МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

КАФЕДРА «Програмна інженерія та інтелектуальні технології управління»

ЗВІТ

з лабораторної роботи № 2

з навчальної дисципліни

“ОСНОВИ ПРОГРАМУВАННЯ JAVA (ДВВ)”

ВИКОНАЛА

Студентка групи КН-222а

Інеса РЕПЕШКО

ПЕРЕВІРИВ

доц. каф. ПІІТУ к.т.н

доцент Валерій ВОЛОВЩИКОВ

Харків 2023

# **Тема роботи**

“Робота з типами-посиланнями”.

# **1.1 Завдання №1 “Індивідуальне завдання”**

Створити двовимірний масив цілих чисел з кількістю рядків і стовпців, вказаних в таблиці. Заповнити масив випадковими додатними значеннями відповідно до правила, наведеного в таблиці. Заповнити одновимірний масив рядків (String) довжини, вказаної в таблиці. Кожен елемент містить повтоення певного символу кількістю, вказаною в таблиці. Здійснити сортування масиву рядків за правилом, вказаним в таблиці.

Умова завдання для варіанту 24 (номер 24 за порядком у списку групи) наведено нижче.



Рисунок 1.1 - Умова завдання № 1 варіант 24

Вивести отриманий масив рядків.

Наприклад, припустимо двовимірний масив цілих містить такі числа:

|  |  |
| --- | --- |
| 3 | 7 |
| 18 | 4 |
| 19 | 2 |

Якщо, припустимо, кількість елементів масиву рядків (String) відповідає кількості рядків масиву чисел, а мінімальні елементи масиву – це кількість повторення певного символу, і символ, який треба повторювати, це 'N', ми отримаємо такий масив рядків:

|  |
| --- |
| NNN |
| NNNN |
| NN |

Реалізувати два підходи: традиційний, побудований на циклах і роботі з окремими елементами і через функції класу Arrays (без циклів).

Додати до класу та окремих функцій коментарі Javadoc.

**1.2 Набір тестових даних завдання № 1**

**1.2.1 Тестовий випадок № 1 “Випадкові числа та символ 'N'”**

Опис: Тестування роботи з масивами з використанням традиційного та функціонального підходу зі створенням масивів за замовчуванням: заповнення масиву чисел розміром 5x3 випадковими додатними цілими непарними числами від 3 до 31 та заповнення масиву рядків символом 'N'.

Тестові дані:

* Значення масиву чисел *intArray*: випадкові додатні цілі непарні числа від 3 до 31.
* Символи для заповнення рядків масиву рядків *stringArray*: 'N'.

Очікуваний результат: Успішне завершення виконання програми. Виведення у консоль повідомлення з результатами з формованим масивом чисел, сформованим масивом рядків та відсортованим масивом рядків.

|  |
| --- |
| 1. Traditional approach:  Array of integers:  [17, 15, 13]  [9, 13, 15]  [21, 15, 7]  [31, 15, 19]  [15, 3, 25]  Array of strings before sorting:  NNNNNNNNNNNNN  NNNNNNNNN  NNNNNNN  NNNNNNNNNNNNNNN  NNN  Array of strings after sorting:  NNNNNNNNNNNNNNN  NNNNNNNNNNNNN  NNNNNNNNN |

|  |
| --- |
| NNNNNNN  NNN    2. Functional approach:  Array of integers:  [3, 19, 19]  [5, 3, 23]  [23, 29, 13]  [15, 9, 21]  [17, 11, 3]  Array of strings before sorting:  NNN  NNN  NNNNNNNNNNNNN  NNNNNNNNN  NNN  Array of strings after sorting:  NNNNNNNNNNNNN  NNNNNNNNN  NNN  NNN  NNN |

**1.2.2 Тестовий випадок № 2 “Валідний тестовий масив та різні символи”**

Опис: Тестування роботи з масивами з використанням традиційного та функціонального підходу з використанням тестового масиву чисел розміром 5x3 з псевдовипадковими додатними цілими непарними числами від 3 до 31 та заповнення масиву рядків символами 'A', 'B', 'C', 'B', 'C'.

Тестові дані:

* Значення масиву чисел *intArray*:

{{17, 31, 13},

{7, 29, 11},

{29, 17, 11},

{25, 9, 3},

{9, 27, 15}}

* Символи для заповнення рядків масиву рядків *stringArray*: 'A', 'B', 'C', 'B', 'C'.

Очікуваний результат: Успішне завершення виконання програми. Значення масиву чисел та масиву рядків мають бути однаковими при застосуванні обох підходів. Виведення у консоль повідомлення з результатами з тестовим масивом чисел, сформованим масивом рядків та відсортованим масивом рядків:

|  |
| --- |
| 1. Traditional approach:  [17, 31, 13]  [7, 29, 11]  [29, 17, 11]  [25, 9, 3]  [9, 27, 15]  Array of strings before sorting:  AAAAAAAAAAAAA  BBBBBBB  CCCCCCCCCCC  BBB  CCCCCCCCC  Array of strings after sorting:  CCCCCCCCCCC  CCCCCCCCC  BBBBBBB  BBB  AAAAAAAAAAAAA    2. Functional approach:  Array of integers:  [17, 31, 13]  [7, 29, 11]  [29, 17, 11]  [25, 9, 3]  [9, 27, 15]  Array of strings before sorting:  AAAAAAAAAAAAA |

|  |
| --- |
| BBBBBBB  CCCCCCCCCCC  BBB  CCCCCCCCC  Array of strings after sorting:  CCCCCCCCCCC  CCCCCCCCC  BBBBBBB  BBB  AAAAAAAAAAAAA |

**1.2.3 Тестовий випадок № 3 “Невалідний масив чисел за кількістю рядків”**

Опис: Тестування роботи з масивами з використанням традиційного підходу з використанням тестового масиву чисел розміром 6x3 з псевдовипадковими додатними цілими непарними числами від 3 до 31 та заповнення масиву рядків символом 'N'.

Тестові дані:

* Значення масиву чисел *intArray*:

{{17, 31, 13},

{7, 29, 11},

{29, 17, 11},

{25, 9, 3},

{9, 27, 15},

{17, 31, 13}}

* Символи для заповнення рядків масиву рядків *stringArray*: 'N'.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| Traditional approach:  A two-dimensional array with an incorrect number of rows was passed. The number of rows should be 5. |

**1.2.4 Тестовий випадок № 4 “Невалідний масив чисел за кількістю стовпців”**

Опис: Тестування роботи з масивами з використанням традиційного підходу з використанням тестового масиву чисел розміром 5x4 з псевдовипадковими додатними цілими непарними числами від 3 до 31 та заповнення масиву рядків символом 'N'.

Тестові дані:

* Значення масиву чисел *intArray*:

{{17, 31, 13, 19},

{7, 29, 11, 31},

{29, 17, 11, 31},

{25, 9, 3, 15},

{9, 27, 15, 27}}

* Символи для заповнення рядків масиву рядків *stringArray*: 'N'.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| Traditional approach: A two-dimensional array with an incorrect number of columns was passed. The number of columns should be 3. |

**1.2.5 Тестовий випадок № 5 “Невалідний масив чисел за значеннями”**

Опис: Тестування роботи з масивами з використанням функціонального підходу з використанням тестового масиву чисел розміром 5x3 з псевдовипадковими цілими числами та заповнення масиву рядків символом 'N'.

Тестові дані:

* Значення масиву чисел *intArray*:

{{17, 31, 13},

{7, 29, 1},

{29, 11, 11},

{25, -9, 3},

{89, 9, 15}}

* Символи для заповнення рядків масиву рядків *stringArray*: 'N'.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| Functional approach:  There are invalid numbers in the array. The array must be filled with random positive odd integers from 3 to 31 including. |

**1.3 Програмний код завдання № 1**

Лістинг 1.1 - Програмний код реалізований у файлі “OperateArrays.java”

|  |
| --- |
| package task1;  import java.util.Arrays; import java.util.Random;  /\*\*  \* The {***@code*** *OperateArrays} class performs some operations on a two-dimensional array of integers*  *\* and an array of strings.*  *\* <p> It provides a traditional approach that uses loops and individual elements*  *\* and a functional approach that uses functions of the Arrays class (without loops).*  *\*/* *class* OperateArrays {  */\*\* The number of rows in the array of integers and the number of items in the array of strings. \*/* private static final int *NUMBER\_OF\_ROWS* = 5;   */\*\* The number of columns in the array of integers. \*/* private static final int *NUMBER\_OF\_COLUMNS* = 3;   */\*\* The character used to fill in strings. \*/* private static final char *CHARACTER* = 'N';   */\*\* The array of strings with test data for testing the operation on arrays with filling with multiple characters.\*/* private static final char[] *CHARACTERS* = new char[] {'A', 'B', 'C', 'B', 'C'};   */\*\**  *\* Fills the provided two-dimensional array of integers with random positive odd numbers between 3 and 31.*  *\** ***@param*** *intArray two-dimensional array of integers.*  *\*/* |

|  |
| --- |
| private static void fillIntArray(int[][] intArray) {  Random random = new Random();   for (int i = 0; i < intArray.length; i++) {  for (int j = 0; j < intArray[i].length; j++) {  intArray[i][j] = random.nextInt(15) \* 2 + 3;  }  }  }   */\*\**  *\* Prints the provided two-dimensional array of integers to the console.*  *\** ***@param*** *intArray two-dimensional array of integers.*  *\*/* private static void printIntArray(int[][] intArray) {  for (int[] row : intArray) {  System.*out*.print('[');   for (int j = 0; j < row.length - 1; j++) {  System.*out*.print(row[j] + ", ");  }   System.*out*.println(row[row.length - 1] + "]");  }  }   */\*\**  *\* Finds the minimum value in the provided row of an array of integers.*  *\** ***@param*** *intArrayRow row of an array of integers;*  *\** ***@return*** *minimum value in the provided row of an array of integers.*  *\*/* private static int findMinValueInRow(int[] intArrayRow) {  int minValue = intArrayRow[0];   for (int i = 1; i < intArrayRow.length; i++) {  if (intArrayRow[i] < minValue) {  minValue = intArrayRow[i];  }  } |

|  |
| --- |
| return minValue;  }   */\*\**  *\* Fills the provided array of strings with items consisting of the {****@value*** *#CHARACTER} character or multiple*  *\* characters from the '{****@link*** *#CHARACTERS}' array according to {****@code*** *fillWithMultiChars} flag based on the*  *\* minimum value in each row of the array of integers.*  *\** ***@param*** *stringArray array of strings;*  *\** ***@param*** *intArray two-dimensional array of integers;*  *\** ***@param*** *fillWithMultiChars flag to check which characters to fill the strings in the array of strings with.*  *\*/* private static void fillStringArray(String[] stringArray, int[][] intArray, boolean fillWithMultiChars) {  if (fillWithMultiChars) {  String[] characterStrings = new String[stringArray.length];   for (int i = 0; i < stringArray.length; i++) {  characterStrings[i] = String.*valueOf*(*CHARACTERS*[i]);  stringArray[i] = characterStrings[i].repeat(*findMinValueInRow*(intArray[i]));  }  } else {  String characterString = String.*valueOf*(*CHARACTER*);   for (int i = 0; i < stringArray.length; i++) {  stringArray[i] = characterString.repeat(*findMinValueInRow*(intArray[i]));  }  }  }   */\*\**  *\* Sorts the provided array of strings in reverse alphabetical order using bubble sort.*  *\** ***@param*** *stringArray array of strings.* |

|  |
| --- |
| *\*/* private static void sortStringArrayByReverseAlphabet(String[] stringArray) {  boolean isUnsorted = true;   while (isUnsorted) {  isUnsorted = false;   for (int i = 0; i < stringArray.length - 1; i++) {  if (stringArray[i].compareTo(stringArray[i + 1]) < 0) {  String tempString = stringArray[i];  stringArray[i] = stringArray[i + 1];  stringArray[i + 1] = tempString;  isUnsorted = true;  }  }  }  }   */\*\**  *\* Prints the provided array of strings to the console.*  *\** ***@param*** *stringArray array of strings.*  *\*/* private static void printStringArray(String[] stringArray) {  for (String s : stringArray) {  System.*out*.println(s);  }  }   */\*\**  *\* Verifies the given two-dimensional array of integers for compliance with the following requirements:*  *\* <ul><li> The number of all rows of the given array must be equal to {****@value*** *#NUMBER\_OF\_ROWS}.</li>*  *\* <li> The number of all columns must be equal to {****@value*** *#NUMBER\_OF\_COLUMNS}.</li>*  *\* <li> The array must be filled with random positive odd integers from 3 to 31.</li>*  *\* <li> If any of the requirements are not met, an error message is printed and the program is closed.</li></ul>* |

|  |
| --- |
| *\** ***@param*** *intArray two-dimensional array of integers;*  *\** ***@param*** *isTraditionalApproach flag to check which approach should be used for verification.*  *\*/* private static void verifyIntArray(int[][] intArray, boolean isTraditionalApproach) {  if (intArray.length != *NUMBER\_OF\_ROWS*) {  System.*err*.printf("A two-dimensional array with an incorrect number of rows was passed. " +  "\nThe number of rows should be %d. Try again!", *NUMBER\_OF\_ROWS*);  System.*exit*(1);  } else {  if (isTraditionalApproach) {  for (int[] row : intArray) {  if (row.length != *NUMBER\_OF\_COLUMNS*) {  System.*err*.printf("A two-dimensional array with an incorrect number of columns was passed. " +  "\nThe number of columns should be %d. Try again!", *NUMBER\_OF\_COLUMNS*);  System.*exit*(1);  }   for (int element : row) {  if (element < 3 || element > 31 || element % 2 == 0) {  System.*err*.println("There are invalid numbers in the array. \nThe array must be filled " +  "with random positive odd integers from 3 to 31 including.");  System.*exit*(1);  }  }  }  } else {  Arrays.*stream*(intArray).forEach(row -> {  if (row.length != *NUMBER\_OF\_COLUMNS*) {  System.*err*.printf("A two-dimensional array with an incorrect number of columns was passed. " +  "\nThe number of columns should be %d. Try again!", |

|  |
| --- |
| *NUMBER\_OF\_COLUMNS*);  System.*exit*(1);  }   Arrays.*stream*(row).forEach(element -> {  if (element < 3 || element > 31 || element % 2 == 0) {  System.*err*.println("There are invalid numbers in the array. \nThe array must be filled " +  "with random positive odd integers from 3 to 31 including. Try again!");  System.*exit*(1);  }  });  });  }  }  }   */\*\**  *\* Operates on arrays using the traditional approach.*  *\* <p> Creates the two-dimensional array of integers and fills it with random positive odd numbers. Prints*  *\* the array of integers. Fills the array of strings with {****@value*** *#CHARACTER} character based on the minimum*  *\* value in each row of the array of integers. Prints the array of strings. Sorts the array of strings in*  *\* reverse alphabetical order and prints it.*  *\*/* static void usingTraditionalApproach() {  int[][] intArray = new int[*NUMBER\_OF\_ROWS*][*NUMBER\_OF\_COLUMNS*];  *fillIntArray*(intArray);  System.*out*.println("Array of integers:");  *printIntArray*(intArray);   String[] stringArray = new String[*NUMBER\_OF\_ROWS*];  *fillStringArray*(stringArray, intArray, false);  System.*out*.println("Array of strings before sorting:");  *printStringArray*(stringArray); |

|  |
| --- |
| *sortStringArrayByReverseAlphabet*(stringArray);  System.*out*.println("Array of strings after sorting:");  *printStringArray*(stringArray);  }   */\*\**  *\* Operates on arrays using the traditional approach and using the test array of integers and filling the array*  *\* of strings with multiple characters or specified character.*  *\* <p> Verifies and prints the array of integers. Fills the array of strings with the {****@value*** *#CHARACTER} character*  *\* or multiple characters from the '{****@link*** *#CHARACTERS}' array according to {****@code*** *fillWithMultiChars} flag*  *\* based on the minimum value in each row of the array of integers. Prints the array of strings. Sorts the array*  *\* of strings in reverse alphabetical order and prints it.*  *\** ***@param*** *intArray two-dimensional array of integers;*  *\** ***@param*** *fillWithMultiChars flag to check which characters to fill the strings in the array of strings with.*  *\*/* static void usingTraditionalApproach(int[][] intArray, boolean fillWithMultiChars) {  *verifyIntArray*(intArray, true);  System.*out*.println("Array of integers:");  *printIntArray*(intArray);   String[] stringArray = new String[*NUMBER\_OF\_ROWS*];  *fillStringArray*(stringArray, intArray, fillWithMultiChars);  System.*out*.println("Array of strings before sorting:");  *printStringArray*(stringArray);   *sortStringArrayByReverseAlphabet*(stringArray);  System.*out*.println("Array of strings after sorting:");  *printStringArray*(stringArray);  }   */\*\**  *\* Operates on arrays using the functional approach.* |

|  |
| --- |
| *\* <p> Creates the two-dimensional array of integers and fills it with random positive odd numbers. Prints*  *\* the array of integers. Fills the array of strings with {****@value*** *#CHARACTER} character based on the minimum*  *\* value in each row of the array of integers. Prints the array of strings. Sorts the array of strings in*  *\* reverse alphabetical order and prints it.*  *\*/* static void usingFunctionalApproach() {  int[][] intArray = new int[*NUMBER\_OF\_ROWS*][*NUMBER\_OF\_COLUMNS*];  Random random = new Random();  Arrays.*stream*(intArray).forEach(row -> Arrays.*setAll*(row, i -> random.nextInt(15) \* 2 + 3));  System.*out*.println("Array of integers:");  Arrays.*stream*(intArray).forEach(row -> System.*out*.println(Arrays.*toString*(row)));   String characterString = String.*valueOf*(*CHARACTER*);  String[] stringArray = new String[*NUMBER\_OF\_ROWS*];  Arrays.*setAll*(stringArray, j -> {  int minNumberInRow = Arrays.*stream*(intArray[j]).min().orElse(0);  return characterString.repeat(minNumberInRow);  });  System.*out*.println("Array of strings before sorting:");  Arrays.*stream*(stringArray).forEach(System.*out*::println);   Arrays.*sort*(stringArray, (s1, s2) -> -s1.compareTo(s2));  System.*out*.println("Array of strings after sorting:");  Arrays.*stream*(stringArray).forEach(System.*out*::println);  }    */\*\* Operates on arrays using the functional approach and using the test array of integers and filling the array*  *\* of strings with multiple characters or specified character.*  *\* <p> Verifies and prints the array of integers. Fills the array of strings with the {****@value*** *#CHARACTER} character*  *\* or multiple characters from the '{****@link*** *#CHARACTERS}' array according to* |

|  |
| --- |
| *{****@code*** *fillWithMultiChars} flag*  *\* based on the minimum value in each row of the array of integers. Prints the array of strings. Sorts the array*  *\* of strings in reverse alphabetical order and prints it.*  *\** ***@param*** *intArray two-dimensional array of integers;*  *\** ***@param*** *fillWithMultiChars flag to check which characters to fill the strings in the array of strings with.*  *\*/* static void usingFunctionalApproach(int[][] intArray, boolean fillWithMultiChars) {  *verifyIntArray*(intArray, false);  System.*out*.println("Array of integers:");  Arrays.*stream*(intArray).forEach(row -> System.*out*.println(Arrays.*toString*(row)));   String[] stringArray = new String[*NUMBER\_OF\_ROWS*];   if (fillWithMultiChars) {  String[] characterStrings = new String[*NUMBER\_OF\_ROWS*];  Arrays.*setAll*(characterStrings, i -> String.*valueOf*(*CHARACTERS*[i]));  Arrays.*setAll*(stringArray, j -> {  int minNumberInRow = Arrays.*stream*(intArray[j]).min().orElse(0);  return characterStrings[j].repeat(minNumberInRow);  });  } else {  String characterString = String.*valueOf*(*CHARACTER*);  Arrays.*setAll*(stringArray, j -> {  int minNumberInRow = Arrays.*stream*(intArray[j]).min().orElse(0);  return characterString.repeat(minNumberInRow);  });  }   System.*out*.println("Array of strings before sorting:");  Arrays.*stream*(stringArray).forEach(System.*out*::println);   Arrays.*sort*(stringArray, (s1, s2) -> -s1.compareTo(s2));  System.*out*.println("Array of strings after sorting:");  Arrays.*stream*(stringArray).forEach(System.*out*::println);  } |

|  |
| --- |
| */\*\**  *\* Calls static methods of the {****@code*** *OperateArrays} class demonstrating the traditional and functional approaches*  *\* with different test data. The {****@code*** *args} are not used.*  *\** ***@param*** *args command-line arguments.*  *\*/* public static void main(String[] args) {  System.*out*.println("1.1. Operate arrays using the traditional approach with creating default arrays:");  *usingTraditionalApproach*();   System.*out*.println("\n1.2. Operate arrays using the functional approach with creating default arrays:");  *usingFunctionalApproach*();   int[][] validTestArray = new int[][] {  {17, 31, 13},  {7, 29, 11},  {29, 17, 11},  {25, 9, 3},  {9, 27, 15}  };  System.*out*.println("\n\n2.1. Operate arrays using the traditional approach and using the test array of " +  "integers \nand filling the array of strings with multiple characters:");  *usingTraditionalApproach*(validTestArray, true);   System.*out*.println("\n2.2. Operate arrays using the functional approach and using the test array of " +  "integers \nand filling the array of strings with multiple characters:");  *usingFunctionalApproach*(validTestArray, true);   /\*  int[][] invalidTestArray1 = new int[][] {  {17, 31, 13},  {7, 29, 11},  {29, 17, 11},  {25, 9, 3}, |

|  |
| --- |
| {9, 27, 15},  {17, 31, 13}  };  System.out.println("\n\n3. Operate arrays using the traditional approach " +  "\nand using the test array with invalid number of rows:");  usingTraditionalApproach(invalidTestArray1, false);  \*/   /\*  int[][] invalidTestArray2 = new int[][] {  {17, 31, 13, 19},  {7, 29, 11, 31},  {29, 17, 11, 31},  {25, 9, 3, 15},  {9, 27, 15, 27}  };  System.out.println("\n\n4. Operate arrays using the traditional approach " +  "\nand using the test array with invalid number of columns:");  usingTraditionalApproach(invalidTestArray2, false);  \*/   int[][] invalidTestArray3 = new int[][] {  {17, 31, 13},  {7, 29, 1},  {29, 11, 11},  {25, -9, 3},  {89, 9, 15}  };  System.*out*.println("\n\n5. Operate arrays using the functional approach " +  "\nand using the test array with invalid data:");  *usingFunctionalApproach*(invalidTestArray3, true);  } } |

Сформований Javadoc для всієї лабораторної роботи міститься у прикріплених файлах до роботи. Структура файлів в проекті наступна:

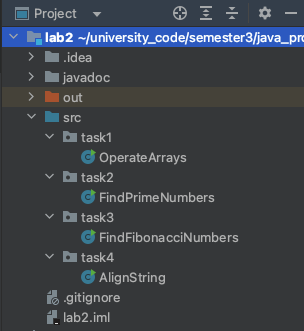


Рисунок 1.2 - Структура файлів в проекті “lab2”

**1.4 Екранні форми за результатами роботи програмного коду**

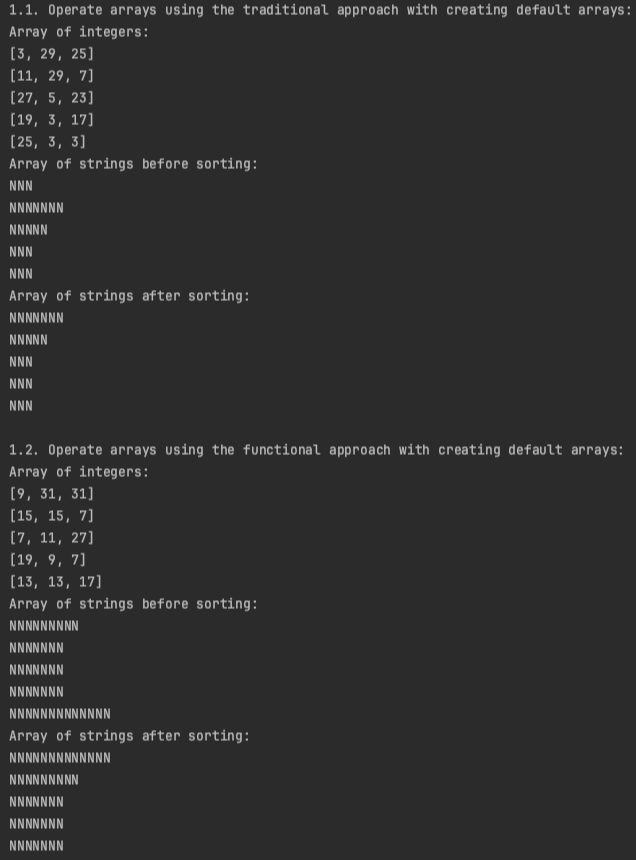


Рисунок 1.3 - Результати роботи програмного коду відповідно до тестового випадку № 1 “Випадкові числа та символ 'N'”

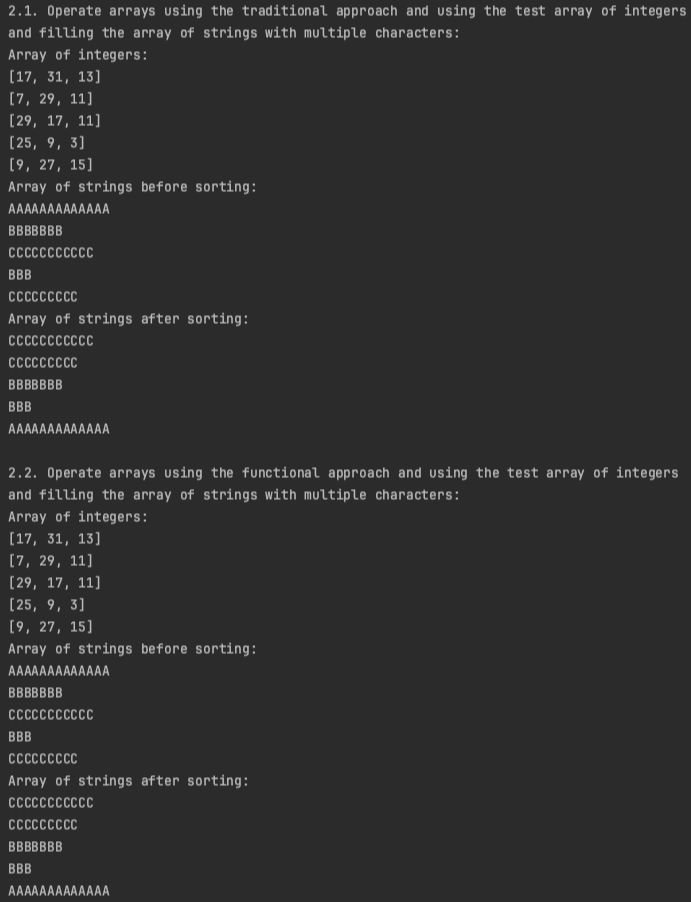


Рисунок 1.4 - Результати роботи програмного коду відповідно до тестового випадку № 2 “Валідний тестовий масив та різні символи”

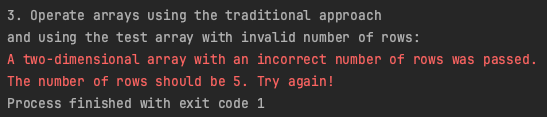


Рисунок 1.5 - Результати роботи програмного коду відповідно до тестового випадку № 3 “Невалідний масив чисел за кількістю рядків”

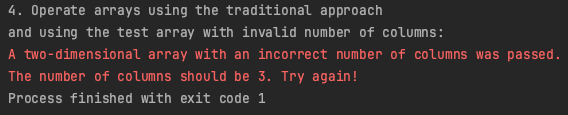


Рисунок 1.6 - Результати роботи програмного коду відповідно до тестового випадку № 4 “Невалідний масив чисел за кількістю стовпців”

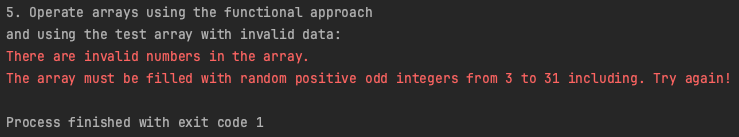


Рисунок 1.7 - Результати роботи програмного коду відповідно до тестового випадку № 5 “Невалідний масив чисел за значеннями”

**2.1 Завдання № 2 “Ератосфенове решето”**

Заповнити масив із трьохсот цілих чисел послідовними додатними значеннями. Замінити всі значення, що не є простими числами, деяким від'ємним значенням. Для цього послідовно виключати всі числа – дільники інших чисел. Вивести на екран додатні значення, що залишилися, (прості числа).

У програмі не застосовувати ділення та знаходження залишку від ділення.

**2.2 Набір тестових даних завдання № 2**

**2.2.1 Тестовий випадок № 1 “Валідна кількість елементів масиву: 300”**

Тестові дані:

* Кількість елементів масиву цілих чисел з послідовними додатними значеннями: 300.
* Значення елементів масиву цілих чисел з послідовними додатними значеннями із заданим розміром:

|  |
| --- |
| Array of integers with consecutive positive values in the range from 1 to 300:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 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, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300] |

Очікуваний результат: Успішне завершення виконання програми. Виведення у консоль повідомлення з результатами пошуку простих чисел із заданого масиву чисел:

|  |
| --- |
| The prime numbers from array:  2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293 |

**2.3 Програмний код завдання № 2**

Лістинг 2.1 - Програмний код реалізований у файлі “FindPrimeNumbers.java”

|  |
| --- |
| package task2;  import java.util.Arrays;  /\*\*  \* The {***@code*** *FindPrimeNumbers} class implements the Eratosthenes sieve algorithm for finding prime numbers without*  *\* division and getting the remainder from division and prints the found prime numbers.*  *\*/* *class* FindPrimeNumbers {  */\*\* The number of array elements to search for prime numbers. \*/* private static final int *ARR\_ELEMS\_NUMBER* = 300;  */\*\* The negative value that is used to indicate non-prime numbers in the array. \*/* private static final int *NEGATIVE\_NUMBER* = -1;   */\*\**  *\* Creates array of integers filled with consecutive positive values in the range from 1 to*  *\* {****@value*** *#ARR\_ELEMS\_NUMBER}.*  *\** ***@return*** *array of integers filled with consecutive positive values.*  *\*/* static int[] createArrayOfNumbers() {  int[] arrayOfNumbers = new int[*ARR\_ELEMS\_NUMBER*];  Arrays.*setAll*(arrayOfNumbers, i -> i + 1);   return arrayOfNumbers;  }   */\*\**  *\* Finds prime numbers in the range from 1 to {****@value*** *#ARR\_ELEMS\_NUMBER} by the Eratosthenes Sieve algorithm.*  *\* The {****@code*** *newArray} array is filled with consecutive positive integer numbers and numbers that are not prime*  *\* are replaced with the {****@value*** *#NEGATIVE\_NUMBER} value.* |

|  |
| --- |
| *\** ***@param*** *newArray array of integers filled with consecutive positive values that must be sorted.*  *\*/* static void findPrimeNumbers(int[] newArray) {  newArray[0] = *NEGATIVE\_NUMBER*;   for (int i = 2; i \* i <= *ARR\_ELEMS\_NUMBER*; i++) {  if (newArray[i - 1] != *NEGATIVE\_NUMBER*) {  for (int j = i + i; j <= *ARR\_ELEMS\_NUMBER*; j += i) {  if (newArray[j - 1] != *NEGATIVE\_NUMBER*) {  newArray[j - 1] = *NEGATIVE\_NUMBER*;  }  }  }  }  }   */\*\**  *\* Prints the prime numbers as formatted text. The {****@value*** *#NEGATIVE\_NUMBER} values in the {****@code*** *newArray}*  *\* array are not printed.*  *\** ***@param*** *newArray sorted array of prime numbers.*  *\*/* static void printPrimeNumbers(int[] newArray) {  int counter = 0;   for (int element : newArray) {  if (element != *NEGATIVE\_NUMBER*) {  if (counter == 21) {  System.*out*.println();  counter = 0;  }   System.*out*.print(element + "\t");  counter++;  }  }  } |

|  |
| --- |
| */\*\**  *\* Calls static methods of the {****@link*** *FindPrimeNumbers} class to find and print prime numbers. The {****@code*** *args}*  *\* are not used.*  *\** ***@param*** *args command-line arguments.*  *\*/* public static void main(String[] args) {  int[] newArray = *createArrayOfNumbers*();  *findPrimeNumbers*(newArray);  System.*out*.println("The prime numbers from an array of numbers from 1 to 300:");  *printPrimeNumbers*(newArray);  } } |

**2.4 Екранні форми за результатами роботи програмного коду**

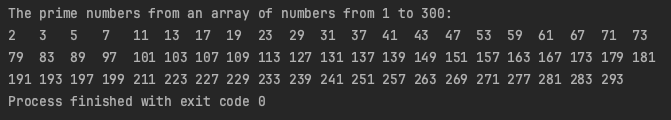


Рисунок 2.1 - Результати роботи програмного коду відповідно до тестового випадку № 1 “Валідна кількість елементів масиву: 300”

**3.1 Завдання № 3 “Знаходження чисел Фібоначчі”**

Реалізувати функцію обчислення чисел Фібоначчі (до 92-го числа включно) з використанням допоміжного масиву (статичного поля). Параметр функції – номер числа Фібоначчі. Пошук чисел Фібоначчі здійснюється за таким правилом :

*F(1) = F(2) = 1;F(n) = F(n - 2) + F(n - 1)* (3.1)

Під час першого виклику функції масив заповнюється до необхідного числа. Під час наступних викликів число або повертається з масиву, або обчислюється з використанням останніх двох чисел, що зберігаються у масиві з подальшим заповненням масиву. Використовувати тип **long** для представлення чисел.

Здійснити тестування функції для різних значень номерів, що вводяться у довільному порядку.

**3.2 Набір тестових даних завдання № 3**

Послідовність виконання тестових випадків має бути збережена відповідно до послідовності наведених нижче тестових випадків.

**3.2.1 Тестовий випадок № 1 “Число Фібоначчі за валідним індексом 9”**

Тестові дані:

* Індекс числа Фібоначчі: “9”.

Очікуваний результат: Успішне обчислення результатів програмою:

* Обчислено числа Фібоначчі від індекса “3” до вказаного індекса “9” включно.
* У масиві чисел Фібоначчі міститься 9 чисел.

Виведення у консоль повідомлення з результатами обчислення значення числа Фібоначчі за вказаним індексом та поточних обчислених чисел масиву чисел Фібоначчі:

|  |
| --- |
| Fibonacci number at the index 9:  34  Array of current Fibonacci numbers:  [1, 1, 2, 3, 5, 8, 13, 21, 34] |

**3.2.2 Тестовий випадок № 2 “Число Фібоначчі за валідним індексом 5”**

Тестові дані:

* Індекс числа Фібоначчі: “5”.

Очікуваний результат: Успішне обчислення результатів програмою:

* Число Фібоначчі за вказаним індексом “5” взято вже з обчисленого масиву чисел Фібоначчі.
* У масиві чисел Фібоначчі міститься 9 чисел.

Виведення у консоль повідомлення з результатами обчислення значення числа Фібоначчі за вказаним індексом та поточних обчислених чисел масиву чисел Фібоначчі:

|  |
| --- |
| Fibonacci number at the index 5:  5  Array of current Fibonacci numbers:  [1, 1, 2, 3, 5, 8, 13, 21, 34] |

**3.2.3 Тестовий випадок № 3 “Число Фібоначчі за валідним індексом 21”**

Тестові дані:

* Індекс числа Фібоначчі: “21”.

Очікуваний результат: Успішне обчислення результатів програмою:

* Обчислено числа Фібоначчі від індекса “10” до вказаного індекса “21” включно.
* У масиві чисел Фібоначчі міститься 21 число.

Виведення у консоль повідомлення з результатами обчислення значення числа Фібоначчі за вказаним індексом та поточних обчислених чисел масиву чисел Фібоначчі:

|  |
| --- |
| The Fibonacci number at the index 21:  10946  The array of current Fibonacci numbers:  [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946]. |

**3.2.4 Тестовий випадок № 4 “Число Фібоначчі за валідним індексом 92”**

Тестові дані:

* Індекс числа Фібоначчі: “92”.

Очікуваний результат: Успішне обчислення результатів програмою:

* Обчислено числа Фібоначчі від індекса “22” до вказаного індекса “92” включно.
* У масиві чисел Фібоначчі міститься 92 числа.

Виведення у консоль повідомлення з результатами обчислення значення числа Фібоначчі за вказаним індексом та поточних обчислених чисел масиву чисел Фібоначчі:

|  |
| --- |
| The Fibonacci number at the index 92:  7540113804746346429  The array of current Fibonacci numbers:  [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733, 1134903170, 1836311903, 2971215073, 4807526976, 7778742049, 12586269025, 20365011074, 32951280099, 53316291173, 86267571272, 139583862445, 225851433717, 365435296162, 591286729879, 956722026041, 1548008755920, 2504730781961, 4052739537881, 6557470319842, 10610209857723, 17167680177565, 27777890035288, 44945570212853, 72723460248141, 117669030460994, 190392490709135, 308061521170129, 498454011879264, |

|  |
| --- |
| 806515533049393, 1304969544928657, 2111485077978050, 3416454622906707, 5527939700884757, 8944394323791464, 14472334024676221, 23416728348467685, 37889062373143906, 61305790721611591, 99194853094755497, 160500643816367088, 259695496911122585, 420196140727489673, 679891637638612258, 1100087778366101931, 1779979416004714189, 2880067194370816120, 4660046610375530309, 7540113804746346429]. |

**3.2.5 Тестовий випадок № 5 “Число Фібоначчі за невалдіним індексом 100”**

Тестові дані:

* Індекс числа Фібоначчі: “100”.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| The Fibonacci number at the index 100:  Invalid value of index. The allowed index value is from 1 to 92 including. |

**3.3 Програмний код завдання № 3**

Лістинг 3.1 - Програмний код реалізований у файлі “FindFibonacciNumbers.java”

|  |
| --- |
| package task3;  /\*\*  \* The {***@code*** *FindFibonacciNumbers} class provides a function for computing of Fibonacci numbers (up to the 92nd*  *\* integer) using an auxiliary array (static field). <p> The search for Fibonacci numbers is carried out according to*  *\* the following rule: {****@code*** *F(1) = F(2) = 1; F(n) = F(n - 2) + F(n - 1)}.*  *\* <p> The class supports a static {****@code*** *FIB\_NUMS\_ARRAY} array for storing Fibonacci numbers. At the first call,*  *\* the array is filled until the required number. At subsequent calls, the number either returns from the array,*  *\* or is calculated using the last two numbers stored in an array.*  *\* <p> Also provides a function to print the Fibonacci array.*  *\*/* *class* FindFibonacciNumbers {  */\*\* The number of elements in the {****@code*** *FIB\_NUMS\_ARRAY} array. Also the maximum possible value that will be found. \*/* private static final int *ARR\_ELEMS\_NUMBER* = 92;  */\*\* The index of the last calculated Fibonacci number in the {****@code*** *FIB\_NUMS\_ARRAY} array. \*/* private static int *lastCalculated* = 2;  */\*\* The auxiliary static array (static field) to store Fibonacci numbers. \*/* private static final long[] *FIB\_NUMS\_ARRAY* = new long[*ARR\_ELEMS\_NUMBER*];  static {  *FIB\_NUMS\_ARRAY*[0] = 1;  *FIB\_NUMS\_ARRAY*[1] = 1;  }   */\*\**  *\* Finds the Fibonacci number at a given {****@code*** *numberIndex}.*  *\** ***@param*** *numberIndex Fibonacci number's index;*  *\** ***@return*** *Fibonacci number for a given index.*  *\*/* static long findFibonacciNumbers(int numberIndex) {  if (numberIndex < 1 || numberIndex > *ARR\_ELEMS\_NUMBER*) {  System.*err*.println("Invalid value of index. The allowed index value is from 1 to 92 including. Try again!"); |

|  |
| --- |
| System.*exit*(1);  }   if (*lastCalculated* < numberIndex) {  for (int n = *lastCalculated*; n < numberIndex; n++) {  *FIB\_NUMS\_ARRAY*[n] = *FIB\_NUMS\_ARRAY*[n - 2] + *FIB\_NUMS\_ARRAY*[n - 1];  }   System.*out*.printf("There were calculated Fibonacci numbers from %d to %d including.\n",  ++*lastCalculated*, numberIndex);  *lastCalculated* = numberIndex;  } else {  System.*out*.println("The Fibonacci number was taken from the array.");  }   return *FIB\_NUMS\_ARRAY*[numberIndex - 1];  }   */\*\**  *\* Prints Fibonacci numbers form the {****@code*** *FIB\_NUMS\_ARRAY} array up to and including the {****@code*** *lastCalculated}*  *\* number in array.*  *\*/* static void printFibonacciArray() {  System.*out*.printf("The array of current Fibonacci numbers (index from 1 to %d including):\n", *lastCalculated*);  int counter = 0;   for (int i = 0; i < *lastCalculated*; i++) {  if (counter == 6) {  System.*out*.println();  counter = 0;  }   System.*out*.printf("%-20d", *FIB\_NUMS\_ARRAY*[i]);  counter++;  } |

|  |
| --- |
| }   */\*\**  *\* Calls the static methods of the {****@link*** *FindFibonacciNumbers} class to find the Fibonacci number at the specified*  *\* index. Prints this number and the current values of the {****@code*** *FIB\_NUMS\_ARRAY} array. The {****@code*** *args} are*  *\* not used.*  *\** ***@param*** *args command-line arguments*  *\*/* public static void main(String[] args) {  int index = 9;  System.*out*.printf("The Fibonacci number at the index %d:\t%d\n", index, *findFibonacciNumbers*(index));  *printFibonacciArray*();  System.*out*.println("\n");   index = 5;  System.*out*.printf("The Fibonacci number at the index %d:\t%d\n", index, *findFibonacciNumbers*(index));  *printFibonacciArray*();  System.*out*.println("\n");   index = 21;  System.*out*.printf("The Fibonacci number at the index %d:\t%d\n", index, *findFibonacciNumbers*(index));  *printFibonacciArray*();  System.*out*.println("\n");   index = 92;  System.*out*.printf("The Fibonacci number at the index %d:\t%d\n", index, *findFibonacciNumbers*(index));  *printFibonacciArray*();  System.*out*.println("\n");   index = 100;  System.*out*.printf("The Fibonacci number at the index %d:\n", index);  *findFibonacciNumbers*(index);  } } |

**3.4 Екранні форми за результатами роботи програмного коду**

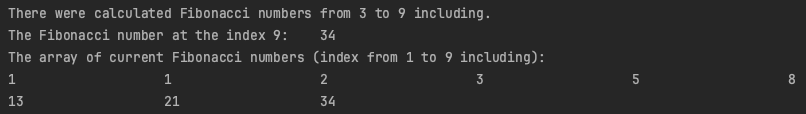


Рисунок 3.1 - Результати роботи програмного коду відповідно до тестового випадку № 1 “Число Фібоначчі за валідним індексом 9”

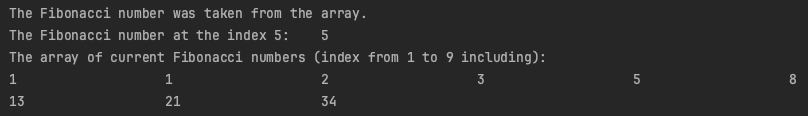


Рисунок 3.2 - Результати роботи програмного коду відповідно до тестового випадку № 2 “Число Фібоначчі за валідним індексом 5”

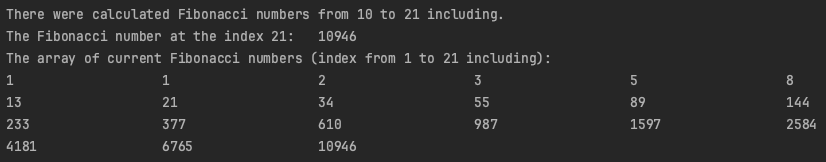
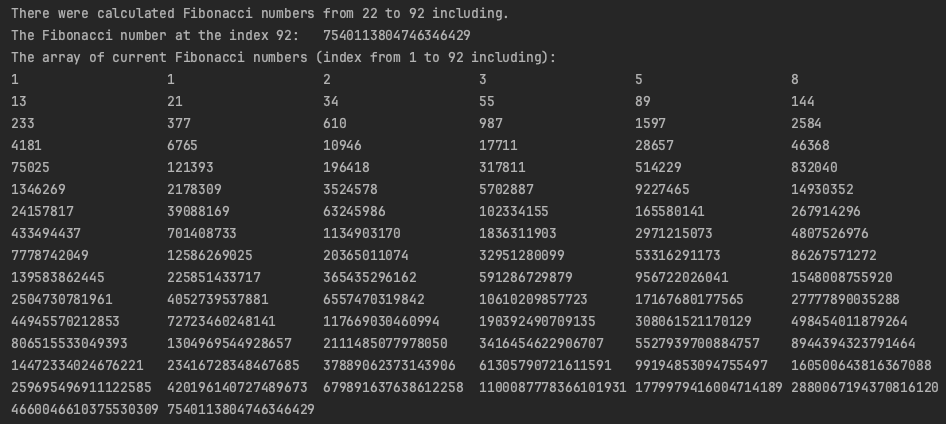


Рисунок 3.3 - Результати роботи програмного коду відповідно до тестового випадку № 3 “Число Фібоначчі за валідним індексом 21”

 Рисунок 3.4 - Результати роботи програмного коду відповідно до тестового випадку № 4 “Число Фібоначчі за валідним індексом 92”

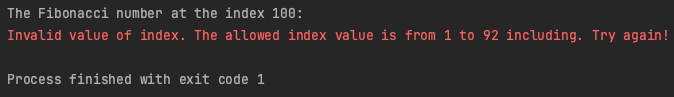


Рисунок 3.5 - Результати роботи програмного коду відповідно до тестового випадку № 5 “Число Фібоначчі за невалідним індексом 100”

## **4.1 Завдання № 4 “Вирівнювання рядку”**

Прочитати аргумент командного рядка і додати в нього пропуски (space characters) так, щоб довжина рядка дорівнювала заданому числу. Пропуски додавати рівномірно між словами (за можливості).

**4.2 Набір тестових даних завдання № 4**

**4.2.1 Тестовий випадок № 1 “Валідний рядок з 4 слів і 1 пробілом між словами”**

Тестові дані:

* Початковий рядок з різних слів та 1 пробілом між всіма словами: “test string test string”.
* Необхідна довжина вирівняного рядка: 50.

Очікуваний результат: Успішне завершення виконання програми. Сформований вирівняний рядок має довжину 50 символів, з яких 10 пробілів між кожним словом. Виведення у консоль повідомлення з результатами у вигляді початкового рядка, необхідної довжини та вирівняного рядка:

|  |
| --- |
| Required length: 50  Initial string: "test string test string"  Aligned string: "test string test string" |

**4.2.2 Тестовий випадок № 2 “Валідний рядок з 1 словом”**

Тестові дані:

* Початковий рядок з одного слова без пробілв: “testString”.
* Необхідна довжина вирівняного рядка: 25.

Очікуваний результат: Успішне завершення виконання програми. Сформований вирівняний рядок має довжину 25 символів, з яких 7 пробілів перед словом та 8 пробілів після слова. Виведення у консоль повідомлення з результатами у вигляді початкового рядка, необхідної довжини та вирівняного рядка:

|  |
| --- |
| Required length: 25  Initial string: "testString"  Aligned string: " testString " |

**4.2.3 Тестовий випадок № 3 “Валідний рядок з 3 словами з цифрами та різною кількістю пробілів між ними”**

Тестові дані:

* Початковий рядок з цифрами замість слів та різною кількістю пробілів між словами: “3 4 0”.
* Необхідна довжина вирівняного рядка: 15.

Очікуваний результат: Успішне завершення виконання програми. Сформований вирівняний рядок має довжину 15 символів, з яких по 6 пробілів між словами. Виведення у консоль повідомлення з результатами у вигляді початкового рядка, необхідної довжини та вирівняного рядка:

|  |
| --- |
| Required length: 15  Initial string: "3 4 0"  Aligned string: "3 4 0". |

**4.2.4 Тестовий випадок № 4 “Валідний рядок без слів”**

Тестові дані:

* Початковий рядок пустий: “”.
* Необхідна довжина вирівняного рядка: 7.

Очікуваний результат: Успішне завершення виконання програми. Сформований вирівняний рядок має довжину 7 символів, з яких всі пробіли. Виведення у консоль повідомлення з результатами у вигляді початкового рядка, необхідної довжини та вирівняного рядка:

|  |
| --- |
| Required length: 7  Initial string: ""  Aligned string: " ". |

**4.2.5 Тестовий випадок № 5 “Валідний рядок з 3 словами та різною кількістю пробілів між ними та валідною довжиною, що = довжині рядка”**

Тестові дані:

* Початковий рядок з різними словами та різною кількістю пробілів між словами: “test string test”.
* Необхідна довжина вирівняного рядка: 27.

Очікуваний результат: Успішне завершення виконання програми. Сформований вирівняний рядок має довжину 27 символів, з яких по 7 пробілів між 1 та 2 словами та 6 пробілів між 2 та 3 словом. Виведення у консоль повідомлення з результатами у вигляді початкового рядка, необхідної довжини та вирівняного рядка:

|  |
| --- |
| Required length: 7  Initial string: "test string test"  Aligned string: "test string test". |

**4.2.6 Тестовий випадок № 6 “Невалідний рядок із пробілами на початку/кінці рядка і валідна довжина”**

Тестові дані:

* Початковий рядок з двох слів з пробілами на початку та кінці рядку: “ test String ”.
* Необхідна довжина вирівняного рядка: 20.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| Error: Invalid string passed.  The string must begin and end with words, not spaces. |

**4.2.7 Тестовий випадок № 7 “Валідний рядок з 3 слів та невалідна довжина < рядка”**

Тестові дані:

* Початковий рядок з 3 слів з довжиною 18 символів з різню кількістю пробілів між словами: “test String test”.
* Необхідна довжина вирівняного рядка: 15.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| Error: Invalid string or the required string length was passed.  The length of the initial string must be equal to or less than the required string length. |

**4.2.8 Тестовий випадок № 8 “Без одного з аргументів командного рядка”**

Тестові дані:

* Початковий рядок з 3 слів з довжиною 18 символів з різню кількістю пробілів між словами: “string12345. test”.
* Необхідна довжина вирівняного рядка не задана.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку:

|  |
| --- |
| Error: Incorrect arguments were passed.  The arguments should be as follows: <string> <required string length>. |

**4.3 Програмний код завдання № 4**

Лістинг 4.1 - Програмний код реалізований у файлі “AlignString.java”

|  |
| --- |
| package task4;  /\*\*  \* The {***@code*** *AlignString} class performs reading a string from the command line and adding spaces so that the string*  *\* length is equal to the given value. Spaces are added evenly between words (if possible).*  *\*/* *class* AlignString {  */\*\**  *\* Verifies the passed {****@code*** *args} command line arguments.*  *\* <ul><li> If fewer or more than 2 arguments are passed, an error message is printed and the program is*  *\* exited. </li></ul>*  *\** ***@param*** *args command-line arguments.*  *\* \*/* static void verifyArgs(String[] args) {  if (args.length != 2) {  System.*err*.println("Incorrect arguments were passed. The arguments should be as follows: " +  "<string> <required string length>. Try again!");  System.*exit*(1);  }  }   */\*\**  *\* Gets the required length of the aligned string from the passed {****@code*** *arg} command line argument, verifies it*  *\* and returns it.*  *\* <ul><li> If the passed required length is less than or equal to zero, an error message is printed and*  *\* the program is exited. </li></ul>*  *\** ***@param*** *arg command line argument with the required length of the aligned string;*  *\** ***@return*** *required length of the aligned string.*  *\* \*/* static int getLength(String arg) {  int requiredLength = Integer.*parseInt*(arg); |

|  |
| --- |
| if (requiredLength <= 0) {  System.*err*.println("Invalid required string length was passed.\n" +  "The desired string length must be greater than zero. Try again!");  System.*exit*(1);  }   return requiredLength;  }   */\*\**  *\* Gets the initial string from the passed {****@code*** *arg} command line argument, verifies it and returns it.*  *\* <ul><li> If the passed initial string starts or ends with spaces or is longer than the required length,*  *\* an error message is printed and the program is exited. </li></ul>*  *\** ***@param*** *requiredLength required length of the aligned string;*  *\** ***@param*** *arg command line argument with the initial string to be aligned;*  *\** ***@return*** *initial string to be aligned.*  *\* \*/* static String getString(int requiredLength, String arg) {  if (arg.startsWith(" ") || arg.endsWith(" ")) {  System.*err*.println("Invalid string passed.\nThe string must begin and end with words, not spaces. " +  "Try again!");  System.*exit*(1);  }   if (arg.length() > requiredLength) {  System.*err*.println("Invalid string or the required string length was passed.\n" +  "The length of the initial string must be equal to or less than the required string length. " +  "Try again!");  System.*exit*(1);  }   return arg;  } |

|  |
| --- |
| */\*\**  *\* Adds spaces between words (if possible) evenly in the {****@code*** *initialString}.*  *\* <ul><li> If the string consists of a single word or none, spaces are added evenly at the start and end of*  *\* the string.</li>*  *\* <li> If the string consists of several words, spaces are added evenly between each word.</li></ul>*  *\** ***@param*** *wordArray array of separate words from the initial string;*  *\** ***@param*** *totalSpacesNum total number of spaces to be added to the string for alignment;*  *\** ***@return*** *string with added spaces evenly between words (if possible).*  *\*/* private static String addSpacesToString(String[] wordArray, int totalSpacesNum) {  if (wordArray.length == 1) {  int startSpacesNum;  startSpacesNum = totalSpacesNum / 2;   return " ".repeat(startSpacesNum) + wordArray[0] + " ".repeat(totalSpacesNum - startSpacesNum);  } else {  int betweenWordsSpacesNum = totalSpacesNum / (wordArray.length - 1);  int remainingSpacesNum = totalSpacesNum % (wordArray.length - 1);  StringBuilder alignedString = new StringBuilder();  alignedString.append(wordArray[0]);   for (int i = 1; i < wordArray.length; i++) {  int spacesNumber = betweenWordsSpacesNum;   if (remainingSpacesNum != 0) {  spacesNumber++;  remainingSpacesNum--;  }   alignedString.append(" ".repeat(spacesNumber));  alignedString.append(wordArray[i]);  }   return alignedString.toString(); |

|  |
| --- |
| }  }   */\*\**  *\* Aligns the {****@code*** *initialString} by removing extra spaces and adding spaces evenly using the*  *\* {****@code*** *addSpacesToString} method.*  *\* <p> The string is converted into a sentence with only one space between words by removing the extra spaces.*  *\* Separate words are extracted from the generated string into an array. The total number of spaces to be added*  *\* to the string to align it to the required length is calculated. To add spaces to a string for alignment,*  *\* {****@code*** *addSpacesToString} method is called.*  *\** ***@param*** *initialString input string to be aligned;*  *\** ***@param*** *requiredLength required length of the aligned string;*  *\** ***@return*** *aligned string.*  *\*/* static String alignString(int requiredLength, String initialString) {  while (initialString.contains(" ")) {  initialString = initialString.replaceAll(" ", " ");  }   String[] wordArray = initialString.split(" ");  int totalSpacesNum = requiredLength - initialString.length() + wordArray.length - 1;   return *addSpacesToString*(wordArray, totalSpacesNum);  }   */\*\**  *\* Reads, aligns and prints a string.*  *\* <p> Reads the initial string and the required length of the aligned string from the command line. Validates*  *\* the passed command line arguments. Aligns the initial string to the required length. Prints the required length,*  *\* the initial string, the aligned string and the length of the aligned string.*  *\** ***@param*** *args command-line arguments.*  *\*/* |

|  |
| --- |
| public static void main(String[] args) {  *verifyArgs*(args);  int requiredLength = *getLength*(args[1]);  String initialString = *getString*(requiredLength, args[0]);  String alignedString = *alignString*(requiredLength, initialString);  System.*out*.printf("The required length of the aligned string:\t%d.\n", requiredLength);  System.*out*.printf("The initial string to be aligned:\t\t\t\"%s\".\n", initialString);  System.*out*.printf("The aligned string:\t\t\t\t\t\t\t\"%s\".\n", alignedString);  System.*out*.printf("The length of the aligned string:\t\t\t%d.\n", alignedString.length());  } } |

**4.4 Екранні форми за результатами роботи програмного коду**

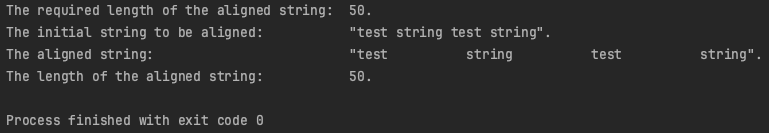


Рисунок 4.1 - Результати роботи програмного коду відповідно до тестового випадку № 1 “Валідний рядок з 4 слів і 1 пробілом між словами”

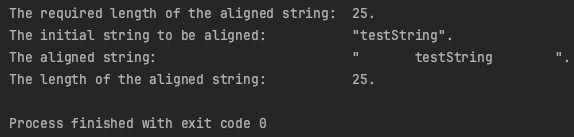


Рисунок 4.2 - Результати роботи програмного коду відповідно до тестового випадку № 2 “Валідний рядок з 1 словом”

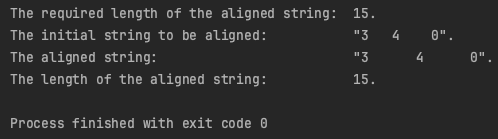


Рисунок 4.3 - Результати роботи програмного коду відповідно до тестового випадку № 3 “Валідний рядок з 3 словами з цифрами та різною кількістю пробілів між ними”

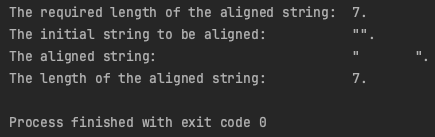


Рисунок 4.4 - Результати роботи програмного коду відповідно до тестового випадку № 4 “Валідний рядок без слів”

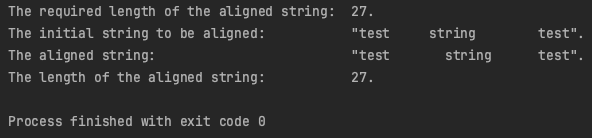


Рисунок 4.5 - Результати роботи програмного коду відповідно до тестового випадку № 5 “Валідний рядок з 3 словами та різною кількістю пробілів між ними та валідною довжиною, що = довжині рядка”

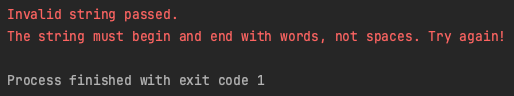


Рисунок 4.6 - Результати роботи програмного коду відповідно до тестового випадку № 6 “Невалідний рядок із пробілами на початку/кінці рядка і валідна довжина”

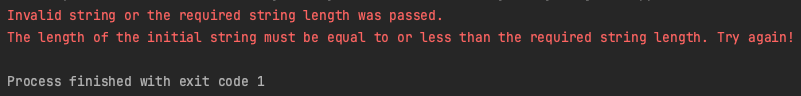


Рисунок 4.7 - Результати роботи програмного коду відповідно до тестового випадку № 7 “Валідний рядок з 3 слів та невалідна довжина < рядка”

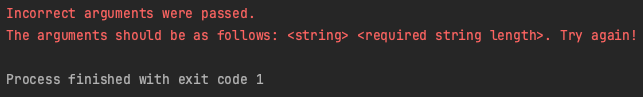


Рисунок 4.8 - Результати роботи програмного коду відповідно до тестового випадку № 8 “Без одного з аргументів командного рядка”

**Висновки до роботи:**

У ході виконання даної лабораторної роботи використано різні засоби, методи та технології для ефективної реалізації поставлених завдань із використанням мови програмування Java.

Під час виконання поставлених завдань опановано наступні функціональні можливості мови програмування Java:

* Для зберігання даних було використано тип-посилання масиви та клас *String*.
* Для заповнення масивів випадковими та псевдовипадковими значеннями відбувалось за допомогою класу *java.util.Random*.
* Для роботи з масивами при функціональному підході використовувались статичні методи класу *java.util.Arrays*.
* Реалізовано різні алгоритми пошуку елементів та сортування масивів чисел та рядків.
* Для модифікації вмісту рядків використовався стандартний клас *StringBuilder*.
* Реалізована робота з аргументами командного рядка.

Використання цих можливостей дозволяє навчитись створювати більш функціональні, ефективні та зрозумілі програми на мові Java, що є важливим для розробки якісного програмного забезпечення та розв'язання різноманітних завдань.