Implementační dokumentace k 2. úloze do IPP 2023/2024
Jméno a příjmení: Jakub Valeš
Login: xvales04

Skript `interpret.php` plní funkci interpretu imperativního jazyka IPPcode24, přičemž na vstupu dostane jeho XML reprezentaci, kterou následně interpretuje, a nakonec vrátí odpovídající návratový kód, případný výpis pomocí instrukce `WRITE` probíhá na standardní výstup. Pokud v době interpretace interpret narazí na nějakou chybu vrátí odpovídající (chybový) návratový kód dle specifikace a na standardní chybový výstup vypíše chybovou hlášku.

Implementace programu navazuje na již vytvořený rámec `ipp-core`, který je navržený objektově, a tudíž i moje vlastní implementace se snaží zachovat objektovou orientaci celého systému. Načtení a zpracování XML vstupu probíhá v souboru `Interpreter.php`, ve kterém se nachází implementace metody `execute` třídy Interpret, která dědí z abstraktní třídy `AbstractInterpret`. Jednotlivé instrukce jsou v této metodě rozpoznány a následně uloženy do objektu třídy `instruction`, z těchto objektů je vytvořeno pole, jejímž klíčem jsou hodnoty `order` z instrukce. Ještě je v této metodě vytvořena hashovací tabulka pro návěští a jejich číslo instrukce, kterou využívají instrukce skoku.

Ze výše zmíněné metody je zavolána metoda `start` třídy `program`, která prochází pole objektů `instruction` a pro skoro každou instrukci zavolá metodu `execute()` odpovídající třídy (některé instrukce jsou provedeny přímo v metodě start). V metodě start jsou využity proměnné `call_stack`, což je pole, které slouží k reprezentaci zásobníku volání a pole `stack`, které je datovým zásobníkem. Obě tyto proměnné jsou sice polem, ale zásobník je simulován pomocí PHP vestavěných funkcí `array_unshift()` a `array_shift()`, které vkládají respektive mažou první prvek v poli. Dále je využita instance objektu `frames`, který představuje paměťové rámce programu (globální, lokální, dočasný). Třída `frames`, z které je objekt instanciován, obsahuje atributy `global_frame`, což je pole reprezentující globální rámec, `temporary_frame`, který představuje dočasný rámec, a dvojdimenzionální pole `local_frames`, do kterého jsou uloženy lokální rámce a jelikož je zapotřebí, aby pracoval jako zásobník jsou znovu využity výše zmíněné PHP vestavěné funkce.

Pro uchování informací o proměnných, které jsou následně uloženy do objektu `frames` je využita třída `variable`, která má atributy, do kterých se ukládá název, hodnota a typ proměnné a následně metodu `set_val()`, která objektu `variable`, se kterým byla metoda invokována, nastaví hodnotu a typ.

Jak bylo výše zmíněno skoro pro všechny instrukce programu je vytvořena stejnojmenná třída se statickou metodou `execute()`, do které se posílají argumenty instrukce, objekt `frames`, reprezentující paměťové rámce, a případně nějaké další argumenty (například do metody READ se navíc posílá objekt třídy `InputReader` z `ipp-core`). Za zmínku stojí třída `defvar`, která kromě metody `execute` navíc obsahuje metody pro hledání proměnné v paměťových rámcích `get_variable`() a `find_val()`, a také metodu `get_val_from_symb()`, která využívá tyto metody a slouží k získání hodnoty a typu ze symbolu, který byl zaslán jako argument metody. Dále pro každou výjimku, kterou program může vyvolat, je vytvořena třída představující tuto výjimku, která dědí ze třídy `IPPException` z `ipp-core` rámce.

Na UML diagramu níže jsou vyobrazeny všechny implementované třídy programu. V levé části jsou výše zmíněné výjimky a v pravé dolní části jsou třídy představující instrukce programu se svoji statickou metodou `execute()`, která slouží k provedení instrukce

Obrázek 1: Třídní diagram