Implementační dokumentace k 2. úloze do IPP 2022/2023
Jméno a příjmení: Josef Kuchař
Login: xkucha28

1 Rozdělení zdrojového kódu

- `arguments.py` Zpracování argumentů a načtení vstupního programu
- `error.py` Pomocná funkce a výčtový typ pro ukončení programu
- `frame.py` Implementace rámců
- `instruction.py` Implementace bázové třídy instrukce
- `instructions.py` Implementace jednotlivých instrukcí
- `interpret.py` Vstupní bod
- `runner.py` Vykonáváná běh interpretovaného programu
- `stack.py` Implementace zásobníku
- `validate.py` Kontrola správnosti XML
- `variable.py` Reprezentace proměnné

2 Zpracování argumentů a načtení vstupního programu

Tato část interpretu je ve svojí vlastní třídě `Arguments`.

K zpracování argumentů příkazové řádky byla použita knihovna `argparse`[1]. Z důvodů požadavků na návratový kód je vytvořena vlastní třída na načtení argumentů – `Parser`. Třída `Parser` dědí z třídy `ArgumentParser` a nahrazuje implementaci třídní funkce `error`, která se stará o vypisování chybových hlášek v případě chybně zadaných argumentů.

Načítání samotného XML vstupu probíhá pomocí knihovny `etree`[2]. V tuto chvíli se pouze kontroluje zda je XML takzvaně well-formed.

3 Kontrola správnosti XML

Ve vstupním XML není skoro nic garantováno, proto je nejprve vhodné zkontrolovat jeho obsah pro pozdnější práci s jeho obsahem. Nakonec byla použito manuální procházení XML stromem, protože kontrola schémetu přes XSD by stejně nebyla dostatečná.

Tato komponenta dělá velmi podobnou práci jako 1. část projektu. To znamená, že kontroluje, jestli má instrukce správný počet argumentů, typy argumentů a podobně. Stejně tak je kontrolovaná správnost typů a jsou k tomu využity regulární výrazy opět z první části projektu.

4 Objektový návrh

Celkovou představu o objektovém návrhu poskytuje diagram 1. Jednotlivé třídy popsané níže jsou seskupené v logických skupinách do patřičných modulů.

4.1 Proměnná

Proměnné jsou reprezentovaný pomocí třídy `Variable`. Uchovává jméno proměnné, typ a hodnotu. Poskytuje rozhraní pro práci s danou proměnnou.

4.2 Zásobník

Zásobník je reprezentovaný pomocí třídy `Stack`. Tato třída má tři funkce. První funkcí je *zásobník návratových adres*. Druhé využití má jako *zásobník lokálních rámců*. Poslední využití je pro instrukce `PUSHS` a `POPS`. Samotná implementace je vlastně jenom rozhraní nad polem s kontrolou prázdnosti.

[1] https://docs.python.org/3/library/argparse.html
[2] https://docs.python.org/3/library/xml.etree.elementtree.html

4.3 Rámec

K reprezentaci rámce slouží třída `Frame`. Implementace využívá Python `dict`. Poskytuje rozhraní pro vytváření nových a získávání existujících proměnných na rámci s patřičnými kontrolami.

K reprezentaci zásobníku lokálních rámců slouží třída `FrameStack`. K implementaci používá již popsanou třídu `Stack`.

Třída `FrameManager` sjednocuje přístup k proměnným. Bez znalosti umístění proměnné (`LF@`, `GF@` nebo `TF@`) je možné transparentně vytvářet a získávat proměnné pomocí vytvořeného rozhraní.

4.4 Argument

Argumenty jsou reprezentovány pomocí třídy `Argument`. Každý argument má jmeno a typ. V konstruktoru se hned hodnoty převádí na interní reprezentaci (např. řetězec na číslo).

4.5 Instrukce

Každá instrukce (opcode) je reprezentovaná vlastní třídou (`Add`, `CreateFrame`, `Write`, ...). Tyto třídy dědí z bázové třídy `BaseInstruction`. Bázová třída poskytuje pomocnou funkci pro vyhodnocení argumentů.

4.6 Interpretovaný program

K reprezentaci celého interpretovaného programu slouží třída `Runner`. Tato třída uchovává seznam instrukcí, zásobník návratových adres, zásobník a index na aktuálně zpracovávanou instrukci.

Poskytuje rozhraní pro skok na návěstí, které využívají instrukce jako `JUMP`, `CALL` a podobně.

5 Návrhové vzory

Instancování jednotlivých instrukcí je prováděno pomocí *Factory pattern*.

6 Rozšiřitelnost

Žádné rozšíření nebylo implementováno, avšak kód je psaný tak, aby byl velice modulární a rozšiřitelný.

Například pro přidání nové instrukce stačí vytvořit novou třídu dědící z třídy `BaseInstruction` a přidat záznam do seznamu instrukcí. Pomocí této úpravy by šly jednoduše implementovat rozšíření **STACK** a **STATI**.

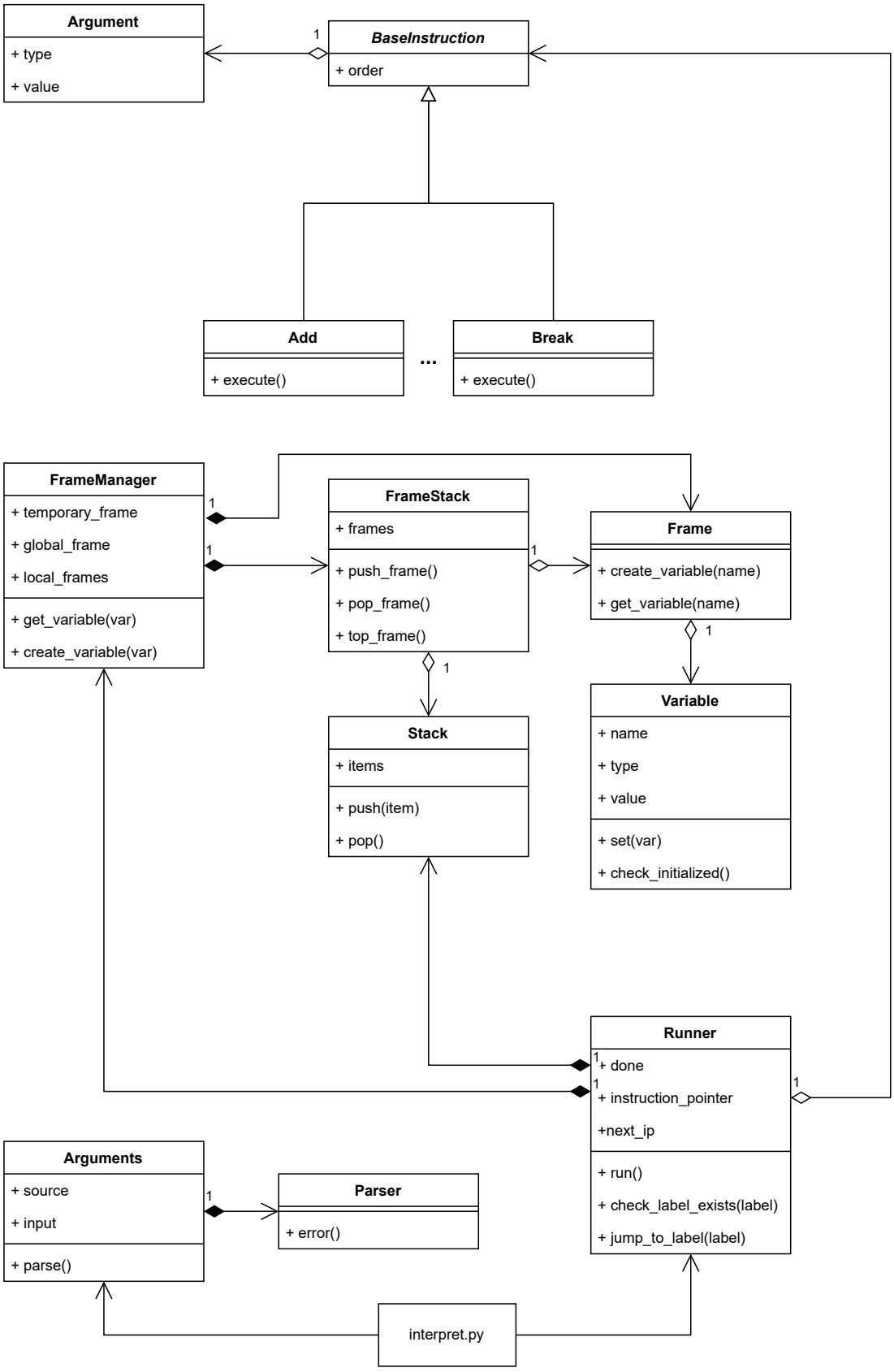
Rozšíření **STATI** by vyžadovalo větší zásah do zdrojového kódu. Bylo by třeba přidat dodatečné interní informace o vykonávání programu. Nicméně bylo by možné toto rozšíření implementovat.

7 Rešení sporných situací

V kapitole zadání 5.4.4 se řeší převod vstupu na bool. Předposlední věta říká, že string „true“ je vyhodnocený jako pravda a zbytek nepravda. Poslední věta říká, že v případně chybějícího vstupu je výsledek nil. Jsou tedy dvě možnosti jak si toto vyložit, byla zvolena druhá varianta, tj. po přečtení chybějícího vstupu je uložen nil. Interpret jazyka IFJ22 z předmětu IFJ se v tomto případě chová obráceně.

8 Testování

Pro ověření funkčnosti implementace byly použity jak studentské testy, tak oficiální příklady.



Obrázek 1: Diagram tříd