Assignment: Musicians

Introduction

The goal of this assignment is to get you (re)acquainted with *abstract classes*. Abstract classes were a subject of the introductory programming course (Imperatief Programmeren or Gameprogrammeren) you followed when you arrived at the university as a first year bachelor student (that brings back memories, doesn't it?).

An important reason why abstract classes (and interfaces) exist is that they help structure your code before you actually have to write implementation details. They provide an abstraction mechanism for you to think about what you're going to need. Furthermore, abstract classes and interfaces are very useful for decoupling pieces of code. An abstract class as well as an interface specifies a contract that users of the class/interface have to adhere to (otherwise the compiler complains). A lot of software design revolves around defining the interfaces and structures that your code is going to use and how they depend on each other. In this assignment, we're going to explore how to use these concepts in a practical setting, and how to use them to help you create code that is reusable and easy to understand.

The topic of the assignment is *musicians*. Not only are musicians a good example for using abstract classes, they also lend themselves to countless derogatory jokes, of which you'll see a lot in this assignment. We're going to build an application that creates bands of musicians (as if we don't have enough of those already). Before we start, open the template project in Visual Studio. You'll see an abstract class called `Musician`. It forms the basis of our representation of musicians. In particular, you see an `hourRate` variable. In order to closely reflect reality, musicians are not paid anything by default. Furthermore, each musician can make a particular kind of noise. I've added three subclasses that each represent a musician type: drummers, singers and bass players (contrary to popular belief, bass players are real musicians). The `Band` class has methods that create musicians. We'll extend that class in the following exercises to add more capabilities to the application.

Exercise 1

Extend the `Band` class by adding a method that computes the total cost of the band, given the number of hours that you'd like to book the band. Call that method in the main program to verify that it works correctly.

Exercise 2

The drummer is actually an exception to the simple calculation above (sigh). Drummers need more time to setup their drumkit, so a drummer does not only ask a rate per hour, but also a fixed rate, that you need to pay regardless of how many hours you want to book the drummer. Extend the application such that drummers now also ask a fixed rate.

Exercise 3

Create one or two new types of musicians of your liking, with a predefined hour rate and appropriate method implementations. One of the musician types should have a fixed rate, like the drummer in the previous exercise. Please take note of where you needed to make changes in the code. How did you solve the "drummer problem" in the previous exercise? Did you write your solution in such a way that adding a new musician with a fixed rate was easy? Or did you have to refactor your code?

Exercise 4

Extend the `Band` class so that you can pass an integer value in the constructor and then a band is created containing exactly that number of musicians of randomly determined types. Ensure that the other methods in the `Band` class still work correctly.

Exercise 5 (advanced)

Rewrite the `Band` class in such a way that it is possible to add new types of musicians (for instance a pianist, a pedal steel player, or a nose flute player), but without having to change the `Band` class in the process. *Note: this means that the `Band` class cannot have any knowledge of specific subclasses of `Musician`.*

Exercise 6 (very advanced)

You possibly had to introduce a lot of new classes in the previous exercise to solve that problem. Rewrite your program and use generic classes to reduce the number of classes you need to add.

"Not bad, fellas. Let's do one more take, with more emphasis on tone, harmony, melody, rhythm, composition, lyrics, musicianship, tempo, and originality."