Sofia University
Department of Mathematics and Informatics

Course : Applied OO Programming part 1

Date: May 26, 2020

Student Name:

Lab No. 14

Problem 1

Use the **`class Invoice`** provided with the sample code in the **`Lab`** folder to **create an array of `Invoice` objects**.

Use the sample data shown below.

Part number	Part description	Quantity	Price
83	Electric sander	7	57.98
24	Power saw	18	99.99
7	Sledge hammer	11	21.50
77	Hammer	76	11.99
39	Lawn mower	3	79.50
68	Screwdriver	106	6.99
56	Jig saw	21	11.00
3	Wrench	34	7.50

Class **`Invoice`** includes four properties- a **`partNumber`** (type int), a **`partDescription`** (type **`String`**), a **`quantity`** of the item being purchased (type **`int`**) and a **`price`** (type **`double`**). Perform the following queries on the array of **`Invoice`** objects and display the results:

a) Use lambdas and streams to sort the **`Invoice`** objects by **`partDescription`**, then display the results.

b) Use lambdas and streams to sort the **`Invoice`** objects by **`price`**, then display the results.

c) Use lambdas and streams to **`map`** each **`Invoice`** to its **`partDescription`** and **`quantity`**, sort the results by **`quantity`**, then display the results.

d) Use lambdas and streams to **map** each **Invoice** to its **partDescription** and the value of the Invoice (i.e., **quantity * price**). Order the results by **Invoice** value.

e) Modify Part (d) to select the **Invoice** values in the range **$200** to **$500**.

f) Group the **Invoice** values into two sets of Invoices- Invoices with values (**quantity * price**) below of equal to **$300** and Invoices with values above **$300**.

g) Create a **Map<String, Invoice>,** where the **Key** id the hashcode of the **Value**. Add the elements of the above created array to this **Map** and output the elements of the map sorted in decreasing order of the **price.**

Problem 2

Write a program that inputs a sentence from the user (assume no punctuation), then determines and displays the unique words in alphabetical order. Treat uppercase and lowercase letters the same.

Problem 3

Write a program that inserts **30** random letters into a **List<Character>**. Perform the following operations and display your results:

a) Sort the **List** in ascending order.

b) Sort the **List** in descending order.

c) Display the **List** in ascending order with duplicates removed. Write a program that inserts **30** random letters into a **List<Character>**.

Problem 4

Write a Stream application to roll a die 6,000,000 times and display a table with the frequencies each side of the die has occurred in that sequence as in the following sample output

```
Face   Frequency
1      999549
2      1001189
3      999596
4      999672
5      998597
6      1001397
```

Use grouping by the side number and count the occurrences in each group

Problem 5

The lambda you pass to a stream's reduce method should be *associative*- that is, regardless of the order in which its subexpressions are evaluated, the result should be the same. The underlined lambda expression in the following code is *not* associative.

```
import java.util.Arrays;
import java.util.stream.IntStream;

public class IntStreamOperations
{
   public static void main(String[] args)
   {
      int[] values = {3, 10, 6, 1, 4, 8, 2, 5, 9, 7};

      // display original values
      System.out.print("Original values: ");
      IntStream.of(values)
               .forEach(value -> System.out.printf("%d ", value));
      System.out.println();

      // count, min, max, sum and average of the values
      System.out.printf("%nCount: %d%n",
         IntStream.of(values).count());
      System.out.printf("Min: %d%n",
         IntStream.of(values).min().getAsInt());
      System.out.printf("Max: %d%n",
         IntStream.of(values).max().getAsInt());
      System.out.printf("Sum: %d%n", IntStream.of(values).sum());
      System.out.printf("Average: %.2f%n",
         IntStream.of(values).average().getAsDouble());

      // sum of values with reduce method
      System.out.printf("%nSum via reduce method: %d%n",
         IntStream.of(values)
                  .reduce(0, (x, y) -> x + y));

      // sum of squares of values with reduce method
      System.out.printf("Sum of squares via reduce method: %d%n",
```

```
            IntStream.of(values).parallel()
                     .reduce(0, (x, y) -> x + y * y));

         // product of values with reduce method
         System.out.printf("Product via reduce method: %d%n",
            IntStream.of(values)
                     .reduce(1, (x, y) -> x * y));

         // even values displayed in sorted order
         System.out.printf("%nEven values displayed in sorted order: ");
         IntStream.of(values)
                  .filter(value -> value % 2 == 0)
                  .sorted()
                  .forEach(value -> System.out.printf("%d ", value));
         System.out.println();

         // odd values multiplied by 10 and displayed in sorted order
         System.out.printf(
            "Odd values multiplied by 10 displayed in sorted order: ");
         IntStream.of(values)
                  .filter(value -> value % 2 != 0)
                  .map(value -> value * 10)
                  .sorted()
                  .forEach(value -> System.out.printf("%d ", value));
         System.out.println();

         // sum range of integers from 1 to 10, exlusive
         System.out.printf("%nSum of integers from 1 to 9: %d%n",
            IntStream.range(1, 10).sum());

         // sum range of integers from 1 to 10, inclusive
         System.out.printf("Sum of integers from 1 to 10: %d%n",
            IntStream.rangeClosed(1, 10).sum());
      }
   } // end class IntStreamOperations
```

When you create parallel streams using the **`parallel()`** operation with that lambda, you might get incorrect results for the sum of the squares, depending on the order in which the subexpressions are evaluated. The proper way to implement the summation of the squares would be *first* to map each int value to the square of that value, *then* to reduce the stream to the sum of the squares. Modify the above program to implement the summation of squares with **`reduce()`** in this manner.

Задача 5

Напишете клас `LambdaTest` и извършете следните действия

a) Напишете подходящи функционални интерфейси за всеки от следните Ламбда изрази:

- `(Integer x, Integer y) -> x + y`
- `(String s1, String s2) -> return String.format("%s, %s", s1, s2)`
- `(Double d1, Double d2) -> {`

```
                        Scanner input = new
                        Scanner(System.in);

                        double d3 =   input.nextDouble();

                        return Math.max(d1,Math.max(d1,d2));

                    };
```

b) Имплементирайте изразите в а) като анонимни класове с помощта на така предложените функционални интерфейси

c) **Напишете в `public static void main(String[] args)` метода на клас `LambdaTest`**

- изпълнете Ламбда изразите, посредством съответните им анонимни класове, а резултатът от изпълнението им да се изведе на стандартния изход
- команди за присвояване на изразите в а) на променливи от подходящо избран **стандартен** функционален интерфейс на Java. (използвайте параметър за тип)
- изпълнете Ламбда изразите, посредством така инициализираните променливи на **стандартен** функционален интерфейс, а резултатът от изпълнението им да се изведе на стандартния изход
- изпълнете Ламбда изразите, посредством така инициализираните променливи, а резултата от изпълнението им да се изведе на стандартния

d) Решете следната задача посредством Ламбда изрази и стандартни функционални интерфейси.

- **Напишете** ***class MySort***, който има статичен **метод** ***SortedSet<Integer> sort(int[] data, Comparator<Integer > sortOrder).***

  Нека този метод връща сортирано множество от различните елементи на масива ***data*** , при което функционалният интерефйс ***sortOrder*** задава на наредбата на

елементите при сортиране. Използвайте клас TreeSet за реализация на сортирането

- Напишете ***class UseSort*** с метод ***main()*** за тестване на ***class MySort***.
  Дефинирайте един **масив *numbers* от цели числа** и **създайте обекти** от class ***MySort.***
  Дефинирайте два Ламбда израза от тип ***Comparator<Integer >*** съответно реферирани като ***upward*** и ***downward*** за наредба във възходящ ред и наредба в низходящ ред.

  Изведете на **стандартен изход** елементите на **зададения масив** и **същия масив след от изпълнението на метода *MySort*.sort() при сортирането на масива *numbers*** във **възходящ** и **низходящ** ред съответно с Ламбда изразите ***upward*** и do***w***nward.

Задача 6

Напишете в **`class ArrayUtils`** метод
`static void filterNumbers(Predicate<Integer> condition, int[] array),`
който **извежда на стандартен изхо**д елементите на масива **`array`**, за които Ламбда израза **`condition`** връща **`true`**

Създайте в **`public static void main(String[] args)`** метода на клас **`ArrayUtils`** масив **`numbers`** от **20 цели числа** с генератор за случайни числа в интервала [10- 50].
Напишете в този метод Ламбда изрази от тип **`Predicate<Integer>`**, които позволяват

a) да се изведат четните числа на масива **`numbers`**
b) да се изведат числата в интервала [30- 40]
c) да се изведат простите числа на масива **`numbers`**

Изпълнете метода **`filterNumbers()`** с всеки от тези Ламбда изрази

Задача 7

Ако list **е** List<Integer>, **обяснете верижно Stream изпълнение на следната команда**

```
list.stream()
.filter(value -> value % 2 != 0)
.reduce(0, (x,y)-> x+ y);
```

Задача 8

Преобразувайте следния блок от сорс код към `Streams API`, за да се реализира външно итериране по елементите на масива `artists`

```
int totalMembers = 0;
for (Artist artist : artists) {
   Stream<Artist> members = artist.getMembers();
   totalMembers += members.count();
}
```

Тествайте получения сорс код като използвате примерния сорс код Lecture9bSampleCode.rar (в директория StreamsSamples) за `class Artist` и `class SampleData`.