Implementační dokumentace k 2 úloze do IPP 2023/2024

Jméno a příjmení: Kininbayev Timur

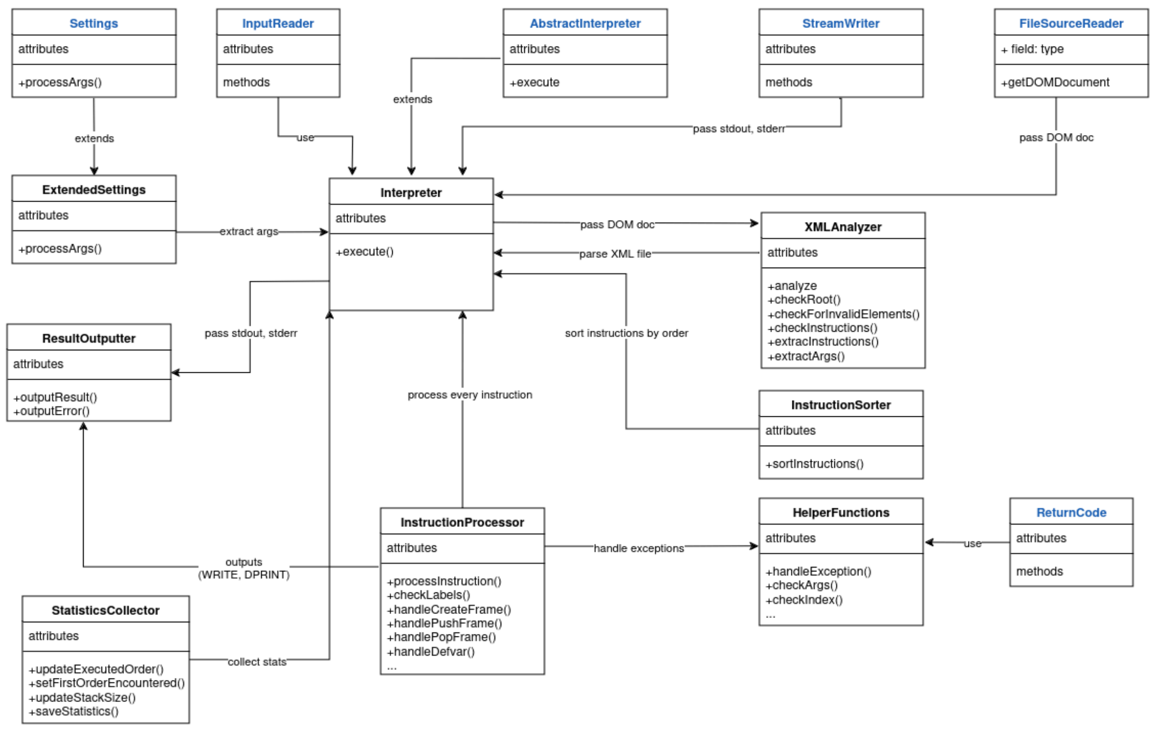
Login: xkinin00

Introduction

This document outlines the implementation of an interpreter for the unstructured imperative language IPPcode24, using PHP 8.3. The interpreter reads an XML representation of IPPcode24 instructions and executes them sequentially, maintaining runtime environments such as variable frames and a data stack.

Structure

The interpreter is designed around several key classes and components that handle different aspects of the execution process. Class diagram below contains main classes and connections, ipp-core classes and interfaces names are blue for clear understanding.



- **`Interpreter` -** The main class that orchestrates the parsing of input XML, instruction execution, and output handling.
- **`XMLAnalyzer` -** Parses the input XML file and validates its structure and content.
- **`InstructionSorter` -** Orders instructions based on their `order` attribute

- **`InstructionProcessor` -** Process individual instructions according to their `opcodes` and `arguments`
- **`ResultOutputter` -** Outputs into selected stream (STDOUT/STDERR) using `StreamWriter`
- **`ExtendedSettings` -** Managing command line arguments for further processing
- **`StatsticsCollector` -** Collects and outputs statistics about the program execution
- **`HelperFunctions` -** Contain various helper functions, mostly used for handling exceptions

Detailed components

Interpreter

- Extends ipp-core `AbstractInterpreter` class
- Responsible for setting up the environment, including source and input readers.
- Utilizes `XMLAnalyzer` to convert XML input into a structured list of instructions.
- Uses `InstructionSorter` to ensure the instructions are processed in the correct order.
- Manages execution flow through `InstructionProcessor`

XML Analyzer

- Ensures the XML structure is correct, checking elements like the root node and language attributes.
- Extracts instructions (opcodes and orders) and their arguments for further processing.
- Instructions are returned as array containing arguments array:

```
foreach ($instructionsNodes as $node) {
            if ($node instanceof \DOMElement) {
                $opcode = $node->getAttribute('opcode');
                $order =
trim($node->getAttribute('order'));
                $args = $this->extractArgs($node);
                $instructionsData[] = [
                    'order' => $order,
                    'opcode' => $opcode,
                    'args' => $args,
                ];
            }
        }
```

Instruction Sorter

- Sorts instructions based on their 'order' attribute to facilitate sequential processing.

Instruction Processor

- Main class containing methods for interpreting every instruction
- Executes instructions based on opcode.

- Manages runtime state, including variable frames (global, local, and temporary) and the data stack.
- Frames are implemented as arrays with <string,mixed> values, key - variable name
- Also contain call stack, labels array and data stack
- Uses `InputReader` and `ResultOutputter` for handling READ, WRITE and DPRINT instructions
- Handle various exceptions via `HelperFunctions` class using `ReturnCode`

Extended Settings

- Extends ipp-core Settings class
- Used for handling –stats arguments (extended functionality)

Result Outputter

- Uses `StreamWriter` for outputting results (WRITE) and errors (DPRINT)

Statistics Collector

- Extended functionality
- Outputs collected statistics to a specified file line by line in the given order (arguments can repeat)
- Collect and handling various metrics about executed program:
  - Total number of instructions executed --insts.
  - Most frequently executed instruction --hot (outputs first order of this instruction)
  - Maximum number of simultaneously initialized variables --vars
  - Maximum stack size --stack
  - Also prints custom message –print=”message” and puts newline char –eol

Execution Process

1. **Initialization**

   1.1. The `Interpreter` initializes settings and prepares the environment based on command-line arguments.

2. **XML Parsing**

   2.1. `XMLAnalyzer` reads and validates the XML file, extracting structured instruction data.

3. **Instruction Sorting**

   3.1. \``InstructionSorter`\` arranges the instructions in the correct execution order.

4. **Execution**

   4.1. \``Interpreter`\` iteratively processes each instruction using `InstructionProcessor`, managing program state and variable scopes.

5. **Statistics Collection**

   5.1. If enabled,\``StatisticsCollector`\` tracks and records execution metrics, writing to a file upon completion.