Java Basics

Data types specifics, Console Interface, Collections (Maps, Lists, Sets), Generics

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Contents

Questions

SOFTWARE UNIVERSITY FOUNDATION

sli.do

#db-advanced

SOFTWARE UNIVERSITY
FOUNDATION

Data Types in Java

Integer Types

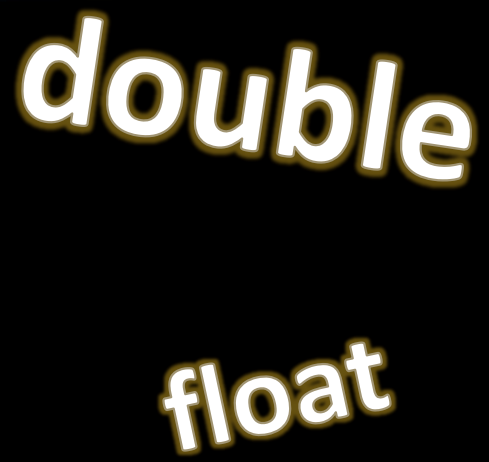
- `byte` (-128 to 127): signed 8-bit
- `short` (-32,768 to 32,767): signed 16-bit
- `int` (-2,147,483,648 to 2,147,483,647): signed 32-bit
- `long` (-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807): signed 64-bit

```
byte b = 1;
int i = 5;
long num = 3L;
short sum = (short) (b + i + num);
```

Capital l is harder to confuse with 1

Floating-Point Types

- Floating-point types are:
  - `float` ($\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$)
    - 32-bits, precision of 7 digits
  - `double` ($\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$)
    - 64-bits, precision of 15-16 digits
- The default value of floating-point types:
  - Is `0.0F` for the `float` type
  - Is `0.0D` for the `double` type



Floating-Point Types – Examples


```
float f = 0.33f;
double d = 1.67;
double sum = f + d;
float fSum = f + d; // This will not compile
double infinity = 3.14 / 0;

System.out.println(f); // 0.33
System.out.println(d); // 1.67
System.out.println(sum); // 2.0000000013113022
System.out.println(infinity); // Infinity
```

"f" specifies a float

Double by default

BigDecimal

- The floating-point arithmetic sometime works incorrectly
  - Don't use **`float`** and **`double`** for financial calculations!
- In Java use the **`BigDecimal`** class for financial calculations:

```
import java.math.BigDecimal;
…
BigDecimal bigF = new BigDecimal("0.33");
BigDecimal bigD = new BigDecimal("1.67");
BigDecimal bigSum = bigF.add(bigD);
System.out.println(bigSum); // 2.00
```

Other Primitive Data Types

- Boolean

```
boolean b = true;
System.out.println(b); // true
System.out.println(!b); // false
```

- Character

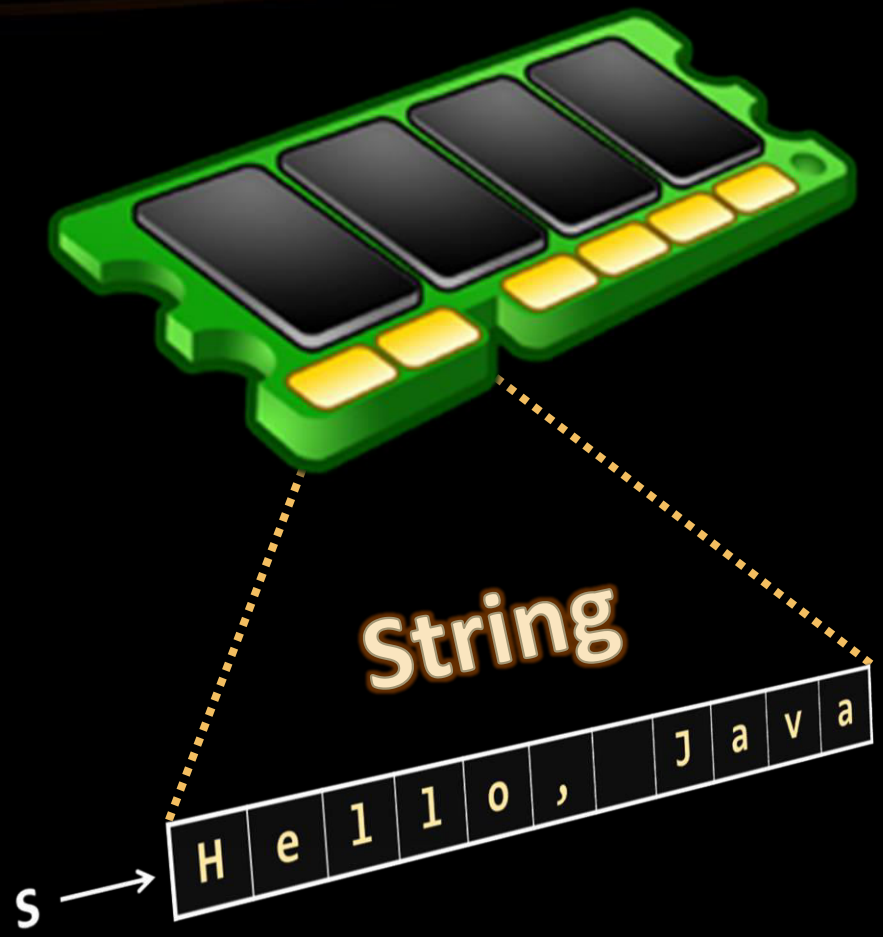
```
char ch = 'Ю';
System.out.println(ch);
char ch = '\u03A9'; // Ω
System.out.println(ch);
```

The String Data Type

- The String data type:
  - Declared by the `String` class
  - Represents a sequence of characters
  - Has a default value `null` (no value)
- Strings are enclosed in quotes:

```
String s = "Hello, Java";
```

- Strings can be concatenated
  - Using the + operator



The Object Type

- The Object type:
  - Is declared by the `java.lang.Object` class
  - Is the base type of all other types
  - Can hold values of any type

Object

```
object dataContainer = 5;
System.out.print("The value of dataContainer is: ");
System.out.println(dataContainer);

dataContainer = "Five";
System.out.print("The value of dataContainer is: ");
System.out.println(dataContainer);
```

Nullable Types: Integer, Long, Boolean, …

- Each primitive type in Java has a corresponding wrapper:
  - `int` → `java.lang.Integer`
  - `double` → `java.lang.Double`
  - `boolean` → `java.lang.Boolean`
- Primitive wrappers can have a value or be `null` (no value)

```
Integer i = 5; // Integer value: 5
i = i + 1; // Integer value: 6
i = null; // No value (null)
i = i + 1; // NullPointerException
```

Type Conversion

- Type conversion and typecasting change one type to another
  - Java supports Explicit type conversion (casting)

```
float heightInMeters = 1.74f; // Explicit conversion
double minHeight = (double) heightInMeters; // Explicit
byte dataLoss = (byte) 12345; // Explicit with data loss
```

  - Java also supports Implicit type conversion

```
double maxHeight = heightInMeters; // Implicit
//double minHeight = heightInMeters; // Compilation error!
```

Implicit casting is not allowed if data loss is possible

SOFTWARE UNIVERSITY FOUNDATION

Console Input and Output

Scanner and Formatted Printing

Reading From the Console

- The `java.util.Scanner` class can read strings

```
import java.util.Scanner;
…

Scanner scanner = new Scanner(System.in);
String name = scanner.nextLine();
```

- The `scanner` takes as a parameter an input stream
- `nextLine()` reads a whole line
- `next()` reads the string before the next delimiter

Reading From the Console (2)

SOFTWARE UNIVERSITY FOUNDATION

- The `java.util.Scanner` class also reads numbers

```
import java.util.Scanner;
…

Scanner scanner = new Scanner(System.in);
int firstNum = scanner.nextInt();
double secondNum = scanner.nextDouble();
```

- The numbers can be separated by any sequence of whitespace characters (e.g. spaces, tabs, new lines, …)
- Exception is thrown when non-number characters are entered

Printing to the Console

- Using `System.out.print()` and `System.out.println()`:

```
String name = "SoftUni";
String location = "Sofia";
double age = 3.5;

System.out.print(name);
System.out.println(" is " + age +
    " years old organization located in " + location + ".");

// Output:
// SoftUni is 0.5 years old organization located in Sofia.
```

Prints without a new line

Prints a new line

Formatted Printing

- Java supports formatted printing by `System.out.printf()`
  - `%s` – prints a string argument

```
System.out.printf("Name: %s", name);
```

  - `%f` – prints a floating-point argument

```
System.out.printf("Height: %f", height);
```

  - `%.2f` – prints a floating-point argument with 2 digits precision

```
System.out.printf("Height: %.2f", height);
```

  - `%n` – prints a new line
- Learn more at http://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html

Lists in Java

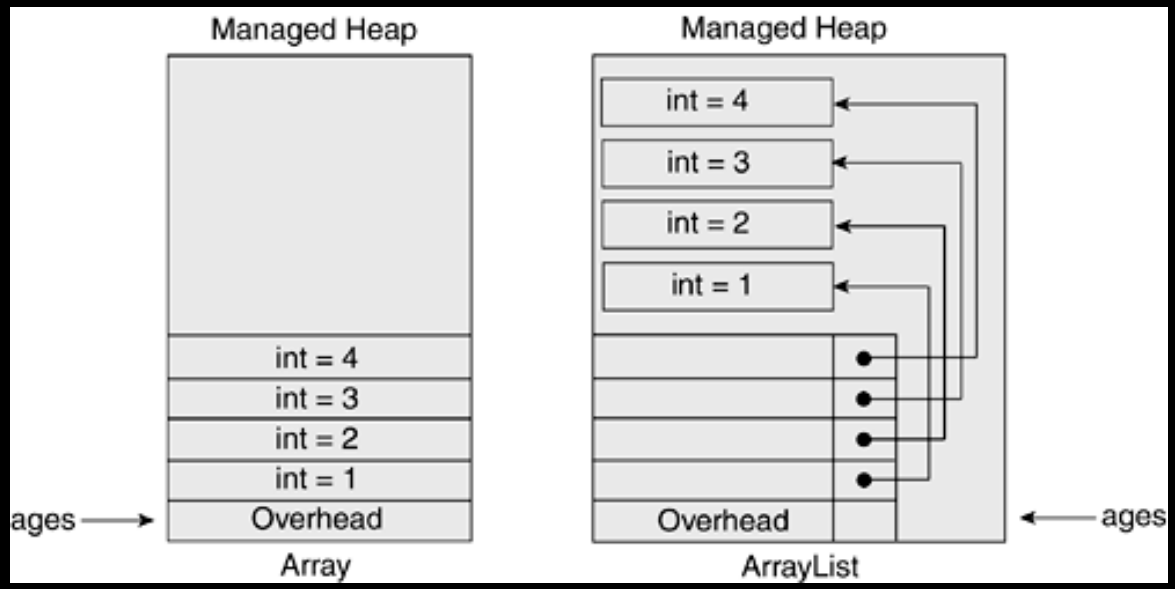
- In Java arrays have fixed length
  - Cannot add / remove / insert elements
- Lists are like resizable arrays
  - Allow add / remove / insert of elements
- Lists in Java are defined through the **`ArrayList<E>`** class
  - Where **`E`** is the type of the list, e.g. **`String`** or **`Integer`**

Prefer Interface rather than Class

```
List<Integer> numbers = new ArrayList<Integer>();
numbers.add(5);
System.out.println(numbers.get(0)); // 5
```

How array lists are stored in the memory

- Reference data type
  - Variable “ages” holds pointers to Objects in the heap as values.
  - Each Object has an address that points to a value in the memory.



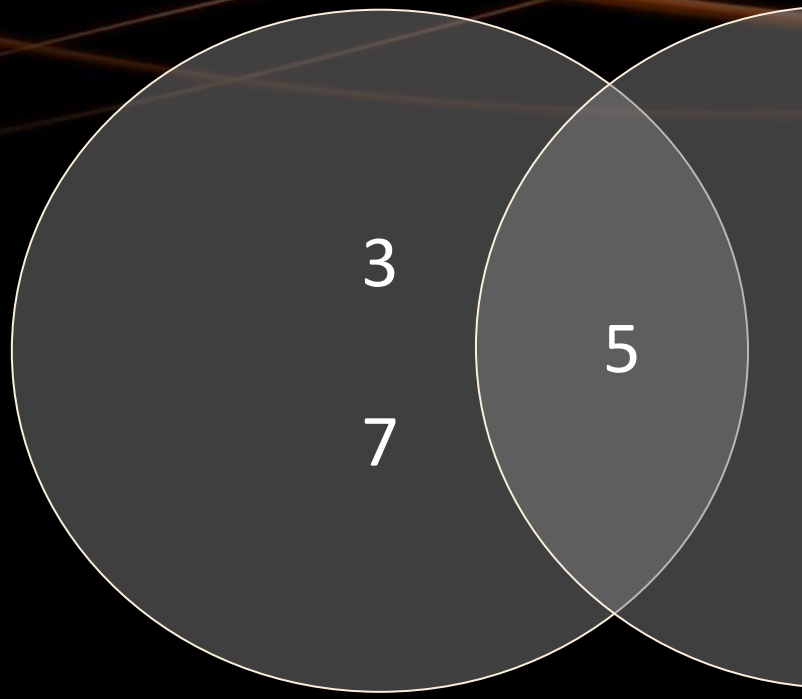
ArrayList<String> – Example


```
List<String> names = new ArrayList<String>() {{
        add("Peter");
        add("Maria");
        add("Katya");
        add("Todor");
}};
names.add("Nakov"); // Peter, Maria, Katya, Todor, Nakov
names.remove(0); // Maria, Katya, Todor, Nakov
names.remove(1); // Maria, Todor, Nakov
names.remove("Todor"); // Maria, Nakov
```

ArrayList<Integer> – Example


```
// This will not compile!
List<int> intArr = new ArrayList<int>();

// This will compile!
List<Integer> nums = new ArrayList<>(
    Arrays.asList(5, -3, 10, 25));
nums.add(55); // 5, -3, 10, 25, 55
System.out.println(nums.get(0)); // 5
System.out.println(nums); // [5, -3, 10, 25, 55]
nums.remove(2); // 5, -3, 25, 55
nums.set(0, 101); // 101, -3, 25, 55
System.out.println(nums); // [101, -3, 25, 55]
```



Sets

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

Sets in Java

- A set keeps unique elements
  - Provides methods for adding/removing/searching elements
  - Offers very fast performance
- `HashSet<E>`
  - The elements are randomly ordered
- `TreeSet<E>`
  - The elements are ordered incrementally
- `LinkedHashSet<E>`
  - The order of appearance is preserved

Sets Methods

- 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: SoftUni party

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

Reservation List

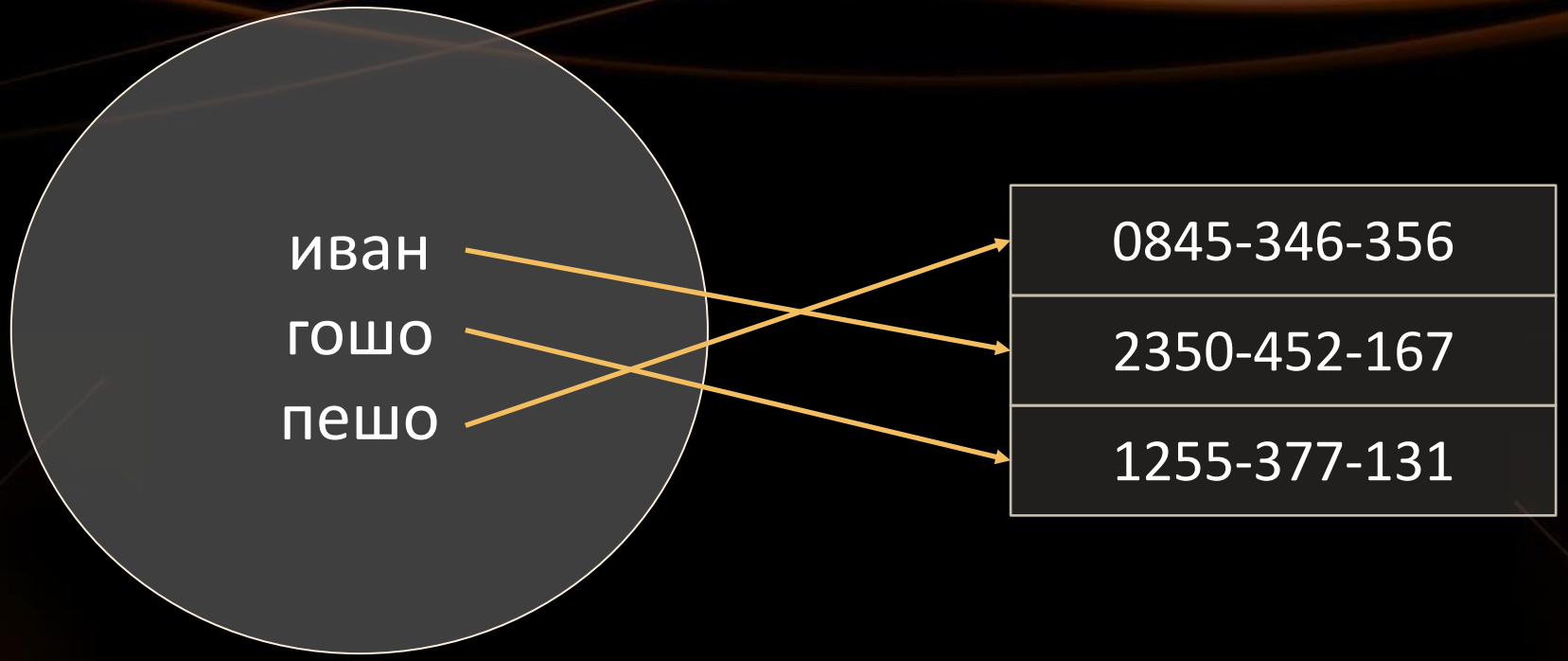
```
7IK9Yo0h
9NoBUajQ
Ce8vwPmE
SVQXQCbc
```

Solution: SoftUni party

```
Set<String> vip = new HashSet<String>();
Set<String> regular = new TreeSet<String>();
while (true)
  String input = sc.nextLine();
  if (input.equals("PARTY"))
    break;
  else
    String sign = Character.toString(input.charAt(0));
    if (numbers.contains(sign))
      vip.add(input);
    else
      regular.add(input);
//TODO: Remove from guest, that came to party
regular.addAll(vip);
//TODO: Print results
```

Return `true` or `false`

SOFTWARE UNIVERSITY FOUNDATION



Associative Arrays

`HashMap<Key, Value>`

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 array

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

Maps Methods

- Initialization

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

Type of keys

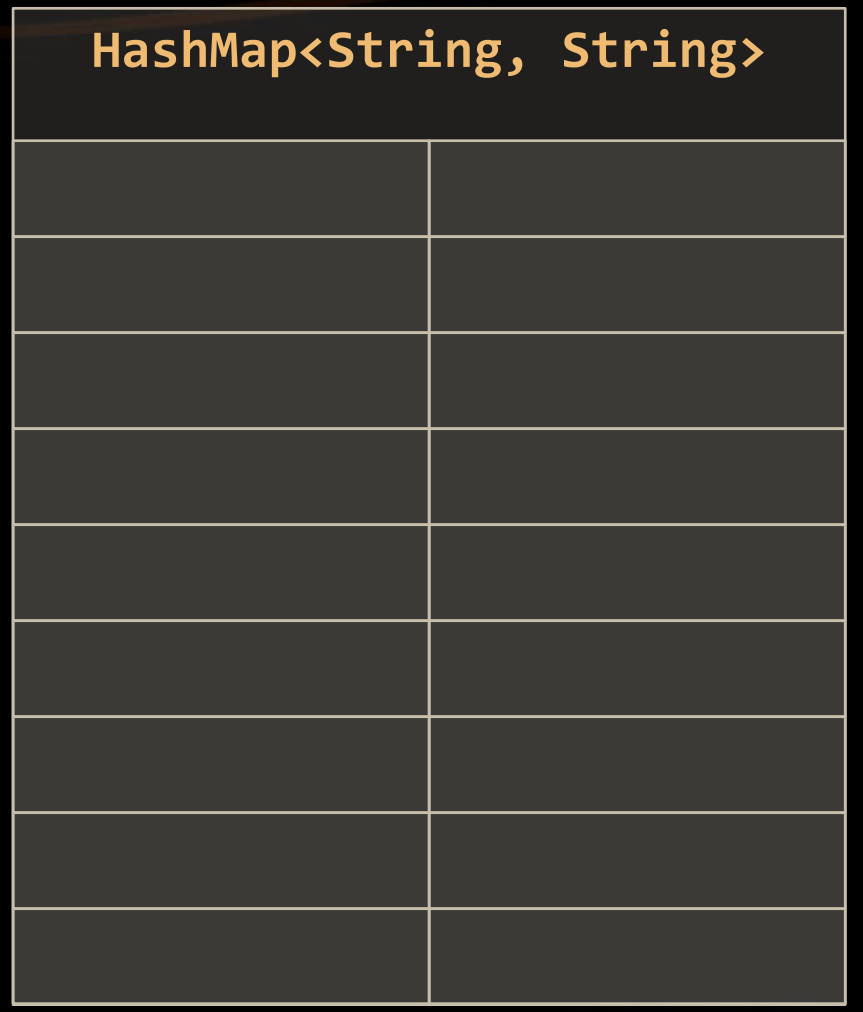
Type of values

- .size()
- .isEmpty()

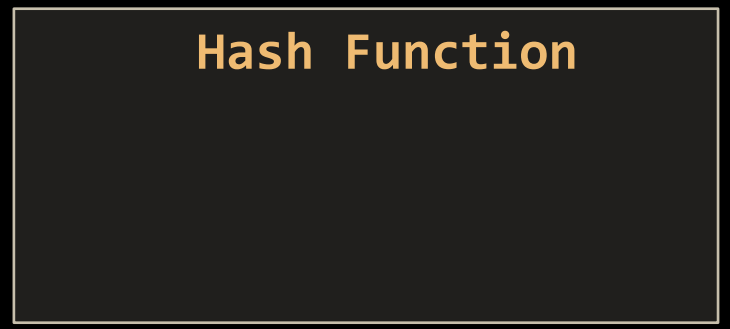
```
Set<String> hash = new HashSet<>();
System.out.println(hash.size()); // 0
System.out.println(hash.isEmpty()); // True
```

HashMap<K, V> – put()

Hash Function



HashMap<K, V> – remove()



HashMap<String, String>	
Gosho	0881-456-987
Pesho	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
```

TreeMap<K, V> – put()

Peter	+359 888 123 456

Tree Map<String, String>	
Key	Value

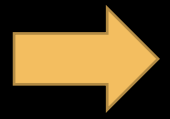
HashMap<K, V>, TreeMap<K, V>, LinkedHashMap<K, V>

- `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 dictionary
  - `containsValue()` – checks if a value is present in the dictionary

Problem: Academy Graduation

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

Student	Java Advanced	Java OOP
Gosho	3.75	5
Mara	4.25	6
Pesho	6	4.5



Student	Average
Gosho	4,375
Mara	5,125
Pesho	7,25

Solution: Count Same Values in Array

```
Map <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
```

Summary

1. Java Intro - minimal Java program
2. Data Types
3. Console Interface
4. Collection
5. Generics

Java Basics

Questions?

SOFTWARE UNIVERSITY FOUNDATION
xsoftware
SmartIT
NETPEAK
SEO and PPC for Business
SUPERHOSTING.BG
INDEAVR
Serving the high achievers
telenor
SOFTWARE GROUP
INFRAGISTICS
DESIGN / DEVELOP / EXPERIENCE

https://softuni.bg/courses/databases-advanced-hibernate

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license

- Attribution: this work may contain portions from
  - "Databases" course by Telerik Academy under CC-BY-NC-SA license

Free Trainings @ Software University

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