GP 2 | **Generative Programming** | **ST 17, Exercise 7**

Deadline: 07.07.2017, 23:55

☐ **Gr. 1, J. Karder, MSc.**
☐ **Gr. 2, P. Fleck, MSc.**

Name ________________________________ **Effort in h** _____

Points ________________________________ **Lecturer** ______

1. GP Sharper (4 + 4 + 10 + 6 Points)

In this exercise you will implement the next generation of productivity tools for Visual Studio using the .NET Compiler Platform ("Roslyn"). Use the "Analyzer with Code Fix" project template from the .NET Compiler Platform SDK and create separate analyzers and respective code fixes for each task. The C# Language Specification[1] can be very helpful here. The following functionality has to be available:

a) To comply with proper layout conventions, first and second *embedded-statements* of an *if-statement* should always be *blocks*.

```
if (condition)
  A();
else
  B();
```

→

```
if (condition) {
  A();
} else {
  B();
}
```

The analyzer should report violations of this style convention and the appropriate code fix should allow the user to quickly add braces.

b) To ensure that control flow is taken over within a switch statement, it should be possible to add a default label. Take care that the according analyzer must not report switch statements that already contain a default label.

```
switch (option) {
  case Option.A:
    // ...
    break;
  case Option.B:
    // ...
    break;
}
```

→

```
switch (option) {
  case Option.A:
    // ...
    break;
  case Option.B:
    // ...
    break;
  default:
    break;
}
```

c) As already shown in Exercise 3, cloning of objects can be automated nicely. Therefore, implement a code fix that can turn normal classes into deep-cloneable ones by adding

1. the proper base class
2. the cloning constructor containing the cloning logic
3. the clone method

[1] https://msdn.microsoft.com/en-us/library/ms228593.aspx

You can reuse the source code and test cases from Exercise 3. The analyzer should recognize classes that cannot be deep-cloneable, e.g. static classes.

d) While being able to convert classes into deep-cloneables, development does not necessarily stop and properties or fields may be added or removed. Implement an analyzer that recognizes cloning constructors and offers a code fix to regenerate the cloning logic.

Please note: Test all implementations extensively; take care of proper documentation, comments and code structure