Implementační dokumentace k 2. úloze do IPP 2023/2024

Jméno a příjmení: Václav Chadim

Login: xchadi09

INTERPRET.PHP

Analýza instrukcí

Pomocí rámce *ipp-core* byly zpracovány vstupní data a to rozhraním **SourceReader** a metody **getDOMDocument()** byla načtena vstupní XML reprezentace dat. Data následně byla rozdělena pomocí metody **getElementsByTagName()** do stuktury **DOMNodeList**, která byla pojmenována jako ***instructions***.

Následně se přes tento list iteruje a načítají se jednotlivé instrukce pomocí metod **getAttribute()** a **getElementsByTagName()** a získaná data se ukládají do příslušných struktur, název instrukce do ***opcode***, pořadí do ***order*** a operandy do ***args***, vzhledem k tomu že instrukce v sadě IPPcode24 (bez rozšíření) mají 0 až 3 operandy tak bylo ***args*** implementováno jako pole do kterého se uloží všechny operandy načítané instrukce a následně se celé skupina dat dané instrukce uloží do pole ***instructionArray***, při iterování je také kontrolováno základní uspořádání vstupních dat (validita pořadí ***order***), a jsou mapovány návěští do pole ***labelMap***. (při načítání dat ***opcode*** se vždy načtený řetězec rozšíří o řetězec CLASS, z důvodu překrývání se s klíčovým slovy jazyka PHP)

Načtené instrukce jsou seřazeny pomocí funkce **sortByOrde**r, aby bylo zaručeno že vzniklá sekvence byla seřazená pro další vykonávání programu. Také je definováno pole ***funcMap***, které mapuje instrukce (již rozšířené o řetězec CLASS) IPPcode24 na metody tříd, které se budou volat pro jejich vykonání. Poslední částí je kontrola existence/neexistence návěští pomocí pole ***labelMap*** a pole ***instructionArray***, součástí je i přidání dat ***pos*** do pole ***labelMap***, která označují pozici pro skoky k vykonávání programu.

Vykonávání instrukcí

Jsou vytvořeny potřebné struktury a instance tříd pro vykonávání instrukcí: zásobník volání ***stackCall***, zásobník rámců ***stackFrame*** a datový zásobník ***stackData***, pole všech proměnných a užitečných informací o nich ***VarList***, (jako např místo jejich momentálního výskytu, jejich hodnoty a typy, stav definování, inicializace), dále pole ***GlobalFrame***, ***TemporaryFrame*** a ***LocalFrame***, ve kterých se nacházejí momentálně existující proměnné v daném kontextu. Zároveň jsou definovány Proměnné ***TFaccess*** a ***LFaccess***, které označují možnost přistupování k daným paměťovým rámcům.

Vykonávání jednotlivých instrukcí je prováděno iterováním přes pole ***instructionArray*** s pomocí promněnné ***ip*** označujícím ukazatel na instrukci, tedy při každé iteraci je z

pole ***instructionArray*** extrahován název ***opcode***, který slouží pro název třídy a je použit jako klíč k nalezení třídní metody z pole ***funcMap***, následně jsou danému volání předány všechny parametry. (v rámci usnadnění jsou při všech voláních předávány všechny data což není optimální, ale poskytuje to jistou míru abstrakce, kdy se nemusíme starat o přiřazovaní určitých dat, určitým voláním) Rovněž je předán i ***input***, ***stdou***t a ***stderr***, které odkazují na třídy **StreamWriter** a **FileInputReade*r***, které určité instrukce používají na volání metod k výpisu či načtení dat.

Jednotlivé instrukce si následně použijí vždy potřebnou skupinu dat, např. volaní třídní metody instrukce DEFVAR, použije skupinu dat: ***GlobalFrame***, ***TemporaryFrame***, ***LocalFrame***, ***instructionPointer***, ***instructionArray***, ***VarList***, ***TFaccess***, ***LFaccess*** a ***stackFrame***. (názvy v této části byly rozvedeny pro přehlednost, v programové části zkráceny), volání instrukce může modifikovat daná data.

Rámec ipp-core

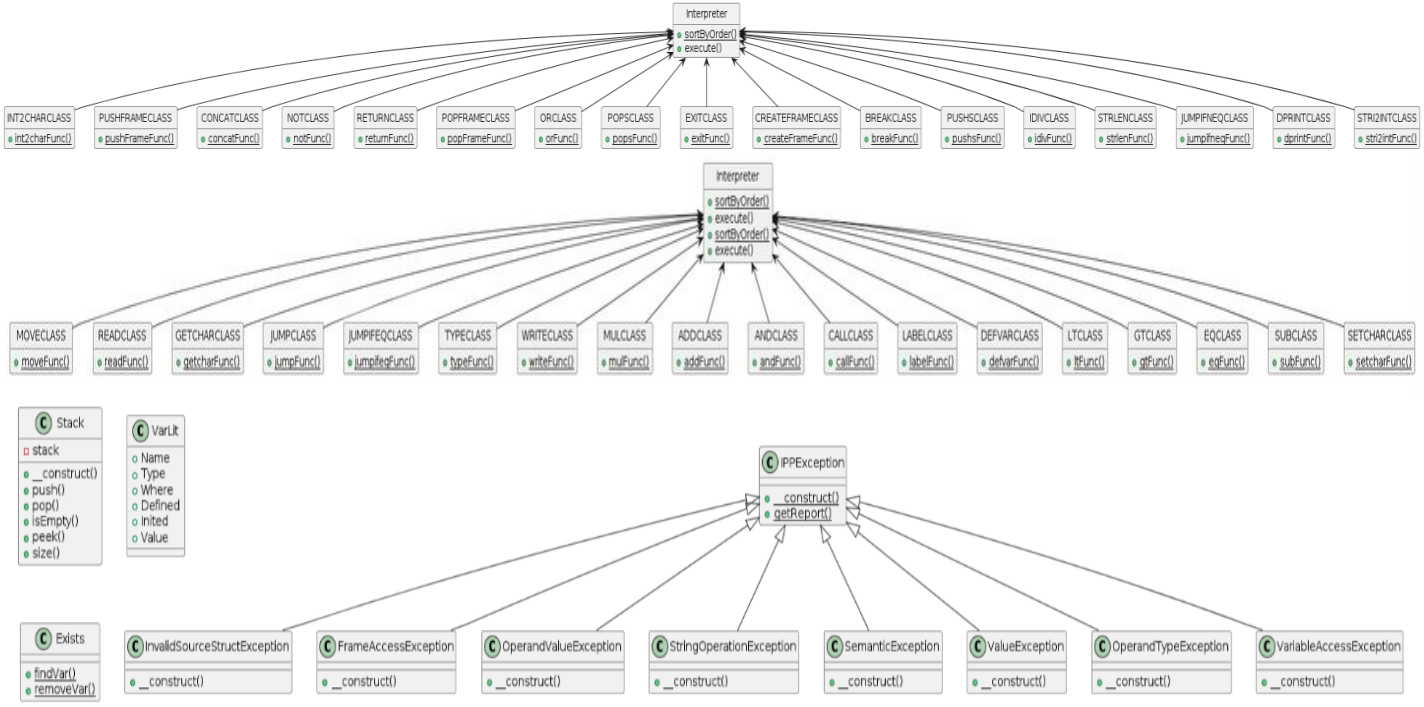
Z rámce ***ipp-core*** byly využity hlavně rozhraní **InputReader**, **SourcerReader**, **OutputWriter** a příslušné třídy **StreamWriter**, **FileSourceReader**, **FileInputReader**. Rozšířena byla třída **IPPException**, pro zajištění správných návratových hodnot a příslušných chybových výpisů.

Návrhový vzor

Ve své implementaci jsem se pokoušel o vzor továrny, kdy jednotlivé instrukce by měli být zpracovány mimo hlavní část (***interpreter***), a klient by pouze volal továrnu a ta by si sama rozhodovala jaké metody zavolá, ovšem k tomu by bylo potřeba ještě rozhraní továrny, které by volalo metody a to bez znalostí klienta o volaných třídách a jejich implementacích.

Zhodnocení a možnost rozšíření

Byla implementována menší část rozšíření ***float***, nicméně jsem nestihl implementovat hlavně aritmetické operace s typem ***float*** a rozšířit rámec *ipp-core*, aby bylo možné tyto operace provádět. V hodné by bylo také abstrahovat analýzu instrukcí a jejich provádění do příslušných instancí tříd a třídních metod. Práce s rámcem *ipp-core* byla poměrně zajímavá a hodnotím ji kladně, nicméně v určitých místech bylo zprovoznění lokálního prostředí poměrně chaotické, kdy část byla v zadání projektu a část v dokumentaci rámce *ipp-core*.


Interpreter
sortByOrder()
execute()
INT2CHARCLASS
int2charFunc()
PUSHFRAMECLASS
pushFrameFunc()
CONCATCLASS
concatFunc()
NOTCLASS
notFunc()
RETURNCLASS
returnFunc()
POPFRAMECLASS
popFrameFunc()
ORCLASS
orFunc()
POPSCLASS
popsFunc()
EXITCLASS
exitFunc()
CREATEFRAMECLASS
createFrameFunc()
BREAKCLASS
breakFunc()
PUSHSCLASS
pushsFunc()
IDIVCLASS
idivFunc()
STRLENCLASS
strlenFunc()
JUMPIFNEQCLASS
jumpIfneqFunc()
DPRINTCLASS
dprintFunc()
STRI2INTCLASS
stri2intFunc()
Interpreter
sortByOrder()
execute()
sortByOrder()
execute()
MOVECLASS
moveFunc()
READCLASS
readFunc()
GETCHARCLASS
getcharFunc()
JUMPCLASS
jumpFunc()
JUMPIFEQCLASS
jumpIfeqFunc()
TYPECLASS
typeFunc()
WRITECLASS
writeFunc()
MULCLASS
mulFunc()
ADDCLASS
addFunc()
ANDCLASS
andFunc()
CALLCLASS
callFunc()
LABELCLASS
labelFunc()
DEFVARCLASS
defvarFunc()
LTCLASS
ltFunc()
GTCLASS
gtFunc()
EQCLASS
eqFunc()
SUBCLASS
subFunc()
SETCHARCLASS
setcharFunc()
Stack
stack
__construct()
push()
pop()
isEmpty()
peek()
size()
VarLit
Name
Type
Where
Defined
Inited
Value
IPPException
__construct()
getReport()
Exists
findVar()
removeVar()
InvalidSourceStructException
__construct()
FrameAccessException
__construct()
OperandValueException
__construct()
StringOperationException
__construct()
SemanticException
__construct()
ValueException
__construct()
OperandTypeException
__construct()
VariableAccessException
__construct()