Technical Peer review

Reviewing each other’s code (paired assignment)

In this assignment you are asked to review parts of each other’s code on various aspects that have been covered in OOD.

**What to do:**

1. Your tutor will pair your group up into pairs of two.
2. Together with your tutor you decide what code base you will assess as a pair (code that you did not develop yourself).
3. You answer the questions below before the final meeting in week 15.
4. In the final meeting in week 15 you present/discuss your answers with the tutor and the other pair.

|  |  |  |
| --- | --- | --- |
| **Student name 1** | Nikolay Valchanov | |
| **Student name 2** | Kiril Katsarski | |
| **Assessed code base** | Classes: AutoScheduler.cs, Utils.cs (Password encryptor) | |
| **Date** | 05-Jun-2022 | |
| **Does the target code apply inheritance to generalize their code where applicable?** | | No |
| If not, where do you foresee possible cases for inheritance?  The target code does not apply inheritance as its usage is not really necessary in the reviewed case. Both classes in the codebase are "service" classes in nature, which implies they do not relate to any other classes. | | |
| **Does the target code apply Single responsibility to isolate individual responsibilities?** | | Somewhat |
| If not, what classes would you propose that split up (elaborate about this)?  The AutoScheduler.cs class successfully applies the Single responsibility princpile as its sole purpose is to automatically schedule work shifts. The Utils.cs class doesn't apply the principle as it also has other properties and methods not related to password encryption. | | |
| **Does the target code apply the Open-closed principle to allow extension of behaviour without modification of existing classes in places where change/extension is expected?** | | Somewhat |
| If not, where do you expect change/extension to happen, and how would you propose to facilitate this?  The application can be extended by having a new class inherit from the AutoScheduler.cs class. As for the Utils.cs class, it's marked as static and takes care of more than one thing, which means that it would be difficult and impractical to extend it via new child classes. | | |
| **Does the target code apply the Liskov principle to take benefit of polymorphism?** | | No |
| If not, how can the target code change to communicate in the same way with child objects as you do with parent objects?  The target code does not make use of inheritance and therefore it does not apply the Liskov principle. | | |
| **When applicable, what other object-oriented design principles are applied in the target base (e.g. interface segregation, dependency inversion, etc.)?** | | |
| The reviewed codebase does not make use of any OOP-specific principles. | | |

|  |  |
| --- | --- |
| **Is the target code readable (clear naming convention, conscious use of white spaces, proper tab use (indentation)).** | Yes |
| if not, what could improve?  The code makes use of comments which help the reader understand the code. | |
| **Below you have space for any other tips you want to share with the programmer of your target code?** | |
| For future projects, the programmers can make use of the SOLID principles which will guarantee that their code complies with the established OOP standards. | |