content` to the playlist positioning it at the tail of the view queue.

`public void Next()` Goes to the next video in the queue throwing an exception (you can choose a library one or create your own) if there is no next video.

`public IPlayable GetCurrentContent()` Returns the `IPlayable` that is currently played.

`public boolean HasMoreContent()` Returns `true` if and and if there is/are more video(s) in the queue after the current one.

`public void Rewind()` Rewinds the playlist so that it can be started from the begin.

Playlist I/O

Define a class `PlaylistIO` that contains two methods

`public static Playlist LoadPlaylistFromFile(string filePath)` Creates, fills and returns a `Playlist` object containing all the videos that are stored in the file with path `filePath`. The format of the file is described below. This method **must** correctly load the `videos.txt` companion file.

`public static void SavePlaylistToFile(Playlist pl, string filePath)` Saves the content of `pl Playlist` in a file with path `filePath`. The saved file must have the format described below (*i.e.*, the same as the `videos.txt` file).

File format Playlist files contain one line for each video. Each such line contains all the information of the video separated by the special character `#` (sharp). More precisely every line contains: *type of video*, *title of video* and *duration in seconds*. Moreover (based on the specific video type) further information is present in the file.

- **Documentary** (`Type=D`) contains the *topic* of the video.
- **Series** (`Type=S`) contains: *series name*, *season number* and *episode number*.
- **Movie** (`Type=M`) contains *genre*

For example the following lines are taken from the companion file `videos.txt`

```
D#Inside Job#6480#economics
S#End Times#2760#Breaking Bad#4#12
M#Back to the Future#6960#science-fiction
```

Hint To implement the the save method it could help to add the `ToRecord()` method to the `IPlayable` interface, such method will return a `string` conform to the convention described above (*i.e.,* `#` separated fields). Alternatively you can check the type of `IPlayable` (*i.e.*, `Documentary`, `Series` or `Movie`) using the `is` operator. As an example of usage of such operator, the following code prints `Ok` only if `p` is of type `Video` (or a direct/indirect subclass like `Movie`)

```
if (p is Video) {
  Console.WriteLine("Ok");
}
```

Main method
Create a `static Main` method that tests the classes and methods developed in the previous part by performing the operations described next.

1. Loads the companion file `videos.txt` into a `Playlist` object.
2. Creates three more `Playlist` classes each containing only the videos of a specific type (*i.e.*, documentary, serie and movie). This operation **should not** change the content of the playlist containing all videos. To create the three playlists use the `Next`, `GetCurrentContent` and `HasMoreContent` methods of the class `Playlist` (**do not** access fields or properties of the class `Playlist` from the `Main`).
3. Saves each of the *type specific* `Playlist` into three distinct files (one per playlist). You can check that the saving operation works correctly by loading the new files using the `LoadPlaylistFromFile` method.

Important note You are not allowed to use any `.NET` or external library classes other than those necessary to load and save content into files. For example you cannot use third party classes that implements queues, lists or other data structures, nor you can use classes other than `string` to manipulate strings.

Submission Within the deadline indicated above, each student must submit

- A zipped archive (`Homework3_StudentId.zip`) with source code (`Homework3_StudentId.cs`) and compiled file (`Homework3_StudentId.exe`) should be uploaded on moodle and
- All the classes which *could* be on a single file `Homework3_StudentId.cs`, where `StudentId` is your identifier (numero di matricola).

Submission steps

1. access the web page of the course on moodle: `https://elearning.unipd.it/dicea/course/view.php?id=792`
2. click on *Homework3* in the *Homeworks* section
3. click on *add submission*
4. upload the zipped archive through the available form
5. click on *Edit submission* to modify, if needed, your submission
6. click on *Submit assignment* to submit the homework; once the assignment is submitted you will not be able to make any more changes

If an error occurs during the submission, send the zipped archive via email to `emanuele.dibuccio@unipd.it` and `michele.schimd@unipd.it`