Implementačná dokumentácia k 2. úlohe do IPP 2019/2020

Meno a priezvisko: Martin Koči
Login: xkocim05

1 Návrh `test.php`

Pri návrhu `test.php` sme sa najskôr zamysleli nad tým čo máme vlastne implementovať. Dohodli sme sa že návrh navrhneme a implementujeme objektovo pomocou návrhového vzoru **Pure Dependency Injection**. Pri návrhu sme najskôr uvažovali že kde a ako budeme ukladať testy. Preto sme navrhli dve triedy `TestCase` a `TestSuite`, kde budeme dáta o testoch ukladať a následne triedu `FileAdministrator`, kde sa budú načítať testy podľa zadaných argumentov skriptu. Pomocou triedy `Tester` môžeme na základe vstupných argumentov skriptu spúšťať jednotlivé `TestSuites`. Ako poslednú vec potrebujeme vypísať testy do `HTML` dokumentu. Pre toto sme vytvorili triedu `HtmlGenerator`, ktorá v sebe má metódy pre generovanie výsledkov testov a triedu `ResultGenerator`, ktorá používa tie metódy a vygeneruje výsledný `HTML` dokument.

1.1 Argumenty `test.php`

Argumenty v `test.php` sme sa rozhodli spracovávať pomocou triedy `ArgParser`, ktorú sme vytvorili špeciálne pre spracovanie argumentov a používame ju aj pre `parser.php` a aj `test.php`. Túto triedu sme sa rozhodli spraviť z toho dôvodu, že väčšina knižníc, ktoré spracovávajú argumenty skriptu, nevracajú výnimky pri zlom argumente skriptu a jednoducho ho ignorujú a taktiež pri opakujúcich sa argumentoch sa stratí ich poradie. Táto trieda má nasledujúce metódy na pridanie argumentov:

```
public function addArgument(string $longOption,
                            bool $expectedValue,
                            $defaultValue = null, array,
                            array $invalidCombinations = array(),
                            array $requiredCombination = array());
```

Táto metóda pridá argument, ktorý má názov `$longOption`, očakávaná hodnota `$expectedValue` predvolenú hodnotu `$defaultValue`, nepovolené `$invalidCombinations` kombinácie a požadované kombinácie `$requiredCombination`.

```
public function addRepeatableArgument(string $longOption,
                                       bool $expectedValue,
                                       $defaultValue = null, array,
                                       array $invalidCombinations = array(),
                                       array $requiredCombination = array());
```

Táto metóda robí presne to isté čo `addArgument(...)` s tým, že pridá danému argumentu možnosť sa opakovať.

```
public function parseArguments();
```

Následne posledná metóda, ktorá spracuje vstupné argumenty skriptu a vráti pole spracovaných argumentov.

1.2 Detailnejší priebeh testovania

Ako prvú vec spracujeme argumenty skriptu pomocou triedy `ArgParser` zavolaním metódy:

```
public function parseArguments();
```

Následne pomocou `FileAdministrator` a na základe spracovaných argumentov vyhľadáme všetky testy pomocou metódy:

```
public function getTestSuites();
```

Pre hľadanie testov v aktuálnom priečinku alebo rekurzívne v podpriečinkoch sme použili `RecursiveDirectoryIterator`. Nájdené testy sa uložia do jednotlivých testovacích prípadov `TestCase` a následne podľa názvu priečinka do testovacích balíkov `TestSuite`.
Následne trieda `Tester` dostane sadu testovacích balíkov a tieto otestujeme pomocou metódy:

```
public function runTests(array $testSuites);
```

Výsledky testov sa uložia do samotných `TestCase` jednotlivých `TestSuite` a tie preberie trieda `ResultGenerator` a pomocou metódy:

```
public function generateResults(array $testSuites); // ich vygeneruje
```

2 Návrh `interpret.py`

Pri návrhu `interpret.py` sme postupovali presne tak isto, ako pri návrhu `test.php`. Ako návrhový vzor sme použili ten istý ako aj v `test.php`. Pri návrhu sme ako prvé uvažovali ako budeme spracovávať vstupný `XML` súbor. Rozhodli sme sa pre vytvorenie triedy `XmlParser`, ktorá spracuje vstupný `XML` súbor a skontroluje ho lexikálne aj syntakticky. Pri tomto procese sme sa zamýšľali nad tým, ako ukladať dané inštrukcie. Tento problém sme vyriešili triedou `Instruction`, ktorá v sebe ukladá jednotlivé dáta o inštrukcii (poradie inštrukcie, operačný kód a jej argumenty) a metódy pre jednotlivé inštrukcie. Argumenty sme sa rozhodli navrhnúť pomocou jednotlivých tried `Label`, `TypeT`, `Variable` a `Constant`. V každej z týchto tried sa pri inicializácii nachádza kontrola pre obsah argumentu (pri chybe sa vyhodí výnimka). K samotnej interpretácii sme sa rozhodli navrhnúť triedu `Program`, ktorá ukladá všetky dáta ktoré potrebujeme počas interpretácie.

2.1 Implementácia `interpret.py`

Pri spracovaní vstupného `XML` súboru sme mali viacej možností na knižnice ktoré môžeme použiť. My sme si vybrali `xml.etree.ElementTree` pretože nám prišla najjednoduchšia a veľmi intuitívna. Na spracovanie argumentov skriptu sme sa rozhodli pracovať s knižnicou `argparse`, ktorá vyhovovala naším požiadavkám, ktoré pri knižnici `getopt` v php neboli uspokojené. S tým rozdielom, že pre zachovanie poradia jednotlivých argumentov pre rozšírenie `STATI` sme tentokrát znova prehľadali vstupné argumenty a podľa toho zapísali štatistiky. Pre kontrolu lexikálnej a syntaktickej správnosti sme používali regulárne výrazy za pomoci knižnice `re`.

2.2 Detailnejší priebeh interpretácie

Na začiatku sa spracujú vstupné argumenty skriptu pomocou:

```
parser_args = parser.parse_args()
```

Skontroluje sa správnosť vstupných argumentov a pristúpi sa k spracovaniu vstupného `XML` súboru pomocou triedy `XMLParser`. Na spracovanie súboru sa zavolá metóda:

```
instructions = xmlParser.parse()
```

Metóda vráti list inštrukcii, ktoré sa predajú cez argumenty metódy triede `Program` pomocou metódy:

```
program.run_program(instructions)
```

Volaním tejto metódy sa zároveň spustí aj interpretácia. V triede `Program` sa na jednotlivých inštrukciách volá metóda:

```
instruction[...].execute()
```

Metódy v triede `Instruction` majú vždy názov podľa operačných kódov inštrukcii. Pomocou vstavanej funkcie `eval()` zavoláme inštrukciu ktorá sa má vykonať. Tu sa môžeme pozastaviť nad funkciou `eval()` či je bezpečná v našom programe, keď dostávame dáta od užívateľa. Môžeme povedať že áno! Dôvod prečo si to môžeme myslieť je ten že pri spracovávaní `XML` súboru dôkladne kontrolujeme vstup a pri neočakávaných vstupoch dávame hneď výnimku. Takto sa vykonajú všetky inštrukcie.

2.3 Rozšírenia

Pre `test.php` sme implementovali rozšírenie **FILES** pridaním argumentov do `ArgParser` a úpravou triedy `FileAdministrator`. Pre načítavanie zo súboru, kontrolu zadaného regulárneho výrazu od užívateľa a jeho použitie na filtráciu testovacích súborov.

Pre `interpret.py` sme implementovali všetky rozšírenia. Rozšírenie **FLOAT** sme implementovali úpravou jednotlivých metód v triede `Instruction` a pri kontrole v `XmlParser`. Rozšírenie **STACK** sme implementovali pridaním metód do triedy `Instruction` a taktiež aj do kontroly v `XmlParser`. A posledné rozšírenie `STATI` sme implementovali pridaním triedy `Stats` (do ktorej sa ukladajú všetky dáta o štatistikách, ktoré získavame počas interpretácie) a samotnou metódou ktorá ich zapisuje do súboru.