MSO Lab Assignment 2: Programming Learning App

Jan Huls (4699610), Arwin Moormans (4965957)

October 2025

Software Design & Patterns

Below you will find a Class diagram to show the structure of the application.

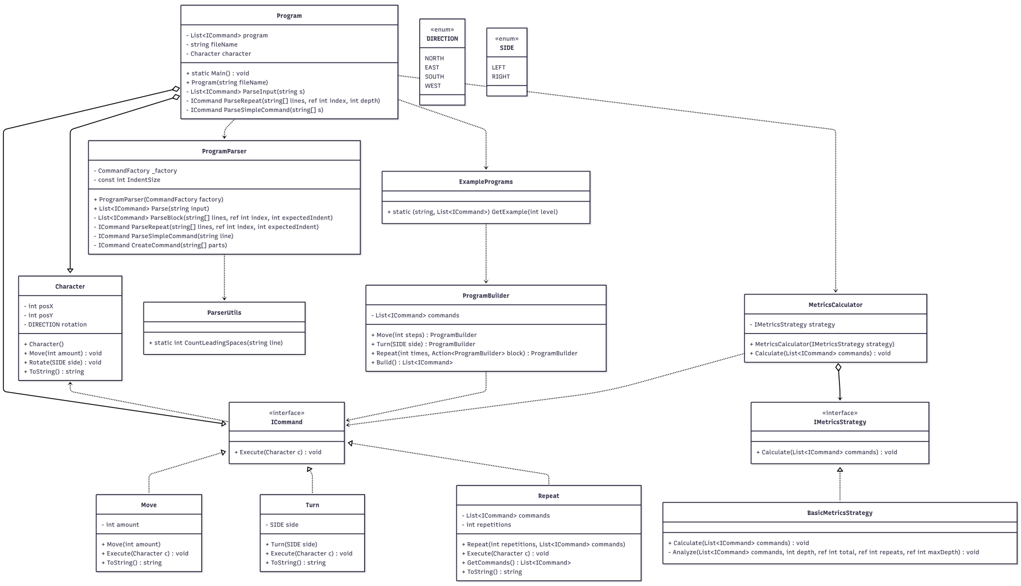
Program
- List<ICommand> program
- string fileName
- Character character
+ static Main() : void
+ Program(string fileName)
- List<ICommand> ParseInput(string s)
- ICommand ParseRepeat(string[] lines, ref int index, int depth)
- ICommand ParseSimpleCommand(string[] s)
«enum»
DIRECTION
NORTH
EAST
SOUTH
WEST
«enum»
SIDE
LEFT
RIGHT
ProgramParser
- CommandFactory _factory
- const int IndentSize
+ ProgramParser(CommandFactory factory)
+ List<ICommand> Parse(string input)
- List<ICommand> ParseBlock(string[] lines, ref int index, int expectedIndent)
- ICommand ParseRepeat(string[] lines, ref int index, int expectedIndent)
- ICommand ParseSimpleCommand(string line)
- ICommand CreateCommand(string[] parts)
ExamplePrograms
+ static (string, List<ICommand>) GetExample(int level)
Character
- int posX
- int posY
- DIRECTION rotation
+ Character()
+ Move(int amount) : void
+ Rotate(SIDE side) : void
+ ToString() : string
ParserUtils
+ static int CountLeadingSpaces(string line)
ProgramBuilder
- List<ICommand> commands
+ Move(int steps) : ProgramBuilder
+ Turn(SIDE side) : ProgramBuilder
+ Repeat(int times, Action<ProgramBuilder> block) : ProgramBuilder
+ Build() : List<ICommand>
MetricsCalculator
- IMetricsStrategy strategy
+ MetricsCalculator(IMetricsStrategy strategy)
+ Calculate(List<ICommand> commands) : void
«interface»
ICommand
+ Execute(Character c) : void
«interface»
IMetricsStrategy
+ Calculate(List<ICommand> commands) : void
Move
- int amount
+ Move(int amount)
+ Execute(Character c) : void
+ ToString() : string
Turn
- SIDE side
+ Turn(SIDE side)
+ Execute(Character c) : void
+ ToString() : string
Repeat
- List<ICommand> commands
- int repetitions
+ Repeat(int repetitions, List<ICommand> commands)
+ Execute(Character c) : void
+ GetCommands() : List<ICommand>
+ ToString() : string
BasicMetricsStrategy
+ Calculate(List<ICommand> commands) : void
- Analyze(List<ICommand> commands, int depth, ref int total, ref int repeats, ref int maxDepth) : void

Figure 1: Class diagram of the Programming Learning App

We have used multiple patterns to structure our program.

The first pattern is the Composite pattern in the repeat command class. This class holds all the functionality of the repeat command, such as the commands it needs to reapeat and how many times. Because one of the repeated commands can be another reapeat command, we use the Composite pattern. This makes a tree like structure where non-repeat commands are leafs and other repeat commands are nodes, meaning `ICommand` is the component and `Repeat` is the composite. This way it is possible to call the Execute function of the root command and that will automaticly execute every command in the tree.

For calculating different metrics we use the Strategy pattern. This ensures that it is possible to add different kind of metrics later. By making this a Strategy pattern it is possible to easily change wich kind of metrics the user wants to see. The `IMetricsStrategy` is the Strategy interface and the `BasicMatricsStrategy` is the concrete Strategy.

At last we have used the Builder pattern to create the example programs. This way it is possible to create different type of complex programs with the same generation code. By using a Builder pattern it is possible to also add more functionality when adding a command to a program. The `ProgramBuilder` is the concrete builder and the `ExamplePrograms` uses the builder to create all the example programs.

Evaluation

Work Distribution & Retrospective