МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Практична робота №1**

з дисципліни «Розробка крос-платформного програмного забезпечення»

тема: «Наслідування. Інтерфейси. Виключення.

Командний рядок. Робота з файлами різних форматів»

**Виконав:**

Студент гр. БС-03

Затуловський Г. А.

**Перевірела:**

ст.викл. Сердаковський В.С.

ac. Рисін. С.В.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2023

**Мета:** ознайомитися з особливостями створення класів та об’єктів мовою програмування Java; вивчити засоби узагальненого програмування мови Java на прикладі колекцій об’єктів; розібратися з особливостями виклику та обробки виняткових ситуацій; навчитися використовувати сторонні бібліотеки класів; сформувати вміння оброблювати аргументи командної строки та навички завантаження файлів даних, представлених в CSV/TSV та JSON.

## Теоретичні відомості:

Наслідування та перевизначення методів.

Використання узагальнених класів (колекцій).

Інтерфейси.

Генерація та обробка виключних ситуацій.

Використання аргументів командної строки.

Читання даних з файлу та запис у файл. Формат CSV/TSV. Формат JSON.

Коментарі JavaDoc.

## Завдання.

1. На основі класу Exception створити власний клас *виключення*, що приймає текст помилки та має метод для запису його та дати і часу у файл журналу, ім’я якого має бути статичним полем класу.

2. Створити клас сутності згідно з варіантом.

2.1. Для зберігання числових та логічних значень мають бути використані відповідні типи даних, для логічних властивостей мають бути передбачені відповідні значення у файлі даних.

2.2. Для всіх полів клас має реалізувати гетери та сетери, останні мають перевіряти передане значення та в випадку помилки генерувати власне виключення з відповідною інформацією. Сетери ключових полів мають перевіряти наявність їх значень (які не мають бути порожні). Для нетекстових полів в класі мають бути також сетери, що приймають значення у вигляді рядка, перетворюють його у необхідний тип та передають у відповідний сетер, у разі виключення при перетворенні строкові сетери мають перехоплювати їх та генерувати власне виключення.

2.3. Клас має містити конструктор, що приймає рядок із значеннями полів, послідовність яких може не співпадати з порядком самих полів в опису класу. Для визначення порядку полів у класі мають бути статичний масив назв властивостей сутності, функція визначення порядку властивостей та масив індексів властивостей, за якими конструктор визначає необхідне поле.

2.4. Забезпечити *реалізацію* класом сутності *інтерфейсу* Comparable для можливості порівняння двох об’єктів створеного класу. Для порівняння вибрати два найбільш логічні поля, порівнюючи об’єкти спочатку по полю, що є більш пріорітетним.

2.5. *Перевизначити* метод toString() класу сутності для формування зручної для перегляду строки зі значеннями полів об’єкту. Додати аналогічний метод для формування строки у форматі JSON.

2.6. Від попередньо визначеного класу *породити* дочірній клас, додавши поле “Id” - унікальний ідентифікатор, значення якого має призначатися в дочірньому конструкторі на основі значення *статичної* змінної класу. Дочірній клас також має *перевизначити* метод toString() та метод для формування строки у форматі JSON, в яких мають викликатися відповідні базові методи.

3. З використанням *узагальнення* класів створити клас-контейнер, що містить колекцію об’єктів сутності та має методи імпортування даних з файлу CSV/TSV та ескпортування їх у файл JSON (*імпорт та експорт мають бути реалізовані самостійно, без використання стороннього коду*). Отриманий файл JSON перевірити будь-яким існуючим засобом (програму, веб-сервіс тощо).

3.1. Колекція класа-контейнера має бути параметризована типом класу сутності, але створювати та зберігати має об'єкти дочірнього класу.

3.2. При імпортуванні клас-контейнер має передбачати, що перший рядок містить назви властивостей сутності та викликати функцію класа сутності для визначення їх порядку. При відсутності у вхідному файлі ключового поля має генеруватися відповідне виключення.

3.3. При імпортуванні клас-контейнер має перехоплювати виключення при спробі створення об'єкта з даних рядка, що містить помилки, записувати тексти виключень в консоль та в файл журналу, але не переривати імпортування, а продовжувати обробку наступних рядків.

3.4. *Перевизначити* метод toString() класу-контейнера, що має використовувати метод toString() класу сутності, для можливості перегляду колекції у зручному вигляді.

4. Методи класів повинні мати коментарі JavaDoc («@param»/«@return»/«@throws» для таких, що мають, відповідно, параметри, результат та можуть викидати виключення). Функціональні блоки мають бути супроводжені коментарями, що коротко описують суть наступного фрагменту коду.

5. За допомогою ПО для роботи з електронними таблицями (наприклад, OpenOffice Calc) або, як альтернатива, будь-якого текстового редактору створити таблицю зі структурою згідно варіанту та наповнити її даними (5-10 рядків). Перший рядок таблиці має містити імена властивостей сутності. Порядок стовпців має відрізнятися від порядку в масиві назв властивостей в класі сутності. Серед даних мають бути рядки з невірними даними – хоча б один з порожнім полем, що порівнюється, та хоча б один з помилковим значенням (наприклад, поза межами відповідного діапазону або з текстом у числовому полі). Зберегти таблицю в форматі CSV/TSV\*.

*\* Деякі програми електронних таблиць зберігають текст в форматі CSV/TSV в кодуванні UNICODE, що може викликати проблеми при читанні такого файлу. Щоб уникнути цього, переконайтеся, що експортований файл містить текст саме у кодуванні ASCII, або передбачте правильне читання UNICODE-тексту у програмі.*

6. Завантажити дані з вхідного файлу до колекції об’єктів, ім’я вхідного файлу має надаватися як *параметр* при виклику програми на виконання. Повідомлення про помилки читання мають виводитися в консоль та дописуватися в *журнальний файл*, ім’я файлу журналу має співпадати з іменем проекту та мати розширення «.log». Вивести зміст колекції в консоль. Відсортувати колекцію. Зберегти *відсортовану* колекцію в файлі у форматі JSON, ім’я файлу має бути утворене шляхом заміни розширення вхідного файлу з даними на «.json».

## Варіанти:

| **Номер варіанту** | **Об'єкт** |
| --- | --- |
| 6. | Вистава: назва п'єси, автор, театр, дата вистави, кількість актів, ціна квитка, музикальний супровід (так/ні) |

## **Лістинг програми:**

**Container\_Spectacle.java:**

import java.util.Collections;  
import java.util.ArrayList;  
import java.util.Scanner;  
import java.io.IOException;  
import java.io.File;  
import java.io.FileWriter;  
  
public class Container\_Spectacle {  
 public ArrayList<Spectacle> Constain = new ArrayList<>();  
  
 public Container\_Spectacle(){}  
 */\*\*  
 \* Method of reading data from a file  
 \** ***@param*** *name contains the name of the data file  
 \** ***@throws*** *WorkException is executed when we receive an error while reading the file  
 \*  
 \*/* public void Output(String name) throws WorkException {  
 try{  
 File file = new File (name);  
 Scanner scan\_file = new Scanner(file);  
 String[] parametrs = scan\_file.nextLine().split(",");  
 int counter = 1;  
 while(scan\_file.hasNextLine())  
 {  
 counter+=1;  
 try{  
 String data = scan\_file.nextLine();  
 Spectacle spectacle = new ID\_Spectacle(data, parametrs);  
 Constain.add(spectacle);  
  
 }  
 catch(WorkException Exeption){  
 System.*out*.print(Exeption.getLocalizedMessage());  
 Exeption.FileError(" ERROR IN READING CSV FILE IN ROW #" + counter);  
 }  
 }  
  
 }  
 catch(IOException Exeption)  
 {  
 throw new WorkException(Exeption.getLocalizedMessage());  
 }  
  
 }  
  
 */\*\*  
 \* The method of writing data to a json file  
 \** ***@param*** *name contains the name of the file to which the data will be written  
 \** ***@throws*** *WorkException is executed when we get an error while writing the file  
 \*  
 \*/* public void Input\_json(String name) throws WorkException {  
 try(FileWriter Input\_in\_file = new FileWriter(name)) {  
 String text = "[\n";  
 for(int i = 0; i < Constain.size(); i++)  
 {  
 text = text + " {" + (Constain.get(i).toJson()) + " }";  
 if(i != (Constain.size() - 1) )  
 text = text + ",";  
 text = text + "\n";  
 }  
 text += "]";  
  
 Input\_in\_file.write(text);  
 Input\_in\_file.flush();  
 }  
 catch(IOException Exeption)  
 {  
 throw new WorkException(Exeption.getLocalizedMessage());  
 }  
 }  
  
 */\*\*  
 \* Method that returns an object representation  
 \** ***@return*** *output container name  
 \*/* @Override  
 public String toString() {  
 return "Spectacle : { " + Constain + "}";  
 }  
  
 */\*\*  
 \* Method that sorts the container  
 \** ***@return*** *returns the sorted container  
 \*/* public void Sort(){  
 Collections.*sort*(Constain);  
 }  
}

**ID\_Spectacle.java:**

public class ID\_Spectacle extends Spectacle {  
 private static int *id\_static* = 0;  
 private int id = 0;  
 public ID\_Spectacle() {  
 this.id = ++*id\_static*;  
 }  
 */\*\*  
 \* A constructor that will receive data from a deadline  
 \** ***@param*** *parametrs a string containing data from  
 \** ***@param*** *value an array of parameter fields  
 \** ***@throws*** *WorkException returns a message if there is an error in the parameters  
 \*/* public ID\_Spectacle(String parametrs, String[] value) throws WorkException {  
 super(parametrs,value);  
 this.id = ++*id\_static*;  
 }  
 public int GET\_id() {  
 return id;  
 }  
  
 */\*\*  
 \* The method of returning the line number to the console  
 \** ***@return*** *returns the line number to the console during compilation  
 \*/* @Override  
 public String toString() {  
  
 return "{ID: " + GET\_id() +", "+ super.toString()+"}\n";  
 }  
 */\*\*  
 \* The method of returning the term number to the Json file  
 \** ***@return*** *returns to the console the line number in the Json file  
 \*/* @Override  
 public String toJson(){  
 return ("\"ID\": " + id + ", " + super.toJson());  
 }  
  
  
  
  
  
}

**Main.java:**

public class Main {  
 public static void main(String[] args) throws WorkException {  
 String name = args[0];  
 Container\_Spectacle Data\_spectacle = new Container\_Spectacle();  
 Data\_spectacle.Output(name + ".csv");  
 System.*out*.println(Data\_spectacle);  
 Data\_spectacle.Sort();  
 Data\_spectacle.Input\_json(name + ".json");  
  
 }  
  
}

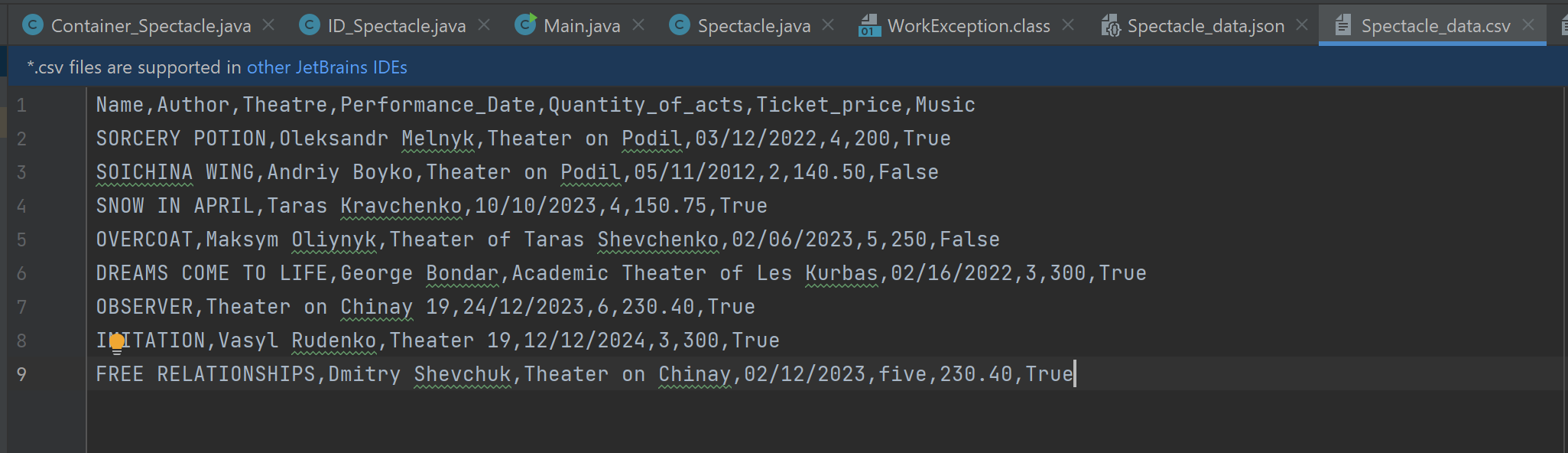
**Spectacle.java:**

import java.text.SimpleDateFormat;  
import java.util.Date;  
public class Spectacle implements Comparable<Spectacle> {  
 private String piece\_name;  
 private String author;  
 private String theatre;  
 private Date performance\_date;  
 private int act\_quantity;  
 private float prise\_ticket;  
 private boolean music;  
 private static final String[] *spectacle\_field* = {"Name","Author","Theatre","Performance\_Date","Quantity\_of\_acts","Ticket\_price","Music"};  
 private int[] spectacle\_order = new int[*spectacle\_field*.length];  
  
 */\*\*  
 \* Констурктор Spectacle по замовчуванню  
 \*/* public Spectacle(){  
  
 }  
  
 */\*\*  
 \* A constructor that stores data and noire from the term  
 \** ***@param*** *parameters comma-separated string of data  
 \** ***@param*** *value array with name fields order  
 \** ***@throws*** *WorkException returns when fields size is incorrect  
 \*/* public Spectacle(String parameters,String[] value) throws WorkException  
 {  
 if(parameters == null)  
 throw new WorkException(" EMPTY PARAMETERS\n");  
  
 String[] parameter = parameters.split(",");  
 if(value.length != 7 || parameter.length != 7)  
 {  
 throw new WorkException(" INCORRECT QUANTITY OF DATA\n");  
 }  
  
 SET\_spectacle\_data(value);  
  
 SET\_piece\_name(parameter[spectacle\_order[0]]);  
  
 SET\_author(parameter[spectacle\_order[1]]);  
  
 SET\_theatre(parameter[spectacle\_order[2]]);  
  
 SET\_performance\_date(parameter[spectacle\_order[3]]);  
  
 SET\_act\_quantity(parameter[spectacle\_order[4]]);  
  
 SET\_prise\_ticket(parameter[spectacle\_order[5]]);  
  
 SET\_music(parameter[spectacle\_order[6]]);  
  
  
 }  
  
 */\*\*  
 \* Method appointment order of fields  
 \** ***@param*** *value array with name fields order  
 \** ***@throws*** *WorkException returns when fields size is incorrect  
 \*/* void SET\_spectacle\_data(String[] value) throws WorkException{  
 for(int i = 0; i < *spectacle\_field*.length; ++i) {  
 for(int j = 0; j < spectacle\_order.length; ++j) {  
 if(*spectacle\_field*[i].equals(value[j])) {  
 spectacle\_order[i] = j;  
 }  
  
 }  
 }  
 }  
  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter  
 \*/* public String GET\_piece\_name()  
 {  
 return piece\_name;  
 }  
  
 */\*\*  
 \* Input setting method  
 \** ***@param*** *piece\_name contains the parameter data  
 \*/* public void SET\_piece\_name(String piece\_name)  
 {  
 this.piece\_name = piece\_name;  
 }  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter author  
 \*/* public String GET\_author()  
 {  
 return author;  
 }  
 */\*\*  
 \* Input setting method  
 \** ***@param*** *author contains the parameter data  
 \*/* public void SET\_author(String author)  
 {  
 this.author = author;  
 }  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter theatre  
 \*/* public String GET\_theatre()  
 {  
 return theatre;  
 }  
 */\*\*  
 \* Input setting method  
 \** ***@param*** *theatre contains the parameter data  
 \*/* public void SET\_theatre(String theatre)  
 {  
 this.theatre = theatre;  
 }  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter performance\_date  
 \*/* public Date GET\_performance\_date()  
 {  
 return performance\_date;  
 }  
  
 */\*\*  
 \* Input setting method  
 \** ***@param*** *performance\_date contains the parameter data  
 \** ***@throws*** *WorkException is thrown if the date was entered incorrectly  
 \*/* public void SET\_performance\_date(String performance\_date) throws WorkException {  
 try {  
 SimpleDateFormat form = new SimpleDateFormat("dd/MM/yyyy");  
  
 this.performance\_date = form.parse(performance\_date);  
 }  
 catch (java.text.ParseException Exeption)  
 {  
 throw new WorkException(Exeption.getLocalizedMessage() + " INCORRECT PERFORMANCE DATE\n");  
 }  
 }  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter act\_quantity  
 \*/* public int GET\_act\_quantity()  
 {  
 return act\_quantity;  
 }  
 */\*\*  
 \* Input setting method  
 \** ***@param*** *act\_quantity contains the parameter data  
 \** ***@throws*** *WorkException is thrown if the number was entered incorrectly  
 \*/* public void SET\_act\_quantity(String act\_quantity) throws WorkException  
 {  
 try  
 {  
 this.act\_quantity = (Integer.*parseInt*(act\_quantity));  
 }  
 catch (NumberFormatException Exeption) {  
 throw new WorkException(Exeption.getLocalizedMessage() + " INCORRECT ACT QUANTITY: MUST BE A NUMBER\n");  
 }  
 }  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter prise\_ticket  
 \*/* public float GET\_prise\_ticket()  
 {  
  
 return prise\_ticket;  
 }  
 */\*\*  
 \* Input setting method  
 \** ***@param*** *prise\_ticket contains the parameter data  
 \** ***@throws*** *WorkException is thrown if the number was entered incorrectly  
 \*/* public void SET\_prise\_ticket(String prise\_ticket) throws WorkException  
 {  
 try {  
 this.prise\_ticket = (Float.*parseFloat*(prise\_ticket));  
 }  
 catch (NumberFormatException Exeption)  
 {  
 throw new WorkException(Exeption.getLocalizedMessage() + " INCORRECT PRICE OF TICKET: MUST BE A NUMBER\n");  
 }  
  
 }  
 */\*\*  
 \* Output setting method  
 \** ***@return*** *returns a parameter music  
 \*/* public boolean GET\_music()  
 {  
 return music;  
 }  
 public void SET\_music(Boolean music)  
 {  
 this.music = music;  
 }  
 public void SET\_music(String music)  
 {  
 SET\_music(Boolean.*parseBoolean*(music));  
  
 }  
  
 */\*\*  
 \*  
 \** ***@param*** *S the object to be compared.  
 \** ***@return*** *returns the adjusted object  
 \*/* @Override  
 public int compareTo(Spectacle S) {  
 int res = this.piece\_name.compareTo(S.piece\_name);  
 if(res == 0)  
 {  
 Integer temp = this.act\_quantity;  
 res = temp.compareTo(S.act\_quantity);  
 }  
 return res;  
 }  
  
 */\*\*  
 \* The method returns the string parameters to the console  
 \** ***@return*** *returns string parameters  
 \*/* @Override  
 public String toString() {  
 return "Name: " + piece\_name +  
 ", Author: " + author +  
 ", Theatre: " + theatre+  
 ", Performance date: " + performance\_date+  
 ", Quantity of acts: " + act\_quantity+  
 ", Ticket price: " + prise\_ticket+  
 ", Music: " + music ;  
  
 }  
  
 */\*\*  
 \* The method returns the string parameters to write to Json  
 \** ***@return*** *returns string parameters  
 \*/* public String toJson() {  
 return ("\"Name\": \'" + piece\_name + '\'' +  
 ", \"Author\": \'" + author + '\'' +  
 ", \"Theatre\": \'" + theatre+ '\'' +  
 ", \"Performance date\": \'" + performance\_date+ '\'' +  
 ", \"Quantity of acts\": " + act\_quantity +  
 ", \"Ticket price\": " + prise\_ticket +  
 ", \"Music\": " + music);  
  
}  
}

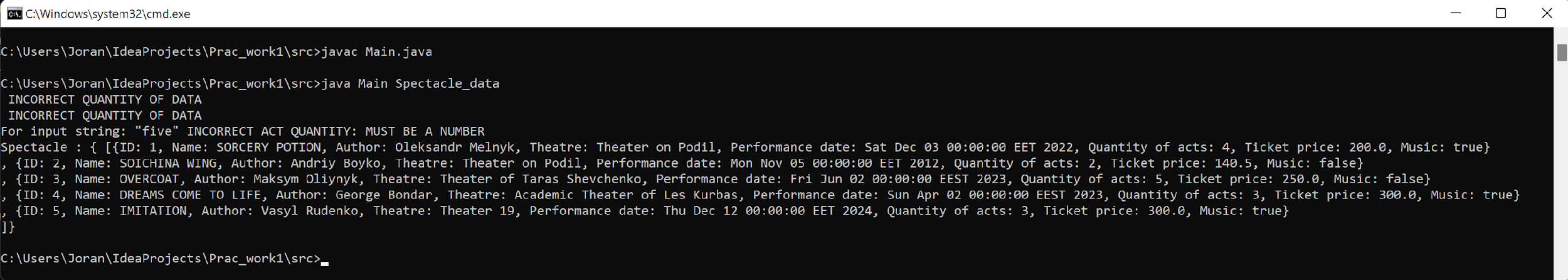
**WorkException.java:**

import java.io.FileWriter;  
  
import java.io.IOException;  
import java.time.LocalDateTime;  
  
public class WorkException extends Exception  
{  
  
 private static String *error\_fileName*;  
 private String error\_info;  
  
 */\*\*  
 \*  
 \** ***@param*** *info contains error messages  
 \*  
 \*/* public WorkException(String info)  
 {  
 super(info);  
 *error\_fileName*= "Errors.log";  
  
  
 }  
  
 */\*\*  
 \* A method of writing an error message while the program is running  
 \** ***@param*** *error contains the error message  
 \*/* public void FileError(String error)  
 {  
 try  
 {  
 if(*error\_fileName* != null){  
 FileWriter Input = new FileWriter(*error\_fileName*,true);  
 Input.write( "TIME OF ERROR: "+LocalDateTime.*now*() + ",TYPE OF ERROR: "+error\_info+" "+error+"\n");  
 Input.close();  
 }  
 else  
 {  
 throw new NullPointerException(" FILE NAME IS EMPTY\n");  
 }  
 }  
 catch(IOException Exeption)  
 {  
 System.*out*.println(Exeption.getLocalizedMessage());  
 }  
 catch(NullPointerException Exeption)  
 {  
 System.*out*.println(Exeption.getLocalizedMessage());  
 }  
  
  
 }  
}

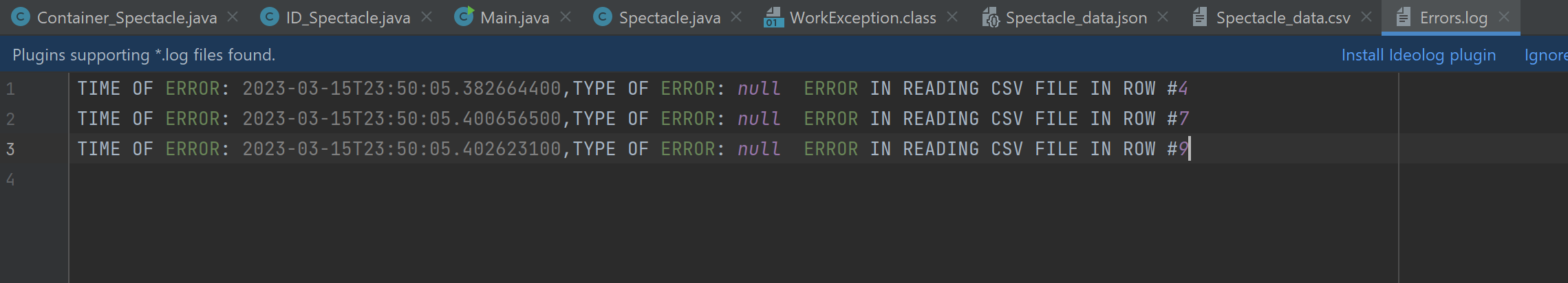
**Spectacle\_data.csv:**

****

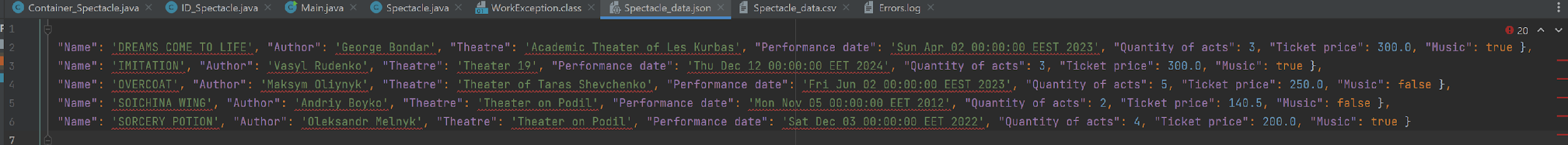
**Результати:**

****

**Errors.log:**

****

**Spectacle\_data.json:**

****

**Висновок:** Ми ознайомилися з особливостями створення класів та об’єктів мовою програмування Java, вивчили засоби узагальненого програмування мови Java на прикладі колекцій об’єктів; розібратися з особливостями виклику та обробки виняткових ситуацій, навчилися використовувати сторонні бібліотеки класів, сформували вміння оброблювати аргументи командної строки та навички завантаження файлів даних, представлених в CSV/TSV та JSON.

## **Контрольні запитання:**

1. **Які значення має термін “платформа”? Наведіть приклади платформ для кожного значення.**

Апаратна платформа - це набір апаратних компонентів, які підтримують виконання програмного забезпечення. Приклади апаратних платформ: PC, Mac, мобільні телефони, планшети

Платформа операційна система - це програмна платформа, яка забезпечує спільне середовище для виконання програмного забезпечення на апаратній платформі. Приклади операційних систем: Windows, macOS, Linux, Android, iOS, Chrome OS

Процесорна платформа або команда архітектурної системи пов’язана зі здатністю процесора виконувати один і той же код, при цьому ігноруються відмінності у внутрішніх пристроях процесорів та їх інтерфейсах. Прикладом платформи процесора є IA-32.

Комп'ютерна платформа — сукупність комп'ютерів, сумісних між собою в тому або іншому сенсі.

1. **Що таке “кросплатформність”? На яких рівнях вона може бути забезпечена?**

Кросплатформність — властивість програмного забезпечення працювати більш ніж на одній програмній (в тому числі — операційній системі) або апаратній платформі; технології, що дозволяють досягти такої властивості. Кросплатформність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення.

“кросплатформність” може бути забезпечена кросплатформність мов програмування, кросплатформність на рівні середовища виконання, кросплатформність на апаратному рівні

**3. В яких поширених мовах програмування реалізовано кожен з рівнів кросплатформності?**

В мовах програмування С#,Java, Python, SQL реалізовано кожен з рівнів кросплатформності

**4. Наведіть приклади кросплатформного програмного забезпечення для кожного рівня кросплатформності та вкажіть, на яких мовах воно написано.**

Кросплатформного програмного забезпечення - Qt(С++), GTK+(С, C++, JS, Python), FLTK(C++), Boost(C++), OpenGL(C, C++), SDL(C++), OpenAL(C), OpenCL C, C++)

**5. Надайте характеристику CORBA та розкрийте її призначення.**

CORBA — загальна архітектура брокера об'єктних запитів) — це запропонований консорціумом OMG технологічний стандарт розробки розподілених застосунків.

Завдання CORBA — інтегрувати розподілені системи, дати можливість програмам, що написані різними мовами та працюють у різних вузлах мережі, взаємодіяти одна з одною так само просто, наче вони знаходяться в адресному просторі одного процесу.

**6. Які засоби розробки та інші компоненти складають екосистему Java?**

Віртуальна машина Java (JVM) надає високопродуктивну базу для виконання програм, написаних на мові Java (та навіть і на інших мовах).

Integrated Development Environments (IDEs) - це програмні засоби, які надають інтегровану середовище розробки, яке допомагає програмістам в створенні програм на мові Java

Java Development Kit (JDK) - це набір інструментів для розробки програм на мові Java, який включає в себе компілятор, віртуальну машину та багато інших інструментів.

**7. Опишіть сутність та призначення інтерфейсу в Java.**

Інтерфейс у мові програмування Java — це абстрактний тип, який використовується для визначення поведінки, яку класи повинні реалізовувати. Інтерфейси схожі до протоколів. Інтерфейси оголошуються за допомогою ключового слова interface, та можуть містити тільки прототипи методів і оголошення констант (або змінних, які оголошені як static чи final). Усі методи інтерфейсу не можуть містити реалізації тіл методів в усіх версіях нижче Java 8. Починаючи з Java 8, default та static методи можуть мати реалізацію за замовчуванням при визначенні інтерфейсу.

**8. Що таке автоматичне пакування примітивних типів в Java та для чого воно потрібно?**

*Автоупакування* – це процес автоматичного перетворення (інкапсулювання) примітивного типу (int, double, boolean тощо) у відповідну йому оболонку типу (Integer, Double, Boolean тощо) у випадку коли потрібно отримати об’єкт даного типу.  
Наприклад, якщо для примітивного типу int потрібно отримати об’єкт, то це буде об’єкт типу Integer. Відповідно конвертування int в Integer називається автоупакуванням.

втоупакування та авторозпакування виконуються у наступних випадках:

* при виконанні присвоєння (**=**);
* при передачі параметру в метод;
* при поверненні з методу;
* при неявному перетворенні у виразах.

**9. Що таке синонімія в Java? Порівняйте її з аналогічним явищем в C/C++.**

Синонімія – явище, коли змінні посилаються на одину і ту же ділянку пам'яті.

Синонімія шкідлива з точки зору читабельності (читачі програми змушені пам'ятати про синонімію). В С++ синоніми створюються за допомогою покажчиків та змінних-посилань.

**10. Для яких цілей використовується JavaDoc? Перелічить його основні елементи.**

Javadoc — генератор документації в HTML-форматі з коментарів вихідного коду на Java від Sun Microsystems. Цей формат був обраний для забезпечення можливості зв'язати воєдино пов'язані документи за допомогою посилань. Коментарі javadoc стали «де факто» стандартом для документування створених Java-класів. Більшість середовищ розробки, таких як Eclipse та Netbeans автоматично генерують документацію за допомогою javadoc.

| **Дескриптор *Параметр*** | **Призначення** | **Використання** | **З версії** |
| --- | --- | --- | --- |
| **@author** *Опис* | Ідентифікує автора. | Клас, інтерфейс | 1.0 |
| {**@code** *Фрагмент коду*} | Відображає інформацію «як є», без обробки HTML-стилю, проте з використанням шрифту коду. | Скрізь | 1.5 |
| **@deprecated** *Опис* | Описує застарілий клас або його член. При використанні бажано вказувати, чим було замінено функціональність. | Клас, інтерфейс, метод, змінна | 1.0 |
| {**@docRoot**} | Визначає шлях до корінного каталогу поточної директорії. | Скрізь | 1.3 |
| **@exception** *Назва виключення Пояснення* **@throws** *Назва виключення Пояснення* | Описує виключення, що може генерувати метод. | Метод | 1.0 та 1.2 відповідно |
| {**@inheritDoc**} | Наслідує коментар безпосередньо від [предка](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). | Підмінений метод | 1.4 |
| {**@link** *Пакет. Клас#Член Опис*} | Вставляє вбудоване посилання на іншу тему. | Скрізь | 1.2 |
| {**@linkplain**} | Вставляє вбудоване посилання на іншу тему, проте відображає його у вигляді відкритої форми шрифту. | Скрізь | 1.4 |
| {**@literal** *Опис*} | Відображає інформацію «як є», без обробки HTML-стилю. | Скрізь | 1.5 |
| **@param** *Назва параметру Опис* | Документує параметр методу, класу чи інтерфейсу. При використанні не потрібно вказувати тип параметру. | Клас, інтерфейс, метод | 1.0 |
| **@return** *Опис* | Описує значення, що повертає метод. | Метод | 1.0 |
| **@see** *Тема* **@see** *Пакет. Клас#Член Опис* | Визначає посилання на іншу тему документації. | Клас, інтерфейс, метод, змінна | 1.0 |
| **@serial** *Опис* | Документує змінну, що серіалізується за замовчуванням. | Змінна | 1.2 |
| **@serialData** *Опис* | Документує дані, записані методами writeObject() та writeExternal(). | Метод | 1.2 |
| **@serialField** *Опис* | Документує компонент ObjectStreamField. | Змінна | 1.2 |
| **@since** *Версія* | Показує, коли було введено дану функціональність. | Скрізь | 1.1 |
| {**@value**} {**@value** *Пакет. Клас#Змінна*} | Повертає значення статичної змінної. | Статична змінна | 1.4 |
| **@version** *Версія* | Визначає версію класу і може використовуватися в ньому лише раз. | Клас, інтерфейс | 1.0 |