Implementační dokumentace k 2. úloze do IPP 2023/2024

Jméno a příjmení: Jakub Pogádl

Login: xpogad00

Introduction

The implementation documentation for the 2nd task of the IPP will detail the development and functionality of an interpreter designed to load XML files and execute instructions specified within them, outputting the results to standard output.

Architecture

The interpreter was developed using static analysis phpstan, passing level 9 with no errors.

Interpret - acts as the entry point for the interpreter
Interpreter – serves as a core class responsible for loading xml file
ProgramFlow – class responsible for controlling flow of the program, loops through every instruction and calls InstructionFactory and executes it, in case of control flow instruction, it can jump to earlier instructions
InstructionFactory – Creates instances of Instruction class based on the opcode
Argument – Stores values and types and returns the value of the correct type
FrameManager – class responsible for managing all frames and stacks, instead of using stack, class works with array instructions
Extensions – classes used to throw exceptions based on
Instruction – abstract class

When class interpret is called, it loads the xml file, stores all instructions into an array and passes them to class ProgramFlow. ProgramFlow loads individual arguments and passes them to InstrsuctionFactory class where correct Instruction is returned. After that, execute method is called on Instruction object. If necessary Instruction object works with FrameManager. During the runtime of program Exception is thrown in, case of an error happening.

Specific Solutions

The interpreter handles edge cases and other incorrect input in the project requirements by throwing exceptions. For example, if a variable does not exist in the frame, a MissingVariableException is thrown. If a variable name is not a string, a WrongTypeException is thrown. Thanks to the entire stack being printed out, debugging is much easier.

Execution

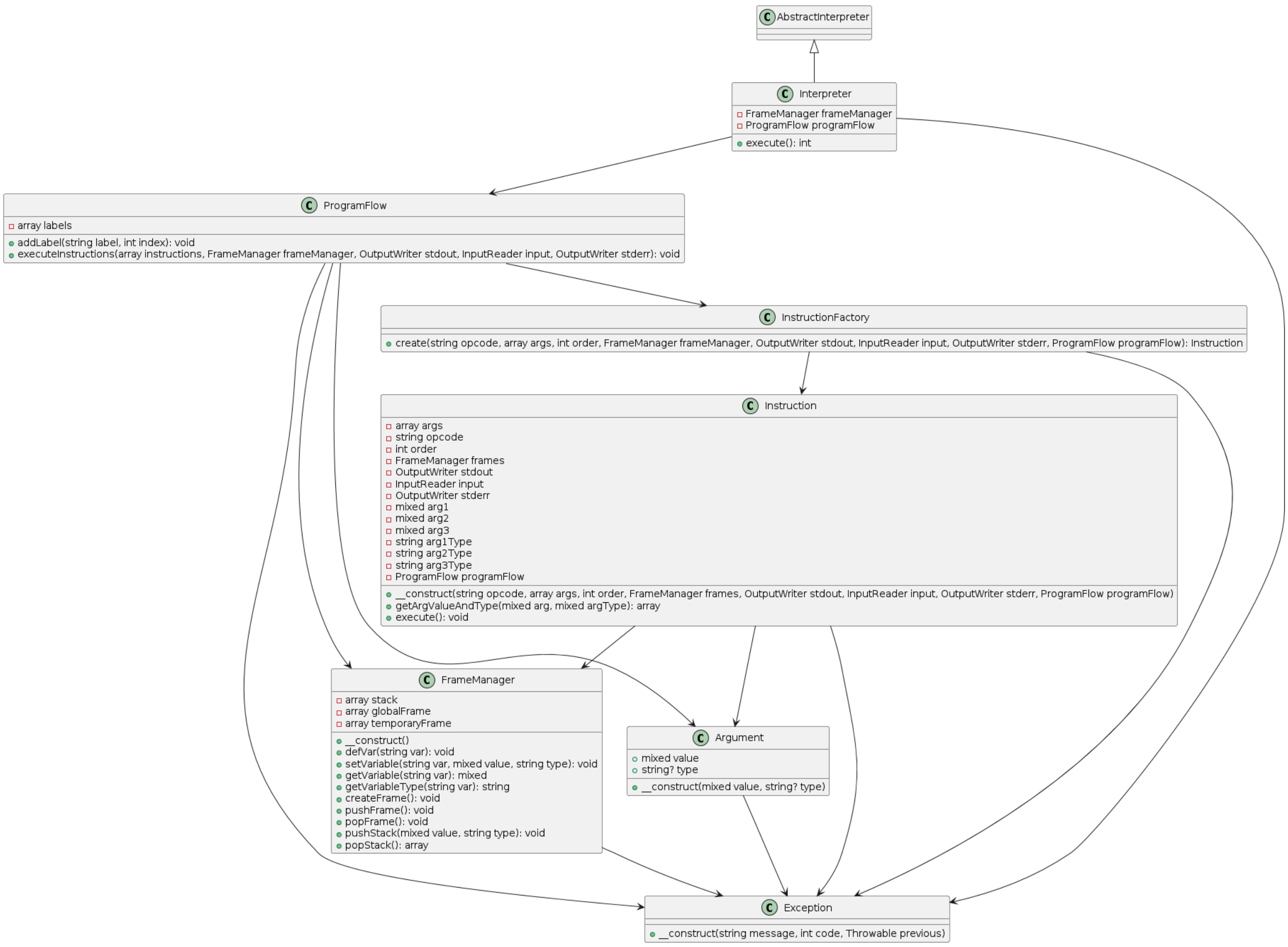
Interpreter can be executed in following way

```
php8.3 interpret.php --source=<source_file> --input=<input>
```

```
php8.3 interpter.php --help
```

Design Patterns

The interpreter uses several design patterns for clean and efficient code. The Factory design pattern is used in the InstructionFactory to create the appropriate subclass of Instruction for each opcode. The Instruction class employs a form of the command pattern for better organization and modularity.



AbstractInterpreter
Interpreter
FrameManager frameManager
ProgramFlow programFlow
execute(): int
ProgramFlow
array labels
addLabel(string label, int index): void
executeInstructions(array instructions, FrameManager frameManager, OutputWriter stdout, InputReader input, OutputWriter stderr): void
InstructionFactory
create(string opcode, array args, int order, FrameManager frameManager, OutputWriter stdout, InputReader input, OutputWriter stderr, ProgramFlow programFlow): Instruction
Instruction
array args
string opcode
int order
FrameManager frames
OutputWriter stdout
InputReader input
OutputWriter stderr
mixed arg1
mixed arg2
mixed arg3
string arg1Type
string arg2Type
string arg3Type
ProgramFlow programFlow
__construct(string opcode, array args, int order, FrameManager frames, OutputWriter stdout, InputReader input, OutputWriter stderr, ProgramFlow programFlow)
getArgValueAndType(mixed arg, mixed argType): array
execute(): void
FrameManager
array stack
array globalFrame
array temporaryFrame
__construct()
defVar(string var): void
setVariable(string var, mixed value, string type): void
getVariable(string var): mixed
getVariableType(string var): string
createFrame(): void
pushFrame(): void
popFrame(): void
pushStack(mixed value, string type): void
popStack(): array
Argument
mixed value
string? type
__construct(mixed value, string? type)
Exception
__construct(string message, int code, Throwable previous)