HW1 – Linting

236651

Winter 2024-2025

Submission deadline: 10.12.2024

For questions about the exercise,   
contact the TA in Charge:  
[kinsbruner@cs.technion.ac.il](mailto:kinsbruner@cs.technion.ac.il)

In this exercise, you will implement a linter: a developer tool that provides advice to a programmer regarding code they have written. The linter will be written in C# using the [Roslyn framework](https://github.com/dotnet/roslyn) for the C# programming language.

The linter will be comprised of two parts: a syntactic analyzer and a code-fix generator.

**Syntax Analysis**

The syntactic analyzer will analyze code for identifier naming violations according to a policy. The analyzer will only analyze class, method, local variable and global constant names. Use the diagnostic ID: CS236651. **You will write the analyzer as a class that inherits from DiagnosticAnalyzer, in a file named NamingSyntacticAnalyzer.cs**.

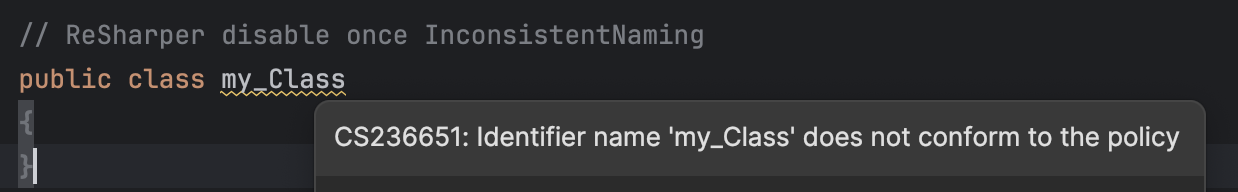
The enforced policy is:

* No identifier shall contain a character that is *not* a lowercase letter, an uppercase letter, a number or an underscore.  
  Hint: Identifiers in C# cannot begin with numbers.
* Class names and method names should be written in UpperCamelCase, that is, multiple words must be concatenated while having the first letter as uppercase and the rest of the letters as lowercase. They must not contain underscores.
* Local variable names should be written in lowerCamelCase, that is, multiple words are concatenated as in UpperCamelCase, except the first word is all lowercase. They must not contain underscores.
* Global constant names should be written in SNAKE\_CASE, that is, multiple words are each in all-uppercase and connected using underscores. They should not contain numbers or lowercase letters.

Note: You do not need to verify the actual words in the name, only that the name fits the allowed shape. For example, MycLass is an allowed class name, and so is MYCLASS (seven one-letter words). Words cannot be “empty”, so \_\_\_A\_ is not a legal constant name.

Note 2: Numbers break words, so Ab5c is not a legal class name, but Ab5C is.

When a violation is found, the linter should issue a warning:

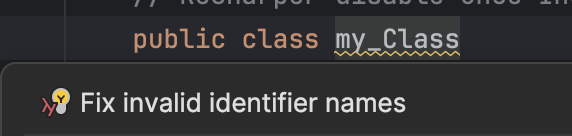


**Extra Challenge**: Try integrating a spell checker – issue a different linter warning for identifier names that are in the correct format but are made up of words that do not actually exist. To assist you, we have uploaded words.txt along with the exercise.  
Hint: Modify your regular expression to use [capturing groups](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Regular_expressions/Capturing_group).  
Let us know in README.md if you solved the challenge!

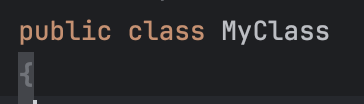
**Code-Fix Generator**

The code-fix generator will suggest automatic corrections to the naming convention. It will only apply whenever the syntactic analyzer has flagged a naming violation. Your code-fix generator can suggest any fix that causes the naming violation to be fixed (yes, any fix). There is no need for “fix all” capabilities. **You will write the analyzer as a class that inherits from CodeFixProvider, in a file named NamingCodeFixGenerator.cs**.

For example, after pressing Alt/Option + Enter on the violating class name:



And after choosing the quick fix:



**Suggestions**

* Express the codestyle properties using regular expressions. You can check your regular expressions using [this website](https://regexr.com).
* C# is similar to Java, and it is well-documented. If you get stuck on a language problem, you should consult an online tutorial.
* You can (and should) look at Roslyn’s [samples](https://github.com/dotnet/roslyn-sdk/tree/main/samples/CSharp/Analyzers).
* Use [JetBrains Rider](https://www.jetbrains.com/rider/) and *not* Visual Studio and the likes. It is a better IDE and it has a Roslyn preset which automatically generates a project skeleton that runs in the IDE itself. To see analyzer warnings in the IDE itself, build the Analyzers project.
* You can use [this website](https://roslynquoter.azurewebsites.net) to explore Roslyn syntax trees.
* You may need to disable Rider’s built-in codestyle checks to test your linter.
* This exercise does not have only one correct solution and it leaves some aspects underspecified.  
  When you make a decision about how to implement underspecified behavior, **write it down in your submission description file**.

**Instructions**

* The exercise will be checked using the latest version of JetBrains Rider.
* You may submit this exercise alone or in pairs.
* You may not use automated code generation tools (e.g., Copilot) while solving the exercise.
* **Submit a README.md file with the names, ID numbers and email addresses of the submitters, as well as any explanations for underspecified behavior described in the previous section.**
* Submit one ZIP file containing all required files in the ZIP file’s root directory. Note that you also have to submit Resources.resx which contains the human-readable text for your diagnostics. Incorrect submission format will result in your submission being rejected by the automated checker.