Implementační dokumentace k 2. úloze do IPP 2024/2025

Jméno a příjmení: Jakub Lůčný
login: xlucnyj00

April 16, 2025

1 Návrh

Řešení projektu je skript, který provádí interpretaci jazyka SOL25. Je rozděleno do 13ti souborů, každý obsahující definici jedné třídy. Skládá se z 8mi hlavních tříd (Interpreter, Parser, BuiltinMethodBuilder, ClassDefinition, ClassInstance, BlockScope, ExpressionEvaluator a MessageSender) a dalších 5ti tříd reprezentující vlastní výjimky, které dědí od třídy `IPPException`, která je součástí poskytnutého rámce ipp-core.

2 Reprezentace dat

Pro vnitřní reprezentaci dat jsou určeny 2 třídy: `ClassDefinition` a `ClassInstance`. Jak už název napovídá, ta první slouží pro ukládání definic tříd a druhá reprezentuje jednotlivé instance tříd spolu s jejími atributy a interní hodnotou.

2.1 ClassDefinition

Slouží pro reprezentaci definic uživatelem definovaných tříd, ale i vestavěných tříd. Během interpretace vzniká právě jedna instance této třídy pro každou definovanou třídu. Každá tato instance obsahuje definice všech definovaných metod uložených v asociativním poli, indexovaným názvem metody. Dále obsahuje jméno otcovské třídy pro podporu jednoduché třídní dědičnosti, které udává, ve které třídě se má hledat metoda, pokud se nenalezne v aktuální třídě. Třída `ClassDefinition` obsahuje dále také třídní atribut reprezentující seznam všech svých instancí, který se rozšiřuje při každém vzniku nové instance. Seznam je také asociativní, a to podle názvu třídy pro snadné vyhledávání.

2.2 ClassInstance

Všechny data při interpretaci, tedy všechny literály, instance jednotlivých použitých tříd, proměnné, nebo i atributy jednotlivých objektů jsou reprezentovány instancí právě této třídy.
Každá instance obsahuje atribut `$value` pro reprezentaci interní celočíselné, případně řetězcové, hodnoty vestavěných tříd jako jsou `Integer` a `String` nebo tříd od nich odvozených. Dále obsahuje seznam všech svých instančních atributů, které jsou opět uloženy jako instance třídy `ClassInstance` v asociativním poli indexovaným názvem atributu. V neposlední řadě obsahuje také řetězcový atribut udávající, o instanci které třídy se vlastně jedná.
Co se týká implementace instancí tříd `Integer` a `String` se stejnou interní hodnotou, tak interpret pro tyto třídy pokaždé vytváří novou instanci, tudíž žádné 2 instance `Integer`/`String` nemůžou být identické, ale můžou se rovnat.

3 Hlavní tok interpretace

Kromě výše popsaných dvou tříd využitých pro reprezentaci dat využívá interpret ještě dalších 6, které vykonávají interpretaci.

3.1 Interpreter, Parser a BuildinMethodBuilder

Interpretace začíná zavoláním metody `execute()` této třídy. Třída slouží především pro vytváření a volání dalších výkonných objektů. Jako první získá vstupní kód jazyka SOL25, který je ve formátu XML. Poté zavolá metodu `parse()` třídy `Parser` pro získání definic všech uživatelských tříd a jejich metod, následně vytvoří instanci třídy `BuiltinMethodBuilder` a zavolá jednotlivé metody pro vytvoření reprezentací všech vestavěných tříd. Nakonec spustí provádění bezparametrické metody `run` třídy `Main` (pokud je definovaná, jinak program končí s chybou) a předá řízení vytvořené instanci třídy `BlockScope` zavoláním její vstupní metody `processBlock()`. Pokud interpretace proběhne bez chyb, končí skript s návratovým kódem 0.

3.2 BlockScope

Třída `BlockScope` slouží pro zpracování jednotlivých příkazů přiřazení, které se nacházejí v blokuu. Začne tím, že získá všechny argumenty, které byly poslány danému bloku a poté provádí jednotlivé příkazy přiřazení. Jelikož instance této třída slouží také jako jeden rozsah platnosti, tak obsahuje instanční atribut, kterým je seznam reprezentující všechny proměnné definované v bloku a dále atribut s návratovou hodnotu bloku, která se rovná výsledku posledního provedeného přiřazení v bloku.

3.3 ExpressionEvaluator

Pokud instance `BlockScope` při svém zpracovávání narazí na výraz, předá řízení instanci třídy `ExpressionEvaluator`, která si při vzniku uloží, v jakém rozsahu platnosti byla vytvořena. Postupně zpracovává výraz a vrací jeho konečnou hodnotu, reprezentovanou instancí třídy `ClassInstance`, jako výsledek. Pokud je součástí výrazu volání nějaké metody, ať už třídní nebo instanční, tak ExpressionEvaluator vytvoří instanci třídy MessageSender s aktuální referencí pseudo-proměnné `self` a předá této nově vytvořené instanci řízení.

3.4 MessageSender

Třída MessageSender obsahuje dvě hlavní metody, jedna pro invokaci třídní zprávy a druhou pro invokaci instanční zprávy. Pokud se jedná o třídní zprávu, tak zkontroluje její správnost a případně vytvoří novou instanci požadované třídy. V druhém případě se jedná o invokaci instanční metody, a tak se pokusí najít definici požadované metody v dané třídě nebo její nadtřídě. V případě úspěchu se metoda zavolá a předá se řízení nové instanci třídy `BlockScope`. Pokud se požadovaná metoda nepodaří najít, tak se ještě zkontoluje, jestli se nejedná o přístup k instančnímu atributu nebo vytváření nového/přepis stávajícího atributu. Při validním přístupu k atributu se vrátí jeho hodnota. Pří vytváření nového atributu se vytvoří nový atribut s názvem odpovídajícm selektroru zprávy a s hodnotou reprezentovanou instancí `ClassInstance`.

4 Další specifika implementace

V zadání není přesně specifikováno, jak se má chovat vestavěná metoda `asString` pro vestavěné třídy `True` a `False`, pouze že by tato metoda děděná od třídy `Object` měla být redefinována rozumnější implementací, a proto jsem se rozhodl redefinovat tuto metodu pro zmíněné třídy, tak že vrací odpovídající řetězec ,,`true`‘‘/,,`false`‘‘.

Řešení obsahuje také jednu známou chybu, která vznikla kvůli specifické vnitřní reprezentaci pseudo-proměnné `super`, která je reprezentována řetězcem `__SUPER__` jako hodnotou interního atributu instance. A proto, pokud by při interpretaci došlo k uložení řetězce `__SUPER__` do instance vestavěné třídy `String` nebo její podtřídy a to v případě, že by se nejednalo o odkazování na pseudo-proměnnou, tak to vede na nedefinované chování interpretu.

K projektu jsem také vytvořil vlastní sadu testů, která je veřejně dostupná na adrese: `https://github.com/Kubikuli/IPP_proj2-tests`.

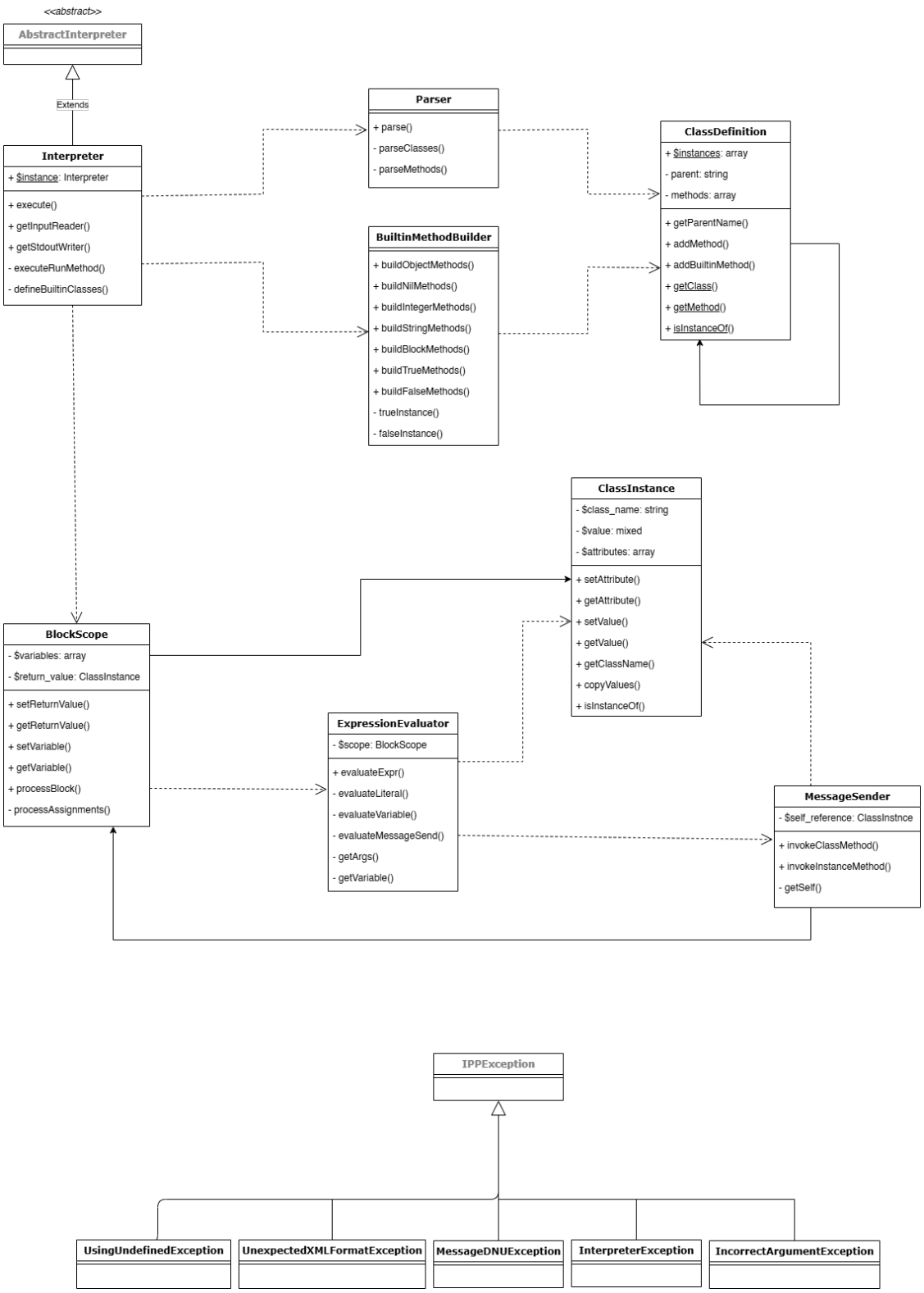

Figure 1: UML diagram tříd ukazující návrh řešení