Documentație proiect BigNumber

Calitatea Sistemelor Software

1. Componenta Echipei

- Rusu Iulian Daniel
- Simion Andra
- Stativa Camelia Georgiana

2. Scurta Descriere

Acest proiect are ca scop realizarea de calcule aritmetice a expresiilor ce folosesc variabile cu valori definite de utilizator.

Sunt doua modalități de evaluare a expresiilor: de la **tastatura** (din consola) sau dintr-un **fișier .xml**. În ambele moduri, pentru a defini o expresie, exista anumite constrangeri:

- Pentru a realiza operații cu radical, trebuie sa procedam variabilele noastre cu caracterul 'S' (e.g. Sa + Sb = radical din a plus radical din b)
- Pentru ridicarea la putere se folosește semnul '^' (e.g. a^b = a ridicat la puterea b)

3. BigNumber(Implementare de către Daniel, Camelia, Andra)

Aceasta clasa are rol de a reprezenta un număr cu oricate cifre, ce nu poate fi reprezentat pe o structura primitive de date.

Numărul este reprezentat ca o lista de cifre, puse în ordine inversă, pentru a facilita calculele între numere.

Modul de construcție al unui BigNumber are la bază un String ce este parcurs de la ultimul caracter pana la primul și fiecare caracter este transformat în cifra și introdus în lista de cifre. Dacă un caracter nu este cifra, atunci aruncăm o excepție explicativă în acest sens. De asemenea, este necesar ca acest numar primit ca și parametru sa fie pozitiv. În caz contrar, aruncăm o excepție și pentru acest lucru (**NumberFormatException**).

Pentru operații avem următoarele metode:

- **Functia raisedTo** primește ca parametrii doua BigNumber-uri base și exponent și are ca și scop realizarea operației **base**$^{\textbf{exponent}}$. Pentru realizarea acestei operații, tratam în primul rand cateva cazuri particulare:
    - exponent = 0 (caz în care returnam 1)
    - verificam daca exponent și base sunt ambele pozitive

  Pentru realizarea eficienta a acestei operații, am folosit un algoritm de ridicare la putere în timp logaritmic. Pentru mai multe detalii in legatura cu acest algoritm, se poate vizita si urmatorul link: https://en.wikipedia.org/wiki/Exponentiation_by_squaring

- **Funcția squareRoot** primește ca parametru un BigNumber și scopul acestei funcții este de realizare a operației de radical, mai exact partea intreaga a radicalului numărului primit ca și parametru.

  Inainte de a executa operatia, ne asiguram ca numarul primit este pozitiv.

  Pentru realizarea eficienta a acestei operații, am folosit un algoritm de cautare binara pe rezultat al radacinii patrate a numărului primit ca și parametru. Algoritmul funcționează astfel:

    - Fie X numărul primit ca și parametru pentru care trebuie sa găsim radacina patrata
    - Se știe ca rezultatul dorit este între 1 is X. Astfel, încercăm sa vedem dacă mijlocul acestui interval ridicat la puterea a 2 este mai mare sau mai mic sau egal decat X.

    - Dacă rezultatul este mai mare, atunci înseamnă ca trebuie sa cautam răspunsul în intervalul [1, X / 2]. În caz contrar, în intervalul [X / 2, X], iar în caz de egalitate, returnam rezultatul.
    - Procedeul se reia pana intervalul ramane gol sau găsim rezultatul.
- **Funcția add** primește ca parametru un număr BigNumber si-l aduna la valoarea this al obiectului care apelează aceasta metoda. Aceasta functie întoarce suma dintre cele doua valori.
- **Funcția subtract** primește ca parametru un număr BigNumber și îl scade din valoarea this al obiectului care apelează aceasta metoda. Aceasta functie întoarce diferenta dintre cele doua valori. Aici validam dacă rezultatul ramane pozitiv, iar în caz contrar aruncăm excepție.

  Validarea se face cu o metoda privată de comparare a două numere BigNumber, care, în prima instanta, compara cele doua numere sa vada daca au același număr de cifre, iar în caz de egalitate, le compara cifra cu cifra de la cifra cea mai semnificativă la cea mai puțin semnificativă.

- **Funcția multiply** primește ca parametru un număr BigNumber și îl inmulteste cu valoarea this al obiectului care apelează aceasta metoda. Aceasta functie întoarce înmulțirea dintre cele doua valori.
- **Funcția division** primește ca parametru un număr BigNumber și împarte valoarea this al obiectului care apelează aceasta metoda la numărul primit ca și parametru. Aceasta functie întoarce catul împărțirii dintre cele doua valori. Aici validam dacă parametrul primit este pozitiv, în caz contrar aruncăm o excepție deoarece împărțirea la 0 nu este posibila. Totodată, tratam și cazuri particulare, precum:
    - împărțirea unui număr la el însuși (rezultatul fiind 1)
    - împărțirea unui număr mic la un număr mai mare (rezultatul fiind 0)
    - împărțirea la unu (rezultatul fiind 1)
- **Funcție removeTraillingZeros** are ca și scop eliminarea zerourilor redundante de la începutul numărului în urma aplicării operațiilor de mai sus.

4. **Parser(Implementare de către Camelia si Andra)**

Aceasta clasa are rol de a parsa un String primit ca și parametru de la consola de intrare sau dintr-un fișier (în funcție de forma de citire aleasă) și de a o evalua în urma parsarii acesteia. Simultan, aceasta clasa se asigura ca parametrul primit reprezinta o expresie corecta, in caz contrar va arunca o excepție corespunzătoare în funcție de eroarea găsită în cadrul parsarii.

Aceasta clasa are doua metode publice ce pot fi apelate si utilizate, dupa cum urmează:

- **getVariables(String)**: obtine variabilele din expresia ce trebuie parsata si le memoreaza cu ajutorul unei structuri Map pentru a le putea asocia și cu valorile acestora
- **execute(String):** are rolul de a parsa expresia primită ca și parametru, totodată, aplicând validarile necesare pentru a ne asigura ca este corecta din toate punctele de vedere

Exemple de expresii valide:

- a+(b+c)
- a/(b+c)
- a*(b+c)
- a-b*(c+d)
- Sa+Sb (reprezinta radical(a) + radical(b))
- a^b (reprezinta a la puterea b)

Algoritmul de Parsare:

Expresiile aritmetice pot fi scrise în una din trei forme:

- **Notație infixă:** Operatorii sunt scrișI între operanzii pe care aceștia operează, de exemplu 3 + 4.
- **Notație prefix:** Operatorii sunt scrisi înaintea operanzilor, de exemplu + 3 4

- **Notație postfixă:** Operatorii se scriu după operanzi.

Expresiile Infix sunt mai greu de evaluat pentru calculatoare din cauza muncii suplimentare necesare pentru a decide prioritatea. Notația infixă este modul în care expresiile sunt scrise și recunoscute de oameni și, în general, sunt introduse în programe. Având în vedere că sunt mai greu de evaluat, ele sunt în general convertite la una dintre cele două forme rămase. Un algoritm foarte cunoscut pentru conversia unei notații infixe într-o notație postfixă este Shunting Yard Algorithm de Edgar Dijkstra. Acest algoritm ia ca intrare o expresie infixă și produce o coadă care are această expresie convertită în notație postfixă.

Același algoritm poate fi modificat astfel încât să scoată rezultatul evaluării expresiei în loc de o coadă. Trucul este folosirea a două stive în loc de una, una pentru operanzi și una pentru operatori. Algoritmul a fost descris succint pe http://www.cis.upenn.edu/matuszek/cit594-2002/Assignments/5-expressions.html și este reprodus aici.

Pași:

1. Dacă încă mai sunt caractere de citit,
  1.1 Obțineți următorul simbol.
  1.2 Dacă caracterul este:
    1.2.1 Un număr: împingeți-l în stiva de valori.
    1.2.2 O variabilă: obțineți valoarea acesteia și adaugati-o in stiva de valori.
    1.2.3 O paranteză din stânga: împingeți-o pe stiva de operator.
    1.2.4 O paranteză dreaptă:
      1 În timp ce lucrul de deasupra stivei de operator nu este o paranteza stanga,
        1 Scoateți operatorul din stiva de operatori.
        2 Scoateți din stiva de valori de două ori, obținând doi operanzi.
        3 Aplicați operatorul la operanzi, în ordinea corectă.
        4 Împingeți rezultatul pe stiva de valori.
      2 Scoateți paranteza din stânga din stiva operator
    1.2.5 Un operator:
      1 În timp ce stiva de operator nu este goală, iar lucrul de sus pe stiva de operatori are

aceeași prioritate sau mai mare ca operatorul curent,

1 Scoateți operatorul din stiva de operatori.

2 Scoateți din stiva de valori de două ori, obținând doi operanzi.

3 Aplicați operatorul la operanzi, în ordinea corectă.

4 Împingeți rezultatul pe stiva de valori.

2 Împingeți acest operator pe stiva de operator.

2. În timp ce stiva de operator nu este goală,

1 Scoateți operatorul din stiva de operatori.

2 Scoateți din stiva de valori de două ori, obținând doi operanzi.

3 Aplicați operatorul la operanzi, în ordinea corectă.

4 Împingeți rezultatul pe stiva de valori.

3. În acest moment, stiva de operator ar trebui să fie goală, iar în stiva ar trebui să aibă fie o singură valoare, care este rezultatul final.

Complexitatea timp și spațiu pentru acest algoritm este O(N), unde N este numărul de caractere din expresia primită ca și parametru.

5. Package-ul XML(Implementare de către Daniel)

Acest package contine toate clasele necesare parsarii XML-urilor ce conțin expresii ce trebuie evaluate.

Structura unui fișier XML corect pentru programul nostru este următoarea:

```
<expression>
   ….
</expression>
<variable>
<name>variabila1</name>
<value>valoare1</value>
</variable>
<variable>
<name>variabila2</name>
<value>valoare2</value>
</variable>
……
```

Un exemplu este:

```
<expression>
   (a+b)
</expression>
<variable>
<name>a</name>
<value>2</value>
</variable>
<variable>
<name>b</name>
<value>3</value>
</variable>
```

Pentru a realiza acest obiectiv de citire din fișierele XML și parsarea și executarea expresiilor acestora, avem de executat următorii pași:

- **Citirea datelor**

  Pentru citirea datelor din fișiere, avem clasa **XMLFileReader**. Aceasta clasa are o metoda de read și singura responsabilitate pe care aceasta clasa o are este de a citi toate datele dintr-un fișier XML și a le converti într-un String fără spații și fără caractere new line.

- **Validarea datelor**

  Pentru a valida datele primite de la **XMLFileReader**, avem clasa **XMLValidator** ce, pentru un String primit ca și parametru, se asigura ca:

  - toate tag-urile deschise sunt închise
  - toate tag-urile obligatorii (e.g. <expression>) exista
  - nu exista variabile fără valori asociate

- După ce Stringul primit este validat, este pasat mai departe către **XMLParser**. Aceasta clasa are ca și scop sa extraga toate datele necesare din cadrul Stringului primit de la **XMLFileReader** și validat de **XMLValidator** în expresia primită și valorile asociate, memorate într-un Map. Ce face mai exact este sa ia tag cu tag, de la primul pana la ultimul și sa extraga conținutului tag-ului respectiv.

6. **Rulare program**

Pentru rularea programului, se va executa **Main.java.**

După apelarea acest program, ne va apărea următorul mesaj:

Please enter the input mode: 1(console), 2(from file) or 3(exit)

Care ne indica exact ce comenzi avem la dispoziție. Astfel, trebuie sa introducem unul din cele 3 numere de mai sus (1, 2 sau 3) în funcție de necesitate.

Dacă introducem 1, vom fi rugați sa introducem o expresie. După introducerea expresiei, dacă aceasta este validă, vom fi rugați sa specificam valorile variabilelor prezente în expresia anterioara. Avand valorile și expresia, putem face parsarea și evaluarea acesteia.

Dacă introducem 2, vom fi rugați sa introducem doua lucruri:

- fișierul de intrare XML din care programul urmează să-și citească datele (expresia, variabilele și valorile acestora)
- fișierul de ieșire (poate fi de orice tip) în care programul va afișa rezultatul expresiei din fișierul de intrare introdus anterior

Dacă introducem 3, programul se va închide.

7. **Assertions(implementat de către Daniel, Andra și Camelia)**

Am folosit assert-urile pentru a împiedica user-ul sa introducă date greșite, dar si pentru a valida preconditiile, postconditiile si invariantii.

- **BigNumber**
  - *Constructor*:
    - Ne asiguram ca numarul nu este negativ
    - Ne asiguram ca numarul nu are caractere diferite de cifre
    - Și ca nu primim un String gol ca și parametru
  - *raisedTo*:
    - Verificam ca base și exponent sunt pozitive sau egale cu 0
    - Verificam ca rezultatul este corect pentru anumite cazuri particulare pentru care știm deja răspunsul
  - *add*:
    - Verificam ca numărul cu care adunăm sa fie pozitiv

        - În timpul adunării cifrelor, ne asigurăm ca valoarea de transport nu depășește 1 și nici nu devine negativă.
        - Ne asigurăm în timpul parcurgerii cifrelor ca nu accesam vreun index inexistent.
        - La final, ne asigurăm ca rezultatul este mai mare decat cele doua numere.
    - *subtract*
        - Ne asiguram ca numarul pe care il scadem din *this* nu este mai mare ca și acesta
        - Ne asiguram ca numarul primit nu este negativ
        - În timpul scaderii cifrelor, ne asigurăm că valoarea de transport nu depășește 1 și nici nu devine negativă.
        - Ne asigurăm în timpul parcurgerii cifrelor ca nu accesam vreun index inexistent.
    - *multiply*
        - Ne asiguram ca numarul primit nu este negativ
        - Ne asigurăm în timpul parcurgerii cifrelor ca nu accesam vreun index inexistent.
        - Ne asigurăm ca rezultatul are un număr de cifre mai mic sau egal cu suma numărului de cifre ale celor doi operanzi
        - Ne asigurăm ca rezultatul este compus doar din cifre
        - Ne asigurăm ca rezultatul este mai mare decat cele doua numere
    - *division*
        - Ne asiguram ca numarul primit nu este negativ sau zero
        - Ne asigurăm în timpul parcurgerii cifrelor ca nu accesam vreun index inexistent.

8. Testare

Clasa care testează functionalitatile operațiilor este **BigNumberShould (implementat de către Andra, Georgiana și Daniel)**

Cazuri de test pe care le-am testat și întâlnit:

- Adunarea a două numere → rezultatul ar trebui sa fie suma celor doua numere și să se execute fără erori

- Adunarea unui număr pozitiv cu unul negativ → rezultatul ar trebui sa fie o excepție aruncată de program
- Adunarea a două numere mai mari → rezultatul sa fie intr-adevar corect
- Scaderea a doua numere → rezultatul sa fie cel corect
- Scaderea unui număr mare dintr-un număr mai mic → rezultatul ar trebui sa fie o excepție aruncată de program
- Înmulțirea a două numere → rezultatul ar trebui sa fie cel corect
- Înmulțirea unui număr pozitiv cu un număr negativ → rezultatul ar trebui sa fie o excepție aruncată de program
- Împărțirea unui număr la 0 → rezultatul ar trebui sa fie o excepție aruncată de program
- Împărțirea unui număr la el însuși → rezultatul ar trebui sa fie 1
- Împărțirea unui număr la 1 → rezultatul ar trebui sa fie numărul însuși
- Împărțirea unui număr la ceva mai mare → rezultatul ar trebui sa fie 0
- Ridicarea la putere a doua numere → rezultatul ar trebui sa fie cel corect
- Ridicarea la putere a unui număr la 0 → rezultatul ar trebui sa fie 1
- Ridicarea la putere a unui număr la 1 → rezultatul ar trebui sa fie numărul însuși
- Ridicarea la putere a unui numar la o putere negativa → rezultatul ar trebui sa fie o excepție aruncată de program
- Radical dintr-un număr negativ → rezultatul ar trebui sa fie o excepție aruncată de program
- Radical dintr-un număr pătrat perfect → rezultatul ar trebui sa fie chiar radacina patrata
- Radical dintr-un numar nepatrat perfect → rezultatul ar trebui sa fie partea intreaga a rădăcinii

Clasa care testează functionalitatea **parsarii** este **ParserTest**: **(implementat de către Georgiana)**

- Cand expresia este corecta → parcarea ar trebui sa extraga in regula variabilele
- Cand expresia contine variabile ce nu exista → rezultatul ar trebui sa fie o excepție aruncată de program

- Cand expresia nu da valori pentru toate variabilele → rezultatul ar trebui sa fie o excepție aruncată de program
- Cand expresia nu are parantezele închise corespunzător → rezultatul ar trebui sa fie o excepție aruncată de program
- Cand expresia are alte erori de sintaxa → rezultatul ar trebui sa fie o excepție aruncată de program

Clasele care testează functionalitatea de citire și extragere a datelor din XML sunt **XMLReaderTest**, **XMLParserTest**, **XMLValidatorTest (implementat de către Andra si Daniel)**

- Cand XML-ul nu este formatat corespunzător → rezultatul ar trebui sa fie o excepție aruncată de program
- Cand XML-ul nu își închide tag-urile → rezultatul ar trebui sa fie o excepție aruncată de program
- Cand XML-ul nu are valorile pentru toate variabilele → rezultatul ar trebui sa fie o excepție aruncată de program
- Cand XML-ul are variabile în plus → rezultatul ar trebui sa fie o excepție aruncată de program