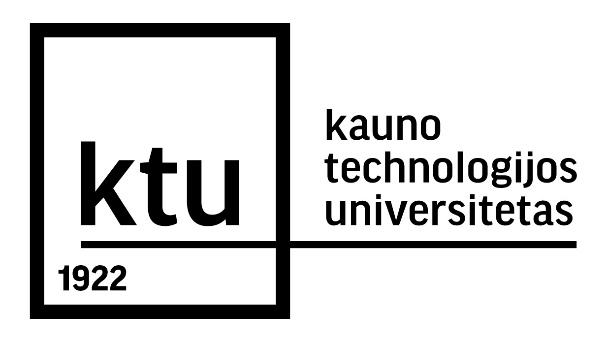
**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INFORMATIKOS FAKULTETAS**

**Modulio P175B014 „Duomenų struktūros“**

Laboratorinio darbo Nr.3 ataskaita

|  |  |
| --- | --- |
|  | **Dėstytojas**  lekt. BUDNIKAS Aurelijus |
|  |
| **Studentas**  Matas Vaitkevičius IFF-8/8 gr. |

**KAUNAS, 2019**

Turinys

[**Lab3.** **– Maišos lentelės duomenų struktūrų tyrimas** 3](#_Toc26209095)

[**Išvados** 44](#_Toc26209096)

**Lab3.** **– Maišos lentelės duomenų struktūrų tyrimas**

1. Pagal duotą **Car** klasės pavyzdį sukurkite individualiai pasirinktų elementų klases (4-5 komponentai), tenkinančias **Parsable** interfeisą. Programinį kodą rašykite į individualų paketą **edu.ktu.ds.lab3.pavarde**.

Book.java:

package edu.ktu.ds.lab3.vaitkevicius;

import edu.ktu.ds.lab3.utils.Ks;

import edu.ktu.ds.lab3.utils.Parsable;

import java.time.LocalDate;

import java.util.\*;

/\*\*

\* @author MV

\*/

public final class Book implements Parsable<Book> {

// Bendri duomenys visiems automobiliams (visai klasei)

private static final int minYear = 1990;

private static final int currentYear = LocalDate.now().getYear();

private static final double minPrice = 100.0;

private static final double maxPrice = 333000.0;

private String author = "";

private String title = "";

private int year = -1;

private double price = -1.0;

public Book() {

}

public Book(String author, String title, int year, double price) {

this.author = author;

this.title = title;

this.year = year;

this.price = price;

validate();

}

public Book(String dataString) {

this.parse(dataString);

validate();

}

public Book(Builder builder) {

this.author = builder.author;

this.title = builder.title;

this.year = builder.year;

this.price = builder.price;

validate();

}

private void validate() {

String errorType = "";

if (year < minYear || year > currentYear) {

errorType = "Netinkami gamybos metai, turi būti ["

+ minYear + ":" + currentYear + "]";

}

if (price < minPrice || price > maxPrice) {

errorType += " Kaina už leistinų ribų [" + minPrice

+ ":" + maxPrice + "]";

}

if (!errorType.isEmpty()) {

Ks.ern("Automobilis yra blogai sugeneruotas: " + errorType);

}

}

@Override

public void parse(String dataString) {

try { // duomenys, atskirti tarpais

Scanner scanner = new Scanner(dataString);

author = scanner.next();

title = scanner.next();

year = scanner.nextInt();

price = scanner.nextDouble();

} catch (InputMismatchException e) {

Ks.ern("Blogas duomenų formatas -> " + dataString);

} catch (NoSuchElementException e) {

Ks.ern("Trūksta duomenų -> " + dataString);

}

}

@Override

public String toString() {

return author + "\_" + title + ":" + year + " "

+ String.format("%4.1f", price);

}

public String getAuthor() {

return author;

}

public String getTitle() {

return title;

}

public int getYear() {

return year;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

@Override

public int hashCode() {

int hash = 5;

hash = 29 \* hash + Objects.hashCode(this.author);

hash = 29 \* hash + Objects.hashCode(this.title);

hash = 29 \* hash + this.year;

hash = 29 \* hash + (int) (Double.doubleToLongBits(this.price) ^ (Double.doubleToLongBits(this.price) >>> 32));

return hash;

}

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null) {

return false;

}

if (getClass() != obj.getClass()) {

return false;

}

final Book other = (Book) obj;

if (this.year != other.year) {

return false;

}

if (Double.doubleToLongBits(this.price) != Double.doubleToLongBits(other.price)) {

return false;

}

if (!Objects.equals(this.author, other.author)) {

return false;

}

if (!Objects.equals(this.title, other.title)) {

return false;

}

return true;

}

// Car klases objektų gamintojas

public static class Builder {

private final static Random RANDOM = new Random(1949); // Atsitiktinių generatorius

private final static String[][] TITLES = { // galimų autorių ir jo knygu pavadinimų masyvas

{"Jonas Biliūnas", "Kliudžiau", "Liūdna Pasaka", "Ir Rados stebuklas",

"Laimės žiburys"},

{"Maironis", "Kur bėga Šešupė",

"Lietuva", "Mūsu vargai", "Pavasario Balsai", "Nuo Birutės kalno"},

{"Balys Sruoga", "Dievų miškas", "Dievų takais"},

{"Salomėja Nėris", "Anksti rytą", "Pėdos smėly", "Laumės dovanos"},

{"Kazys Binkis", "Eilėraščiai", "Atžalynas", "Meškeriotojas",

"100 pavasarių"},

{"Vincas Krėvė", "Skirgaila", "Skerdžius", "Miglose"}

};

private String author = "";

private String title = "";

private int year = -1;

private double price = -1.0;

public Book build() {

return new Book(this);

}

public Book buildRandom() {

int ma = RANDOM.nextInt(TITLES.length); // pavadinimo indeksas 0..

int mo = RANDOM.nextInt(TITLES[ma].length - 1) + 1;// autoriaus indeksas 1..

return new Book(TITLES[ma][0],

TITLES[ma][mo],

1990 + RANDOM.nextInt(20),// metai tarp 1990 ir 2009

800 + RANDOM.nextDouble() \* 88000);// kaina tarp 800 ir 88800

}

public Builder year(int year) {

this.year = year;

return this;

}

public Builder author(String author) {

this.author = author;

return this;

}

public Builder title(String title) {

this.title = title;

return this;

}

public Builder price(double price) {

this.price = price;

return this;

}

}

}

1. Sudarykite individualiai pasirinktų elementų panaudojimo klasės testus su skirtingais duomenų rinkiniais.

BooksGenerator.java:

package edu.ktu.ds.lab3.vaitkevicius;

import edu.ktu.ds.lab3.gui.ValidationException;

import java.util.Arrays;

import java.util.Collections;

import java.util.stream.IntStream;

public class BooksGenerator {

private static final String ID\_CODE = "TA"; // \*\*\*\*\* Nauja

private static int serNr = 10000; // \*\*\*\*\* Nauja

private Book[] books;

private String[] keys;

private int currentBookIndex = 0, currentBookIdIndex = 0;

public static Book[] generateShuffleBooks(int size) {

Book[] books = IntStream.range(0, size)

.mapToObj(i -> new Book.Builder().buildRandom())

.toArray(Book[]::new);

Collections.shuffle(Arrays.asList(books));

return books;

}

public static String[] generateShuffleIds(int size) {

String[] keys = IntStream.range(0, size)

.mapToObj(i -> ID\_CODE + (serNr++))

.toArray(String[]::new);

Collections.shuffle(Arrays.asList(keys));

return keys;

}

public Book[] generateShuffleBooksAndIds(int setSize,

int setTakeSize) throws ValidationException {

if (setTakeSize > setSize) {

setTakeSize = setSize;

}

books = generateShuffleBooks(setSize);

keys = generateShuffleIds(setSize);

this.currentBookIndex = setTakeSize;

return Arrays.copyOf(books, setTakeSize);

}

// Imamas po vienas elementas iš sugeneruoto masyvo. Kai elementai baigiasi sugeneruojama

// nuosava situacija ir išmetamas pranešimas.

public Book getBook() {

if (books == null) {

throw new ValidationException("booksNotGenerated");

}

if (currentBookIndex < books.length) {

return books[currentBookIndex++];

} else {

throw new ValidationException("allSetStoredToMap");

}

}

public String getBookId() {

if (keys == null) {

throw new ValidationException("booksIdsNotGenerated");

}

if (currentBookIdIndex < keys.length) {

return keys[currentBookIdIndex++];

} else {

throw new ValidationException("allKeysStoredToMap");

}

}

}

ManualTest.java

package edu.ktu.ds.lab3.vaitkevicius;

import edu.ktu.ds.lab3.utils.HashType;

import edu.ktu.ds.lab3.utils.Ks;

import edu.ktu.ds.lab3.utils.ParsableHashMap;

import edu.ktu.ds.lab3.utils.ParsableMap;

import java.util.Locale;

public class ManualTest {

public static void main(String[] args) {

Locale.setDefault(Locale.US); // suvienodiname skaičių formatus

executeTest();

}

public static void executeTest() {

Book c1 = new Book("AAAuotirus", "PAVAADINIMAS", 1997, 1700);

Book c2 = new Book.Builder().buildRandom();

Book c3 = new Book("Gendaris Lunara 2001 700");

Book c4 = new Book("Ziusepe France 1946 9500");

Book c5 = new Book("Razda Grande 2001 80.3");

Book c6 = new Book.Builder().buildRandom();

Book c7 = new Book.Builder().buildRandom();

// Raktų masyvas

String[] booksIds = {"TA156", "TA102", "TA178", "TA171", "TA105", "TA106", "TA107", "TA108"};

int id = 0;

ParsableMap<String, Book> booksMap

= new ParsableHashMap<>(String::new, Book::new, HashType.DIVISION);

// Reikšmių masyvas

Book[] books = {c1, c2, c3, c4, c5, c6, c7};

for (Book c : books) {

booksMap.put(booksIds[id++], c);

}

booksMap.println("Porų išsidėstymas atvaizdyje pagal raktus");

Ks.oun("Ar egzistuoja pora atvaizdyje?");

Ks.oun(booksMap.contains(booksIds[6]));

Ks.oun(booksMap.contains(booksIds[7]));

Ks.oun("Pašalinamos poros iš atvaizdžio:");

Ks.oun(booksMap.remove(booksIds[1]));

Ks.oun(booksMap.remove(booksIds[7]));

booksMap.println("Porų išsidėstymas atvaizdyje pagal raktus");

Ks.oun("Atliekame porų paiešką atvaizdyje:");

Ks.oun(booksMap.get(booksIds[2]));

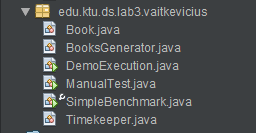
Ks.oun(booksMap.get(booksIds[7]));

Ks.oun("Išspausdiname atvaizdžio poras String eilute:");

Ks.ounn(booksMap);

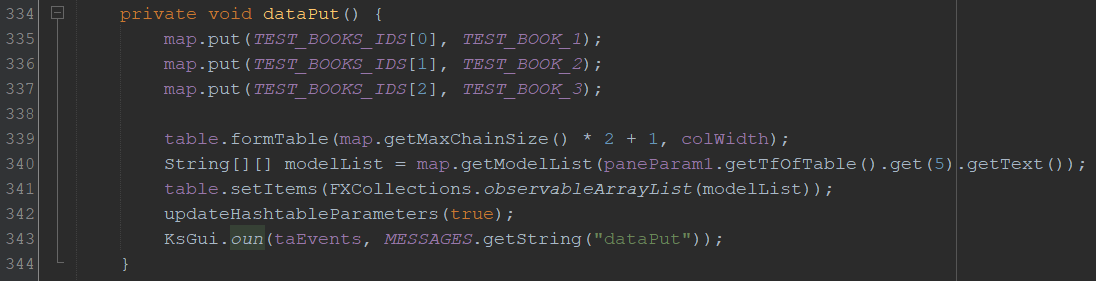
}

}

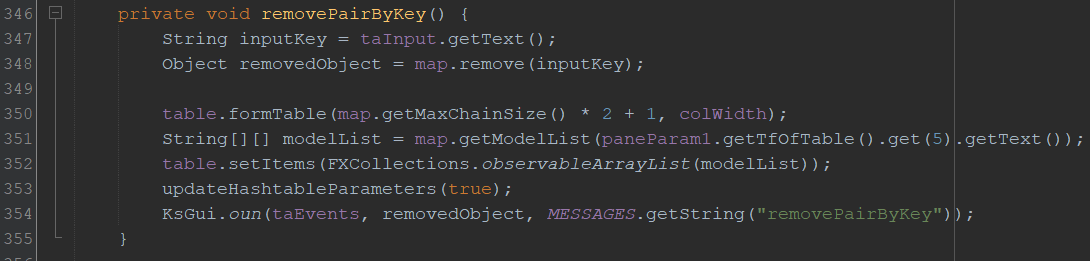


1. Pakete **edu.ktu.ds.lab3.pavarde** sudarykite individualių elementų atvaizdžių panaudojimo klasę, kurioje būtų atvaizdžio formavimas, poros šalinimas atvaizdyje, raktą atitinkančios reikšmės paieška ir pan., panaudojant klasę **edu.ktu.ds.lab3.utils.HashMap** (3 metodai). Sukurtų metodų veikimą demonstruokite pateiktoje grafinėje JavaFX aplinkoje arba sukurkite nuosavą, pasinaudodami paskaitų medžiaga.

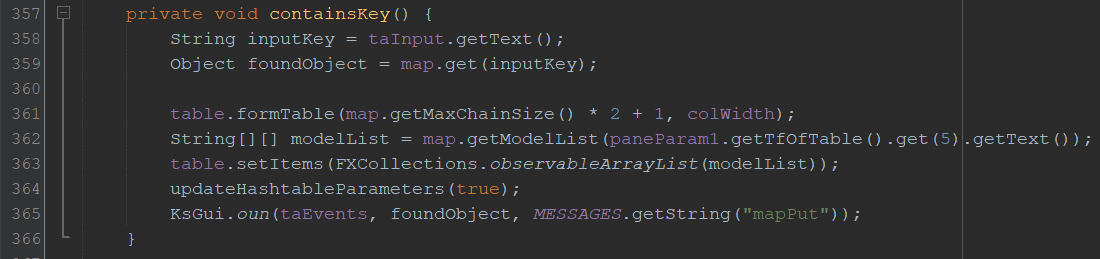
Duomenų idėjimas:

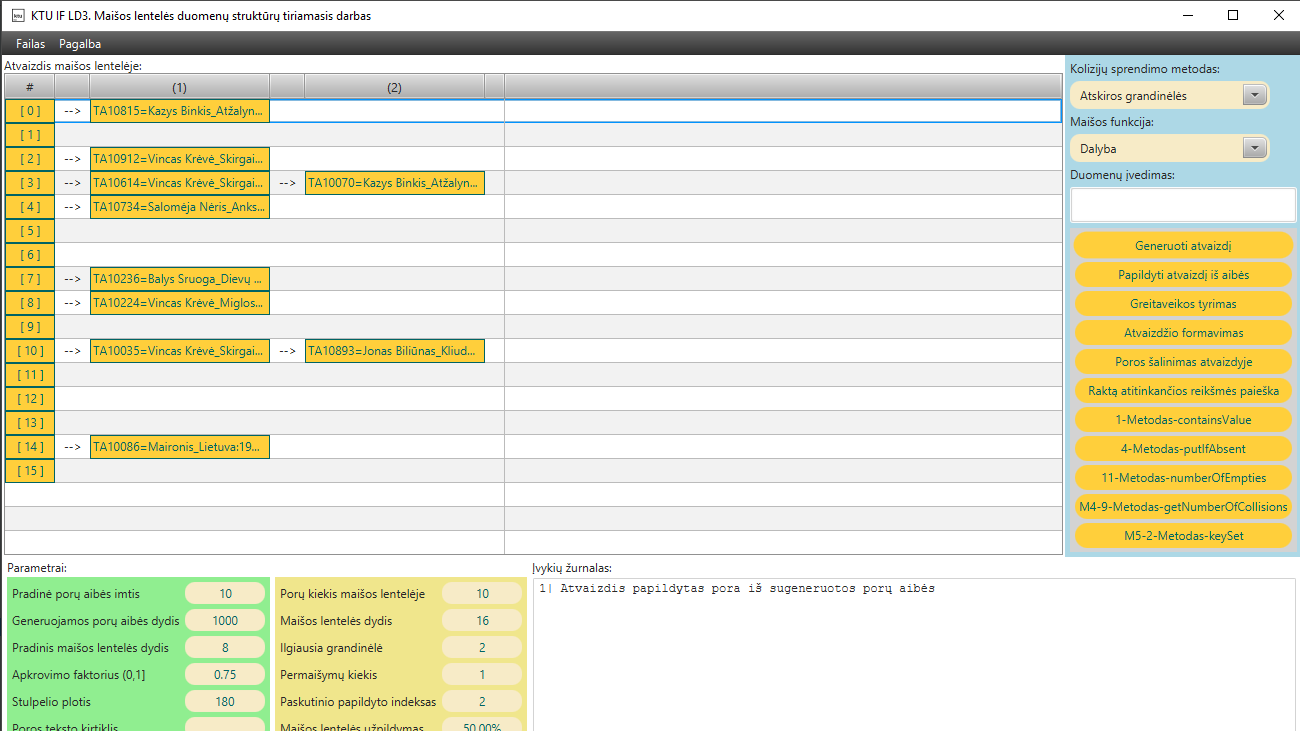


Poros šalinimas atvaizdyje pagal raktą:



Raktą atitinkančios reikšmės paieška:





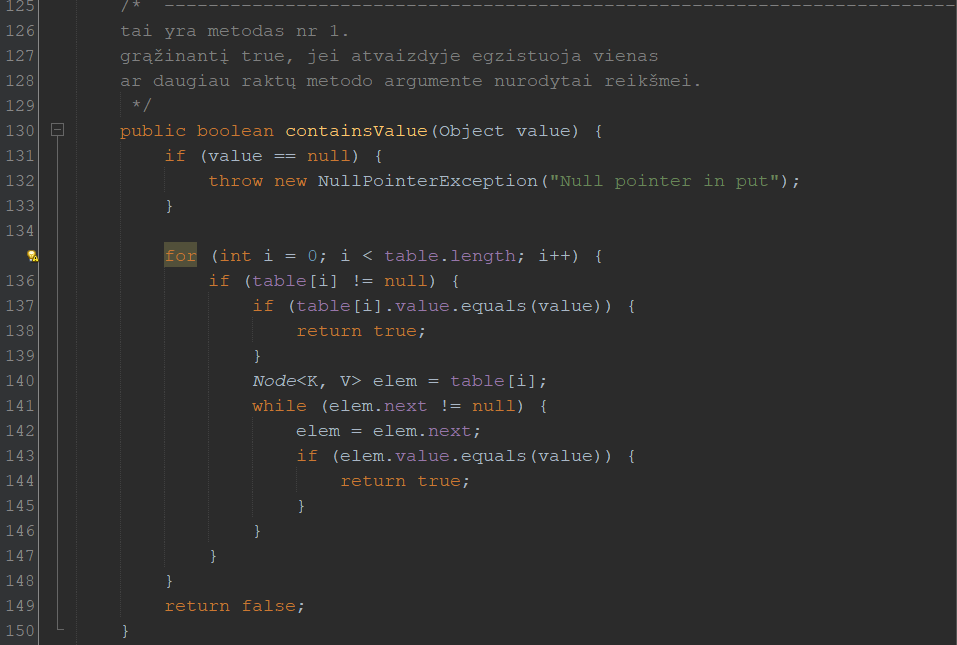
4.Klasėje **edu.ktu.ds.lab3.utils.HashMap** realizuokite metodus (1, 4, 11 metodai):

a) ***boolean containsValue(Object value)***, grąžinantį true, jei atvaizdyje egzistuoja vienas ar daugiau raktų metodo argumente nurodytai reikšmei.

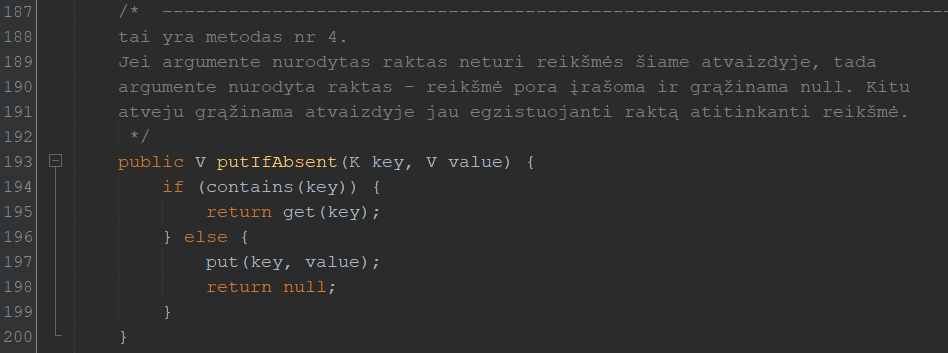
b) ***V putIfAbsent(K key, V value)***. Jei argumente nurodytas raktas neturi reikšmės šiame atvaizdyje, tada argumente nurodyta raktas - reikšmė pora įrašoma ir grąžinama null. Kitu atveju grąžinama atvaizdyje jau egzistuojanti raktą atitinkanti reikšmė.

c) ***int numberOfEmpties***, grąžinantį maišos lentelės masyvo tuščių elementų skaičių.

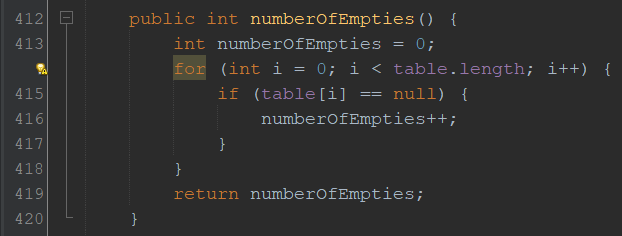
***boolean containsValue(Object value)***



***V putIfAbsent(K key, V value)***



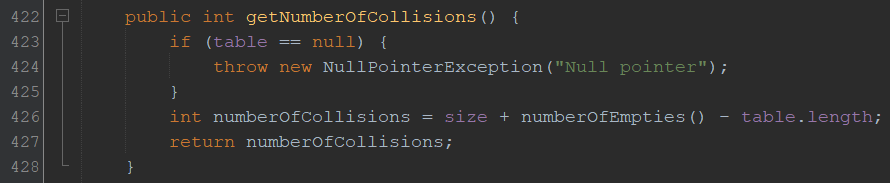
***int numberOfEmpties***



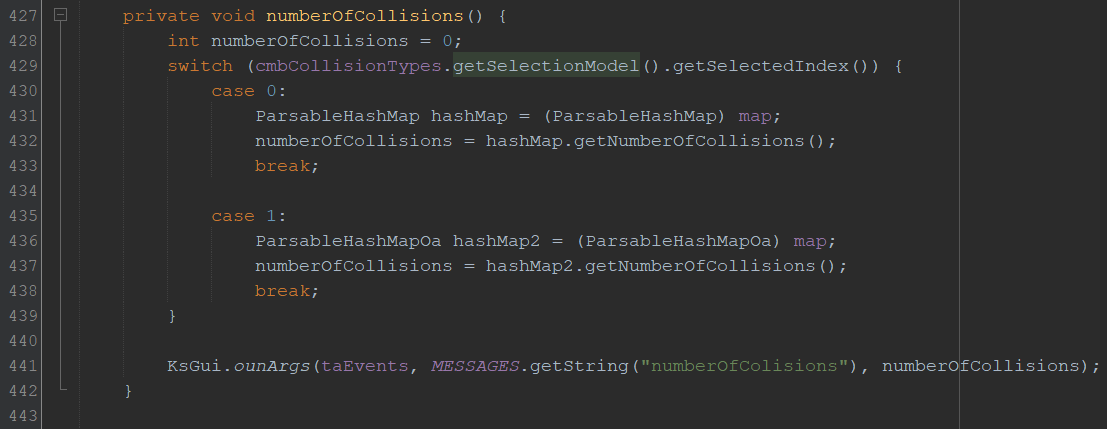
5.Klasėje **edu.ktu.ds.lab3.utils.HashMap** sukurkite individualiai pagal variantą nurodytus metodus (M4 ir M5 užduotys), atlikite jų rankinį testavimą klasėje **ManualTest**.



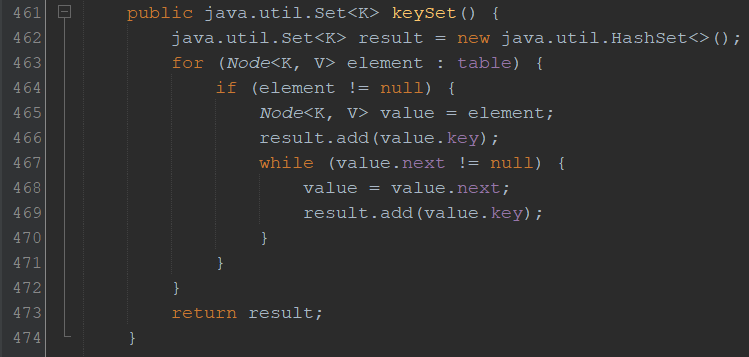
HashMap.java – 9 Metodas



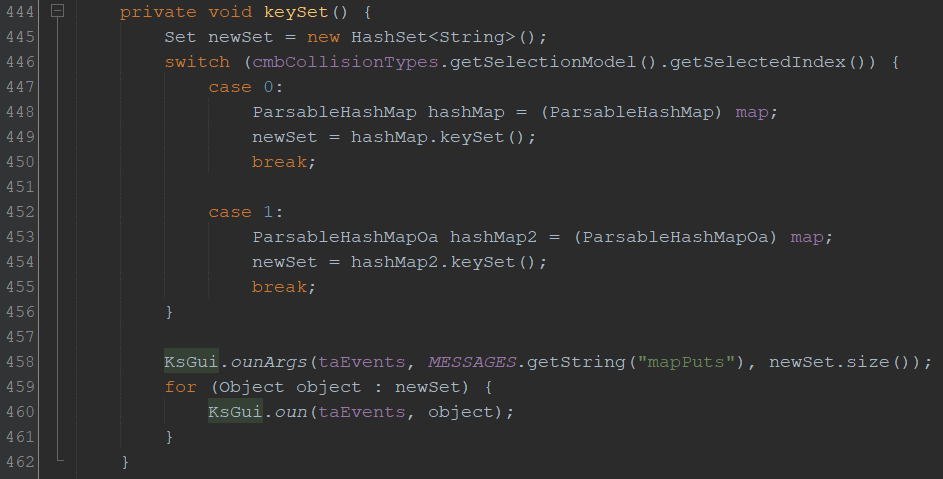
MainWindow.java



HashMap.java – 2 Metodas



MainWindow.java



6. Pakete **edu.ktu.ds.lab3.pavarde** sukurkite interfeisą **edu.ktu.ds.lab3.utils.Map** maišos lentele realizuojančią klasę **HashMapOa**, kolizijas sprendžiant individualiai pagal variantą nurodytu atviros adresacijos metodu: tiesinio dėstymo, kvadratinio dėstymo ar dvigubos maišos (M6 užduotis).



HashMapOa.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package edu.ktu.ds.lab3.utils;

import java.util.Arrays;

/\*\*

\*

\* @author NZXT-PC

\*/

public class HashMapOa<K, V> implements Map<K, V> {

public static final int DEFAULT\_INITIAL\_CAPACITY = 16;

public static final float DEFAULT\_LOAD\_FACTOR = 0.75f;

public static final HashType DEFAULT\_HASH\_TYPE = HashType.DIVISION;

// Maišos lentelė

protected Node<K, V>[] table;

// Lentelėje esančių raktas-reikšmė porų kiekis

protected int size = 0;

// Apkrovimo faktorius

protected float loadFactor;

// Maišos metodas

protected HashType ht;

//--------------------------------------------------------------------------

// Maišos lentelės įvertinimo parametrai

//--------------------------------------------------------------------------

// Permaišymų kiekis

protected int rehashesCounter = 0;

// Einamas poros indeksas maišos lentelėje

protected int index = 0;

/\* Klasėje sukurti 4 perkloti konstruktoriai, nustatantys atskirus maišos

\* lentelės parametrus. Jei kuris nors parametras nėra nustatomas -

\* priskiriama standartinė reikšmė.

\*/

public HashMapOa() {

this(DEFAULT\_HASH\_TYPE);

}

public HashMapOa(HashType ht) {

this(DEFAULT\_INITIAL\_CAPACITY, ht);

}

public HashMapOa(int initialCapacity, HashType ht) {

this(initialCapacity, DEFAULT\_LOAD\_FACTOR, ht);

}

public HashMapOa(float loadFactor, HashType ht) {

this(DEFAULT\_INITIAL\_CAPACITY, loadFactor, ht);

}

public HashMapOa(int initialCapacity, float loadFactor, HashType ht) {

if (initialCapacity <= 0) {

throw new IllegalArgumentException("Illegal initial capacity: " + initialCapacity);

}

if ((loadFactor <= 0.0) || (loadFactor > 1.0)) {

throw new IllegalArgumentException("Illegal load factor: " + loadFactor);

}

this.table = new Node[initialCapacity];

this.loadFactor = loadFactor;

this.ht = ht;

}

@Override

public boolean isEmpty() {

return size == 0;

}

@Override

public int size() {

return size;

}

@Override

public void clear() {

Arrays.fill(table, null);

size = 0;

index = 0;

rehashesCounter = 0;

}

@Override

public String[][] toArray() {

String[][] result = new String[table.length][];

int count = 0;

for (Node<K, V> n : table) {

String[] list = new String[1];

list[0] = n.toString();

result[count] = list;

count++;

}

return result;

}

@Override

public String toString() {

StringBuilder result = new StringBuilder();

for (Node<K, V> node : table) {

if (node != null) {

result.append(node.toString()).append(System.lineSeparator());

}

}

return result.toString();

}

private int findPosition(K key) {

int index = hash(key, ht);

int index0 = index;

for (int i = 0; i < table.length; i++) {

if (table[index] == null || table[index].key.equals(key)) {

// if (table[index] == null || table[index].key == key) {

return index;

}

// index = (index0 + i + 1) % table.length;

index = (index0 + i) % table.length;

}

return -1;

}

@Override

public V put(K key, V value) {

if (key == null || value == null) {

throw new IllegalArgumentException("Key or value is null in put(Key key, Value value)");

}

int index = findPosition(key);

if (index == -1) {

rehash();

put(key, value);

return value;

}

if (table[index] == null) {

table[index] = new Node<K, V>(key, value);

size++;

if (size > table.length \* loadFactor) {

rehash();

}

} else {

table[index].value = value;

}

return value;

}

/\* -----------------------------------------------------------------------------------------

tai yra metodas nr 4.

Jei argumente nurodytas raktas neturi reikšmės šiame atvaizdyje, tada

argumente nurodyta raktas - reikšmė pora įrašoma ir grąžinama null. Kitu

atveju grąžinama atvaizdyje jau egzistuojanti raktą atitinkanti reikšmė.

\*/

public V putIfAbsent(K key, V value) {

if (contains(key)) {

return get(key);

} else {

put(key, value);

return null;

}

}

public double averageChainSize() {

if (table == null) {

throw new NullPointerException("Null pointer");

}

return size \* 1.0 / (table.length - numberOfEmpties());

}

@Override

public V get(K key) {

if (key == null) {

throw new IllegalArgumentException("Key is null in get(Key key)");

}

index = findPosition(key);

return (table[index] != null) ? table[index].value : null;

}

@Override

public V remove(K key) {

if (key == null) {

throw new NullPointerException("Null pointer in remove");

}

int index = findPosition(key);

if (table[index] != null) {

V value = table[index].value;

table[index] = null;

size--;

return value;

} else {

return null;

}

}

@Override

public boolean contains(K key) {

return get(key) != null;

}

/\* -----------------------------------------------------------------------------------------

tai yra metodas nr 1.

grąžinantį true, jei atvaizdyje egzistuoja vienas

ar daugiau raktų metodo argumente nurodytai reikšmei.

\*/

public boolean containsValue(Object value) {

if (value == null) {

throw new NullPointerException("Null pointer in put");

}

for (int i = 0; i < table.length; i++) {

if (table[i] != null) {

if (table[i].value.equals(value)) {

return true;

}

}

}

return false;

}

/\*\*

\* Permaišymas

\*

\* @param node

\*/

private void rehash() {

HashMapOa<K, V> newMap = new HashMapOa<>(table.length \* 2, loadFactor, ht);

for (int i = 0; i < table.length; i++) {

if (table[i] != null) {

newMap.put(table[i].key, table[i].value);

}

}

table = newMap.table;

rehashesCounter++;

}

private int hash(K key, HashType hashType) {

int h = key.hashCode();

switch (hashType) {

case DIVISION:

return Math.abs(h) % table.length;

case MULTIPLICATION:

double k = (Math.sqrt(5) - 1) / 2;

return (int) (((k \* Math.abs(h)) % 1) \* table.length);

case JCF7:

h ^= (h >>> 20) ^ (h >>> 12);

h = h ^ (h >>> 7) ^ (h >>> 4);

return h & (table.length - 1);

case JCF8:

h = h ^ (h >>> 16);

return h & (table.length - 1);

default:

return Math.abs(h) % table.length;

}

}

/\*\*

\* Grąžina formuojant maišos lentelę įvykusių permaišymų kiekį.

\*

\* @return Permaišymų kiekis.

\*/

public int getRehashesCounter() {

return rehashesCounter;

}

/\*\*

\* Grąžina maišos lentelės talpą.

\*

\* @return Maišos lentelės talpa.

\*/

public int getTableCapacity() {

return table.length;

}

public int numberOfEmpties() {

int numberOfEmpties = 0;

for (int i = 0; i < table.length; i++) {

if (table[i] == null) {

numberOfEmpties++;

}

}

return numberOfEmpties;

}

public int getNumberOfCollisions() {

if (table == null) {

throw new NullPointerException("Null pointer");

}

int numberOfCollisions = size + numberOfEmpties() - table.length;

return numberOfCollisions;

}

public java.util.Set<K> keySet() {

java.util.Set<K> test = new java.util.HashSet<>();

for (Node<K, V> element : table) {

if (element != null) {

Node<K, V> value = element;

test.add(value.key);

}

}

return test;

}

protected static class Node<K, V> {

// Raktas

protected K key;

// Reikšmė

protected V value;

// Rodyklė į sekantį grandinėlės mazgą

protected Node() {

}

protected Node(K key, V value) {

this.key = key;

this.value = value;

}

@Override

public String toString() {

return key + "=" + value;

}

}

}

ParsableHashMapOa.java

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package edu.ktu.ds.lab3.utils;

import static edu.ktu.ds.lab3.utils.HashMapOa.DEFAULT\_HASH\_TYPE;

import static edu.ktu.ds.lab3.utils.HashMapOa.DEFAULT\_INITIAL\_CAPACITY;

import static edu.ktu.ds.lab3.utils.HashMapOa.DEFAULT\_LOAD\_FACTOR;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.io.UncheckedIOException;

import java.nio.charset.Charset;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

import java.util.function.Function;

/\*\*

\*

\* @author husky

\*/

public class ParsableHashMapOa<K, V extends Parsable<V>> extends HashMapOa<K, V> implements ParsableMap<K, V> {

private final Function<String, K> keyCreateFunction; // funkcija bazinio rakto objekto kūrimui

private final Function<String, V> valueCreateFunction; // funkcija bazinio reikšmės objekto kūrimui

/\*\*

\* Konstruktorius su funkcija bazinių rakto ir reikšmės objektų kūrimui

\*

\* @param keyCreateFunction

\* @param valueCreateFunction

\*/

public ParsableHashMapOa(Function<String, K> keyCreateFunction,

Function<String, V> valueCreateFunction) {

this(keyCreateFunction, valueCreateFunction, DEFAULT\_HASH\_TYPE);

}

/\*\*

\* Konstruktorius su funkcija bazinių rakto ir reikšmės objektų kūrimui

\*

\* @param keyCreateFunction

\* @param valueCreateFunction

\* @param ht

\*/

public ParsableHashMapOa(Function<String, K> keyCreateFunction,

Function<String, V> valueCreateFunction,

HashType ht) {

this(keyCreateFunction, valueCreateFunction, DEFAULT\_INITIAL\_CAPACITY, ht);

}

/\*\*

\* Konstruktorius su funkcija bazinių rakto ir reikšmės objektų kūrimui

\*

\* @param keyCreateFunction

\* @param valueCreateFunction

\* @param initialCapacity

\* @param ht

\*/

public ParsableHashMapOa(Function<String, K> keyCreateFunction,

Function<String, V> valueCreateFunction,

int initialCapacity,

HashType ht) {

this(keyCreateFunction, valueCreateFunction, initialCapacity, DEFAULT\_LOAD\_FACTOR, ht);

}

/\*\*

\* Konstruktorius su funkcija bazinių rakto ir reikšmės objektų kūrimui

\*

\* @param keyCreateFunction

\* @param valueCreateFunction

\* @param initialCapacity

\* @param loadFactor

\* @param ht

\*/

public ParsableHashMapOa(Function<String, K> keyCreateFunction,

Function<String, V> valueCreateFunction,

int initialCapacity,

float loadFactor,

HashType ht) {

super(initialCapacity, loadFactor, ht);

this.keyCreateFunction = keyCreateFunction;

this.valueCreateFunction = valueCreateFunction;

}

@Override

public V put(String dataString) {

return super.put(

create(keyCreateFunction, dataString, "Nenustatyta raktų kūrimo funkcija"),

create(valueCreateFunction, dataString, "Nenustatyta reikšmių kūrimo funkcija")

);

}

@Override

public void load(String filePath) {

if (filePath == null || filePath.length() == 0) {

return;

}

clear();

try (BufferedReader fReader = Files.newBufferedReader(Paths.get(filePath), Charset.defaultCharset())) {

fReader.lines()

.map(String::trim)

.filter(line -> !line.isEmpty())

.forEach(this::put);

} catch (FileNotFoundException e) {

Ks.ern("Tinkamas duomenų failas nerastas: " + e.getLocalizedMessage());

} catch (IOException | UncheckedIOException e) {

Ks.ern("Failo skaitymo klaida: " + e.getLocalizedMessage());

}

}

@Override

public void save(String filePath) {

throw new UnsupportedOperationException("Saugojimas.. nepalaikomas");

}

@Override

public void println() {

if (super.isEmpty()) {

Ks.oun("Atvaizdis yra tuščias");

} else {

String[][] data = getModelList("=");

for (int i = 0; i < data.length; i++) {

for (int j = 0; j < data[i].length; j++) {

String format = (j == 0 | j % 2 == 1) ? "%7s" : "%15s";

Object value = data[i][j];

Ks.ouf(format, (value == null ? "" : value));

}

Ks.oufln("");

}

}

Ks.oufln("\*\*\*\*\*\* Bendras porų kiekis yra " + super.size());

}

@Override

public void println(String title) {

Ks.ounn("========" + title + "=======");

println();

Ks.ounn("======== Atvaizdžio pabaiga =======");

}

@Override

public String[][] getModelList(String delimiter) {

String[][] result = new String[table.length][];

int count = 0;

for (HashMapOa.Node<K, V> n : table) {

List<String> list = new ArrayList<>();

list.add("[ " + count + " ]");

if (n != null) {

list.add("-->");

list.add(split(n.toString(), delimiter));

// list.add(n.toString());

}

result[count] = list.toArray(new String[0]);

count++;

}

return result;

}

private String split(String s, String delimiter) {

int k = s.indexOf(delimiter);

if (k <= 0) {

return s;

}

return s.substring(0, k);

}

@Override

public int getMaxChainSize() {

return 1;

}

@Override

public int getLastUpdatedChain() {

return -1;

}

@Override

public int getChainsCounter() {

return size;

}

private static <T, R> R create(Function<T, R> function, T data, String errorMessage) {

return Optional.ofNullable(function)

.map(f -> f.apply(data))

.orElseThrow(() -> new IllegalStateException(errorMessage));

}

}

7. Atlikite pagal variantą nurodytą greitaveikos tyrimą (G1, G2 užduotis), sudarykite vykdymo laikų grafikus ir atlikite rezultatų analizę. Tai atlikite su Jūsų individualios klasės atvaizdžių poromis ir su faile <lab3\_projekto\_direktorija>/data/zodynas.txt esančiais žodžiais (tyrimui užtenka, kad raktas ir reikšmė būtų tas pats žodis).



SimpleBenchmark.java

package edu.ktu.ds.lab3.vaitkevicius;

import edu.ktu.ds.lab3.gui.ValidationException;

import edu.ktu.ds.lab3.utils.HashType;

import edu.ktu.ds.lab3.utils.Ks;

import edu.ktu.ds.lab3.utils.ParsableHashMap;

import edu.ktu.ds.lab3.utils.HashMapOa;

import edu.ktu.ds.lab3.utils.ParsableHashMapOa;

import java.io.FileNotFoundException;

import java.io.RandomAccessFile;

import java.util.\*;

import java.util.concurrent.BlockingQueue;

import java.util.concurrent.Semaphore;

import java.util.HashMap;

import java.util.concurrent.ThreadLocalRandom;

/\*\*

\* @author eimutis

\*/

public class SimpleBenchmark {

public static final String FINISH\_COMMAND = " ";

private static final ResourceBundle MESSAGES = ResourceBundle.getBundle("edu.ktu.ds.lab3.gui.messages");

private final Timekeeper timekeeper;

private final String[] BENCHMARK\_NAMES = {"Put", "BookRemKTU", "BookRemJava",

"DictionaryRemKTU", "DictionaryRemJava"};

private final int[] COUNTS = {10000, 20000, 40000, 80000};

private final ParsableHashMap<String, Book> hashMapPut

= new ParsableHashMap<>(String::new, Book::new, 10, 0.75f, HashType.DIVISION);

private final edu.ktu.ds.lab3.utils.HashMap<String, Book> hashMapBookRemoveKtu

= new edu.ktu.ds.lab3.utils.HashMap<>(10, 0.75f, HashType.DIVISION);

private final HashMap<String, Book> hashMapBookRemoveJava = new HashMap<>();

private final edu.ktu.ds.lab3.utils.HashMap<String, String> hashMapDictionaryRemoveKtu

= new edu.ktu.ds.lab3.utils.HashMap<>(10, 0.75f, HashType.DIVISION);

private final HashMap<String, String> hashMapDictionaryRemoveJava

= new HashMap<String, String>();

private final Queue<String> chainsSizes = new LinkedList<>();

/\*\*

\* For console benchmark

\*/

public SimpleBenchmark() {

timekeeper = new Timekeeper(COUNTS);

}

/\*\*

\* For Gui benchmark

\*

\* @param resultsLogger

\* @param semaphore

\*/

public SimpleBenchmark(BlockingQueue<String> resultsLogger, Semaphore semaphore) {

semaphore.release();

timekeeper = new Timekeeper(COUNTS, resultsLogger, semaphore);

}

public static void main(String[] args) {

executeTest();

}

public static void executeTest() {

// suvienodiname skaičių formatus pagal LT lokalę (10-ainis kablelis)

Locale.setDefault(new Locale("LT"));

Ks.out("Greitaveikos tyrimas:\n");

new SimpleBenchmark().startBenchmark();

}

public void startBenchmark() {

try {

benchmark();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

} catch (Exception ex) {

ex.printStackTrace(System.out);

}

}

public void benchmark() throws InterruptedException, FileNotFoundException {

try {

chainsSizes.add(MESSAGES.getString("maxChainLength"));

chainsSizes.add(" kiekis " + BENCHMARK\_NAMES[0] + " " + BENCHMARK\_NAMES[1]);

for (int k : COUNTS) {

Book[] bookArray = BooksGenerator.generateShuffleBooks(k);

String[] bookArray2 = BooksGenerator.generateShuffleIds(k);

Book[] booksSearchArray = bookArray;

String[] booksSearchIdsArray = bookArray2;

shuffleProc(booksSearchArray);

shuffleProc(booksSearchIdsArray);

String[] wordStringArray = new String[k];

RandomAccessFile dictionaryFile = new RandomAccessFile

("data\\zodynas.txt", "r");

try {

for (int i = 0; i < k; i++) {

wordStringArray[i] = dictionaryFile.readLine();

hashMapDictionaryRemoveKtu.put(wordStringArray[i], wordStringArray[i]);

hashMapDictionaryRemoveJava.put(wordStringArray[i], wordStringArray[i]);

}

} catch (Exception e) {

e.printStackTrace();

}

shuffleProc(wordStringArray);

hashMapBookRemoveKtu.clear();

hashMapBookRemoveJava.clear();

timekeeper.startAfterPause();

timekeeper.start();

for (int i = 0; i < k; i++) {

hashMapBookRemoveKtu.put(bookArray2[i], bookArray[i]);

hashMapBookRemoveJava.put(bookArray2[i], bookArray[i]);

}

timekeeper.finish(BENCHMARK\_NAMES[0]);

for (int i = 0; i < k; i++) {

hashMapBookRemoveKtu.remove(booksSearchIdsArray[i]);

}

timekeeper.finish(BENCHMARK\_NAMES[1]);

for (int i = 0; i < k; i++) {

hashMapBookRemoveJava.remove(booksSearchIdsArray[i]);

}

timekeeper.finish(BENCHMARK\_NAMES[2]);

for (int i = 0; i < k; i++) {

hashMapDictionaryRemoveKtu.remove(wordStringArray[i]);

}

timekeeper.finish(BENCHMARK\_NAMES[3]);

for (int i = 0; i < k; i++) {

hashMapBookRemoveJava.remove(wordStringArray[i]);

}

timekeeper.finish(BENCHMARK\_NAMES[4]);

timekeeper.seriesFinish();

}

StringBuilder sb = new StringBuilder();

chainsSizes.forEach(p -> sb.append(p).append(System.lineSeparator()));

timekeeper.logResult(sb.toString());

timekeeper.logResult(FINISH\_COMMAND);

} catch (ValidationException e) {

timekeeper.logResult(e.getMessage());

}

}

static <Type> void shuffleProc(Type[] ar) {

Random rnd = ThreadLocalRandom.current();

for (int i = ar.length - 1; i > 0; i--) {

int index = rnd.nextInt(i + 1);

// Simple swap

Type a = ar[index];

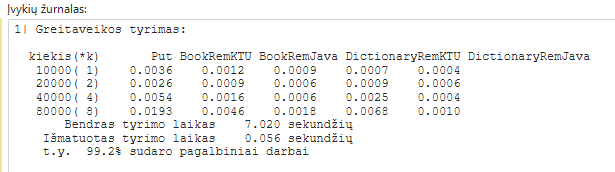
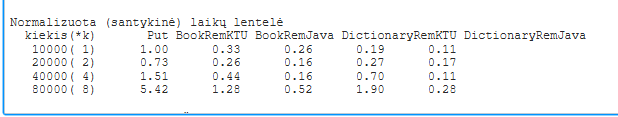
ar[index] = ar[i];

ar[i] = a;

}

}

}

# **Išvados**

Atliekant trečiąją laboratorinio darbo dalį išmokau maišos lentelių tipų duomenų struktūras. Išmokau atvaizdžio ADT realizacijos maišos lentelę, supratau kaip spręsti kolizijas pagal tiesinio dėstymo adresacijos metodą ir įtvirtinau grafinės vartotojo sąsajos programavimo žinias.