POLITECHNIKA KRAKOWSKA
WYDZIAŁ INŻYNIERII ELEKTRYCZNEJ I KOMPUTEROWEJ

Asercja w języku JAVA

Systemy odporne na błędy

Sprawozdanie z laboratorium nr 2

Maria Guz
Karol Pątko
Wojciech Maludziński

Wstęp

Asercja jest prostą formą testu akceptacyjnego. Decydując się na korzystanie z asercji zakładamy, że dany predykat powinien być prawdziwy. Sytuacja przeciwna powoduje przerwanie działania programu. Jedną z zalet używania mechanizmu asercji jest możliwość zidentyfikowania, gdzie dokładnie nastąpił błąd, dlatego dobrze sprawdza się przy testowaniu kodu.

Cel zajęć

Celem zajęć laboratoryjnych nr 2 jest powtórzenie i utrwalenie wiadomości związanych z mechanizmem asercji oraz jego implementacja w języku JAVA. Poza opracowaniem teoretycznym tematu studenci mają obowiązek przygotować praktyczne zadanie ukazujące działanie asercji w języku JAVA.

Treść zadania

Stwórz klasę NumberSet, której szkielet wygląda następująco:

```
package number.set;
/**
*
* @author slawek
* @version 1.0
*
*/
public class NumberSet {
   public static int MAX_SIZE = 100;
   private int[] nSet = new int[MAX_SIZE];
   private int size;
   /**
    * Metoda dodaje liczbę do zbioru liczb
    * (zezwalamy na dodanie liczby już istniejącej)
    *
    * @param i
    * liczba którą dodajemy
    * @throws Exception
    * występuje w przypadku przepełnienia zbioru
    */
   public void add(int i) throws Exception {
   }
   /**
    * Metoda usuwa liczbę ze zbioru (każde wystąpienie)
    *
    * @param i
    * liczba do usunięcia
    * @throws Exception
    * występuje jeśli zbiór nie posiada liczby
   którą chcemy usunąć
    */
   public void remove(int i) throws Exception {
   }
   /**
    * Metoda losuje jedną liczbę ze zbioru oraz ją usuwa
    *
    * @return wylosowana liczba
    * @throws Exception
    * występuje jeśli zbiór jest pusty
    */
   public int getRandomValue() throws Exception {
```

```
        return 0;
    }
    /**
     * Metoda zwraca sumę elementów
     *
     * @return Suma liczb.
     * @throws Exception
     * występuje jeśli zbiór jest pusty.
     */
    public int getSumOfElements() throws Exception {
        return 0;
    }
    /**
     * Metoda dzieli każdy element zbioru przez podaną
     wartość bez reszty
     *
     * @param d
     * liczba przez którą dzielimy
     * @throws Exception
     * występuje w przypadku dzielenia przez 0
    (można zastąpić asercją)
     */
    public void divideAllElementsBy(int d) throws Exception {
    }
    /** 4
     * Metoda sprawdza czy w zbiorze istnieje podany element
     *
     * @param i
     * element do sprawdzenia
     * @return true w przypadku odnalezienia wartości,
     * false w przeciwnym razie.
     */
    public boolean contains(int i) {
        return false;
    }
    /**
     * Metoda zwraca rozmiar zbioru (liczbę elementów)
     *
     * @return rozmiar zbioru
     */
    public int getSize() {
        return 0;
    }
}
```

Do każdej metody należy wymyślić kilka asercji, które będą sprawdzać poprawność programu. Stwórz klasę testującą, która będzie wywoływać metody z klasy NumberSet.

Rozwiązanie zadania

Kod został rozdzielony na pliki *Main.java*, *NumberSet.java* oraz *NumberSetTest.java*. Klasa *Main* zawiera funkcję *main()*, oraz obiekt klasy *NumberSetTest* testującej *NumberSet*. W pliku *NumberSet.java* znajduje się klasa *NumberSet* oraz metody realizujące operacje na zbiorze. Klasa *NumberSetTest* jest klasą testującą *NumberSet*. Wszystkie metody zostały omówione na kolejnych stronach.

Main.java

```
public class Main {
    public static void main(String[] args) throws Exception {
        new NumberSetTest();
    }
}
```

NumberSet.java

```
import java.util.NoSuchElementException;
import java.util.Random;

public class NumberSet {
   public static int MAX_SIZE = 100;
   private int[] nSet;
   private int size;
   /**
    * Constructor
    */
   NumberSet() {
       assert MAX_SIZE > 0: "MAX_SIZE is <= 0";
       nSet = new int[MAX_SIZE];
       size=0;
       assert size == 0;
       assert nSet != null : "nSet is null";
   }

   /**
    * Metoda dodaje liczbę do zbioru liczb
    * (zezwalamy na dodanie liczby już istniejącej)
    *
    * @param i liczba ktora dodajemy
    * @throws Exception występuje w przypadku przepełnienia zbioru
    */
   public void add(int i) throws Exception {
       int oldSize = size;
       assert nSet != null: "nSet is null";

       if (size >= MAX_SIZE) {
           throw new ArrayIndexOutOfBoundsException("(add) Adding unsuccessfull due to full set");
       }
       else {
           nSet[size] = i;
           setSize(size+1);
       }
       assert nSet[size-1] == i: "nSet[size-1] differ from i";
       assert nSet != null: "nSet is null";
       assert size <= MAX_SIZE: "size is greater than MAX size";
       assert oldSize+1 == size: "new size is incorect";
   }

   /**
    * metoda uzywana w remove do asserta dla przypadku z redundacja kodu
    */
   public Integer countOccurred(int val){
       assert nSet != null: "nSet is null";
       Integer ret = 0;

       for(int i=0; i<size; ++i){
           if(nSet[i]==val){
               ++ret;
           }
       }
       assert ret != null: "return value is null";
       assert ret >= 0: "return value must be >= 0";
       return ret;
   }

   /**
    * Metoda usuwa liczbę ze zbioru (każde wystąpienie)
    *
    * @param val liczba do usunięcia
    * @throws Exception występuje jeśli zbiór nie posiada liczby którą chcemy usunąć
    */
   public void remove(int val) throws Exception {
       assert nSet != null: "nSet is null";
       assert size > 0: "size return value <= 0";

       Boolean isInSet = contains(val);
       assert isInSet != null: "isInSet isn't initialized";
       if (!isInSet)
           throw new NoSuchElementException("(remove) No elements to remove");
       int countOccurred = countOccurred(val);
       int[] tempArr = new int[size];
       int tempSize = 0;

       assert tempArr != null : "tempArr isn't initialized";
```

```
        for (int i = 0; i < size; ++i) {
            if (nSet[i] != val) {
                tempArr[tempSize] = nSet[i];
                ++tempSize;
                assert tempSize < size: "new size must be less ";
            }
        }
        assert tempSize == size - countOccurred: "new size is wrong";

        nSet = tempArr;
        size = tempSize;
        assert nSet!=null;
        assert size >= 0: "new size is less than 0";
        assert size <= MAX_SIZE: "new size is bigger than MAX_SIZE";
    }

    /**
     * Metoda losuje jedną liczbę ze zbioru oraz usuwa ją
     *
     * @return wylosowana liczba
     * @throws Exception występuje jeśli zbiór jest pusty
     */
    public int getRandomIndex() throws Exception {
        assert nSet != null: "nSet is null";
        int oldSize = size;
        if (size == 0)
            throw new ArrayIndexOutOfBoundsException("Set is empty");

        Random random = new Random();
        int getRandomIndex = random.nextInt(size);
        assert getRandomIndex<size && getRandomIndex>=0: "Index out of bonds";
        setSize(size-1);

        System.out.println("deleted random value: " + nSet[getRandomIndex] + " from index: "
+getRandomIndex);
        nSet[getRandomIndex] = nSet[size];

        assert nSet != null;
        assert oldSize-1 == size;

        return nSet[getRandomIndex];
    }
    /**
     * Metoda zwraca sumę elementów
     *
     * @return Suma liczb.
     * @throws Exception występuje jeśli zbiór jest pusty.
     */
    public int getSumOfElements() throws Exception {
        assert nSet != null: "nSet is null";

        if (size == 0) {
            throw new ArrayIndexOutOfBoundsException("(getSumOfElements) set is empty");
        }

        int sum = 0;
        for (int i = 0; i < size; ++i) {
            sum += nSet[i];
        }

        assert isObjectInteger(sum): "Sum is not integer";
        return sum;
    }

    /**
     * Metoda dzieli każdy element zbioru przez podaną wartość bez reszty
     *
     * @param d liczba przez którą dzielimy
     * @throws Exception występuje w przypadku dzielenia przez 0 (można zastąpić asercją)
     *
     */
    public void divideAllElements(int d) {
        assert nSet != null: "nSet is null";

        if(d == 0){ throw new IllegalArgumentException("Dividing by 0 is prohibited"); }

        for (int i = 0; i < size; ++i) {
            assert d != 0: "Dividing by 0 is prohibited";
            nSet[i] /= d;
            assert isObjectInteger(nSet[i]): "Error during dividing";
        }
        assert nSet != null: "nSet is null";
        assert size <= MAX_SIZE: "size is over MAX_SIZE";
```

```
        assert size >=0 : "size is less than 0";
    }

    /**
     * Metoda sprawdza czy w zbiorze istnieje podany element
     *
     * @param i element do sprawdzenia
     * @return true
     * w przypadku odnalezienia wartości,
     * w przypadku odnalezienia wartości
     * <p>
     * false
     * w przeciwnym razie.
     */
    public Boolean contains(int i) {
        assert nSet != null: "nSet is null";

        Boolean isInSet = false;
        for (int j = 0; j < size; ++j) {
            if (nSet[j] == i) {
                isInSet = true;
                break;
            }
        }

        assert isInSet != null: "isInSet variable isn't initialized";

        return isInSet;
    }


    public int getSize(){
        assert size >= 0 : "Size is less than 0";
        assert size <= MAX_SIZE : "Size is bigger than MAX_SIZE";
        return size;
    }

    /**
     * Metoda ustawia rozmiar zbioru (liczbę elementów)
     *
     * @param size - nowy rozmiar zbioru
     */
    public void setSize(int size) {
        assert size <= MAX_SIZE: "size must be less than MAX_SIZE";
        assert size >= 0: "size can't be less than 0";

        if(size <= MAX_SIZE) {
            this.size = size;
        }
            else{
                throw new IllegalArgumentException("size must be lower than MAX_SIZE");
            }


        assert this.size == size: "something is wrong with the size assignment";
    }

    /**
     * Metoda wyświetla zbiór.
     */
    void showArr() {
        StringBuilder sb = new StringBuilder("nSet: ");
        for (int i = 0; i < size; ++i) {
            sb.append(nSet[i]);
            sb.append("  ");
            if(i%10 == 0 && i != 0){
                sb.append("\n");
            }
        }
        System.out.println(sb.toString());
    }

    /**
     *
     * @param o objekt do sprawdzenia typu
     * @return prawda dla integerów, fałsz dla reszty
     */
    private boolean isObjectInteger(Object o) { return Integer.class.isInstance(o); }
}
```

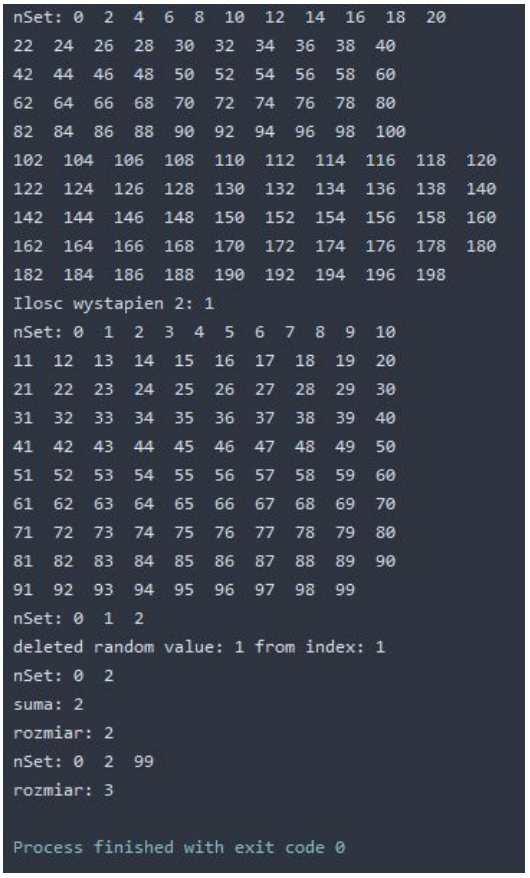
NumberSetTest.java

```
public class NumberSetTest  {
   public NumberSetTest() throws Exception {
       NumberSet numberSet = new NumberSet();
       for(int i=0; i<100; ++i){
           try {
               numberSet.add(i * 2);
           }catch(Exception e){
               System.out.println(e);
           }
       }
       numberSet.showArr();
       System.out.println("Ilosc wystapien 2: "+numberSet.countOccurred(2));

       //dzielenie
       try {
           numberSet.divideAllElements(2);
       }catch(Exception e){
           System.out.println(e);
       }
       numberSet.showArr();
       for(int i = 3; i<=99; ++i) {
           try{
               numberSet.remove(i);
           }catch(Exception e){
               System.out.println(e);
           }
       }
       numberSet.showArr();
       numberSet.getRandomIndex();
       numberSet.showArr();

       System.out.println("suma: "+numberSet.getSumOfElements());

       System.out.println("rozmiar: "+numberSet.getSize());
       try{
           numberSet.add(99);
       }
       catch (Exception e){
           System.out.println(e);
       }
       numberSet.showArr();
       System.out.println("rozmiar: "+numberSet.getSize());
   }
}
```

Działanie programu

Działanie programu bez wywołania błędów

Klasa testująca wywołuje kolejno metody odpowiedzialne za utworzenie obiektu *NumberSet*, dodanie 100 elementów do zestawu, wyświetlenie zestawu, zliczenie ilości wystąpień, wartości 2 w zestawie, podzielenie elementów przez 2, wyświetlenie zestawu, usunięcie wartości od 3 do 99, wyświetlenie zestawu, usunięcie losowej wartości, obliczenie sumy elementów, wyświetlenie rozmiaru zestawu, dodanie wartości 99, wyświetlenie zestawu, wyświetlenie rozmiaru zestawu. W efekcie otrzymujemy następujące wyjście programu.



Generowanie błędów

Aby pokazać działanie asercji zawartych wewnątrz metod, w niektórych przypadkach nieunikniona była modyfikacja kodu tak, aby błędy zostały wygenerowane. Poniżej zostały opisane metody oraz przykłady wywołań w przypadku gdy dochodzi do błędu programu z włączoną asercją oraz bez asercji. Fragmenty programu, które są przyczyną błędu zostały przytoczone w ramkach.

metoda add(int i)

```
public void add(int i) throws Exception {
        int oldSize = size;
        assert nSet != null: "nSet is null";

        if (size >= MAX_SIZE) {
            throw new ArrayIndexOutOfBoundsException("(add) Adding unsuccessfull due to full set");
        }
        else {
            nSet[size] = i;
            setSize(size+1);
        }
        assert nSet[size-1] == i: "nSet[size-1] differ from i";
        assert nSet != null: "nSet is null";
        assert size <= MAX_SIZE: "size is greater than MAX size";
        assert oldSize+1 == size: "new size is incorect";
    }
```

Opis metody

Metoda *add(int i)* dodaje liczbę całkowitą (int) wskazaną jako parametr *i* metody. We wnętrzu metody znajdują się cztery różne asercje. Są one odpowiedzialne za sprawdzenie kolejno:

- czy zestaw liczb został poprawnie utworzony, [`assert nSet != null`]
- czy wartość ostatniego elementu jest wartością równą podawanej przez nas jako parametr funkcji, [`assert nSet[size-1]`]
- czy rozmiar zbioru jest mniejszy od rozmiary maksymalnego oraz [`assert size <= MAX_SIZE`]
- czy nowy rozmiar zbioru jest większy od wcześniejszego dokładnie o 1. [`assert oldSize+1 == size`]

Wygenerowane błędy

```
nSet = null;
```

Błąd polega na nadpisaniu referencji obiektu klasy *NumberSet* w trakcie wywoływania metody.

Wywołanie metody bez asercji

```
java.lang.NullPointerException
java.lang.NullPointerException
java.lang.NullPointerException
java.lang.NullPointerException
java.lang.NullPointerException
java.lang.NullPointerException
java.lang.NullPointerException
java.lang.NullPointerException
nSet:

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: nSet is null
    at NumberSet.add(NumberSet.java:31)
    at NumberSetTest.<init>(NumberSetTest.java:8)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
nSet[size-1] = -5;
```

Błąd polega na przypisaniu stałej wartości na indeksie, na którym wartość została dodana.

Wywołanie metody bez asercji

```
nSet: -5  -5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5  -5
-5  -5  -5  -5  -5  -5  -5  -5  -5

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: nSet[size-1] differ from i
    at NumberSet.add(NumberSet.java:44)
    at NumberSetTest.<init>(NumberSetTest.java:8)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
if (size >= MAX_SIZE) {
   throw new ArrayIndexOutOfBoundsException("(add) Adding unsuccessfull due to full set");
}
else {
   nSet[size] = i;
   setSize(size+10);
}
```

Błąd polega na zwiększaniu aktualnego rozmiaru tablicy o 10 zamiast o 1 w trakcie wywoływania metody.

Wywołanie metody bez asercji

```
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
nSet: 0  0  0  0  0  0  0  0  0  0  2
0  0  0  0  0  0  0  0  0  4
0  0  0  0  0  0  0  0  0  6
0  0  0  0  0  0  0  0  0  8
0  0  0  0  0  0  0  0  0  10
0  0  0  0  0  0  0  0  0  12
0  0  0  0  0  0  0  0  0  14
0  0  0  0  0  0  0  0  0  16
0  0  0  0  0  0  0  0  0  18
0  0  0  0  0  0  0  0  0

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: new size is incorect
    at NumberSet.add(NumberSet.java:46)
    at NumberSetTest.<init>(NumberSetTest.java:8)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
size = MAX_SIZE+1;
```

Błąd polega na próbie dodania większej ilości elementów niż wskazuje wartość *MAX_SIZE*.

Wywołanie metody bez asercji

```
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
java.lang.ArrayIndexOutOfBoundsException: (add) Adding unsuccessfull due to full set
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 100 out of bounds for length 100
    at NumberSet.showArr(NumberSet.java:236)
    at NumberSetTest.<init>(NumberSetTest.java:13)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: size is greater than MAX size
    at NumberSet.add(NumberSet.java:48)
    at NumberSetTest.<init>(NumberSetTest.java:8)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

metoda countOccurred(int val)

```
public Integer countOccurred(int val){
       assert nSet != null: "nSet is null";
       Integer ret = 0;

       for(int i=0; i<size; ++i){
           if(nSet[i]==val){
               ++ret;
           }
       }
       assert ret != null: "return value is null";
       assert ret >= 0: "return value must be >= 0";
```

```
        return ret;
    }
```

Opis metody

Metoda *countOccurred(int val)* zlicza wystąpienia podanej jako parametr metody wartości. Wewnątrz metody znajdują się trzy asercje odpowiedzialne za sprawdzenie kolejno:

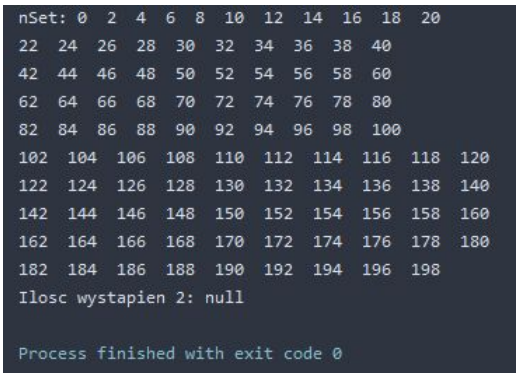
- czy zestaw liczb został poprawnie utworzony, [`assert nSet != null`]
- czy ilość wystąpień danej liczby jest różna od wartości *null* [`assert ret != null`] oraz
- czy ilość wystąpień danej liczby jest liczbą nieujemną. [`assert ret >= 0`]

Wygenerowane błędy

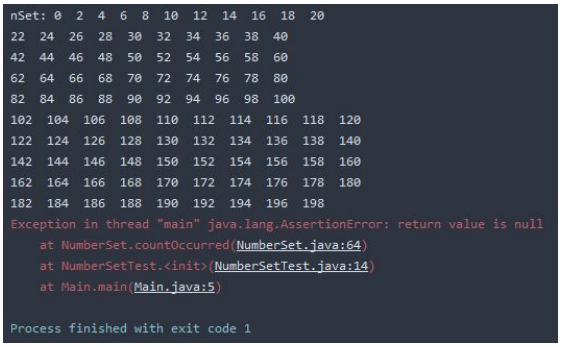
```
ret = null;
```

Błąd polega na przypisaniu wartości *null* do obiektu *ret* podczas wykonywania metody.

Wywołanie metody bez asercji



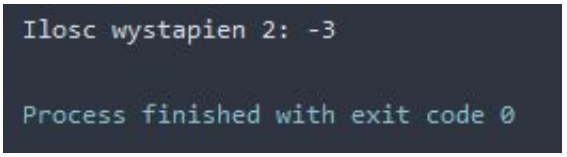
Wywołanie metody z asercją



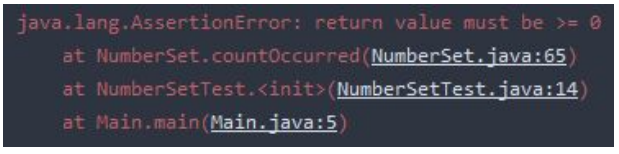

```
ret = -3;
```

Błąd polega na przypisaniu wartości ujemnej do obiektu *ret* podczas wykonywania metody.

Wywołanie metody bez asercji



Wywołanie metody z asercją



Jedna z asercji [`assert nSet != null`] nie została opisana w tym punkcie, ponieważ została ona opisana w punkcie wyżej.

metoda remove(int val)

```
public void remove(int val) throws Exception {
        assert nSet != null: "nSet is null";
        assert size > 0: "size return value <= 0";
```

```
        Boolean isInSet = contains(val);
        assert isInSet != null: "isInSet isn't initialized";
        if (!isInSet)
            throw new NoSuchElementException("(remove) No elements to remove");
        int countOccurred = countOccurred(val);
        int[] tempArr = new int[size];
        int tempSize = 0;

        assert tempArr != null : "tempArr isn't initialized";
        for (int i = 0; i < size; ++i) {
            if (nSet[i] != val) {
                tempArr[tempSize] = nSet[i];
                ++tempSize;
                assert tempSize < size: "new size must be less ";
            }
        }
        assert tempSize == size - countOccurred: "new size is wrong";

        nSet = tempArr;
        size = tempSize;
        assert nSet!=null;
        assert size >= 0: "new size is less than 0";
        assert size <= MAX_SIZE: "new size is bigger than MAX_SIZE";
    }
```

Opis metody

Metoda *remove(int val)* usuwa wszystkie wystąpienia liczby podanej jako parametr z zestawu - tworzy nowy zestaw, gdzie zapisuje wszystkie elementy różne od podanego jako parametr, i nadpisuje stary zestaw nowym zestawem, i stary rozmiar zestawu nowym rozmiarem. We wnętrzu metody znajduje się siedem różnych asercji:

- Pierwsza zapewnia, że *nSet* istnieje (nie jest nullem) [`assert nSet != null`]
- Druga zapewnia, że *nSet* nie jest pusty [`assert size >= 0`]
- Trzecia asercja zapewnia, że powstał obiekt *Boolean*, który jest odpowiedzialny za przechowywanie informacji, czy podana liczba występuje w zbiorze [`assert isInSet != null`]
- Czwarta asercja zapewnia, że tymczasowa tablica powstała [`assert tempArr != null`]
- Piąta asercja zapewnia, że rozmiar tablicy tymczasowej jest mniejszy niż rozmiar tablicy oryginalnej [`assert tempSize < size`]
- Szósta asercja zapewnia, że wszystkie elementy o wartości podanej w parametrze zostały usunięte [`assert tempSize == size - countOccurred`]
- Dziewiąta asercja zapewnia, że nie został przekroczony rozmiar zbioru [`assert size <= MAX_SIZE`]

Wygenerowane błędy

```
size = 0;
```

Błąd polega na zmianie wartości *size* na niedozwoloną podczas wywołania metody.

Wywołanie metody bez asercji

```
java.util.NoSuchElementException: (remove) No elements to remove

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: size return value <= 0
    at NumberSet.remove(NumberSet.java:77)
    at NumberSetTest.<init>(NumberSetTest.java:16)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
Boolean isInSet = contains(val);
//...
//-------------------------------------------------------

//...
private Boolean contains(int i) {
//...
   return null;
}
```

Błąd polega na błędnym zwróceniu wartości *null* przez metodę *contains(int i)*.

Wywołanie metody bez asercji

```
java.lang.NullPointerException

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: isInSet isn't initialized
    at NumberSet.remove(NumberSet.java:79)
    at NumberSetTest.<init>(NumberSetTest.java:16)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
int[] tempArr = null;
```

Błąd polega na przypisaniu wartości *null* do obiektu *tempArr*.

Wywołanie metody bez asercji

```
java.lang.NullPointerException

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: tempArr isn't initialized
    at NumberSet.remove(NumberSet.java:88)
    at NumberSetTest.<init>(NumberSetTest.java:16)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
for (int i = 0; i < size; ++i) {
   if (nSet[i] != val) {
       tempArr[tempSize] = nSet[i];
       tempSize+=2;
       assert tempSize < size: "new size must be less";
   }}
```

Błąd polega na przypisaniu wartości większej od *size* do obiektu *tempSize*.

Wywołanie metody bez asercji

```
java.lang.ArrayIndexOutOfBoundsException: Index 1000 out of bounds for length 100

Process finished with exit code 0
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: new size must be less or equal
    at NumberSet.remove(NumberSet.java:92)
    at NumberSetTest.<init>(NumberSetTest.java:16)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
size = tempSize - 10;
assert tempSize == size - countOccurred: "new size is wrong";
nSet = tempArr;
```

Błąd polega na nadpisaniu rozmiaru tablicy oryginalnej niepoprawną wartością

Wywołanie metody bez asercji

```
 61  62  63  64  65  66  67  68  69  70
 71  72  73  74  75  76  77  78  79  80
 81  82  83  84  85  86  87  88  89  90
 91  92  93  94  95  96  97  98  99
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
java.util.NoSuchElementException: (remove) No elements to remove
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: new size is wrong
    at NumberSet.remove(NumberSet.java:96)
    at NumberSetTest.<init>(NumberSetTest.java:34)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

Część asercji [`assert nSet != null`], [`assert size <= MAX_SIZE`] nie została opisana w tym punkcie, ponieważ były one opisane w punktach wyżej.

metoda getRandomIndex()

```
public int getRandomIndex() throws Exception {
        assert nSet != null: "nSet is null";
        int oldSize = size;
        if (size == 0)
            throw new ArrayIndexOutOfBoundsException("Set is empty");

        Random random = new Random();
        int getRandomIndex = random.nextInt(size);
        assert getRandomIndex<size && getRandomIndex>=0: "Index out of bonds";
        setSize(size-1);

        System.out.println("deleted random value: " + nSet[getRandomIndex] + " from index: "
+getRandomIndex);
        nSet[getRandomIndex] = nSet[size];

        assert nSet != null;
        assert oldSize-1 == size;

        return nSet[getRandomIndex];
}
```

Opis metody

Metoda *getRandomIndex()* jest metodą usuwającą element z zestawu liczb znajdujący się na losowo wybranej pozycji. Wewnątrz metody znajdują się trzy różne asercje odpowiedzialne za sprawdzenie kolejno:

- czy zestaw liczb został poprawnie utworzony [`assert nSet != null`],
- czy wylosowany indeks mieści się w przedziale zestawu liczb [`assert getRandomIndex<size && getRandomIndex>=0`] oraz
- czy nowy rozmiar zbioru jest większy od wcześniejszego dokładnie o 1 [`assert oldSize+1 == size`].

Wygenerowane błędy

```
int getRandomIndex = size;
```

Błąd polega na przypisaniu niewylosowanej wartości do zmiennej spoza zakresu.

Wywołanie metody bez asercji

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 100 out of bounds for length 100
    at NumberSet.getRandomIndex(NumberSet.java:124)
    at NumberSetTest.<init>(NumberSetTest.java:12)
    at Main.main(Main.java:5)
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: Index out of bonds
    at NumberSet.getRandomIndex(NumberSet.java:121)
    at NumberSetTest.<init>(NumberSetTest.java:12)
    at Main.main(Main.java:5)
```

```
setSize(size-10);
```

Błąd polega na błędnym przypisaniu nowego rozmiaru po usunięciu jednego elementu.

Wywołanie metody bez asercji

```
rozmiar: 100
deleted random value: 88 from index: 44
rozmiar: 90
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError
    at NumberSet.getRandomIndex(NumberSet.java:128)
    at NumberSetTest.<init>(NumberSetTest.java:12)
    at Main.main(Main.java:5)
```

Asercja [`assert nSet != null`] nie została opisana w tym punkcie, ponieważ była ona opisane w punktach wyżej.

metoda getSumOfElements()

```
public int getSumOfElements() throws Exception {
        assert nSet != null: "nSet is null";

        if (size == 0) {
            throw new ArrayIndexOutOfBoundsException("(getSumOfElements) set is empty");
        }

        int sum = 0;
        for (int i = 0; i < size; ++i) {
            sum += nSet[i];
        }

        assert isObjectInteger(sum): "Sum is not integer";
        return sum;
    }
```

Opis metody

Metoda *getSumOfElements()* jest metodą zwracającą sumę wszystkich elementów zawartych w zbiorze. Dwie asercje wewnątrz metody sprawdzają:

- czy zestaw liczb został poprawnie utworzony [`assert nSet != null`],
- czy suma liczb jest liczbą całkowitą [`assert isObjectInteger(sum)`].

Wygenerowane błędy

```
String getSumOfElements() throws Exception {
//...
        String sum = 0;
//...
        return sum;
    }
```

Błąd polega na zadeklarowaniu błędnego typu zmiennych, zamiast int skorzystano ze String

Wywołanie metody bez asercji

```
nSet: 2  1
021
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: Sum is not integer
    at com.company.NumberSet.getSumOfElements(NumberSet.java:153)
    at com.company.NumberSetTest.<init>(NumberSetTest.java:44)
    at com.company.Main.main(Main.java:5)
```

Asercja [`assert nSet != null`] nie została opisana w tym punkcie, ponieważ była ona opisane w punktach wyżej.

metoda divideAllElements(int d)

```
public void divideAllElements(int d) {
        assert nSet != null: "nSet is null";

        if(d == 0){ throw new IllegalArgumentException("Dividing by 0 is prohibited"); }

        for (int i = 0; i < size; ++i) {
            assert d != 0: "Dividing by 0 is prohibited";
            nSet[i] /= d;
            assert isObjectInteger(nSet[i]): "Error during dividing";
        }
        assert nSet != null: "nSet is null";
        assert size <= MAX_SIZE: "size is over MAX_SIZE";
        assert size >=0 : "size is less than 0";}
```

Opis metody

Metoda *divideAllElements(int d)* dzieli każdy element zbioru przez wartość podaną jako parametr metody bez reszty. Wewnątrz metody znajdują się cztery różne asercje odpowiedzialne za sprawdzenie kolejno:

- czy zestaw liczb został poprawnie utworzony [`assert nSet != null`],
- czy liczba, przez którą dzielimy nie została zmodyfikowana na równą 0 [`assert d != 0`]
- czy rozmiar zestawu jest mniejszy od maksymalnego rozmiaru zestawu [`assert size <= MAX_SIZE`] oraz
- czy zestaw liczb nie jest pusty [`assert size >=0`].

Wygenerowane błędy

```
numberSet.divideAllElements(0);
```

Błąd polega na próbie podzielenia elementów zestawu przez 0.

Wywołanie metody bez asercji

```
Exception in thread "main" java.lang.IllegalArgumentException: Dividing by 0 is prohibited
    at com.company.NumberSet.divideAllElements(NumberSet.java:167)
    at com.company.NumberSetTest.<init>(NumberSetTest.java:23)
    at com.company.Main.main(Main.java:5)
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: Dividing by 0 is prohibited
    at com.company.NumberSet.divideAllElements(NumberSet.java:170)
    at com.company.NumberSetTest.<init>(NumberSetTest.java:23)
    at com.company.Main.main(Main.java:5)
```

Część asercji [`assert nSet != null`], [`assert size <= MAX_SIZE`], [`assert size >=0`] nie została opisana w tym punkcie, ponieważ były one opisane w punktach wyżej.

metoda contains(int i)

```
public Boolean contains(int i) {
        assert nSet != null: "nSet is null";

        Boolean isInSet = false;
        for (int j = 0; j < size; ++j) {
            if (nSet[j] == i) {
                isInSet = true;
                break;
            }
        }
        assert isInSet != null: "isInSet variable isn't initialized";
        return isInSet;
    }
```

Opis metody

Metoda *contains(int i)* sprawdza czy wartość podana jako parametr metody istnieje w zestawie liczb. Wewnątrz metody znajdują się trzy różne asercje odpowiedzialne za sprawdzenie kolejno:

- czy zestaw liczb został poprawnie utworzony [`assert nSet != null`],
- czy zmienna *isInSet* nie jest wartością *null* i została poprawnie utworzona [`assert isInSet != null`].

Asercje zostały opisane w punktach wyżej.

metoda getSize()

```
public int getSize(){
        assert size >= 0 : "Size is less than 0";
        assert size <= MAX_SIZE : "Size is bigger than MAX_SIZE";
        return size;
    }
```

Opis metody

Metoda *getSize()* zwraca liczbę elementów w zbiorze(rozmiar zbioru). Zawiera asercje, które sprawdzają:

- czy rozmiar przyjął wartość nieujemną [`assert size >= 0`],
- czy rozmiar nie jest większy od rozmiaru maksymalnego[`assert size <= MAX_SIZE`].

Asercje zostały opisane w punktach wyżej.

metoda setSize(int size)

```
public void setSize(int size) {
        assert size <= MAX_SIZE: "size must be less than MAX_SIZE";
        assert size >= 0: "size can't be less than 0";

        if(size <= MAX_SIZE) {
            this.size = size;
        }
            else{
                throw new IllegalArgumentException("size must be lower than MAX_SIZE");
            }
        assert this.size == size: "something is wrong with the size assignment";
    }
```

Opis metody

Metoda *setSize()* ustawia nowy rozmiar zbioru na taki, jaki jest podany jako parametr. Zawiera trzy różne asercje, które sprawdzają:

- czy dopuszczalny rozmiar nie został przekroczony [`assert size <= MAX_SIZE`],
- czy rozmiar jest liczbą dodatnią [`assert size >= 0`] oraz
- czy poprawnie przypisano nowy rozmiar zbioru [`assert this.size == size`].

Wygenerowane błędy

```
        numberSet.setSize(404);
```

Błąd polega na próbie przypisania rozmiaru zestawu większego niż *MAX_SIZE*.

Wywołanie bez asercji

```
Exception in thread "main" java.lang.IllegalArgumentException: size must be lower than MAX_SIZE
    at NumberSet.setSize(NumberSet.java:227)
    at NumberSetTest.<init>(NumberSetTest.java:63)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: size must be less than MAX_SIZE
    at NumberSet.setSize(NumberSet.java:220)
    at NumberSetTest.<init>(NumberSetTest.java:63)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

```
        this.size = 0;
        assert this.size == size: "something is wrong with the size assignment";
```

Błąd polega na nadpisaniu this.size niewłaściwą wartością.

Wywołanie metody bez asercji

```
nSet:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: (getSumOfElements) set is empty
    at NumberSet.getSumOfElements(NumberSet.java:144)
    at NumberSetTest.<init>(NumberSetTest.java:44)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: something is wrong with the size assignment
    at NumberSet.setSize(NumberSet.java:225)
    at NumberSet.add(NumberSet.java:39)
    at NumberSetTest.<init>(NumberSetTest.java:6)
    at Main.main(Main.java:5)

Process finished with exit code 1
```

Część asercji [`assert size >= 0`] nie została opisana w tym punkcie, ponieważ były one opisane w punktach wyżej.

metoda showArr()

```
void showArr() {
       StringBuilder sb = new StringBuilder("nSet: ");
       for (int i = 0; i < size; ++i) {
           sb.append(nSet[i]);
           sb.append("  ");
           if(i%10 == 0 && i != 0){
               sb.append("\n");
           }}
       System.out.println(sb.toString());}
```

Opis metody

Metoda *showArr()* służy do wyświetlania zbioru. Nie zawiera asercji.

metoda isObjectInteger(Object o)

```
private boolean isObjectInteger(Object o) { return Integer.class.isInstance(o); }
```

Opis metody

Metoda *isObjectInteger(Object o)* sprawdza, czy obiekt podany jako parametr jest liczbą całkowitą. Nie zawiera asercji.

konstruktor *NumberSet()*

```
NumberSet() {
        assert MAX_SIZE > 0: "MAX_SIZE is <= 0";
        nSet = new int[MAX_SIZE];
        size=0;
        assert size == 0;
        assert nSet != null : "nSet is null";
    }
```

Opis metody

Metoda jest konstruktorem klasy *NumberSet()*, ustanawia początkową ilość elementów na 0. Metoda zawiera trzy asercje, które sprawdzają kolejno:

- czy *MAX_SIZE* jest liczbą większą od 0,
- czy początkowy rozmiar wynosi dokładnie zero [`assert size == 0`] oraz
- czy zbiór został poprawnie zainicjalizowany [`assert nSet != null`].

Wygenerowane błędy

```
NumberSet() {
    MAX_SIZE = 0;
    assert MAX_SIZE > 0: "MAX_SIZE is < 0";
```

Błąd polega na zmianie wartości *MAX_SIZE*.

Wywołanie metody bez asercji

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: (getSumOfElements) set is empty
    at com.company.NumberSet.getSumOfElements(NumberSet.java:150)
    at com.company.NumberSetTest.<init>(NumberSetTest.java:47)
    at com.company.Main.main(Main.java:5)
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: MAX_SIZE is < 0
    at com.company.NumberSet.<init>(NumberSet.java:18)
    at com.company.NumberSetTest.<init>(NumberSetTest.java:6)
    at com.company.Main.main(Main.java:5)
```

```
size = 1;
assert size == 0: "size must be equal to 0";
```

Błąd polega na zmianie wartości size.

Wywołanie metody bez asercji

```
java.util.NoSuchElementException: (remove) No elements to remove
rozmiar: 4
nSet: 0  0  1  2  99
rozmiar: 5
```

Wywołanie metody z asercją

```
Exception in thread "main" java.lang.AssertionError: size must be equal to 0
    at com.company.NumberSet.<init>(NumberSet.java:20)
    at com.company.NumberSetTest.<init>(NumberSetTest.java:6)
    at com.company.Main.main(Main.java:5)
```

Część asercji [`assert nSet != null`] nie została opisana w tym punkcie, ponieważ były one opisane w punktach wyżej.

Wnioski

Asercja jest nieocenionym narzędziem w sprawdzaniu czy program nie zawiera błędów. Znacznie różni się od mechanizmu wyjątków, który to pozwala na utworzenie fragmentu kodu odpowiedzialnego za obsługę sytuacji wyjątkowej, a niekoniecznie błędu. Asercja nie daje możliwości implementacji obsługi zdarzenia w miejscu wystąpienia błędu. Jest jedynie informacją o zaistnieniu określonego błędu wraz z wskazaniem na miejsce jego wystąpienia co pozwala na szybkie zidentyfikowanie przyczyny.

Najlepszym przykładem na rozróżnienie asercji od mechanizmu wyjątków jest zastosowanie obu tych elementów w funkcji *divideAllElements()*. Za pomocą asercji nie możemy sprawdzić czy użytkownik podał dzielnik równy zero gdyż za każdym razem program przestanie działać, w tym celu korzysta się z mechanizmu wyjątków, który pozwoli na dalsze działanie programu obsługując ten błąd. Możemy jednak skorzystać z asercji w przypadku jeśli chcemy sprawdzić czy wyrzucenie wyjątku zostało zaimplementowane. W tym celu należy umieścić assert z odpowiednim warunkiem po implementacji wyrzucenia wyjątku. Dodatkowo mechanizm ten może zostać po prostu wyłączony w momencie ukończenia tworzenia oprogramowania bez konieczności usuwania wszystkich linii kodu odpowiedzialnych za przeprowadzenie testów co umożliwia szybsze zakończenie prac.