Лабораторна робота №6.

Серіалізація/десеріалізація об’єктів. Бібліотека класів користувача

**Мета роботи:**

* Тривале зберігання та відновлення стану об’єктів.
* Ознайомлення з принципами серіалізації/десеріалізації об’єктів.
* Використання бібліотек класів користувача.

**1. Завдання / вимоги до лабораторної роботи:**

1. Реалізувати і продемонструвати тривале зберігання/відновлення [раніше розробленого контейнера](https://oop-khpi.github.io/#task_05) за допомогою [серіалізації/десеріалізації](https://docs.oracle.com/javase/8/docs/technotes/guides/serialization/index.html).
2. Обмінятися відкомпільованим (без початкового коду) службовим класом (Utility Class) рішення задачі [л.р. №3](https://oop-khpi.github.io/" \l "task_03_app) з іншим студентом (визначає викладач).
3. Продемонструвати послідовну та вибіркову обробку елементів розробленого контейнера за допомогою власного і отриманого за обміном службового класу.
4. Реалізувати та продемонструвати порівняння, сортування та пошук елементів у контейнері.
5. Розробити консольну програму та забезпечити діалоговий режим роботи з користувачем для демонстрації та тестування рішення.
   1. **Розробник:**

*студент Литвин Ігнатій Ігоревич; КІТ-26А; Варіант №7*

* 1. **Рекомендації**

1. [Java Object Serialization Specification](https://docs.oracle.com/javase/8/docs/platform/serialization/spec/serialTOC.html).
2. [Java object serialization. Tutorial](http://www.vogella.com/tutorials/JavaSerialization/article.html).
3. [Пять вещей, которые вы не знали о сериализации](http://www.ibm.com/developerworks/ru/library/j-5things1/).

**2. Опис програми**

Програма реалізована у вигляді інтерактивного консольного вікна з діалоговим режимом роботи з користувачем.

Основне призначення: демонстрація послідовної та вибіркової обробки елементів розробленого контейнера за допомогою власного і отриманого за обміном службового класу.

Програма працює лише з текстом написаним на латинкою. Для обробки даних використовуються класи-утиліти. Регулярних вирази не використовуються при виконанні завдання.

Для збереження початкових даних завдання л.р. №3 у вигляді масиву рядків з можливістю додавання, видалення і зміни елементів було розроблено клас-контейнер, що ітерується. Додатково у класі-контейнері реалізовано сортування та пошук елементів у контейнері.

Також було реалізовано тривале зберігання/відновлення раніше розробленого контейнера за допомогою серіалізації/десеріалізації.

**2.1 Засоби**

Серіалізація об'єкту це здатність об'єкту зберігати повну копію його і будь-яких інших об'єктів на які він посилається, використовуючи потік виводу (наприклад, у зовнішній файл). Таким чином, об'єкт може бути відтворений з серіалізованої (збереженої) копії трохи пізніше, коли це буде потрібно.

Серіалізація об'єктів, як нова можливість введена в JDK 1.1, надає функцію для перетворення груп або окремих об'єктів, в потік бітів або масив байтів, для зберігання або передачі по мережі. І як було сказано, даний потік бітів або масив байтів, можна перетворити назад в об'єкти Java. Головним чином це відбувається автоматично завдяки класам ObjectInputStream і ObjectOutputStream. Програміст може вирішити реалізувати цю можливість, шляхом реалізації інтерфейсу Serializable при створенні класу.

Серіалізація зберігає інформацію про те, якого типу об'єкт, щоб в подальшому, при десеріалізациі, ця інформація використовувалася для відтворення точного типу об'єкта, яким він був.

Деякі класи системного рівня, такі як Thread, OutputStream та його підкласи, і Socket, не серіалізуються. Якщо клас містить такі об'єкти, вони повинні позначатися як " transient". 

Процес серіалізації також відомий як **маршалинг** об'єкту, десеріалізація ж відома як **демаршалинг.**

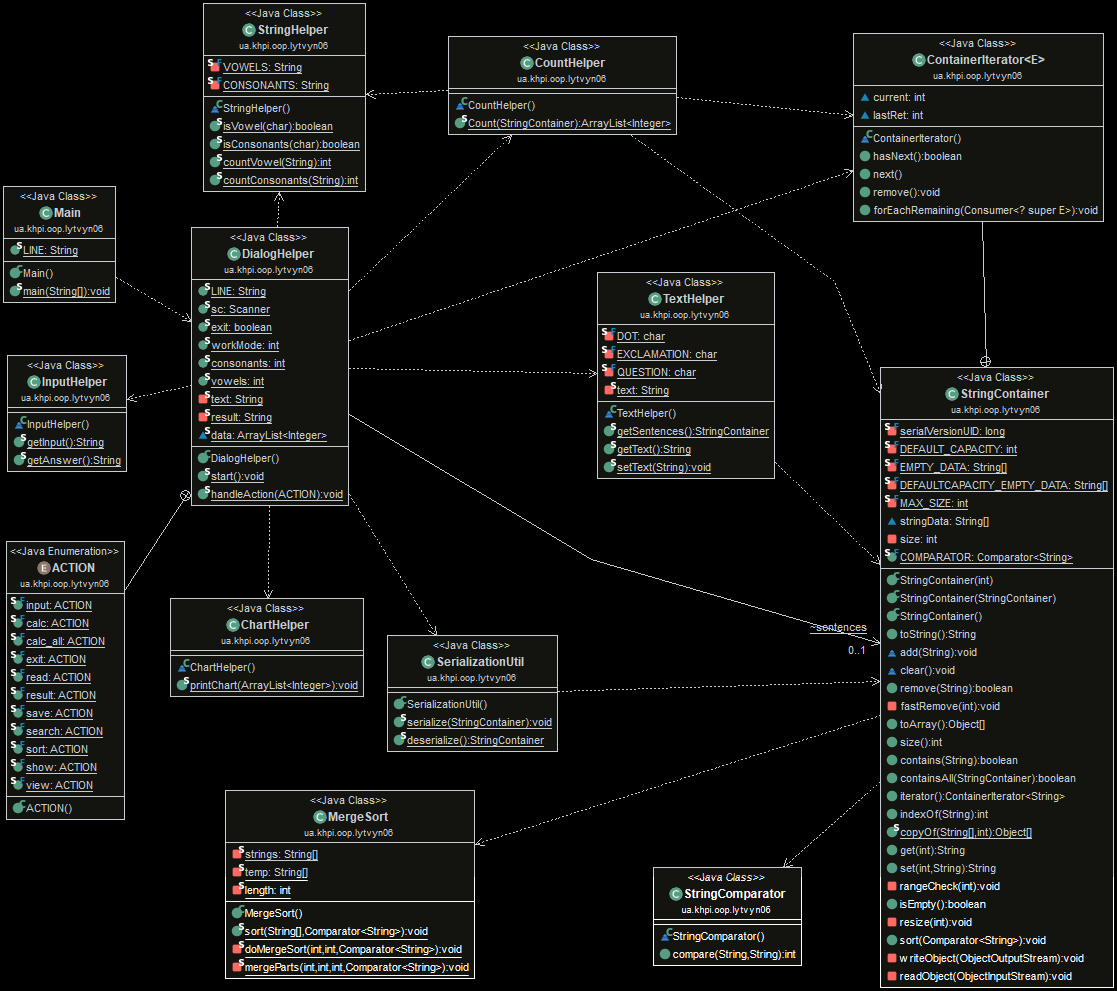
**2.2 Ієрархія та структура класів**

Рисунок 1 « Ієрархія та структура класів»

**2.3 Важливі фрагменти програми**

/\*\*

\* Виконує серіалізацію (збереження до файлу) отриманого контейнеру типу

\* <tt>StringСontainer</tt>.

\*

\* **@param** sentences контейнер, що буде серіалізовано

\*/

**public** **static** **void** **serialize**(**StringСontainer** sentences) {

**ObjectOutputStream** **out** = **null**;

**try** {

/\* Відкриваємо потік для запису \*/

out = **new** ObjectOutputStream(

**new** BufferedOutputStream(**new** FileOutputStream("Data.ser")));

/\* Записуємо контейнер \*/

out.writeObject(sentences);

**System**.***out***.println(" Записано: " + sentences);

} **catch** (**IOException** **ex**) {

ex.printStackTrace();

/\* Обов'язково зачиняємо потік \*/

} **finally** {

**if** (out != **null**)

**try** {

out.close();

} **catch** (**IOException** **ex**) {

ex.printStackTrace();

}

}

}

/\*\*

\* Виконує десеріалізацію (відновлення з файлу) контейнеру типу

\* <tt>StringСontainer</tt>.

\*

\* **@return** <tt>sentences</tt> контейнер, що було відновлено з файлу

\*/

**public** **static** **StringСontainer** **deserialize**() {

**StringСontainer** **sentences** = **new** StringСontainer();

**ObjectInputStream** **in** = **null**;

**try** {

/\* Відкриваємо потік для зчитування \*/

in = **new** ObjectInputStream(

**new** BufferedInputStream(**new** FileInputStream("Data.ser")));

/\* Відновлюємо контейнер \*/

sentences = **new** StringСontainer((**StringСontainer**) in.readObject());

**System**.***out***.println(" Зчитано: " + sentences);

} **catch** (**IOException** **ex**) {

ex.printStackTrace();

} **catch** (**Exception** **ex**) {

ex.printStackTrace();

/\* Обов'язково зачиняємо потік \*/

} **finally** {

**if** (in != **null**)

**try** {

in.close();

} **catch** (**IOException** **ex**) {

ex.printStackTrace();

}

}

**return** sentences;

}

}

/\*\*

\* class MergeSort - Утилітарний клас, що забезпечує сортування масивів.

\*

\* **@author** student Lytvyn I.I. KIT-26A

\*

\*/

**public** **class** **MergeSort** {

/\*\*

\* Масив для сортування

\*/

**private** **static** **String**[] *strings*;

/\*\*

\* Тимчасовий масив для сортування

\*/

**private** **static** **String**[] *temp*;

/\*\*

\* Довжина масиву

\*/

**private** **static** **int** *length*;

/\*\*

\* Виконує сортування отриманного масиву

\*

\* **@param** array масив, що буде відсортовано

\* **@param** comparator компоратор, що буде використовуватися при порівнянні

\*/

**public** **static** **void** **sort**(**String**[] array,

Comparator<? **super** **String**> comparator) {

*strings* = array;

*length* = array.length;

*temp* = **new** **String**[*length*];

*doMergeSort*(0, *length* - 1, comparator);

}

/\*\*

\* Відсортовує масив елементів типу String за алгоритмом MergeSort

\*

\* **@param** low початковий індекс елементів масиву

\* **@param** high кінцевий індекс елементів масиву

\* **@param** comparator компоратор, що буде використовуватися при порівнянні

\*/

**private** **static** **void** **doMergeSort**(**int** low, **int** high,

Comparator<? **super** **String**> comparator) {

/\*

\* Перевірка, чи початок менший, ніж кінець, якщо так, тоді масив

\* сортується

\*/

**if** (low < high) {

/\* Отримуємо середній індекс \*/

**int** **middle** = low + (high - low) / 2;

/\* Сортуємо ліву частину \*/

*doMergeSort*(low, middle, comparator);

/\* Сортуємо праву частину \*/

*doMergeSort*(middle + 1, high, comparator);

/\* Сортуємо обидві частини \*/

*mergeParts*(low, middle, high, comparator);

}

}

/\*\*

\* Виконує сортування частин

\*

\* **@param** low початковий індекс елементів масиву

\* **@param** middle середній індекс елементів масиву

\* **@param** high кінцевий індекс елементів масиву

\*/

**private** **static** **void** **mergeParts**(**int** low, **int** middle, **int** high,

Comparator<? **super** **String**> comparator) {

/\* Копіювання обох частин до допоміжного масиву \*/

**for** (**int** **i** = low; i <= high; i++) {

*temp*[i] = *strings*[i];

}

**int** **i** = low;

**int** **j** = middle + 1;

**int** **k** = low;

/\*

\* Копіювання найменших значень зліва або праворуч назад до початкового

\* масиву

\*/

**while** (i <= middle && j <= high) {

**if** (comparator.compare(*temp*[i], *temp*[j]) <= 0) {

*strings*[k] = *temp*[i];

i++;

} **else** {

*strings*[k] = *temp*[j];

j++;

}

k++;

}

/\*

\* Копіювання решти частин лівої частини масиву в цільовий масив.

\* Оскільки ми сортуємо на місці, будь-які залишкові елементи з правого

\* боку вже знаходяться в правильному положенні.

\*/

**while** (i <= middle) {

*strings*[k] = *temp*[i];

k++;

i++;

}

}

}

/\*\*

\* Компаратор, що впорядковує {@code String} об'єкти лексикографічно, ігноруючи

\* випадкові відмінності.

\*

\* **@author** student Lytvyn I.I. KIT-26A

\*

\*/

**class** **StringComparator** **implements** Comparator<String> {

/\*\*

\* Порівнює два аргументи для впорядкування. Повертає від'ємне ціле число,

\* нульове значення або позитивне ціле число, якщо перший аргумент менше,

\* дорівнює, або більше, ніж другий.

\*

\* **@param** first перший рядок, що потрібно порівняти

\* **@param** second другий рядок, що потрібно порівняти

\* **@return** від'ємне ціле число, нульове значення або позитивне ціле число,

\* якщо перший аргумент менше, дорівнює, або більше, ніж другий

\*/

**public** **int** **compare**(**String** first, **String** second) {

**int** **firstLength** = first.length();

**int** **secondLength** = second.length();

**int** **min** = **Math**.*min*(firstLength, secondLength);

**for** (**int** **i** = 0; i < min; i++) {

**char** **firstChar** = first.charAt(i);

**char** **secondChar** = second.charAt(i);

**if** (firstChar != secondChar) {

firstChar = **Character**.*toUpperCase*(firstChar);

secondChar = **Character**.*toUpperCase*(secondChar);

**if** (firstChar != secondChar) {

firstChar = **Character**.*toLowerCase*(firstChar);

secondChar = **Character**.*toLowerCase*(secondChar);

**if** (firstChar != secondChar) {

**return** firstChar - secondChar;

}

}

}

}

**return** firstLength - secondLength;

}

}

1. **РЕЗУЛЬТАТ РОБОТИ**

Для налагодження роботи програми було успішно проведено її тестування.

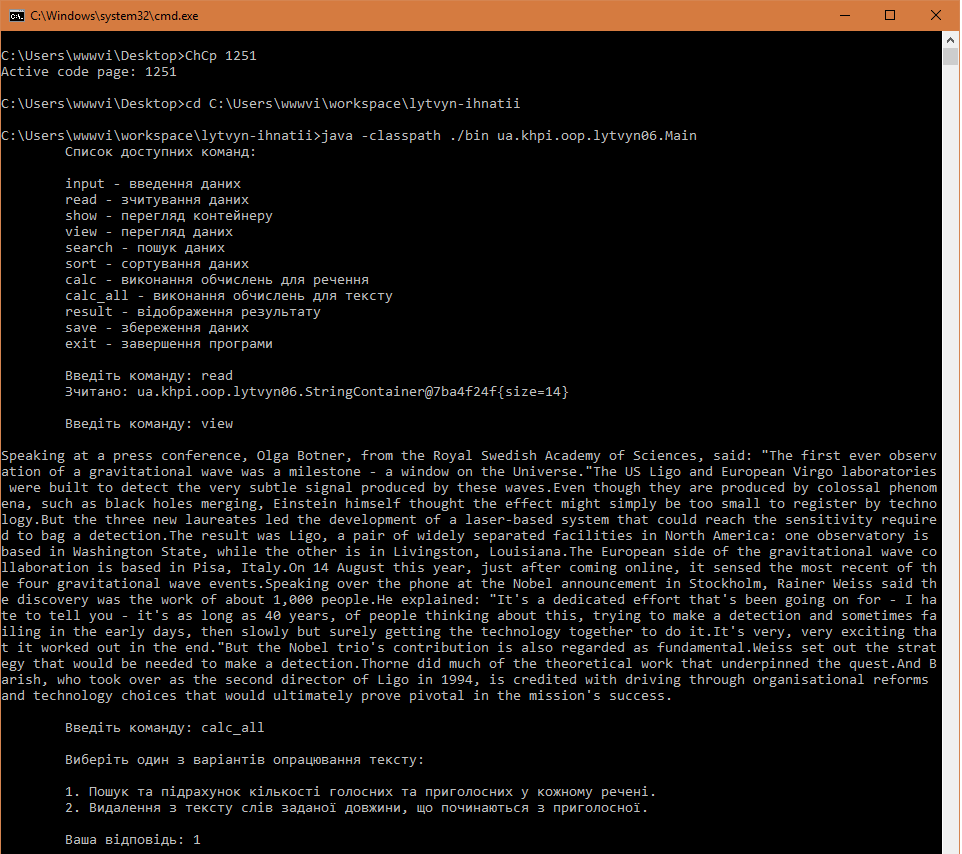
****

Рисунок 2 "Результат послідовної обробки"

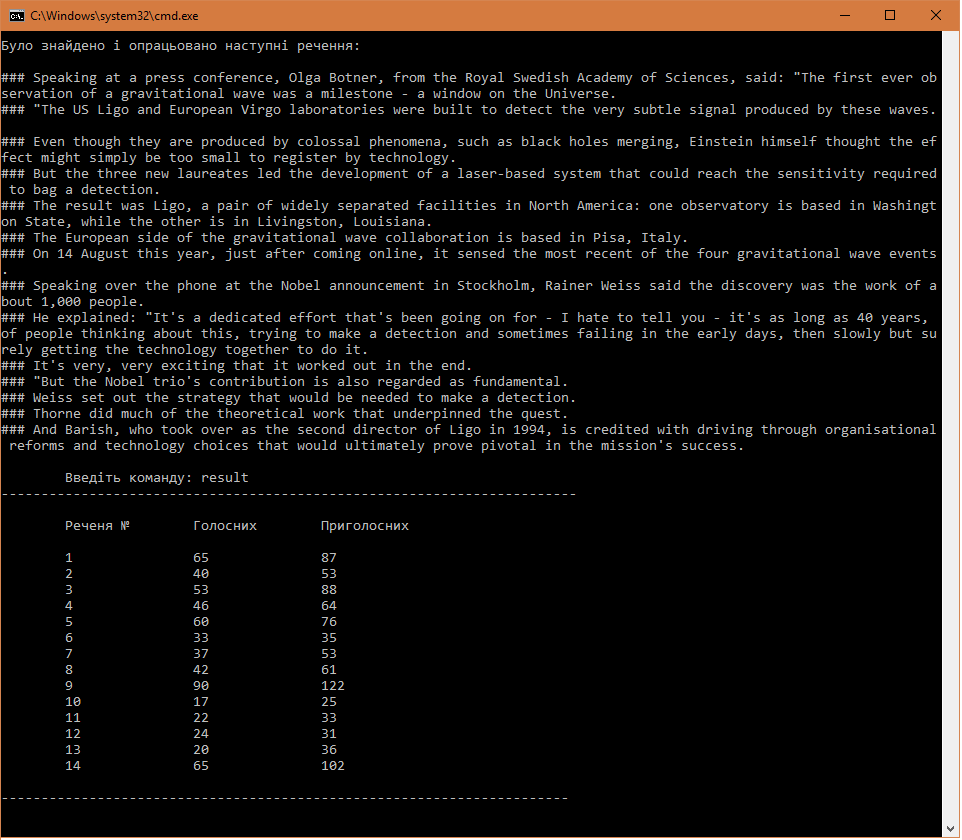
****

Рисунок 3 "Результат послідовної обробки"

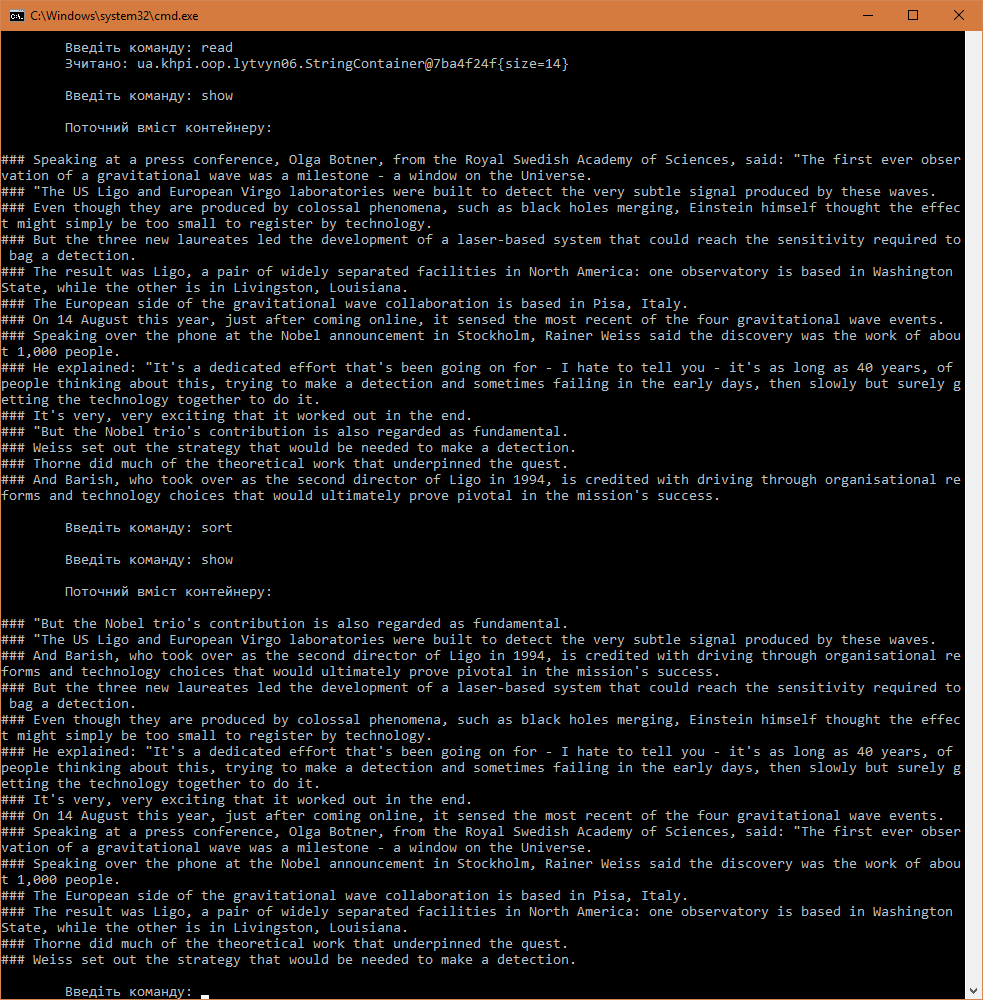
****

Рисунок 4 "Результат сортування"

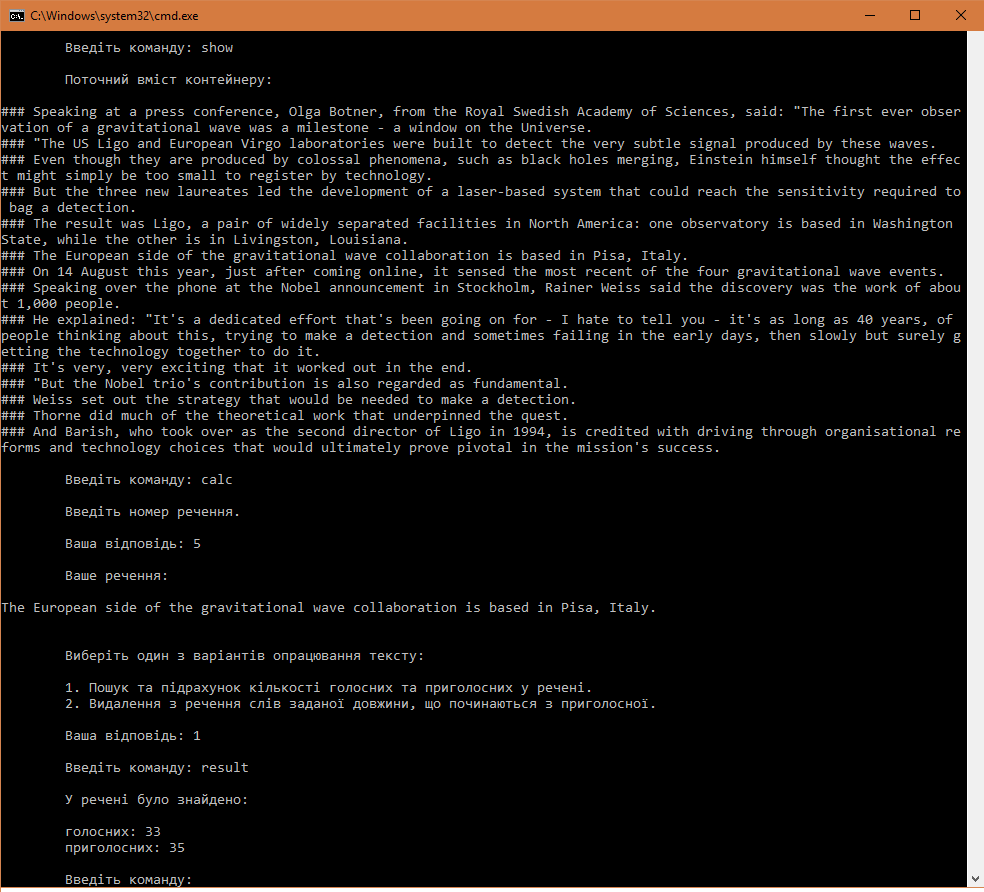
****

Рисунок 5 "Результат вибіркової обробки"

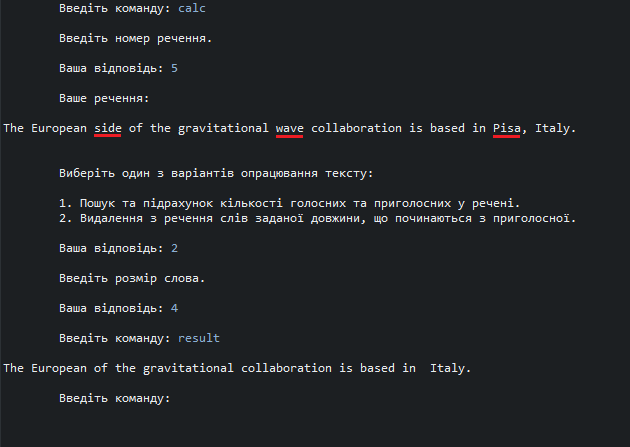
****

Рисунок 6 "Результат вибіркової обробки"

**ВИСНОВКИ**

*Створено і налагоджено програму, що повністю виконую поставлене індивідуальне завдання та відповідає вимогам.*

*Було отримано і вдосконалено навички у тривалому зберіганні та відновленні стану об’єктів (серіалізація/десеріалізація об’єктів), використанні бібліотек класів користувача.*