Object-oriented programming

Laboratory classes

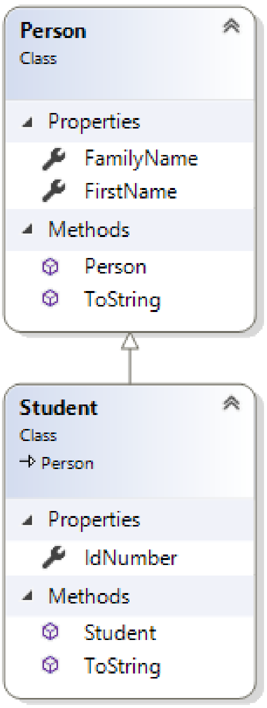
Task 3: Inheritance, abstraction, polymorphism

A. Introductory task

1. Create a new console application, name it *Task3a* and then add a new class representing a *Person* containing *FirstName* and *FamilyName*. Add a constructor to initialise both attributes with new values.
2. Create the *Student* class inheriting from the *Person* class and containing an additional *IdNumber*. Can you compile the project?
3. Add a missing constructor to the *Student* class that initialises all the fields in the class. To assign the fields of the *Person* superclass, use its constructor.
4. Override the *ToString()* method in both created classes so that it returns:
   - *FirstName FamilyName* - for the class *Person*,
   - *FirstName FamilyName* [*IdNumber*] - for the class *Student*.

   In the Student class, use the ToString () method from the superclass. Can this be done using the cast mechanism?
5. In the *Main()* method of the program, create 3 objects as described below:
   - object of the *Person* class assigned to a variable of type *Person*,
   - object of the *Student* class assigned to a variable of type *Person*,
   - object of the *Student* class assigned to a variable of type *Student*.

   Call the *ToString()* method on behalf of each object and write the result of it to the screen.
6. Create an array of *Person* objects and insert all created objects into it. Then in the loop, call the *ToString()* method for each element of the collection, each time displaying the effect of the action on the screen.

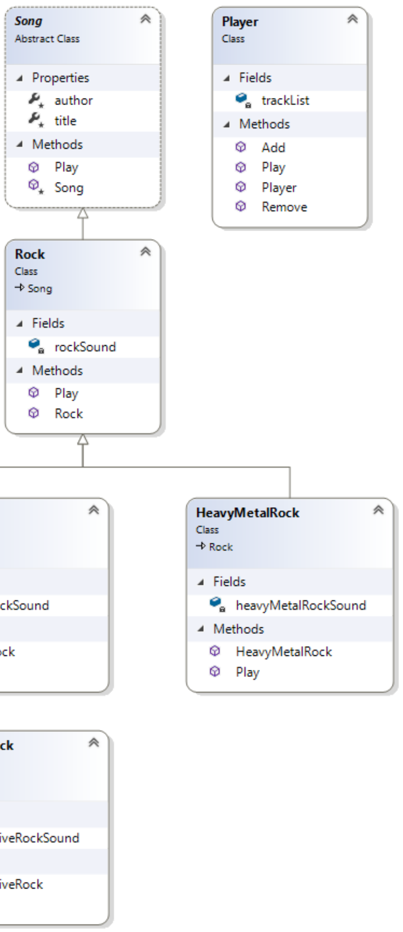


B. Individual task

The goal of the task is to simulate a simple music player. The result of the work is a console application that first asks the user to provide songs that are to be included in the playlist. Then creates a list and plays all the songs in order, removing each song after it is played.

1. You must define at least 5 music genres (classes) that make up the hierarchy of inheritance.
2. Each species has at least one not inherited field that adds some sound to the genre.
3. Each class has an overridden *Play()* method, which displays the title and artist of the song on the screen, as well as the sounds specific to a given musical genre. This method obtains sounds from inherited genres, calling the overridden *Play()* method from the superclass.
4. Each class has exactly one constructor that sets all field values of the created object (inherited fields are to be initialized by calling the superclass constructor).
5. The *Player* class (acting as a music player) contains the *Add(Song song)* and *Remove(int songNumber)* methods, which are used to add and remove songs, respectively. It also has the *Play(int songNumber)* method, which plays the song of the given number from the playlist (by calling the *Play()* function of that song).
6. After start-up, the program in a loop asks the user to provide the song genre, title and artist, then creates a song of the given genre and adds it to the playlist. Each time you enter a song, the application asks you if you want to enter another one.
7. When the user completes entering the songs, the program plays all the songs in the order they were added to the list.

Example – do NOT copy, create your own hierarchy.


Song
Abstract Class
Properties
author
title
Methods
Play
Song
Player
Class
Fields
trackList
Methods
Add
Play
Player
Remove
Rock
Class
→ Song
Fields
rockSound
Methods
Play
Rock
AlternativeRock
Class
→ Rock
Fields
alternativeRockSound
Methods
AlternativeRock
Play
HeavyMetalRock
Class
→ Rock
Fields
heavyMetalRockSound
Methods
HeavyMetalRock
Play
IndieAlternativeRock
Class
→ AlternativeRock
Fields
indieAlternativeRockSound
Methods
IndieAlternativeRock
Play