

Sets and Maps

Java Collections API – Sets and Maps



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

sli.do

#java-advanced

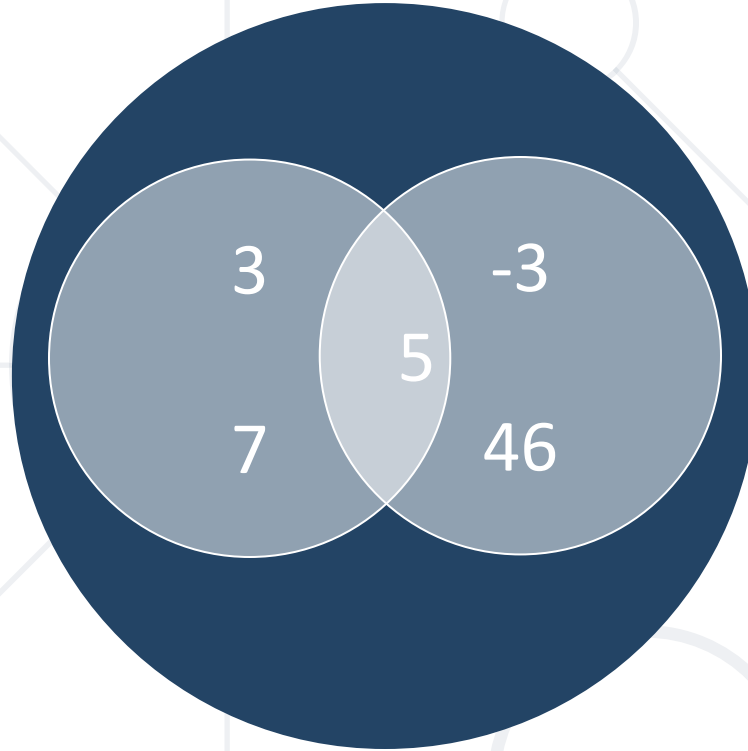
1. Sets

- `HashSet<E>`
- `TreeSet<E>`
- `LinkedHashSet<E>`

2. Maps

- `HashMap<K, V>`
- `TreeMap<K, V>`
- `LinkedHashMap<K, V>`





Sets

HashSet<E>, TreeSet<E> and LinkedHashSet<E>

- A set keeps unique elements
- Provides methods for **adding/removing/searching** elements
- Offers very fast performance
- Types:
 - **HashSet<E>**
 - Does not guarantee the constant order of elements over time
 - **TreeSet<E>**
 - The elements are ordered incrementally
 - **LinkedHashSet<E>**
 - The order of appearance is preserved

- Initialization:

```
Set<String> hash = new HashSet<String>();
```

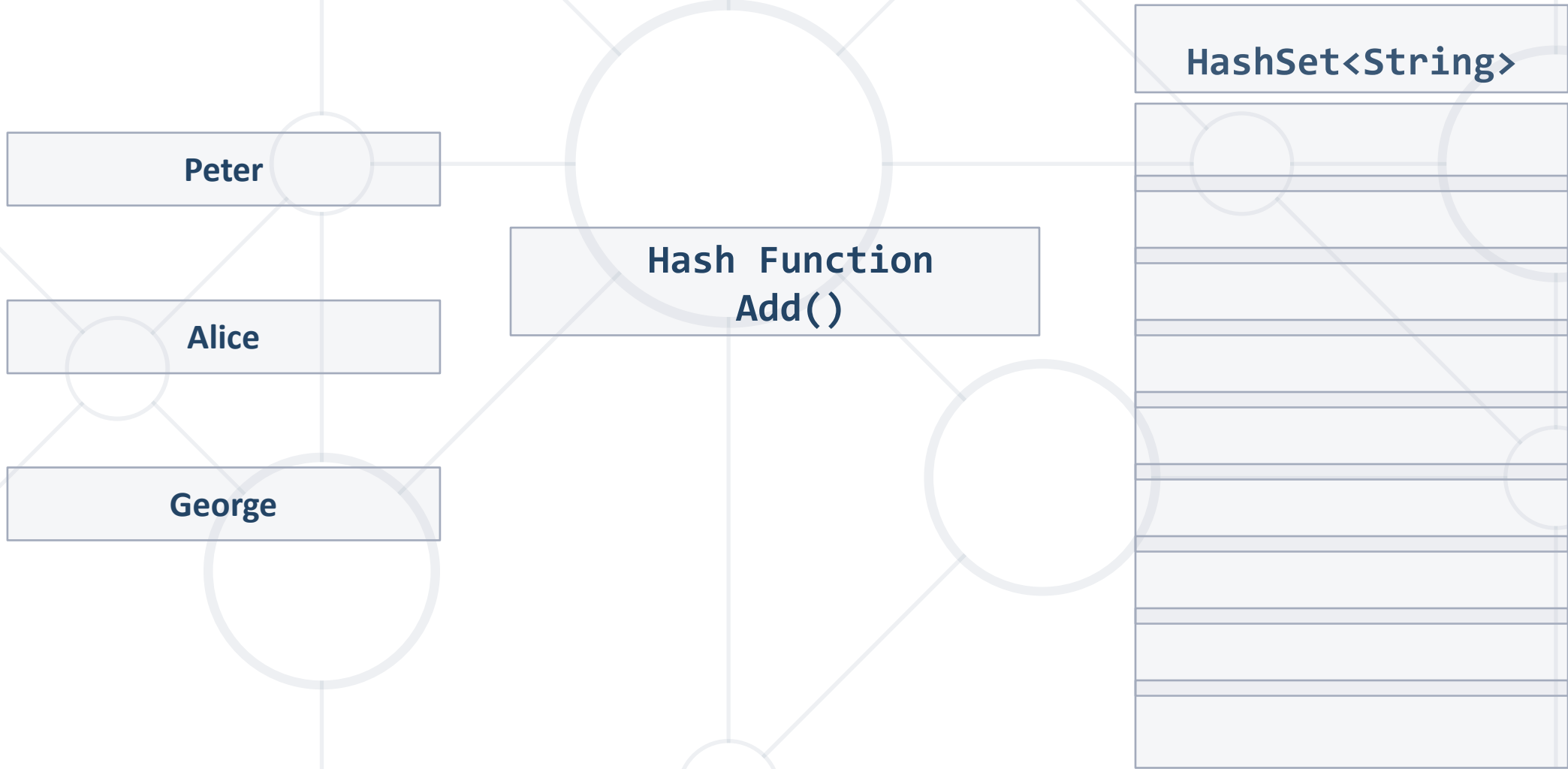
- For easy reading you can use diamond inference syntax:

```
Set<String> tree = new TreeSet<>();
```

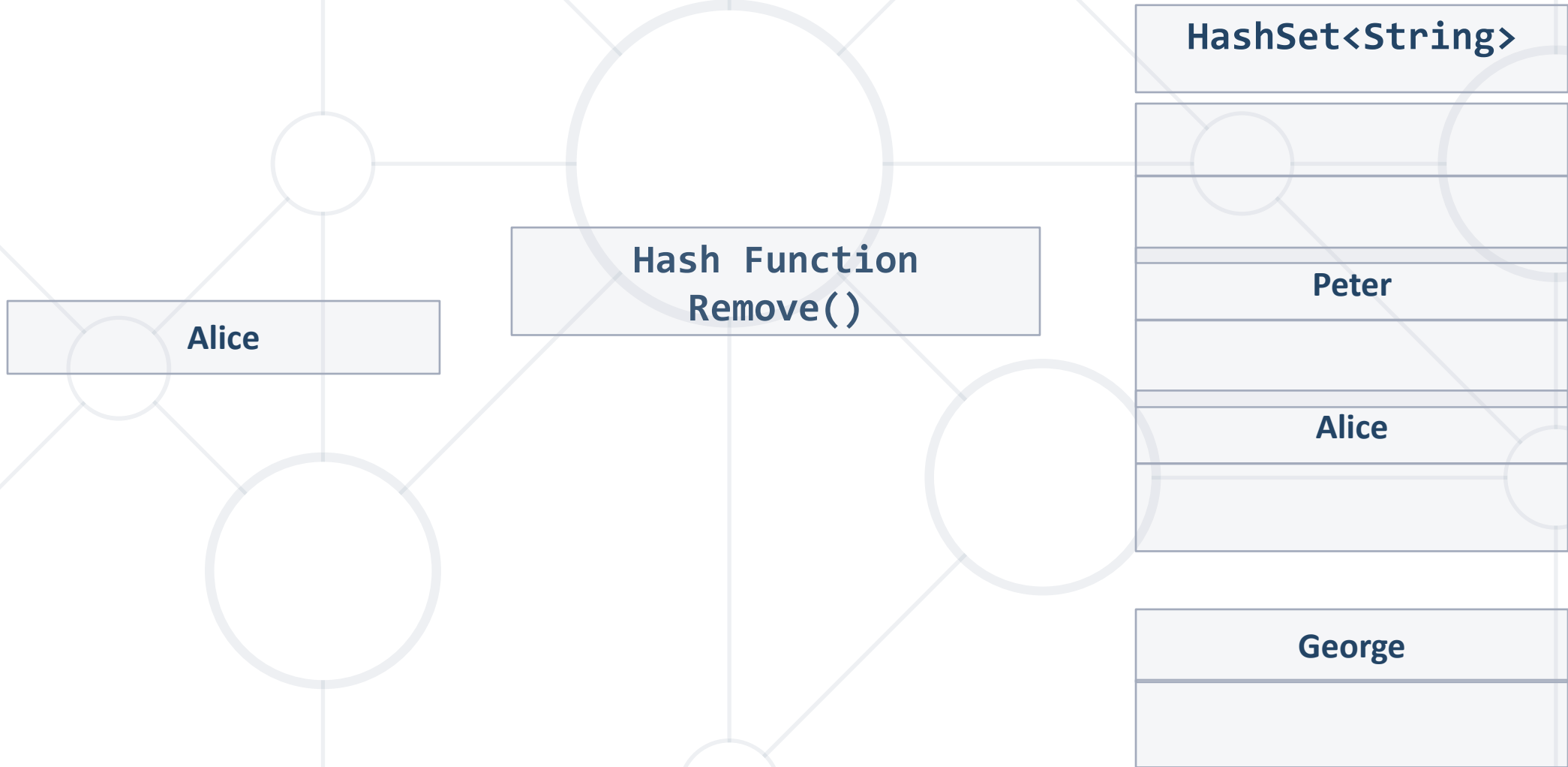
- **.size()**
- **.isEmpty()**

```
Set<String> hash = new HashSet<>();  
System.out.println(hash.size());           // 0  
System.out.println(hash.isEmpty());        // True
```

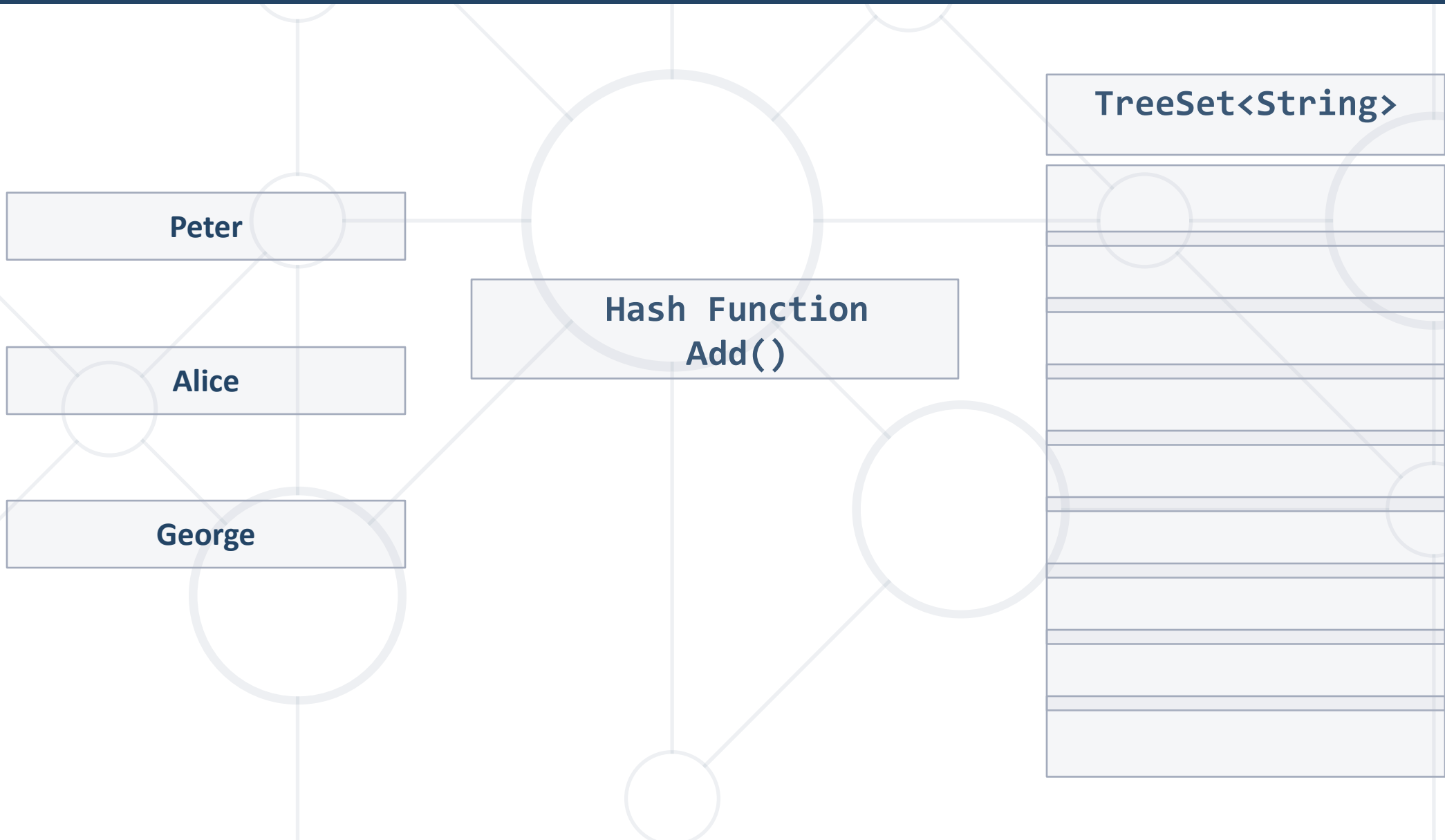
HashSet<E> – Add()



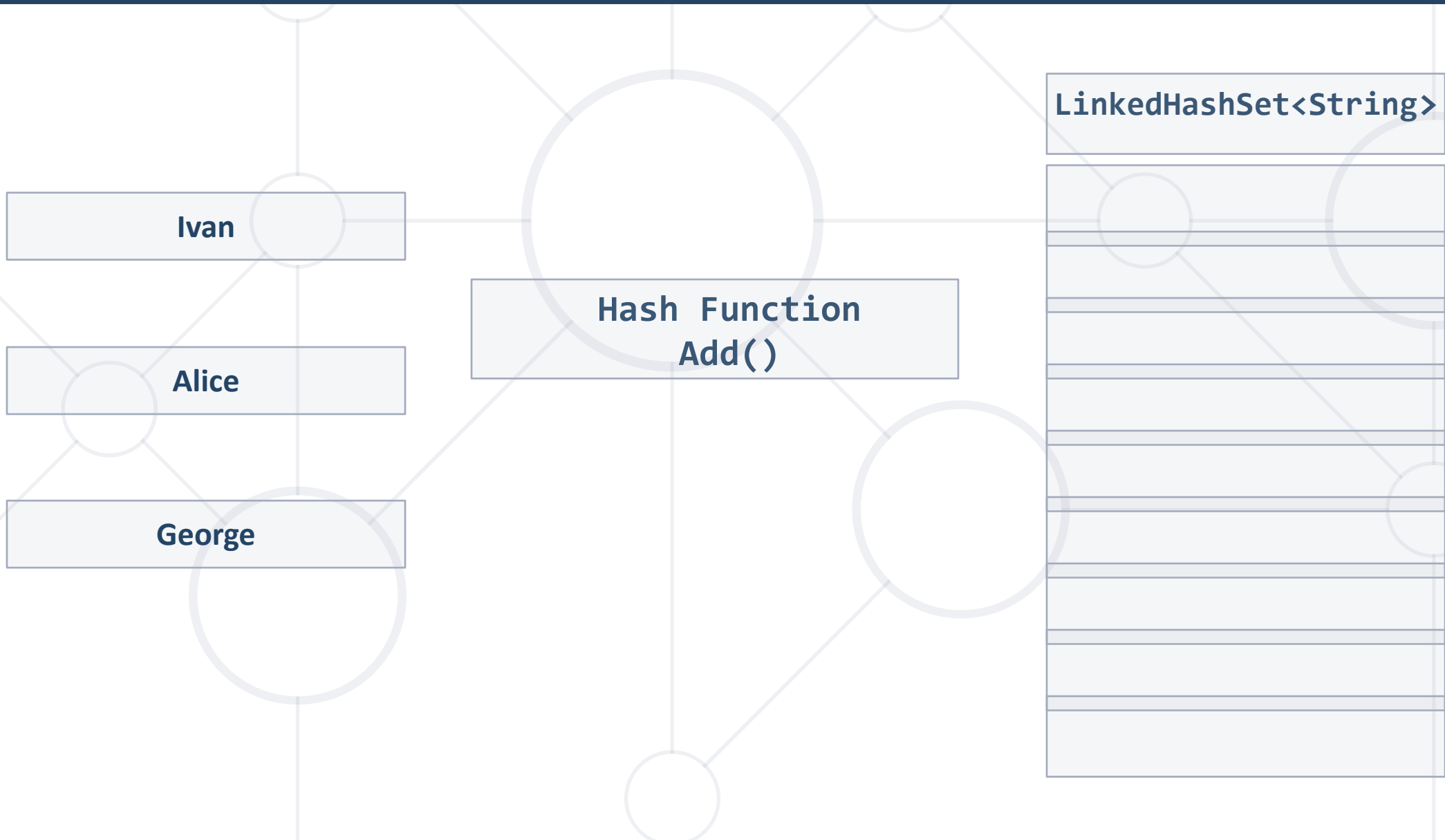
HashSet<E> – Remove()



TreeSet<E> – Add()

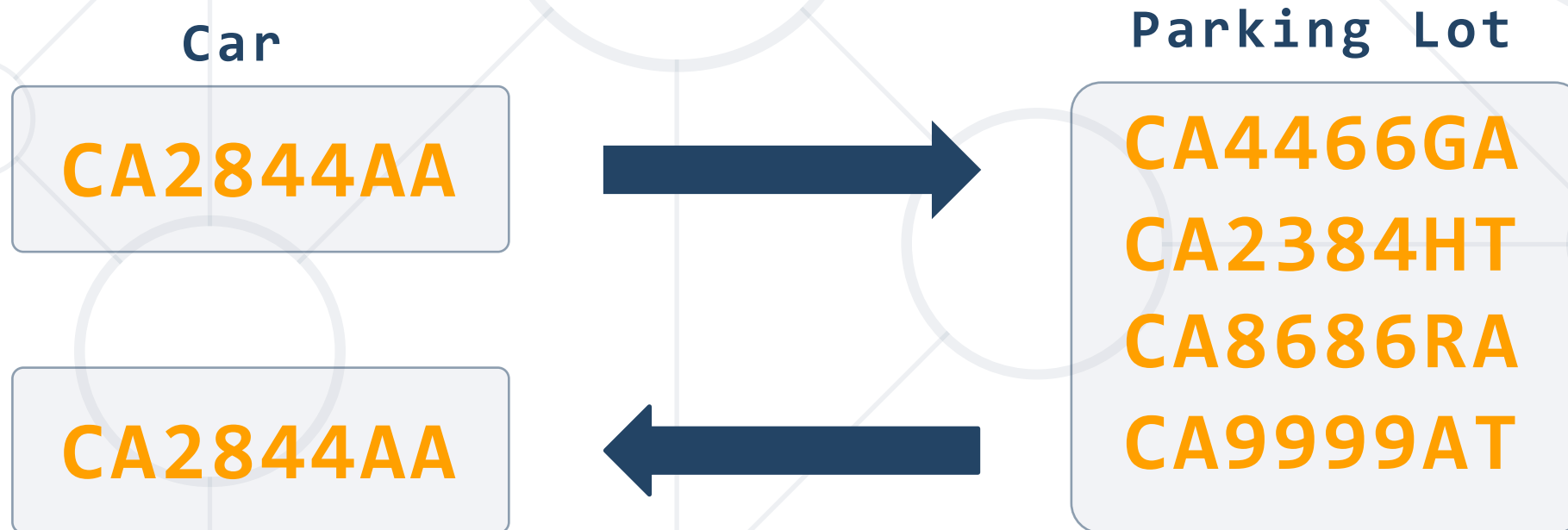


LinkedHashSet<E> – Add()



Problem: Parking Lot

- Write a program that:
 - Adds car number for every car that enters the parking lot
 - Removes car number when the car goes out



Check your solution here: <https://judge.softuni.bg/Contests/1462/>

Solution: Parking Lot

```
LinkedHashSet<String> parkingLot = new LinkedHashSet<>();  
while(true)  
    String input = sc.nextLine();  
    if (input.equals("END"))  
        break;  
    else  
        String[] reminder = input.split(", ");  
        if (reminder[0].equals("IN"))  
            parkingLot.add(reminder[1]);  
        else  
            parkingLot.remove(reminder[1]);
```



Problem: SoftUni Party

- Guests are two types:
 - **Regular**
 - **VIPs** – their tickets start with digit
- Until **PARTY** command, you will receive guest invitations
- Until **END** command, you will receive a second list with guests that actually came to the party
- Find how many guests didn't come to the party
- Print all guests that didn't come (VIPs first)

Reservation List

7IK9Yo0h
9NoBUajQ
Ce8vwPmE
SVQXQCbc

```
Set<String> vip = new TreeSet<>();
Set<String> regular = new TreeSet<>();
String guestId = scanner.nextLine();

while (!guestId.equals("PARTY")) {
    if (Character.isDigit(guestId.charAt(0)))
        vip.add(guestId);
    else
        regular.add(guestId);
    guestId = scanner.nextLine();
}

//TODO: Remove the guests who came to the party
//TODO: Print results
```

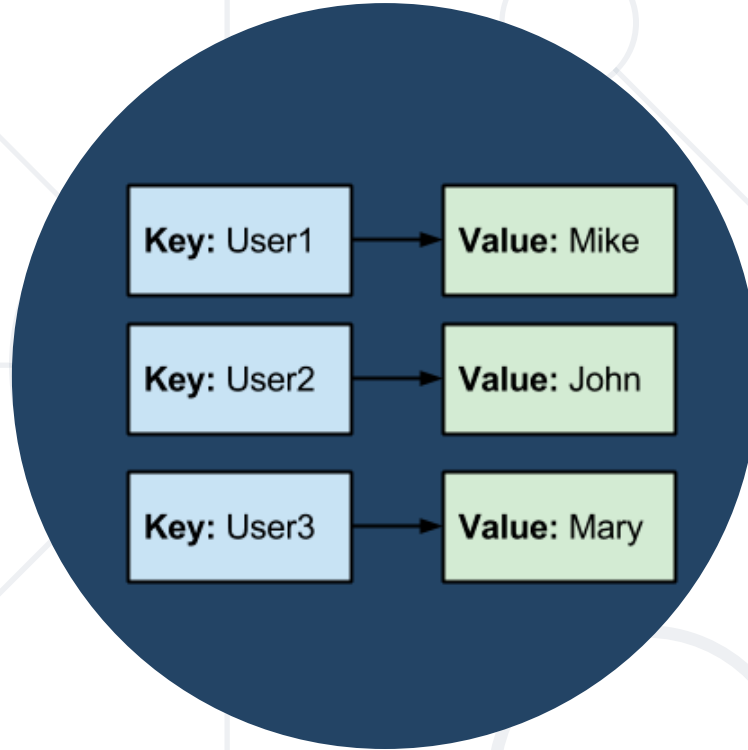
Return **true**
or **false**

Problem: "Voina" – Number Game

- Create a game that is played by two players:
 - Each one **has 20 unique numbers** (read from console, separated with space)
 - Every round each player **bets his first number** from the deck
 - Player with a bigger number wins and places both numbers **at the bottom** of his deck
 - A game ends after **50 rounds** or when a player has **0 numbers**

Solution: "Voina" – Number Game

```
LinkedHashSet<Integer> firstPlayer = getPlayerNumbers();
LinkedHashSet<Integer> secondPlayer = getPlayerNumbers();
for (int i = 0; i < 50; i++) {
    int firstNumber = firstPlayer.iterator().next();
    firstPlayer.remove(firstNumber);
    //TODO: get top number for second player
    if (firstNumber > secondNumber) {
        firstPlayer.add(firstNumber);
        firstPlayer.add(secondNumber);
    } else if (secondNumber > firstNumber)
        //TODO: finish logic about second player win or draw
    }
    //TODO: print result
}
```

Associative Arrays

Associative Arrays (Maps)

- **Associative arrays** are arrays indexed by keys
 - Not by the numbers 0, 1, 2, ...
- Hold a set of pairs **<key, value>**

■ Traditional

key	0	1	2	3	4
value	8	-3	12	408	33

■ Associative array

key	value
John Smith	+1-555-8976
Lisa Smith	+1-555-1234
Sam Doe	+1-555-5030

- Initialization

```
Map<String, Integer> hash = new HashMap<String, Integer>();
```

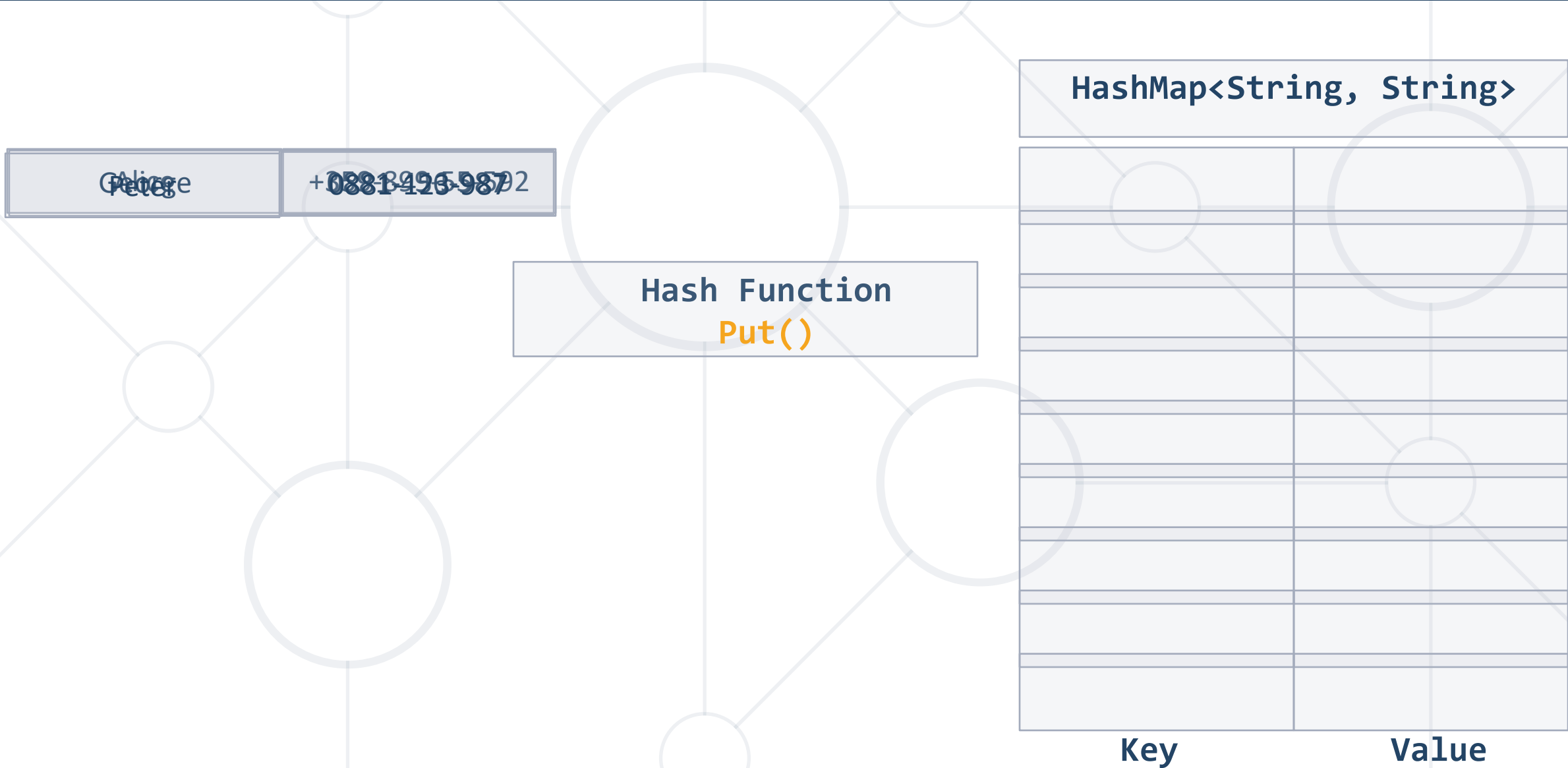
Type of keys

Type of values

- **.size()**
- **.isEmpty()**

```
Map<String, Integer> hash = new HashMap<>();  
System.out.println(hash.size());           // 0  
System.out.println(hash.isEmpty());        // True
```

HashMap<K, V> – Put()



HashMap<K, V> – Remove()

Peter

Hash Function
Remove()

HashMap<String, String>

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

Key

Value

Looping Through Maps – Example

```
Map<String, Integer> vehicles = new HashMap<>();  
vehicles.put("BMW", 5);  
vehicles.put("Mercedes", 3);  
vehicles.put("Audi", 4);  
vehicles.put("BMW", 10);  
for(String key: vehicles.keySet())  
    System.out.println(key + " - " + vehicles.get(key));
```

Override first value

Return set of all keys

Return value
for key



```
Audi - 4  
Mercedes - 3  
BMW - 10
```

Problem: Count Real Numbers

- Write a program that **counts** in a given array of **double** values the number of occurrences of each value

-2.5	4	3	-2.5	-5.5	4	3	3	-2.5	3
------	---	---	------	------	---	---	---	------	---

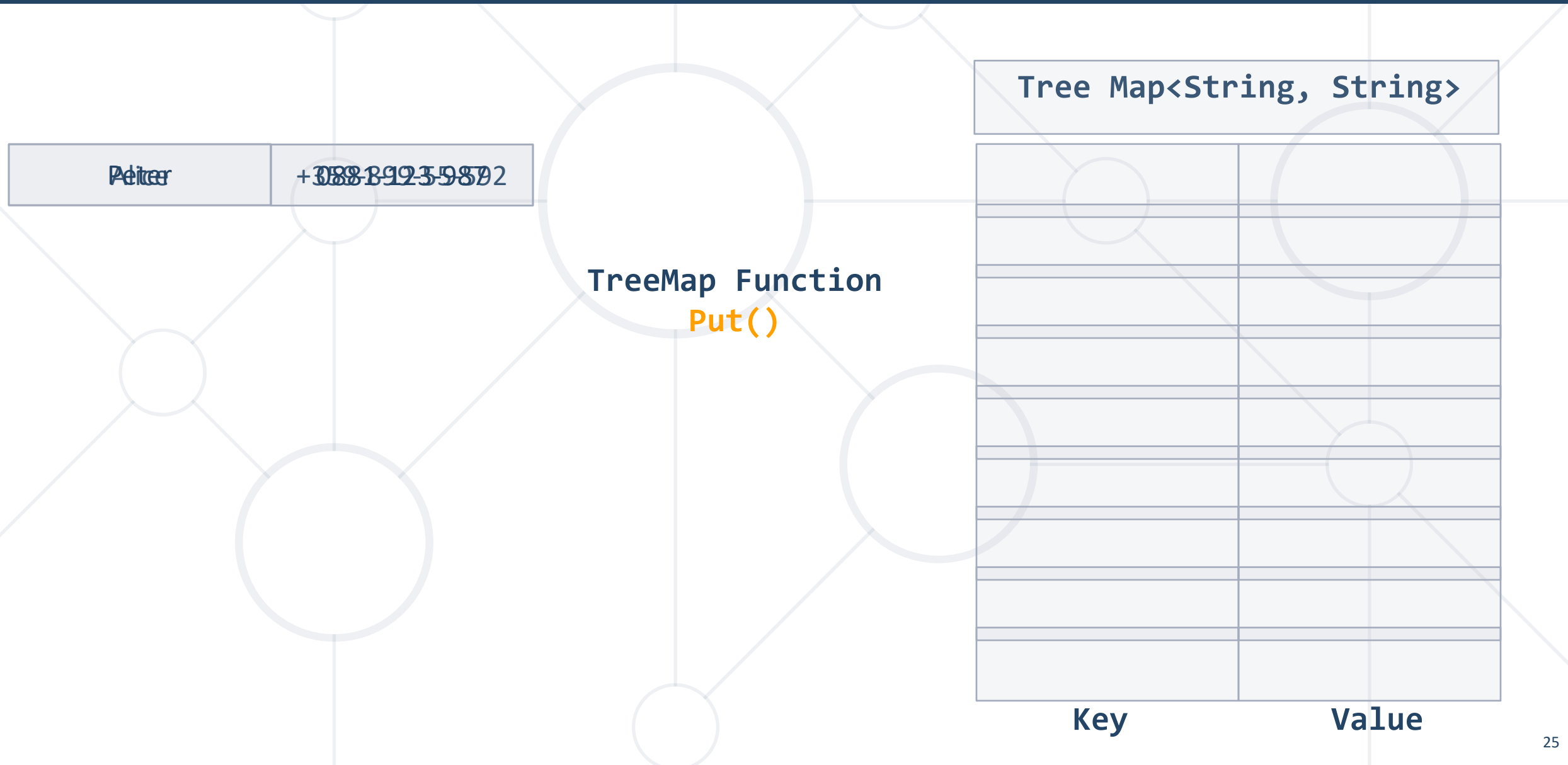


-2.5	->	3
4.0	->	2
3.0	->	4
-5.5	->	1

Solution: Count Real Numbers

```
LinkedHashMap<Double, Integer> result = new LinkedHashMap<>();
for (Double number : input) {
    if (!result.containsKey(number)) {
        result.put(number, 1);
    } else {
        result.put(number, result.get(number) + 1);
    }
}
for (Double key : result.keySet()) {
    System.out.println(key + " -> " + result.get(key));
}
```


TreeMap<K, V> – Put()



Problem: Academy Graduation

- Write a program that:
 - Reads a list of students and their score for some courses
 - Prints on the console sorted list with average score for each student

Student	Java Advanced	Java OOP
George	3.75	5
Maria	4.25	6
Peter	6	4.5



Student	Average
George	4,375
Maria	5,125
Peter	5,25

Solution: Academy Graduation

```
TreeMap <String,Double[]> graduationList = new TreeMap<>();
for (int i = 0; i < numberOfStudents; i++) {
    String name = scanner.nextLine();
    String[] scoresStrings = scanner.nextLine().split(" ");
    Double[] scores = new Double[scoresStrings.length];

    for (int j = 0; j < scoresStrings.length; j++) {
        scores[j] = Double.parseDouble(scoresStrings[j]);
    }
    graduationList.put(name, scores);
}
//TODO: print results
```

- **size()** - the number of key-value pairs
- **keySet()** - a set of unique keys
- **values()** - a collection of all values
- Basic operations - **put()**, **remove()**, **clear()**
- Boolean methods:
 - **containsKey()** - checks if a key is present in the Map
 - **containsValue()** - checks if a value is present in the Map

- 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) Second criteria  
            res = e1.getKey().compareTo(e2.getKey());  
        return res; }) Terminates the stream  
    .forEach(e -> System.out.println(e.getKey() + " " + e.getValue()));
```

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 the largest 3, if there are less than 3, print all of them

10 30 15 20 50 5



50 30 20

1 2 3



3 2 1

20 30



30 20

Check your solution here: <https://judge.softuni.org/Contests/1462/>

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.org/Contests/1462/>

- **HashSet<E>, TreeSet<E>** and **LinkedHashSet<E>** hold unique elements and are very fast
- **HashMap<K, V>, TreeMap<K, V>** and **LinkedHashMap<K, V>** are associative arrays where a value is accessed by its key



SoftUni Diamond Partners



SCHWARZ



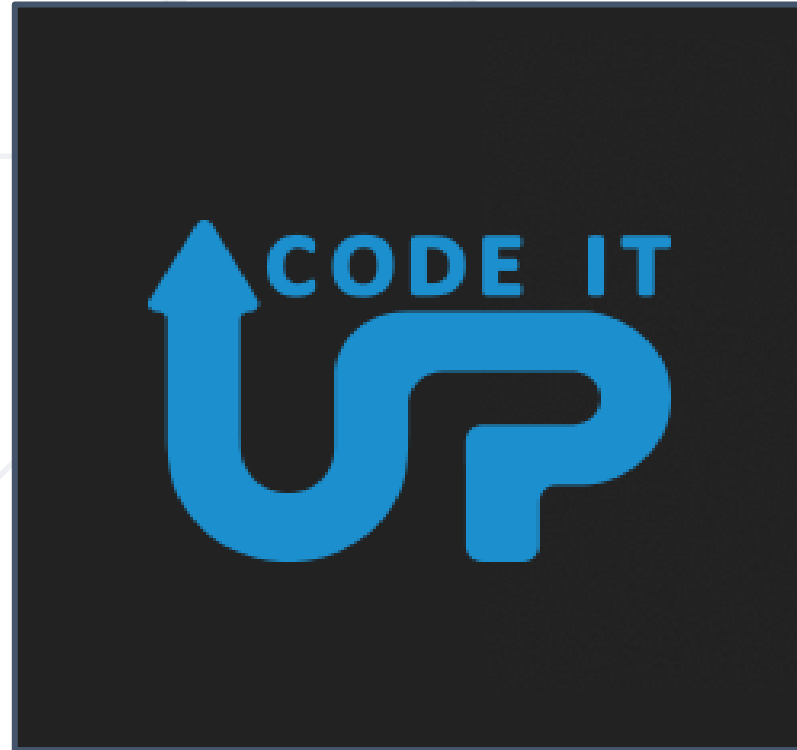
**SUPER
HOSTING
.BG**



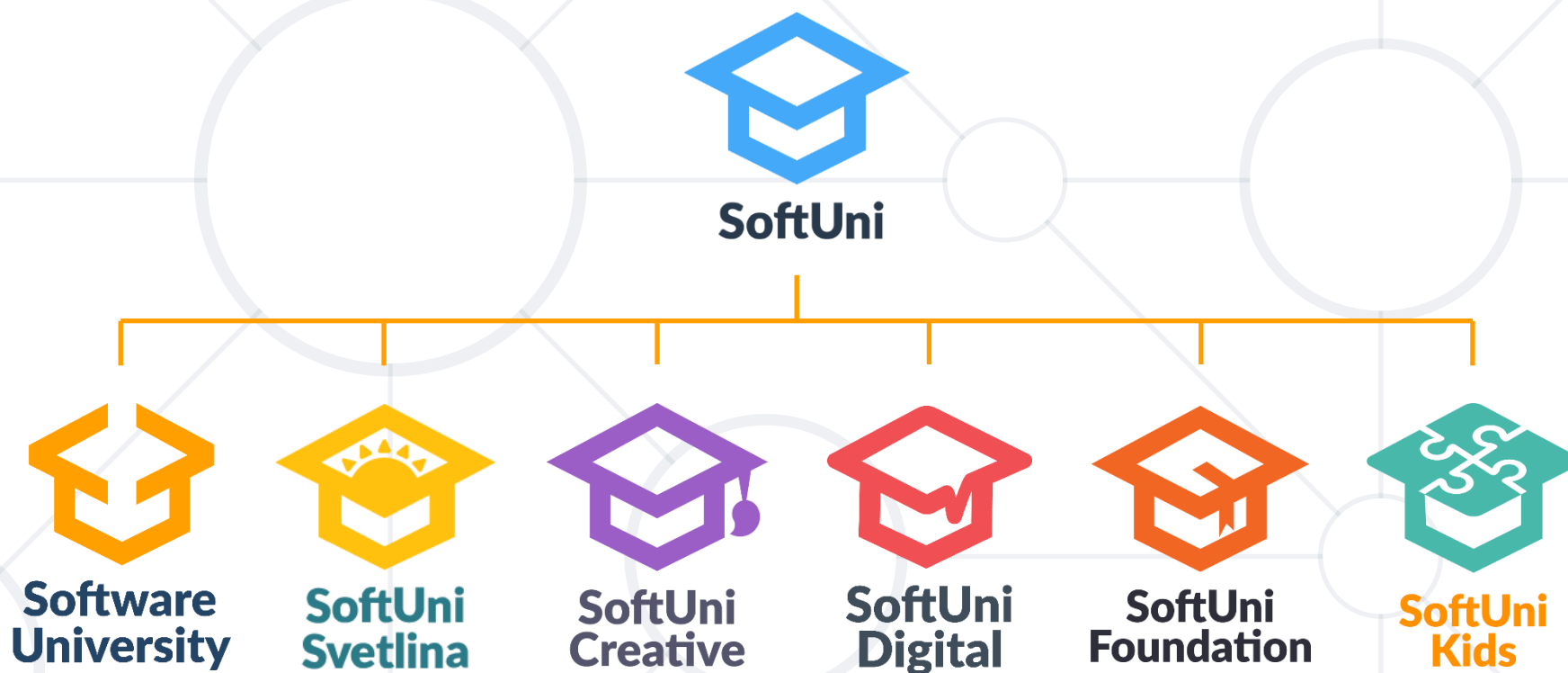
INDEAVR
Serving the high achievers

Bosch.io





Questions?



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- 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>

