Implementační dokumentace k 2. úloze do IPP 2023/2024
Jméno a příjmení: Adam Juliš
Login: xjulis00

Návrh:

Podobně jako u autorovi první zkušenosti s OOP (analyzátor parse.py) se i zde snažil při návrhu použít návrhový vzor příkaz. Třída `Interpret` s metodou `execute` by mohla nejprve načíst vstupní XML soubor po instrukcích a poté (na základě instrukcí ze vstupního XML souboru) volat jednotlivé vnitřní metody pro zpracování těchto metod. Unifikovaný přístup by měl zajistit snadnou rozšiřitelnost. Volání metod pro vykonání instrukcí budou řízeny podobně jako u `parse.py`, tedy názvem zpracovávajících metod dle názvu instrukcí. To pomůže přehlednosti v případě pozdějšího (třeba i výraznějšího) rozšíření jazyka `IPPcode24`.

Dále bude třeba řešit přístup k datům skrz rámce (a tedy i samotnou správu rámců) a datový zásobník. Třída `ManageData` a její objekt vytvořený v třídě `Interpret` by mohla být rozhraním pro správu dat programu, které budou uloženy jak v rámcích (třída `Frames`), tak pro práci s datovým zásobníkem (`Stack`). Přístup by byl jako u návrhového vzoru fasáda, tedy klient (třída `Interpret`) bude zasílat přes vytvořený objekt požadavky (např. přidej, odeber, modifikuj proměnnou) a jednotlivé rozdělení těchto úkonů se bude řešit čistě v `ManageData`.

Zvláštní přístup bude vyžadovat ošetření podmínek. Vytvoří se k tomu soubor `ExceptionsStudent`, který bude obsahovat různé třídy pro jednotlivé výjimky, které mohou nastat.

Realizace návrhu:

Při pokusu implementovat návrh se narazilo na několik problému a odchýlení od zamýšleného návrhového vzoru. Návrh mohl být mnohem detailnější, autorovi přišel ale přístup návrhu po předchozím projektu dostatečný a přímočarý. Změny oproti návrh jsou shrnuty v následujícím popisu skutečně implementovaných tříd.

Implementované řešení:

Třída s výjimkami se po prostudování ipp-core rozdělila na jednotlivé třídy uložené v adresáři `Exception`, inspirované formátem již implementovaných výjimek, kdy každá třída a soubor mají na starost právě jednu výjimku. Názvy tříd odpovídají názvům chyb, které ošetřují.

Požadavky pro práci s proměnnými u zásobníku se ukázaly rozdílné než u rámců, a proto došlo k rozdělení třídy `ManageData` na třídu `Frames` a `DataStack`. Oba se vytvoří před hlavní smyčkou programu (metoda `mainLoop` třídy `Interpret`).

Třída `Frames` slouží jako rozhraní nad jednotlivými rámci třídy `Frame`, umožnuje manipulaci s těmito rámci (obsahuje pro lokální rámce pole objektů `Frame` a pro globální a dočasný rámec samostatné objekty `Frame`) a umí modifikovat a volat dotazy nad daty (proměnnými) uloženými v rámcích.

Třída `Frame` pak implementuje samotné rámce. Tedy obsahuje dvojrozměrné pole, které podle názvu proměnných obsahuje jejich hodnoty a typy. Dále vlastní metody, které nad těmito daty operují (`loadData` pro inicializaci proměnné nebo aktualizaci hodnot, `isExist` pro ověření, zda proměnná v poli existuje nebo metody pro získání jednotlivých proměnných a jejich dat).

Třída `DataStack` pak obsahuje také dvojrozměrné pole (tentokrát si nemusí udržovat jméno proměnné a ukládá se pouze hodnota a typ), s kterým pracuje pomocí zásobníkových instrukcí (`array_pop`, `array_push`).

Abstraktní třída `Regex` vznikla jako jednoduchý způsob, jak si ukládat hodnoty regulárních výrazů pro použití kdekoliv v kódu.

Třída `StringMethod` je pomocná třída implementující specifickou práci s řetězci, která se vyskytovala v různých třídách (například metodu `replaceEscapeSequences` pro nalezení speciální sekvence znaků a nahrazení odpovídajícím znakem z ASCII. Využívá ji třída `Frames` a `Interpret`. Metody jsou statické, aby nebylo třeba vytvářet instance třídy pouze pro toto specifické použití.

Popis běhu programu:

Třída `Interpret` nejprve deklaruje všechny potřebné vlastnosti pro své metody, tedy proměnnou uchovávající číslo aktuálně zpracovávané instrukce, pole pro ukládání návěští a jejich číslo, zásobník čísel instrukce pro operand `CALL`, pole s `DOMElement` pro ukládání jednotlivých instrukcí a potom objekt `Frames` a `DataStack`. `Execute` volaná při startu interpretu spustí hlavní metody pro samotný běh, první kontroluje validnost XML vstupu a obstarává jeho uložení do pole `DOMElement`. Druhá pak spustí hlavní smyčku programu, která postupně prochází pole instrukcí a podle názvu instrukce vytvoří název příslušné metody, kterou zavolá (formát je `processNazevinstrukce`).

Každá metoda má k dispozici celou instrukci a zároveň přístup ke všem vlastnostem třídy, díky tomu je kód jednoduše rozšiřitelný o další instrukce. Jednotlivé metody instrukcí validují a zpracují případné argumenty instrukcí a pak vykonají logiku, která reprezentuje jejich funkci v `IPPcode24`.

Autor se snažil dodržet pravidlo jediné zodpovědnosti pro metodu, někdy metoda kontroluje více věcí, pokud usoudil, že je to vhodné, aby se vyhnul „spaghetti" kódu. Pokud je však logika (například pro validaci argumentů) u více metod stejná, jsou z metod vyjmuty do pomocných metod.

Pokud program bez vyvolání výjimky (a tedy ukončení programu) nebo bez zavolání instrukce `EXIT` (která může program ukončit předčasně) projde celé pole instrukcí, volá v metodě `execute` funkci `return` a úspěšně se ukončí.

Limitace:

Program je náchylný na vstup, který neodpovídá validní syntaktické a lexikální analýze. Kromě chyb v samotném XML interpret neimplementuje kontroly analyzátoru kódu, takže je možné, že některé z těchto testů mohou program destabilizovat. Program se zaměřuje výhradně na sémantickou a běhovou kontrolu.

Uml diagram:

Diagram je pro přehlednost zjednodušen, v třídě `Interpret` nejsou vypsány metody pro zpracování jednotlivých příkazů vstupního jazyka, zastupuje je v diagramu metoda `processAnyNameOfInstruction()`. Stejným způsobem je zjednodušeno i zobrazení tříd pro vyvolání výjimek – všechny třídy v souboru `\Exception` zastřešuje v diagramu třída `AnyException`. Její vztahy zobrazeny nejsou, korelují totiž s druhy volaných výjimek v daných třídách.