**ЗБИРКА ЛАБОРАТОРИСКИ ЗАДАЧИ – НП**

Лаб 1

Да се напише класа која чува низа на цели броеви IntegerArray. Класата треба да е **immutable**. Тоа значи дека, откако еднаш ќе се инстанцира да не може да се менува состојбата на објектот, односно да не може да се менуваат податоците зачувани во него и да не може да се наследува од неа final. За потребите на класата треба да се имплементираат следните public методи:

* IntegerArray(int a[]) - конструктор кој прима низа од цели броеви
* length():int - метод кој ја враќа должината на низата
* getElementAt(int i):int - го враќа елементот на позиција i, може да претпоставите дека индекост i секогаш ќе има вредност во интервалот [0,length()-1]
* sum():int - метод кој ја враќа сумата на сите елемeнти во низата
* average():double - метод кој ја враќа средната вредност на елементите во низата - аритметичка средина
* getSorted():IntegerArray- враќа нов објект од истата класа кој ги содржи истите вредности од тековниот објект, но сортирани во растечки редослед
* concat(IntegerArray ia):IntegerArray - враќа нов објект од истата класа во кој се содржат сите елементи од this објектот и по нив сите елементи од ia објектот притоа запазувајќи го нивниот редослед
* toString():String - враќа текстуална репрезентација на објектот каде елементите се одделени со запиркa и едно празно место после запирката и на почетокот и крајот на стрингот има средни загради. Пример за низа која ги содржи боревите 2,1 и 4 враќа "[2, 1, 4]".

Покрај класата IntegerArray треба да напишете дополнително уште една класа која ќе служи за вчитување на низа од цели броеви од влезен тек на податоци. Оваа класа треба да се вика ArrayReader и во неа треба да имате еден public static метод за вчитување на низа од цели броеви од InputStream.

* readIntegerArray(InputStream input):IntegerArray - вчитува низа од цели броеви од input зададена во следниот формат: Во првата линија има еден цел борј кој кажува колку елементи има во низата, а во наредниот ред се дадени елементите на низата одделени со едно или повеќе празни места. Помош, искористете ја класата java.util.Scanner.

Секогаш кога работите со низи во Java можете да искористите дел од методите во класата Arrays. За да пристапите до класата најпрвин треба да ја импортирате со следнава наредба

import java.util.Arrays;

Во продолжение се дадени дел од методите кои можат да ви помогнат. За тоа како работат консултираје ја нивната документација.

[copyOf(int[] original, int newLength)](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#copyOf%28int%5B%5D,%20int%29)

[equals(int[] a, int[] a2)](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#equals%28int%5B%5D,%20int%5B%5D%29)

[sort(int[] a)](http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#sort%28int%5B%5D%29)

| **Input** | **Result** |
| --- | --- |
| testImmutability  14598 | 6  14 9 9 7 10 10  [14, 9, 9, 7, 10, 10]  59  9.83  6  14 9 9 7 10 10  [14, 9, 9, 7, 10, 10]  59  9.83  6  14 9 9 7 10 10  [14, 9, 9, 7, 10, 10]  59  9.83  6  7 9 9 10 10 14  [7, 9, 9, 10, 10, 14]  59  9.83  6  7 9 9 10 10 14  [7, 9, 9, 10, 10, 14]  59  9.83  6  14 9 9 7 10 10  [14, 9, 9, 7, 10, 10]  59  9.83  6  14 9 9 7 10 10  [14, 9, 9, 7, 10, 10]  59  9.83  6  14 9 9 7 10 10  [14, 9, 9, 7, 10, 10]  59  9.83 |
| testSorting  5  8 12 3 66 17 | [3, 8, 12, 17, 66] |

import java.io.ByteArrayInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.util.Arrays;

import java.util.Random;

import java.util.Scanner;

public class IntegerArrayTester {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String s = scanner.nextLine();

IntegerArray ia = null;

switch (s) {

case "testSimpleMethods":

ia = new IntegerArray(generateRandomArray(scanner.nextInt()));

testSimpleMethods(ia);

break;

case "testConcat":

testConcat(scanner);

break;

case "testEquals":

testEquals(scanner);

break;

case "testSorting":

testSorting(scanner);

break;

case "testReading":

testReading(new ByteArrayInputStream(scanner.nextLine().getBytes()));

break;

case "testImmutability":

int a[] = generateRandomArray(scanner.nextInt());

ia = new IntegerArray(a);

testSimpleMethods(ia);

testSimpleMethods(ia);

IntegerArray sorted\_ia = ia.getSorted();

testSimpleMethods(ia);

testSimpleMethods(sorted\_ia);

sorted\_ia.getSorted();

testSimpleMethods(sorted\_ia);

testSimpleMethods(ia);

a[0] += 2;

testSimpleMethods(ia);

ia = ArrayReader.readIntegerArray(new ByteArrayInputStream(integerArrayToString(ia).getBytes()));

testSimpleMethods(ia);

break;

}

scanner.close();

}

static void testReading(InputStream in) {

IntegerArray read = ArrayReader.readIntegerArray(in);

System.out.println(read);

}

static void testSorting(Scanner scanner) {

int[] a = readArray(scanner);

IntegerArray ia = new IntegerArray(a);

System.out.println(ia.getSorted());

}

static void testEquals(Scanner scanner) {

int[] a = readArray(scanner);

int[] b = readArray(scanner);

int[] c = readArray(scanner);

IntegerArray ia = new IntegerArray(a);

IntegerArray ib = new IntegerArray(b);

IntegerArray ic = new IntegerArray(c);

System.out.println(ia.equals(ib));

System.out.println(ia.equals(ic));

System.out.println(ib.equals(ic));

}

static void testConcat(Scanner scanner) {

int[] a = readArray(scanner);

int[] b = readArray(scanner);

IntegerArray array1 = new IntegerArray(a);

IntegerArray array2 = new IntegerArray(b);

IntegerArray concatenated = array1.concat(array2);

System.out.println(concatenated);

}

static void testSimpleMethods(IntegerArray ia) {

System.out.print(integerArrayToString(ia));

System.out.println(ia);

System.out.println(ia.sum());

System.out.printf("%.2f\n", ia.average());

}

static String integerArrayToString(IntegerArray ia) {

StringBuilder sb = new StringBuilder();

sb.append(ia.length()).append('\n');

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

sb.append(ia.getElementAt(i)).append(' ');

sb.append('\n');

return sb.toString();

}

static int[] readArray(Scanner scanner) {

int n = scanner.nextInt();

int[] a = new int[n];

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

a[i] = scanner.nextInt();

}

return a;

}

static int[] generateRandomArray(int k) {

Random rnd = new Random(k);

int n = rnd.nextInt(8) + 2;

int a[] = new int[n];

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

a[i] = rnd.nextInt(20) - 5;

}

return a;

}

}

**2.**

Да се напише метод кој ќе прима еден цел број и ќе ја печати неговата репрезентација како Римски број.

Пример

Aко ако се повика со парамететар 1998, излезот треба да биде MCMXCVIII.

import java.util.Scanner;

import java.util.stream.IntStream;

public class RomanConverterTest {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt();

IntStream.range(0, n)

.forEach(x -> System.out.println(RomanConverter.toRoman(scanner.nextInt())));

scanner.close();

}

}

class RomanConverter {

/\*\*

\* Roman to decimal converter

\*

\* @param n number in decimal format

\* @return string representation of the number in Roman numeral

\*/

public static String toRoman(int n) {

// your solution here

int[] vrednost = {1000,900,500,400,100,90,50,40,10,9,5,4,1};

String[] roman = {"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};

StringBuilder rom = new StringBuilder();

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

while (n >= vrednost[i]) { //746-500=246-100=146-100=46-40=6-5=1

n = n - vrednost[i]; //DCCXLVI

rom.append(roman[i]);

}

}

return rom.toString();

}

}

3. Треба да се креира апликација за банка која ќе управуваа со сметките на повеќе корисниците и ќе врши трансакции помеѓу нив. Банката работи само со долари и притоа сите суми на пари се претставуваат како стрингови со знакот $ на крај, и точка помеѓу бројот на центи и бројот на долари без празни места. Бројот на центи треба да се состои од две цифри без разлика на износот.

Пример:

10 долари 10.00$

15 долари и 50 центи 15.50$

За потребите на ваквата апликација треба да се напишат класите Account,Transaction и Bank. Класата Account претставува една сметка на еден корисник и треба да ги чува следните податоци:

* Име на корисникот,
* единствен идентификационен број (long)
* тековното салдо на сметката.

Оваа класа исто така треба да ги имплементира и следниве методи

* Account(String name, String balance) – конструктор со параметри (id-то треба да го генерирате сами со помош на класата java.util.Random)
* getBalance():String
* getName():String
* getId():long
* setBalance(String balance)
* toString():String – враќа стринг во следниот формат, \n означува нов ред
* Name:Andrej Gajduk\n
* Balance:20.00$\n

Класата Transaction претставува трансакција (префрлување пари од една на друга сметка), од страна на банката за што честопати се наплаќа провизија. За почеток треба да се напише класата Transaction со податочни членови за идентификационите броеви на две сметки, едната од која се одземаат парите и друга на која се додаваат парите, текстуален опис и износ на трансакцијата.

За оваа класа треба да ги имплементирате методите:

* Transaction(long fromId, long toId, Stirng description, String amount) – конструктор со параметри
* getAmount():String
* getFromId():long
* getToId():long

Оваа класа треба да е **immutable**, а можете и да ја направите и **апстрактна** бидејќи не е наменета директно да се користи туку само како основна класа за изведување на други класи.

Како што споменавме претходно банката наплаќа провизија за одредени трансакции. Има два типа на провизија, фискна сума и процент. Кај фиксна сума за било која трансакција без разлика на износот на трансакцијата се наплаќа исто провизија (пример 10$). Кај процент за секој еден долар од трансакцијата банката наплаќа одреден процент провизија (на пример 5%, или 5 центи на секој долар – процентите секогаш се целобројни и провизија се наплаќа само на цели долари).

За да се прави разлика меѓу различните типови на провизија, треба да напишете уште две класи кои ќе наследуваат од Transaction кои треба да ги именувате FlatAmountProvisionTransaction и FlatPercentProvisionTransaction.

Првата класа FlatAmountProvisionTransaction треба да содржи соодветен конструктор

* FlatAmountProvisionTransaction(long fromId, long toId,String amount, String flatProvision) кој го иницијализира полето за опис на "FlatAmount" и соодветен get метод
* getFlatAmount():String

Слично и класата FlatPercentProvisionTransaction треба да има соодветен конструктор

* FlatPercentProvisionTransaction (long fromId, long toId, String amount, int centsPerDolar) кој го иницијализира полето за опис на "FlatPercent" и соодветен get метод
* getPercent():int

Исто така треба да се преоптовари equals(Object o):boolean методот и за двете класи.

За крај треба да ја имплементирате класата Bank која ги чува сметките од своите корисници и дополнително врши трансакции. Класата освен сметките на своите корисници, треба да ги чува и сопственото име и вкупната сума на трансфери како и вкупната наплатена провизија од страна на банката за сите трансакции.

Класата Bank треба да ги нуди следните методи:

* Bank(String name, Account accounts[]) – конструктор со соодветните параметри (направете сопствена копија на низата од сметки)
* makeTransaction(Transaction t):boolean – врши проверка дали корисникот ги има потребните средства на сметка и дали и двете сметки на кои се однесува трансакцијата се нависитина во банката и ако и двата услови се исполнето ја извршува трансакцијата и враќа true, во спротивно враќа false
* totalTransfers():String – ја дава вкупната сума на пари кои се префрлени во сите трансакции до сега
* totalProvision():String – ја дава вкупната провизија наплатена од банката за сите извршени трансакции до сега
* toString():String - го враќа името на банката во посебна линија во формат
* Name:Banka na RM\n
* \n

по што следат податоците за сите корисници.

Провизијата се наплаќа така што на основната сума на трансакцијата се додава вредноста не провизијата и таа сума се одзема од првата сметка.

За сите класи да се напишат соодветни equals и hashCode методи.

| **Input** | **Result** |
| --- | --- |
| typical\_usage  Banka na RM  50  Alana Pagel  96124963.59$  Kathe Vath  104071827.99$  Sara Flenniken  103489869.36$  Shauna Irick  99230857.77$  Leon Brokaw  100949324.96$  Guillermina Haymond  101372350.52$  Sherie Pestana  100772483.38$  Shirlene Reichard  98451795.79$  Jerrica Kafka  100608943.69$  Nida Garg  96374183.01$  Ellamae Marshell  101834753.75$  Ranee Ricca  96386667.01$  Elois Philip  96767866.34$  Roland June  99245215.19$  Marvis Lemarr  95732058.60$  Burl Friese  100232415.35$  Reggie Olah  104652507.25$  Isis Ballard  100717280.66$  Bernardine Munford  104257015.83$  Johnette Greenhaw  103471815.46$  Olimpia Ryan  103239439.43$  Sha Duguay  96926529.60$  Donn Kurt  100902436.77$  Joel Cao  96522430.27$  Jamika Epstein  104102902.74$  Moira Haar  102396011.82$  Tiffani Bellantoni  98824115.92$  Tennille Hanzlik  101943340.74$  Dwayne Bazile  96723845.51$  Luigi Blaise  103007704.71$  Paris Fagan  100668747.81$  Ngan Raynor  100211914.45$  Alden Mirarchi  102954898.08$  Tangela Fernandez  99531915.08$  Liza Breaux  103806099.54$  Annice Lipman  97834836.56$  Broderick Arcand  102290753.08$  Terry Cygan  97565192.65$  Adriene Janz  104240885.04$  Joaquin Larocque  96872074.82$  Sabrina Hilger  102611005.33$  Glayds Hinch  101573590.11$  Nereida Mickley  101278626.39$  Elaina Hupp  103829407.46$  Karissa Brousseau  98142582.89$  America Kyle  99146314.73$  Lucie Nakata  102881106.42$  Chantel Grewell  96314595.10$  Logan Hogans  97531477.92$  Herb Mojarro  96535514.77$  transaction  FlatPercent  97854.22$  20  4 33  transaction  FlatAmount  98091.14$  10273.03$  19 9  transaction  FlatAmount  97781.88$  10029.55$  27 41  transaction  FlatPercent  104642.02$  45  30 1  transaction  FlatAmount  95760.77$  9811.63$  7 20  transaction  FlatAmount  101486.10$  10377.16$  11 24  transaction  FlatAmount  99204.78$  9813.90$  17 17  transaction  FlatAmount  102725.61$  9895.83$  3 25  transaction  FlatAmount  96297.44$  10480.20$  43 4  transaction  FlatAmount  98375.44$  10360.05$  18 27  print  stop | Transaction amount: 97854.22$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 98091.14$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 97781.88$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 104642.02$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 95760.77$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 101486.10$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 99204.78$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 102725.61$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 96297.44$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 98375.44$  Transaction description: FlatAmount  Transaction successful? true  Name: Banka na RM  Name: Alana Pagel  Balance: 96124963.59$  Name: Kathe Vath  Balance: 104176470.01$  Name: Sara Flenniken  Balance: 103489869.36$  Name: Shauna Irick  Balance: 99118236.33$  Name: Leon Brokaw  Balance: 100928197.38$  Name: Guillermina Haymond  Balance: 101372350.52$  Name: Sherie Pestana  Balance: 100772483.38$  Name: Shirlene Reichard  Balance: 98346223.39$  Name: Jerrica Kafka  Balance: 100608943.69$  Name: Nida Garg  Balance: 96472274.15$  Name: Ellamae Marshell  Balance: 101834753.75$  Name: Ranee Ricca  Balance: 96274803.75$  Name: Elois Philip  Balance: 96767866.34$  Name: Roland June  Balance: 99245215.19$  Name: Marvis Lemarr  Balance: 95732058.60$  Name: Burl Friese  Balance: 100232415.35$  Name: Reggie Olah  Balance: 104652507.25$  Name: Isis Ballard  Balance: 100707466.76$  Name: Bernardine Munford  Balance: 104148280.34$  Name: Johnette Greenhaw  Balance: 103363451.29$  Name: Olimpia Ryan  Balance: 103335200.20$  Name: Sha Duguay  Balance: 96926529.60$  Name: Donn Kurt  Balance: 100902436.77$  Name: Joel Cao  Balance: 96522430.27$  Name: Jamika Epstein  Balance: 104204388.84$  Name: Moira Haar  Balance: 102498737.43$  Name: Tiffani Bellantoni  Balance: 98824115.92$  Name: Tennille Hanzlik  Balance: 101933904.75$  Name: Dwayne Bazile  Balance: 96723845.51$  Name: Luigi Blaise  Balance: 103007704.71$  Name: Paris Fagan  Balance: 100517016.89$  Name: Ngan Raynor  Balance: 100211914.45$  Name: Alden Mirarchi  Balance: 102954898.08$  Name: Tangela Fernandez  Balance: 99629769.30$  Name: Liza Breaux  Balance: 103806099.54$  Name: Annice Lipman  Balance: 97834836.56$  Name: Broderick Arcand  Balance: 102290753.08$  Name: Terry Cygan  Balance: 97565192.65$  Name: Adriene Janz  Balance: 104240885.04$  Name: Joaquin Larocque  Balance: 96872074.82$  Name: Sabrina Hilger  Balance: 102611005.33$  Name: Glayds Hinch  Balance: 101671371.99$  Name: Nereida Mickley  Balance: 101278626.39$  Name: Elaina Hupp  Balance: 103722629.82$  Name: Karissa Brousseau  Balance: 98142582.89$  Name: America Kyle  Balance: 99146314.73$  Name: Lucie Nakata  Balance: 102881106.42$  Name: Chantel Grewell  Balance: 96314595.10$  Name: Logan Hogans  Balance: 97531477.92$  Name: Herb Mojarro  Balance: 96535514.77$  Total provisions: 147701.05$  Total transfers: 992219.40$ |
| typical\_usage  Banka na RM  50  Alana Pagel  99491717.74$  Kathe Vath  101034259.42$  Sara Flenniken  102239934.88$  Shauna Irick  98340958.01$  Leon Brokaw  97863887.92$  Guillermina Haymond  99718355.69$  Sherie Pestana  97717745.56$  Shirlene Reichard  95050275.07$  Jerrica Kafka  101397911.06$  Nida Garg  102118425.55$  Ellamae Marshell  99210032.91$  Ranee Ricca  98977997.00$  Elois Philip  101387679.55$  Roland June  99206459.15$  Marvis Lemarr  101591681.64$  Burl Friese  99820214.42$  Reggie Olah  102426247.25$  Isis Ballard  103575023.55$  Bernardine Munford  100898571.41$  Johnette Greenhaw  95369550.73$  Olimpia Ryan  100877885.02$  Sha Duguay  101840155.71$  Donn Kurt  97414029.95$  Joel Cao  101283649.59$  Jamika Epstein  97272302.74$  Moira Haar  99618843.49$  Tiffani Bellantoni  100029287.21$  Tennille Hanzlik  96074275.74$  Dwayne Bazile  101633795.28$  Luigi Blaise  98076759.13$  Paris Fagan  95329168.92$  Ngan Raynor  98852735.01$  Alden Mirarchi  104400645.29$  Tangela Fernandez  98512617.18$  Liza Breaux  104278086.43$  Annice Lipman  101869272.36$  Broderick Arcand  103295196.13$  Terry Cygan  95478724.92$  Adriene Janz  97867714.22$  Joaquin Larocque  95948559.19$  Sabrina Hilger  98272587.48$  Glayds Hinch  99533008.66$  Nereida Mickley  100636571.70$  Elaina Hupp  104550334.32$  Karissa Brousseau  102506217.95$  America Kyle  97875987.79$  Lucie Nakata  97583055.41$  Chantel Grewell  101373719.89$  Logan Hogans  97260587.80$  Herb Mojarro  104846882.01$  transaction  FlatPercent  97175.67$  37  8 1  transaction  FlatAmount  104536.23$  9583.35$  28 8  transaction  FlatAmount  102757.10$  10461.81$  40 18  transaction  FlatPercent  97198.46$  26  42 31  transaction  FlatAmount  103913.44$  10259.30$  47 47  transaction  FlatPercent  97485.74$  16  33 20  transaction  FlatPercent  100186.45$  19  24 37  transaction  FlatPercent  102032.31$  44  19 14  transaction  FlatPercent  99672.79$  19  19 6  transaction  FlatPercent  103522.42$  17  30 47  print  stop | Transaction amount: 97175.67$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 104536.23$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 102757.10$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 97198.46$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 103913.44$  Transaction description: FlatAmount  Transaction successful? true  Transaction amount: 97485.74$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 100186.45$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 102032.31$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 99672.79$  Transaction description: FlatPercent  Transaction successful? true  Transaction amount: 103522.42$  Transaction description: FlatPercent  Transaction successful? true  Name: Banka na RM  Name: Alana Pagel  Balance: 99491717.74$  Name: Kathe Vath  Balance: 101131435.09$  Name: Sara Flenniken  Balance: 102239934.88$  Name: Shauna Irick  Balance: 98340958.01$  Name: Leon Brokaw  Balance: 97863887.92$  Name: Guillermina Haymond  Balance: 99718355.69$  Name: Sherie Pestana  Balance: 97817418.35$  Name: Shirlene Reichard  Balance: 95050275.07$  Name: Jerrica Kafka  Balance: 101369316.87$  Name: Nida Garg  Balance: 102118425.55$  Name: Ellamae Marshell  Balance: 99210032.91$  Name: Ranee Ricca  Balance: 98977997.00$  Name: Elois Philip  Balance: 101387679.55$  Name: Roland June  Balance: 99206459.15$  Name: Marvis Lemarr  Balance: 101693713.95$  Name: Burl Friese  Balance: 99820214.42$  Name: Reggie Olah  Balance: 102426247.25$  Name: Isis Ballard  Balance: 103575023.55$  Name: Bernardine Munford  Balance: 101001328.51$  Name: Johnette Greenhaw  Balance: 95104013.87$  Name: Olimpia Ryan  Balance: 100975370.76$  Name: Sha Duguay  Balance: 101840155.71$  Name: Donn Kurt  Balance: 97414029.95$  Name: Joel Cao  Balance: 101283649.59$  Name: Jamika Epstein  Balance: 97153080.95$  Name: Moira Haar  Balance: 99618843.49$  Name: Tiffani Bellantoni  Balance: 100029287.21$  Name: Tennille Hanzlik  Balance: 96074275.74$  Name: Dwayne Bazile  Balance: 101519675.70$  Name: Luigi Blaise  Balance: 98076759.13$  Name: Paris Fagan  Balance: 95208047.76$  Name: Ngan Raynor  Balance: 98949933.47$  Name: Alden Mirarchi  Balance: 104400645.29$  Name: Tangela Fernandez  Balance: 98399533.84$  Name: Liza Breaux  Balance: 104278086.43$  Name: Annice Lipman  Balance: 101869272.36$  Name: Broderick Arcand  Balance: 103295196.13$  Name: Terry Cygan  Balance: 95578911.37$  Name: Adriene Janz  Balance: 97867714.22$  Name: Joaquin Larocque  Balance: 95948559.19$  Name: Sabrina Hilger  Balance: 98159368.57$  Name: Glayds Hinch  Balance: 99533008.66$  Name: Nereida Mickley  Balance: 100514101.76$  Name: Elaina Hupp  Balance: 104550334.32$  Name: Karissa Brousseau  Balance: 102506217.95$  Name: America Kyle  Balance: 97875987.79$  Name: Lucie Nakata  Balance: 97583055.41$  Name: Chantel Grewell  Balance: 101466983.01$  Name: Logan Hogans  Balance: 97260587.80$  Name: Herb Mojarro  Balance: 104846882.01$  Total provisions: 207594.13$  Total transfers: |

import java.util.\*;

import java.util.stream.Collectors;

public class BankTester {

public static void main(String[] args) {

Scanner jin = new Scanner(System.in);

String test\_type = jin.nextLine();

switch (test\_type) {

case "typical\_usage":

testTypicalUsage(jin);

break;

case "equals":

testEquals();

break;

}

jin.close();

}

private static void testEquals() {

Account a1 = new Account("Andrej", "20.00$");

Account a2 = new Account("Andrej", "20.00$");

Account a3 = new Account("Andrej", "30.00$");

Account a4 = new Account("Gajduk", "20.00$");

List<Account> all = Arrays.asList(a1, a2, a3, a4);

if (!(a1.equals(a1) && !a1.equals(a2) && !a2.equals(a1) && !a3.equals(a1)

&& !a4.equals(a1)

&& !a1.equals(null))) {

System.out.println("Your account equals method does not work properly.");

return;

}

Set<Long> ids = all.stream().map(Account::getId).collect(Collectors.toSet());

if (ids.size() != all.size()) {

System.out.println("Different accounts have the same IDS. This is not allowed");

return;

}

FlatAmountProvisionTransaction fa1 = new FlatAmountProvisionTransaction(10, 20, "20.00$", "10.00$");

FlatAmountProvisionTransaction fa2 = new FlatAmountProvisionTransaction(20, 20, "20.00$", "10.00$");

FlatAmountProvisionTransaction fa3 = new FlatAmountProvisionTransaction(20, 10, "20.00$", "10.00$");

FlatAmountProvisionTransaction fa4 = new FlatAmountProvisionTransaction(10, 20, "50.00$", "50.00$");

FlatAmountProvisionTransaction fa5 = new FlatAmountProvisionTransaction(30, 40, "20.00$", "10.00$");

FlatPercentProvisionTransaction fp1 = new FlatPercentProvisionTransaction(10, 20, "20.00$", 10);

FlatPercentProvisionTransaction fp2 = new FlatPercentProvisionTransaction(10, 20, "20.00$", 10);

FlatPercentProvisionTransaction fp3 = new FlatPercentProvisionTransaction(10, 10, "20.00$", 10);

FlatPercentProvisionTransaction fp4 = new FlatPercentProvisionTransaction(10, 20, "50.00$", 10);

FlatPercentProvisionTransaction fp5 = new FlatPercentProvisionTransaction(10, 20, "20.00$", 30);

FlatPercentProvisionTransaction fp6 = new FlatPercentProvisionTransaction(30, 40, "20.00$", 10);

if (fa1.equals(fa1) &&

!fa2.equals(null) &&

fa2.equals(fa1) &&

fa1.equals(fa2) &&

fa1.equals(fa3) &&

!fa1.equals(fa4) &&

!fa1.equals(fa5) &&

!fa1.equals(fp1) &&

fp1.equals(fp1) &&

!fp2.equals(null) &&

fp2.equals(fp1) &&

fp1.equals(fp2) &&

fp1.equals(fp3) &&

!fp1.equals(fp4) &&

!fp1.equals(fp5) &&

!fp1.equals(fp6)) {

System.out.println("Your transactions equals methods do not work properly.");

return;

}

Account accounts[] = new Account[]{a1, a2, a3, a4};

Account accounts1[] = new Account[]{a2, a1, a3, a4};

Account accounts2[] = new Account[]{a1, a2, a3};

Account accounts3[] = new Account[]{a1, a2, a3, a4};

Bank b1 = new Bank("Test", accounts);

Bank b2 = new Bank("Test", accounts1);

Bank b3 = new Bank("Test", accounts2);

Bank b4 = new Bank("Sample", accounts);

Bank b5 = new Bank("Test", accounts3);

if (!(b1.equals(b1) &&

!b1.equals(null) &&

!b1.equals(b2) &&

!b2.equals(b1) &&

!b1.equals(b3) &&

!b3.equals(b1) &&

!b1.equals(b4) &&

b1.equals(b5))) {

System.out.println("Your bank equals method do not work properly.");

return;

}

//accounts[2] = a1;

if (!b1.equals(b5)) {

System.out.println("Your bank equals method do not work properly.");

return;

}

long from\_id = a2.getId();

long to\_id = a3.getId();

Transaction t = new FlatAmountProvisionTransaction(from\_id, to\_id, "3.00$", "3.00$");

b1.makeTransaction(t);

if (b1.equals(b5)) {

System.out.println("Your bank equals method do not work properly.");

return;

}

b5.makeTransaction(t);

if (!b1.equals(b5)) {

System.out.println("Your bank equals method do not work properly.");

return;

}

System.out.println("All your equals methods work properly.");

}

private static void testTypicalUsage(Scanner jin) {

String bank\_name = jin.nextLine();

int num\_accounts = jin.nextInt();

jin.nextLine();

Account accounts[] = new Account[num\_accounts];

for (int i = 0; i < num\_accounts; ++i)

accounts[i] = new Account(jin.nextLine(), jin.nextLine());

Bank bank = new Bank(bank\_name, accounts);

while (true) {

String line = jin.nextLine();

switch (line) {

case "stop":

return;

case "transaction":

String descrption = jin.nextLine();

String amount = jin.nextLine();

String parameter = jin.nextLine();

int from\_idx = jin.nextInt();

int to\_idx = jin.nextInt();

jin.nextLine();

Transaction t = getTransaction(descrption, from\_idx, to\_idx, amount, parameter, bank);

System.out.println("Transaction amount: " + t.getAmount());

System.out.println("Transaction description: " + t.getDescription());

System.out.println("Transaction successful? " + bank.makeTransaction(t));

break;

case "print":

System.out.println(bank.toString());

System.out.println("Total provisions: " + bank.totalProvision());

System.out.println("Total transfers: " + bank.totalTransfers());

System.out.println();

break;

}

}

}

private static Transaction getTransaction(String description, int from\_idx, int to\_idx, String amount, String o, Bank bank) {

switch (description) {

case "FlatAmount":

return new FlatAmountProvisionTransaction(bank.getAccounts()[from\_idx].getId(),

bank.getAccounts()[to\_idx].getId(), amount, o);

case "FlatPercent":

return new FlatPercentProvisionTransaction(bank.getAccounts()[from\_idx].getId(),

bank.getAccounts()[to\_idx].getId(), amount, Integer.parseInt(o));

}

return null;

}

}

class Account {

private String name;

private long ID;

private String balance;

public Account(String name, String balance) {

this.name = name;

this.balance = balance;

Random random = new Random();

this.ID = random.nextLong();

}

public String getName() {

return name;

}

public long getId() {

return ID;

}

public String getBalance() {

return balance;

}

public void setBalance(String balance) {

this.balance = balance;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append("Name: ").append(name).append("\n");

// Parse the balance to double and format it

double balanceValue = Double.parseDouble(balance.substring(0, balance.length() - 1)); // Remove the dollar sign

String formattedBalance = String.format("%.2f$", balanceValue);

sb.append("Balance: ").append(formattedBalance).append("\n");

return sb.toString();

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Account)) return false;

Account account = (Account) o;

return ID == account.ID && Objects.equals(name, account.name) && Objects.equals(balance, account.balance);

}

@Override

public int hashCode() {

return Objects.hash(name, ID, balance);

}

}

abstract class Transaction {

private long fromId;

private long toId;

private String description;

private String amount;

public Transaction(long fromId, long toId, String description, String amount) {

this.fromId = fromId;

this.toId = toId;

this.description = description;

this.amount = amount;

}

public abstract double getProvision();

public long getFromId() {

return fromId;

}

public void setFromId(long fromId) {

this.fromId = fromId;

}

public long getToId() {

return toId;

}

public void setToId(long toId) {

this.toId = toId;

}

public String getAmount() {

return amount;

}

public String getDescription() {

return description;

}

public void setAmount(String amount) {

this.amount = amount;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Transaction)) return false;

Transaction that = (Transaction) o;

return fromId == that.fromId && toId == that.toId && Objects.equals(description, that.description) && Objects.equals(amount, that.amount);

}

@Override

public int hashCode() {

return Objects.hash(fromId, toId, description, amount);

}

}

class FlatAmountProvisionTransaction extends Transaction {

private String flatProvision;

public FlatAmountProvisionTransaction(long fromId, long toId, String amount, String flatProvision) {

super(fromId, toId, "FlatAmount", amount);

this.flatProvision = flatProvision;

}

public double parseStringToDouble(String s) {

return Double.parseDouble(s.substring(0, s.length() - 1));

}

@Override

public double getProvision() {

return parseStringToDouble(flatProvision);

}

public String getFlatProvision() {

return flatProvision;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof FlatAmountProvisionTransaction)) return false;

if (!super.equals(o)) return false;

FlatAmountProvisionTransaction that = (FlatAmountProvisionTransaction) o;

return Objects.equals(flatProvision, that.flatProvision);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), flatProvision);

}

}

class FlatPercentProvisionTransaction extends Transaction {

private int centsPerDolar;

public FlatPercentProvisionTransaction(long fromId, long toId, String amount, int centsPerDolar) {

super(fromId, toId, "FlatPercent", amount);

this.centsPerDolar = centsPerDolar;

}

public int getPercent() {

return centsPerDolar;

}

public double parseStringToDouble(String s) {

return Double.parseDouble(s.substring(0, s.length() - 1));

}

@Override

public double getProvision() {

return (int) parseStringToDouble(getAmount()) \* centsPerDolar / 100.0;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof FlatPercentProvisionTransaction)) return false;

if (!super.equals(o)) return false;

FlatPercentProvisionTransaction that = (FlatPercentProvisionTransaction) o;

return centsPerDolar == that.centsPerDolar;

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), centsPerDolar);

}

}

class Bank {

String name;

Account accounts[];

public double totalProvision;

public double totalTransfers;

public Bank(String name, Account[] accounts) {

this.name = name;

this.accounts = new Account[accounts.length];

System.arraycopy(accounts, 0, this.accounts, 0, accounts.length);

this.totalProvision = 0;

this.totalTransfers = 0;

}

public int findId(long id) {

int index = -1;

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

if (id == accounts[i].getId()) {

index = i;

break;

}

}

return index;

}

public double parseStringToDouble(String s) {

return Double.parseDouble(s.substring(0, s.length() - 1));

}

public boolean makeTransaction(Transaction t) {

//postoet smetkite

if (findId(t.getFromId()) == -1 || findId(t.getToId()) == -1) {

return false;

}

//dali imat dovolno sredstva

int indexFrom = findId(t.getFromId());

int indexTo = findId(t.getToId());

double balanceFrom = parseStringToDouble(accounts[indexFrom].getBalance());

double balanceTo = parseStringToDouble(accounts[indexTo].getBalance());

double transactionAmount = parseStringToDouble(t.getAmount());

if (balanceFrom < transactionAmount) {

return false;

}

double provision = t.getProvision();

totalProvision += provision;

totalTransfers += transactionAmount;

if (indexFrom == indexTo) {

accounts[indexFrom].setBalance(String.format("%2f$", balanceFrom - provision));

} else {

accounts[indexFrom].setBalance(String.format("%2f$", balanceFrom - provision - transactionAmount));

accounts[indexTo].setBalance(String.format("%2f$", balanceTo + transactionAmount));

}

return true;

}

public String totalProvision() {

return String.format("%.2f$", totalProvision);

}

public String totalTransfers() {

return String.format("%.2f$", totalTransfers);

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Bank)) return false;

Bank bank = (Bank) o;

return Double.compare(totalProvision, bank.totalProvision) == 0 && Double.compare(totalTransfers, bank.totalTransfers) == 0 && Objects.equals(name, bank.name) && Arrays.equals(accounts, bank.accounts);

}

@Override

public int hashCode() {

int result = Objects.hash(name, totalProvision, totalTransfers);

result = 31 \* result + Arrays.hashCode(accounts);

return result;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(String.format("Name: %s\n\n", name));

for (Account account : accounts) {

sb.append(account.toString());

}

return sb.toString();

}

public Account[] getAccounts() {

return accounts;

}

}

**ЛАБ 2**

Се со цел да се подобри комуникацијата на факултетот потребно е да се направи систем за чување на контакти за секој студент.

Да се креира класа Contact. За потребите на оваа класа да се дефинираат следниве методи:

* Contact(String date) - конструктор каде што date е датумот кога е креиран контактот даден во следниов формат YYYY-MM-DD
* isNewerThan(Contact c):boolean - метод кој враќа true доколку контактот е креиран подоцна од контактот c и обратно
* getType():String - метод кој враќа вредност "Email" или "Phone" во зависност од типот на контактот

Од класата Contact не треба да може директно да се инстанцира објект.

Од оваа класа се изведуваат класите EmailContact и PhoneContact.

За класата EmailContact дополнително се чува e-маил кој што е од типот String. Да се дефинираат следниве методи:

* EmailContact(String date, String email) - конструктор
* getEmail():String - метод кој што го враќа е-маилот
* getType():String- имплементација на методот од класата Contact

За класата PhoneContact дополнително се чува телефонски број кој што е од типот String и оператор кој што е енумерација и се дефинира на следниов начин enum Operator { VIP, ONE, TMOBILE }. За оваа класа да се дефинираат следниве методи:

* PhoneContact(String date, String phone) - конструктор
* getPhone():String - метод кој што го враќа телефонскиот број
* getOperator():Operator - метод кој што го враќа операторот (070, 071, 072 – TMOBILE, 075,076 – ONE, 077, 078 – VIP)
* getType():String- имплементација на методот од класата Contact

\*Забелешка: Сите телефонски броеви се во формат 07X/YYY-ZZZ каде што X има вредност {0,1,2,5,6,7,8}

Потоа да се дефинира класата Student каде што се чува низа на контакти за секој студент

* Student(String firstName, String lastName, String city, int age, long index) – конструктор
* addEmailContact(String date, String email):void – метод што додава е-маил контакт во низата на контакти
* addPhoneContact(String date, String phone):void – метод што додава телефонски контакт во низата на контакти
* getEmailContacts():Contact[] – враќа низа на email контактите на студентот
* getPhoneContacts():Contact[] – враќа низа на phone контактите на студентот
* getCity():String - метод кој го враќа градот
* getFullName():String - метод кој го враќа целосното име на студентот во формат IME PREZIME
* getIndex():long - метод кој го враќа индексот на студентот
* getLatestContact():Contact – го враќа најновиот контакт (според датум) од студентот
* toString() – претставува JSON репрезентација на класата студент пр. {"ime":"Jovan", "prezime":"Jovanov", "vozrast":20, "grad":"Skopje", "indeks":101010, "telefonskiKontakti":["077/777-777", "078/888-888"], "emailKontakti":["jovan.jovanov@example.com", "jovanov@jovan.com", "jovan@jovanov.com"]}

\*Забелешка: Во класата Student да се чува само една низа од контакти Contact[], а не две низи одделно (PhoneContact[] и EmailContact[])

\*Напомена да не се користи instanceOf или getClass при имплементација на овие методи

Дополнително да се дефинира класа Faculty. За оваа класа да се дефинираат следниве методи:

* Faculty(String name, Student [] students) – конструктор
* countStudentsFromCity(String cityName):int – враќа колку студенти има од даден град
* getStudent(long index):Student – метод кој го враќа студентот кој го има дадениот индекс
* getAverageNumberOfContacts():double – враќа просечен број на контакти по студент
* getStudentWithMostContacts():Student – метод кој го враќа студентот со најмногу контакти (доколку има повеќе студенти со ист број на контакти да го врати студентот со најголем индекс)
* toString() – претставува JSON репрезентација на класата Faculty пример: {"fakultet":"FINKI", "studenti":[STUDENT1, STUDENT2, ...]} каде што треба да има целосни информации за секој студент.

import java.text.DecimalFormat;

import java.util.Arrays;

import java.util.Objects;

import java.util.Scanner;

enum Operator {

VIP,

ONE,

TMOBILE

}

abstract class Contact {

public String date;

public Contact(String date) {

this.date = date;

}

public boolean isNewerThan(Contact c){

return this.getDateAsInt() > c.getDateAsInt();

}

public int getDateAsInt(){

String[] datum = this.date.split("-");

return Integer.parseInt(datum[0])\*10000 + Integer.parseInt(datum

[1])\*100 + Integer.parseInt(datum[2]);

}

abstract public String getType();

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Contact)) return false;

Contact contact = (Contact) o;

return Objects.equals(date, contact.date);

}

@Override

public int hashCode() {

return Objects.hash(date);

}

}

class Student {

String firstName;

String lastName;

String city;

int age;

long index;

Contact[] contacts;

int n;

public Student(String firstName, String lastName, String city, int age, long index) {

this.firstName = firstName;

this.lastName = lastName;

this.city = city;

this.age = age;

this.index = index;

this.contacts = new Contact[0];

this.n = 0;

}

public void addEmailContact(String date, String email){

contacts = Arrays.copyOf(contacts, n+1);

contacts[n]= new EmailContact(date, email);

n++;

}

public void addPhoneContact(String date, String phone){

contacts = Arrays.copyOf(contacts, n+1);

contacts[n]= new PhoneContact(date, phone);

n++;

}

public Contact[] getEmailContacts(){

Contact[] niza;

int i=0;

for (Contact c: contacts) {

if(c.getType().equals("Email")){

i++;

}

}

niza = new Contact[i];

i=0;

for (Contact c:contacts) {

if(c.getType().equals("Email")){

niza[i] = c;

i++;

}

}

return niza;

}

public Contact[] getPhoneContacts(){

Contact[] nova;

int i=0;

for (Contact c: contacts) {

if(c.getType().equals("Phone")){

i++;

}

}

nova = new Contact[i];

i=0;

for (Contact c: contacts) {

if(c.getType().equals("Phone")){

nova[i] = c;

i++;

}

}

return nova;

}

public String getCity() {

return city;

}

public String getFullName() {

return firstName + " " + lastName;

}

public long getIndex() {

return index;

}

public Contact getLatestContact(){

if(contacts.length==0){

return null;

}

Contact latest =contacts[0];

for (int i = 1; i < contacts.length; i++) {

if(contacts[i].isNewerThan(latest)){

latest=contacts[i];

}

}

return latest;

}

@Override

public String toString() {

return "{\"ime\":\"" + firstName + "\", \"prezime\":\"" + lastName + "\", \"vozrast\":" + age +

", \"grad\":\"" + city + "\", \"indeks\":" + index +

", \"telefonskiKontakti\":" + Arrays.toString(getPhoneContacts()) +

", \"emailKontakti\":" + Arrays.toString(getEmailContacts()) + "}";

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof Student)) return false;

Student student = (Student) o;

return age == student.age && index == student.index && n == student.n && Objects.equals(firstName, student.firstName) && Objects.equals(lastName, student.lastName) && Objects.equals(city, student.city) && Arrays.equals(contacts, student.contacts);

}

@Override

public int hashCode() {

int result = Objects.hash(firstName, lastName, city, age, index, n);

result = 31 \* result + Arrays.hashCode(contacts);

return result;

}

}

class Faculty {

String name;

Student[] students;

public Faculty(String name, Student[] students) {

this.name = name;

this.students = students;

}

public int countStudentsFromCity(String cityName){

int counter=0;

for (Student s: students) {

if(s.city.equals(cityName)){

counter++;

}

}

return counter;

}

public Student getStudent(long index){

for (Student s:students) {

if(s.index == index) return s;

}

return null;

}

public double getAverageNumberOfContacts(){

double average;

int suma=0;

int counter=0;

for (Student s : students){

suma+= s.contacts.length;

counter++;

}

average=(double)suma/counter;

return average;

}

public Student getStudentWithMostContacts(){

Student max=students[0];

for (Student s:students) {

if(s.contacts.length > max.contacts.length){

max = s;

}

else if(s.contacts.length == max.contacts.length){

if(s.index > max.index){

max =s;

}

}

}

return max;

}

@Override

public String toString() {

return "{\"fakultet\":\"" + name + "\", \"studenti\":" + Arrays.toString(students) + "}";

}

}

class EmailContact extends Contact{

public String email;

public String date;

public EmailContact(String date, String email) {

super(date);

this.email = email;

}

public String getEmail() {

return email;

}

@Override

public String getType() {

return "Email";

}

@Override

public String toString() {

return

"\"" + email + "\"" ;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof EmailContact)) return false;

EmailContact that = (EmailContact) o;

if (!super.equals(o)) return false;

return Objects.equals(email, that.email) && Objects.equals(date, that.date);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), email, date);

}

}

class PhoneContact extends Contact {

public String phone;

public Operator operator;

public PhoneContact(String date, String phone) {

super(date);

this.phone = phone;

}

public String getPhone() {

return phone;

}

public Operator getOperator() {

String[] niza = phone.split("/");

int nov = Integer.parseInt(niza[0].substring(2));

if (nov == 0 || nov == 1 || nov == 2) {

return Operator.TMOBILE;

} else if (nov == 5 || nov == 6) {

return Operator.ONE;

} else return Operator.VIP;

}

@Override

public String getType() {

return "Phone";

}

@Override

public String toString() {

return "\"" + phone + "\"" ;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (!(o instanceof PhoneContact)) return false;

PhoneContact that = (PhoneContact) o;

if (!super.equals(o)) return false;

return Objects.equals(phone, that.phone) && operator == that.operator;

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), phone, operator);

}

}

public class ContactsTester {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int tests = scanner.nextInt();

Faculty faculty = null;

int rvalue = 0;

long rindex = -1;

DecimalFormat df = new DecimalFormat("0.00");

for (int t = 0; t < tests; t++) {

rvalue++;

String operation = scanner.next();

switch (operation) {

case "CREATE\_FACULTY": {

String name = scanner.nextLine().trim();

int N = scanner.nextInt();

Student[] students = new Student[N];

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

rvalue++;

String firstName = scanner.next();

String lastName = scanner.next();

String city = scanner.next();

int age = scanner.nextInt();

long index = scanner.nextLong();

if ((rindex == -1) || (rvalue % 13 == 0))

rindex = index;

Student student = new Student(firstName, lastName, city,

age, index);

students[i] = student;

}

faculty = new Faculty(name, students);

break;

}

case "ADD\_EMAIL\_CONTACT": {

long index = scanner.nextInt();

String date = scanner.next();

String email = scanner.next();

rvalue++;

if ((rindex == -1) || (rvalue % 3 == 0))

rindex = index;

faculty.getStudent(index).addEmailContact(date, email);

break;

}

case "ADD\_PHONE\_CONTACT": {

long index = scanner.nextInt();

String date = scanner.next();

String phone = scanner.next();

rvalue++;

if ((rindex == -1) || (rvalue % 3 == 0))

rindex = index;

faculty.getStudent(index).addPhoneContact(date, phone);

break;

}

case "CHECK\_SIMPLE": {

System.out.println("Average number of contacts: "

+ df.format(faculty.getAverageNumberOfContacts()));

rvalue++;

String city = faculty.getStudent(rindex).getCity();

System.out.println("Number of students from " + city + ": "

+ faculty.countStudentsFromCity(city));

break;

}

case "CHECK\_DATES": {

rvalue++;

System.out.print("Latest contact: ");

Contact latestContact = faculty.getStudent(rindex)

.getLatestContact();

if (latestContact.getType().equals("Email"))

System.out.println(((EmailContact) latestContact)

.getEmail());

if (latestContact.getType().equals("Phone"))

System.out.println(((PhoneContact) latestContact)

.getPhone()

+ " ("

+ ((PhoneContact) latestContact).getOperator()

.toString() + ")");

if (faculty.getStudent(rindex).getEmailContacts().length > 0

&& faculty.getStudent(rindex).getPhoneContacts().length > 0) {

System.out.print("Number of email and phone contacts: ");

System.out

.println(faculty.getStudent(rindex)

.getEmailContacts().length

+ " "

+ faculty.getStudent(rindex)

.getPhoneContacts().length);

System.out.print("Comparing dates: ");

int posEmail = rvalue

% faculty.getStudent(rindex).getEmailContacts().length;

int posPhone = rvalue

% faculty.getStudent(rindex).getPhoneContacts().length;

System.out.println(faculty.getStudent(rindex)

.getEmailContacts()[posEmail].isNewerThan(faculty

.getStudent(rindex).getPhoneContacts()[posPhone]));

}

break;

}

case "PRINT\_FACULTY\_METHODS": {

System.out.println("Faculty: " + faculty.toString());

System.out.println("Student with most contacts: "

+ faculty.getStudentWithMostContacts().toString());

break;

}

}

}

scanner.close();

}

}

2. Да се напише класа која чува матрица од double вредности DoubleMatrix. Класата треба да е immutable, односно откако еднаш ќе се инстанцира да не може да се менува состојбата на објектот, односно да не може да се менуваат податоците зачувани во него. За потребите на класата треба да се имплементираат следните методи:

* DoubleMatrix(double a[], int m, int n) - конструктор кој прима низа од реални броеви каде што m и n се димензиите на матрицата. Од елементите на низата треба да се креира матрица. Доколку нема доволно елементи во низата тогаш да се фрли исклучок InsufficientElementsException, а доколку има повеќе елементи да се земат последните m x n вредности и со нив се потполнува матрицата, т.е. да се игнорираат вишокот на броеви од почетокот на низата
* getDimensions():String - метод кој враќа стринг во формат [m x n]
* rows():int - метод кој враќа број на редови
* columns():int - метод кој враќа број на колони
* maxElementAtRow(int row):double- метод кој го враќа максималниот елемент во дадениот ред, доколку вредноста е ред кој не постои да се фрли исклучок InvalidRowNumberException (row има вредност [1, m])
* maxElementAtColumn(int column):double- метод кој го враќа максималниот елемент во дадената колона, доколку вредноста е колона кој не постои да се фрли исклучок InvalidColumnNumberException (column има вредност [1, n])
* sum() : double - метод кој ја враќа сумата на сите елементи од матрицата
* toSortedArray():double[] – метод кој враќа еднодимензионална низа каде вредностите се сортирани во опаѓачки редослед
* toString() - методот, каде броевите се заокружени на 2 децимални места, меѓу себе се одделени со табулаторско место \t а редовите на матрицата се одделени со нов ред
* да се преоптоварат equals() и hashCode() методите

**Забелешка**: Исклучоците не треба да се фаќаат, треба само да се фрлаат

Да се дефинира класаInsufficientElementsException која што наследува од класата Exception и при фрлање на исклучок се добива порака"Insufficient number of elements"

Да се дефинира класа InvalidRowNumberExceptionкоја што наследува од класата Exception и при фрлање на исклучок се добива порака "Invalid row number"

Да се дефинира класа InvalidColumnNumberException која што наследува од класата Exception и при фрлање на исклучок се добива порака "Invalid column number"

Покрај класата DoubleMatrix треба да напишете дополнително уште една класа која ќе служи за вчитување на матрица од реални броеви од влезен тек на податоци. Оваа класа треба да се вика MatrixReader и во неа треба да имате еден public static метод за вчитување на матрица од реални броеви од InputStream

* read(InputStream input):DoubleMatrix - вчитува матрица од реални броеви од input зададена во следниот формат: Во првата линија има два цели броеви кои кажуваат колку редови и колони има матрицата, а во наредните редови се дадени елементите на матрицата по редови, одделени со едно или повеќе празни места

import java.io.ByteArrayInputStream;

import java.io.InputStream;

import java.text.DecimalFormat;

import java.util.Arrays;

import java.util.Objects;

import java.util.Scanner;

class InvalidColumnNumberException extends Exception{

public InvalidColumnNumberException(String message) {

super(message);

}

}

class InvalidRowNumberException extends Exception{

public InvalidRowNumberException(String message) {

super(message);

}

}

class MatrixReader {

public static DoubleMatrix read(InputStream input) throws InsufficientElementsException {

Scanner scanner = new Scanner(input);

int m = scanner.nextInt();

int n = scanner.nextInt();

double []niza = new double[m\*n];

for (int i = 0; i < m\*n; i++) {

niza[i] = scanner.nextDouble();

}

return new DoubleMatrix(niza, m,n);

}

@Override

public int hashCode() {

return super.hashCode();

}

@Override

public boolean equals(Object obj) {

return super.equals(obj);

}

}

class InsufficientElementsException extends Exception{

public InsufficientElementsException(String message) {

super(message);

}

}

final class DoubleMatrix {

double[] a;

int m;

int n;

double[][] matrix;

public DoubleMatrix(double[] a, int m, int n) throws InsufficientElementsException {

this.a = a;

this.m = m;

this.n = n;

if (a.length < m \* n) {

throw new InsufficientElementsException("Insufficient number of elements");

} else if (a.length > m \* n) {

matrix = new double[m][n];

int temp = a.length - m \* n;

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

for (int j = 0; j < n; j++) {

this.matrix[i][j] = a[temp];

temp++;

}

}

} else {

matrix = new double[m][n];

int temp = 0;

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

for (int j = 0; j < n; j++) {

this.matrix[i][j] = a[temp];

temp++;

}

}

}

}

public String getDimensions() {

return "[" + m + " x " + n + "]";

}

public int rows() {

return m;

}

public int columns() {

return n;

}

public double maxElementAtRow(int row) throws InvalidRowNumberException {

if (row > m || row < 1) {

throw new InvalidRowNumberException("Invalid row number");

}

double max = matrix[row-1][0];

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

if (max <= matrix[row-1][i]) {

max = matrix[row-1][i];

}

}

return max;

}

public double maxElementAtColumn(int column) throws InvalidColumnNumberException {

if (column > n || column < 1) {

throw new InvalidColumnNumberException("Invalid column number");

}

double max = matrix[0][column-1];

for (int i = 1; i < m; i++) {

if (max <= matrix[i][column-1]) {

max = matrix[i][column-1];

}

}

return max;

}

public double sum() {

double suma = 0.0;

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

for (int j = 0; j < n; j++) {

suma += matrix[i][j];

}

}

return suma;

}

public double[] toSortedArray() {

double[] a = new double[m \* n];

int temp = 0;

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

for (int j = 0; j < n; j++) {

a[temp++] = matrix[i][j];

}

}

Arrays.sort(a);

for (int i = 0; i < a.length / 2; i++) {

double temp2 = a[i];

a[i] = a[a.length - 1 - i];

a[a.length - 1 - i] = temp2;

}

return a;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

DoubleMatrix that = (DoubleMatrix) o;

return m == that.m && n == that.n && Arrays.deepEquals(matrix, that.matrix);

}

@Override

public int hashCode() {

int result = Objects.hash(m, n);

result = 31 \* result + Arrays.deepHashCode(matrix);

return result;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

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

for (int j = 0; j < n; j++) {

sb.append(String.format("%.2f", matrix[i][j]));

if (j != n - 1) {

sb.append("\t");

}

}

if (i != m - 1)

sb.append("\n");

}

return sb.toString();

}

}

public class DoubleMatrixTester {

public static void main(String[] args) throws Exception {

Scanner scanner = new Scanner(System.in);

int tests = scanner.nextInt();

DoubleMatrix fm = null;

double[] info = null;

DecimalFormat format = new DecimalFormat("0.00");

for (int t = 0; t < tests; t++) {

String operation = scanner.next();

switch (operation) {

case "READ": {

int N = scanner.nextInt();

int R = scanner.nextInt();

int C = scanner.nextInt();

double[] f = new double[N];

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

f[i] = scanner.nextDouble();

try {

fm = new DoubleMatrix(f, R, C);

info = Arrays.copyOf(f, f.length);

} catch (InsufficientElementsException e) {

System.out.println("Exception caught: " + e.getMessage());

}

break;

}

case "INPUT\_TEST": {

int R = scanner.nextInt();

int C = scanner.nextInt();

StringBuilder sb = new StringBuilder();

sb.append(R + " " + C + "\n");

scanner.nextLine();

for (int i = 0; i < R; i++)

sb.append(scanner.nextLine() + "\n");

fm = MatrixReader.read(new ByteArrayInputStream(sb

.toString().getBytes()));

info = new double[R \* C];

Scanner tempScanner = new Scanner(new ByteArrayInputStream(sb

.toString().getBytes()));

tempScanner.nextDouble();

tempScanner.nextDouble();

for (int z = 0; z < R \* C; z++) {

info[z] = tempScanner.nextDouble();

}

tempScanner.close();

break;

}

case "PRINT": {

System.out.println(fm.toString());

break;

}

case "DIMENSION": {

System.out.println("Dimensions: " + fm.getDimensions());

break;

}

case "COUNT\_ROWS": {

System.out.println("Rows: " + fm.rows());

break;

}

case "COUNT\_COLUMNS": {

System.out.println("Columns: " + fm.columns());

break;

}

case "MAX\_IN\_ROW": {

int row = scanner.nextInt();

try {

System.out.println("Max in row: "

+ format.format(fm.maxElementAtRow(row)));

} catch (InvalidRowNumberException e) {

System.out.println("Exception caught: " + e.getMessage());

}

break;

}

case "MAX\_IN\_COLUMN": {

int col = scanner.nextInt();

try {

System.out.println("Max in column: "

+ format.format(fm.maxElementAtColumn(col)));

} catch (InvalidColumnNumberException e) {

System.out.println("Exception caught: " + e.getMessage());

}

break;

}

case "SUM": {

System.out.println("Sum: " + format.format(fm.sum()));

break;

}

case "CHECK\_EQUALS": {

int val = scanner.nextInt();

int maxOps = val % 7;

for (int z = 0; z < maxOps; z++) {

double work[] = Arrays.copyOf(info, info.length);

int e1 = (31 \* z + 7 \* val + 3 \* maxOps) % info.length;

int e2 = (17 \* z + 3 \* val + 7 \* maxOps) % info.length;

if (e1 > e2) {

double temp = work[e1];

work[e1] = work[e2];

work[e2] = temp;

}

DoubleMatrix f1 = fm;

DoubleMatrix f2 = new DoubleMatrix(work, fm.rows(),

fm.columns());

System.out

.println("Equals check 1: "

+ f1.equals(f2)

+ " "

+ f2.equals(f1)

+ " "

+ (f1.hashCode() == f2.hashCode() && f1

.equals(f2)));

}

if (maxOps % 2 == 0) {

DoubleMatrix f1 = fm;

DoubleMatrix f2 = new DoubleMatrix(new double[]{3.0, 5.0,

7.5}, 1, 1);

System.out

.println("Equals check 2: "

+ f1.equals(f2)

+ " "

+ f2.equals(f1)

+ " "

+ (f1.hashCode() == f2.hashCode() && f1

.equals(f2)));

}

break;

}

case "SORTED\_ARRAY": {

double[] arr = fm.toSortedArray();

String arrayString = "[";

if (arr.length > 0)

arrayString += format.format(arr[0]) + "";

for (int i = 1; i < arr.length; i++)

arrayString += ", " + format.format(arr[i]);

arrayString += "]";

System.out.println("Sorted array: " + arrayString);

break;

}

}

}

scanner.close();

}

}

3.

Да се дефинира интерфејс Movable што ќе ги дефинира основните својства на еден движечки објект:

* движење нагоре (void moveUp())
* движење надолу (void moveLeft())
* движење надесно (void moveRight())
* движење налево (void moveLeft())
* пристап до моменталните **x,y** координати на објектот (int getCurrentXPosition() и int getCurrentYPosition()).

Постојат два типа на движечки објекти: движечка точка (MovingPoint) и движечки круг (MovingCircle). Да се дефинираат овие две класи коишто го имплементираат интерфејсот Movable.

Во класата MovingPoint се чуваат информации за:

* x и y координати (цели броеви)
* xSpeed и ySpeed : степенот на поместување на движечката точка во x насока и y насока (цели броеви)

За класата да се имплементираат:

* конструктор со аргументи: MovablePoint(int x, int y, int xSpeed, int ySpeed),
* методите наведени во интерфејсот Movable
* toString метод кој дава репрезентација на објектите во следнот формат Movable point with coordinates (5,35)

Во класата MovingCircle се чуваат информации за:

* радиусот на движечкиот круг (цел број)
* центарот на движечкиот круг (објект од класата MovingPoint).

За класата да се имплементираат:

* конструктор со аргументи: MovableCircle(int radius, MovablePoint center)
* методите наведени во интерфејсот Movable
* toString метод којшто дава репрезентација на објектите во следниот формат Movable circle with center coordinates (48,21) and radius 3

Првите четири методи од Movable (moveUp, modeDown, moveRight, moveLeft) треба да фрлат исклучок од тип ObjectCanNotBeMovedException доколку придвижувањето во соодветната насока не е возможно, односно со придвижувањето се излегува од дефинираниот простор во класата MovablesCollection. При движење на објекти од тип MovableCircle се смета дека кругот излегол од просторот, доколку неговиот центар излезе од центар. Дозволено е дел до кругот да излезе од просторот, доколку центарот е се уште во просторот. Справете се со овие исклучоци на соодветните места. *Погледнете во тест примерите какви пораки треба да се печатат кога ќе се фати исклучок од овој тип и имплементирајте го истото.*

Да се дефинира класа MovablesCollection во која што ќе се чуваат информации за:

* низа од движечки објекти (Movable [] movable)
* статичка променлива за максималната вредност на координатата X (минималната е предодредена на 0)
* статичка променлива за максималната вредност на координатата Y (минималната е предодредена на 0)

За класата да се имплементираат следните методи:

* конструктор MovablesCollection(int x\_MAX, int y\_MAX)
* void addMovableObject(Movable m) - метод за додавање на движечки објект во колекцијата од сите движечки објекти. Пред да се додади објектот, мора да се провери дали истиот е може да се вклопи во дефинираниот простор, односно истиот да не излегува од границите 0-X\_MAX за x координатата и 0-Y\_MAX за y координатата. Доколку станува збор за движечки круг, потребно е целиот круг да се наоѓа во наведениот интервал на вредности. Доколку движечкиот објект не може да биде вклопен во просторот, да се фрли исклучок од тип MovableObjectNotFittableException. Потребно е да се справите со исклучокот на соодветното место во main методот. *Погледнете во тест примерите какви пораки треба да се печатат кога ќе се фати исклучок од овој тип и имплементирајте го истото.*
* void moveObjectsFromTypeWithDirection (TYPE type, DIRECTION direction)- метод за придвижување на движечките објекти од тип type во насока direction. TYPE и DIRECTION се енумерации кои се задедени во почетниот код. Во зависност од насоката зададена во аргументот, да се повика соодветниот метод за придвижување.
* toString() - метод кој дава репрезентација на колекцијата од движечки објекти во следниот формат: Collection of movable objects with size [големина на колекцијата]: , по што во нов ред следуваат информации за сите движечки објекти во колекцијата.

| **Input** | **Result** |
| --- | --- |
| 3  1 21 9 19 20 32  1 18 41 18 13 32  0 13 55 18 4 | ===COLLECTION CONSTRUCTOR AND ADD METHOD TEST===  Movable circle with center (21,9) and radius 32 can not be fitted into the collection  Movable circle with center (18,41) and radius 32 can not be fitted into the collection  Collection of movable objects with size 1:  Movable point with coordinates (13,55)  MOVE POINTS TO THE LEFT  Point (-5,55) is out of bounds  Collection of movable objects with size 1:  Movable point with coordinates (13,55)  MOVE CIRCLES DOWN  Collection of movable objects with size 1:  Movable point with coordinates (13,55)  CHANGE X\_MAX AND Y\_MAX  MOVE POINTS TO THE RIGHT  Collection of movable objects with size 1:  Movable point with coordinates (31,55)  MOVE CIRCLES UP  Collection of movable objects with size 1:  Movable point with coordinates (31,55) |
| 10  0 38 42 15 1  1 8 55 7 24 39  1 2 55 16 10 37  0 49 41 9 29  0 5 33 16 26  1 23 17 13 11 15  1 42 11 18 27 15  1 23 25 0 26 16  0 36 8 14 24  0 3 41 1 0 | ===COLLECTION CONSTRUCTOR AND ADD METHOD TEST===  Movable circle with center (8,55) and radius 39 can not be fitted into the collection  Movable circle with center (2,55) and radius 37 can not be fitted into the collection  Movable circle with center (42,11) and radius 15 can not be fitted into the collection  Collection of movable objects with size 7:  Movable point with coordinates (38,42)  Movable point with coordinates (49,41)  Movable point with coordinates (5,33)  Movable circle with center coordinates (23,17) and radius 15  Movable circle with center coordinates (23,25) and radius 16  Movable point with coordinates (36,8)  Movable point with coordinates (3,41)  MOVE POINTS TO THE LEFT  Point (-11,33) is out of bounds  Collection of movable objects with size 7:  Movable point with coordinates (23,42)  Movable point with coordinates (40,41)  Movable point with coordinates (5,33)  Movable circle with center coordinates (23,17) and radius 15  Movable circle with center coordinates (23,25) and radius 16  Movable point with coordinates (22,8)  Movable point with coordinates (2,41)  MOVE CIRCLES DOWN  Point (23,-1) is out of bounds  Collection of movable objects with size 7:  Movable point with coordinates (23,42)  Movable point with coordinates (40,41)  Movable point with coordinates (5,33)  Movable circle with center coordinates (23,6) and radius 15  Movable circle with center coordinates (23,25) and radius 16  Movable point with coordinates (22,8)  Movable point with coordinates (2,41)  CHANGE X\_MAX AND Y\_MAX  MOVE POINTS TO THE RIGHT  Collection of movable objects with size 7:  Movable point with coordinates (38,42)  Movable point with coordinates (49,41)  Movable point with coordinates (21,33)  Movable circle with center coordinates (23,6) and radius 15  Movable circle with center coordinates (23,25) and radius 16  Movable point with coordinates (36,8)  Movable point with coordinates (3,41)  MOVE CIRCLES UP  Collection of movable objects with size 7:  Movable point with coordinates (38,42)  Movable point with coordinates (49,41)  Movable point with coordinates (21,33)  Movable circle with center coordinates (23,17) and radius 15  Movable circle with center coordinates (23,51) and radius 16  Movable point with coordinates (36,8)  Movable point with coordinates (3,41) |

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import java.util.Scanner;

enum TYPE {

POINT,

CIRCLE

}

enum DIRECTION {

UP,

DOWN,

LEFT,

RIGHT

}

public class CirclesTest {

public static void main(String[] args) {

System.out.println("===COLLECTION CONSTRUCTOR AND ADD METHOD TEST===");

MovablesCollection collection = new MovablesCollection(100, 100);

Scanner sc = new Scanner(System.in);

int samples = Integer.parseInt(sc.nextLine());

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

String inputLine = sc.nextLine();

String[] parts = inputLine.split(" ");

int x = Integer.parseInt(parts[1]);

int y = Integer.parseInt(parts[2]);

int xSpeed = Integer.parseInt(parts[3]);

int ySpeed = Integer.parseInt(parts[4]);

if (Integer.parseInt(parts[0]) == 0) { //point

collection.addMovableObject(new MovablePoint(x, y, xSpeed, ySpeed));

} else { //circle

int radius = Integer.parseInt(parts[5]);

collection.addMovableObject(new MovableCircle(radius, new MovablePoint(x, y, xSpeed, ySpeed)));

}

}

System.out.println(collection.toString());

System.out.println("MOVE POINTS TO THE LEFT");

collection.moveObjectsFromTypeWithDirection(TYPE.POINT, DIRECTION.LEFT);

System.out.println(collection.toString());

System.out.println("MOVE CIRCLES DOWN");

collection.moveObjectsFromTypeWithDirection(TYPE.CIRCLE, DIRECTION.DOWN);

System.out.println(collection.toString());

System.out.println("CHANGE X\_MAX AND Y\_MAX");

MovablesCollection.setxMax(90);

MovablesCollection.setyMax(90);

System.out.println("MOVE POINTS TO THE RIGHT");

collection.moveObjectsFromTypeWithDirection(TYPE.POINT, DIRECTION.RIGHT);

System.out.println(collection.toString());

System.out.println("MOVE CIRCLES UP");

collection.moveObjectsFromTypeWithDirection(TYPE.CIRCLE, DIRECTION.UP);

System.out.println(collection.toString());

}

}

**Лаб 3**

Треба да се развие систем за електронска нарачка од пицерија. Менито на пицеријата се состои од следново:

* Pizza:
* Standard: 10\$
* Pepperoni: 12\$
* Vegetarian: 8\$
* Extra
* Ketchup 3\$
* Coke 5\$

За да го претставите менито, секоја ставка треба да имплементира interface Item која опишува една ставка од менито и ги дефинира следниве методи:

* getPrice():int - ја дава цената за конкретната ставка

Следно, дефинирајте две класи ExtraItem и PizzaItem за да може да правите разлика меѓу пици и останатите работи во нарачката. И двете класи треба да имаат еден конструктор кој прима еден String аргумент.

* ExtraItem(String type) - валидни вредности за type се { "Coke", "Ketchup" }
* PizzaItem(String type) - валидни вредности за type се { Standard , Pepperoni , Vegetarian }

Ако за type се проследи некоја невалидна вредност (која ја нема на менито) треба да се фрли исклучок InvalidExtraTypeException, односно InvalidPizzaTypeException.

Последно имплементирајте ја класата Order. Таа треба да ги нуди следните функционалности:

* Order() - креира нова празна нарачка
* addItem(Item item, int count) - соодветната ставка се додава во нарачката (count означува колку примероци сакаме од дадената ставка). Aко count е поголем од 10 се фрла исклучок ItemOutOfStockException(item). Доколку во нарачката веќе ја има соодветната ставка Item тогаш истата се заменува со нова. Следниот код резултира со нарачка со една стандардна пица:
* Order order = new Order();
* order.addItem(new PizzaItem("Standard"), 2);
* order.addItem(new PizzaItem("Standard"), 1);
* getPrice():int - ја враќа вкупната цена на нарачката
* displayOrder() - ја печати содржината на нарачката со соодветни редни броеви пред секоја ставка, името, количината и збирна сума на ставката, како и вкупна сума за целата нарачка. За редниот број се резервирани 3 места порамнети во десно, за имињата на ставките се резервирани 15 места со порамнување од лево, за кардиналноста две места порамнети во десно и за цената на една ставка 5 места порамнети во десно. За "Total:" се резервирани 22 места со порамнување од лево и за вкупната цена 5 места порамнети во десно. Пример:

1.Standard x 2 20$

2.Vegetarian x 1 8$

3.Coke x 3 15$

Total: 43$

Редоследот по кој се печатат ставките е оној по кој тие се внесувани во нарачката. Доколку некоја ставка се внесе повторно нејзиното место не се менува.

* removeItem(int idx) - се отстранува нарачката со даден индекс (сите нарачки со поголеми индекси се поместуваат во лево). Доколку не постои нарачка со таков индекс треба да се фрли исклучок ArrayIndexOutOfBоundsException(idx)
* lock() - ја заклучува нарачката. За да може нарачката да се заклучи треба истата да има барем една ставка, во спротивно фрлете исклучок EmptyOrderException.

Откако ќе се заклучи нарачката треба веќе да не може да се менува со методите removeItem, addItem. Повикот на овие методи резултира со исклучок од типот OrderLockedException.

**For example:**

| **Input** | **Result** |
| --- | --- |
| 0 Pizza Fake | InvalidPizzaTypeException |
| 0 Extra Ketchup | 3 |

// package PizzaOrder;

import java.util.ArrayList;

import java.util.List;

import java.util.Objects;

import java.util.Scanner;

class InvalidExtraTypeException extends Exception{

public InvalidExtraTypeException(String message) {

super(message);

}

}

class InvalidPizzaTypeException extends Exception{

public InvalidPizzaTypeException(String message) {

super(message);

}

}

class ItemOutOfStockException extends Exception{

public ItemOutOfStockException(String message) {

super(message);

}

}

class EmptyOrder extends Exception{

public EmptyOrder(String message) {

super(message);

}

}

class OrderLockedException extends Exception{

public OrderLockedException(String message) {

super(message);

}

}

interface Item{

int getPrice();

String getType();

}

class ExtraItem implements Item{

String type;

public ExtraItem(String type) throws InvalidExtraTypeException {

if(type.equals("Coke") || type.equals("Ketchup")){

//(!Objects.equals(type,"Coke") && !Objects.equals(type,"Ketchup"))

this.type = type;

}else{

throw new InvalidExtraTypeException("InvalidExtraTypeException");

}

}

@Override

public int getPrice() {

if(type.equals("Ketchup")) return 3;

else if(type.equals("Coke")) return 5;

else return 0;

}

@Override

public String getType() {

if(type.equals("Ketchup")) return "Ketchup";

else if(type.equals("Coke")) return "Coke";

else return null;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

ExtraItem extraItem = (ExtraItem) o;

return Objects.equals(type, extraItem.type);

}

@Override

public int hashCode() {

return Objects.hash(type);

}

}

class PizzaItem implements Item{

String type;

public PizzaItem(String type) throws InvalidPizzaTypeException {

if(type.equals("Standard") || type.equals("Pepperoni")||

type.equals("Vegetarian")){

this.type = type;

}else{

throw new InvalidPizzaTypeException("InvalidPizzaTypeException");

}

}

@Override

public int getPrice() {

if(type.equals("Standard")) return 10;

else if (type.equals("Pepperoni")) return 12;

else if(type.equals("Vegetarian")) return 8;

else return 0;

}

@Override

public String getType() {

if(type.equals("Standard")) return "Standard";

else if (type.equals("Pepperoni")) return "Pepperoni";

else if(type.equals("Vegetarian")) return "Vegetarian";

else return null;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

PizzaItem pizzaItem = (PizzaItem) o;

return Objects.equals(type, pizzaItem.type);

}

@Override

public int hashCode() {

return Objects.hash(type);

}

}

class Product{

Item item;

int count;

public Product(Item item, int count) {

this.item = item;

this.count = count;

}

public Item getItem() {

return item;

}

public void setItem(Item item) {

this.item = item;

}

public int getCount() {

return count;

}

public void setCount(int count) {

this.count = count;

}

}

class Order{

List<Product> productList;

private boolean locked;

public Order() {

this.productList=new ArrayList<Product>();

this.locked=false;

}

void addItem(Item item, int count) throws ItemOutOfStockException, OrderLockedException {

if(count>10){

throw new ItemOutOfStockException("Item"+item+"out of stock");

}

if(locked){

throw new OrderLockedException("OrderLockedException");

}

for (Product product : productList) {

if(product.item.equals(item)){

product.setCount(count);

return;

}

}

productList.add(new Product(item,count));

}

public int getPrice(){

return productList.stream()

.mapToInt(i -> i.item.getPrice()\*i.getCount())

.sum();

}

public void displayOrder(){

for (int i = 0; i < productList.size(); i++) {

System.out.println(String.format("%3d.%-15sx%2d%5d$",i+1,

productList.get(i).item.getType(),productList.get(i).count,productList.get(i).item.getPrice()\*productList.get(i).count));

}

System.out.println(String.format("%-22s%5d$","Total:",getPrice()));

}

void removeItem(int i) throws OrderLockedException {

if(i<0||i>=productList.size()){

throw new ArrayIndexOutOfBoundsException(i);

}

if(locked){

throw new OrderLockedException("OrderLockedException");

}

productList.remove(i);

}

void lock() throws EmptyOrder {

if(productList.isEmpty()){

throw new EmptyOrder("Empty Order");

}else{

locked=true;

}

}

}

public class PizzaOrderTest {

public static void main(String[] args) {

Scanner jin = new Scanner(System.in);

int k = jin.nextInt();

if (k == 0) { //test Item

try {

String type = jin.next();

String name = jin.next();

Item item = null;

if (type.equals("Pizza")) item = new PizzaItem(name);

else item = new ExtraItem(name);

System.out.println(item.getPrice());

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

}

if (k == 1) { // test simple order

Order order = new Order();

while (true) {

try {

String type = jin.next();

String name = jin.next();

Item item = null;

if (type.equals("Pizza")) item = new PizzaItem(name);

else item = new ExtraItem(name);

if (!jin.hasNextInt()) break;

order.addItem(item, jin.nextInt());

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

}

jin.next();

System.out.println(order.getPrice());

order.displayOrder();

while (true) {

try {

String type = jin.next();

String name = jin.next();

Item item = null;

if (type.equals("Pizza")) item = new PizzaItem(name);

else item = new ExtraItem(name);

if (!jin.hasNextInt()) break;

order.addItem(item, jin.nextInt());

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

}

System.out.println(order.getPrice());

order.displayOrder();

}

if (k == 2) { // test order with removing

Order order = new Order();

while (true) {

try {

String type = jin.next();

String name = jin.next();

Item item = null;

if (type.equals("Pizza")) item = new PizzaItem(name);

else item = new ExtraItem(name);

if (!jin.hasNextInt()) break;

order.addItem(item, jin.nextInt());

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

}

jin.next();

System.out.println(order.getPrice());

order.displayOrder();

while (jin.hasNextInt()) {

try {

int idx = jin.nextInt();

order.removeItem(idx);

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

}

System.out.println(order.getPrice());

order.displayOrder();

}

if (k == 3) { //test locking & exceptions

Order order = new Order();

try {

order.lock();

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

try {

order.addItem(new ExtraItem("Coke"), 1);

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

try {

order.lock();

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

try {

order.removeItem(0);

} catch (Exception e) {

System.out.println(e.getClass().getSimpleName());

}

}

}

}

2. Во оваа задача треба да имплементирате класа која ќе претставува телефонски именик. Именикот се состои од повеќе контакти при што, за секој контакт се чуваат неговото име и максимум до 5 телефонски броја за тој контакт. За потребите на класата Contact треба да ги имплементирате следниве методи.

* Contact(String name, String... phonenumber) - конструктор со параметри - името треба да е подолго од 4 караткери, но максимум до 10 карактери и не смее да содржи други знаци освен латинични букви и бројки во спротивно се фрла исклучок од тип InvalidNameException - телефонските броеви мора да се состојат од точно 9 цифри при што првите три цифри се "070", "071", "072", "075","076","077" или "078" во спротивно се фрла исклучок од тип InvalidNumberException - контактот може да содржи максимум 5 броја во спротивно се фрла исклучок MaximumSizeExceddedException
* getName():String - get метод за името
* getNumbers():String[] - get метод за броевите кои треба да се лексикографски подредени (нека враќа копија од оригиналната низа)
* addNumber(String phonenumber) - метод кој додава нов број во контактот - за овој метод важат истите ограничувања за форматот на броевите како и во конструкторот
* toString():String - враќа текстуален опис во следниот формат Во прв ред името на контактот, во втор ред бројот на телефонски броеви и понатаму во одделни редови секој број поединечно повторно сортирани лексикографски
* valueOf(String s):Contact - статички метод кој за дадена тексутална репрезентација на контактот ќе врати соодветен објект - доколку настане било каков проблем при претварањето од тексутална репрезентација во објект Contact треба да се фрли исклучок од тип InvalidFormatException

Користејќи ја класата Contact која ја напишавте сега треба да се развие и класа за телефонски именик PhoneBook. Оваа класа содржи низа од не повеќе од 250 контакти и ги нуди следниве методи

* PhoneBook() - празен конструктор
* addContact(Contact contact):void - додава нов контакт во именикот, притоа доколку се надмине максималниот капацитет од 250 се фрла исклучок MaximumSizeExceddedException - дополнително ограничување е што сите имиња на контакти мора да бидат единствени, доколку контактот што сакате да го додадете има исто име со некој од веќе постоечките контакти треба да фрлите исклучок од типот InvalidNameException
* getContactForName(String name):Contact - го враќа контактот со соодветното име доколку таков постои во спротивно враќа null
* numberOfContacts():int - го враќа бројот на контакти во именикот
* getContacts():Contact[] - враќа низа од сите контакти сортирани според нивното име (нека враќа копија од низата)
* removeContact(String name):boolean - го брише соодветниот контакт од именикот и раќа true доколку постои, во спротивно враќа false
* toString():String - враќа текстуален опис на именикот каде се наредени сите контакти подредени според нивното име, одделени со по еден празен ред
* saveAsTextFile(PhoneBook phonebook,String path):boolean - статички метод кој го запишува именикот во текстуална датотека која се наоѓа на локација path,доколку не постои датотеката треба да се креира- методот враќа false само доколку има некаков проблем при запишување на податоците во датотеката
* loadFromTextFile(String path):Phonebook - статички метод кој вчитува именик претходно запишан со методот saveAsTextFile - доколку датотеката не постои или неможе да се отвори за читање се пропагира оригиналниот IOException, а доколку настане проблем при парсирањето на текстот од датотеката треба да се фрли исклучок InvalidFormatException
* getContactsForNumber(String number\_prefix):Contact[] - за даден префикс од број (првите неколку цифри) ги враќа сите контакти кои имаат барем еден број со тој префикс - низата не треба да содржи дупликат контакти или null елементи подредена според имињата на контактите

\*Сите исклучоци освен IOException треба сами да ги напишете - секаде каде што е можно додате дополнително објаснување или податочни членови во врска со причината за исклучокот

**For example:**

| **Input** | **Result** |
| --- | --- |
| test\_usage  add  Andrej  078123456 075123456  add  Gajduk  071123456 076123456  add  Tesst  070123456  add  Sample  075123456 076123456 078123456  print  remove  Gajduk  print | 4  [Andrej  2  075123456  078123456  , Gajduk  2  071123456  076123456  , Sample  3  075123456  076123456  078123456  , Tesst  1  070123456  ]  Andrej  2  075123456  078123456  Gajduk  2  071123456  076123456  Sample  3  075123456  076123456  078123456  Tesst  1  070123456  3  [Andrej  2  075123456  078123456  , Sample  3  075123456  076123456  078123456  , Tesst  1  070123456  ]  Andrej  2  075123456  078123456  Sample  3  075123456  076123456  078123456  Tesst  1  070123456 |
| test\_phonebook\_exceptions  Abraham  Abraham | Abraham |

// package PhoneContact;

import java.io.\*;

import java.util.\*;

import java.util.stream.IntStream;

class InvalidNameException extends Exception {

public String name;

public InvalidNameException(String name) {

super();

this.name=name;

}

}

class InvalidNumberException extends Exception {

public InvalidNumberException() {

super();

}

}

class MaximumSizeExceddedException extends Exception {

public MaximumSizeExceddedException() {

super();

}

}

class InvalidFormatException extends Exception {

public InvalidFormatException() {

super();

}

}

class Contact implements Comparable<Contact>{

private String name;

private String[] phonenumber;

public Contact(String name, String... phonenumber) throws InvalidNameException, InvalidNumberException, MaximumSizeExceddedException {

// if (name.length() < 4 || name.length() > 10) {

// throw new InvalidNameException(name);

// }

if (!checkName(name))

throw new InvalidNameException(name);

this.name = name;

if (!Arrays.stream(phonenumber).allMatch(this::checkPhone)) {

throw new InvalidNumberException();

}

if (phonenumber.length > 5) {

throw new MaximumSizeExceddedException();

}

this.phonenumber = new String[phonenumber.length];

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

this.phonenumber[i] = phonenumber[i];

}

// IntStream.range(0, phonenumber.length).forEach(i->this.phonenumber[i]=phonenumber[i]);

}

private boolean checkName(String name) {

return name.matches("[a-zA-Z0-9]{5,10}");

}

private boolean checkPhone(String phone) {

return phone.matches("07[0125678][0-9]{6}");

}

public String getName() {

return name;

}

public String[] getNumbers() {

return Arrays.stream(phonenumber)

.sorted()

.toArray(String[]::new);

}

void addNumber(String number) throws InvalidNumberException {

if (!checkPhone(number)) {

throw new InvalidNumberException();

}

phonenumber = Arrays.copyOf(phonenumber, phonenumber.length + 1);

phonenumber[phonenumber.length - 1] = number;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append(name).append("\n");

sb.append(phonenumber.length).append("\n");

Arrays.stream(phonenumber)

.sorted()

.forEach(i -> sb.append(i)

.append("\n"));

return sb.toString();

}

public static Contact valueOf(String s) throws InvalidFormatException {

try {

return new Contact(s);

} catch (Exception e) {

throw new InvalidFormatException();

}

}

@Override

public int compareTo(Contact o) {

return name.compareTo(o.name);

}

public boolean hasNumber(String s) {

return Arrays.stream(phonenumber)

.anyMatch(i -> i.startsWith(s));

}

}

class PhoneBook{

Contact []contacts;

public PhoneBook() {

this.contacts=new Contact[0];

}

void addContact(Contact contact) throws MaximumSizeExceddedException, InvalidNameException {

if(contacts.length>=250){

throw new MaximumSizeExceddedException();

}

// Contact []n=new Contact[contacts.length+1];

// System.arraycopy(contacts, 0, n, 0, contacts.length);

// n[contacts.length-1]=contact;

// contacts=n;

if(contactExsists(contact)){

throw new InvalidNameException(contact.getName());

}

contacts = Arrays.copyOf(contacts, contacts.length + 1);

contacts[contacts.length - 1] = contact;

}

private boolean contactExsists(Contact c){

return Arrays.stream(contacts)

.anyMatch(i->i.getName().equals(c.getName()));

}

public Contact getContactForName(String name){

return Arrays.stream(contacts)

.filter(i->i.getName().equals(name))

.findFirst()

.orElse(null);

}

public int numberOfContacts(){

return contacts.length;

}

public Contact[] getContacts() {

return Arrays.stream(contacts)

.sorted(Comparator.comparing(Contact::getName))

.toArray(Contact[]::new);

}

boolean removeContact(String name){

for (Contact c : contacts) {

if (c.getName().equals(name)) {

contacts=IntStream.range(0,contacts.length)

.filter(i->!contacts[i].getName().equals(name))

.mapToObj(i->contacts[i])

.toArray(Contact[]::new);

return true;

}

}

return false;

}

@Override

public String toString() {

StringBuilder sb=new StringBuilder();

Arrays.stream(contacts)

.sorted()

.forEach(i->sb.append(i).append("\n"));

return sb.toString();

}

static boolean saveAsTextFile(PhoneBook phonebook, String path){

try {

ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream(path));

oos.writeObject(phonebook);

oos.close();

} catch (IOException e) {

return false;

}

return true;

}

static PhoneBook loadFromTextFile(String path) throws InvalidFormatException {

PhoneBook p=null;

ObjectInputStream ois= null;

try {

ois = new ObjectInputStream(new FileInputStream(path));

p=(PhoneBook)ois.readObject();

ois.close();

} catch (IOException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

throw new InvalidFormatException();

}

return p;

}

Contact[] getContactsForNumber(String number\_prefix){

return Arrays.stream(contacts)

.filter(i -> i.hasNumber(number\_prefix))

.toArray(Contact[]::new);

}

}

public class PhonebookTester {

public static void main(String[] args) throws Exception {

Scanner jin = new Scanner(System.in);

String line = jin.nextLine();

switch( line ) {

case "test\_contact":

testContact(jin);

break;

case "test\_phonebook\_exceptions":

testPhonebookExceptions(jin);

break;

case "test\_usage":

testUsage(jin);

break;

}

}

private static void testFile(Scanner jin) throws Exception {

PhoneBook phonebook = new PhoneBook();

while ( jin.hasNextLine() )

phonebook.addContact(new Contact(jin.nextLine(),jin.nextLine().split("\\s++")));

String text\_file = "phonebook.txt";

PhoneBook.saveAsTextFile(phonebook,text\_file);

PhoneBook pb = PhoneBook.loadFromTextFile(text\_file);

if ( ! pb.equals(phonebook) ) System.out.println("Your file saving and loading doesn't seem to work right");

else System.out.println("Your file saving and loading works great. Good job!");

}

private static void testUsage(Scanner jin) throws Exception {

PhoneBook phonebook = new PhoneBook();

while ( jin.hasNextLine() ) {

String command = jin.nextLine();

switch ( command ) {

case "add":

phonebook.addContact(new Contact(jin.nextLine(),jin.nextLine().split("\\s++")));

break;

case "remove":

phonebook.removeContact(jin.nextLine());

break;

case "print":

System.out.println(phonebook.numberOfContacts());

System.out.println(Arrays.toString(phonebook.getContacts()));

System.out.println(phonebook.toString());

break;

case "get\_name":

System.out.println(phonebook.getContactForName(jin.nextLine()));

break;

case "get\_number":

System.out.println(Arrays.toString(phonebook.getContactsForNumber(jin.nextLine())));

break;

}

}

}

private static void testPhonebookExceptions(Scanner jin) {

PhoneBook phonebook = new PhoneBook();

boolean exception\_thrown = false;

try {

while ( jin.hasNextLine() ) {

phonebook.addContact(new Contact(jin.nextLine()));

}

}

catch ( InvalidNameException e ) {

System.out.println(e.name);

exception\_thrown = true;

}

catch ( Exception e ) {}

if ( ! exception\_thrown ) System.out.println("Your addContact method doesn't throw InvalidNameException");

/\*

exception\_thrown = false;

try {

phonebook.addContact(new Contact(jin.nextLine()));

} catch ( MaximumSizeExceddedException e ) {

exception\_thrown = true;

}

catch ( Exception e ) {}

if ( ! exception\_thrown ) System.out.println("Your addContact method doesn't throw MaximumSizeExcededException");

\*/

}

private static void testContact(Scanner jin) throws Exception {

boolean exception\_thrown = true;

String names\_to\_test[] = { "And\nrej","asd","AAAAAAAAAAAAAAAAAAAAAA","Ð�Ð½Ð´Ñ€ÐµÑ˜A123213","Andrej#","Andrej<3"};

for ( String name : names\_to\_test ) {

try {

new Contact(name);

exception\_thrown = false;

} catch (InvalidNameException e) {

exception\_thrown = true;

}

if ( ! exception\_thrown ) System.out.println("Your Contact constructor doesn't throw an InvalidNameException");

}

String numbers\_to\_test[] = { "+071718028","number","078asdasdasd","070asdqwe","070a56798","07045678a","123456789","074456798","073456798","079456798" };

for ( String number : numbers\_to\_test ) {

try {

new Contact("Andrej",number);

exception\_thrown = false;

} catch (InvalidNumberException e) {

exception\_thrown = true;

}

if ( ! exception\_thrown ) System.out.println("Your Contact constructor doesn't throw an InvalidNumberException");

}

String nums[] = new String[10];

for ( int i = 0 ; i < nums.length ; ++i ) nums[i] = getRandomLegitNumber();

try {

new Contact("Andrej",nums);

exception\_thrown = false;

} catch (MaximumSizeExceddedException e) {

exception\_thrown = true;

}

if ( ! exception\_thrown ) System.out.println("Your Contact constructor doesn't throw a MaximumSizeExceddedException");

Random rnd = new Random(5);

Contact contact = new Contact("Andrej",getRandomLegitNumber(rnd),getRandomLegitNumber(rnd),getRandomLegitNumber(rnd));

System.out.println(contact.getName());

System.out.println(Arrays.toString(contact.getNumbers()));

System.out.println(contact.toString());

contact.addNumber(getRandomLegitNumber(rnd));

System.out.println(Arrays.toString(contact.getNumbers()));

System.out.println(contact.toString());

contact.addNumber(getRandomLegitNumber(rnd));

System.out.println(Arrays.toString(contact.getNumbers()));

System.out.println(contact.toString());

}

static String[] legit\_prefixes = {"070","071","072","075","076","077","078"};

static Random rnd = new Random();

private static String getRandomLegitNumber() {

return getRandomLegitNumber(rnd);

}

private static String getRandomLegitNumber(Random rnd) {

StringBuilder sb = new StringBuilder(legit\_prefixes[rnd.nextInt(legit\_prefixes.length)]);

for ( int i = 3 ; i < 9 ; ++i )

sb.append(rnd.nextInt(10));

return sb.toString();

}

}

**Лаб 4**

Довршете ги методите

**For example:**

| **Input** | **Result** |
| --- | --- |
| no imput | 2015-06-18  2015-06-18  2015  2016-01-01  2016-04-18  2015-06-08  2016-08-21  true  P1M20D |

//package zad1;

import java.sql.Date;

import java.sql.Timestamp;

import java.time.\*;

import java.time.temporal.TemporalAdjusters;

/\*

\* LocalDate test

\*/

public class LocalDateTest {

public static void main(String[] args) {

System.out.println(create());

System.out.println(parse());

System.out.println(with().getYear());

System.out.println(withAdjuster());

System.out.println(plus());

System.out.println(minus());

System.out.println(plusPeriod());

System.out.println(isAfter());

System.out.println(until());

}

static LocalDate create() {

/\*

\* Create a {@link LocalDate} of 2015-06-18 by using {@link LocalDate#of}

\*/

return LocalDate.of(2015,6,18);

}

static LocalDate parse() {

/\*

\* Create a {@link LocalDate} of 2015-06-18 from String by using {@link LocalDate#parse}

\*/

return LocalDate.parse("2015-06-18");

}

static LocalDate with() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*

\* Create a {@link LocalDate} from {@link ld} with year 2015

\* by using {@link LocalDate#withYear} or {@link LocalDate#with}

\*/

return ld.withYear(2015);

}

static LocalDate withAdjuster() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*

\* Create a {@link LocalDate} from {@link ld} adjusted into first day of next year

\* by using {@link LocalDate#with} and {@link TemporalAdjusters#firstDayOfNextYear}

\*/

return ld.with(TemporalAdjusters.firstDayOfNextYear());

}

static LocalDate plus() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*

\* Create a {@link LocalDate} from {@link ld} with 10 month later

\* by using {@link LocalDate#plusMonths} or {@link LocalDate#plus}

\*/

return ld.plusMonths(10);

}

static LocalDate minus() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*

\* Create a {@link LocalDate} from {@link ld} with 10 days before

\* by using {@link LocalDate#minusDays} or {@link LocalDate#minus}

\*/

return ld.minusDays(10);

}

static LocalDate plusPeriod() {

LocalDate ld = DateAndTimes.LD\_20150618;

Period p=Period.of(1,2,3);

/\*

\* Define a {@link Period} of 1 year 2 month 3 days

\* Create a {@link LocalDate} adding the period to {@link ld} by using {@link LocalDate#plus}

\*/

return ld.plus(p);

}

static boolean isAfter() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalDate ld2 = DateAndTimes.LD\_20150807;

/\*

\* Check whether {@link ld2} is after {@link ld} or not

\* by using {@link LocalDate#isAfter} or {@link LocalDate#isBefore}

\*/

return ld2.isAfter(ld);

}

static Period until() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalDate ld2 = DateAndTimes.LD\_20150807;

/\*

\* Create a period from {@link ld} till {@link ld2}

\* by using {@link LocalDate#until}

\*/

return ld.until(ld2);

}

}

class DateAndTimes {

public static final LocalDate LD\_20150618 = LocalDate.of(2015, 6, 18);

public static final LocalDate LD\_20150807 = LocalDate.of(2015, 8, 7);

}

СТАРТЕР:

import java.sql.Date;

import java.sql.Timestamp;

import java.time.\*;

import java.time.temporal.TemporalAdjusters;

/\*\*

\* LocalDate test

\*/

public class LocalDateTest {

public static void main(String[] args) {

System.out.println(create());

System.out.println(parse());

System.out.println(with().getYear());

System.out.println(withAdjuster());

System.out.println(plus());

System.out.println(minus());

System.out.println(plusPeriod());

System.out.println(isAfter());

System.out.println(until());

}

static LocalDate create() {

/\*\*

\* Create a {@link LocalDate} of 2015-06-18 by using {@link LocalDate#of}

\*/

return null;

}

static LocalDate parse() {

/\*\*

\* Create a {@link LocalDate} of 2015-06-18 from String by using {@link LocalDate#parse}

\*/

return null;

}

static LocalDate with() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*\*

\* Create a {@link LocalDate} from {@link ld} with year 2015

\* by using {@link LocalDate#withYear} or {@link LocalDate#with}

\*/

return null;

}

static LocalDate withAdjuster() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*\*

\* Create a {@link LocalDate} from {@link ld} adjusted into first day of next year

\* by using {@link LocalDate#with} and {@link TemporalAdjusters#firstDayOfNextYear}

\*/

return null;

}

static LocalDate plus() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*\*

\* Create a {@link LocalDate} from {@link ld} with 10 month later

\* by using {@link LocalDate#plusMonths} or {@link LocalDate#plus}

\*/

return null;

}

static LocalDate minus() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*\*

\* Create a {@link LocalDate} from {@link ld} with 10 days before

\* by using {@link LocalDate#minusDays} or {@link LocalDate#minus}

\*/

return null;

}

static LocalDate plusPeriod() {

LocalDate ld = DateAndTimes.LD\_20150618;

/\*\*

\* Define a {@link Period} of 1 year 2 month 3 days

\* Create a {@link LocalDate} adding the period to {@link ld} by using {@link LocalDate#plus}

\*/

return null;

}

static boolean isAfter() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalDate ld2 = DateAndTimes.LD\_20150807;

/\*\*

\* Check whether {@link ld2} is after {@link ld} or not

\* by using {@link LocalDate#isAfter} or {@link LocalDate#isBefore}

\*/

return false;

}

static Period until() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalDate ld2 = DateAndTimes.LD\_20150807;

/\*\*

\* Create a period from {@link ld} till {@link ld2}

\* by using {@link LocalDate#until}

\*/

return null;

}

}

class DateAndTimes {

public static final LocalDate LD\_20150618 = LocalDate.of(2015, 6, 18);

public static final LocalDate LD\_20150807 = LocalDate.of(2015, 8, 7);

}

2.

LocalDateTime

**For example:**

| **Input** | **Result** |
| --- | --- |
| no input | 2015-06-20T23:07:30  2015-06-20T23:07:30  2015-07-01T23:00  2016-04-18T18:07:30.500  2015\_06\_18\_23\_07\_30  2015-06-1823:07:30.500  2015-06-18T23:07:30.500 2015-06-18T23:07:30.500 |

//package LocalDateTime;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.time.temporal.TemporalAdjusters;

/\*\*

\* LocalDateTime tests

\*/

public class LocalDateTimeTest {

public static void main(String[] args) {

System.out.println(localDateTimeOf());

System.out.println(localDateTimeParse());

System.out.println(localTimeWith());

System.out.println(localDatePlusMinus());

System.out.println(localDateTimeFormat());

System.out.println(toLocalDateAndTime());

System.out.println(toLocalDateTime());

}

static LocalDateTime localDateTimeOf() {

/\*\*

\* Create a {@link LocalDateTime} of 2015-06-20 23:07:30

\* by using {@link LocalDateTime#of}

\*/

LocalDateTime ld=LocalDateTime.of(2015,6,20,23,7,30);

return ld;

}

static LocalDateTime localDateTimeParse() {

/\*\*

\* Create a {@link LocalDateTime} of 2015-06-20 23:07:30

\* by using {@link LocalDateTime#parse}

\*/

return LocalDateTime.parse("2015-06-20T23:07:30");

}

static LocalDateTime localTimeWith() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDateTime} from {@link ldt}

\* with first day of the next month and

\* also truncated to hours.

\*/

return ldt.with(TemporalAdjusters.firstDayOfNextMonth())

.truncatedTo(ChronoUnit.HOURS);

}

static LocalDateTime localDatePlusMinus() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDateTime} from {@link ldt}

\* with 10 month later and 5 hours before

\* by using {@link LocalDateTime#plus\*} or

\* {@link LocalDateTime#minus\*}

\*/

return ldt.plusMonths(10).minusHours(5);

}

static String localDateTimeFormat() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Format {@link ldt} to a {@link String} as

\* "2015\_06\_18\_23\_07\_30"

\* by using {@link LocalDateTime#format} and

\* {@link DateTimeFormatter#ofPattern}

\*/

return ldt.format(DateTimeFormatter.ofPattern("yyyy\_MM\_dd\_HH\_mm\_ss"));

}

static String toLocalDateAndTime() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDate} and a

\* {@link LocalTime} from {@link ldt}

\* by using {@link LocalDateTime#toLocalDate}

\* and {@link LocalDateTime#toLocalTime}

\*/

LocalDate localDate =ldt.toLocalDate();

LocalTime localTime = ldt.toLocalTime();

return localDate.toString() + localTime.toString();

}

static String toLocalDateTime() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create two equal {@link LocalDateTime}

\* from {@link ld} and {@link lt}

\* by using {@link LocalDate#atTime} and

\* {@link LocalTime#atDate}

\*/

LocalDateTime localDateTime1 = ld.atTime(lt);

LocalDateTime localDateTime2 = lt.atDate(ld);

return localDateTime1.toString() + " " + localDateTime2.toString();

}

static class DateAndTimes {

public static final LocalDate LD\_20150618 = LocalDate.of(2015, 6, 18);

public static final LocalTime LT\_23073050 = LocalTime.of(23, 7, 30, 500000000);

public static final LocalDateTime LDT\_20150618\_23073050 = LocalDateTime.of(2015, 6, 18, 23, 7, 30, 500000000);

}

}

СТАРТЕР:

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.time.temporal.TemporalAdjusters;

/\*\*

\* LocalDateTime tests

\*/

public class LocalDateTimeTest {

public static void main(String[] args) {

System.out.println(localDateTimeOf());

System.out.println(localDateTimeParse());

System.out.println(localTimeWith());

System.out.println(localDatePlusMinus());

System.out.println(localDateTimeFormat());

System.out.println(toLocalDateAndTime());

System.out.println(toLocalDateTime());

}

static LocalDateTime localDateTimeOf() {

/\*\*

\* Create a {@link LocalDateTime} of 2015-06-20 23:07:30 by using {@link LocalDateTime#of}

\*/

return null;

}

static LocalDateTime localDateTimeParse() {

/\*\*

\* Create a {@link LocalDateTime} of 2015-06-20 23:07:30 by using {@link LocalDateTime#parse}

\*/

return null;

}

static LocalDateTime localTimeWith() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDateTime} from {@link ldt}

\* with first day of the next month and also truncated to hours.

\*/

return null;

}

static LocalDateTime localDatePlusMinus() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDateTime} from {@link ldt} with 10 month later and 5 hours before

\* by using {@link LocalDateTime#plus\*} or {@link LocalDateTime#minus\*}

\*/

return null;

}

static String localDateTimeFormat() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Format {@link ldt} to a {@link String} as "2015\_06\_18\_23\_07\_30"

\* by using {@link LocalDateTime#format} and {@link DateTimeFormatter#ofPattern}

\*/

return null;

}

static String toLocalDateAndTime() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDate} and a {@link LocalTime} from {@link ldt}

\* by using {@link LocalDateTime#toLocalDate} and {@link LocalDateTime#toLocalTime}

\*/

LocalDate localDate = null;

LocalTime localTime = null;

return localDate.toString() + localTime.toString();

}

static String toLocalDateTime() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create two equal {@link LocalDateTime} from {@link ld} and {@link lt}

\* by using {@link LocalDate#atTime} and {@link LocalTime#atDate}

\*/

LocalDateTime localDateTime1 = null;

LocalDateTime localDateTime2 = null;

return localDateTime1.toString() + " " + localDateTime2.toString();

}

static class DateAndTimes {

public static final LocalDate LD\_20150618 = LocalDate.of(2015, 6, 18);

public static final LocalTime LT\_23073050 = LocalTime.of(23, 7, 30, 500000000);

public static final LocalDateTime LDT\_20150618\_23073050 = LocalDateTime.of(2015, 6, 18, 23, 7, 30, 500000000);

}

}

РЕШЕНИЕ:

package lab4\_task2\_LocalDateTime;

//package LocalDateTime;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.LocalTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.time.temporal.TemporalAdjusters;

/\*\*

\* LocalDateTime tests

\*/

public class LocalDateTimeTest {

public static void main(String[] args) {

System.out.println(localDateTimeOf());

System.out.println(localDateTimeParse());

System.out.println(localTimeWith());

System.out.println(localDatePlusMinus());

System.out.println(localDateTimeFormat());

System.out.println(toLocalDateAndTime());

System.out.println(toLocalDateTime());

}

static LocalDateTime localDateTimeOf() {

/\*\*

\* Create a {@link LocalDateTime} of 2015-06-20 23:07:30

\* by using {@link LocalDateTime#of}

\*/

LocalDateTime ld=LocalDateTime.of(2015,6,20,23,7,30);

return ld;

}

static LocalDateTime localDateTimeParse() {

/\*\*

\* Create a {@link LocalDateTime} of 2015-06-20 23:07:30

\* by using {@link LocalDateTime#parse}

\*/

return LocalDateTime.parse("2015-06-20T23:07:30");

}

static LocalDateTime localTimeWith() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDateTime} from {@link ldt}

\* with first day of the next month and

\* also truncated to hours.

\*/

return ldt.with(TemporalAdjusters.firstDayOfNextMonth())

.truncatedTo(ChronoUnit.HOURS);

}

static LocalDateTime localDatePlusMinus() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDateTime} from {@link ldt}

\* with 10 month later and 5 hours before

\* by using {@link LocalDateTime#plus\*} or

\* {@link LocalDateTime#minus\*}

\*/

return ldt.plusMonths(10).minusHours(5);

}

static String localDateTimeFormat() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Format {@link ldt} to a {@link String} as

\* "2015\_06\_18\_23\_07\_30"

\* by using {@link LocalDateTime#format} and

\* {@link DateTimeFormatter#ofPattern}

\*/

return ldt.format(DateTimeFormatter.ofPattern("yyyy\_MM\_dd\_HH\_mm\_ss"));

}

static String toLocalDateAndTime() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link LocalDate} and a

\* {@link LocalTime} from {@link ldt}

\* by using {@link LocalDateTime#toLocalDate}

\* and {@link LocalDateTime#toLocalTime}

\*/

LocalDate localDate =ldt.toLocalDate();

LocalTime localTime = ldt.toLocalTime();

return localDate.toString() + localTime.toString();

}

static String toLocalDateTime() {

LocalDate ld = DateAndTimes.LD\_20150618;

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create two equal {@link LocalDateTime}

\* from {@link ld} and {@link lt}

\* by using {@link LocalDate#atTime} and

\* {@link LocalTime#atDate}

\*/

LocalDateTime localDateTime1 = ld.atTime(lt);

LocalDateTime localDateTime2 = lt.atDate(ld);

return localDateTime1.toString() + " " + localDateTime2.toString();

}

static class DateAndTimes {

public static final LocalDate LD\_20150618 = LocalDate.of(2015, 6, 18);

public static final LocalTime LT\_23073050 = LocalTime.of(23, 7, 30, 500000000);

public static final LocalDateTime LDT\_20150618\_23073050 = LocalDateTime.of(2015, 6, 18, 23, 7, 30, 500000000);

}

}

3. LocalTime API

**For example:**

| **Input** | **Result** |
| --- | --- |
| no input | 23:07  23:07:03.100  23:07:03.100  21:07:30.500  23:37:30.500  20:07:30.500  19:37:10.300  true  23:07 |

СТАРТЕР:

import java.time.Duration;

import java.time.LocalTime;

import java.time.temporal.ChronoField;

import java.time.temporal.ChronoUnit;

/\*\*

\* LocalTime API tests

\*/

public class LocalTimeTest {

public static void main(String[] args) {

System.out.println(localTimeOfHourToMinute());

System.out.println(localTimeOfHourToNanoSec());

System.out.println(localTimeParse());

System.out.println(localTimeWith());

System.out.println(localTimePlus());

System.out.println(localTimeMinus());

System.out.println(localTimeMinusDuration());

System.out.println(localDateIsBefore());

System.out.println(localTimeTruncatedTo());

}

static LocalTime localTimeOfHourToMinute() {

/\*\*

\* Create a {@link LocalTime} of 23:07 by using {@link LocalTime#of}

\*/

return null;

}

static LocalTime localTimeOfHourToNanoSec() {

/\*\*

\* Create a {@link LocalTime} of 23:07:03.1 by using {@link LocalTime#of}

\*/

return null;

}

static LocalTime localTimeParse() {

/\*\*

\* Create a {@link LocalTime} of 23:07:03.1 from String by using {@link LocalTime#parse}

\*/

return null;

}

static LocalTime localTimeWith() {

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create a {@link LocalTime} from {@link lt} with hour 21

\* by using {@link LocalTime#withHour} or {@link LocalTime#with}

\*/

return null;

}

static LocalTime localTimePlus() {

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create a {@link LocalTime} from {@link lt} with 30 minutes later

\* by using {@link LocalTime#plusMinutes} or {@link LocalTime#plus}

\*/

return null;

}

static LocalTime localTimeMinus() {

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create a {@link LocalTime} from {@link lt} with 3 hours before

\* by using {@link LocalTime#minusHours} or {@link LocalTime#minus}

\*/

return null;

}

static LocalTime localTimeMinusDuration() {

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Define a {@link Duration} of 3 hours 30 minutes and 20.2 seconds

\* Create a {@link LocalTime} subtracting the duration from {@link lt} by using {@link LocalTime#minus}

\*/

return null;

}

static boolean localDateIsBefore() {

LocalTime lt = DateAndTimes.LT\_23073050;

LocalTime lt2 = DateAndTimes.LT\_12100000;

/\*\*

\* Check whether {@link lt2} is before {@link lt} or not

\* by using {@link LocalTime#isAfter} or {@link LocalTime#isBefore}

\*/

return false;

}

static LocalTime localTimeTruncatedTo() {

LocalTime lt = DateAndTimes.LT\_23073050;

/\*\*

\* Create a {@link LocalTime} from {@link lt} truncated to minutes by using {@link LocalTime#truncatedTo}

\*/

return null;

}

static class DateAndTimes {

public static final LocalTime LT\_23073050 = LocalTime.of(23, 7, 30, 500000000);

public static final LocalTime LT\_12100000 = LocalTime.of(12, 10);

}

}

4. ZonedDateTime

**For example:**

| **Input** | **Result** |
| --- | --- |
| no input | 2015-07-10T02:14:25+09:00[Asia/Tokyo]  2015-06-18T23:07:25+09:00[Asia/Tokyo]  2015\_06\_18\_23\_07\_30\_JST  2015-06-18T23:07:30.500-07:00[America/Los\_Angeles]  2015-06-18T07:07:30.500-07:00[America/Los\_Angeles]  2015-06-18T23:07:30.500-07:00[America/Los\_Angeles] |

import java.time.LocalDateTime;

import java.time.ZoneId;

import java.time.ZonedDateTime;

import java.time.format.DateTimeFormatter;

/\*\*

\* ZonedDateTime tests

\*/

public class ZonedDateTimeTest {

public static void main(String[] args) {

System.out.println(zonedDateTimeOf());

System.out.println(zonedDateTimeParse());

System.out.println(zonedDateTimeFormat());

System.out.println(toPST());

System.out.println(sameInstantAs());

System.out.println(sameLocalAs());

}

static ZonedDateTime zonedDateTimeOf() {

/\*\*

\* Create a {@link ZonedDateTime} with time of 2015-07-10 2:14:25.000 as Japan Standard Time

\* by using {@link ZonedDateTime#of} and {@link ZoneId#of}

\*/

return null;

}

static ZonedDateTime zonedDateTimeParse() {

/\*\*

\* Create a {@link ZonedDateTime} with time of 2015-06-18 23:07:25.000 as Japan Standard Time

\* by using {@link ZonedDateTime#parse}

\*/

return null;

}

static String zonedDateTimeFormat() {

ZonedDateTime zdt = DateAndTimes.ZDT\_20150618\_23073050;

/\*\*

\* Format {@link zdt} to a {@link String} as "2015\_06\_18\_23\_07\_30\_JST"

\* by using {@link ZonedDateTime#format}

\*/

return null;

}

static ZonedDateTime toPST() {

LocalDateTime ldt = DateAndTimes.LDT\_20150618\_23073050;

/\*\*

\* Create a {@link ZonedDateTime} from {@link ldt} with Pacific Standard Time

\*/

return null;

}

static ZonedDateTime sameInstantAs() {

ZonedDateTime zdt = DateAndTimes.ZDT\_20150618\_23073050;

/\*\*

\* Create a {@link ZonedDateTime} same instant as {@link zdt} with Pacific Standard Time

\* by using {@link ZonedDateTime#withZoneSameInstant}

\*/

return null;

}

static ZonedDateTime sameLocalAs() {

ZonedDateTime zdt = DateAndTimes.ZDT\_20150618\_23073050;

/\*\*

\* Create a {@link ZonedDateTime} same local time as {@link zdt} with Pacific Standard Time

\* by using {@link ZonedDateTime#withZoneSameLocal}

\*/

return null;

}

static class DateAndTimes {

public static final LocalDateTime LDT\_20150618\_23073050 = LocalDateTime.of(2015, 6, 18, 23, 7, 30, 500000000);

public static final ZonedDateTime

ZDT\_20150618\_23073050 = ZonedDateTime.of(LDT\_20150618\_23073050, ZoneId.of("Asia/Tokyo"));

}

}