MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL TECHNICAL UNIVERSITY
"KHARKIV POLYTECHNIC INSTITUTE"

GUIDELINES

to the laboratory work for the course “Advanced Java programming course”

for students of specialty

121 “Software engineering” and 122 “Computer Science”

Approved
by the Editorial and Publishing
Council of the University,
Protocol №1, February 13, 2025.

Kharkiv
NTU “KhPI”
2025

Guidelines to the laboratory work for the courses “Advanced Java programming course” for students of specialties 121 “Software engineering” and 122 “Computer Science” // Author: L. V. Ivanov. – Kharkiv: NTU “KhPI”, 2025. – 92 p.

Author: L. V. Ivanov

Reviewer: docent A. A. Pashnev

Department of Software Engineering and Management Intelligent Technologies

CONTENTS

Laboratory training 1 Working with big numbers and data sets 6
1 Training tasks 6
1.1 Individual task 6
1.2 Finding an integer power 6
1.3 Filtering and sorting 6
1.4 Type conversion function library 7
2 Guidelines 7
3 Sample Programs 7
3.1 Finding factorials 7
3.2 The product of the entered numbers 12
3.3 Obtaining a table of prime numbers using data streams 12
3.4 “Country” and “Census” classes 13
4 Quiz 20
Laboratory training 2 Working with dates and text. Localization 22
1 Training tasks 22
1.1 Individual task 22
1.2 Date entering 23
1.3 Phone number verification 23
1.4 Obtaining an array of substrings (advanced task) 23
2 Guidelines 23
3 Sample programs 24
3.1 View available localizations 24
3.2 The First Substring that Matches the Pattern 24
3.3 Checking the phone number format 24
3.4 Output words of sentence 25
3.5 "Country" and "Census" Classes 26
4 Quiz 33

Laboratory training 3 Advanced features of working with files 34
1 Training tasks 34
1.1 Individual assignment 34
1.2 List of files of all subdirectories 34
1.3 Working with text files using the Stream API 34
1.4 Implementation of an array of points through a two-dimensional array 35
1.5 Use of SAX and DOM technologies (additional task) 35
1.6 The Implementation of the comparator interface 35
2 Guidelines 35
3 Sample programs 36
3.1 Using DOM technology 36
3.2 Reading from a file and sorting strings 37
3.3 Serialization and deserialization using the XStream library 38
3.4 "Country" and "Census" classes 42
4 Quiz 50
Laboratory training 4 Development of GUI applications 52
1 Training tasks 52
1.1 Individual task 52
1.2 Using ObservableList 52
1.3 Mini calculator 53
1. Dictionary (advanced task) 53
2 Guidelines 53
3 Sample programs 53
3.1 Working with ObservableList 53
3.2 Text fields and buttons 54
3.3 Using radio buttons 56
3.4 Working with the file dialogs 57
3.5 GUI Application for processing data about censuses 58
4 Quiz 70

Laboratory training 5 Reflection. Creation of database applications 72
1 Training tasks 72
1.1 Creating a database console application 72
1.2 View all class fields 72
1.3 Creating a database GUI application (advanced task) 73
1.4 Obtaining data about method annotations (advanced task) 73
2 Guidelines 73
3 Sample programs 73
3.1 Obtaining data about class methods 73
3.2 Calculating the results of Math class functions 74
3.3 Output of information about methods marked with a certain annotation 75
3.4 Database application for working with census data 76
4 Quiz 89
List of sources 91
Recommended reading: 91
Internet resources: 91

LABORATORY TRAINING 1
Working with big numbers and data sets

1 Training tasks

1.1 Individual task

Design and implement classes to represent the entities of the third laboratory training of the Java Basic Programming Course. The solution should be based on the previously created classes.

The program must demonstrate:

– reproduction of the functionality of laboratory trainings No. 3 and No. 4 of the Java Basic Programming Course;

– using Stream API tools for all sequence processing and output functions;

– testing methods of individual classes using JUnit.

1.2 Finding an integer power

Write a program that fills a number of type `BigInteger` with random digits and calculates the integer power of this number. For the result, use `BigInteger` type. Implement two ways: using a `pow()` method and a function that provides multiplication of long integers. Compare the results.

Provide testing of class methods using JUnit.

1.3 Filtering and sorting

Create a list of objects of type `BigDecimal`. Fill the list with random values. Sort by decreasing absolute value. Find the product of positive numbers. Implement three approaches:

– using loops and conditional statements (without facilities added in Java 8);

– without explicit loops and branches, using functions that have been defined in the Java Collection Framework interfaces since Java 8;

– using Stream API tools.

Provide testing of classes using JUnit.

1.4 Type conversion function library

Use the Stream API to organize a search for all divisors of a positive integer. Create a separate static function that accepts an integer and returns an array of integers. Inside the function create an `IntStream` object. Apply `range()` function and filter. Do not use explicit loops.

2 Guidelines

The goal of this laboratory training is obtaining practical skills in working with big numbers and data sets.

The following questions should be explored:

– general features of the Java SE platform;

– using `BigInteger` and `BigDecimal` types;

– problems of storage and processing of data sets;

– using the Java collection framework for data processing;

– using the Java 8 Stream API;

– testing in Java; using JUnit.

Necessary theoretical concepts and syntax constructs are discussed in lectures on the topics “Working with Big Numbers”, “Using the Java Collection Framework” and “Using the Java 8 Stream API”.

Most tasks can be implemented based on sample programs discussed below.

3 Sample Programs

3.1 Finding factorials

The traditional mathematical problem of calculating the factorial for large integers causes difficulties associated with restrictions on the size of the result. For `int` type, int the maximum certain value is `12!`, for the `long` type it is `20!` For values `171!` and more, even an approximate value cannot be obtained using `double`. Using `BigInteger` and `BigDecimal` allow you to get factorials of large numbers. The size of the result is actually limited by the display capabilities of the console window.

The following program calculates the factorial of integers using different types: `long, double, BigInteger` and `BigDecimal`.

```
package ua.inf.iwanoff.java.advanced.first;

import java.math.BigDecimal;
import java.math.BigInteger;
import java.util.Scanner;

public class Factorial {
    public static long factorialLong(int n) {
        long result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }

    public static double factorialDouble(int n) {
        double result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }

    public static BigInteger factorialBigInteger(int n) {
        BigInteger result = BigInteger.ONE;
        BigInteger current = BigInteger.ZERO;
        for (int i = 1; i <= n; i++) {
            result = result.multiply(current = current.add(BigInteger.ONE));
        }
        return result;
    }

    public static BigDecimal factorialBigDecimal(int n) {
        BigDecimal result = BigDecimal.ONE;
        BigDecimal current = BigDecimal.ZERO;
        for (int i = 1; i <= n; i++) {
            result = result.multiply(current = current.add(BigDecimal.ONE));
        }
        return result;
    }

    public static void main(String[] args) {
        int n = new Scanner(System.in).nextInt();
        System.out.println(factorialLong(n));
        System.out.println(factorialDouble(n));
        System.out.println(factorialBigInteger(n));
        System.out.println(factorialBigDecimal(n));
    }
}
```

The execution of the program for large values of `n` takes a long time.

Manual testing is not enough to verify the certainty of the results. To test these functions, we can use the capabilities of the JUnit library. In advance, we need to create a separate test folder and mark it as the root of the tests (`Mark`

`Directory as | Test Sources Root` function of the context menu). Next, you can generate tests: the `Code | Generate... | Test...` function of main menu. In the dialog box, select the functions for which tests should be created. In our case, these are all functions except `main()`. The following code will be generated:

```
package ua.inf.iwanoff.java.advanced.first;

import static org.junit.jupiter.api.Assertions.*;

class FactorialTest {

    @org.junit.jupiter.api.Test
    void factorialLong() {
    }

    @org.junit.jupiter.api.Test
    void factorialBigInteger() {
    }

    @org.junit.jupiter.api.Test
    void factorialBigDecimal() {
    }

    @org.junit.jupiter.api.Test
    void factorialDouble() {
    }
}
```

Errors related to the use of JUnit tools are corrected according to the instructions that were described above.

To test the `factorialDouble()` function, it is necessary to add an auxiliary `compareDoubles()` function, as well as to create constants for some factorial values. The code of the `FactorialTest.java` file will be as follows:

```
package ua.inf.iwanoff.java.advanced.first;
package ua.inf.iwanoff.java.advanced.first;

import java.math.BigDecimal;

import static org.junit.jupiter.api.Assertions.*;

class FactorialTest {
    public static final String FACTORIAL_5 = "120";
    public static final String FACTORIAL_50 =
            "30414093201713378043612608166064768844377641568960512000000000000";
    public static final String FACTORIAL_500 =
"1220136825991110068701238785423046926253574342803192842192413588385845373153881 9" +
"9760549644750220328186301361647714820358416337872207817720048078520515932928547 7" +
"9075719393306037729608590862704291745478824249127263443056701732707694610628023 1" +
"0452644218878789465754777149863494367781037644274033827365397471386477878495438 4" +
"8959553753799042324106127132698432774571554630997720278101456108118837370953101 6" +
"3563244329870295638966289116589747695720879269288712817800702651745077684107196 2" +
"4390394322536422605234945850129918571501248706961568141625359056693423813008856 2" +
```

```
"492468915641267756544818865065938479517753608940057452389403357984763639449053 13" +
"062323749066445048824665075946735862074637925184200459369692981022263971952597 19" +
"094521782333175693458150855233282076282002340262690789834245171200620771464097 94" +
"561161276291459512372299133401695523638509428855920187274337951730145863575708 28" +
"355780158735432768888680120399882384702151467605445407663535984174430480128938 31" +
"389688163948746965881750450692636533817505547812864000000000000000000000000000 0" +
"000000000000000000000000000000000000000000000000000000000000000000000000000000 0" +
"000000000000000";

    public static final BigDecimal EPS = BigDecimal.ONE;

    private boolean compareDoubles(double d, String s) {
        return new BigDecimal(d).subtract(new BigDecimal(s)).abs()
                                                            .compareTo(EPS) <= 0;
    }

    @org.junit.jupiter.api.Test
    void factorialLong() {
        assertEquals(Factorial.factorialLong(5) + "", FACTORIAL_5);
        assertEquals(Factorial.factorialLong(50) + "", FACTORIAL_50);
        assertEquals(Factorial.factorialLong(500) + "", FACTORIAL_500);
    }

    @org.junit.jupiter.api.Test
    void factorialDouble() {
        assertTrue(compareDoubles(Factorial.factorialDouble(5), FACTORIAL_5));
        assertTrue(compareDoubles(Factorial.factorialDouble(50), FACTORIAL_50));
        assertTrue(compareDoubles(Factorial.factorialDouble(500), FACTORIAL_500));
    }

    @org.junit.jupiter.api.Test
    void factorialBigInteger() {
        assertEquals(Factorial.factorialBigInteger(5) + "", FACTORIAL_5);
        assertEquals(Factorial.factorialBigInteger(50) + "", FACTORIAL_50);
        assertEquals(Factorial.factorialBigInteger(500) + "", FACTORIAL_500);
    }

    @org.junit.jupiter.api.Test
    void factorialBigDecimal() {
        assertEquals(Factorial.factorialBigDecimal(5) + "", FACTORIAL_5);
        assertEquals(Factorial.factorialBigDecimal(50) + "", FACTORIAL_50);
        assertEquals(Factorial.factorialBigDecimal(500) + "", FACTORIAL_500);
    }
}
```

The values of `50!` and `500!` were obtained from the page https://zeptomath.com/calculators/factorial.php and are used as a reference in our tests.

As expected, the correct values of `50!` and `500!` can be obtained only using `BigInteger` and `BigDecimal`.

Assume that we want to create a program that defines two variables and calculates their arithmetic mean (average).

A new project should be created. A new package called `ua.inf.iwanoff.java.first` will contain the `Average` class. Within class wizard window, generation of `main()` function should be checked. The following code will be generated:

```
package ua.inf.iwanoff.java.first;

public class Average {

    public static void main(String[] args) {

    }

}
```

Now, instead of the comment inside the `main()` function, we create an object of the `Scanner` class, describe two variables, for example, `a` and `b`, read their values from the keyboard and calculate the value of `c`, which is their arithmetic mean.

```
Scanner s = new Scanner(System.in);
double a = s.nextDouble();
int b = s.nextInt();
double c = (a + b) / 2;
```

To enable the use of the `Scanner` class before the description of the `Average` class, we add an import statement (`import java.util.Scanner;`). The definition (description) of a variable, as in C++, can be combined with the assignment of an initial value (initialization). In this case, we create a variable `s` of type `Scanner`, `b` of type `int`, and also variables `a` and `c` of type `double`.

Necessary result soon calculated within initialization of `c` variable. This result should be shown using `System.out.println()` function.

```
System.out.println("Arithmetical mean: " + c);
```

The "plus" character in this line means concatenation of stings String type constant `"Arithmetic mean: "` will be concatenated with string representation of `c`.

The full text of our class will be as follows.

```
package ua.inf.iwanoff.java.first;

import java.util.Scanner;
```

```
public class Average {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        double a = s.nextDouble();
        int b = s.nextInt();
        double c = (a + b) / 2;
        System.out.println("Arithmetic mean: " + c);
    }

}
```

The result appears in the console view:

```
Arithmetical mean: 2.4
```

3.2 The product of the entered numbers

In the example below, integers are entered, displayed as a decrease, and their product is calculated. The entering ends with zero:

```
package ua.inf.iwanoff.java.fourth;

import java.util.*;

public class Product {

    public static void main(String[] args) {
        Queue<Integer> queue = new PriorityQueue<>(100,
                                                   new Comparator<Integer>() {
            @Override
            public int compare(Integer i1, Integer i2) {
                return -Double.compare(i1, i2);
            }
        });
        Scanner scanner = new Scanner(System.in);
        Integer k;
        do {
            k = scanner.nextInt();
            if (k != 0) {
                queue.add(k);
            }
        }
        while (k != 0);
        int p = 1;
        while ((k = queue.poll()) != null) {
            p *= k;
            System.out.print(k + " ");
        }
        System.out.println();
        System.out.println(p);
    }
}
```

3.3 Obtaining a table of prime numbers using data streams

The following program allows you to get a table of prime numbers in a given range. To obtain simple numbers, it is advisable to use `IntStream`:

```
package ua.inf.iwanoff.java.advanced.first;

import java.util.stream.IntStream;

public class PrimeFinder {
    private static boolean isPrime(int n) {
        return n > 1 && IntStream.range(2, n - 1).noneMatch(k -> n % k == 0);
    }

    public static void printAllPrimes(int from, int to) {
        IntStream primes = IntStream.range(from,
                                           to + 1).filter(PrimeFinder::isPrime);
        primes.forEach(System.out::println);
    }

    public static void main(String[] args) {
        printAllPrimes(6, 199);
    }
}
```

The `isPrime()` method checks whether the number n is prime. For numbers greater than 1, a set of consecutive integers is formed, for each of which it is checked whether n is divisible by this number. In the `printAllPrimes()` method, we form a stream of simple numbers using a filter and output the numbers using the `forEach()` method.

3.4 “Country” and “Census” classes

In the examples from the Java Basic Programming Course, class hierarchies to represent country and population censuses were considered. In laboratory training # 3 of this course, an abstract class `AbstractCountry` was created, as well as concrete classes `Census` and `CountryWithArray`. Next, in laboratory training # 4, classes `CountryWithList` and `CountryWithSet` were created.

Now, using the `CountryWithList` and `Census` classes, we will create an application that reproduces the search and sorting implemented in the examples of the specified labs through the use of the Stream API.

We can create a new package called `ua.inf.iwanoff.java.advanced.first` within previously created project. We add `CensusChecker` class to the package. It is advisable to create the `containsWord()` method, implementing it with the help of streams. For example, the class code could be as follows:

```
package ua.inf.iwanoff.java.advanced.first;
```

```
import ua.inf.iwanoff.java.third.Census;
import java.util.Arrays;

/**
 * Provides a static method to search for a word in a comment
 * The method uses StreamAPI facilities
 */
public class CensusChecker {
    /**
     * Checks whether the word can be found in the comment text
     * @param census reference to a census
     * @param word a word that should be found in a comment
     * @return {@code true}, if the word is contained in the comment text
     *         {@code false} otherwise
     */
    public static boolean containsWord(Census census, String word) {
        return Arrays.stream(census.getComments().split("\s"))
                .anyMatch(s -> s.equalsIgnoreCase(word));
    }
}
```

It is also possible to define `main()` function to test class, but a better approach is to use the capabilities of unit testing (JUnit).

In advance, create a folder test in the root of the project and mark it as the root of the tests (`Mark Directory as | Test Sources Root` function of the context menu). In the code window, we select the name of the class and using the context menu `Generate... | Test...` select the functions for which test methods should be generated. In our case, this is the `containsWord()` method. IntelliJ IDEA automatically generates all necessary parallel packages of the test branch and creates the class called `CensusWithStreamsTest`. It looks like this:

```
package ua.inf.iwanoff.java.advanced.first;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class CensusCheckerTest {

    @Test
    void containsWord() {
    }
}
```

If errors are highlighted in the generated code, we correct them as described above.

Now we can add the necessary testing. The code of the file `CensusCheckerTest.java` will be as follows:

```
package ua.inf.iwanoff.java.advanced.first;

import ua.inf.iwanoff.java.third.Census;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class CensusCheckerTest {

    @Test
    void containsWord() {
        Census census = new Census();
        census.setComments("The first census in independent Ukraine");
        assertTrue(CensusChecker.containsWord(census, "Ukraine"));
        assertTrue(CensusChecker.containsWord(census, "FIRST"));
        assertFalse(CensusChecker.containsWord(census, "rain"));
        assertFalse(CensusChecker.containsWord(census, "censuses"));
    }
}
```

After completing the tests, we will receive a successful exit code. If the expected results are changed in the code, the tests will throw an exception and the assertion that failed will be underlined in the code.

We create a separate class to work with the list of censuses via the Stream API:

```
package ua.inf.iwanoff.java.advanced.first;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.Census;
import static ua.inf.iwanoff.java.advanced.first.CensusChecker.containsWord;

import java.util.Comparator;

/**
 * Class for processing data about the country in which the census is being
conducted.
 * Stream API tools are used to process sequences
 */
public class CountryProcessorWithStreams {
    /**
     * Returns the population density for the specified year
     *
     * @param country reference to a country
     * @param year specified year (e.g. 1959, 1979, 1989, etc.)
     * @return population density for the specified year
     */
    public static double density(CountryWithArrayList country, int year) {
        Census census = country.getList().stream()
                .filter(c -> c.getYear() == year)
                .findFirst().orElse(null);
        return census == null ? 0 : census.getPopulation() / country.getArea();
    }

    /**
     * Finds and returns a year with the maximum population
     *
```

```
     * @param country reference to a country
     * @return year with the maximum population
     */
    public static int maxYear(CountryWithArrayList country) {
        return country.getList().stream()
                .max(Comparator.comparing(Census::getPopulation))
                .get().getYear();
    }

    /**
     * Creates and returns an array of censuses
     * with the specified word in the comments
     *
     * @param country reference to a country
     * @param word a word that is found
     * @return array of censuses with the specified word in the comments
     */
    public static Census[] findWord(CountryWithArrayList country, String word)
{
        return country.getList().stream()
                .filter(c -> containsWord(c, word))
                .toArray(Census[]::new);
    }

    /**
     * Sorts the sequence of censuses by population
     *
     * @param country reference to a country
     */
    public static void sortByPopulation(CountryWithArrayList country) {
        country.setList(country.getList().stream()
                .sorted()
                .toList());
    }

    /**
     * Sorts the sequence of censuses in the alphabetic order of comments
     *
     * @param country reference to a country
     */
    public static void sortByComments(CountryWithArrayList country) {
        country.setList(country.getList().stream()
                .sorted(Comparator.comparing(Census::getComments))
                .toList());
    }
}
```

The given code uses a function call: `toArray(Census[]::new)`. This ensures that an array of the required type (references to `Census`) is created, rather than an array of references to `Object`, which is returned by the corresponding function without parameters.

We add a `CountryWithStreams` class for testing. This is done in the same way as for `CensusChecker`. It is advisable to choose all methods, as well as `setUp/@Defore`. We will get the following code:

```
package ua.inf.iwanoff.java.advanced.first;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

class CountryProcessorWithStreamsTest {

    @BeforeEach
    void setUp() {
    }

    @Test
    void density() {
    }

    @Test
    void maxYear() {
    }

    @Test
    void findWord() {
    }

    @Test
    void sortByPopulation() {
    }

    @Test
    void sortByComments() {
    }
}
```

Since it is necessary to perform several tests on the object and these tests must be independent, it is advisable to create the object before executing each test method. For this, we need to add a method with `@BeforeEach` annotation. The corresponding method can also be generated automatically (`Generate... | SetUp Method` in the context menu). We create a new country object in the method called `setUp()`. We create the appropriate field of type `CountryWithStreams` manually. The object will be used in test methods.

For convenient testing of functions related to searching and sorting, we can create `getYears()` function that retrieves an array of years from an array of censuses. This static function will use a static variable index to fill specific array items. The variable cannot be local because we are using it in a lambda expression. We get the following code:

```
package ua.inf.iwanoff.java.advanced.first;
```

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.Census;

import java.util.Arrays;

class CountryProcessorWithStreamsTest {
    private CountryWithArrayList country;
    static int index;

    static int[] getYears(Census[] censuses) {
        int[] years = new int[censuses.length];
        index = 0;
        Arrays.stream(censuses).forEach(c -> years[index++] = c.getYear());
        return years;
    }

    @BeforeEach
    void setUp() {
        country = new CountryWithArrayList();
        country.setArea(603628);
        country.addCensus(1959, 41869000,
                          "The first census after World War II");
        country.addCensus(1970, 47126500, "Population increases");
        country.addCensus(1979, 49754600, "No comments");
        country.addCensus(1989, 51706700, "The last soviet census");
        country.addCensus(2001, 48475100,
                          "The first census in the independent Ukraine");
    }

    @Test
    void density() {
        assertEquals(CountryProcessorWithStreams.density(country, 1979),
                                                          82.42593120266125);
    }

    @Test
    void maxYear() {
        assertEquals(CountryProcessorWithStreams.maxYear(country), 1989);
    }

    @Test
    void findWord() {

        assertArrayEquals(getYears(CountryProcessorWithStreams.findWord(
                    country, "census")), new int[] { 1959, 1979, 1989, 2001 });
    }

    @Test
    void sortByPopulation() {
        CountryProcessorWithStreams.sortByPopulation(country);
        assertArrayEquals(getYears(country.getCensuses()),
                                  new int[] { 1959, 1970, 2001, 1979, 1989 });
    }

    @Test
    void sortByComments() {
        CountryProcessorWithStreams.sortByComments(country);
```

```
        assertArrayEquals(getYears(country.getCensuses()),
                                    new int[] { 1970, 1989, 2001, 1959, 1979 });
    }
}
```

Arrays of years corresponding to correct sorting and searching results were manually prepared.

We demonstrate the features new classes in the `main()` function of the `CountryWithStreamsDemo` class:

```
package ua.inf.iwanoff.java.advanced.first;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.StringRepresentations;
import ua.inf.iwanoff.java.third.Census;

import static ua.inf.iwanoff.java.third.CountryDemo.setCountryData;

/**
 * The class demonstrates data processing using the StreamAPI
 */
public class CountryWithStreamsDemo {

   /**
     * Displays census data that contains a certain word in comments
     * @param country reference to a country
     * @param word a word that is found
     */
    public static void printWord(CountryWithArrayList country, String word) {
        Census[] result = CountryProcessorWithStreams.findWord(country, word);
        if (result.length == 0) {
            System.out.println(
                  "The word \"" + word + "\" is not present in the comments.");
        }
        else {
            System.out.println(
                  "The word \"" + word + "\" is present in the comments:");
            for (Census census : result) {
                System.out.println(StringRepresentations.toString(census));
            }
        }
    }

    /**
     * Performs testing search methods
     * @param country reference to a country
     */
    public static void testSearch(CountryWithArrayList country) {
        System.out.println("Population density in 1979: " +
                            CountryProcessorWithStreams.density(country, 1979));
        System.out.println("The year with the maximum population: " +
                           CountryProcessorWithStreams.maxYear(country) + "\n");
        printWord(country, "census");
        printWord(country, "second");
    }

    /**
     * Performs testing search methods
```

```
     * @param country reference to a country
     */
    public static void testSorting(CountryWithArrayList country) {
        CountryProcessorWithStreams.sortByPopulation(country);
        System.out.println("\nSorting by population:");
        System.out.println(StringRepresentations.toString(country));

        CountryProcessorWithStreams.sortByComments(country);
        System.out.println("\nSorting comments alphabetically:");
        System.out.println(StringRepresentations.toString(country));
    }

    /**
     * Demonstration of the program.
     * @param args command line arguments (not used)
     */
    public static void main(String[] args) {
        CountryWithArrayList country =
             (CountryWithArrayList) setCountryData(new CountryWithArrayList());
        testSearch(country);
        testSorting(country);
    }
}
```

.

4 Quiz

1) Name the main components of the Java SE platform.

2) What are classes `BigInteger` and `BigDecimal`?

3) How can you create a type number `BigInteger`?

4) What is the difference between internal representation of double and BigDecimal?

5) Is it possible to apply mathematical operations to numbers of types `BigInteger` and `BigDecimal`?

6) What is the `MathContext` class used for?

7) What is JCF? What standard interfaces does JCF provide?

8) How to create a read-only collection?

9) What are the `Queue` interface methods used to add items?

10) Why are the methods for working with the queue implemented in two versions: with exception throwing and without exception throwing?

11) What is the use of the `PriorityQueue` class?

12) What are the stacks used for?

13) What are the standard ways to implement the stack?

14) What are the advantages and features of Stream API?

15) How to get a stream from a collection?

16) How to get a stream from an array?

17) What is the difference between intermediate and terminal operations?

18) What are the streams for working with primitive types?

19) What are the standard means of checking assertions in Java?

20) What is unit testing?

21) What is JUnit?

22) How are test methods annotated in JUnit?

23) How to make a logical grouping of tests?

24) How to use JUnit in a development environment?

LABORATORY TRAINING 2
Working with dates and text. Localization

1 Training tasks

1.1 Individual task

Design and implement classes to represent the entities of the third laboratory training of the Java Basic Programming Course. The solution should be based on the previously created class hierarchy.

The program should implement the support of various localizations, in particular, Ukrainian and English (American). It is necessary to provide for the translation of text, the output of numbers, as well as dates and times, taking into account different localizations. Add (modify) search for words in comments (or other text) using regular expressions. Sort the entities alphabetically using the Collator class.

Create a derived class from the class that represents the second entity. Add the time and date when a certain event occurs as a field. To represent the time and date, use classes of `java.time` package (taking into account the time zone). If the class that represented the second entity of the individual task did not contain a String type field, you should add a field, e.g. a comment to the event. For a new (or existing) text field, provide for the possibility of output in Ukrainian or English, depending on localization.

The program must demonstrate:

presentation of the implementation of tasks of laboratory trainings # 3 and # 4 of the Java Basic Programming Course; use regular expressions to search for text;

– formatting numerical data in various ways, as well as taking into account localization;

– output of data about dates and times of events taking into account localization;

– output of comments in Ukrainian and English;

– the ability to sort alphabetically using a `Collator` class;

– calculation and output of time intervals between events related to the second essence of the task; finding and output the smallest of the intervals.

1.2 Date entering

Implement a program in which the user enters a string. The program checks whether the string corresponds to the date representation accepted in Ukraine. The regular expressions must be used for checking. If the string does not meet the requirements, an error message is displayed. Otherwise, objects of types `Date`, `GregorianCalendar` and `LocalDate` are created and output to the console.

1.3 Phone number verification

Develop a program to verify the correctness of the fact that the string is the telephone number of the Kyivstar operator. Use regular expressions.

1.4 Obtaining an array of substrings (advanced task)

A string longer than 20 characters contains letters and numbers. Get from this string an array of substrings that contain letters between numbers (groups of numbers). Define numbers as separators.

2 Guidelines

The goal of this laboratory training is obtaining practical skills in working with dates and text in Java.

The following questions should be explored:

– working with dates and times;

– working with text in Java;

– localization;

– using the `java.text` package;

– regular expressions.

Necessary theoretical concepts and syntax constructs are discussed in lectures on the topics “Working with Dates and Times” and “Working with Text in Java”.

Most tasks can be implemented based on sample programs discussed below.

3 Sample programs

3.1 View available localizations

The `getAvailableLocales()` method allows us to view the available localizations. The `toString()` method returns a short representation of the localization. With the help of the function `getDisplayName()` we can get more detailed information:

```
package ua.inf.iwanoff.java.advanced.second;

import java.util.Locale;
import java.text.NumberFormat;

public class AvailableLocales {

    public static void main(String[] args) {
        Locale list[] = NumberFormat.getAvailableLocales();
        for (Locale locale : list) {
            System.out.printf("%-25s %s%n",
                              locale.toString(), locale.getDisplayName());
        }
    }
}
```

3.2 The First Substring that Matches the Pattern

The following program finds the first substring that matches the pattern.

```
package ua.inf.iwanoff.java.advanced.second;

import java.util.Scanner;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexTest {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.printf("%nEnter the regular expression: ");
        String ex = scan.nextLine();
        Pattern pattern = Pattern.compile(ex);
        System.out.printf("Enter string to search: ");
        Matcher matcher = pattern.matcher(scan.nextLine());
        if (matcher.find()) {
            System.out.printf("Text found " + "\"%s\", starting at position "
                    + "%d and ending with position %d.%n", matcher.group(),
                    matcher.start(), matcher.end());
        }
    }
}
```

3.3 Checking the phone number format

Suppose we need to write a program that checks the entered string for compliance with the format of a landline phone number of the Kharkiv telephone

network. Assume that numbers should be written as groups of numbers separated by hyphens. A phone number without a country or city code can be seven or six digits long. Only the last six or seven digits are mandatory for input. Note that the number may contain the country code (three digits), with a "+" sign at the beginning and the city code (57- for seven-digit numbers or 572- for six-digit numbers).

```
package ua.inf.iwanoff.java.advanced.second;

import java.util.Scanner;
import java.util.regex.Pattern;
import java.util.regex.Matcher;

public class RegexTest {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.printf("%nEnter a Phone Number: ");
        String phone = scan.nextLine();
        Pattern p = Pattern.compile(
                    "\\+?(380-)?((57-)?\\d{3}|(572-)?\\d{2})-\\d{2}-\\d{2}$");
        Matcher m = p.matcher(phone);
        if (m.matches()) {
            System.out.println("\"" + phone + "\" - correct number format");
        }
        else {
            System.out.println("\"" + phone + "\" - wrong number format");
        }
    }
}
```

3.4 Output words of sentence

Suppose it is necessary to enter a sentence from the keyboard and output its words in separate lines. We will use the `split()` method of the String class:

```
package ua.inf.iwanoff.java.advanced.second;

import java.util.Scanner;

public class UsingSplit {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String sentence = scan.nextLine();
        String[] words = sentence.split("\\s");
        for (String word : words) {
            System.out.println(word);
        }
    }
}
```

A regular expression is specified in the `split()` method call, which defines the delimiter between words.

3.5 "Country" and "Census" Classes

In the examples from the Java Basic Programming Course, class hierarchies to represent country and population censuses were considered. In laboratory training # 3 of this course, an abstract class `AbstractCountry` was created, as well as concrete classes `Census` and `CountryWithArray`. Next, in laboratory training # 4, classes `CountryWithList` and `CountryWithSet` were created.

Now, using the classes `CountryWithList` and `Census`, we will create an application that

– reproduces the sorting defined in work example laboratory training # 3 of the Java Basic Programming Course;

– performs formatting of numerical data in various ways, as well as taking into account localization;

– outputs data about dates and times of events taking into account localization;

– outputs text in English and Ukrainian;

– implements the ability to sort alphabetically using the `Collator` class;

– provides verification of the occurrence of words in the comment using regular expressions.

Before starting the implementation of the program, it is advisable to determine the bundle of resources with all necessary strings. File `censuses_en.properties`:

```
Ukraine=Ukraine
area=Area
unit=sq.km.
densityIn1979=Population density in 1979
census=Census
year=Year
population=Population
comments=Comments
commentsDate=Date and time of adding comments
firstPostwarCensus=The first census after World War II
populationIncreases=Population increases
noComments=No comments
lastSovietCensus=The last soviet census
```

```
firstCensusInIndependentUkraine=The first census in the independent Ukraine
yearWithMaximumPopulation=The year with the maximum population
sortingByPopulation=Sorting by population
sortingByComments=Sorting by comments
word=The word
isNotPresentInTheComments=is not present in the comments
isPresentInTheComments=is present in the comments
```

The code of the `censuses_uk.properties` file typed in the IntelliJ IDEA environment with the `Transparent native-to-ascii conversion` option enabled:

```
Ukraine=Україна
area=Територія
unit=кв. км.
densityIn1979=Густота населення у 1979 році
census=Перепис
year=Рік
population=Населення
comments=Коментар
commentsDate=Дата і час додавання коментаря
firstPostwarCensus=Перший перепис після другої світової війни
populationIncreases=Нас побільшало
noComments=Просто перепис
lastSovietCensus=Останній радянський перепис
firstCensusInIndependentUkraine=Перший перепис у незалежній Україні
yearWithMaximumPopulation=Рік з найбільшим населенням
sortingByPopulation=Сортування за кількістю населення
sortingByComments=Сортування за алфавітом коментарів
word=Слово
isNotPresentInTheComments=не міститься в коментарях
isPresentInTheComments=міститься в коментарях
```

The actual content of the `censuses_uk.properties` file will be different and will contain the codes of Ukrainian letters.

In the new `CensusWithDates` class, we add a field `timeOfEditing` of type `ZonedDateTime` for storing the date and time of comment editing. The source code of the `CensusWithDates` class will be as follows:

```
package ua.inf.iwanoff.java.advanced.second;

import ua.inf.iwanoff.java.third.Census;
import java.time.ZonedDateTime;

/**
 * The class is responsible for presenting the census.
 * Added comment editing time
 */
public class CensusWithDates extends Census {
    private ZonedDateTime timeOfEditing;

    /**
     * Constructor initializes the object with default values
     */
```

```
    public CensusWithDates() {
    }

    /**
     * Constructor initializes the object with given values
     *
     * @param year          year of census
     * @param population    population in the specified year
     * @param comments      the text of the comment
     * @param timeOfEditing time of comment editing
     */
    public CensusWithDates(int year, int population,
                           String comments, ZonedDateTime timeOfEditing) {
        super(year, population, comments);
        this.timeOfEditing = timeOfEditing;
    }

    /**
     * Returns time of comment editing
     *
     * @return time of comment editing of a ZonedDateTime object
     */
    public ZonedDateTime getTimeOfEditing() {
        return timeOfEditing;
    }

    /**
     * Sets the time of comment editing
     *
     * @param timeOfEditing the time of comment editing
     */
    public void setTimeOfEditing(ZonedDateTime timeOfEditing) {
        this.timeOfEditing = timeOfEditing;
    }
}
```

In the `StringsWithLocalization` class we define methods for obtaining string representation of census and country data, taking into account localization:

```
package ua.inf.iwanoff.java.advanced.second;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;

import java.time.format.DateTimeFormatter;
import java.time.format.FormatStyle;
import java.util.Locale;
import java.util.ResourceBundle;

/**
 * A class that allows getting representation
 * of various application objects in the form of strings
 */
public class StringsWithLocalization {

    /**
     * Provides a census data in the form of a string
     *
     * @param census reference to a census
     * @return string representation of a census data
     */
```

```
    public static String toString(CensusWithDates census) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        DateTimeFormatter dateFormatter =
                            DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL)
                            .withLocale(Locale.getDefault());
        return String.format(Locale.getDefault(),
                       bundle.getString("census") + ". "
                       + bundle.getString("year") + ": %d, "
                       + bundle.getString("population") + ": %d, "
                       + bundle.getString("comments") + ": %s, "
                       + bundle.getString("commentsDate") + ": %s ",
                               census.getYear(), census.getPopulation(),
                               bundle.getString(census.getComments()),
                               census.getTimeOfEditing().format(dateFormatter));
    }

    /**
     * Returns a string representation of the country
     *
     * @param country reference to a country
     * @return a string representation of the country
     */
    public static String toString(CountryWithArrayList country) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        StringBuilder result = new
StringBuilder(String.format(Locale.getDefault(),
                "%s. %s:%10.1f %s",
                bundle.getString(country.getName()),
                bundle.getString("area"),
                country.getArea(),
                bundle.getString("unit")));
        country.getList().forEach(census -> result.append("\n")
                .append(toString((CensusWithDates)census)));
        return result.toString();
    }

}
```

In the `CountryProcessorWithLocalization` class we implement methods for searching for words in comments and sorting them alphabetically:

```
package ua.inf.iwanoff.java.advanced.second;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.Census;

import java.text.Collator;
import java.util.*;
import java.util.regex.Pattern;

/**
 * Class for processing data about the country and censuses
 * taking into account localization and working with regular expressions
 */
public class CountryProcessorWithLocalization {
    /**
     * Checks whether the word can be found in the comment text
     * taking into account localization.
     * The use of regular expression are used
     *
```

```
     * @param census reference to a census
     * @param word a word that should be found in a comment
     * @return {@code true}, if the word is contained in the comment text
     *         {@code false} otherwise
     */
    public static boolean containsWord(Census census, String word) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        // (?iu) means i (case-insensitive) + u (Unicode for Cyrillic)
        // Pattern.quote() escapes special characters in a word, if any
        // The Pattern.UNICODE_CHARACTER_CLASS parameter is required for
        // support Unicode characters in the expression template
        Pattern pattern = Pattern.compile("(?iu)\\b" + Pattern.quote(word) +
                                          "\\b", Pattern.UNICODE_CHARACTER_CLASS);
        return pattern.matcher(bundle.getString(census.getComments())).find();
    }

    /**
     * Creates and returns an array of censuses
     * with the specified word in the comments
     *
     * @param country reference to a country
     * @param word a word that should be found
     * @return array of censuses with the specified word in the comments
     */
    public static Census[] findWord(CountryWithArrayList country, String word)
{
        return country.getList().stream()
                .filter(c -> containsWord(c, word))
                .toArray(Census[]::new);
    }
    /**
     * Sorts the sequence of censuses in the alphabetic order of comment
     * taking into account localization
     *
     * @param reference to a country
     */
    public static void sortByComments(CountryWithArrayList country) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        List<CensusWithDates> arr = new ArrayList<>();
        country.getList().forEach(census -> arr.add((CensusWithDates) census));
        arr.sort(new Comparator<>() {
            Collator collator = Collator.getInstance(Locale.getDefault());

            @Override
            public int compare(CensusWithDates c1, CensusWithDates c2) {
                return collator.compare(bundle.getString(c1.getComments()),
                                        bundle.getString(c2.getComments()));
            }
        });
        country.setList(new ArrayList<>());
        country.getList().addAll(arr);
    }
}
```

The `CountryWithLocalizationDemo` class demonstrates the capabilities of the new solution:

```
package ua.inf.iwanoff.java.advanced.second;

import ua.inf.iwanoff.java.advanced.first.CountryProcessorWithStreams;
```

```
import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.Census;

import java.time.*;
import java.util.Locale;
import java.util.ResourceBundle;

import static
ua.inf.iwanoff.java.advanced.second.CountryProcessorWithLocalization.sortByComm
ents;

/**
 * Country class workability testing program
 * taking into account localization and working with time and dates
 */
public class CountryWithLocalizationDemo {
    /**
     * Displays census data that contain a specific word in the comments
     *
     * @param country reference to a country
     * @param word a word that should be found
     */
    public static void printWord(CountryWithArrayList country, String word) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        Census[] result = CountryProcessorWithLocalization.findWord(
                                                                country, word);
        if (result.length == 0) {
            System.out.println(String.format("%s \"%s\" %s.",
                    bundle.getString("word"), word,
                                bundle.getString("isNotPresentInTheComments")));
        }
        else {
            String s = bundle.getString("word") + " \"" + word + "\" " +
                                    bundle.getString("isPresentInTheComments");
            System.out.println(s);
            for (Census census : result) {
                System.out.println(StringsWithLocalization.toString(
                                                        (CensusWithDates) census));
            }
        }
    }

    /**
     * Tests search methods
     *
     * @param country reference to a country
     */
    public static void testSearch(CountryWithArrayList country) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        System.out.println(String.format(Locale.getDefault(), "%s:%5.1f",
                        bundle.getString("densityIn1979"),
                        CountryProcessorWithStreams.density(country, 1979)));
        System.out.println(String.format(Locale.getDefault(), "%s: %d",
                        bundle.getString("yearWithMaximumPopulation"),
                        CountryProcessorWithStreams.maxYear(country)));
        printWord(country, "census");
        printWord(country, "cens");
    }

    /**
     * Tests sorting methods
```

```
     *
     * @param country reference to a country
     */
    public static void testSorting(CountryWithArrayList country) {
        ResourceBundle bundle = ResourceBundle.getBundle("censuses");
        CountryProcessorWithStreams.sortByPopulation(country);
        System.out.println("\n" + bundle.getString("sortingByPopulation"));
        System.out.println(StringsWithLocalization.toString(country));
        sortByComments(country);
        System.out.println("\n" + bundle.getString("sortingByComments"));
        System.out.println(StringsWithLocalization.toString(country));
    }

    /**
     * Auxiliary function of creating a new country object
     *
     * @return reference to the new country object
     */
    public static CountryWithArrayList setCountryData() {
        CountryWithArrayList country = new CountryWithArrayList();
        country.setName("Ukraine");
        country.setArea(603628);
        country.addCensus(new CensusWithDates(1959,
                          41869000, "firstPostwarCensus",
                          ZonedDateTime.of(2025, Month.MARCH.getValue(),
                          1, 10, 0, 0, 0, ZoneId.of("Europe/Kiev"))));
        country.addCensus(new CensusWithDates(1970,
                          47126500, "populationIncreases",
                          ZonedDateTime.of(2025, Month.MARCH.getValue(),
                          3, 10, 30, 0, 0, ZoneId.of("Europe/Kiev"))));
        country.addCensus(new CensusWithDates(1979,
                          49754600, "noComments",
                          ZonedDateTime.of(2025, Month.MARCH.getValue(),
                          1, 11, 30, 0, 0, ZoneId.of("Europe/Kiev"))));
        country.addCensus(new CensusWithDates(1989,
                          51706700, "lastSovietCensus",
                          ZonedDateTime.of(2025, Month.MARCH.getValue(),
                          2, 14, 0, 0, 0, ZoneId.of("Europe/Kiev"))));
        country.addCensus(new CensusWithDates(2001,
                          48475100, "firstCensusInIndependentUkraine",
                          ZonedDateTime.of(2025, Month.MARCH.getValue(),
                          4, 10, 0, 0, 0, ZoneId.of("Europe/Kiev"))));
        return country;
    }

    /**
     * Demonstration of functions' work
     *
     * @param args command line arguments (not used)
     */
    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        System.out.println(Locale.getDefault());
        CountryWithArrayList country = setCountryData();
        testSearch(country);
        testSorting(country);
        System.out.println("-------------------------------------");
        Locale.setDefault(Locale.forLanguageTag("uk"));
        System.out.println(Locale.getDefault());
        country = setCountryData();
        testSearch(country);
```

```
        testSorting(country);
    }
}
```

As expected, the word "census" is in the comments only if we are working with the English localization.

4 Quiz

1) What is the difference between classes `Calendar` and `Date`?

2) How to create an instance of `GregorianCalendar`?

3) What tools for working with the calendar does Java 8 provide?

4) What facilities for working with time zones does Java 8 provide?

5) What is the difference between internationalization and localization?

6) How is a class `ResourceBundle` used to internationalize and localize applications?

7) What is the general purpose of the java.text package?

8) What are the advantages of sorting by `Collator` class?

9) What is data formatting?

10) How is localization-sensitive data formatting done?

11) What is a regular expression?

12) How to work with regular expressions in Java?

13) What are the methods of dividing the text into tokens?

14) What is the advantage of using the method `split()` compared to means of `StringTokenizer`?

LABORATORY TRAINING 3

Advanced features of working with files

1 Training tasks

1.1 Individual assignment

Create a new Maven project and transfer to this project the previously created classes that represent entities of the individual tasks of laboratory trainings # 3 and # 4 of the Java Basic Programming Course.

The program must demonstrate:

– reproduction of the implementation of individual tasks of laboratory training # 3 and # 4 of the Java Basic Programming Course;

– outputting data to a text file using Stream API with subsequent reading;

– serialization of objects into an XML file and a JSON file and corresponding deserialization using the XStream library;

– recording events related to program execution in the system log;

– testing individual classes using JUnit.

Note: localization and translation of texts can be carried out at the wish of the student.

1.2 List of files of all subdirectories

Enter the name of a specific folder. Display the names of all files in this directory, as well as all files in subdirectories, their subdirectories, and so on. Implement two approaches:

– search by means of a `java.io.File` class using a recursive function;

– search by means of the `java.nio.file` package.

Display both results sequentially on the screen. If the folder does not exist, display an error message.

1.3 Working with text files using the Stream API

Use the `Files.lines()` function to read strings from a text file, sort by increasing length, and output strings that contain the letter "a" to another file.

1.4 Implementation of an array of points through a two-dimensional array

Create the classes "Student" and "Academic group" (with array of students as the field). Create objects. Provide file creation and reading from files using the following approaches:

– using Stream API tools for working with text files;

– serialization and deserialization in XML and JSON (by means of XStream).

1.5 Use of SAX and DOM technologies (additional task)

Complete task 1.4 using tools for working with JSON files of the org.json library.

1.6 The Implementation of the comparator interface

Prepare an XML document with data about the students of the academic group. Using SAX technology read data from an XML document and display data on the console. Using DOM technology, read data from the same XML document, modify the data, and write it to a new document.

2 Guidelines

The goal of this laboratory training is obtaining practical skills in using advanced features of working with files in Java.

The following questions should be explored:

– Java tools for working with files;

– use of build automation tools;

– working with XML documents;

– working with the file system;

– using java.nio tools for reading and writing data;

– using the java 8 Stream API;

– working with JSON files;

– serialization into XML and JSON files using XStream tools;

– logging.

Necessary theoretical concepts and syntax constructs are discussed in lectures on the topic “Working with files of special format”.

Most tasks can be implemented based on sample programs discussed below.

3 Sample programs

3.1 Using DOM technology

Suppose an XML document with about the continent data (`Continent.xml`) is prepared:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContinentData Name="Europe">
    <CountriesData>
        <CountryData Name="Ukraine" Area="603700" Population="46314736" >
            <CapitalData Name="Kiev" />
        </CountryData>
        <CountryData Name="France" Area="547030" Population="61875822" >
            <CapitalData Name="Moscow" />
        </CountryData>
        <CountryData Name="Germany" Area="357022" Population="82310000" >
            <CapitalData Name="Berlin" />
        </CountryData>
    </CountriesData>
</ContinentData>
```

Note: error with the capital of France is intentional.

It is necessary use DOM tools to read the data, fix the error and save it in a new file. The program will look like this:

```
package ua.inf.iwanoff.java.advanced.third;

import java.io.*;
import org.w3c.dom.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class ContinentWithDOM {

    public static void main(String[] args) {
        try {
            Document doc;
            DocumentBuilder db =
                     DocumentBuilderFactory.newInstance().newDocumentBuilder();
            doc = db.parse(new File("Continent.xml"));
            Node rootNode = doc.getDocumentElement();
          mainLoop:
            for (int i = 0; i < rootNode.getChildNodes().getLength(); i++) {
                Node countriesNode = rootNode.getChildNodes().item(i);
                if (countriesNode.getNodeName().equals("CountriesData")) {
                    for (int j = 0; j <
```

```
                            countriesNode.getChildNodes().getLength(); j++) {
                        Node countryNode =
                                        countriesNode.getChildNodes().item(j);
                        if (countryNode.getNodeName().equals("CountryData")) {
                            // Find the attribute by name:
                            if (countryNode.getAttributes()
                                    .getNamedItem("Name")
                                    .getNodeValue().equals("France")) {
                                for (int k = 0; k < countryNode
                                        .getChildNodes().getLength(); k++) {
                                    Node capitalNode =
                                        countryNode.getChildNodes().item(k);
                                    if (capitalNode.getNodeName()
                                                        .equals("CapitalData")) {
                                        capitalNode.getAttributes()
                                                .getNamedItem("Name")
                                                .setNodeValue("Paris");
                                        break mainLoop;
                                    }
                                }
                            }
                        }
                    }
                }
            }
            Transformer transformer =
                            TransformerFactory.newInstance().newTransformer();
            transformer.transform(new DOMSource(doc),
                new StreamResult(
                    new FileOutputStream(new File("CorrectedContinent.xml"))));
        }
        catch (Exception  e) {
            e.printStackTrace();
        }
    }
}
```

3.2 Reading from a file and sorting strings

Suppose we need to read strings from a text file, sort them in reverse alphabetical order, and write strings that start with the letter "F" to a new file.

A file with strings (`strings.txt`) can be like this:

```
First
Second
Third
Fourth
Fifth
```

The program can be as follows:

```
package uua.inf.iwanoff.java.advanced.third;

import java.io.BufferedReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Stream;
```

```
public class ReadAndSort {
    public static void main(String[] args) throws IOException {
        try (BufferedReader reader =
                        Files.newBufferedReader(Paths.get("strings.txt"))) {
            Stream<String> stream = reader.lines()
                    .sorted((s1, s2) -> s2.compareTo(s1))
                    .filter(s -> s.startsWith("F"));
            Files.write(Paths.get("results.txt"), stream.toList());
        }
    }
}
```

After running of the program, we will get the file called `results.txt`:

```
Fourth
First
Fifth
```

3.3 Serialization and deserialization using the XStream library

Suppose you need to serialize and deserialize data about a line described by two points. We are creating a new Maven project called `LineAndPoints`. We add a dependency on the xstream library to the `pom.xml` file. We will get the following `pom.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                             http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ua.inf.iwanoff.java.advanced.third</groupId>
    <artifactId>LineAndPoints</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>25</maven.compiler.source>
        <maven.compiler.target>25</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.thoughtworks.xstream</groupId>
            <artifactId>xstream</artifactId>
            <version>1.4.21</version>
        </dependency>
    </dependencies>

</project>
```

We create `Line` and `Point` classes. These classes have no parameterless constructors and public properties, so they cannot be serialized using `java.beans.XMLEncoder` and `.beans.XMLDecoder`. But XStream allows you to serialize them because this library focuses on fields, not properties.

The `Point` class:

```
package ua.inf.iwanoff.java.advanced.third;

public class Point {
    private double x, y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return x + " " + y;
    }
}
```

The Line class:

```
package ua.inf.iwanoff.java.advanced.third;

public class Line {
    private Point first, second;

    public Line(double firstX, double firstY, double secondX, double secondY) {
        first = new Point(firstX, firstY);
        second = new Point(secondX, secondY);
    }

    @Override
    public String toString() {
        return first + " " + second;
    }
}
```

The following class can be created to serialize data:

```
package ua.inf.iwanoff.java.advanced.third;

import com.thoughtworks.xstream.XStream;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class XMLSerialization {
    public static void main(String[] args) {
        XStream xStream = new XStream();
        Line line = new Line(1, 2, 3, 4);
        xStream.alias("line", Line.class);
        String xml = xStream.toXML(line);
        try (FileWriter fw = new FileWriter("Line.xml");
             PrintWriter out = new PrintWriter(fw)) {
            out.println(xml);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
}
```

Note: In newer versions of the JDK, running a program may result in a message `WARNING: A terminally deprecated method in sun.misc.Unsafe has been called`, followed by warnings detailing the problem. These warnings are caused by the direct use of reflection to create objects and access data. These warnings can be ignored, as they do not affect the performance of the program. To completely get rid of these warnings, you need to add the Java virtual machine parameters `--sun-misc-unsafe-memory-access=allow` to the runtime configuration settings (`Run | Edit Configurations...`). To make the necessary line for entering virtual machine parameters appear in the application configuration window (`Application`), press `Alt-V`.

After executing the program, we get an XML file:

```
<line>
  <first>
    <x>1.0</x>
    <y>2.0</y>
  </first>
  <second>
    <x>3.0</x>
    <y>4.0</y>
  </second>
</line>
```

Note: if no alias is used, the root tag will be: `<ua.inf.iwanoff.java.advanced.third.Line>`

We deserialize objects in another program:

```
package ua.inf.iwanoff.java.advanced.third;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.security.AnyTypePermission;

import java.io.File;

public class XMLDeserialization {
    public static void main(String[] args) {
        XStream xStream = new XStream();
        xStream.addPermission(AnyTypePermission.ANY);
        xStream.alias("line", Line.class);
        Line newLine = (Line) xStream.fromXML(new File("Line.xml"));
        System.out.println(newLine);
    }
}
```

In order to use XStream tools for working with JSON files, we need to add one more dependency to the pom.xml file:

```
        <dependency>
            <groupId>org.codehaus.jettison</groupId>
            <artifactId>jettison</artifactId>
            <version>1.5.4</version>
        </dependency>
```

The program of serializing into a JSON file will be as follows:

```
package ua.inf.iwanoff.java.advanced.third;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JsonHierarchicalStreamDriver;

import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class JSONSerialization {
    public static void main(String[] args) {
        XStream xStream = new XStream(new JsonHierarchicalStreamDriver());
        Line line = new Line(1, 2, 3, 4);
        xStream.alias("line", Line.class);
        String xml = xStream.toXML(line);
        try (FileWriter fw = new FileWriter("Line.json");
             PrintWriter out = new PrintWriter(fw)) {
            out.println(xml);
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

The following JSON file will be obtained:

```
{"line": {
  "first": {
    "x": 1.0,
    "y": 2.0
  },
  "second": {
    "x": 3.0,
    "y": 4.0
  }
}}
```

The program to deserialize from a JSON file would be:

```
package ua.inf.iwanoff.java.advanced.third;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;
import com.thoughtworks.xstream.security.AnyTypePermission;

import java.io.File;

public class JSONDeserialization {
```

```
    public static void main(String[] args) {
        XStream xStream = new XStream(new JettisonMappedXmlDriver());
        xStream.addPermission(AnyTypePermission.ANY);
        xStream.alias("line", Line.class);
        Line newLine = (Line) xStream.fromXML(new File("Line.json"));
        System.out.println(newLine);
    }
}
```

3.4 "Country" and "Census" classes

Suppose we want to reimplement a previously created project related to the country and censuses. The base classes that implement the basic functionality and data structures for representing countries and censuses were presented in examples of laboratory trainings # 3 and # 4 of the Java Basic Programming Course. It is necessary to add derived classes in which override the implementation of all methods related to the processing of sequences through the use of Stream API tools. In addition to reproducing the implementation of the functionality, the new project should include:

– outputting data to a text file using Stream API with subsequent reading;

– serialization of objects into an XML file and a JSON file and corresponding deserialization using the XStream library;

– recording events related to program execution in the system log;

– testing individual classes using JUnit.

Taking into account the addition of dependencies on external libraries, it is advisable to create a new Maven project into which to transfer previously created classes. You can copy files from one project to another via the clipboard: in the `Projects` sub-window, select the necessary files and copy them to the clipboard (`Copy` function of the context menu); in another project, select the required package and insert the files using the `Paste` function. You can also copy the entire package.

The `FileUtils` class will be responsible for storing in a text file, reading from a text file, serializing and deserializing data (from XML and JSON), as well as writing to the event log in the program. Logging will be done every time we

read or write data to files of various formats. To work with XStream and log4j tools, you need to add the following dependencies to the `pom.xml` file:

```
        <dependency>
            <groupId>com.thoughtworks.xstream</groupId>
            <artifactId>xstream</artifactId>
            <version>1.4.21</version>
        </dependency>
        <dependency>
            <groupId>org.codehaus.jettison</groupId>
            <artifactId>jettison</artifactId>
            <version>1.5.4</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.25.1</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.25.1</version>
        </dependency>
```

We also need to configure the log4j properties by adding a file `log4j2.xml` to the `java\src\resources` folder. In our case, this file will be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="INFO">
    <Appenders>
        <Console name="ConsoleAppender" target="SYSTEM_OUT"/>
        <File name="FileAppender" fileName="country-${date:yyyyMMdd}.log"
                    immediateFlush="false" append="true"/>
    </Appenders>
    <Loggers>
        <Root level="debug">
            <AppenderRef ref="FileAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

Individual actions related to reading and writing can be implemented as static methods. The code of `FileUtils` class will be as follows:

```
package ua.inf.iwanoff.java.advanced.third;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;
import com.thoughtworks.xstream.security.AnyTypePermission;
import org.apache.logging.log4j.Logger;
import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.Census;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * The class implements writing and reading data in TXT, XML and JSON formats.
 * Country and census data are read and written.
 * At the same time, events are recorded in the system log.
 */
public class FileUtils {
    private static Logger logger = null;

    public static Logger getLogger() {
        return logger;
    }

    public static void setLogger(Logger logger) {
        FileUtils.logger = logger;
    }

    /**
     * Creates and returns a list of stings with data
     * about the country and about all population censuses
     *
     * @param country the country
     * @return a list of strings with country data
     */
    public static List<String> toListOfStrings(CountryWithArrayList country) {
        ArrayList<String> list = new ArrayList<>();
        list.add(country.getName() + " " + country.getArea());
        Arrays.stream(country.getCensuses()).forEach(c -> list.add(
            c.getYear() + " " + c.getPopulation() + " " + c.getComments()));
        return list;
    }

    /**
     * Reads data about the country from the list of strings and puts data
     * into the appropriate fields
     *
     * @param country the country
     * @param list a list of strings with country data
     */
    public static void fromListOfStrings(CountryWithArrayList country,
                                         List<String> list) {
        String[] words = list.get(0).split("\s");
        country.setName(words[0]);
        country.setArea(Double.parseDouble(words[1]));
        country.setList(new ArrayList<>());
        list.remove(0);
        list.stream().forEach(s -> { String[] line = s.split("\s");
            country.addCensus(Integer.parseInt(line[0]),
                              Integer.parseInt(line[1]),
                    s.substring(s.indexOf(line[2]))); });
    }

    /**
     * Writes country and census data to the specified file
     *
     * @param country the country
```

```
     * @param fileName the name of the file
     */
    public static void writeToTxt(CountryWithArrayList country,
                                  String fileName) {
        if (logger != null) {
            logger.info("Write to text file");
        }
        try {
            Files.write(Path.of(fileName), toListOfStrings(country));
        }
        catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }
            throw new RuntimeException(e);
        }
    }

    /**
     * Reads country and census data from the specified file
     *
     * @param fileName the name of the file
     * @return the object that was created
     */
    public static CountryWithArrayList readFromTxt(String fileName) {
        CountryWithArrayList country = new CountryWithArrayList();
        if (logger != null) {
            logger.info("Read from text file");
        }
        try {
            List<String> list = Files.readAllLines(Path.of(fileName));
            fromListOfStrings(country, list);
        }
        catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }
            throw new RuntimeException(e);
        }
        return country;
    }

    /**
     * Serializes country and census data into the specified XML file
     *
     * @param country the country
     * @param fileName the name of the file
     */
    public static void serializeToXML(CountryWithArrayList country,
                                      String fileName) {
        if (logger != null) {
            logger.info("Serializing to XML");
        }
        XStream xStream = new XStream();
        xStream.alias("country", CountryWithArrayList.class);
        xStream.alias("census", Census.class);
        String xml = xStream.toXML(country);
        try (FileWriter fw = new FileWriter(fileName);
             PrintWriter out = new PrintWriter(fw)) {
            out.println(xml);
        }
```

```
        catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }
            throw new RuntimeException(e);
        }
    }

    /**
     * Deserializes country and census data from the specified XML file
     *
     * @param fileName the name of the file
     * @return the object that was created
     */
    public static CountryWithArrayList deserializeFromXML(String fileName) {
        if (logger != null) {
            logger.info("Deserializing from XML");
        }
        try {
            XStream xStream = new XStream();
            xStream.addPermission(AnyTypePermission.ANY);
            xStream.alias("country", CountryWithArrayList.class);
            xStream.alias("census", Census.class);
            CountryWithArrayList country
                  = (CountryWithArrayList) xStream.fromXML(new File(fileName));
            return country;
        }
        catch (Exception e) {
            if (logger != null) {
                logger.error(e.toString());
            }
            throw new RuntimeException(e);
        }
    }

    /**
     * Serializes country and census data into the specified JSON file
     *
     * @param country the country
     * @param fileName the name of the file
     */
    public static void serializeToJSON(CountryWithArrayList country,
                                       String fileName) {
        if (logger != null) {
            logger.info("Serializing to JSON");
        }
        XStream xStream = new XStream(new JettisonMappedXmlDriver());
        xStream.alias("country", CountryWithArrayList.class);
        xStream.alias("census", Census.class);
        String xml = xStream.toXML(country);
        try (FileWriter fw =
             new FileWriter(fileName); PrintWriter out = new PrintWriter(fw)) {
            out.println(xml);
        }
        catch (IOException e) {
            if (logger != null) {
                logger.error(e.toString());
            }
            throw new RuntimeException(e);
        }
    }
```

```
    /**
     * Deserializes country and census data from the specified JSON file
     *
     * @param fileName the name of the file
     * @return the object that was created
     */
    public static CountryWithArrayList deserializeFromJSON(String fileName) {
        if (logger != null) {
            logger.info("Deserializing from JSON");
        }
        try {
            XStream xStream = new XStream(new JettisonMappedXmlDriver());
            xStream.addPermission(AnyTypePermission.ANY);
            xStream.alias("country", CountryWithArrayList.class);
            xStream.alias("census", Census.class);
            CountryWithArrayList country =
                    (CountryWithArrayList) xStream.fromXML(new File(fileName));
            return country;
        }
        catch (Exception e) {
            if (logger != null) {
                logger.error(e.toString());
            }
            throw new RuntimeException(e);
        }
    }
}
```

The code of `Program` class, in which all the created functions are demonstrated, will be as follows:

```
package ua.inf.iwanoff.java.advanced.third;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.third.StringRepresentations;

import static ua.inf.iwanoff.java.advanced.third.FileUtils .*;
import static ua.inf.iwanoff.java.third.CountryDemo.setCountryData;

/**
 * The class demonstrates writing and reading data in TXT, XML and JSON
formats.
 * At the same time, events are recorded in the system log.
 */
public class Program {

    /**
     * Demonstration of the program.
     * Data in TXT, XML and JSON formats are sequentially written and read.
     * At the same time, events are recorded in the system log
     *
     * @param args command line arguments (not used)
     */
    public static void main(String[] args) {
        Logger logger = LogManager.getLogger(Program.class);
        FileUtils.setLogger(logger);
```

```
        logger.info("Program started");
        CountryWithArrayList country =
             (CountryWithArrayList) setCountryData(new CountryWithArrayList());
        writeToTxt(country, "Country.txt");
        country = readFromTxt("Country.txt");
        System.out.println(StringRepresentations.toString(country));
        serializeToXML(country, "Country.xml");
        country = deserializeFromXML("Country.xml");
        System.out.println(StringRepresentations.toString(country));
        serializeToJSON(country, "Country.json");
        country = deserializeFromJSON("Country.json");
        System.out.println(StringRepresentations.toString(country));
        logger.info("Program finished");
    }
}
```

The result of the work will be output to the console of data about the country, read from various sources. New files will appear in the root directory of the project. Text file (`Country.txt`):

```
Ukraine 603628.0
1959 41869000 The first census after World War II
1970 47126500 Population increases
1979 49754600 No comments
1989 51706700 The last soviet census
2001 48475100 The first census in the independent Ukraine
```

The XML file (`Country.xml`):

```
<country>
  <name>Ukraine</name>
  <area>603628.0</area>
  <list>
    <census>
      <year>1959</year>
      <population>41869000</population>
      <comments>The first census after World War II</comments>
    </census>
    <census>
      <year>1970</year>
      <population>47126500</population>
      <comments>Population increases</comments>
    </census>
    <census>
      <year>1979</year>
      <population>49754600</population>
      <comments>No comments</comments>
    </census>
    <census>
      <year>1989</year>
      <population>51706700</population>
      <comments>The last soviet census</comments>
    </census>
    <census>
      <year>2001</year>
      <population>48475100</population>
      <comments>The first census in the independent Ukraine</comments>
    </census>
  </list>
```

```
</country>
```

Unfortunately, the JSON file (`Country.json`) generated by the program will be poorly formatted (the entire content of the file is in one line). But if you open this file in the IntelliJ IDEA environment and apply code formatting (`Code | Reformat Code`), its content in the editor window will be as follows:

```
{
  "country": {
    "name": "Ukraine",
    "area": 603628,
    "list": [
      {
        "census": [
          {
            "year": 1959,
            "population": 41869000,
            "comments": "First census after World War II"
          },
          {
            "year": 1970,
            "population": 47126500,
            "comments": "Population increases"
          },
          {
            "year": 1979,
            "population": 49754600,
            "comments": "No comments"
          },
          {
            "year": 1989,
            "population": 51706700,
            "comments": "The last soviet census"
          },
          {
            "year": 2001,
            "population": 48475100,
            "comments": "The first census in the independent Ukraine"
          }
        ]
      }
    ]
  }
}
```

In addition, a log file with the .log extension will be created, to which the following text fragment will be added after each program launch:

```
Program started
Write to text file
Read from text file
Serializing to XML
Deserializing from XML
Serializing to JSON
Deserializing from JSON
Program finished
```

Information about exceptions that occurred while working with files will be also logged.

4 Quiz

1) What are purposes of XML documents?

2) What restrictions are imposed on the structure of the XML document, the syntax and location of the tags?

3) What is the difference between SAX and DOM technologies?

4) How do I read and write XML documents?

5) What is XSLT?

6) What is the difference between a valid and well-formed XML document?

7) What are differences between document template definition and document schema?

8) Is document template definition (DTD) also XML document?

9) Which classes correspond to the Java Beans specification?

10) What are the disadvantages and advantages of XML serialization?

11) What tasks do build automation tools solve?

12) What is the main difference between Apache Maven and Apache Ant?

13) What is GAV?

14) What is the structure of `pom.xml` file?

15) What are the typical tasks of working with the file system?

16) What standard Java tools provide the ability to work with the file system? What are the differences between these means?

17) What are the ways to get information about files and subdirectories?

18) How is data read and written using the Stream API?

19) What is the JSON format, and what are its advantages?

20) What are the main elements of a JSON file?

21) What tools exist to support working with JSON files?

22) What XML serialization is performed by means of XStream?

23) What JSON-serialization is performed by means of XStream?

24) What is a logger and logging level?

25) What facilities exist to maintain logs?

26) What are the advantages of the Log4j library compared to standard logging tools?

27) What are output levels (priorities)?

LABORATORY TRAINING 4
Development of GUI applications

1 Training tasks

1.1 Individual task

It is necessary to implement in Java language with the help of JavaFX tools the graphical user interface application, in which the data processing of individual tasks of previous laboratory trainings is carried out. The main window should contain a menu in which you need to implement the following functions:

– creating a new data set;

– loading data from an XML document for editing;

– storing the changed data in an XML document;

– search according to the criteria specified in the Laboratory training # 2 of the Java Basic Programming Course;

– sorting according to the criteria specified in the Laboratory training # 2 of the Java Basic Programming Course;

– showing "About" window with short data about the program and the author.

In the left part of the window, you should arrange the boxes for entering the scalar data, the display area for the search results, and the buttons that provide the basic functions of the program. In the middle of the window, you should place a table to display and edit the data.

1.2 Using ObservableList

Create a list (`ObservableList`) of real numbers of type `Double`. Change the initial order of numbers so that positive numbers are placed first (without changing their relative order), and then negative numbers in the opposite order. Each change in the state of the list should result in the output of the list items to the console window.

1.3 Mini calculator

Create a graphical user interface application, which, after entering numbers in two boxes of `TextField` type, performs one of four arithmetic operations (depending on the radio button selected). The result should be displayed in a different text field.

1. Dictionary (advanced task)

Develop a GUI application for viewing the words of a small English-Ukrainian (English-Russian, etc.) dictionary. Implement search functions for words, adding new words. Use `Map` to store data.

2 Guidelines

The goal of this laboratory training is obtaining practical skills in Development of graphical user interface applications.

The following questions should be explored:

– using Java to create GUI applications;

– overview of JavaFX platform;

– the theoretical basis of creating JavaFX applications;

– working with JavaFX visual components;

– working with tabular data in JavaFX;

– visual design of GUI applications.

Necessary theoretical concepts and syntax constructs are discussed in lectures on the topic “Event driven programming in Java”.

Most tasks can be implemented based on sample programs discussed below.

3 Sample programs

3.1 Working with ObservableList

Suppose, in some list of integers, odd numbers should be arranged first, then even numbers, while maintaining the relative order of numbers. We can propose an algorithm in which the even number we found is removed from the list and added

to and at the end of the list. It is important to create two counters (`i` and `j`) so as not to transfer numbers several times.

The program will look like this:

```
package ua.inf.iwanoff.java.advanced.fourth;

import javafx.collections.FXCollections;
import javafx.collections.ListChangeListener;
import javafx.collections.ObservableList;

public class ObservableListDemo {
    static void orderList(ObservableList<Integer> list) {
        for (int i = 0, j = 0; j < list.size(); j++) {
            if (list.get(i) % 2 == 0) {
                list.add(list.get(i));
                list.remove(i);
            }
            else {
                i++;
            }
        }
    }

    public static void main(String[] args) {
        ObservableList<Integer> list = FXCollections.observableArrayList();
        list.addListener((
          ListChangeListener<? super Integer>) c -> System.out.println(list));
        list.addAll(1, 12, 2, 37, 6, 8, 11);
        orderList(list);
        list.clear();
    }
}
```

The text in the console window will look like this:

```
[1, 12, 2, 37, 6, 8, 11]
[1, 12, 2, 37, 6, 8, 11, 12]
[1, 2, 37, 6, 8, 11, 12]
[1, 2, 37, 6, 8, 11, 12, 2]
[1, 37, 6, 8, 11, 12, 2]
[1, 37, 6, 8, 11, 12, 2, 6]
[1, 37, 8, 11, 12, 2, 6]
[1, 37, 8, 11, 12, 2, 6, 8]
[1, 37, 11, 12, 2, 6, 8]
[]
```

3.2 Text fields and buttons

Assume that we want to create GUI application that allows user to enter two integer numbers in text fields and then obtains their sum in the third text field after clicking button.

The root container of our application will be `FlowPane`. So, it's enough to create three text boxes and one button and add them sequentially to the pane. The program will be as follows:

```
package ua.inf.iwanoff.java.advanced.fourth;

import javafx.application.Application;
import javafx.event.Event;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.stage.Stage;

public class TextFieldsAndButton extends Application {
    private Button button;
    private TextField field1, field2, field3;

    @Override
    public void start(Stage stage) throws Exception {
        stage.setTitle("Sum");
        // determine the size of the horizontal
        // and vertical gaps between the elements:
        FlowPane rootNode = new FlowPane(10, 10);
        rootNode.setAlignment(Pos.CENTER);
        Scene scene = new Scene(rootNode, 200, 200); // window size
        stage.setScene(scene);
        button = new Button("Find the sum");  // define the text on the button
        button.setOnAction(this::buttonClick);// define the event handler
        field1 = new TextField();
        field2 = new TextField();
        field3 = new TextField();
        rootNode.getChildren().addAll(field1, field2, button, field3);
        stage.show();
    }

    private void buttonClick(Event event) {
        try {
            int i = Integer.parseInt(field1.getText());
            int j = Integer.parseInt(field2.getText());
            int k = i + j;
            field3.setText(k + "");
        }
        catch (NumberFormatException e1) {
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Invalid data!");
            alert.showAndWait();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

3.3 Using radio buttons

In the example below, along with the selection of the `RadioButton` button, the text of the selected button is displayed in the Label. In order for the buttons to work consistently, they are combined into a `ToggleGroup` group:

```
package ua.inf.iwanoff.java.advanced.fourth;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.RadioButton;
import javafx.scene.control.ToggleGroup;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;


public class ToggleGroupDemo extends Application {
    private Label label = new Label("No button");

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Toggle Group Demo");

        RadioButton radioButtonFirst = new RadioButton("First");
        radioButtonFirst.setOnAction(this::showButtonText);
        RadioButton radioButtonSecond = new RadioButton("Second");
        radioButtonSecond.setOnAction(this::showButtonText);
        RadioButton radioButtonThird = new RadioButton("Third");
        radioButtonThird.setOnAction(this::showButtonText);

        ToggleGroup radioGroup = new ToggleGroup();
        radioButtonFirst.setToggleGroup(radioGroup);
        radioButtonSecond.setToggleGroup(radioGroup);
        radioButtonThird.setToggleGroup(radioGroup);
        VBox vbox = new VBox(radioButtonFirst,
                             radioButtonSecond, radioButtonThird, label);
        vbox.setSpacing(10);
        vbox.setPadding(new Insets(10, 10, 10, 10));
        Scene scene = new Scene(vbox, 150, 120);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private void showButtonText(ActionEvent actionEvent) {
        label.setText(((RadioButton)actionEvent.getSource()).getText());
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}
```

3.4 Working with the file dialogs

Suppose we need to read two numbers from a text file, and write down their sum in another text file. The window will contain two buttons: to select the input and output files respectively. We can offer such a program:

```
package ua.inf.iwanoff.java.advanced.fourth;

import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.Scanner;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.FlowPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public class SumWriter extends Application {
    private double a, b;
    private FileChooser chooser;
    private File file;

    @Override
    public void start(Stage stage) throws Exception {
        stage.setTitle("Sum");
        // determine the size of the horizontal
        // and vertical gaps between the elements:
        FlowPane rootNode = new FlowPane(10, 10);
        rootNode.setAlignment(Pos.CENTER);
        Scene scene = new Scene(rootNode, 200, 100); // window size
        stage.setScene(scene);
        // define the text on the button:
        Button buttonOpen = new Button("Load data");
        // define the function that handles an event:
        buttonOpen.setOnAction(this::buttonOpenClick);
        // define the text on the button:
        Button buttonSave = new Button("Save sum");
        // define the function that handles an event:
        buttonSave.setOnAction(this::buttonSaveClick);
        rootNode.getChildren().addAll(buttonOpen, buttonSave);
        chooser = new FileChooser();
        chooser.setInitialDirectory(new File("."));
        stage.show();
    }

    private void buttonOpenClick(ActionEvent event) {
        if ((file = chooser.showOpenDialog(null)) != null) {
            readFromFile();
        }
    }

    private void buttonSaveClick(ActionEvent event) {
```

```
        if ((file = chooser.showSaveDialog(null)) != null) {
            writeToFile();
        }
    }

    private void readFromFile() {
        try (Scanner scanner = new Scanner(file)) {
            a = scanner.nextDouble();
            b = scanner.nextDouble();
        }
        catch (Exception e) {
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Error reading from file!");
            alert.showAndWait();
        }
    }

    private void writeToFile() {
        try (PrintWriter out = new PrintWriter(new FileWriter(file))) {
            out.println(a + b);
            out.close();
        }
        catch (Exception e) {
            Alert alert = new Alert(AlertType.ERROR);
            alert.setTitle("Error");
            alert.setHeaderText("Cannot create a file!");
            alert.showAndWait();
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

3.5 GUI Application for processing data about censuses

Suppose we need to create a graphical user interface program that allows the user to enter and edit population census data, store data in an XML file, read data from previously created XML files, edit data, search for data according to the specific criteria, sort data, and storing data in a new file.

To the previously created project, we add a new package that will contain graphical user interface classes, including the main class controller, FXML documents, and additional controllers.

We create a new class named `CensusesFX`. Its content will be as follows:

```
package ua.inf.iwanoff.java.advanced.fourth;

import javafx.application.Application;
import javafx.stage.Stage;

public class CensusesFX extends Application {
```

```
    @Override
    public void start(Stage primaryStage) {

    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Also, a new FXML document named `CensusesForm.fxml` should be added to the project. In order for this file to be copied to the required location after compiling, it should be placed in a folder src\main\resources\ua\inf\iwanoff\java\advanced\fourth (relative to the project folder). The content of the file can be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.layout.BorderPane?>

<BorderPane xmlns:fx="http://javafx.com/fxml/1">
    <!-- TODO Add Nodes -->
</BorderPane>
```

Add a reference to the FXML document class and create a scene (the corresponding statements can be copied from the code of the previous examples)):

```
package ua.inf.iwanoff.java.advanced.fourth;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class CensusesFX extends Application {

    @Override
    public void start(Stage primaryStage) {
        try {
            BorderPane root = (BorderPane)FXMLLoader.load(getClass()
                                          .getResource("CensusesForm.fxml"));
            Scene scene = new Scene(root, 700, 500);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Population censuses");
            primaryStage.show();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        launch(args);
```

```
    }
}
```

It is also necessary to add a controller class:

```
package ua.inf.iwanoff.java.advanced.fourth;

public class CensusesController {

}
```

Further work involves adding and adjusting visual components using the Scene Builder application. We start the application using the context menu concerned with `CensusesForm.fxml` file in `Package Explorer (Open with SceneBuilder)`. First, we add a reference to the controller class. On the `Controller` tab in the left part of the Scene Builder window, in the `ControllerClass` text field, we enter the name of the controller class, specifying all subfolders. In our case, this is `ua.inf.iwanoff.java.advanced.fourth.CensusesController`.

Then we add the main menu. On the `Controls` panel, in the left part of the window (palette), we find the `MenuBar` component and add it to the top of the root container (the `insert TOP` position in the hierarchy of objects). The main menu that we have added already contains the `File` submenu (with the `Close` item), `Edit` (with the `Delete` item), and `Help` (with the `About` item). From the `Menu` tab we can add new submenus (`Menu`) and individual items (`MenuItem`). In our case, we'll also add the submenu `Run` and get the following main menu:

– File submenu with `New`, `Open...`, `Save...` and `Exit` items

– Edit submenu with `Add row` and `Remove last row` items

– Run submenu with the `Sort by population` and `Sort by comments` items;

– Help submenu with the `About...` item.

We can add a `Separator` component between the `Save...` and `Exit` items.

On the left side of the root panel, we need to place another panel (`AnchorPane`) for allocation of some controls: buttons, input lines, and the area of the search results. In particular, we'll add a `Label` with the text `Text to search:`,

the `TextField` line for entering words (sequence of letters) for the search in comments, as well as the corresponding buttons with the text `Search for a word` and `Search the letter sequence`. At the bottom of the added panel, we place the `TextArea` component to output the search results. For `TextField` and `TextArea` components, we need to define the names, respectively `textFieldCountry`, `textFieldArea`, `textFieldText`, and `textAreaResults`, since the contents of these components need to be accessed in the program.

For the `textAreaResults` results output area, we specify the value of the `AnchorPane.bottomAnchor` property, which will automatically change the height if the main window is resized. You can set this property directly in the text of an FXML document or with the help of Scene Builder by defining the required value in the lower text field of the `Anchor Pane Constraints` symbolic image in the `Layout` tab in the right part of the window.

In the central part of the root pane, we place the `TableView` component, which will display and edit the population census data. From the beginning, this component contains two columns (`TableColumn`) with headings `C1` and `C2`. This text should be changed to `Year` and `Population`. Two more columns (`Population density` and `Comments`) should be added to the table. The table and columns should also be renamed, respectively, `tableViewCensuses`, `tableColumnYear`, `tableColumnPopulation`, `tableColumnDensity`, and `tableColumnComments`. For the entire table (`tableViewCensuses`), and for all columns, besides `tableColumnDensity`, the editable property should be set to `true`, and for `tableColumnDensity`, it should be set to `false`. In the preview (`Preview | Show Preview in Window`) of the Scene Builder editor, the future main window of the application is represented on the fig. 4.1.

Now we can develop controller. Earlier created classes of the console application will act as a model. We can add a reference to `CountryWithArrayList` into the controller class. Named components must match

private fields that can be generated automatically. The mechanism for generating code using `Quick Fix` context menu has been described earlier.

It is also necessary to add event handler methods related to menu items, buttons, and text changes. To add an event, we create the corresponding functions with annotation and a parameter of `ActionEvent` type. References to these functions can be added in Scene Builder or manually.

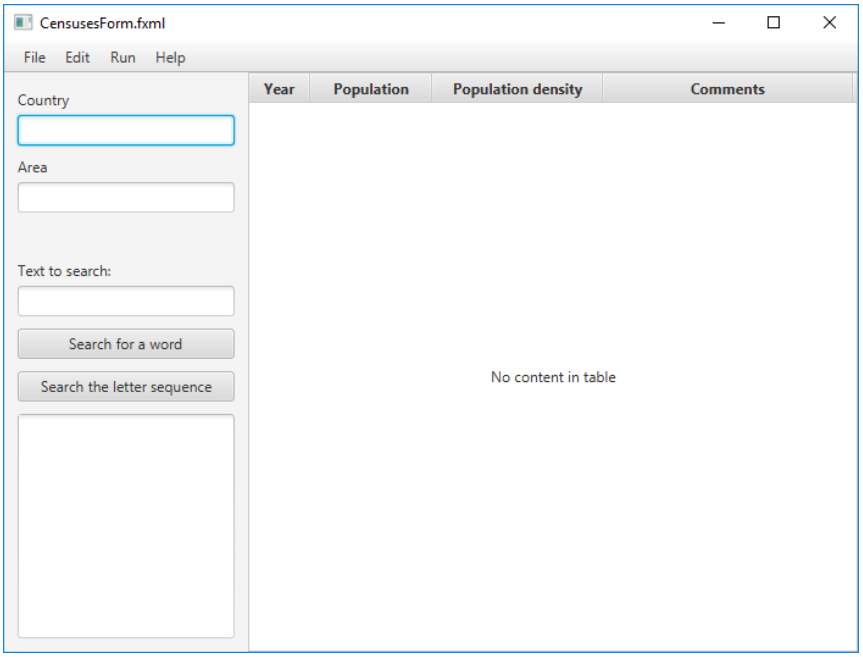

Figure 4.1 – Main window

The text of the `CensusesForm.fxml` file may be as follows:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<BorderPane prefHeight="500.0" prefWidth="700.0"
            xmlns="http://javafx.com/javafx/8"
xmlns:fx="http://javafx.com/fxml/1"

fx:controller="ua.inf.iwanoff.java.advanced.fourth.CensusesController">
    <top>
        <MenuBar BorderPane.alignment="CENTER">
            <Menu mnemonicParsing="false" text="File">
                <MenuItem mnemonicParsing="false" text="New"
                                                  onAction="#doNew"/>
                <MenuItem mnemonicParsing="false" text="Open..."
                                                  onAction="#doOpen"/>
                <MenuItem mnemonicParsing="false" text="Save..."
                                                  onAction="#doSave"/>
                <MenuItem mnemonicParsing="false" text="Exit"
                                                  onAction="#doExit"/>
```

```
            </Menu>
            <Menu mnemonicParsing="false" text="Edit">
                <MenuItem mnemonicParsing="false" text="Add row"
                                                  onAction="#doAdd"/>
                <MenuItem mnemonicParsing="false" text="Remove last row"
                                                  onAction="#doRemove"/>
            </Menu>
            <Menu mnemonicParsing="false" text="Run">
                <MenuItem mnemonicParsing="false" text="Sort by population"
                          onAction="#doSortByPopulation"/>
                <MenuItem mnemonicParsing="false" text="Sort by comments"
                          onAction="#doSortByComments"/>
            </Menu>
            <Menu mnemonicParsing="false" text="Help">
                <MenuItem mnemonicParsing="false" text="About..."
                          onAction="#doAbout"/>
            </Menu>
        </MenuBar>
    </top>
    <left>
        <AnchorPane prefHeight="472.0" prefWidth="200.0"
                     BorderPane.alignment="CENTER">
            <Label text="Country" AnchorPane.leftAnchor="11.0"
                   AnchorPane.rightAnchor="11.0" AnchorPane.topAnchor="14.0" />
            <TextField fx:id="textFieldCountry" prefHeight="22.0"
                       AnchorPane.leftAnchor="11.0"
                       AnchorPane.rightAnchor="11.0"
                       AnchorPane.topAnchor="35.0"
                       onAction="#nameChanged" />
            <Label text="Area" AnchorPane.leftAnchor="11.0"
                   AnchorPane.rightAnchor="11.0"
                   AnchorPane.topAnchor="69.0" />
            <TextField fx:id="textFieldArea" prefHeight="22.0"
                       AnchorPane.leftAnchor="11.0"
                       AnchorPane.rightAnchor="11.0"
                       AnchorPane.topAnchor="90.0" onAction="#areaChanged" />
            <Label text="Text to search:" AnchorPane.leftAnchor="11.0"
                   AnchorPane.rightAnchor="11.0" AnchorPane.topAnchor="154.0"/>
            <TextField fx:id="textFieldText" prefHeight="22.0"
                       AnchorPane.leftAnchor="11.0"
                       AnchorPane.rightAnchor="11.0"
                       AnchorPane.topAnchor="175.0" />
            <Button mnemonicParsing="false" prefHeight="22.0"
                    text="Search for a word" AnchorPane.leftAnchor="11.0"
                    AnchorPane.rightAnchor="11.0" AnchorPane.topAnchor="210.0"
                    onAction="#doSearchByWord"/>
            <Button mnemonicParsing="false" prefHeight="22.0"
                    text="Search the letter sequence"
                    AnchorPane.leftAnchor="11.0"
                    AnchorPane.rightAnchor="11.0" AnchorPane.topAnchor="245.0"
                    onAction="#doSearchBySubstring" />
            <TextArea fx:id="textAreaResults" AnchorPane.bottomAnchor="11.0"
                      AnchorPane.leftAnchor="11.0"
                      AnchorPane.rightAnchor="11.0"
                      AnchorPane.topAnchor="280.0" />
        </AnchorPane>
    </left>
    <center>
        <TableView fx:id="tableViewCensuses" prefHeight="473.0"
                   prefWidth="114.0"
                   BorderPane.alignment="CENTER" editable="true" >
```

```
            <columns>
                <TableColumn fx:id="tableColumnYear" prefWidth="50.0"
                             text="Year" editable="true" />
                <TableColumn fx:id="tableColumnPopulation" prefWidth="100.0"
                             text="Population"  editable="true" />
                <TableColumn fx:id="tableColumnDensity" prefWidth="140.0"
                             text="Population density"  editable="false" />
                <TableColumn fx:id="tableColumnComments" prefWidth="205.0"
                             text="Comments"  editable="true" />
            </columns>
        </TableView>
    </center>
</BorderPane>
```

Some auxiliary methods are useful for implementing event handlers. We can list all the code for the `CensusesController` class:

```
package ua.inf.iwanoff.java.advanced.fourth;

import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.TableColumn.CellEditEvent;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.control.cell.TextFieldTableCell;
import javafx.stage.FileChooser;
import javafx.util.converter.IntegerStringConverter;
import ua.inf.iwanoff.java.third.Census;
import ua.inf.iwanoff.java.fourth.CountryWithArrayList;
import ua.inf.iwanoff.java.advanced.third.FileUtils;
import ua.inf.iwanoff.java.third.CensusUtilities;
import ua.inf.iwanoff.java.third.CountryUtilities;

import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

/**
 * Controller class associated with the CensusesForm.fxml document
 *
 * Implementation of the Initializable interface provides the ability to
 * initialize the visual components described in the FXML document
 * in the initialize() method
 *
 */
public class CensusesController implements Initializable {
    // Reference to the model class:
    private CountryWithArrayList country = new CountryWithArrayList();

    // List whose contents will be displayed in the table:
    private ObservableList<Census> observableList;
```

```
    /**
     * Dialog box for an arbitrary message
     *
     * @param message message text
     */
    public static void showMessage(String message) {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("");
        alert.setHeaderText(message);
        alert.showAndWait();
    }

    /**
     * Error dialog box
     *
     * @param message message text
     */
    public static void showError(String message) {
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText(message);
        alert.showAndWait();
    }

    /**
     * Create a file selection dialog
     *
     * @param title window title
     */
    public static FileChooser getFileChooser(String title) {
        FileChooser fileChooser = new FileChooser();
        // We are starting to search from the current folder:
        fileChooser.setInitialDirectory(new File("."));
        // Set filters to find files:
        fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("XML files (*.xml)", "*.xml"));
        fileChooser.getExtensionFilters().add(
                new FileChooser.ExtensionFilter("All files (*.*)", "*.*"));
        // Specify the title of the window:
        fileChooser.setTitle(title);
        return fileChooser;
    }

    // Fields related to visual elements:

    @FXML
    private TextField textFieldCountry;
    @FXML
    private TextField textFieldArea;
    @FXML
    private TextField textFieldText;
    @FXML
    private TextArea textAreaResults;
    @FXML
    private TableView<Census> tableViewCensuses;
    @FXML
    private TableColumn<Census, Integer> tableColumnYear;
    @FXML
    private TableColumn<Census, Integer> tableColumnPopulation;
    @FXML
    private TableColumn<Census, Number> tableColumnDensity;
```

```
    @FXML
    private TableColumn<Census, String> tableColumnComments;

    /**
     * The method of initializing the visual components
     * described in the FXML document
     *
     */
    @Override
    public void initialize(URL location, ResourceBundle resources) {
        // Write an empty string instead of "No content in table":
        tableViewCensuses.setPlaceholder(new Label(""));
    }

    // Event handlers:

    @FXML
    private void doNew(ActionEvent event) {
        country = new CountryWithArrayList();
        observableList = null;
        textFieldCountry.setText("");
        textFieldArea.setText("");
        textFieldText.setText("");
        textAreaResults.setText("");
        tableViewCensuses.setItems(null);
        tableViewCensuses.setPlaceholder(new Label(""));
    }

    @FXML
    private void doOpen(ActionEvent event) {
        FileChooser fileChooser = getFileChooser("Open an XML file");
        File file;
        if ((file = fileChooser.showOpenDialog(null)) != null) {
            try {
                country =
                         FileUtils.deserializeFromXML(file.getCanonicalPath());
                // Fill in the text fields with read data:
                textFieldCountry.setText(country.getName());
                textFieldArea.setText(country.getArea() + "");
                textAreaResults.setText("");
                // Clear and update the table:
                tableViewCensuses.setItems(null);
                updateTable();
            }
            catch (IOException e) {
                showError("File not found");
            }
            catch (Exception e) {
                showError("Invalid file format");
            }
        }
    }

    @FXML
    private void doSave(ActionEvent event) {
        FileChooser fileChooser = getFileChooser("Save an XML file");
        File file;
        if ((file = fileChooser.showSaveDialog(null)) != null) {
            try {
                FileUtils.serializeToXML(createUpdatedCountry(),
                                         file.getCanonicalPath());
```

```
            showMessage("Results saved successfully");
        }
        catch (Exception e) {
            showError("Error writing to file");
        }
    }
}

@FXML
private void doExit(ActionEvent event) {
    Platform.exit(); // correct completion of the JavaFX application
}

@FXML
private void doAdd(ActionEvent event) {
    country.addCensus(0, 0, "");
    updateTable(); // create new data
}

@FXML
private void doRemove(ActionEvent event) {
    // We cannot remove the row if there is no data:
    if (observableList == null) {
        return;
    }
    // If there are rows, delete the last one:
    if (!observableList.isEmpty()) {
        observableList.removeLast();
    }
    // If there are no rows, we indicate that the data is missing:
    if (observableList.isEmpty()) {
        observableList = null;
    }
}

@FXML
private void doSortByPopulation(ActionEvent event) {
    CountryUtilities.sortByPopulation(country);
    tableViewCensuses.setItems(null);
    updateTable();
}

@FXML
private void doSortByComments(ActionEvent event) {
    CountryUtilities.sortByComments(country);
    tableViewCensuses.setItems(null);
    updateTable();
}

@FXML
private void doAbout(ActionEvent event) {
    Alert alert = new Alert(AlertType.INFORMATION);
    alert.setTitle("About...");
    alert.setHeaderText("Population censuses data");
    alert.setContentText("Version 1.0");
    alert.showAndWait();
}

@FXML
private void nameChanged(ActionEvent event) {
    // When the user changed the data in textFieldCountry,
```

```
        // we automatically update the name:
        country.setName(textFieldCountry.getText());
    }

    @FXML
    private void areaChanged(ActionEvent event) {
        // When the user changed the data in textFieldArea, we automatically
        // update the values of the territory and population density:
        try {
            double area = Double.parseDouble(textFieldArea.getText());
            country.setArea(area);
            setDensity();
        }
        catch (NumberFormatException e) {
            // If an error, we return, as it was:
            textFieldArea.setText(country.getArea() + "");
        }
    }

    @FXML
    private void doSearchByWord(ActionEvent event) {
        textAreaResults.setText("");
        for (int i = 0; i < country.censusesCount(); i++) {
            Census c = country.getCensus(i);
            if (CensusUtilities.containsWord(c, textFieldText.getText())) {
                showResults(c);
            }
        }
    }

    @FXML
    private void doSearchBySubstring(ActionEvent event) {
        textAreaResults.setText("");
        for (int i = 0; i < country.censusesCount(); i++) {
            Census c = country.getCensus(i);
            if (CensusUtilities.containsSubstring(c, textFieldText.getText())){
                showResults(c);
            }
        }
    }

    private void showResults(Census census) {
        textAreaResults.appendText("Census of the " +
                                           census.getYear() + " year.\n");
        textAreaResults.appendText("Comments:" + census.getComments() + "\n");
        textAreaResults.appendText("\n");
    }

    private CountryWithArrayList createUpdatedCountry() {
        // Rewriting data into the model from observableList
        CountryWithArrayList country = new CountryWithArrayList();
        for (Census c : observableList) {
            country.addCensus(c);
        }
        country.setName(textFieldCountry.getText());
        double area = Double.parseDouble(textFieldArea.getText());
        country.setArea(area);
        return country;
    }

    private void setDensity() {
```

```
        // Determine the mechanism of automatic recalculation of cells
        // for tableColumnDensity when other data changes:
        tableColumnDensity.setCellFactory(cell -> new TableCell<>() {
            @Override
            protected void updateItem(Number item, boolean empty) {
                int current = this.getTableRow().getIndex();
                if (observableList != null &&
                        current >= 0 &&
                        current < observableList.size() &&
                        country.getArea() > 0) {
                    double population =
                                    observableList.get(current).getPopulation();
                    double density = population / country.getArea();
                    setText(String.format("%7.2f", density));
                }
                else {
                    setText("");
                }
            }
        });
    }

    private void updateYear(CellEditEvent<Census, Integer> t) {
        // Update year data:
        Census c =
               t.getTableView().getItems().get(t.getTablePosition().getRow());
        c.setYear(t.getNewValue());
    }

    private void updatePopulation(CellEditEvent<Census, Integer> t) {
        // Update population data:
        Census c =
               t.getTableView().getItems().get(t.getTablePosition().getRow());
        c.setPopulation(t.getNewValue());
        setDensity(); // recalculate population density
    }

    private void updateComments(CellEditEvent<Census, String> t) {
        // Update comments:
        Census c =
               t.getTableView().getItems().get(t.getTablePosition().getRow());
        c.setComments(t.getNewValue());
    }

    private void updateTable() {
        // Fill in the observableList:
        List<Census> list = new ArrayList<>();
        observableList = FXCollections.observableList(list);
        for (int i = 0; i < country.censusesCount(); i++) {
            list.add((Census) country.getCensus(i));
        }
        tableViewCensuses.setItems(observableList);

        // We specify the columns associated with their properties
        // and the mechanism of editing, depending on the type of cells:
        tableColumnYear.setCellValueFactory(
                                            new PropertyValueFactory<>("year"));
        tableColumnYear.setCellFactory(
                TextFieldTableCell.forTableColumn(
                                    new IntegerStringConverter()));
        tableColumnYear.setOnEditCommit(this::updateYear);
```

```
        tableColumnPopulation.setCellValueFactory(
                                    new PropertyValueFactory<>("population"));
        tableColumnPopulation.setCellFactory(
                TextFieldTableCell.forTableColumn(
                                    new IntegerStringConverter()));
        tableColumnPopulation.setOnEditCommit(this::updatePopulation);
        tableColumnDensity.setSortable(false); // this column cannot be sorted
        setDensity();
        tableColumnComments.setCellValueFactory(
                                    new PropertyValueFactory<>("comments"));
        tableColumnComments.setCellFactory(
                                        TextFieldTableCell.forTableColumn());
        tableColumnComments.setOnEditCommit(this::updateComments);
    }
}
```

The work of the program can be improved by switching the availability of individual elements using the `setDisable()` method depending on the state of the program.

Since the JavaFX library is added to a previously created project, additional settings must be made.

4 Quiz

1) What Java libraries are used for creation of GUI applications?

2) What is an applet?

3) What is the idea of event-driven programming?

4) What is JavaFX? What are the benefits of JavaFX?

5) What are JavaFX properties, how are they implemented and what opportunities do they provide?

6) How the binding between JavaFX objects is implemented?

7) What is `ObservableList`?

8) What is FXML? What are the benefits of FXML?

9) What is MVC?

10) What is the layout and how is it implemented in JavaFX?

11) What standard containers does JavaFX provide and how are they different?

12) How are the `RadioButton` buttons working?

13) What are the features of modal dialog windows?

14) How JavaFX uses standard file choose windows?

15) How to work with tabular data in JavaFX?

16) How to associate table columns with properties of objects in a list?

17) How to provide editing table cells in JavaFX?

18) How to provide the visual editing of the application window of the graphical user interface?

LABORATORY TRAINING 5

Reflection. Creation of database applications

1 Training tasks

1.1 Creating a database console application

Create a database console application for working with the entities of individual tasks of previous laboratory trainings.

Domain entity data must be stored in a relational database. The MySQL DBMS should be used (or another DBMS, upon agreement with the teacher). In the console application, you should create two database tables, respectively for the first and second entity of the individual task (first and second table).

After creating the tables, you must perform the following actions:

– import into the database from a JSON file;

– display of both tables;

– search by features defined in the second laboratory training of the Java basic programming course;

– performing sorting according to the features defined in the second laboratory training of the Java Basic Programming Course;

– adding a new record to the table;

– deleting certain records from the table;

– export from the database to a JSON file.

1.2 View all class fields

Implement a program that reads positive integer values from the text file (the numbers are separated by spaces, it should be read to the end of the file), places these numbers into an array, sorts them in descending order and increasing the sum of digits and stores both results in two new text files. The above actions should be implemented in separate static functions. To define the sort order, create two classes that implement the Comparator interface.

1.3 Creating a database GUI application (advanced task)

Create classes Student and Academic Group (with an array of students as a field). Create objects, store data about the students of the academic group in the archive. Read data from the archive in another program.

1.4 Obtaining data about method annotations (advanced task)

Create classes Student and Academic Group (with an array of students as a field). Create objects, store data about the students of the academic group in the archive. Read data from the archive in another program.

2 Guidelines

The goal of this laboratory training is obtaining practical skills in working with Reflection and creation of Database Applications.

The following questions should be explored:

– using RTTI;

– using class loaders;

– reflection;

– creating and using annotations;

– using annotations in Java libraries and frameworks;

– Java tools for working with databases.

Necessary theoretical concepts and syntax constructs are discussed in lectures on the topics “Reflection”, Working with databases” and “Multithreading”.

Most tasks can be implemented based on sample programs discussed below.

3 Sample programs

3.1 Obtaining data about class methods

Suppose we need to create a program that displays data about methods of a class: name, resulting type and types of parameters. The program will look like this:

```
package ua.inf.iwanoff.oop.fifth;

import java.lang.reflect.Method;
```

```
import java.util.Scanner;

public class ShowAllMethods {

    public static void main(String[] args) {
        System.out.print("Enter the full name of the class: ");
        String className = new Scanner(System.in).next();
        try {
            Class<?> c = Class.forName(className);
            for (Method m : c.getMethods()) {
                System.out.printf("Name: %s Resulting type: %s%n",
                            m.getName(), m.getReturnType().getCanonicalName());
                for (Class<?> type : m.getParameterTypes()) {
                    System.out.printf(
                            "  Parameter type: %s%n", type.getCanonicalName());
                }
            }
        }
        catch (ClassNotFoundException e) {
            System.err.println("Error entering class name!");
        }
    }
}
```

3.2 Calculating the results of Math class functions

In the following example, functions `getMethod()` and `invoke()` are used to call static methods of `java.lang.Math` class (only methods with one parameter can be called):

```
package ua.inf.iwanoff.oop.fifth;

import java.lang.reflect.Method;
import java.util.InputMismatchException;
import java.util.Scanner;

public class SomeFunction {
    private Method method = null;
    private String methodName = null;
    private String className = null;

    public SomeFunction(String className, String methodName) {
        this.className = className;
        this.methodName = methodName;
    }

    public double getY(double x) throws Exception {
        if (method == null) {
            // Create an object of type Method, if it was not created before.
            // The method must accept one double argument:
            method = Class.forName(className).getMethod(methodName,
                                                        double.class);
        }
        // Call the method:
        return (Double) method.invoke(null, x);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.println("Enter the name of the static class Math method:");
        SomeFunction someFunction = new SomeFunction("java.lang.Math",
                                                              scanner.next());
        System.out.println(
                        "Enter the beginning, end of the interval, and step:");
        try {
            double from = scanner.nextDouble();
            double to = scanner.nextDouble();
            double step = scanner.nextDouble();
            for (double x = from; x <= to; x += step) {
                System.out.println(x + "\t" + someFunction.getY(x));
            }
        }
        catch (InputMismatchException e) {
            System.out.println("Error entering data");
        }
        catch (Exception e) {
            System.out.println("Error calling function");
        }
    }
}
```

3.3 Output of information about methods marked with a certain annotation

In the following example, several classes are loaded by name and methods without parameters marked with the annotation `@ToInvoke` are called. We define the annotation:

```
package ua.inf.iwanoff.java.advanced.fifth;

import java.lang.annotation.*;

@Target(value = ElementType.METHOD)
@Retention(value = RetentionPolicy.RUNTIME)
public @interface ToInvoke { }
```

In the program, we create two classes with methods marked with an annotation `@ToInvoke` and call these methods:

```
package ua.inf.iwanoff.java.advanced.fifth;

import java.lang.reflect.*;

class ATest {

    public void aFirst() {
        System.out.println("aFirst launched");
    }

    @ToInvoke
    public void aSecond() {
        System.out.println("aSecond launched");
    }

    public void aThird() {
```

```
        System.out.println("aThird launched");
    }
}

class BTest {

    @ToInvoke
    public void bFirst() {
        System.out.println("bFirst launched");
    }

    public void bSecond() {
        System.out.println("bSecond launched");
    }

    @ToInvoke
    public void bThird() {
        System.out.println("bThird launched");
    }
}

public class LaunchIfAnnotated {

    static void invokeFromClass(String className) {
        System.out.println("---------- class " + className + " ----------");
        try {
            Class<?> cls = Class.forName(className);
            Method[] methods = cls.getMethods();
            for (Method m : methods) {
                if (m.isAnnotationPresent(ToInvoke.class)) {
                    m.invoke(cls.newInstance());
                }
                else {
                   System.out.println("No annotation before " + m.getName());
                }
            }
        }
        catch (ClassNotFoundException | SecurityException |
               InstantiationException | IllegalAccessException |
               IllegalArgumentException | InvocationTargetException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        invokeFromClass("ua.inf.iwanoff.java.advanced.fifth.ATest");
        invokeFromClass("ua.inf.iwanoff.java.advanced.fifth.BTest");
    }
}
```

3.4 Database application for working with census data

In laboratory training No. 3, a Maven project was created, in which work is carried out with data about the country and population censuses. Now this project can be expanded.

In the console application, we should create two relational database tables, for countries and censuses, respectively.

After creating the tables, we should perform the following actions:

– import into the database from a JSON file;

– display data of both tables;

– sorting of censuses of a certain country by population growth;

– adding a new census to a specific country;

– deletion of the census;

– adding a new country;

– search for censuses with a certain word in the comment;

– export from the database to a JSON file.

We add a new package `ua.inf.iwanoff.java.advanced.fifth` to the project. In it, we create derived classes to represent the census and the country, adding the necessary constructors. The new field id added in these classes can be useful when interacting with the database. The `CensusForDB` class will be as follows:

```
package ua.inf.iwanoff.java.advanced.fifth;

import ua.inf.iwanoff.java.third.Census;

/**
 * Class to represent the census.
 * Added identifier for potential database storage
 */
public class CensusForDB extends Census {
    private long id = -1;

    /**
     * Returns the census ID
     *
     * @return Census ID
     */
    public long getId() {
        return id;
    }

    /**
     * Sets the census ID
     *
     * @param id Census ID
     */
    public void setId(long id) {
        this.id = id;
    }

    /**
     * The constructor initializes the object with default values
     */
```

```
    public CensusForDB() {
    }

    /**
     * The constructor initializes the object with the specified values
     *
     * @param year census year
     * @param population the population
     * @param comments comment text
     */
    public CensusForDB(int year, int population, String comments) {
        super(year, population, comments);
    }
}
```

The code of class `CountryForDB` will be as follows

```
package ua.inf.iwanoff.java.advanced.fifth;

import ua.inf.iwanoff.java.fourth.CountryWithArrayList;

/**
 * A class to represent the country in which the census is conducted.
 * The identifier added for potential database storage
 */
public class CountryForDB extends CountryWithArrayList {
    private long id = -1;

    /**
     * Returns the country ID
     *
     * @return Country ID
     */
    public long getId() {
        return id;
    }

    /**
     * Sets the country ID
     *
     * @param id Country ID
     */
    public void setId(long id) {
        this.id = id;
    }

    /**
     * The constructor initializes the object with default values
     */
    public CountryForDB() {
    }

    /**
     * The constructor initializes the object with the specified values
     *
     * @param name the name of the country
     * @param area the area of the country
     */
    public CountryForDB(String name, double area) {
        setName(name);
        setArea(area);
```

```
    }
}
```

Since several countries can be present, it is advisable to create a new class `Countries` that contains a list of countries:

```
package ua.inf.iwanoff.java.advanced.fifth;

import java.util.List;
import java.util.ArrayList;

/**
 * A class to represent a list of countries
 */
public class Countries {
    private List<CountryForDB> list = new ArrayList<>();

    /**
     * Returns the list of countries (java.util.List)
     * @return list of countries
     */
    public List<CountryForDB> getList() {
        return list;
    }

    /**
     * Returns a string representation of the object
     *
     * @return a string representation of the list
     */
    @Override
    public String toString() {
        return list.toString();
    }
}
```

Now you should go to the settings related to the database. First, install the MySQL database server (if not already installed). You can install the client-server version of the MySQL DBMS for Windows using the MySQL Installer program (`http://dev.mysql.com/downloads/installer/`). This program can be downloaded without registration. Choose an option `mysql-installer-community-8.0.36.0.msi` (the version may change). On the first page of the MySQL installer, select a configuration option. It is possible to choose either `Developer Default or Server only`, which is enough for our work. There are also options for advanced users. If `Server only` option was selected, click the `Execute` button on the next page. The server installation process begins.

The server installation will require the presence of Microsoft Visual C++ 2019 Redistributable Package. If it has not been previously installed, you should

click the `Execute` button and install the appropriate package. After that, in the main installer window, you can click `Next`, then `Execute` again and `Next` again.

On the `Type and Networking`, you can leave everything unchanged. On the `Authentication Method` page, you can leave the `Use Strong Password Encryption for Authentication` option. Next, press `Next` twice again, enter the password. In real practice, complex passwords should be used, but for our work we will set a password `"root"`. On the `Apply Configuration` page, click the `Execute` button, then `Finish`. On the following pages, we confirm the installation of MySQL.

The following dependency must be added to the `pom.xml` file:

```
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <version>9.4.0</version>
    <scope>runtime</scope>
</dependency>
```

The main work will be done in the class DbUtils:

```
package ua.inf.iwanoff.java.advanced.fifth;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;
import com.thoughtworks.xstream.security.AnyTypePermission;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import java.util.ArrayList;

/**
 * The class implements work with a database in the task
 * of processing data on population censuses.
 * MySQL database server is used
 */
public class DbUtils {
    /**
     * Enumeration for defining order of census data output
     */
    public enum Show { SORTED, UNSORTED };

    // Constants that contain the necessary SQL queries:
    public static final String DROP_TABLES =
                            "DROP TABLES IF EXISTS censuses, countries";
    public static final String DROP_DATABASE =
                            "DROP DATABASE IF EXISTS countriesDB";
    public static final String CREATE_DATABASE = "CREATE DATABASE countriesDB";
    public static final String CREATE_TABLE_COUNTRIES = """
```

```
            CREATE TABLE countriesDB.countries (
              CountryID INT NOT NULL AUTO_INCREMENT,
              Name VARCHAR(128) NULL,
              Area DECIMAL(10,2) NULL,
              PRIMARY KEY (CountryID));
            """;
    public static final String CREATE_TABLE_CENSUSES = """
            CREATE TABLE countriesDB.censuses (
              CensusID INT NOT NULL AUTO_INCREMENT,
              Year INT NULL,
              Population INT NULL,
              CountryID INT NULL,
              Comment VARCHAR(256) NULL,
              PRIMARY KEY (CensusID),
              INDEX CountryID_idx (CountryID ASC) VISIBLE,
              CONSTRAINT CountryID
                FOREIGN KEY (CountryID)
                REFERENCES countriesDB.countries (CountryID)
                ON DELETE NO ACTION
                ON UPDATE NO ACTION);

            """;
    private static final String INSERT_INTO_COUNTRIES = """
        INSERT INTO countriesDB.countries (Name, Area) VALUES (?, ?);
        """;
    private static final String INSERT_INTO_CENSUSES = """
        INSERT INTO countriesDB.censuses (Year, Population, CountryID, Comment)
        VALUES (?, ?, ?, ?);
        """;
    private static final String SELECT_BY_NAME =
        "SELECT * FROM countriesDB.countries WHERE Name = ?";
    private static final String SELECT_ALL_COUNTRIES =
        "SELECT * FROM countriesDB.countries";
    private static final String SELECT_FROM_CENSUSES =
        "SELECT * FROM countriesDB.censuses WHERE CountryID = ?";
    private static final String SELECT_FROM_CENSUSES_ORDER_BY_POPULATION =
"SELECT * FROM countriesDB.censuses WHERE CountryID = ? ORDER BY Population";
    private static final String SELECT_FROM_CENSUSES_WHERE_WORD = """
      SELECT c.CensusID, c.Year, c.Population, c.Comment, l.Name FROM
      countriesDB.censuses c
      INNER JOIN countriesDB.countries l ON c.CountryID = l.CountryID
      WHERE c.Comment LIKE '%key_word%';
        """;
    private static final String DELETE_BY_YEAR =
        "DELETE FROM countriesDB.censuses WHERE CountryID=? AND Year=?";

    private static Connection connection;

    /**
     * Deserializes data from the specified JSON file
     *
     * @param fileName the name of the file
     * @return the object that was created
     */
    public static Countries importFromJSON(String fileName) {
        try {
            XStream xStream = new XStream(new JettisonMappedXmlDriver());
            xStream.addPermission(AnyTypePermission.ANY);
            xStream.alias("countries", Countries.class);
            xStream.alias("country", CountryForDB.class);
            xStream.alias("census", CensusForDB.class);
```

```
            return (Countries) xStream.fromXML(new File(fileName));
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Serializes data into the specified JSON file
     *
     * @param country the country
     * @param fileName the name of the file
     */
    public static void exportToJSON(Countries countries, String fileName) {
        XStream xStream = new XStream(new JettisonMappedXmlDriver());
        xStream.alias("countries", Countries.class);
        xStream.alias("country", CountryForDB.class);
        xStream.alias("census", CensusForDB.class);
        String xml = xStream.toXML(countries);
        try (FileWriter fw = new FileWriter(fileName);
             PrintWriter out = new PrintWriter(fw)) {
            out.println(xml);
        }
        catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Serializes data from database into the specified JSON file
     *
     * @param fileName the name of the file
     */
    public static void exportToJSON(String fileName) {
        Countries countries = getCountriesFromDB();
        exportToJSON(countries, fileName);
    }

    /**
     *  Creating a connection to the database
     */
    public static void createConnection() {
        try {
            connection = DriverManager.getConnection(
                       "jdbc:mysql://localhost/mysql?user=root&password=root");
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Creation of database and tables with deletion of previous ones
     */
    public static boolean createDatabase() {
        try {
            Statement statement = connection.createStatement();
            statement.executeUpdate(DROP_TABLES);
            statement.executeUpdate(DROP_DATABASE);
            statement.executeUpdate(CREATE_DATABASE);
            statement.executeUpdate(CREATE_TABLE_COUNTRIES);
```

```
            statement.executeUpdate(CREATE_TABLE_CENSUSES);
            return true;
        }
        catch (SQLException e) {
            return false;
        }
    }

    /**
     * Entry of all data from the object to the database
     *
     * @param countries an object with data about countries
     */
    public static void addAll(Countries countries) {
        for (CountryForDB c : countries.getList()) {
            addCountry(c);
        }
    }

    /**
     *  Putting all data from the object to the database
     *
     *  @param country the country about which data is entered
     */
    public static void addCountry(CountryForDB country) {
        try {
            PreparedStatement preparedStatement =
                           connection.prepareStatement(INSERT_INTO_COUNTRIES);
            preparedStatement.setString(1, country.getName());
            preparedStatement.setDouble(2, country.getArea());
            preparedStatement.execute();
            for (int i = 0; i < country.censusesCount(); i++) {
                addCensus(country.getName(),
                                            (CensusForDB)country.getCensus(i));
            }
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Creates a Countries object into which all data
     * from the database is entered
     *
     * @return object to which data about countries is put
     */
    public static Countries getCountriesFromDB() {
        try {
            Countries countries = new Countries();
            Statement statement = connection.createStatement();
            ResultSet setOfCountries =
                                    statement.executeQuery(SELECT_ALL_COUNTRIES);
            while (setOfCountries.next()) {
                countries.getList().add(getCountryFromDB(setOfCountries));
            }
            return countries;
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
```

```
    }

    /**
     * Creates a country object, filling it with data from the database
     *
     * @param name the name of the country
     * @return an object filled with data from the database
     */
    public static CountryForDB getCountryByName(String name) {
        try {
            PreparedStatement preparedStatement =
                                connection.prepareStatement(SELECT_BY_NAME);
            preparedStatement.setString(1, name);
            ResultSet setOfCountries = preparedStatement.executeQuery();
            setOfCountries.next();
            return getCountryFromDB(setOfCountries);
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Creates and returns a country object from the result dataset
     *
     * @param setOfCountries the data set from which the data is obtained
     * @return created country with census data
     * @throws SQLException exception related to SQL query error
     */
    public static CountryForDB getCountryFromDB(ResultSet setOfCountries)
                                                        throws SQLException {
        CountryForDB country =
                            new CountryForDB(setOfCountries.getString("Name"),
                                             setOfCountries.getDouble("Area"));
        int id = setOfCountries.getInt("CountryID");
        country.setId(id);
        PreparedStatement preparedStatement =
                             connection.prepareStatement(SELECT_FROM_CENSUSES);
        preparedStatement.setInt(1, id);
        ResultSet setOfCensuses = preparedStatement.executeQuery();
        while (setOfCensuses.next()) {
            CensusForDB census = new CensusForDB(setOfCensuses.getInt("Year"),
                                         setOfCensuses.getInt("Population"),
                                         setOfCensuses.getString("Comment"));
            census.setId(setOfCensuses.getInt("CensusID"));
            country.addCensus(census);
        }
        return country;
    }

    /**
     * Gets the country ID by name
     *
     * @param countryName the name of the country
     * @return Country ID in database
     */
    public static int getIdByName(String countryName) {
        try {
            PreparedStatement preparedStatement =
                                connection.prepareStatement(SELECT_BY_NAME);
            preparedStatement.setString(1, countryName);
```

```
            ResultSet resultSet = preparedStatement.executeQuery();
            resultSet.next();
            return resultSet.getInt("CountryID");
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Outputs all data from the database to the console,
     * sequentially for each country
     */
    public static void showAll() {
        try {
            PreparedStatement preparedStatement =
                            connection.prepareStatement(SELECT_ALL_COUNTRIES);
            ResultSet resultSet = preparedStatement.executeQuery();
            ArrayList<String> names = new ArrayList<>();
            while (resultSet.next()) {
                String name = resultSet.getString("Name");
                names.add(name);
            }
            resultSet.close();
            for (String name : names) {
                showCountry(name, Show.UNSORTED);
            }
        }
        catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Displays data about countries by name on the console
     *
     * @param countryName the name of the country
     * @param byPopulation the output order specified by the Show enumeration
     */
    public static void showCountry(String countryName, Show byPopulation) {
        try {
            PreparedStatement preparedStatement =
                                    connection.prepareStatement(SELECT_BY_NAME);
            preparedStatement.setString(1, countryName);
            ResultSet resultSet = preparedStatement.executeQuery();
            System.out.printf("%s\t  %s\t  %s%n", "ID", "Country", "Area");
            resultSet.next();
            System.out.printf("%s\t  %s\t  %s%n",
                    resultSet.getString("CountryID"),
                    resultSet.getString("Name"), resultSet.getString("Area"));
            resultSet.close();
            PreparedStatement anotherStatement;
            if (byPopulation == Show.SORTED) {
                anotherStatement = connection.prepareStatement(
                                    SELECT_FROM_CENSUSES_ORDER_BY_POPULATION);
            }
            else {
                anotherStatement =
                            connection.prepareStatement(SELECT_FROM_CENSUSES);
            }
            anotherStatement.setInt(1, getIdByName(countryName));
```

```
            ResultSet anotherSet = anotherStatement.executeQuery();
            System.out.printf("%s\t  %s\t  %s \t%s%n", "ID", "Year",
                              "Population", "Comments");
            while (anotherSet.next()) {
                System.out.printf("%s\t  %s\t  %s\t\t%s%n",
                        anotherSet.getString("CensusID"),
                        anotherSet.getString("Year"),
                        anotherSet.getString("Population"),
                        anotherSet.getString("Comment"));
            }
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Displays census data sorted by population growth
     *
     * @param countryName the country for which censuses are output
     */
    public static void showSortedByPopulation(String countryName) {
        showCountry(countryName, Show.SORTED);
    }

    /**
     * Outputs to the console data about all censuses,
     * the comments of which contain a certain word
     *
     * @param word the word to search for
     */
    public static void findWord(String word) {
        try {
            String query =
                     SELECT_FROM_CENSUSES_WHERE_WORD.replace("key_word", word);
            PreparedStatement preparedStatement =
                                            connection.prepareStatement(query);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                System.out.printf("%s\t  %s\t  %s\t  %s\t\t%s%n",
                        resultSet.getString("CensusID"),
                        resultSet.getString("Name"),
                        resultSet.getString("Year"),
                        resultSet.getString("Population"),
                        resultSet.getString("Comment"));
            }
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Adds information about the census of a specific country to the database
     *
     * @param countryName the name of the country whose census is being added
     * @param census the census to add
     */
    public static void addCensus(String countryName, CensusForDB census) {
        CountryForDB country = getCountryByName(countryName);
        try {
```

```
            PreparedStatement preparedStatement =
                            connection.prepareStatement(INSERT_INTO_CENSUSES);
            preparedStatement.setInt(1, census.getYear());
            preparedStatement.setInt(2, census.getPopulation());
            preparedStatement.setInt(3, getIdByName(country.getName()));
            preparedStatement.setString(4, census.getComments());
            preparedStatement.execute();
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Deletes the specified census from the database
     *
     * @param countryName the name of the country
     * @param year the year of the census to remove
     */
    public static void removeCensus(String countryName, int year) {
        try {
            PreparedStatement preparedStatement =
                                    connection.prepareStatement(DELETE_BY_YEAR);
            preparedStatement.setInt(1, getIdByName(countryName));
            preparedStatement.setInt(2, year);
            preparedStatement.execute();
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Closes the connection to the database
     */
    public static void closeConnection() {
        try {
            connection.close();
        }
        catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
}
```

In the class with the `main()` function, we also add a function for creating the necessary objects with saving data into a JSON file. From this file, data is then read and imported into the database, then all planned actions on the data are performed:

```
package ua.inf.iwanoff.java.advanced.fifth;

import static ua.inf.iwanoff.java.advanced.fifth.DbUtils.*;

/**
 * The class demonstrates working with the MySQL database
 */
public class DbProgram {
```

```
    /**
     * Demonstration of the program.
     * Data is imported from a JSON file, sorted, added and deleted.
     * The result is exported to another JSON file
     *
     * @param args command line arguments (not used)
     */
    public static void main(String[] args) {
        Countries countries = createCountries();
        exportToJSON(countries, "Countries.json");
        countries = importFromJSON("Countries.json");
        createConnection();
        if (createDatabase()) {
            addAll(countries);
            showAll();
            System.out.println("\nCensuses in Ukraine by population growth:");
            showSortedByPopulation("Ukraine");
            System.out.println("\nAdding census:");
            addCensus("Ukraine", new CensusForDB(2026, 49000000,
                                                    "Maybe there will be?"));
            showAll();
            System.out.println("\nRemoving census:");
            removeCensus("Ukraine", 2026);
            showAll();
            System.out.println("\nAdding the country:");
            addCountry(new CountryForDB("Germany", 357588));
            addCensus("Germany", new CensusForDB(2011, 80200000,
                                        "The first census in united Germany"));
            showAll();
            System.out.println("\nSearching for the word \"census\":");
            findWord("census");
            exportToJSON("CountriesFromDB.json");
            closeConnection();
        }
    }

    /**
     * Creating the Countries object and filling it with data
     * to demonstrate the program
     *
     * @return an object that contains the necessary data
     */
    static Countries createCountries() {
        CountryForDB country = new CountryForDB();
        country.setName("Ukraine");
        country.setArea(603628);
        country.addCensus(new CensusForDB(1959, 41869000,
                          "The first census after World War II"));
        country.addCensus(new CensusForDB(1970, 47126500,
                          "Population increases"));
        country.addCensus(new CensusForDB(1979, 49754600, "No comments"));
        country.addCensus(new CensusForDB(1989, 51706700,
                          "The last soviet census"));
        country.addCensus(new CensusForDB(2001, 48475100,
                          "The first census in the independent Ukraine"));
        Countries countries = new Countries();
        countries.getList().add(country);
        return countries;
    }
}
```

The result of execution is output to the console of working with data, as well as a new file `CountriesFromDB.json`.

4 Quiz

1) What is the essence of RTTI?

2) Why use the keyword instanceof?

3) Give a definition of reflection.

4) What information is stored in the class field?

5) How to load classes in Java?

6) What are the different types of class loaders?

7) How to load a class by name?

8) What are custom loaders for?

9) How to get information about class methods?

10) How to call a method by name?

11) How to call a static method?

12) How to get information about class fields?

13) Why are annotations used?

14) What are the standard annotations?

15) How does the presence of an instruction affect the execution of an annotated method?

16) How are the annotation characteristics determined?

17) How to set the lifetime of the annotation?

18) How to create your own annotation?

19) How to check the presence of an annotation?

20) How to get a list of all annotations?

21) What features does the Lombok library provide?

22) What are the approaches to the implementation of database applications, depending on the task?

23) Define JDBC.

24) What is a JDBC driver?

25) What are the different methods of executing SQL commands?

LIST OF SOURCES

Recommended Reading:

1 Schildt H. Java: A Beginner's Guide: 8th Edition, McGraw-Hill Education, 2018, 684 p.

2 Schildt H. Java: The Complete Reference: 11th Edition, McGraw-Hill Education, 2018, 1208 p.

3 Bloch J. Effective Java: 3rd Edition, Addison Wesley, 2017, 412 p.

4 Horstmann C. S. Core Java Volume I – Fundamentals: 11th Edition, Prentice Hall 2018, 889 p.

5 Horstmann C. S. Core Java SE 9 for the Impatient: 2nd Edition Addison-Wesley Professional, 2017, 576 p.

6 Eckel B. Thinking in Java 4th Edition: Pearson, 2006, 1150 p.

7 Deitel P., Deitel H. Java How to Program, Early Objects: 11th Edition, Pearson, 2017, 1296 p.

8 Deitel P., Deitel H. Java How To Program, Late Objects: 11th Edition, Pearson, 2017, 1248 p...

Internet Resources:

9 Java Programming Language // https://www.geeksforgeeks.org/java/

10 Java Tutorials // http://docs.oracle.com/javase/tutorial

11 Java Tutorial // https://www.w3schools.com/java/

12 Java Tutorial // http://www.java2s.com/Tutorial/Java/CatalogJava.htm

13 Learn Java Programming // https://www.programiz.com/java-programming

14 Bruce Eckel. On Java 8 // http://www.sufuq.com/books/java/On%20Java%208.pdf

15 Bruce Eckel. Thinking in Java, 4th Edition // http://sd.blackball.lv/library/Thinking_in_Java_4th_edition.pdf

Educational edition

GUIDELINES

to the laboratory work for the courses “Advanced Java programming course” for students of specialties 121 “Software engineering” and 122 “Computer Science”

Author:
Ivanov Lev Vadymovych

Responsible for the issue Assoc. Prof. Kopp A.M.
The paper has been recommended for publication by Prof. Hamaiun I.P.

In the author's edition

Plan of 2025 year, position 162

Signed for publication on 13.02.2025. Headset Times New Roman.

Publishing Centre of NTU "KhPI
Certificate of state registration DK № 5478 of 21.08.2017

Electronic publication