Calculator ismertető

Tartalom

1.0 A program felépítése

A „Calculator” rendeltetése szerint demonstrátor. A program Java programnyelven íródott és a Spring keretrendszert használja, a Maven projektépítő rendszer alatt. A fő forráskód két egymásba ágyazott csomagban található, ezek:

com.imaginifer.calculator – Ez tartalmazza a *Calculator* alkalmazásindító „main” osztályt, a végpontokat tartalmazó *CalculatorController* osztályt és a *CalcDTO* adathordozó objektum osztályát.

com.imaginifer.calculator.service – Ebben található a tényleges munkavégző logikát tartalmazó CalculatorService osztály, a műveleteket segítő Operation tárolóosztály és az *Operator* enumerátor osztálya.

Rajtuk kívül két másik, a *com.imaginifer.calculatortests* és a *com.imaginifer.calculator.servicetests* tesztcsomag is található, előbbiben a *CalculatorController* osztályt ill. a végpontjait vizsgálni hivatott integrációs tesztek, utóbbiban a *CalculatorService* logikáját ellenőrző egységtesztek vannak, egy absztrakt osztály leszármazottaiban. Az alkalmazáshoz nem tartozik sem felhasználói felület, sem adatbázis.

1.1 CalculatorController felépítése

A *CalculatorController* két végpontot tartalmaz, címük a „/calculator” gyökér-url alá rendelt „/calculate” és „/fibonacci”. Előbbi a *requestToCalculate*, utóbbi pedig a *requestToFindFibonacciNr* függvényhez tartozik, mindkettő „post” kéréssel elérhető. Mindkettő egy egyetlen „txt” nevű karakterlánc mezőt tartalmazó *CalcDTO* objektumot fogad és ugyanilyet is küld vissza feldolgozás után, amit a Spring Boot JSON formában továbbít. Az osztályban példányosítja a *CalculatorService* „beanjét” a Spring.

1.2 CalculatorService felépítése

A *CalculatorService* osztály két kívülről elérhető függvényt tartalmaz. A *calculateFibonacciNr* nevű, egy karakterláncot fogadó és kiadó függvény meghívja a *getFibonacciNr* metódust. A *calculate* nyilvános függvénynek szintén egyetlen karakterlánc a bemenete és a kimenete, ez az

identifyOperations, *executeOperations* és *formatResponseNumber* metódusokat hívja meg amik pedig az *identifyIllegalOperations* és *setOperationsPrecedence* függvényeket hívják. Az osztályban található még az egész szám kulccsal lebegőpontos számokat tároló *operandi* adatstruktúra, valamint az Operation objektumokat tároló *operations* tömblista.

1.3 Operation felépítése

Ez a tárolóosztály két egész szám és egy Operator enumerátor mezőt tartalmaz, reprezentálandó egy számtani művelet bal és jobb oldalát, valamint a műveleti jelet. Az *updateOperation* metódusa ha az első két paraméterének bármelyikét azonosítja az osztály mezőiben, azt lecseréli a harmadikra.

1.4 Operator felépítése

Ebben az osztályban találhatók a műveleteket jelölő enumerátorok, illetve az ezeket műveleti jelekkel megfeleltető *OPERATORS* statikusan feltöltött hashmap.

2.0 A program működése

Az alkalmazásnak két fő funkciója van: aritmetikai alapműveletek – tehát összeadás, kivonás, szorzás és osztás – elvégzésére képes számológépként való működés, valamint egy megadott sorrendű Fibonacci-szám kiszámítása.

2.1 Fibonacci-sorozat számító működése

A „/calculator/fibonacci” végponton a *CalculatorController* osztály *requestToFindFibonacciNr* függvénye a kérés JSON objektumát *CalcDTO* formában fogadja, ennek *txt* mezőjét továbbadja a *CalculatorService* osztály *calculateFibonacciNr* metódusának. Ez kísérletet tesz a bemenet karakterlánc *nr* egész számmá való alakítására, hiba esetén „Input error!” üzenetet küld vissza. Siker esetén ha az *nr* 0 vagy 1, a függvény azonnal vissza is küldi a bemeneti paramétert, ha viszont negatív, az „Input error!” üzenetet. Ha egyik sem, a szám a meghívott *getFibonacciNr* függvény paramétere lesz.

Ebben négy egész szám változó van: a 0-ra és 1-re inicializált *first* és *second*, a keresett számot tárolni hivatott *number* és a *q* ciklusköveő. Ezután egy ciklusba lép, ami addig tart, amíg *q* kisebb, mint *n*. Ebben *number* értékül kapja *first* és *second* összegét, előbbi az utóbbiét, utóbbi pedig *number* értékét, *q* pedig inkrementálódik. A ciklusok után a függvény visszatér *numberrel* a meghívó *calculateFibonacciNr* metódusba, ami karakterlánccá alakítva értékül adja a visszaküldendő változónak, majd szintén visszaküldi, a kontrollerbe. A *requestToFindFibonacciNr* függvény egy új *CalcDTO* objektum konstruktorbemenetébe helyezi, majd a Spring Boot JSON objektummá alakítva visszaküldi a kliens felé.

2.2 Számológép működése

A „/calculator/calculate” végponton a *CalculatorController* osztály *requestToCalculate* függvénye a kérés JSON objektumát *CalcDTO* formában fogadja, ennek *txt* mezőjét továbbadja a *CalculatorService* osztály *calculate* metódusának.

Ez először is kiüríti az *operandi* és *operations* adatstruktúrákat korábbi számítások maradványaitól, majd továbbítja a karakterláncot az *identifyOperations* függvénynek.
Itt először a karakterlánc szóközökkel elemeire bontódik. A keletkezett tömböt egy for-ciklus végigiterálja, megkísérelve elemeit lebegőpontos számokként azonosítani. Siker esetén – ha az előző indexen nem tárol számot az *operandi*, vagyis nem hiányzik egy műveleti jel, mert akkor kivétellel leáll – a ciklus indexét kulcsként használva tárolja az *operandiban* és továbblép, kudarc

esetén ellenőrzi, hogy az adott karakterlánc szerepel-e az *OPERATORS* kulcsai között. Ha igen, és nem a tömb utolsó vagy első indexén áll, akkor az előző és következő indexekkel, mint *operandi* azonosítókkal létrehoz egy *Operation* objektumot az *operationsben*, a megfelelő műveleti enumerátorral. Ha nem, tovább dobja a kivételt.

A ciklus után vizsgálja, hogy üres maradt-e az *operations*, és ha igen, kivételt dob. Ezután meghívódik az *identifyIllegalOperations* függvény, aminek rendeltetése, hogy kiszűrje a nullával osztást a műveletek közül, amit lebegőpontos számokkal a Java készségesen elvégezne, végtelen eredménnyel.

Ezután a program továbblép a *calculate* metódusban az *executeOperations* meghívására. Ez elsőként a *setOperationsPrecedence* függvénnyel műveleti sorrend szerint rendezi az *operations* listát, egy alacsony precedenciájú műveleteket tartalmazó halmazzal és egy anonim komparátorral. Ezután a sorrendbe tett műveleteket rendben végrehajtja. A ciklus elején a *newKey* változóba lementi az operandi utolsó azonosító kulcsa utáni számot, és deklarál egy *newOperand* változót. Ezután egy *Operator* alapú switch-case útválasztó elvégzi a megfelelő aritmetikai műveletet, eredményét lementi a *newOperandba*, amit a *newKey* azonosítóval beszúr az *operandi* végére.

A ciklus végén a soron következő *Operation* objektumok között ellenőrzi, melyikben szerepel-e az épp elvégzett művelet változó-azonosítói közül bármelyik, ennek adott bal- vagy jobb oldali változójának azonosítóját lecseréli a *newKey*-re.

Az *operations* lista végeztével az *operandi* utolsó kulcsa alatti végeredmény a *formatResponseNumber* függvény által három tizedesjegy pontosságú karakterlánccá alakul, majd visszakerül a calculate metóduson át a kontrollerbe. A *requestToCalculate* függvény egy új *CalcDTO* objektum konstruktorbemenetébe helyezi, majd a Spring Boot JSON objektummá alakítva visszaküldi a kliens felé.

3.0 A program használata

A Calculator nem rendelkezik se grafikus, se szöveges felhasználói felülettel, ami használatát megnehezíti. Postman alkalmazással, kézzel összeállított JSON objektumok által lehetséges a végpontok elérése, de körülményes, így a program tényleges működésébe leginkább az automatizált egységtesztek adnak betekintést.

Az integrációs tesztek vizsgálták a *CalculatorController* inicializálódását, a végpontok válaszait és a róluk visszaküldött JSON objektumokat.

3.1 Fibonacci-sorozat számító használata

Karakterláncként megadott egész számra válaszul szintén karakterláncként megadja az adott sorszámú Fibonacci-számot. Üres, null, negatív vagy nem egész szám bemenetre „Input error!” üzenettel válaszol.

Vizsgált tesztesetek:
null bemenet → „Input error!”
üres bemenet → „Input error!”
nem szám bemenet → „Input error!”
tizedestört bemenet → „Input error!”
negatív bemenet → „Input error!”
bemenet szóközzel → „Input error!”
0 → 0
1 → 1
2 → 1
3 → 2
5 → 5
8 → 21

3.2 Számológép használata

Karakterláncként megadott, szóközzel tagolt aritmetikai műveletek három tizedesjegy pontosságú eredményét adja ki, szintén karakterláncként. A program tetszőleges elemű műveletsort képes elvégezni, a műveleti sorrend betartásával, de a zárójeleket nem ismeri. Tizedestörteket ponttal jelölve ismeri fel. Nem kellően tagolt, nem számokat, zárójeleket tartalmazó bemenetre „Input error!” üzenettel válaszol.

Vizsgált tesztesetek:
null bemenet → „Input error!”
üres bemenet → „Input error!”
„qwertzuiop” → „Input error!”
13 → „Input error!”
13+6 → „Input error!”
5 2 → „Input error!”
2 + 2 → 4,000
5 - 2 → 3,000
2 * 2 → 4,000
4 / 2 → 2,000
13.3 - 11.3 → 2,000
5 - -2 → 7,000
5 / 2 → 2,500
1 - 546 → -545,000
-1 * -5 → 5,000
6 + 4 + 9 + 8.2 → 27,200
6 - 4 + 9 - 8.2 → 2,800
4 * 15 / 6 * 2 → 20,000
8 + 6 * 0 → 8
(13 + 6) * 2 → „Input error!”
13 + - 6 * 2 → „Input error!”
5 13 + 6 2 → „Input error!”
"5 + 2" → „Input error!”