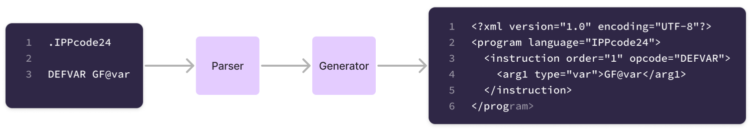
Implementační dokumentace k 1. úloze do IPP 2023/2024

Jméno a příjmení: **Jakub Všetečka**
Login: **xvsete00**

1 Design Philosophy

The script is designed with an emphasis on modular architecture and object-oriented principles. The system comprises various classes like `XMLGenerator`, `Argument`, `Instruction`, `ArgumentParser`, and `InstructionParser`, each serving a distinct role, thereby promoting code reuse and ease of maintenance. The pipeline of the script is visualized in the figure 1.



```
.IPPcode24

DEFVAR GF@var
```

```
<?xml version="1.0" encoding="UTF-8"?>
<program language="IPPcode24">
  <instruction order="1" opcode="DEFVAR">
    <arg1 type="var">GF@var</arg1>
  </instruction>
</program>
```

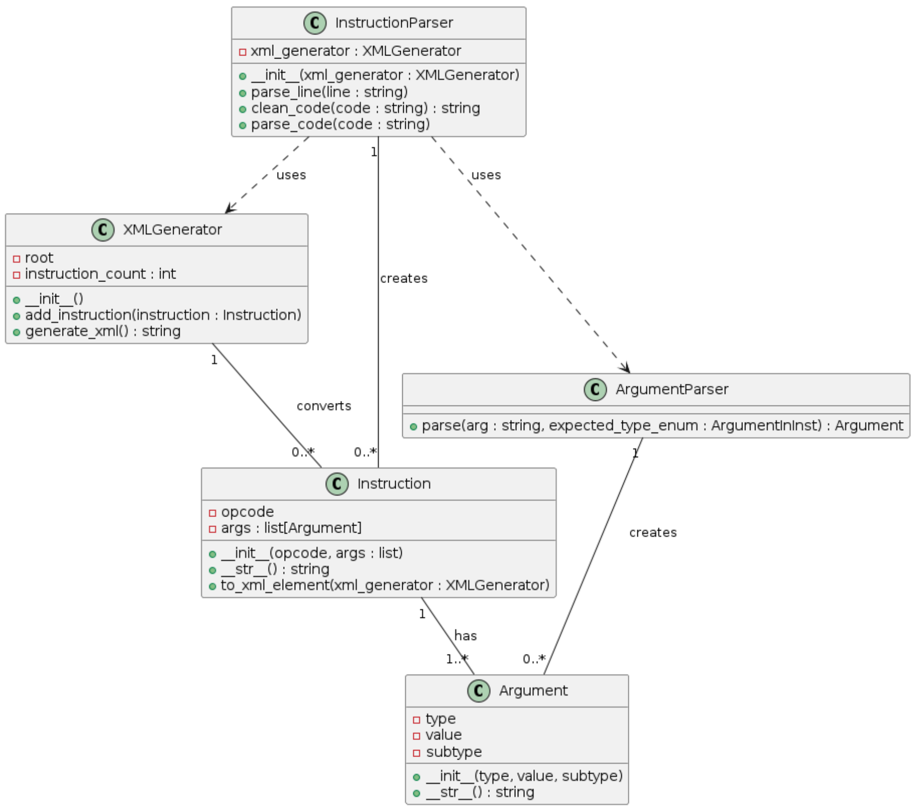
Figure 1: Pipeline

2 Internal Representation

Each script component is represented as an object. `XMLGenerator` handles the XML structure, creating a skeleton for the output. Instructions and their arguments, encapsulated within their respective classes, act as building blocks for the script's functionality. The `ArgumentParser` and `InstructionParser` manage input parsing, transforming text into structured data. Enums ensure consistency in defining argument types and instruction formats, aiding in clear, manageable code. These objects are more closely detailed in the class diagram 2.

3 Solution Procedure

The approach involves parsing the input, building an internal representation, and outputting an XML document. The script begins by cleaning the input using `InstructionParser`, which employs regular expressions to remove extraneous elements like comments and



InstructionParser
xml_generator : XMLGenerator
__init__(xml_generator : XMLGenerator)
parse_line(line : string)
clean_code(code : string) : string
parse_code(code : string)
uses
uses
creates
1
XMLGenerator
root
instruction_count : int
__init__()
add_instruction(instruction : Instruction)
generate_xml() : string
1
converts
0..*
0..*
ArgumentParser
parse(arg : string, expected_type_enum : ArgumentInInst) : Argument
1
creates
Instruction
opcode
args : list[Argument]
__init__(opcode, args : list)
__str__() : string
to_xml_element(xml_generator : XMLGenerator)
1
has
1..*
0..*
Argument
type
value
subtype
__init__(type, value, subtype)
__str__() : string

Figure 2: Class Diagram

unnecessary whitespace. It then leverages enums defined for argument types and instruction formats to facilitate regex-based parsing of instructions, identifying and validating each component of instruction. After parsing, Instruction objects are created, which are subsequently used by `XMLGenerator` to construct the XML output.