МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

КАФЕДРА «Програмна інженерія та інтелектуальні технології управління»

ЗВІТ

з лабораторної роботи № 5

з навчальної дисципліни

“ОСНОВИ ПРОГРАМУВАННЯ JAVA (ДВВ)”

ВИКОНАЛА

Студентка групи КН-222а

Інеса РЕПЕШКО

ПЕРЕВІРИВ

старший викладач каф. ПІІТУ к.т.н

Лев ІВАНОВ

Харків 2023

# **Тема роботи**

“Робота з винятками та файлами в Java”.

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

Спроектувати та реалізувати класи для представлення сутностей попередньої лабораторної роботи. Рішення повинне базуватися на раніше створеній ієрархії класів. Слід створити абстрактний клас, похідний від класу, який представляє основну сутність. Цей клас, повинен декларувати відповідні операції читання і запису і реалізувати можливості демонстрації взаємодії з файлами різних форматів.

Слід також створити два похідних класи від класу. Один клас повинен бути доповненим можливостями створення текстового файлу, читання даних з цього текстового файлу та запису цих даних в інший файл після сортування. Другий клас повинен реалізовувати створення бінарного файлу даних, читання даних з цього файлу даних та запису цих даних в інший файл дних після сортування.

Окрім роботи з файлами повинно бути реалізоване виведення результатів у консольне вікно.

Умову завдання з лабораторної роботи № 3 для варіанту 24 (номер 24 за порядком у списку групи) наведено нижче.

Изображение выглядит как текст, снимок экрана, Шрифт, чек

Автоматически созданное описание

Рисунок 1.1 – Умова 1 завдання № 1 варіант 24

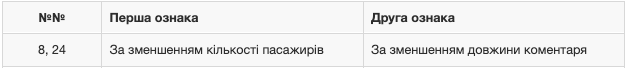


Рисунок 1.2 – Умова 2 завдання № 1 варіант 24

**1.2 Набір тестових даних завдання № 1**

**1.2.1 Тестовий випадок № 1** **“Тестування класу** **MetroStationWithFile, MetroStationWithTextFile, MetroStationWithBinaryFile та MetroStationWithBinarySerialization”**

Опис: Тестування функціональності наступних класів у роботі з обʼєктом класу MetroStationWithList для запису і читання даних з текстового і бінарного файлів, а також із використанням серіалізації:

* MetroStationWithFile;
* MetroStationWithTextFile;
* MetroStationWithBinaryFile;
* MetroStationWithBinarySerialization.

Тестові дані:

Таблиця 1.1 – Тестові дані тестування для класів

|  |  |  |  |
| --- | --- | --- | --- |
| Індекс | Пасажиропотік | Коментар | |
| Тестування створення об’єкту станції метро з масивом годин роботи станції метро з послідовним додаванням даних окремих годин роботи в кінець масиву годин роботи (валідні дані):   * Назва: «Derzhprom» * Рік відкриття: 1995 | | | |
|  | 320 | | "Medium ridership" |
|  | 88 | | "Very low ridership" |
|  | 107 | | "Low ridership" |
|  | 688 | | "High ridership" |
|  | 1234 | | "Very high ridership" |
|  | 0 | | null |

Очікуваний результат: Успішне завершення виконання програми. Запис у відповідні файли даних об’єкту у відповідному стані. Виведені результати створення, сортування та читання даних об’єкта з текстових і бінарних файлів повинні збігатися для всіх методів читання даних. Виведення у консоль повідомлення з результатами:

|  |
| --- |
| CREATION OF THE METRO STATION:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  There are no hours for this station.  Add 6 valid Operating Hours at Metro Station:  true true true true true  Add one Operating Hour with invalid data at Metro Station: true  Add one Operating Hour with duplicate data at Metro Station: false  OPERATIONS OF WRITING AND READING FROM A FILE:  Write to file Metro Station with Operating Hours:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  Hour { ridership = 320, comment = 'Medium ridership' }  Hour { ridership = 88, comment = 'Very low ridership' }  Hour { ridership = 107, comment = 'Low ridership' }  Hour { ridership = 688, comment = 'High ridership' }  Hour { ridership = 1234, comment = 'Very high ridership' }  Hour { ridership = 0, comment = '' }  Write to file Metro Station with removed Operating Hours:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  There are no hours for this station.  Read from file Metro Station without Operating Hours:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  There are no hours for this station.  Read from file Metro Station with Operating Hours and set its data:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  Hour { ridership = 320, comment = 'Medium ridership' }  Hour { ridership = 88, comment = 'Very low ridership' }  Hour { ridership = 107, comment = 'Low ridership' }  Hour { ridership = 688, comment = 'High ridership' }  Hour { ridership = 1234, comment = 'Very high ridership' }  Hour { ridership = 0, comment = '' }  Write to file Metro Station with Operating Hours sorted by decreasing ridership:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  Hour { ridership = 1234, comment = 'Very high ridership' }  Hour { ridership = 688, comment = 'High ridership' }  Hour { ridership = 320, comment = 'Medium ridership' }  Hour { ridership = 107, comment = 'Low ridership' }  Hour { ridership = 88, comment = 'Very low ridership' }  Hour { ridership = 0, comment = '' }  Write to file Metro Station with Operating Hours sorted by descending comment length:  Station: Name: 'Derzhprom'. Opened: 1995. Hours:  Hour { ridership = 1234, comment = 'Very high ridership' }  Hour { ridership = 88, comment = 'Very low ridership' }  Hour { ridership = 320, comment = 'Medium ridership' }  Hour { ridership = 688, comment = 'High ridership' }  Hour { ridership = 107, comment = 'Low ridership' }  Hour { ridership = 0, comment = '' } |

**1.3 Програмний код завдання № 1**

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

|  |
| --- |
| package task1;  import java.io.Serial; import java.io.Serializable; import java.util.Arrays;  */\*\*  \* The {****@code*** *Hour} class performs hour with {****@code*** *ridership} and {****@code*** *comment}.  \* It implements {****@link*** *Serializable} interface.  \*/* public class Hour implements Comparable<Hour>, Serializable {  */\*\* Unique version identifier for serialisation and deserialisation which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;  */\*\* Ridership is the number of passengers visiting a metro station per hour. \*/* private int ridership;   */\*\* Comment on the {****@code*** *ridership} metric. \*/* private String comment;   */\*\*  \* The constructor initialises the hour object with the default values.  \*/* public Hour() {  }   */\*\*  \* The constructor initialises the hour object with the specified values.  \** ***@param*** *ridership the ridership;  \** ***@param*** *comment the comment.  \*/* public Hour(int ridership, String comment) {  if (ridership < 0) {  this.ridership = 0;  }   if (comment == null) {  this.comment = "";  }   this.ridership = ridership;  this.comment = comment;  }   */\*\*  \* Gets the {****@code*** *ridership} of the hour.  \** ***@return*** *the {****@code*** *ridership}.  \*/* public int getRidership() {  if (ridership < 0) {  return 0;  }  return ridership;  }   */\*\*  \* Sets the {****@code*** *ridership} of the hour.  \** ***@param*** *ridership the {****@code*** *ridership} to be set.  \*/* public void setRidership(int ridership) {  if (ridership < 0) {  this.ridership = 0;  }   this.ridership = ridership;  }   */\*\*  \* Gets the {****@code*** *comment} for the hour.  \** ***@return*** *the {****@code*** *comment}.  \*/* public String getComment() {  if (comment == null) {  return "";  }   return comment;  }   */\*\*  \* Sets the {****@code*** *comment} for the hour.  \** ***@param*** *comment the {****@code*** *comment} to be set.  \*/* public void setComment(String comment) {  if (comment == null) {  this.comment = "";  }   this.comment = comment;  }   */\*\*  \* Gets the length of a comment in the hour.  \** ***@return*** *the length of a comment.  \*/* public int getCommentLength() {  if (comment == null) {  return 0;  }   return getComment().length();  }   */\*\*  \* Calculates the count of words of a comment in the hour.  \** ***@return*** *the length of a comment.  \*/* public int calculateWordCountOfComment() {  if (comment == null  || comment.isEmpty()) {  return 0;  }   String[] wordArray = comment.split(" ");   return wordArray.length;  }   */\*\*  \* Provides the string representing the Hour object.  \** ***@return*** *the string representing the Hour object.  \*/* @Override  public String toString() {  return "Hour\t{ "  + "ridership = " + getRidership()  + ",\tcomment = \'" + getComment() + "\' }";  }   */\*\*  \* Checks metro station this hour is equivalent to another.  \** ***@param*** *obj the hour with which check the equivalence;  \** ***@return*** *{****@code*** *true}, if two hours are the same and {****@code*** *false} otherwise.  \*/* @Override  public boolean equals(Object obj) {  if (this == obj) {  return true;  }   if (!(obj instanceof Hour hour)) {  return false;  }   return Integer.*compare*(hour.getRidership(), getRidership()) == 0  && hour.getComment().equals(getComment());  }   */\*\*  \* Calculates the hash code of the hour.  \* If two objects are equal, they must have the same hash code.  \* If this method is called multiple times on the same object, it must return the same number each time.  \** ***@return*** *the hash code of the hour.  \*/* @Override  public int hashCode() {  return Integer.*hashCode*(getRidership()) \* getComment().hashCode();  }   */\*\*  \* Compares this Hour object with another Hour object based on ridership.  \** ***@param*** *h the object to be compared;  \** ***@return*** *negative number, if this object is smaller, zero, if they are equal,  \* positive number, if this object is larger.  \*/* @Override  public int compareTo(Hour h) {  return Integer.*compare*(h.getRidership(), getRidership());  }   */\*\*  \* Prints the array of hours.  \** ***@param*** *hours array of hours to print.  \*/* public static void printHourArray(Hour[] hours) {  System.*out*.println("Array of hours:");   for (Hour hour : hours) {  System.*out*.println(hour);  }  }   */\*\*  \* Tests of the functionality of the {****@code*** *Hour} class.  \* The {****@code*** *args} are not used.  \** ***@param*** *args the command-line arguments.  \*/* public static void main(String[] args) {  System.*out*.println("Create Hour with default constructor:");  Hour hour = new Hour();  System.*out*.println(hour);  System.*out*.println("Length of comment:\t" + hour.getCommentLength());  System.*out*.println("Count of words in comment:\t" + hour.calculateWordCountOfComment());   System.*out*.println("\nCreate Hour with parameterized constructor:");  System.*out*.println("Valid data for hour:");  hour = new Hour(100, "Low ridership");  System.*out*.println(hour);  System.*out*.println("Invalid data for hour:");  Hour invalidHour = new Hour(-200, null);  System.*out*.println(invalidHour);   System.*out*.println("\nSet values for the Hour:");  hour.setRidership(200);  hour.setComment("Medium ridership");  System.*out*.println(hour);   System.*out*.println("\nGet values for the Hour:");  System.*out*.println("Hour\t{ "  + "ridership = " + hour.getRidership()  + ",\tcomment = \'" + hour.getComment() + "\' }");  System.*out*.println("Get length of comment:\t" + hour.getCommentLength());  System.*out*.println("Get count of words in comment:\t" + hour.calculateWordCountOfComment() + "\n");   Hour[] hours = { hour,  new Hour(50, "Very low ridership"),  new Hour(200, "Medium ridership"),  new Hour(100, "Low ridership"),  new Hour(700, "High ridership"),  new Hour(1200, "Very high ridership"),  invalidHour  };  *printHourArray*(hours);   System.*out*.println("\nCheck for equal values of Hours at index 0 and 1:\t" + hours[0].equals(hours[1]));  System.*out*.println("Hour at index 0:\t" + hours[0]);  System.*out*.println("Hour at index 1:\t" + hours[1]);  System.*out*.println("Check for equal values of Hours at index 0 and 2:\t" + hours[0].equals(hours[2]));  System.*out*.println("Hour at index 0:\t" + hours[0]);  System.*out*.println("Hour at index 2:\t" + hours[2]);   System.*out*.println("\nComparison of Hours at index 0 and 1:\t" + hours[0].compareTo(hours[1]));  System.*out*.println("Hashcode of Hour at index 0:\t" + hours[0].hashCode());  System.*out*.println("Hashcode of Hour at index 1:\t" + hours[1].hashCode());  System.*out*.println("Comparison of Hours at index 0 and 2:\t" + hours[0].compareTo(hours[2]));  System.*out*.println("Hashcode of Hour at index 0:\t" + hours[0].hashCode());  System.*out*.println("Hashcode of Hour at index 2:\t" + hours[2].hashCode());  System.*out*.println("Comparison of Hours at index 1 and 2:\t" + hours[1].compareTo(hours[2]));  System.*out*.println("Hashcode of Hour at index 1:\t" + hours[1].hashCode());  System.*out*.println("Hashcode of Hour at index 2:\t" + hours[2].hashCode());   System.*out*.println("\nSort array of Hours by descending ridership:");  Arrays.*sort*(hours);  *printHourArray*(hours);  } } |

Лістинг 1.2 - Програмний код реалізований у файлі “AbstractMetroStation.java”

|  |
| --- |
| package task1;  import java.io.Serial; import java.io.Serializable; import java.util.Arrays;  */\*\*  \* Abstract class representing metro station with {****@code*** *name}, {****@code*** *opened} year and operating hour data.  \* Access to the sequence of hours, {****@code*** *name} and {****@code*** *opened} year is represented by abstract methods.  \* It implements {****@link*** *Serializable} interface.  \*/* public abstract class AbstractMetroStation implements Serializable {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;   */\*\*  \* Gets the {****@code*** *name} for the metro station.  \* The derived class must provide an implementation of this method.  \** ***@return*** *the {****@code*** *name}.  \*/* public abstract String getName();   */\*\*  \* Sets the {****@code*** *name} for the metro station.  \* The derived class must provide an implementation of this method.  \** ***@param*** *name the {****@code*** *name} to be set.  \*/* public abstract void setName(String name);   */\*\*  \* Gets the {****@code*** *opened} year for the metro station.  \* The derived class must provide an implementation of this method.  \** ***@return*** *the {****@code*** *opened}.  \*/* public abstract int getOpened();   */\*\*  \* Sets the {****@code*** *opened} year for the metro station.  \* The derived class must provide an implementation of this method.  \** ***@param*** *opened the {****@code*** *opened} year to be set.  \*/* public abstract void setOpened(int opened);   */\*\*  \* Gets the {****@code*** *hour} with index {****@code*** *i}.  \* The derived class must provide an implementation of this method.  \** ***@param*** *i the index of hour array element;  \** ***@return*** *the object of class {****@code*** *Hour} with index {****@code*** *i}.  \*/* public abstract Hour getHour(int i);   */\*\*  \* Sets the {****@code*** *hour} with index {****@code*** *i}.  \* The derived class must provide an implementation of this method.  \** ***@param*** *i index of {****@code*** *hour} in array of hours;  \** ***@param*** *hour the object of class {****@code*** *Hour} with index {****@code*** *i} to be set.  \*/* public abstract void setHour(int i, Hour hour);   */\*\*  \* Gets the array of operating hours for the metro station.  \* The derived class must provide an implementation of this method.  \** ***@return*** *the array of operating hours.  \*/* public abstract Hour[] getHours();   */\*\*  \* Sets the array of operating hours for the metro station.  \* The derived class must provide an implementation of this method.  \** ***@param*** *hours the array of operating hours to be set.  \*/* public abstract void setHours(Hour[] hours);   */\*\*  \* Adds a link to the new operating {****@code*** *hour} at the end of the hour array.  \* The derived class must provide an implementation of this method.  \** ***@param*** *hour the object of class {****@code*** *Hour} to be added;  \** ***@return*** *{****@code*** *true}, if the link was added successfully, {****@code*** *false} otherwise.  \*/* public abstract boolean addHour(Hour hour);   */\*\*  \* Creates a new operating {****@code*** *hour} and adds a link to it at the end of the hour array.  \* The derived class must provide an implementation of this method.  \** ***@param*** *ridership the ridership;  \** ***@param*** *comment the comment;  \** ***@return*** *{****@code*** *true}, if the link was added successfully, {****@code*** *false} otherwise.  \*/* public abstract boolean addHour(int ridership, String comment);   */\*\*  \* Counts the number of hours in the hours array.  \* The derived class must provide an implementation of this method.  \** ***@return*** *the number of hours.  \*/* public abstract int countHours();   */\*\*  \* Removes the sequence of hours from hours array.  \* The derived class must provide an implementation of this method.  \*/* public abstract void removeHours();   */\*\*  \* Provides the string representing the object that is inherited from this abstract class.  \** ***@return*** *the string representing the object that is inherited from this abstract class.  \*/* @Override  public String toString() {  StringBuilder string = new StringBuilder();  string.append("Station:\t")  .append("Name: \'").append(getName()).append("\'.\t")  .append("Opened: ").append(getOpened()).append(".\t")  .append("Hours:\n");   if (countHours() <= 0) {  string.append("There are no hours for this station.\n");  } else {  for (Hour h : getHours()) {  string.append(h).append("\n");  }  }   return string.toString();  }   */\*\*  \* Checks whether this metro station is equivalent to another.  \** ***@param*** *obj the metro station with which check the equivalence.  \** ***@return*** *{****@code*** *true}, if two weathers are the same, {****@code*** *false} otherwise.  \*/* @Override  public boolean equals(Object obj) {  if (this == obj) {  return true;  }   if (!(obj instanceof AbstractMetroStation ms)) {  return false;  }   if (!ms.getName().equals(getName())  || Integer.*compare*(ms.getOpened(), getOpened()) != 0) {  return false;  }   return Arrays.*equals*(getHours(), ms.getHours());  }   */\*\*  \* Calculates the hash code of the metro station.  \* If two objects are equal, they must have the same hash code.  \* If this method is called multiple times on the same object, it must return the same number each time.  \** ***@return*** *the hash code of the metro station.  \*/* @Override  public int hashCode() {  return getName().hashCode() \* Integer.*hashCode*(getOpened()) \* Arrays.*hashCode*(getHours());  }   */\*\*  \* An additional static function for adding hour reference to the provided array of hours.  \** ***@param*** *hours the array to which the hour is added;  \** ***@param*** *hour the link that is added;  \** ***@return*** *updated array of hours.  \*/* public static Hour[] addHourToArray(Hour[] hours, Hour hour) {  if (hour.getRidership() < 0  || hour.getComment() == null) {  return hours;  }   Hour[] newHours;   if (hours == null) {  newHours = new Hour[1];  } else {  newHours = new Hour[hours.length + 1];  System.*arraycopy*(hours, 0, newHours, 0, hours.length);  }   newHours[newHours.length - 1] = hour;   return newHours;  }   */\*\*  \* Calculates the total ridership for an array of metro station operating hours.  \** ***@return*** *null, if there is no pointer to the hours array, or it is empty, the total ridership otherwise.  \*/* public Integer calculateTotalRidership() {  if (countHours() == 0) {  return null;  }   int totalRidership = 0;   for (Hour hour : getHours()) {  totalRidership += hour.getRidership();  }   return totalRidership;  }   */\*\*  \* Finds the hours with the minimal ridership in the array of metro station operating hours.  \** ***@return*** *null, if there is no pointer to the hours array, or it is empty,  \* array of hours with minimal ridership otherwise.  \*/* public Hour[] findHoursWithMinRidership() {  if (countHours() == 0) {  return null;  }   Hour minHour = getHours()[0];   for (Hour hour : getHours()) {  if (hour.getRidership() < minHour.getRidership()) {  minHour = hour;  }  }   Hour[] hours = null;   for (Hour hour : getHours()) {  if (hour.getRidership() == minHour.getRidership()) {  hours = *addHourToArray*(hours, hour);  }  }   return hours;  }   */\*\*  \* Finds the hours with the maximum count of words in the comment in the array of metro station operating hours.  \** ***@return*** *null, if there is no pointer to the hours array, or it is empty,  \* array of hours with the maximum word count in comment otherwise  \*/* public Hour[] findHoursWithMaxWordCountOfComment() {  if (countHours() == 0) {  return null;  }   Hour maxHour = getHours()[0];   for (Hour hour : getHours()) {  if (hour.calculateWordCountOfComment() > maxHour.calculateWordCountOfComment()) {  maxHour = hour;  }  }   Hour[] hours = null;   for (Hour hour : getHours()) {  if (hour.calculateWordCountOfComment() == maxHour.calculateWordCountOfComment()) {  hours = *addHourToArray*(hours, hour);  }  }   return hours;  }   */\*\*  \* Finds the total ridership for an array of metro station operating hours and prints the result to the console.  \*/* private void printTotalRidership() {  Integer totalRidership = calculateTotalRidership();  System.*out*.print("Total ridership for station:\t");   if (totalRidership == null) {  System.*out*.println("There is no ridership hours.");  } else {  System.*out*.println(totalRidership);  }  }   */\*\*  \* Prints the array of hours.  \** ***@param*** *hours the array of hours to be printed.  \*/* private void printHours(Hour[] hours) {  for (Hour hour : hours) {  System.*out*.println(hour);  }  }   */\*\*  \* Finds the hours with the minimal ridership in the array of metro station operating hours  \* and prints the result to the console.  \*/* private void printHoursWithMinRidership() {  Hour[] hours = findHoursWithMinRidership();  System.*out*.print("Hours with minimal ridership:\t");   if (hours == null) {  System.*out*.println("There is no ridership hours.");  } else {  System.*out*.println();  printHours(hours);  }  }   */\*\*  \* Finds the hours with the maximum count of words in the comment in the array of metro station operating hours  \* and prints the result to the console.  \*/* private void printHoursWithMaxWordCountOfComment() {  Hour[] hours = findHoursWithMaxWordCountOfComment();  System.*out*.print("Hours with the maximum word count in a comment:\t");   if (hours == null) {  System.*out*.println("There is no ridership hours.");  } else {  System.*out*.println();  printHours(hours);  }  }   */\*\*  \* Sorts a sequence of hours by decreasing ridership using bubble sorting.  \*/* public void sortByDecreasingRidership() {  if (countHours() == 0) {  return;  }   boolean unsorted = true;   while (unsorted) {  unsorted = false;   for (int i = 0; i < getHours().length - 1; i++) {  if (getHours()[i].getRidership() < getHours()[i + 1].getRidership()) {  Hour temp = getHours()[i];  getHours()[i] = getHours()[i + 1];  getHours()[i + 1] = temp;  unsorted = true;  }  }  }  }   */\*\*  \* Sorts a sequence of hours by descending comment length using insertion sorting.  \*/* public void sortByDescendingCommentLength() {  if (countHours() == 0) {  return;  }   for (int i = 0; i < getHours().length; i++) {  Hour key = getHours()[i];  int j;   for (j = i - 1; j >= 0  && Integer.*compare*(getHours()[j].getCommentLength(), key.getCommentLength()) < 0; j--) {  getHours()[j + 1] = getHours()[j];  }   getHours()[j + 1] = key;  }  }   */\*\*  \* An additional function for adding hours to a sequence of hours in hours array.  \** ***@return*** *The object is inherited from this abstract class.  \*/* public AbstractMetroStation createMetroStationHours() {  System.*out*.println("CREATION OF THE METRO STATION:");  setName("Derzhprom");  setOpened(1995);  System.*out*.println(this);   System.*out*.println("Add 6 valid Operating Hours at Metro Station:");  System.*out*.print(addHour(320, "Medium ridership") + "\t");  Hour hour = new Hour(88, "Very low ridership");  System.*out*.println(addHour(hour) + "\t"  + addHour(107, "Low ridership") + "\t"  + addHour(688, "High ridership") + "\t"  + addHour(1234, "Very high ridership"));   System.*out*.println("Add one Operating Hour with invalid data at Metro Station:\t"  + addHour(-1, null));   System.*out*.println("Add one Operating Hour with duplicate data at Metro Station:\t"  + addHour(1234, "Very high ridership"));   return this;  }   */\*\*  \* Calls up search methods and print results of searching.  \*/* private void showSearchResults() {  printTotalRidership();  printHoursWithMinRidership();  printHoursWithMaxWordCountOfComment();  }   */\*\*  \* Performs testing of search methods.  \*/* public void testSearchData() {  System.*out*.println("SEARCHING RESULTS:");  setName("Universytet");  setOpened(1984);  System.*out*.println("Search data for Metro Station without Operating Hours:");  removeHours();  showSearchResults();  System.*out*.println();   System.*out*.println("Create the Metro Station:");  createMetroStationHours();  System.*out*.println(this);  showSearchResults();  System.*out*.println();   System.*out*.println("Add new two Operating Hours with min ridership and max word count in comment for searching:");  System.*out*.println(addHour(75, "Very low ridership"));  System.*out*.println(addHour(2000, "Maximum possible ridership for station"));  System.*out*.println(this);  showSearchResults();  }   */\*\*  \* Performs testing of sorting methods.  \*/* public void testSortingData() {  System.*out*.println();  System.*out*.println("SORTING RESULTS:");  setName("Derzhprom");  setOpened(1995);  System.*out*.println("Sort data for Metro Station without Operating Hours:");  removeHours();  sortByDecreasingRidership();  sortByDescendingCommentLength();  System.*out*.println(this);   System.*out*.println("Create the Metro Station:");  createMetroStationHours();  System.*out*.println(this);   System.*out*.println("Sort Operating Hours by decreasing ridership:");  sortByDecreasingRidership();  System.*out*.println(this);   System.*out*.println("Sort Operating Hours by descending comment length:");  sortByDescendingCommentLength();  System.*out*.println(this);  } } |

Лістинг 1.3 - Програмний код реалізований у файлі “MetroStationWithCollection.java”

|  |
| --- |
| package task1;  import java.io.Serial;  */\*\*  \* An abstract class {****@link*** *MetroStationWithCollection} representing a Metro Station with a collection of operating  \* hours. Extends the {****@link*** *AbstractMetroStation} class.  \*/* public abstract class MetroStationWithCollection extends AbstractMetroStation {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;   */\*\* The name of the metro station. \*/* private String name;   */\*\* The opened year of the metro station. \*/* private int opened;   */\*\*  \* The constructor initialises the object with the default values.  \*/* public MetroStationWithCollection() {}   */\*\*  \* The constructor initialises the object with the specified values with metro station {****@code*** *name}  \* and {****@code*** *opened} year.  \** ***@param*** *name the name of metro station;  \** ***@param*** *opened the opened year of metro station.  \*/* public MetroStationWithCollection(String name, int opened) {  this.name = name;  this.opened = opened;  }   public abstract void setHour(int i, Hour hour);   */\*\*  \* Gets the {****@code*** *name} for the metro station.  \** ***@return*** *the {****@code*** *name} of metro station.  \*/* @Override  public String getName() {  return name;  }   */\*\*  \* Sets the {****@code*** *name} for the metro station.  \** ***@param*** *name the {****@code*** *name} of metro station to be set.  \*/* @Override  public void setName(String name) {  this.name = name;  }   */\*\*  \* Gets the {****@code*** *opened} year for the metro station.  \** ***@return*** *the {****@code*** *opened} year of metro station.  \*/* @Override  public int getOpened() {  return opened;  }   */\*\*  \* Sets the {****@code*** *opened} year for the metro station.  \** ***@param*** *opened the {****@code*** *opened} year of metro station to be set.  \*/* @Override  public void setOpened(int opened) {  this.opened = opened;  }   */\*\*  \* Performs testing of the functionality of the {****@code*** *MetroStationWithCollection} class.  \*/* public void testMetroStationWithCollection() {  System.*out*.println("Initial Metro Station data:");  System.*out*.println(this);   System.*out*.println("Get Metro Station Name and Opened Year:");  System.*out*.println("Name:\t" + getName() + "\tOpened:\t" + getOpened());  System.*out*.println();   System.*out*.println("Reset the Operating Hours for the Metro Station:");  Hour[] hoursArray = {  new Hour(23, "Very low ridership"),  new Hour(345, "Medium ridership"),  new Hour(87, "Low ridership"),  new Hour(1007, "Very high ridership")  };  setHours(hoursArray);  System.*out*.println(this);   System.*out*.println("Set the Operating Hour by index and get all Operating Hours:");  setHour(0, new Hour(250, "Medium ridership"));  hoursArray = getHours();  for (Hour hour : hoursArray) {  System.*out*.println(hour);  }  System.*out*.println();   System.*out*.println("Get Operating Hour by index:");  System.*out*.println(getHour(1));  System.*out*.println("Get count of all Operating Hours:\t" + countHours());  System.*out*.println();  } } |

Лістинг 1.4 - Програмний код реалізований у файлі “MetroStationWithList.java”

|  |
| --- |
| package task1;  import java.io.Serial; import java.util.ArrayList; import java.util.Arrays; import java.util.Comparator; import java.util.List; import java.util.Collections; import java.util.Set; import java.util.LinkedHashSet;  */\*\*  \* The {****@link*** *MetroStationWithList} class represents metro station data with an ArrayList of operating hours.  \* This class is inherited from the abstract {****@link*** *MetroStationWithCollection}.  \*/* public class MetroStationWithList extends MetroStationWithCollection {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;   */\*\* List of operating hours of the metro station. \*/* private List<Hour> hours = new ArrayList<>();   */\*\*  \* The constructor initialises the metro station object with the default values.  \*/* public MetroStationWithList() {}   */\*\*  \* The constructor initialises the metro station object with the specified values with {****@code*** *name},  \* {****@code*** *opened} year and operating {****@code*** *hours}.  \** ***@param*** *name the name of metro station;  \** ***@param*** *opened the opened year of metro station;  \** ***@param*** *hours the operating hours of metro station.  \*/* public MetroStationWithList(String name, int opened, ArrayList<Hour> hours) {  super(name, opened);  Set<Hour> uniqueSet = new LinkedHashSet<>(hours);  this.hours = new ArrayList<>(uniqueSet);  }   */\*\*  \* The constructor initialises the metro station object with the specified values with {****@code*** *name} and {****@code*** *opened} year.  \** ***@param*** *name the name of metro station;  \** ***@param*** *opened the opened year of metro station.  \*/* public MetroStationWithList(String name, int opened) {  super(name, opened);  }   */\*\*  \* Gets the array of operating hours for the metro station.  \** ***@return*** *the array of hours.  \*/* @Override  public Hour[] getHours() {  return hours.toArray(new Hour[0]);  }   */\*\*  \* Gets the list of operating hours for the metro station.  \** ***@return*** *the list of operating hours for the metro station.  \*/* public List<Hour> getHoursList() {  return hours;  }   */\*\*  \* Sets the list of operating hours for the metro station.  \** ***@param*** *hours the array of hours to be set.  \*/* @Override  public void setHours(Hour[] hours) {  Set<Hour> uniqueSet = new LinkedHashSet<>(Arrays.*asList*(hours));  this.hours = new ArrayList<>(uniqueSet);  }   */\*\*  \* Gets the {****@code*** *hour} with index {****@code*** *i} from the hours list.  \** ***@return*** *the object of class {****@code*** *Hour} with index {****@code*** *i}.  \*/* @Override  public Hour getHour(int i) {  return hours.get(i);  }   */\*\*  \* Sets the {****@code*** *hour} with index {****@code*** *i} to hours list.  \** ***@param*** *i index of {****@code*** *hour} in hours list;  \** ***@param*** *hour the object of class {****@code*** *Hour} with index {****@code*** *i} to be set.  \*/* @Override  public void setHour(int i, Hour hour) {  if (hours.contains(hour)) {  return;  }   hours.set(i, hour);  }   */\*\*  \* Adds a link to the new {****@code*** *hour} at the end of the hours list.  \** ***@param*** *hour the object of class {****@code*** *Hour} to be added to the hours list;  \** ***@return*** *{****@code*** *true}, if the link was added successfully, {****@code*** *false} otherwise.  \*/* @Override  public boolean addHour(Hour hour) {  if (hours.contains(hour)) {  return false;  }   return hours.add(hour);  }   */\*\*  \* Creates a new {****@code*** *hour} and adds a link to it at the end of the sequence at the hours list.  \** ***@param*** *ridership the ridership;  \** ***@param*** *comment the comment;  \** ***@return*** *{****@code*** *true}, if the link was added successfully, {****@code*** *false} otherwise.  \*/* @Override  public boolean addHour(int ridership, String comment) {  return addHour(new Hour(ridership, comment));  }   */\*\*  \* Counts the number of hours in the sequence at hours list.  \** ***@return*** *the number of hours.  \*/* @Override  public int countHours() {  return hours.size();  }   */\*\*  \* Removes the sequence of hours from hours list.  \*/* @Override  public void removeHours() {  hours.clear();  }   */\*\*  \* Overridden decreasing ridership sorting method using the standard sort function of class {****@code*** *Collections}.  \* Is provided by the implementation of the Comparable interface for the {****@code*** *Hour} class.  \*/* @Override  public void sortByDecreasingRidership() {  Collections.*sort*(hours);  }   */\*\*  \* Overridden descending comment length sorting method using the default sort function of interface {****@code*** *List}.  \* Is provided by {****@code*** *Comparator}.  \*/* @Override  public void sortByDescendingCommentLength() {  hours.sort(Comparator.*comparing*(Hour::getCommentLength).reversed());  } } |

Лістинг 1.5 - Програмний код реалізований у файлі “MetroStationWithFile.java”

|  |
| --- |
| package task1;  import java.io.FileNotFoundException; import java.io.IOException; import java.io.Serial; import java.util.InputMismatchException;  */\*\*  \* The {****@link*** *MetroStationWithFile} class is a base class that declares read and write operations  \* and implements the ability to demonstrate interaction with files of different formats.  \* This class is inherited from the {****@link*** *MetroStationWithList}.  \*/* public abstract class MetroStationWithFile extends MetroStationWithList {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;   */\*\*  \* Writes Metro Station and Operating Hours data to a given file.  \** ***@param*** *fileName file name for writing data;  \** ***@throws*** *Exception if an error occurs while writing or processing data.  \*/* public abstract void writeToFile(String fileName) throws Exception;   */\*\*  \* Reads Metro Station and Operating Hours data from a given file.  \** ***@param*** *fileName file name for reading data;  \** ***@throws*** *Exception if an error occurs while reading or processing data from a file.  \*/* public abstract void readFromFile(String fileName) throws Exception;   */\*\*  \* Test interaction (read and write operations) with files of different formats.  \** ***@param*** *MSWithHoursFile the file to store the data of the class object with the operating hours;  \** ***@param*** *MSWithoutHoursFile the file to store the data of the class object without the operating hours;  \** ***@param*** *SortedByCommentFile the file to store the data of the class object sorted by descending comment length;  \** ***@param*** *SortedByRidershipFile the file to store the data of the class object sorted by decreasing ridership.  \*/* public void testWithFile(String MSWithHoursFile,  String MSWithoutHoursFile,  String SortedByRidershipFile,  String SortedByCommentFile) {  createMetroStationHours();   try {  System.*out*.println("\n\nOPERATIONS OF WRITING AND READING FROM A FILE:");  System.*out*.println("Write to file Metro Station with Operating Hours:");  writeToFile(MSWithHoursFile);  System.*out*.println(this);   System.*out*.println("Write to file Metro Station with removed Operating Hours:");  removeHours();  writeToFile(MSWithoutHoursFile);  System.*out*.println(this);   readFromFile(MSWithoutHoursFile);  System.*out*.println("Read from file Metro Station without Operating Hours:");  System.*out*.println(this);   System.*out*.println("Read from file Metro Station with Operating Hours and set its data:");  readFromFile(MSWithHoursFile);  System.*out*.println(this);   System.*out*.println("Write to file Metro Station with Operating Hours sorted by decreasing ridership:");  sortByDecreasingRidership();  writeToFile(SortedByRidershipFile);  System.*out*.println(this);   System.*out*.println("Write to file Metro Station with Operating Hours sorted by descending comment length:");  sortByDescendingCommentLength();  writeToFile(SortedByCommentFile);  System.*out*.println(this);  } catch (FileNotFoundException e) {  System.*err*.println("Failed to read file.");  e.printStackTrace();  } catch (IOException e) {  System.*err*.println("Failed to write to file.");  e.printStackTrace();  } catch (ClassNotFoundException e) {  System.*err*.println("Class definition of a serialized object cannot be found.");  e.printStackTrace();  } catch (InputMismatchException e) {  System.*err*.println("Wrong data format.");  e.printStackTrace();  } catch (Exception e) {  e.printStackTrace();  }  } } |

Лістинг 1.6 - Програмний код реалізований у файлі “MetroStationWithTextFile.java”

|  |
| --- |
| package task1;  import java.io.FileReader; import java.io.FileWriter; import java.io.IOException; import java.io.PrintWriter; import java.io.Serial; import java.util.Scanner;  */\*\*  \* This class inherits from the base {****@link*** *MetroStationWithFile} class.  \* The {****@link*** *MetroStationWithTextFile} class is extended with the ability to create a text file,  \* read data from this text file and write this data to another file after sorting.  \*/* public class MetroStationWithTextFile extends MetroStationWithFile {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;  */\*\*  \* Writes Metro Station and Operating Hours data to a given file.  \** ***@param*** *fileName file name for writing data;  \** ***@throws*** *IOException if an I/O error occurs.  \*/* @Override  public void writeToFile(String fileName) throws IOException {  try (PrintWriter out = new PrintWriter(new FileWriter(fileName))) {  out.println("Station:\tName: \'" + getName() + "\'.\tOpened: " + getOpened() + ".\tHours: " + countHours());   for (Hour hour : getHours()) {  out.println("Hour\t{ ridership = " + hour.getRidership() + ",\tcomment = \'" + hour.getComment() + "\' }");  }  }  }   */\*\*  \* Reads Metro Station and Operating Hours data from a given file.  \** ***@param*** *fileName file name for reading data;  \** ***@throws*** *IOException if an I/O error occurs.  \*/* @Override  public void readFromFile(String fileName) throws IOException {  try (Scanner scanner = new Scanner(new FileReader(fileName))) {  String line = scanner.nextLine();  setName(line.split("Name: \'")[1].split("**\'**.")[0]);  setOpened(Integer.*parseInt*(line.split("Opened: ")[1].split(".\t")[0]));   while (scanner.hasNextLine()) {  line = scanner.nextLine();   if (line.startsWith("Hour\t{ ridership = ")) {  int ridership = Integer.*parseInt*(line.split("ridership = ")[1].split(",\t")[0]);  String comment = line.split("comment = \'")[1].replace("\' }", "");  addHour(new Hour(ridership, comment));  }  }  }  }   */\*\*  \* Tests interaction (read and write operations) with text files.  \* The {****@code*** *args} are not used.  \** ***@param*** *args the command-line arguments.  \*/* public static void main(String[] args) {  String path = "resources/task1/textFiles/";  new MetroStationWithTextFile().testWithFile(  path + "MetroStationWithHours.txt",  path + "MetroStationWithoutHours.txt",  path + "SortedByDecrRidership.txt",  path + "SortedByDescCommentLength.txt"  );  } } |

Лістинг 1.7 - Програмний код реалізований у файлі “MetroStationWithBinaryFile.java”

|  |
| --- |
| package task1;  import java.io.DataInputStream; import java.io.DataOutputStream; import java.io.FileInputStream; import java.io.FileOutputStream; import java.io.IOException; import java.io.Serial;  */\*\*  \* This class inherits from the base {****@link*** *MetroStationWithFile} class.  \* The {****@link*** *MetroStationWithBinaryFile} class is extended with the ability to create a binary data file,  \* read data from this data file and write this data to another file after sorting.  \*/* public class MetroStationWithBinaryFile extends MetroStationWithFile {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;  */\*\*  \* Writes Metro Station and Operating Hours data to a given file.  \** ***@param*** *fileName file name for writing data;  \** ***@throws*** *IOException if an I/O error occurs.  \*/* @Override  public void writeToFile(String fileName) throws IOException {  try (DataOutputStream out = new DataOutputStream(new FileOutputStream(fileName))) {  out.writeUTF(getName());  out.writeInt(getOpened());  out.writeInt(countHours());   for (Hour hour : getHours()) {  out.writeInt(hour.getRidership());  out.writeUTF(hour.getComment());  }  }  }   */\*\*  \* Reads Metro Station and Operating Hours data from a given file.  \** ***@param*** *fileName file name for reading data;  \** ***@throws*** *IOException if an I/O error occurs.  \*/* @Override  public void readFromFile(String fileName) throws IOException {  try (DataInputStream in = new DataInputStream(new FileInputStream(fileName))) {  setName(in.readUTF());  setOpened(in.readInt());  int hoursCount = in.readInt();   for (int i = 0; i < hoursCount; i++) {  int ridership = in.readInt();  String comment = in.readUTF();  addHour(new Hour(ridership, comment));  }  }  }   */\*\*  \* Tests interaction (read and write operations) with binary data files.  \* The {****@code*** *args} are not used.  \** ***@param*** *args the command-line arguments.  \*/* public static void main(String[] args) {  String path = "resources/task1/binaryFiles/";  new MetroStationWithBinaryFile().testWithFile(  path + "MetroStationWithHours.dat",  path + "MetroStationWithoutHours.dat",  path + "SortedByDecrRidership.dat",  path + "SortedByDescCommentLength.dat"  );  } } |

Лістинг 1.8 - Програмний код реалізований у файлі “MetroStationWithBinarySerialization.java”

|  |
| --- |
| package task1;  import java.io.FileOutputStream; import java.io.IOException; import java.io.ObjectOutputStream; import java.io.FileInputStream; import java.io.ObjectInputStream; import java.io.Serializable; import java.io.Serial;  */\*\*  \* This class inherits from the {****@link*** *MetroStationWithFile} class and implements {****@link*** *Serializable} interface.  \* The {****@link*** *MetroStationWithBinarySerialization} class is extended with the ability to serialization  \* and deserialization of objects of this class.  \*/* public class MetroStationWithBinarySerialization extends MetroStationWithFile {  */\*\* Unique version identifier for serialization and deserialization which ensures the uniqueness of the class. \*/* @Serial  private static final long *serialVersionUID* = -424242424242424242L;  */\*\*  \* Implements serialization of class objects.  \* Writes object of {****@code*** *MetroStationWithBinarySerialization} class to a bit stream to be saved in a given file.  \** ***@param*** *fileName file name for writing;  \** ***@throws*** *IOException if an I/O error occurs.  \*/* @Override  public void writeToFile(String fileName) throws IOException {  try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(fileName))) {  out.writeObject(this);  }  }   */\*\*  \* Implements deserialization of class objects.  \* Reads the bit stream, creating saved objects of {****@code*** *MetroStationWithBinarySerialization} class  \* and recreating their state at the time of saving object.  \** ***@param*** *fileName file name for reading;  \** ***@throws*** *IOException if an I/O error occurs;  \** ***@throws*** *ClassNotFoundException class definition of a serialized object cannot be found.  \*/* @Override  public void readFromFile(String fileName) throws IOException, ClassNotFoundException {  try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(fileName))) {  MetroStationWithBinarySerialization metroStation = (MetroStationWithBinarySerialization) in.readObject();  setName(metroStation.getName());  setOpened(metroStation.getOpened());  setHours(metroStation.getHours());  }  }   */\*\*  \* Tests interaction (Serialization and deserialization) with binary object files.  \* The {****@code*** *args} are not used.  \** ***@param*** *args the command-line arguments.  \*/* public static void main(String[] args) {  String path = "resources/task1/objectFiles/";  new MetroStationWithBinarySerialization().testWithFile(  path + "MetroStationWithHours.dat",  path + "MetroStationWithoutHours.dat",  path + "SortedByDecrRidership.dat",  path + "SortedByDescCommentLength.dat"  );  } } |

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

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.3 - Структура файлів в проекті “lab5”

**1.4** **Екранні форми за результатами роботи програмного коду завдання № 1**

**Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание**

Рисунок 1.4 – Структура файлів в проекті у директорії «resources/task1»

Изображение выглядит как текст, снимок экрана, черно-белый, дизайн

Автоматически созданное описание

Рисунок 1.5 - Результати роботи програмного коду відповідно до тестового випадку № 1. Тестування класу MetroStationWithTextFile

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 1.6 – Вміст текстового файлу «MetroStationWithHours.txt»

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.7 – Вміст текстового файлу «MetroStationWithoutHours.txt»

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 1.8 – Вміст текстового файлу «SortedByDecrRidership.txt»

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание

Рисунок 1.9 – Вміст текстового файлу «SortedByDescCommentLength.txt»

Изображение выглядит как текст, снимок экрана, черно-белый, чек

Автоматически созданное описание

Рисунок 1.10 - Результати роботи програмного коду відповідно до тестового випадку № 1. Тестування класу MetroStationWithBinaryFile

Изображение выглядит как текст, снимок экрана, Шрифт

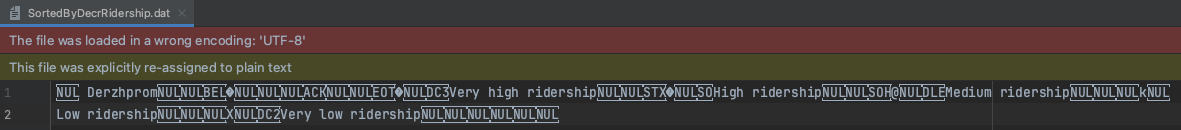
Автоматически созданное описание

Рисунок 1.11 – Вміст бінарного файлу «MetroStationWithHours.dat»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.12 – Вміст бінарного файлу «MetroStationWithoutHours.dat»



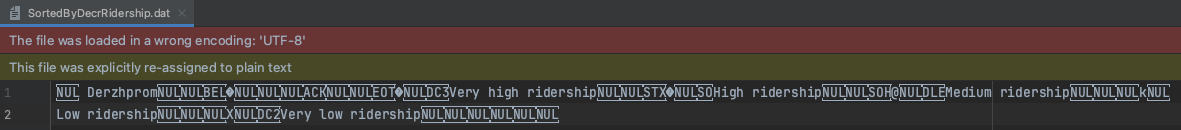
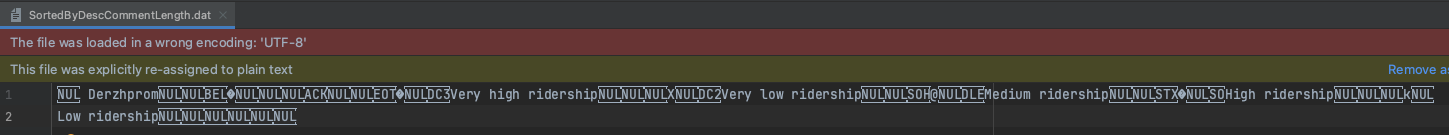


Рисунок 1.13 – Вміст бінарного файлу «SortedByDecrRidership.dat»



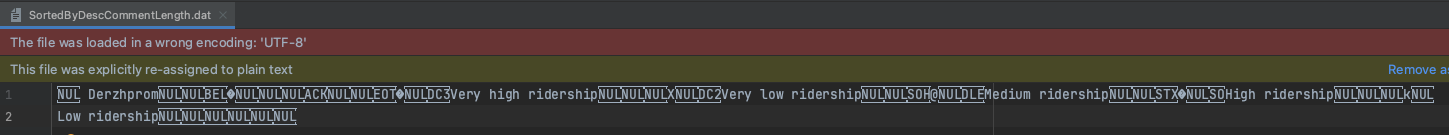


Рисунок 1.14 – Вміст бінарного файлу «SortedByDescCommentLength.dat»

Изображение выглядит как текст, снимок экрана, черно-белый, дизайн

Автоматически созданное описание

Рисунок 1.15 - Результати роботи програмного коду відповідно до тестового випадку № 1. Тестування класу MetroStationWithBinarySerialization

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 1.16 – Вміст бінарного файлу серіалізації «MetroStationWithHours.dat»

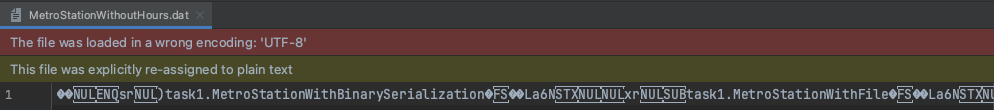


Рисунок 1.17 – Вміст бінарного файлу серіалізації «MetroStationWithoutHours.dat»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.18 – Вміст бінарного файлу серіалізації «SortedByDecrRidership.dat»

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.19 – Вміст бінарного файлу серіалізації «SortedByDescCommentLength.dat»

**2.1 Завдання № 2 “Сортування цілих”**

Реалізувати програму читання з текстового файлу цілих додатних значень (числа розділені пробілами, слід читати до кінця файлу), занесення цих чисел у масив, сортування за зменшенням та за збільшенням суми цифр та зберігання обох результатів у двох нових текстових файлах. Перелічені дії реалізувати в окремій статичній функції. Для визначення порядку сортування створити класи, які реалізують інтерфейс Comparator.

**2.2 Набір тестових даних завдання № 2**

**2.2.1 Тестовий випадок № 1** **“****Тестування з файлом валідними значеннями”**

Опис: Тестування читання даних з файлу та сортування за збільшенням та зменшенням суми цифр чисел для валідних даних – цілих позитивних значень.

Тестові дані в файлі:

«7 78 123 592 999

345678 52943786 999999999

10 2 789 5243

35 68364 99978763».

Очікуваний результат: Успішне завершення виконання програми. Запис у файл результатів сортування:

У файлі «OutputValidSortedAsc.txt»:

|  |
| --- |
| 10 2 123 7 35 5243 78 592 789 999 68364 345678 52943786 99978763 999999999 |

У файлі «OutputValidSortedDesc.txt»:

|  |
| --- |
| 999999999 99978763 52943786 345678 999 68364 789 592 78 5243 35 7 123 2 10 |

**2.2.2 Тестовий випадок № 2** **“****Тестування файлу з невалідними даними: відʼємні цілі числа”**

Опис: Тестування читання даних з файлу та сортування за збільшенням та зменшенням суми цифр чисел для невалідних даних – цілих позитивних та негативних значень.

Тестові дані в файлі:

«87766 5492 374859 75 -200 1 64551».

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку, до якого будуть додані дані трейсу стеку:

|  |
| --- |
| Wrong numeric value.  task2.NonPositiveIntegerException: Wrong numeric value: -200. The numeric value must be positive. |

**2.2.3 Тестовий випадок № 3 “****Тестування файлу з невалідними даними: відʼємні цілі числа та символьна строка”**

Опис: Тестування читання даних з файлу та сортування за збільшенням та зменшенням суми цифр чисел для невалідних даних – цілих позитивних та негативних значень зі строкою символів серед значень.

Тестові дані в файлі:

«87766 5492 test 75 -200 1 64551».

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку, до якого будуть додані дані трейсу стеку:

|  |
| --- |
| Wrong data format.  java.util.InputMismatchException |

**2.2.4 Тестовий випадок № 4 “****Тестування з неіснуючим файлом”**

Опис: Тестування читання даних з неіснуючого файлу.

Тестові дані: відсутні.

Очікуваний результат: Завершення виконання програми з помилкою. Виведення у консоль повідомлення про помилку, до якого будуть додані дані трейсу стеку:

|  |
| --- |
| Failed to read file.  java.io.FileNotFoundException: resources/task2/InputInvalidNonExistent.txt (No such file or directory) |

**2.3 Програмний код завдання № 2**

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

|  |
| --- |
| package task2;  public class NonPositiveIntegerException extends Exception {  private static final long *serialVersionUID* = 3712552941739374213L;   public NonPositiveIntegerException(int wrongValue) {  super("Wrong numeric value: " + wrongValue + ". The numeric value must be positive.");  } } |

Лістинг 2.2 - Програмний код реалізований у файлі “IntegerNumbers.java”

|  |
| --- |
| package task2;  import java.io.FileReader; import java.io.FileWriter; import java.io.IOException; import java.io.PrintWriter; import java.io.BufferedReader; import java.io.BufferedWriter; import java.util.Scanner; import java.util.Arrays; import java.util.Comparator; import java.util.InputMismatchException; import java.io.FileNotFoundException;  public class IntegerNumbers {  private static int findSumOfNumberDigits(int number) {  int sumOfNumberDigits = 0;   while (number > 0) {  sumOfNumberDigits += number % 10;  number /= 10;  }   return sumOfNumberDigits;  }   private static class DigitSumComparator implements Comparator<Integer> {  private final boolean ascendingOrder;   public DigitSumComparator(boolean ascendingOrder) {  this.ascendingOrder = ascendingOrder;  }   @Override  public int compare(Integer firstNumber, Integer secondNumber) {  if (this.ascendingOrder) {  return Integer.*compare*(*findSumOfNumberDigits*(firstNumber), *findSumOfNumberDigits*(secondNumber));  } else {  return Integer.*compare*(*findSumOfNumberDigits*(secondNumber), *findSumOfNumberDigits*(firstNumber));  }  }  }   private static void writeInFile(String outFileName,  Integer[] numbers) throws IOException {  try (PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(outFileName)))) {  for (Integer number : numbers) {  writer.print(number + " ");  }  }  }   private static Integer[] readFromFile(String inFileName) throws NonPositiveIntegerException, IOException {  Integer[] numbers = {};   try (BufferedReader reader = new BufferedReader(new FileReader(inFileName));  Scanner scanner = new Scanner(reader)) {  while (scanner.hasNext()) {  int number = scanner.nextInt();   if (number < 0) {  throw new NonPositiveIntegerException(number);  }   Integer[] array = new Integer[numbers.length + 1];  System.*arraycopy*(numbers, 0, array, 0, numbers.length);  array[array.length - 1] = number;  numbers = array;  }  }   return numbers;  }   public static void sortIntegerNumbers(String inFileName,  String firstOutFileName,  String secondOutFileName) throws NonPositiveIntegerException, IOException {  Integer[] numbers = *readFromFile*(inFileName);  Arrays.*sort*(numbers, new DigitSumComparator(true));  *writeInFile*(firstOutFileName, numbers);  Arrays.*sort*(numbers, new DigitSumComparator(false));  *writeInFile*(secondOutFileName, numbers);  }   public static void main(String[] args) {  String path = "resources/task2/";   try {  *sortIntegerNumbers*(path + "InputValid.txt",  path + "OutputValidSortedAsc.txt",  path + "OutputValidSortedDesc.txt");  /\*  sortIntegerNumbers(path + "InputInvalidNonPositive.txt",  path + "OutputInvalidNonPositiveFirst.txt",  path + "OutputInvalidNonPositiveSecond.txt");  \*/  /\*  sortIntegerNumbers(path + "InputInvalidString.txt",  path + "OutputInvalidStingFirst.txt",  path + "OutputInvalidStingSecond.txt");  \*/  /\*  sortIntegerNumbers(path + "InputInvalidNonExistent.txt",  path + "OutputInvalidNonExistentFirst.txt",  path + "OutputInvalidNonExistentSecond.txt");  \*/  } catch (FileNotFoundException e) {  System.*err*.println("Failed to read file.");  e.printStackTrace();  } catch (IOException e) {  System.*err*.println("Failed to write to file.");  e.printStackTrace();  } catch (NonPositiveIntegerException e) {  System.*err*.println("Wrong numeric value.");  e.printStackTrace();  } catch (InputMismatchException e) {  System.*err*.println("Wrong data format.");  e.printStackTrace();  } catch (SecurityException e) {  e.printStackTrace();  }  } } |

**2.4 Екранні форми за результатами роботи програмного коду завдання № 2**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2.1 – Структура файлів в проекті в директорії «resources/task2»

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 2.2 – Тестові дані до тестового випадку № 1 “ Тестування з файлом валідними значеннями” у файлі «InputValid.txt»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.3 – Записані результати сортування до тестового випадку № 1 “Тестування з файлом валідними значеннями” у файлі «OutputValidSortedAsc.txt»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.4 – Записані результати сортування до тестового випадку № 1 “Тестування з файлом валідними значеннями” у файлі «OutputValidSortedDesc.txt»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.5 - Результати роботи програмного коду відповідно до тестового випадку № 2 “Тестування файлу з невалідними даними: відʼємні цілі числа”у файлі «InputInvalidNonPositive.txt»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.7 - Результати роботи програмного коду відповідно до тестового випадку № 2 “Тестування файлу з невалідними даними: відʼємні цілі числа”

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 2.8 – Тестові дані до тестового випадку № 3 “ Тестування файлу з невалідними даними: відʼємні цілі числа та символьна строка” у файлі «InputInvalidString.txt»

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.9 - Результати роботи програмного коду відповідно до тестового випадку № 3 “ Тестування файлу з невалідними даними: відʼємні цілі числа та символьна строка”

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.4 - Результати роботи програмного коду відповідно до тестового випадку № 4 “ Тестування з неіснуючим файлом”

**3.1 Завдання № 3 “Робота з ZIP-архівом (додаткове завдання)”**

Описати класи Студент і Академічна група (з полем – масивом студентів). Створити об'єкти, здійснити запис даних про студентів академічної групи в архів. В іншій програмі здійснити читання з архіву.

**3.2 Набір тестових даних завдання № 3**

**3.2.1 Тестовий випадок №** **1** **“****Тестування класу Student”**

Опис: Тестування функціональності класу Student.

Тестові дані:

Таблиця 3.1 – Тестові дані для тестування класу Student

|  |  |  |  |
| --- | --- | --- | --- |
| **Ідентифікатор студента** | **Рейтинговий бал** | **Імʼя студента** | **Прізвище студента** |
| Тестування параметризованого конструктора класу із виводом даних про студента: | | | |
| 0 | 90.00 | "TestName" | "TestSurname" |
| Тестування встановлення та отримання даних про студента із виведенням даних у консоль: | | | |
| 1 | 91.01 | “Inessa” | “Repeshko” |

Очікуваний результат: Успішне завершення виконання програми. Виведення у консоль повідомлення з результатами:

|  |
| --- |
| Initial student:  Student ID: 0 Rating Score: 90.0 First Name: TestName Last Name: TestSurname  Test setting and getting student data:  Student ID: 1  Rating Score: 91.01  First Name: Inessa  Last Name: Repeshko |

**3.2.2 Тестовий випадок № 2 “****Тестування класу Group”**

Опис: Тестування функціональності класу Group.

Тестові дані:

Таблиця 3.2 – Тестові дані для тестування класу Group

|  |  |  |  |
| --- | --- | --- | --- |
| **Ідентифікатор студента** | **Рейтинговий бал** | **Імʼя студента** | **Прізвище студента** |
| Тестування параметризованого конструктора класу без списку студентів із виводом даних про групу:   * Ідентифікатор групи: «KN-222a». * Код спеціальності: «121». * Список студентів: відсутній. | | | |
| Тестування параметризованого конструктора класу зі списком студентів із виводом даних про групу:   * Ідентифікатор групи: «KN-222f». * Код спеціальності: «121». * Список студентів: | | | |
| 1 | 87.05 | Anna | Beregova |
| 2 | 83.28 | Vera | Boyko |
| 3 | 85.92 | Oleg | Stryzh |
| 4 | 88.75 | Fedor | Poychenko |
| 5 | 86.63 | Tyler | Fog |
| Тестування встановлення та отримання даних про групу із виведенням даних у консоль:   * Ідентифікатор групи: «AB-123q». * Код спеціальності: «72». * Список студентів: | | | |
| 10 | 90.0 | Aleksandr | Frank |
| 11 | 81.2 | Alevtina | Prud |

Очікуваний результат: Успішне завершення виконання програми. Виведення у консоль повідомлення з результатами:

|  |
| --- |
| Test creating group with parametrized constructor without students:  Group ID: KN-222a  Specialty ID: 121  Student list:  There are no students.  Test creating group with parametrized constructor with students:  Group ID: KN-222f  Specialty ID: 121  Student list:  Student ID: 1 Rating Score: 87.05 First Name: Anna Last Name: Beregova  Student ID: 2 Rating Score: 83.28 First Name: Vera Last Name: Boyko  Student ID: 3 Rating Score: 85.92 First Name: Oleg Last Name: Stryzh  Student ID: 4 Rating Score: 88.75 First Name: Fedor Last Name: Poychenko  Student ID: 5 Rating Score: 86.63 First Name: Tyler Last Name: Fog  Test setting and getting group data:  Group ID: AB-123q  Specialty ID: 72  Student list:  Student ID: 10 Rating Score: 90.0 First Name: Aleksandr Last Name: Frank  Student ID: 11 Rating Score: 81.2 First Name: Alevtina Last Name: Prud |

**3.2.3 Тестовий випадок № 3 “****Тестування класів** **StoreToZip та ReadFromZip”**

Опис: Тестування функціональності класів StoreToZip та ReadFromZip”.

Тестові дані:

Таблиця 3.3 – Тестові дані для тестування класів StoreToZip та ReadFromZip

|  |  |  |  |
| --- | --- | --- | --- |
| **Ідентифікатор студента** | **Рейтинговий бал** | **Імʼя студента** | **Прізвище студента** |
| Тестування запису даних про групу у вигляді списку студентів у Zip-файл і читання даних із Zip-файлу із виводом списку студентів в консоль:   * Ідентифікатор групи: «KN-222f». * Код спеціальності: «121». * Список студентів: | | | |
| 1 | 87.05 | Anna | Beregova |
| 2 | 83.28 | Vera | Boyko |
| 3 | 85.92 | Oleg | Stryzh |
| 4 | 88.75 | Fedor | Poychenko |
| 5 | 86.63 | Tyler | Fog |

Очікуваний результат: Успішне завершення виконання програми. Запис у Zip-файл даних про список студентів. Читання даних із Zip-файлу із виведенням у консоль повідомлення з результатами:

|  |
| --- |
| Student list:  Student ID: 1 Rating Score: 87.05 First Name: Anna Last Name: Beregova  Student ID: 2 Rating Score: 83.28 First Name: Vera Last Name: Boyko  Student ID: 3 Rating Score: 85.92 First Name: Oleg Last Name: Stryzh  Student ID: 4 Rating Score: 88.75 First Name: Fedor Last Name: Poychenko  Student ID: 5 Rating Score: 86.63 First Name: Tyler Last Name: Fog |

**3.3 Програмний код завдання № 3**

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

|  |
| --- |
| package task3;  public class Student {  private int studentID;  private double ratingScore;  private String firstName;  private String lastName;   public Student(int studentID, double ratingScore, String firstName, String lastName) {  this.studentID = studentID;  this.ratingScore = ratingScore;  this.firstName = firstName;  this.lastName = lastName;  }   public int getStudentID() {  return studentID;  }   public double getRatingScore() {  return ratingScore;  }   public String getFirstName() {  return firstName;  }   public String getLastName() {  return lastName;  }   public void setStudentID(int studentID) {  this.studentID = studentID;  }   public void setRatingScore(double ratingScore) {  this.ratingScore = ratingScore;  }   public void setFirstName(String firstName) {  this.firstName = firstName;  }   public void setLastName(String lastName) {  this.lastName = lastName;  }   @Override  public String toString() {  StringBuilder string = new StringBuilder();  string.append("\tStudent ID:\t").append(getStudentID())  .append("\tRating Score:\t").append(getRatingScore())  .append("\tFirst Name:\t").append(getFirstName())  .append("\tLast Name:\t").append(getLastName());   return string.toString();  }   public static void main(String[] args) {  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* STUDENT TESTING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  System.*out*.println("Initial student:");  Student student = new Student(0, 90.00, "TestName", "TestSurname");  System.*out*.println(student);  System.*out*.println();   System.*out*.println("Test setting and getting student data:");  student.setStudentID(1);  student.setRatingScore(91.01);  student.setFirstName("Inessa");  student.setLastName("Repeshko");  System.*out*.println("\tStudent ID:\t\t" + student.getStudentID()  + "\n\tRating Score:\t" + student.getRatingScore()  + "\n\tFirst Name:\t\t" + student.getFirstName()  + "\n\tLast Name:\t\t" + student.getLastName());  } } |

Лістинг 3.2 - Програмний код реалізований у файлі “Group.java”

|  |
| --- |
| package task3;  public class Group {  private String groupID;  private int specialtyID;  private Student[] students;   public Group(String groupID, int specialtyID) {  this.groupID = groupID;  this.specialtyID = specialtyID;  }   public Group(String groupID, int specialtyID, Student... students) {  this.groupID = groupID;  this.specialtyID = specialtyID;  this.students = students;  }   public String getGroupID() {  return groupID;  }   public int getSpecialtyID() {  return specialtyID;  }   public Student[] getStudents() {  return students;  }   public void setGroupID(String groupID) {  this.groupID = groupID;  }   public void setSpecialtyID(int specialtyID) {  this.specialtyID = specialtyID;  }   public void setStudents(Student... students) {  this.students = students;  }   @Override  public String toString() {  StringBuilder string = new StringBuilder();  string.append("\tGroup ID:\t").append(getGroupID())  .append("\n\tSpecialty ID:\t").append(getSpecialtyID())  .append("\n\tStudent list:\n");   if (getStudents() == null || students.length == 0) {  string.append("\t\tThere are no students.\n");  } else {  for (Student student : getStudents()) {  string.append("\t").append(student).append("\n");  }  }   return string.toString();  }   public static Group createTestGroup() {  Student student1 = new Student(1, 87.05, "Anna", "Beregova");  Student student2 = new Student(2, 83.28, "Vera", "Boyko");  Student student3 = new Student(3, 85.92, "Oleg", "Stryzh");  Student student4 = new Student(4, 88.75, "Fedor", "Poychenko");  Student student5 = new Student(5, 86.63, "Tyler", "Fog");   return new Group("KN-222f",  121,  student1, student2, student3, student4, student5);  }   public static void main(String[] args) {  System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* GROUP TESTING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  System.*out*.println("Test creating group with parametrized constructor without students:");  Group firstGroup = new Group("KN-222a", 121);  System.*out*.println(firstGroup);   System.*out*.println("Test creating group with parametrized constructor with students:");  Group secondGroup = *createTestGroup*();  System.*out*.println(secondGroup);   System.*out*.println("Test setting and getting group data:");  secondGroup.setGroupID("AB-123q");  secondGroup.setSpecialtyID(72);  Student student1 = new Student(10, 90.00, "Aleksandr", "Frank");  Student student2 = new Student(11, 81.20, "Alevtina", "Prud");  secondGroup.setStudents(student1, student2);  System.*out*.println("\tGroup ID:\t" + secondGroup.getGroupID()  + "\n\tSpecialty ID:\t" + secondGroup.getSpecialtyID()  + "\n\tStudent list:");  for (Student student : secondGroup.getStudents()) {  System.*out*.println("\t" + student);  }  } } |

Лістинг 3.3 - Програмний код реалізований у файлі “StoreToZip.java”

|  |
| --- |
| package task3;  import java.io.DataOutputStream; import java.io.FileOutputStream; import java.io.IOException; import java.util.zip.ZipEntry; import java.util.zip.ZipOutputStream;  public class StoreToZip {  public static void main(String[] args) {  Group group = Group.*createTestGroup*();   try (ZipOutputStream zOut = new ZipOutputStream(new FileOutputStream("resources/task3/Group.zip"));  DataOutputStream out = new DataOutputStream(zOut)) {  for (Student student : group.getStudents()) {  zOut.putNextEntry(new ZipEntry(Integer.*toString*(student.getStudentID())));  out.writeDouble(student.getRatingScore());  out.writeUTF(student.getFirstName());  out.writeUTF(student.getLastName());  zOut.closeEntry();  }  } catch (IOException | SecurityException e) {  e.printStackTrace();  }  } } |

Лістинг 3.4 - Програмний код реалізований у файлі “ReadFromZip.java”

|  |
| --- |
| package task3;  import java.io.DataInputStream; import java.io.FileInputStream; import java.io.FileNotFoundException; import java.io.IOException; import java.util.zip.ZipEntry; import java.util.zip.ZipInputStream;  public class ReadFromZip {  public static void main(String[] args) {  System.*out*.println("Student list:");   try (ZipInputStream zIn = new ZipInputStream(new FileInputStream("resources/task3/Group.zip"));  DataInputStream in = new DataInputStream(zIn)) {  ZipEntry entry;   while ((entry = zIn.getNextEntry()) != null) {  System.*out*.print("\tStudent ID:\t" + entry.getName());  System.*out*.print("\tRating Score:\t" + in.readDouble());  System.*out*.print("\tFirst Name:\t" + in.readUTF());  System.*out*.println("\tLast Name:\t" + in.readUTF());  zIn.closeEntry();  }  } catch (FileNotFoundException e) {  System.*err*.println("Failed to read file.");  e.printStackTrace();  } catch (IOException | SecurityException e) {  e.printStackTrace();  }  } } |

**3.4 Екранні форми за результатами роботи програмного коду завдання № 3**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.1 - Результати роботи програмного коду відповідно до тестового випадку № 1 “Тестування класу Student”

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.2 - Результати роботи програмного коду відповідно до тестового випадку № 2 “Тестування класу Group”

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 3.3 – Структура файлів в директорії “Group” після отримання файлів з архіву “Group.zip” створеного програмним кодом

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Рисунок 3.4 - Результати роботи програмного коду читання даних з архіву “Group.zip” до тестового випадку № 3 “Тестування класів StoreToZip та ReadFromZip”

**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 даної лабораторної роботи, були отримані значущі навички програмування, зокрема вміння ефективно працювати з винятками та файлами. Розширено ієрархію класів з попередньої лабораторної роботи, додаючи нові класи для забезпечення можливості оптимальної обробки текстових та бінарних файлів у ході виконання індивідуального завдання.

Тестування реалізацій було проведено для перевірки правильності та однаковості отриманих результатів. Серед виконаних завдань також включено глибше вивчення процесу серіалізації. Для зручності управління даними студентів були створені класи, спрямовані на роботу з архівами.

Додатково, продемонстровано навички сортування цілих чисел за сумою їхніх цифр, які були отримані з файлу. Результати сортування були успішно записані в інші файли для подальшого використання.

Загалом, ця лабораторна робота стала важливим кроком у поглибленні розуміння винятків, роботи з текстовими та бінарними файлами, а також у вивченні процесу серіалізації в Java. Вона також сприяла розвитку навичок у вирішенні завдань та підвищенню рівня ефективності в програмуванні.

Тому дана лабораторна робота є не тільки корисною для поглиблення розуміння узагальнень та колекцій у Java, але й сприяла розвитку високоорганізованих навичок програмування та розв'язання завдань. Використання цих можливостей дозволяє навчитись створювати більш функціональні, ефективні, масштабовані та зрозумілі програми на мові Java, що є важливим для розробки якісного програмного забезпечення та розв'язання різноманітних завдань.