**Proiect Laborator TSS**

Cerinta: Se va introduce o lista de numere separate prin spatiu. Trebuie sa se verifice daca lista are intre 1 si 3 elemente.Conditia pentru numere este ca suma lor sa fie > 30103. Apoi se va verifica daca numerele date au o anumită proprietate: să fie un număr palindrom, să aibă suma cifrelor un număr prim. Programul va intoarce un string:

* input invalid, daca inputul nu este numar
* conditii neindeplinite, daca lista numerelor nu contine intre 1 si 3 elemente sau daca suma nu este > 30103
* conditie partial indeplinita, daca unele numere din lista nu sunt palindrom sau suma cifrelor nu este numar prim
* “DA, DA, DA”, daca toate numerele din lista indeplinesc conditia, un “DA” pentru fiecare numar.

1.Pe baza cerintelor programului, sa se genereze date de test folosind a) equivalence partitioning b) boundary value analysis si c) cause-effect graphing. Sa se implementeze testele obtinute folosind JUnit.

* Equivalence Partitioning:

**Intrarile programului**: lista de numere → numbers

**Clasele intrarilor:**

N\_1 = {numbers cu un element ce indeplineste conditiile}

N\_2 = {numbers cu doua elemente ce indeplinesc conditiile}

N\_3 = {numbers cu trei elemente ce indeplinesc conditiile}

N\_4 = {numbers cu mai mult de trei elemente}

N\_5 = {numbers cu trei elemente din care doua indeplinesc conditiile}

N\_6 = {numbers cu trei elemente din care unul indeplineste conditiile}

N\_7 = {numbers cu elemente invalide}

**Clasele iesirilor:**

I\_1:”DA”

I\_2:”DA , DA”

I\_3:”DA, DA, DA”

I\_4:”Conditii neindeplinite”

I\_5:”Conditie partial indeplinita”

I\_6:”Input invalid. Introduceti numere valide.”

**Clasele de echivalenta:**

C\_1:{numbers in N\_1, iesirea I\_1}

C 2:{numbers in N\_2, iesirea I\_2}

C 3: {numbers in N\_3, iesirea I\_2}

C 4: {numbers in N\_4, iesirea I\_4}

C\_5: {numbers in N\_5, iesirea I\_5}

C\_6: {numbers in N\_6, iesirea I\_5}

C\_7: {numbers in N\_7, iesirea I\_6}

* Boundary Value Analysis:

**Intrari**: lista de numere → numbers

**Frontiere pentru lista de numere:**

– min 1 element

– max 3 elemente

– suma elementelor min 30103

Vom defini testele astfel:

– 1 element = 30103

– 3 elemente cu suma 30103

* Cause-Effect Graphing:

C1: input valid

C2: lista valida & suma numerelor din lista > 30103

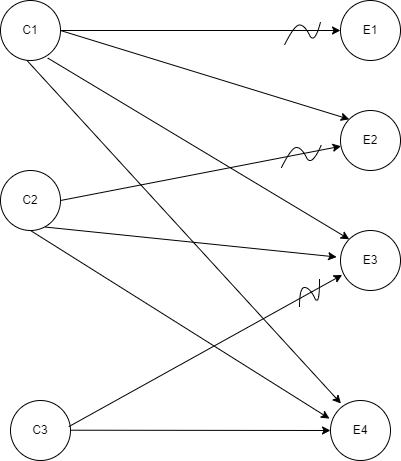
C3: lista cu numere palindrom & suma cifrelor prima

E1:”Input invalid. Introduceti numere valide.”

E2:”Conditii neindeplinite”

E3:”Conditie partial indeplinita”

E4:”DA, DA, DA”



Parcurgerea efectelor:

E1:”Input invalid. Introduceti numere valide.”

~C1 = 1 => C1 = 0; restul de valori pt C2,C3 nu mai conteaza

| C1 | 0 |
| --- | --- |
| C2 | 0 |
| C3 | 0 |
| E1 | 1 |
| E2 | 0 |
| E3 | 0 |
| E4 | 0 |

E2:”Conditii neindeplinite”

C1 ^ ~C2=1 => C1=1, C2=0

| C1 | 0 | 1 |
| --- | --- | --- |
| C2 | 0 | 0 |
| C3 | 0 | 0 |
| E1 | 1 | 0 |
| E2 | 0 | 1 |
| E3 | 0 | 0 |
| E4 | 0 | 0 |

E3:”Conditie partial indeplinita”

C1^C2^~C3=1 => C1=1,C2=1, C3=0

| C1 | 0 | 1 | 1 |
| --- | --- | --- | --- |
| C2 | 0 | 0 | 1 |
| C3 | 0 | 0 | 0 |
| E1 | 1 | 0 | 0 |
| E2 | 0 | 1 | 0 |
| E3 | 0 | 0 | 1 |
| E4 | 0 | 0 | 0 |

E4:”DA, DA, DA”

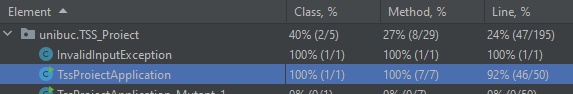
C1^C2^C3 = 1 => C1=1,C2=1,C3=1

| C1 | 0 | 1 | 1 | 1 |
| --- | --- | --- | --- | --- |
| C2 | 0 | 0 | 1 | 1 |
| C3 | 0 | 0 | 0 | 1 |
| E1 | 1 | 0 | 0 | 0 |
| E2 | 0 | 1 | 0 | 0 |
| E3 | 0 | 0 | 1 | 0 |
| E4 | 0 | 0 | 0 | 1 |
| **Input** | {1abc,2} | {1,2,3,4} | {15123, 16123, 17123} | {15151,16361,1001} |
| **Output** | ”Input invalid. Introduceti numere valide.” | “Conditii neindeplinite” | “Conditie partial indeplinita” | “DA, DA, DA” |

2.Sa se stabileasca nivelul de acoperire realizat de fiecare test realizat anterior, am folosit Tool-ul din IntelliJ.

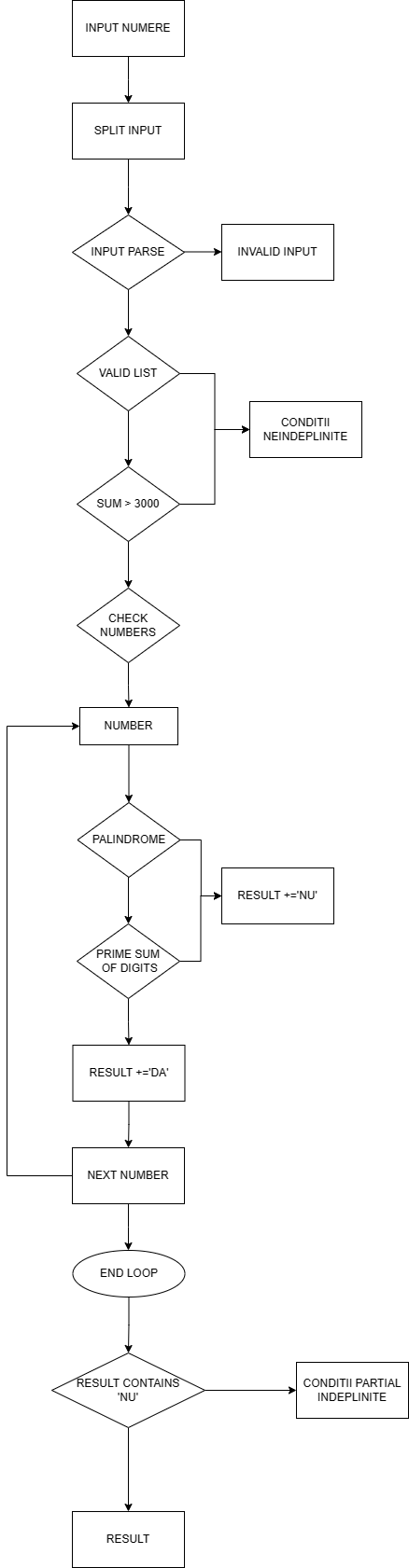
In urma rularii testelor, pentru toate 3 am obtinut urmatorul rezultat:

Method:100%, Class: 100%, Line:92%.



Ceea ce indica un coverage bun al testelor.

3. Sa se transforme programul intr-un graf orientat si, pe baza acestuia, sa se gaseasca un set de teste care satisface criteriul modified condition/decision coverage (MC/DC).



**Modified condition / decision coverage (MC/DC)**

Pentru a satisface criteriul MC/DC, trebuie să testăm toate combinațiile posibile de intrări pentru fiecare conditie.

* INPUT PARSE, teste posibile:
  + {12,15,5} → VALID
  + {1xy, 23,1} → INVALID INPUT
* VALID LIST
  + {lista cu un element} : {15151}
  + {lista cu 2 elemente}: {15151,16361}
  + {lista cu 3 elemente} : {15151,16361, 1001}
  + {lista cu mai mult de 3 elemente}: {15151,16361, 1001, 121} → CONDITII NEINDEPLINTE
  + {lista fara niciun element}: {} → CONDITII NEINDEPLINTE
* SUM >= 30103, teste posibile:
  + {15050, 17032,10000} → suma >= 30103
  + {12,15,5} → suma < 30103 → CONDITII NEINDEPLINTE
* PALINDROME
  + {lista cu numere palindrom}:{15151,16361, 1001}
  + {lista cu numere !palindrom}: {15151,16361, 1002}
* PRIME SUM OF DIGITS
  + {lista cu numere care au suma cifrelor prima}:{15151,16361, 1001}
  + {lista cu numere care nu au suma cifrelor prima} : {40104}
* RESULT CONTAINS ‘NU’
  + {lista ce genereaza mesajul ‘NU’}: {15152,16367}
  + {lista ce nu genereaza mesajul ‘NU’}: {15151,16361, 1001}

4. Sa se scrie un mutant de ordinul 1 echivalent al programului.

– modificare ce nu schimba rezultatul programului

public static boolean hasSumGreater(List<Integer> numbers) {

int sum = 0;

for (int num : numbers) {

sum += num;

}

//modified this condition

return sum > 30102;

}

5. Pentru unul dintre cazurile de testare de mai sus sa se scrie un mutant ne-echivalent care sa fie omorat de catre test si un mutant ne-echivalent care sa nu fie omorat de catre test.

Ne vom folosi de testul: **TestsCauseEffectGraphingE4**

**Mutant ne-echivalent care sa fie omorat de test**

public static boolean hasPrimeSumOfDigits(int n) {

// int sum = 0;

// while (n > 0) {

// sum += n % 10;

// n /= 10;

// }

// return isPrime(sum);

//modified this

return false;

}

**Mutant ne-echivalent care sa nu fie omorat de test**

public static String checkNumbers(List<Integer> numbers) {

StringBuilder result = new StringBuilder();

if (isValidList(numbers) && hasSumGreater(numbers)) {

for (int num : numbers) {

boolean isPalindrome = isPalindrome(num);

boolean hasPrimeSum = hasPrimeSumOfDigits(num);

if (isPalindrome && hasPrimeSum) {

result.append("DA, ");

} else {

result.append("NU, ");

}

}

//nu vom mai verifica daca result contine nu

return result.toString().substring(0, result.length() - 2);

} else {

return "Conditii neindeplinite";

}

}