Testarea Sistemelor Software

Unit Testing (Java)

Echipa:

Atasie Oana (341)

Burta Mihai (342)

Matei Andreea (341)

Stoica Liviu (342)

Tudorache Theodor (342)

Specificatia problemei:

Date fiind numerele a, b, k, s, sa se gaseasca primele k numere prime din intervalul [a, b] care au suma cifrelor egala cu s, avand urmatoarele conditii:

- k >= 0
- s >= 0
- a >= 0
- b >= 0
- a <= b

Implementarea solutiei:

```
public static List<Integer> findPrimes(int k, int a, int b, int s)
throws IllegalArgumentException {
   if ( k < 0 ) throw new IllegalArgumentException("K is negative.");
   if ( s < 0 ) throw new IllegalArgumentException("S is negative.");
   if ( a < 0 || b < 0) throw new IllegalArgumentException("Range is
negative.");
   if ( a > b ) throw new IllegalArgumentException("Range is
reversed.");

   List<Integer> primes = new ArrayList<>();
   int copy;
   int digit;
   int sum;
   boolean found;
   int number, divisor;
```

```
    for(number = a; primes.size() < k && number <= b; number++)
    {
       if(number == 0 || number == 1) continue;
       found = false;
       for(divisor = 2; !found && divisor <= sqrt(number); divisor++)
       {
          if(number % divisor == 0) found = true;
       }

       if(!found)
       {
          copy = number;
          sum = 0;
          while(copy != 0)
          {
             digit = copy % 10;
             copy = copy / 10;
             sum += digit;
          }
          if(sum == s) primes.add(number);
       }
    }
    return primes;
}
```

1. Testare Functionala

a) Partitionarea de echivalenta

b) Analiza valorile de frontiera

c) Partitionarea in categorii

Am parcurs pasii corespunzatori metodei de testare:

1. Descompune specificatia în unități: avem o singură unitate.

2. Identifică parametrii: k, a, b, s, numerele din [a, b]

3. Găsește categorii:

- k: Daca este mai mare decat 0
- a: Daca este mai mare decat 0
- b: Daca este mai mare decat 0
- Relatia dintre a si b: Daca a <= b sau a > b

- Intre a si b exista cel putin un numar prim/niciunul.
- s: Daca este mai mare decat 0
- Intre a si b exista cel putin un numar cu suma cifrelor s/niciunul.

4. Partiționeaza fiecare categorie în alternative:

- k: < 0, 0, > 0
- a: < 0, 0, 1, > 1, numar prim
- b: < 0, 0, 1, > 1, numar prim
- a & b: a < b; a = b; a > b
- [a, b]: Niciun numar prim in interval; Cel putin un numar prim in interval
- s: < 0, 0, > 0;
- [a, b] & s: Niciun numar cu suma cifrelor s in interval; Cel putin un numar cu suma cifrelor s in interval

5. Scrie specificația de testare

- *k*

  1) {k | k < 0}

  2) k = 0

  3) {k | k > 0}

- *a*

  1) {a | a < 0}

  2) a = 0

  3) a = 1

  4) {a | a > 1, a nu e prim}

  5) {a | a > 1, a e prim}

- *b*

  1) {b | b < 0}

2) b = 0

3) b = 1

4) {b | b > 1 && b < a}

5) {b | b > 1 && b = a, b nu e prim}

6) {b | b > 1 && b > a, b nu e prim}

7) {b | b > 1 && b = a, b e prim}

8) {b | b > 1 && b > a, b e prim}

- *s*

  1) {s | s < 0}

  2) s = 0

  3) {s | s > 0}

- *n din [a, b] ->*

  1) |{n | a <= n <= b, n nu e prim}| = b - a + 1

  2) |{n | a <= n <= b, n e prim}| >= 1

- *n in functie de s ->*

  1) |{n | a <= n <= b, suma cifrelor lui n nu este s}| >= 1

  2) |{n | a <= n <= b, suma cifrelor lui n este s}| = b - a + 1

6. Creează cazuri de testare

k1	k2	k3a1	k3a2b1	k3a2b2
k3a2b3	k3a2b6s1	k3a2b6s2	k3a2b6s3nab2ns1	k3a2b6s3nab2ns2
k3a2b8s1	k3a2b8s2	k3a2b8s3nab2ns1	k3a2b8s3nab2ns2	k3a3b1
k3a3b2	k3a3b3	k3a3b6s1	k3a3b6s2	k3a3b6s3nab2ns1
k3a3b6s3nab2ns2	k3a3b8s1	k3a3b8s2	k3a3b8s3nab2ns1	k3a3b8s3nab2ns2
k3a4b1	k3a4b2	k3a4b3	k3a4b4	k3a4b5s1

k3a4b5s2	k3a4b5s3nab1ns1	k3a4b5s3nab1ns2	k3a4b6s1	k3a4b6s2
k3a4b6s3nab1ns1	k3a4b6s3nab1ns2	k3a4b6s3nab2ns1	k3a4b6s3nab2ns2	k3a4b8s1
k3a4b8s2	k3a4b8s3nab2ns1	k3a4b8s3nab2ns2	k3a5b1	k3a5b2
k3a5b3	k3a5b4	k3a5b4	k3a5b6s2	k3a5b6s3nab2ns1
k3a5b6s3nab2ns2	k3a5b7s1	k3a5b7s2	k3a5b7s3nab2ns1	k3a5b7s3nab2ns2
k3a5b8s1	k3a5b8s2	k3a5b8s3nab2ns1	k3a5b8s3nab2ns2	

In total: 59 cazuri de testare

7. Creează date de test

Nume test	**Intrari**				**Rezultat afișat (expected)**
	k	**a**	**b**	**s**	
k1	-1				Exceptie: K is negative
k2	0	0	0	0	[]
k3a1	1	-1			Exceptie: Range is negative
k3a2b1	1	0	-1		Exceptie: Range is negative
k3a2b2	1	0	0	0	[]
k3a2b3	1	0	1	0	[]
k3a2b6s1	1	0	4	-1	Exceptie: S is negative
k3a2b6s2	1	0	4	0	[]
k3a2b6s3nab2ns1	1	0	9	17	[]
k3a2b6s3nab2ns2	1	0	14	2	[2]
k3a2b8s1	1	0	3	-1	Exceptie: S is negative
k3a2b8s2	1	0	3	0	[]
k3a2b8s3nab2ns1	1	0	7	17	[]
k3a2b8s3nab2ns2	1	0	7	7	[7]
k3a3b1	1	1	-1	0	Exceptie: Range is negative
k3a3b2	1	1	0	0	Exceptie: Range is reversed
k3a3b3	1	1	1	1	[]

k3a3b6s1	1	1	2	-1	Exceptie: S is negative
k3a3b6s2	1	1	2	0	[]
k3a3b6s3nab2ns1	1	1	9	17	[]
k3a3b6s3nab2ns2	2	1	14	2	[2, 11]
k3a3b8s1	3	1	3	-1	Exceptie: S is negative
k3a3b8s2	5	1	3	0	[]
k3a3b8s3nab2ns1	2	1	7	17	[]
k3a3b8s3nab2ns1	1	1	7	7	[7]
k3a4b1	3	4	-1	0	Exceptie: Range is negative
k3a4b2	4	4	0	0	Exceptie: Range is reversed
k3a4b3	2	4	1	0	Exceptie: Range is reversed
k3a4b4	8	4	2	0	Exceptie: Range is reversed
k3a4b5s1	1	4	4	-1	Exceptie: S is negative
k3a4b5s2	9	4	4	0	[]
k3a4b5s3nab1ns1	3	4	4	5	[]
k3a4b5s3nab1ns2	5	4	4	4	[]
k3a4b6s1	6	24	28	-1	Exceptie: S is negative
k3a4b6s2	2	24	28	0	[]
k3a4b6s3nab1ns1	1	24	28	19	[]
k3a4b6s3nab1ns2	4	24	28	9	[]
k3a4b6s3nab2ns1	8	27	30	19	[]
k3a4b6s3nab2ns2	11	27	30	11	[29]
k3a4b8s1	3	4	5	-1	Exceptie: S is negative
k3a4b8s2	3	4	5	0	[]
k3a4b8s3nab2ns1	10	24	29	100	[]
k3a4b8s3nab2ns2	8	24	29	11	[29]
k3a5b1	8	3	-1	0	Exceptie: Range is negative
k3a5b2	7	3	0	0	Exceptie: Range is reversed
k3a5b3	6	3	1	0	Exceptie: Range is reversed
k3a5b4	4	3	2	0	Exceptie: Range is reversed

k3a5b6s1	5	23	28	-1	Exceptie: S is negative
k3a5b6s2	2	23	28	0	[]
k3a5b6s3nab2ns1	10	23	27	19	[]
k3a5b6s3nab2ns2	16	23	27	5	[23]
k3a5b7s1	9	23	23	-1	Exceptie: S is negative
k3a5b7s2	3	23	23	0	[]
k3a5b7s3nab2ns1	1	23	23	19	[]
k3a5b7s3nab2ns2	4	23	23	5	[23]
k3a5b8s1	2	3	5	-1	Exceptie: S is negative
k3a5b8s2	6	3	5	0	[]
k3a5b8s3nab2ns1	5	23	29	100	[]
k3a5b8s3nab2ns2	8	23	29	11	[29]

Exemple de implementare a testelor:

```
@Test
void k1() {
	try {
		resultList = Main.findPrimes(-1, 0, 0, 0);
		Assertions.fail("K should be negative.");
	} catch (IllegalArgumentException e) {
		expectedErrorMessage = "K is negative.";
		Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
	}
}

@Test
void k3a2b3() {
	try {
		expectedList = new ArrayList<>();
		resultList = Main.findPrimes(1, 0, 1, 0);
		Assertions.assertEquals(expectedList, resultList);
	} catch (IllegalArgumentException e) {
		Assertions.fail("Encountered exception: " +
e.getMessage());
	}
}

@Test
void k3a3b2() {
```

```
      try {
            resultList = Main.findPrimes(1, 1, 0, 0);
            Assertions.fail("Range should be reversed.");
      } catch (IllegalArgumentException e) {
            expectedErrorMessage = "Range is reversed.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void k3a4b5s2() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(9, 4, 4, 0);
            Assertions.assertEquals(expectedList, resultList);
      } catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void k3a5b7s3nab2ns2() {
      try {
            expectedList = new ArrayList<>(Arrays.asList(23));
            resultList = Main.findPrimes(4, 23, 23, 5);
            Assertions.assertEquals(expectedList, resultList);
      } catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}
```

2. Testare Structurala

a) Acoperire la nivel de instructiune

b) Acoperire la nivel de decizie sau acoperire la nivel de ramura

c) Acoperire la nivel de conditie

d) Acoperire la nivel de conditie/decizie

e) Acoperire la nivel de conditii multiple

In programul nostru avem 11 decizii:

- k < 0
- s < 0
- a < 0 || b < 0
- a > b
- primes.size() < k && number <= b
- number == 0 || number == 1
- !found && divisor <= sqrt(number)
- number % divisor == 0
- !found
- copy != 0
- sum == s

Trebuie generate date de test astfel încât să fie parcurse toate combinațiile posibile de adevărat și fals ale conditiilor individuale.

Pentru a realiza acest lucru, am generat urmatoarele date de test:

Conditia	**Valorile de adevar**	**Intrari**				**Cand se ajunge la respectivele valori de adevar**
		k	***a***	***b***	***s***	
k < 0	A	-1	0	0	-1	
k < 0	F	1	0	0	-1	
s < 0	A	1	0	0	-1	
s < 0	F	1	-1	0	1	
a < 0 || b < 0	A || A	1	-1	-1	1	
a < 0 || b < 0	A || F	1	-1	0	1	
a < 0 || b < 0	F || A	1	0	-1	1	
a < 0 || b < 0	F || F	1	1	0	1	
a > b	A	1	1	0	1	
a > b	F	1	0	0	1	

primes.size() < k && number <= b	A && A	1	1	2	2	Primul pas din *for*
primes.size() < k && number <= b	A && F	1	1	2	1	Al doilea pas din *for*
primes.size() < k && number <= b	F && A	0	1	1	1	Primul pas din *for*
primes.size() < k && number <= b	F && F	1	1	2	2	Al treilea pas din *for*
number == 0 || number == 1	A || A					Niciodata (este imposibil)
number == 0 || number == 1	A || F	1	0	2	1	Primul pas din *for*
number == 0 || number == 1	F || A	1	1	2	1	Primul pas din *for*
number == 0 || number == 1	F || F	1	2	2	2	Primul pas din *for*
!found && divisor <= sqrt(number)	A && A	1	4	4	4	Primul pas din primul *for*, primul pas din al doilea *for*
!found && divisor <= sqrt(number)	A && F	1	2	2	2	Primul pas din primul *for*, primul pas din al doilea *for*
!found && divisor <= sqrt(number)	F && A	1	12	12	3	Primul pas din primul *for*, al doilea pas din al doilea *for*
!found && divisor <= sqrt(number)	F && F	1	9	9	9	Primul pas din primul *for*, al treilea pas din al doilea *for*
number % divisor == 0	A	1	4	4	4	Primul pas din primul *for*, primul pas din al doilea *for*
number % divisor == 0	F	1	9	9	9	Primul pas din primul *for*, primul pas din al doilea *for*
!found	A	1	3	3	3	Primul pas din primul *for*
!found	F	1	4	4	4	Primul pas din primul *for*, dupa primul pas din al doilea *for*
copy != 0	A	1	3	3	3	Primul pas din primul *for*, primul pas din *while*
copy != 0	F	1	5	5	5	Primul pas din primul *for*, al doilea pas din *while*
sum == s	A	1	7	7	7	
sum == s	F	1	7	7	9	

Exemple de implementare a testelor:

```
@Test
void kLessThanZeroTrue() {
      try {
            resultList = Main.findPrimes(-1, 0, 0, -1);
            Assertions.fail("K should be negative.");
      }
      catch (IllegalArgumentException e) {
            expectedErrorMessage = "K is negative.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void aGreaterThanBTrue() {
      try {
            resultList = Main.findPrimes(1, 1, 0, 1);
            Assertions.fail("Range should be reversed.");
      }
      catch (IllegalArgumentException e) {
            expectedErrorMessage = "Range is reversed.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void numberEqualsZeroOrOneTrueFalse() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 0, 2, 1);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void numberModDivisorTrue() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 4, 4, 4);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
```

```
        }
}
```

f) MC/DC

In programul nostru avem 11 decizii:

- k < 0
- s < 0
- a < 0 || b < 0
- a > b
- primes.size() < k && number <= b
- number == 0 || number == 1
- !found && divisor <= sqrt(number)
- number % divisor == 0
- !found
- copy != 0
- sum == s

Trebuie generate date de test astfel încât:

- Fiecare condiție individuală dintr-o decizie ia atât valoare True cât și valoare False
- Fiecare decizie ia atât valoare True cât și valoare False
- Fiecare condiție individuală influențează în mod independent decizia din care face parte

Pentru a realiza acest lucru, am generat urmatoarele date de test:

Conditia	**Valorile de adevar**	**Intrari**				**Cand se ajunge la respectivele valori de adevar**
		k	***a***	***b***	***s***	
k < 0	A	-1	0	0	-1	
k < 0	F	1	0	0	-1	
s < 0	A	1	0	0	-1	
s < 0	F	1	-1	0	1	

a < 0 || b < 0	A || F	1	-1	0	1	
a < 0 || b < 0	F || A	1	0	-1	1	
a < 0 || b < 0	F || F	1	1	0	1	
a > b	A	1	1	0	1	
a > b	F	1	0	0	1	
primes.size() < k && number <= b	A && A	1	1	2	2	Primul pas din *for*
primes.size() < k && number <= b	A && F	1	1	2	1	Al doilea pas din *for*
primes.size() < k && number <= b	F && A	0	1	1	1	Primul pas din *for*
number == 0 || number == 1	A || F	1	0	2	1	Primul pas din *for*
number == 0 || number == 1	F || A	1	1	2	1	Primul pas din *for*
number == 0 || number == 1	F || F	1	2	2	2	Primul pas din *for*
!found && divisor <= sqrt(number)	A && A	1	4	4	4	Primul pas din primul *for*, primul pas din al doilea *for*
!found && divisor <= sqrt(number)	A && F	1	2	2	2	Primul pas din primul *for*, primul pas din al doilea *for*
!found && divisor <= sqrt(number)	F && A	1	12	12	3	Primul pas din primul *for*, al doilea pas din al doilea *for*
number % divisor == 0	A	1	4	4	4	Primul pas din primul *for*, primul pas din al doilea *for*
number % divisor == 0	F	1	9	9	9	Primul pas din primul *for*, primul pas din al doilea *for*
!found	A	1	3	3	3	Primul pas din primul *for*
!found	F	1	4	4	4	Primul pas din primul *for*, dupa primul pas din al doilea *for*
copy != 0	A	1	3	3	3	Primul pas din primul *for*, primul pas din *while*
copy != 0	F	1	5	5	5	Primul pas din primul *for*, al doilea pas din *while*
sum == s	A	1	7	7	7	
sum == s	F	1	7	7	9	

Exemple de implementare a testelor:

```
@Test
void sLessThanZeroTrue() {
      try {
            resultList = Main.findPrimes(1, 0, 0, -1);
            Assertions.fail("S should be negative.");
      }
      catch (IllegalArgumentException e) {
            expectedErrorMessage = "S is negative.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void aLessThanZeroOrBLessThanZeroFalseTrue() {
      try {
            resultList = Main.findPrimes(1, 0, -1, 1);
            Assertions.fail("Range should be negative.");
      }
      catch (IllegalArgumentException e) {
            expectedErrorMessage = "Range is negative.";
            Assertions.assertEquals(expectedErrorMessage,
e.getMessage());
      }
}

@Test
void notFoundAndDivisorLessThanSqrtNumberTrueTrue() {
      try {
            expectedList = new ArrayList<>();
            resultList = Main.findPrimes(1, 4, 4, 4);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
      }
}

@Test
void sumEqualsSTrue() {
      try {
            expectedList = new ArrayList<>(Arrays.asList(7));
            resultList = Main.findPrimes(1, 7, 7, 7);
            Assertions.assertEquals(expectedList, resultList);
      }
      catch (IllegalArgumentException e) {
            Assertions.fail("Encountered exception: " +
e.getMessage());
```

```
        }
}
```

g) Testarea circuitelor independente

h) Testare la nivel de cale

3. Testarea Mutantilor