Implementační dokumentace k 2. úloze do IPP 2023/2024
Jméno a příjmení: Milan Jakubec
Login: xjakub41

1 Návrh

Návrh se snaží zohlednit a rozšířit rámec ipp-core a je možno jej pozorovat v přiloženém diagramu tříd. Některé vztahy, jako například třídy využívající výjimky, byly pro jednoduchost diagramu vynechány. Dále s ohledem na vysoký počet instrukcí a způsob návrhu se pro zjednodušení v třídním diagramu nenachází třída pro každou instrukci či výjimku. Při zanalyzování možností využití návrhového vzoru byl použit pouze návrhový vzor Factory, který je využit pro instanciaci jednotlivých instrukcí.

1.1 Rozdělení zdrojového kódu

Výjimky implementované studentem se nacházejí ve složce exceptions. Třída pro práci s argumenty jednotlivých instrukcí je v souboru Arg.php, implementace datových rámců v souboru Frame.php a třída, která se stará o správu těchto rámců v souboru FrameHandler.php. Jednotlivé instrukce, jejichž reprezentace se nachází v souboru RawInstruction.php, jsou instanciovány v InstructionFactory.php, samotný interpret a jeho metody lze nalézt v souboru Interpreter.php, následně v souboru Stack.php lze nalézt implementaci zásobníku, v souboru Variable.php reprezentaci jednotlivých proměnných, a v souboru XMLValidator.php implementaci pro validování vstupního XML.

1.2 Objektový návrh

1.2.1 Výjimky

Některý typy výjimek byly již implementovány v rámci ipp-core, nicméně to pro účely interpretu nebylo dostačující. Byla proto využita IPPException.php z ipp-core, v kombinaci s abstraktní třídou ReturnCode z tohoto rámce k tvorbě dalších výjimek, reprezentujicích situace, jako např. sémantická chyba, chyba při přístupu k proměnné či rámci aj. Třídy těchto výjimek rozšiřují IPPException a redefinují její konstruktor.

1.2.2 Argumenty

Pro práci s argumenty byla vytvořena třída Arg. Instance této třídy reprezentuje jeden argument vrámci dané instrukce, má dán typ a hodnotu. Kromě toho třída má metodu castValue k nastavení správné hodnoty argumentu dle jeho typu.

1.2.3 Proměnné

Proměnná je reprezentována třídou Variable, každá instance této třídy obsahuje název, typ a hodnotu proměnné. Třída má implementované gettery a settery pro možnosti získání nebo nastavení všech těchto atributů. Kromě toho obsahuje metodu isInitialized pro kontrolu, zda proměnná je inicializována.

1.2.4 Rámce

Jednotlivé instance třídy Frame představují datové rámce. Rámec má pouze jeden atribut - pole proměnných, a dvě metody: getVariable, addVariable. První slouží k získání proměnné z rámce, druhá k přidání proměnné do rámce. Jelikož existují tři typy rámců, existuje třída FrameHandler, která je spravuje a poskytuje rozhraní pro vkládání a hledání proměnných dle typu rámce, společně s možností správý dočasných a lokálních rámců.

1.2.5 Interpret

Interpreter je středobodem celého programu a rozšiřuje AbstractInterpreter z ipp-core. Uchovává v sobě seznam instrukcí a návěští, datový zásobník, zásobník volání a správce rámců a poskytuje rozhraní

jednak pro skákání na návštěstí, jednak ale také pro čtení vstupu a zápisu na standartní výstup či na standartní chybový výstup, k čemuž využívá datové položky své rodičovské třídy.

1.2.6 Instrukce

Implementace instrukcí je řešena abstraktní třídou $\mathrm{RawInstruction}$, která obsahuje abstraktní metodu $\mathrm{execute}()$. Této třídě je předán XML element instrukce, ze které si sama zparsuje a připraví argumenty. Jednotlivé instrukce rozšiřují $\mathrm{RawInstruction}$ a mají odlišné implementace vykonávací metody dle svých činností. Díky tomu je projekt snadno rozšiřitelný, přidání nových instrukcí je velice jednoduché.

1.2.7 Zásobník

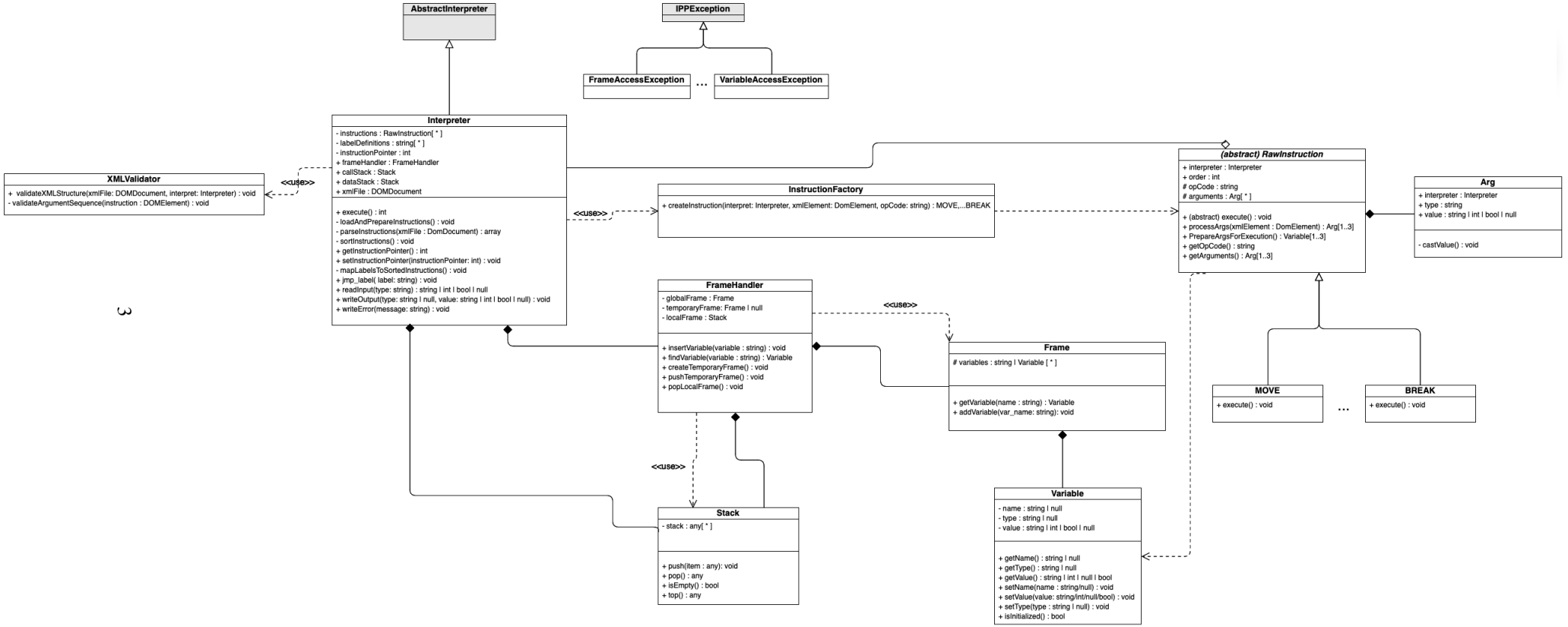
Pro potřeby tohoto projektu byl ve třídě Stack naimplementován zásobník s využitím pole a jednoduchého rozhraní pro vkládání, mazání, zjišťování vrcholku zásobníku a kontrolu, zda je prázdný.

2 Validace XML

Jelikož parser neodchytí nutně úplně všechny chyby, které by ve zdrojovém XML kódu mohly nastat, byl naimplementován jednoduchý validátor XML ve třídě $\mathrm{XMLValidator}$. Tento validátor kontroluje banální, ale možné chyby, zda například ve zdrojovém XML nejsou duplicitní, negativní či jinak nesmyslné ordery, zda se v něm nenachází invalidní elementy, či zda jsou v pořádku sekvence argumentů v jednotlivých elementech instrukcí.

3 Testování

K testování byly použity oficiální příklady a následně studentské komunitní testy, na kterých spolupracovali studenti 2. i 3. ročníku, aby bylo pokryto co nejvíce testovacích případů. V součtu byl projekt otestován zhruba 500 různými testovacími případy.


AbstractInterpreter
IPPException
FrameAccessException
...
VariableAccessException
Interpreter
- instructions : RawInstruction[*]
- labelDefinitions : string[*]
- instructionPointer : int
+ frameHandler : FrameHandler
+ callStack : Stack
+ dataStack : Stack
+ xmlFile : DOMDocument
+ execute() : int
- loadAndPrepareInstructions() : void
- parseInstructions(xmlFile : DomDocument) : array
- sortInstructions() : void
+ getInstructionPointer() : int
+ setInstructionPointer(instructionPointer: int) : void
- mapLabelsToSortedInstructions() : void
+ jmp_label(label: string) : void
+ readInput(type: string) : string | int | bool | null
+ writeOutput(type: string | null, value: string | int | bool | null) : void
+ writeError(message: string) : void
XMLValidator
+ validateXMLStructure(xmlFile: DOMDocument, interpret: Interpreter) : void
- validateArgumentSequence(instruction : DOMElement) : void
<<use>>
<<use>>
InstructionFactory
+ createInstruction(interpret: Interpreter, xmlElement: DomElement, opCode: string) : MOVE,...BREAK
(abstract) RawInstruction
+ interpreter : Interpreter
+ order : int
# opCode : string
# arguments : Arg[*]
+ (abstract) execute() : void
+ processArgs(xmlElement : DomElement) : Arg[1..3]
+ PrepareArgsForExecution() : Variable[1..3]
+ getOpCode() : string
+ getArguments() : Arg[1..3]
Arg
+ interpreter : Interpreter
+ type : string
+ value : string | int | bool | null
- castValue() : void
FrameHandler
- globalFrame : Frame
- temporaryFrame: Frame | null
- localFrame : Stack
+ insertVariable(variable : string) : void
+ findVariable(variable : string) : Variable
+ createTemporaryFrame() : void
+ pushTemporaryFrame() : void
+ popLocalFrame() : void
<<use>>
Frame
variables : string | Variable [*]
+ getVariable(name : string) : Variable
+ addVariable(var_name: string): void
MOVE
+ execute() : void
...
BREAK
+ execute() : void
<<use>>
Stack
- stack : any[*]
+ push(item : any): void
+ pop() : any
+ isEmpty() : bool
+ top() : any
Variable
- name : string | null
- type : string | null
- value : string | int | bool | null
+ getName() : string | null
+ getType() : string | null
+ getValue() : string | int | null | bool
+ setName(name : string/null) : void
+ setValue(value: string/int/null/bool) : void
+ setType(type : string | null) : void
+ isInitialized() : bool