Associative Arrays, Lambda and Stream API

Collections and Queries

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Questions?

Software University

sli.do

#fund-java

Table of Contents

Associative Arrays

Collection of Key and Value Pairs

Associative Arrays (Maps)

- Associative arrays are arrays indexed by **keys**
  - Not by the numbers 0, 1, 2, … (like arrays)
- Hold a set of pairs **{key → value}**

Key	Value
John Smith	+1-555-8976
Lisa Smith	+1-555-1234
Sam Doe	+1-555-5030

Collections of Key and Value Pairs

Software University

- **HashMap**<K, V>
  - Keys are **unique**
  - Uses a **hash-table** + **list**
- **LinkedHashMap**<K, V>
  - Keys are **unique**
  - Keeps the keys in **order of addition**
- **TreeMap**<K, V>
  - Keys are **unique**
  - Keeps its **keys always sorted**
  - Uses a **balanced search tree**

Built-In Methods

- `put(key, value)` method

```
HashMap<String, Integer> airplanes = new HashMap<>();
airplanes.put("Boeing 737", 130);
airplanes.put("Airbus A320", 150);
```

- `remove(key)` method

```
HashMap<String, Integer> airplanes = new HashMap<>();
airplanes.put("Boeing 737", 130);
airplanes.remove("Boeing 737");
```

Built-In Methods (2)

- containsKey(key)

```
HashMap<String, Integer> map = new HashMap<>();
map.put("Airbus A320", 150);
if (map.containsKey("Airbus A320"))
   System.out.println("Airbus A320 key exists");
```

- containsValue(value)

```
HashMap<String, Integer> map = new HashMap<>();
map.put("Airbus A320", 150);
System.out.println(map.containsValue(150)); //true
System.out.println(map.containsValue(100)); //false
```

HashMap: Put()

Pesho	0881-123-987
Gosho	0881-123-789
Alice	0881-123-978

Hash Function

HashMap<String, String>

Key Value

HashMap: Remove()

Pesho

Hash Function

HashMap<String, String>

Pesho	0881-123-987
Gosho	0881-123-789
Alice	0881-123-978

Key Value

TreeMap<K, V> – Example

Pesho	0881-123-987
Alice	+359-899-55-592

Comparator Function

TreeMap <String, String>

Key Value

Iterating Through Map

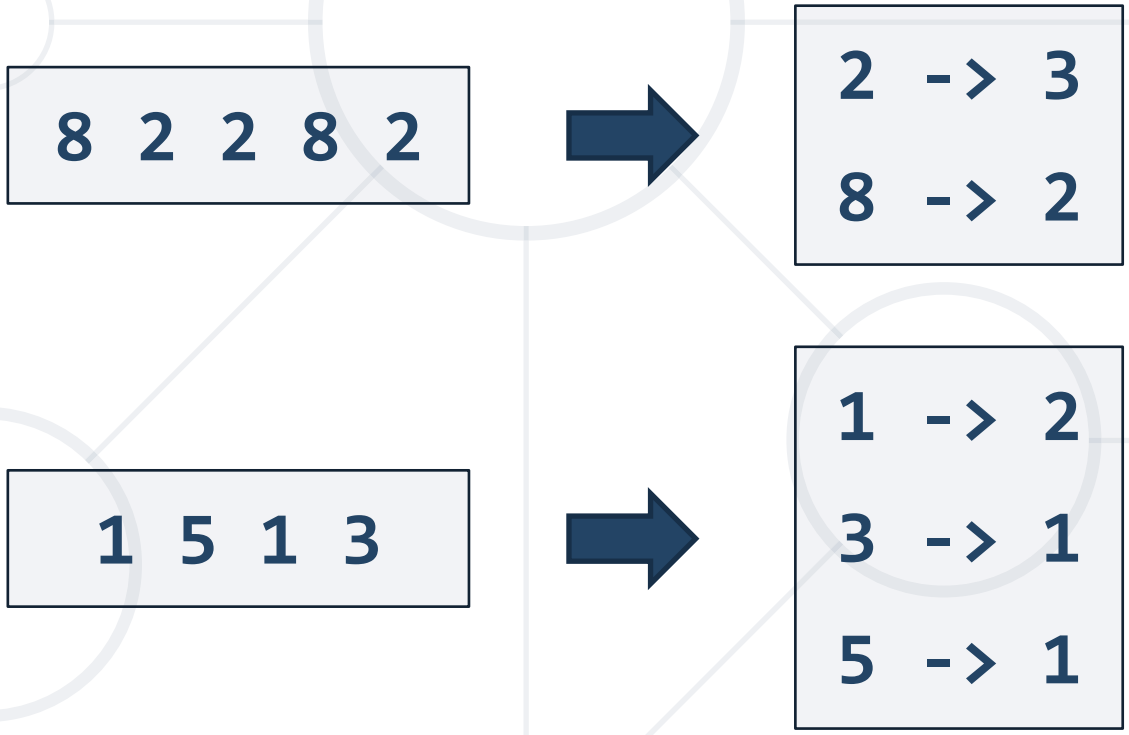
- Iterate through objects of type `Map.Entry<K, V>`
- Cannot modify the collection (**read-only**)

```
Map<String, Double> fruits = new LinkedHashMap<>();
fruits.put("banana", 2.20);
fruits.put("kiwi", 4.50);
for (Map.Entry<K, V> entry : fruits.entrySet()) {
System.out.printf("%s -> %.2f%n",
                  entry.getKey(), entry.getValue());
}
```

entry.getKey() -> fruit name
entry.getValue() -> fruit price

Problem: Count Real Numbers

- Read a list of real numbers and print them in ascending order along with their number of occurrences



Check your solution here: https://judge.softuni.bg/Contests/1311/

Solution: Count Real Numbers

```
double[] nums = Arrays.stream(sc.nextLine().split(" "))
                .mapToDouble(Double::parseDouble).toArray();
Map<Double, Integer> counts = new TreeMap<>();
for (double num : nums) {
  if (!counts.containsKey(num))
    counts.put(num, 0);
  counts.put(num, counts.get(num) + 1);
}
for (Map.Entry<Double, Integer> entry : counts.entrySet()) {
  DecimalFormat df = new DecimalFormat("#.#######");
  System.out.printf("%s -> %d%n", df.format(entry.getKey()), entry.getValue());
}
```

Overwrite the value

Check your solution here: https://judge.softuni.bg/Contests/1311/

Problem: Words Synonyms

- Read **2 * N** lines of pairs **word** and **synonym**
- Each word may have **many** synonyms

```
3
cute
adorable
cute
charming
smart
clever
```

```
cute - adorable, charming
smart - clever
```

Check your solution here: https://judge.softuni.bg/Contests/1311/

Solution: Word Synonyms

```
int n = Integer.parseInt(sc.nextLine());
Map<String, ArrayList<String>> words = new LinkedHashMap<>();
for (int i = 0; i < n; i++) {
  String word = sc.nextLine();
  String synonym = sc.nextLine();
  words.putIfAbsent(word, new ArrayList<>());
  words.get(word).add(synonym);
}
//TODO: Print each word and synonyms
```

Adding the key if it does not exist

Check your solution here: https://judge.softuni.bg/Contests/1311/

Lambda Expressions

Anonymous Functions

Lambda Functions

- A lambda expression is an anonymous function containing expressions and statements

```
(a -> a > 5)
```

- Lambda expressions
- Use the lambda operator **->**
  - Read as "**goes to**"
- The **left** side specifies the **input** parameters
- The **right** side holds the **expression** or **statement**

Lambda Functions

- Lambda functions are **inline methods** (functions) that take input parameters and return values:

```
x -> x / 2
```

```
static int func(int x) { return x / 2; }
```

```
x -> x != 0
```

```
static boolean func(int x) { return x != 0; }
```

```
() -> 42
```

```
static int func() { return 42; }
```

Stream API

Traversing and Querying Collections

Processing Arrays with Stream API (1)

Software University

- `min()` - finds the **smallest** element in a collection:

```
int min = Arrays.stream(new int[]{15, 25, 35}).min().getAsInt();
```

15

```
int min = Arrays.stream(new int[]{15, 25, 35}).min().orElse(2);
```

```
int min = Arrays.stream(new int[]{}).min().orElse(2); // 2
```

- `max()` - finds the **largest** element in a collection:

```
int max = Arrays.stream(new int[]{15, 25, 35}).max().getAsInt();
```

35

Processing Arrays with Stream API (2)

- `sum()` - finds the **sum** of all elements in a collection:

```
int sum = Arrays.stream(new int[]{15, 25, 35}).sum();
```

75

- `average()` - finds the **average** of all elements:

25.0

```
double avg = Arrays.stream(new int[]{15, 25, 35})
                   .average().getAsDouble();
```

Processing Collections with Stream API (1)

```
ArrayList<Integer> nums = new ArrayList<>() {{
    add(15); add(25); add(35);
}};
```

- min()

```
int min = nums.stream().mapToInt(Integer::intValue)
               .min().getAsInt();
```

15

```
int min = nums.stream()
               .min(Integer::compareTo).get();
```

Processing Collections with Stream API (2)

- max()

```
int max = nums.stream().mapToInt(Integer::intValue)
               .max().getAsInt();
```

35

```
int max = nums.stream()
               .max(Integer::compareTo).get();
```

- sum()

```
int sum = nums.stream()
               .mapToInt(Integer::intValue).sum();
```

75

Processing Collections with Stream API (3)

Software University

- average()

```
double avg = nums.stream()
                 .mapToInt(Integer::intValue)
                 .average()
                 .getAsDouble();
```

25.0

Manipulating Collections

- `map()` - manipulates elements in a collection:

```
int[] nums = Arrays.stream(sc.nextLine().split(" "))
            .mapToInt(e -> Integer.parseInt(e))
            .toArray();
```

Parse each element to Integer

```
String[] words = {"abc", "def", "geh", "yyy"};
words = Arrays.stream(words)
              .map(w -> w + "yyy")
              .toArray(String[]::new);
//abcyyy, defyyy, gehyyy, yyyyyy
```

Converting Collections

- Using `toArray()`, `toList()` to convert collections:

```
int[] nums = Arrays.stream(sc.nextLine().split(" "))
                   .mapToInt(e -> Integer.parseInt(e))
                   .toArray();
```

```
List<Integer> nums = Arrays.stream(sc.nextLine()
                           .split(" "))
                           .map(e -> Integer.parseInt(e))
                           .collect(Collectors.toList());
```

Filtering Collections

- Using `filter()`

```
int[] nums = Arrays.stream(sc.nextLine().split(" "))
                   .mapToInt(e -> Integer.parseInt(e))
                   .filter(n -> n > 0)
                   .toArray();
```

Problem: Word Filter

- Read a string array
- Print only words which length is even

```
kiwi orange banana apple
```

```
kiwi
orange
banana
```

```
pizza cake pasta chips
```

```
cake
```

Check your solution here: https://judge.softuni.bg/Contests/1311/

Solution: Word Filter

```
String[] words = Arrays.stream(sc.nextLine().split(" "))
                .filter(w -> w.length() % 2 == 0)
                .toArray(String[]::new);

for (String word : words) {
  System.out.println(word);
}
```

Check your solution here: https://judge.softuni.bg/Contests/1311/

Sorting Collections

- Using `sorted()` to sort collections:

```
nums = nums.stream()
           .sorted((n1, n2) -> n1.compareTo(n2))
           .collect(Collectors.toList());
```

Ascending (Natural) Order

```
nums = nums.stream()
           .sorted((n1, n2) -> n2.compareTo(n1))
           .collect(Collectors.toList());
```

Descending Order

Sorting Collections by Multiple Criteria

- Using `sorted()` to sort collections by multiple criteria:

```
Map<Integer, String> products = new HashMap<>();
products.entrySet()
        .stream()
        .sorted((e1, e2) -> {
            int res = e2.getValue().compareTo(e1.getValue());
            if (res == 0)
                res = e1.getKey().compareTo(e2.getKey());
            return res; })
        .forEach(e -> System.out.println(e.getKey() + " " + e.getValue()));
```

Second criteria

Terminates the stream

Using Functional ForEach (1)

```
Map<String, ArrayList<Integer>> arr = new HashMap<>();
arr.entrySet().stream()
    .sorted((a, b) -> {
      if (a.getKey().compareTo(b.getKey()) == 0) {
        int sumFirst = a.getValue().stream().mapToInt(x -> x).sum();
        int sumSecond = b.getValue().stream().mapToInt(x -> x).sum();
        return sumFirst - sumSecond;
      }
      return b.getKey().compareTo(a.getKey());
    })
```

Second criteria

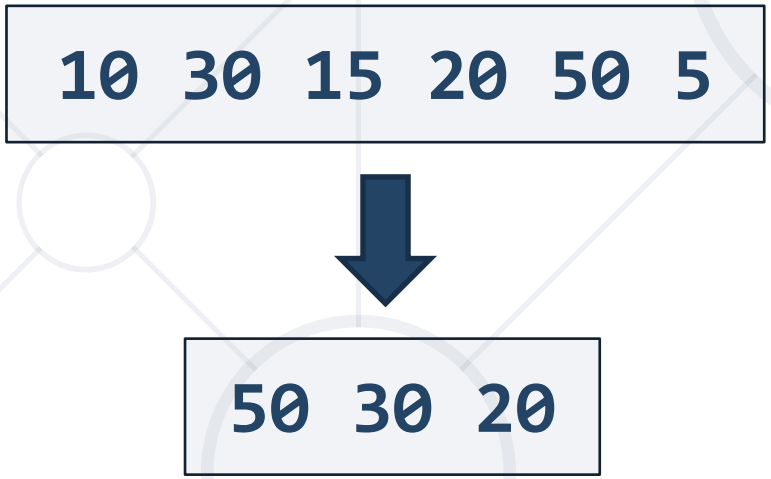
Descending sorting

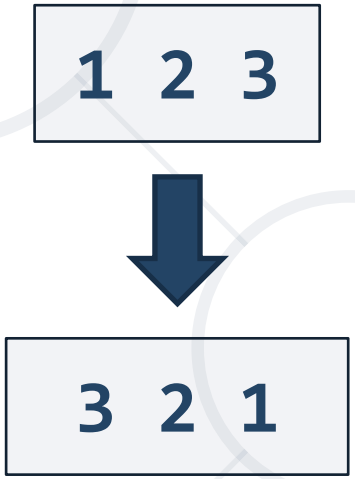
Using Functional ForEach (2)

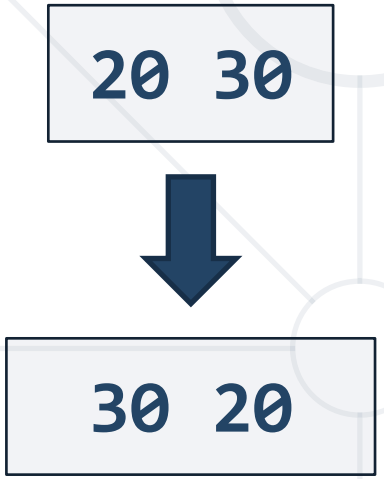
```
.forEach(pair -> {
  System.out.println("Key: " + pair.getKey());
  System.out.print("Value: ");
  pair.getValue().sort((a, b) -> a.compareTo(b));
  for (int num : pair.getValue()) {
    System.out.printf("%d ", num);
  }
  System.out.println();
});
```

Problem: Largest 3 Numbers

- Read a list of numbers
- Print largest 3, if there are less than 3, print all of them







Check your solution here: https://judge.softuni.bg/Contests/1311/

Solution: Largest 3 Numbers

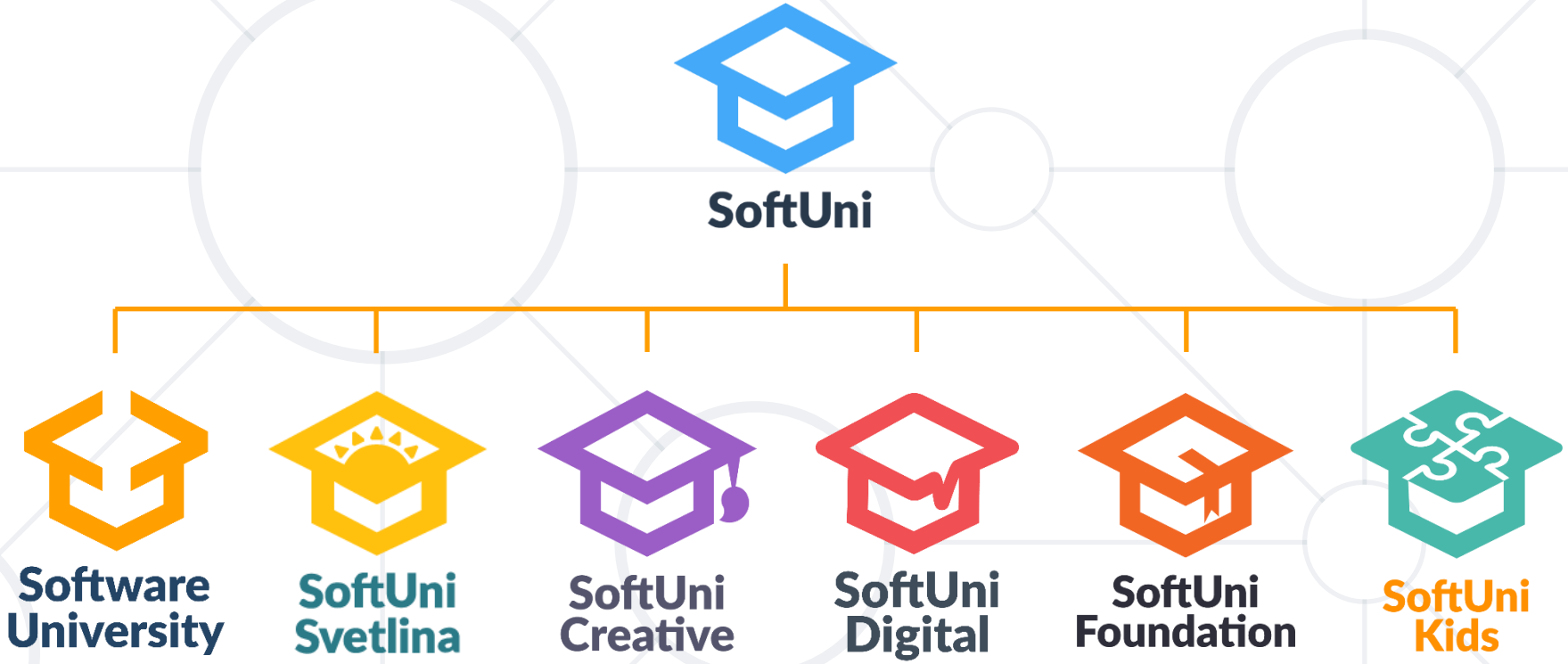
```
List<Integer> nums = Arrays
                .stream(sc.nextLine().split(" "))
                .map(e -> Integer.parseInt(e))
                .sorted((n1, n2) -> n2.compareTo(n1))
                .limit(3)
                .collect(Collectors.toList());
for (int num : nums) {
            System.out.print(num + " ");
        }
```

Check your solution here: https://judge.softuni.bg/Contests/1311/

Summary

- Maps hold **{key → value}** pairs
  - **Keyset** holds a set of **unique keys**
  - **Values** hold a collection of values
  - Iterating over a map
    takes the entries as **Map.Entry<K, V>**
- **Lambda** and **Stream API** help
  collection processing

Questions?



Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg