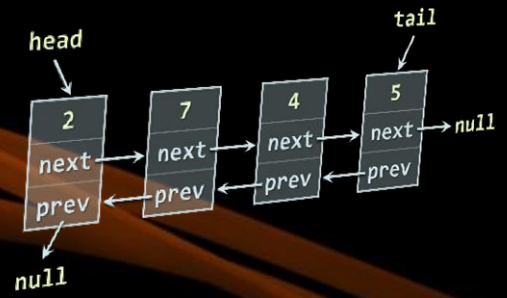
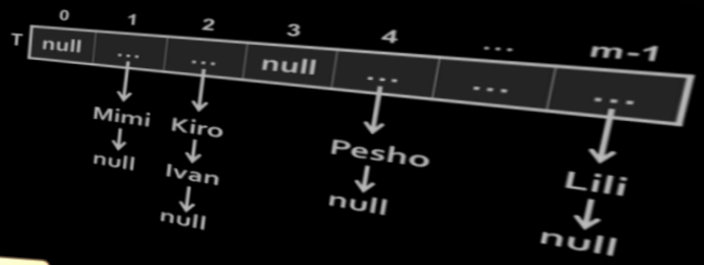
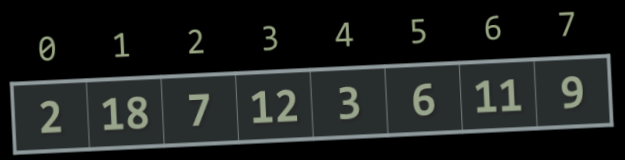
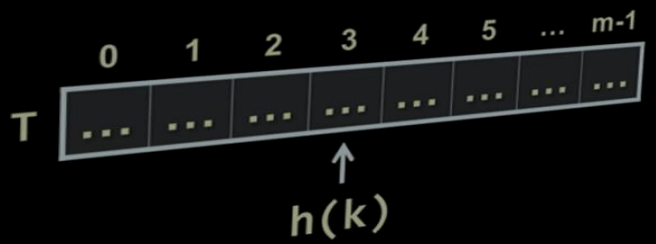


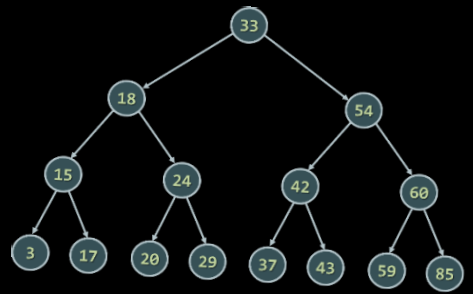
Java Collections Basics

Arrays, Lists, Strings, Sets, Maps





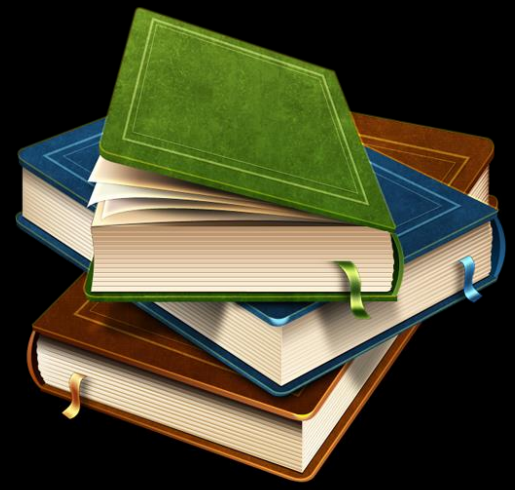




Bogomil Dimitrov
Technical Trainer

Software University
http://softuni.bg

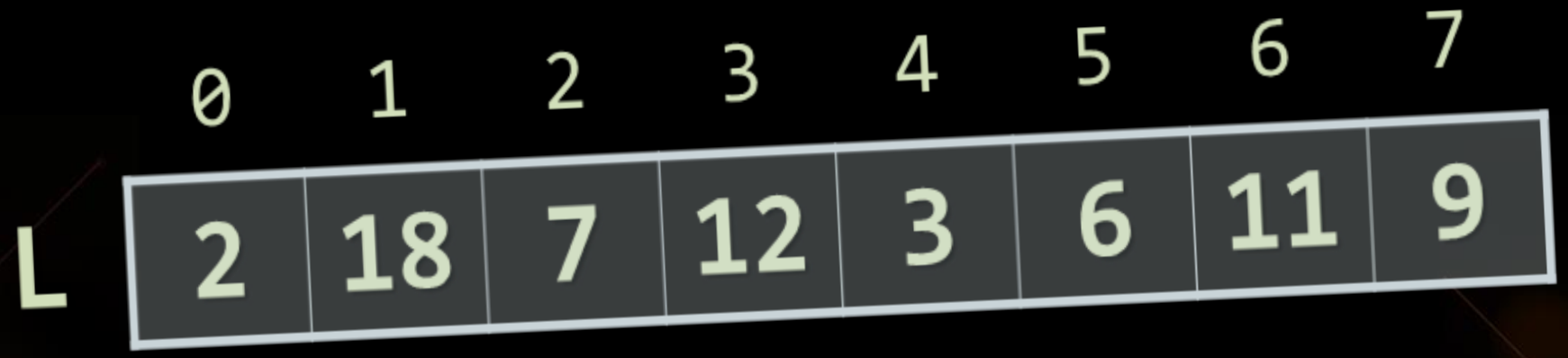
Table of Contents



Warning: Not for Absolute Beginners

- The "**Java Basics**" course is NOT for absolute beginners
  - Take the "C# Basics" course at SoftUni first: https://softuni.bg/courses/csharp-basics
  - The course is for beginners, but with previous coding skills
- Requirements
  - Coding skills – entry level
  - Computer English – entry level
  - Logical thinking

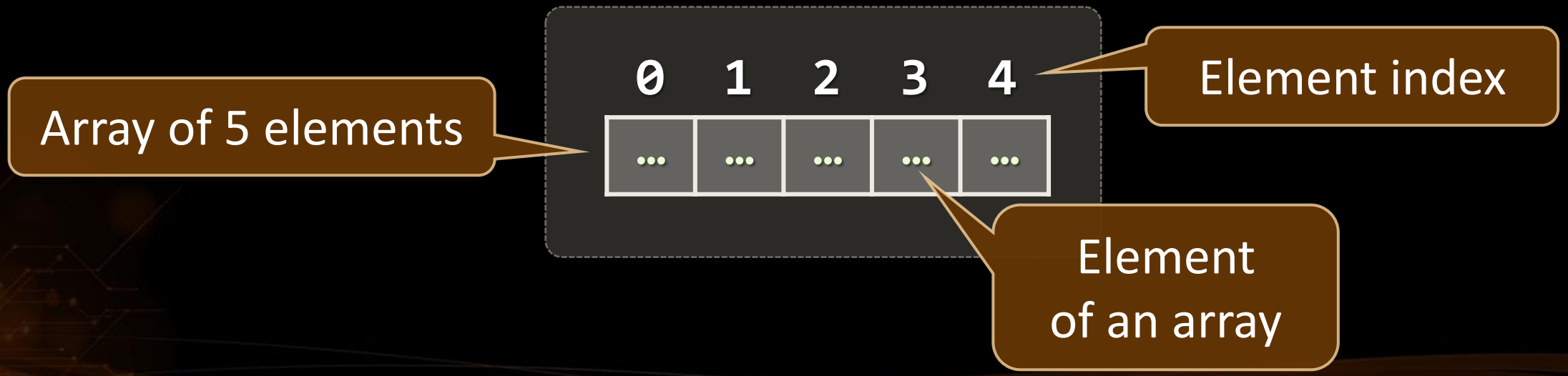
SOFTWARE UNIVERSITY FOUNDATION



Arrays

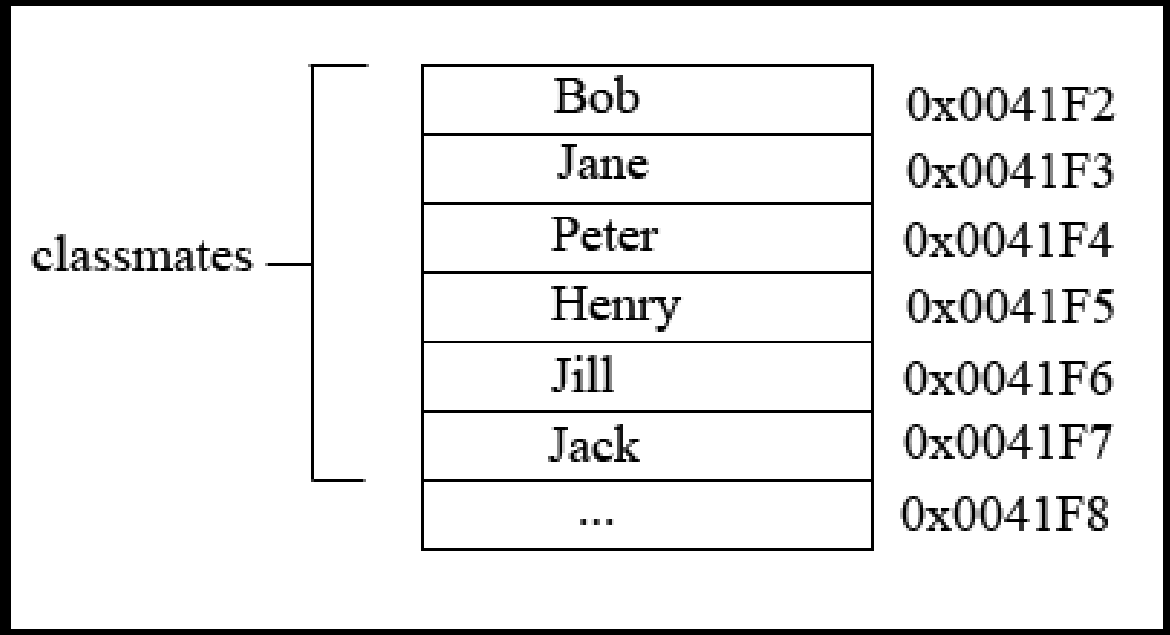
What are Arrays?

- In programming array is a sequence of elements
  - All elements are of the same type
  - The order of the elements is fixed
  - Has fixed size (`length`)



How arrays are stored in the memory

- Reference data type
  - Variable “classmates” holds addresses in the heap as values.
  - Each address points to a separate value in the heap memory.



Working with Arrays in Java

- Allocating an array of 10 integers:

```
int[] numbers = new int[10];
```

- Assigning values to the array elements:

```
for (int i=0; i<numbers.length; i++)
    numbers[i] = i+1;
```

- Accessing array elements by index:

```
numbers[3] = 20;
numbers[5] = numbers[2] + numbers[7];
```

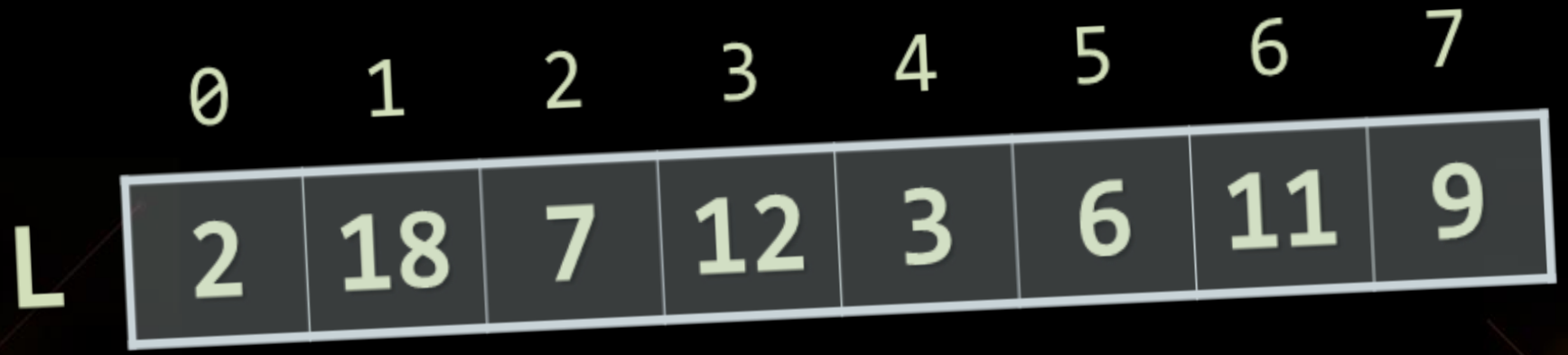
Arrays of Strings

- You may define an array of any type, e.g. **String**:

```
String[] names = { "Peter", "Maria", "Katya", "Todor" };
for (int i = 0; i<names.length; i++) {
  System.out.printf("names[%d] = %s\n", i, names[i]);
}
for (String name : names) {
  System.out.println(name);
}

names[4] = "Nakov"; // ArrayIndexOutOfBoundsException
names.length = 5; // array.length is read-only field
```

Read, Sort and Print Array of n Strings


```
Scanner scanner = new Scanner(System.in);
int n = scanner.nextInt();
scanner.nextLine();
String[] lines = new String[n];
for (int i = 0; i < n; i++) {
  lines[i] = scanner.nextLine();
}

Arrays.sort(lines);

for (int i = 0; i < lines.length; i++) {
  System.out.println(lines[i]);
}
```



Arrays

Live Demo

Lists

Using `ArrayList<E>`

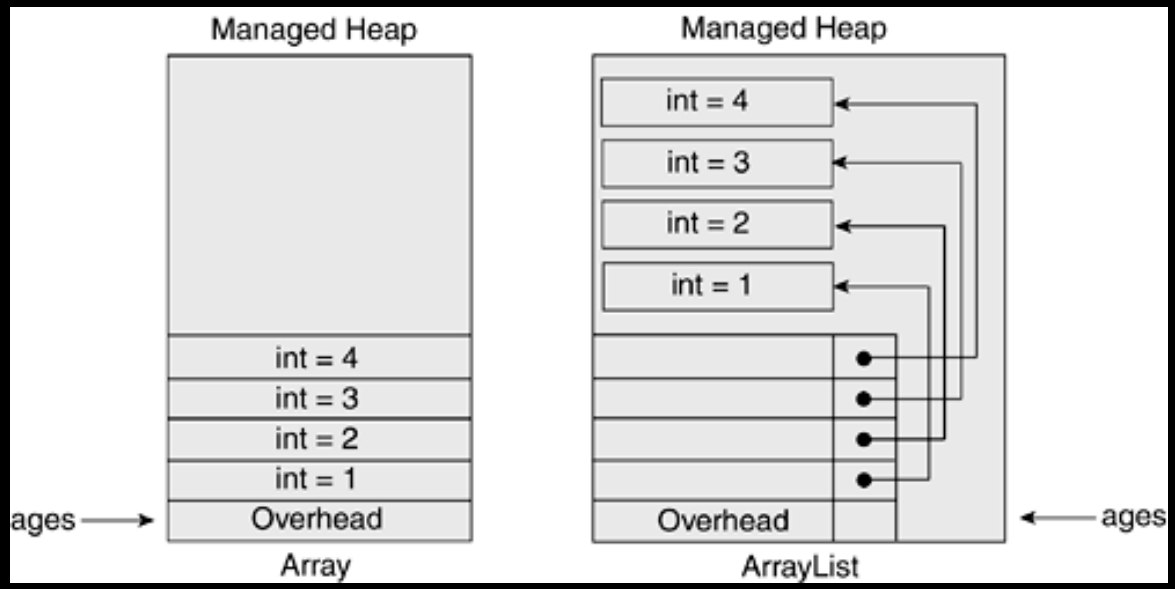
Lists in Java

- 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**

```
ArrayList<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.

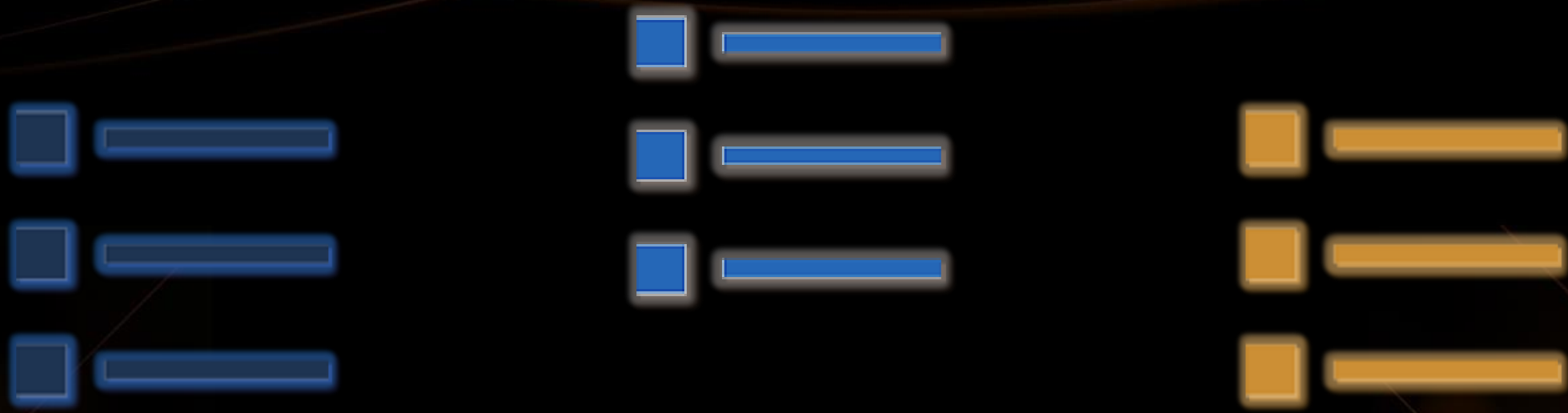


ArrayList<String> – Example


```
ArrayList<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
names.addAll(Arrays.asList("Alice", "Tedy"));
   // Maria, Nakov, Alice, Tedy
names.add(3, "Sylvia"); // Maria, Nakov, Alice, Sylvia, Tedy
names.set(2, "Mike"); // Maria, Nakov, Mike, Sylvia, Tedy
System.out.println(names);
```

ArrayList<Integer> – Example


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

ArrayList<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]
```



ArrayList<E>

Live Demo

SOFTWARE UNIVERSITY FOUNDATION

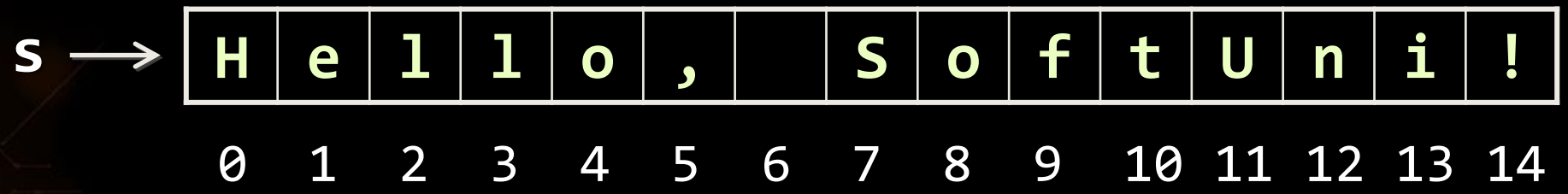
Strings

Basic String Operations

What Is String?

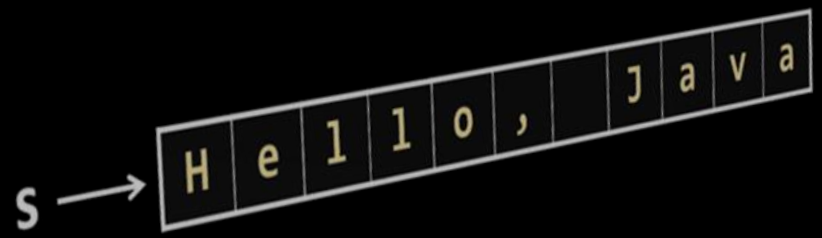
- Strings are indexed sequences of Unicode characters
  - Represented by the `String` class in Java
  - Characters accessed by index: `0` … `length()-1`
- Example:

```
string s = "Hello, SoftUni!";
```



Working with Strings

- Strings in Java
  - Know their number of characters: `length()`
  - Can be accessed by index: `charAt(0` … `length()-1)`
  - Reference types
    - Stored in the heap (dynamic memory)
    - Can have `null` value (missing value)



- Strings cannot be modified (immutable)
  - Most string operations return a new `String` instance
  - `StringBuilder` class is used to build stings

Strings – Examples


```
String str = "SoftUni";

System.out.println(str);
for (int i = 0; i < str.length(); i++) {
    System.out.printf("str[%d] = %s\n", i, str.charAt(i));
}

System.out.println(str.indexOf("Uni")); // 4
System.out.println(str.indexOf("uni")); // -1 (not found)
System.out.println(str.substring(4, 7)); // Uni
System.out.println(str.replace("Soft", "Hard")); // HardUni
System.out.println(str.toLowerCase()); // softuni
System.out.println(str.toUpperCase()); // SOFTUNI
```

Strings – Examples (2)

```
String firstName = "Steve";
String lastName = "Jobs";
int age = 56;
System.out.println(firstName + " " + lastName +
    " (age: " + age + ")"); // Steve Jobs (age: 56)
String allLangs = "C#, Java; HTML, CSS; PHP, SQL";
String[] langs = allLangs.split("[, ;]+");
for (String lang : langs) {
    System.out.println(lang);
}
System.out.println("Langs = " + String.join(", ", langs));
System.out.println("  \n\n Software   University  ".trim());
```

Comparing Strings in Java

- The == operator does not work correctly for strings!
  - Use `String.equals(String)` and `String.compareTo(String)`

```
String[] words = "yes yes".split(" ");
System.out.println("words[0] = " + words[0]); // yes
System.out.println("words[1] = " + words[0]); // yes
System.out.println(words[0] == words[1]); // false
System.out.println(words[0].equals(words[1])); // true
System.out.println("Alice".compareTo("Mike")); // < 0
System.out.println("Alice".compareTo("Alice")); // == 0
System.out.println("Mike".compareTo("Alice")); // > 0
```

Regular Expressions

- Regular expressions match text by pattern, e.g.
  - `[0-9]+` matches a non-empty sequence of digits
  - `[a-zA-Z]*` matches a sequence of letters (including empty)
  - `[A-Z][a-z]+ [A-Z][a-z]+` matches a name (first name + space + last name)
  - `\s+` matches any whitespace; `\S+` matches non-whitespace
  - `\d+` matches digits; `\D+` matches non-digits
  - `\w+` matches letters (Unicode); `\W+` matches non-letters
  - `\+\d{1,3}([ -]*[0-9]+)+` matches international phone

Validation by Regular Expression – Example

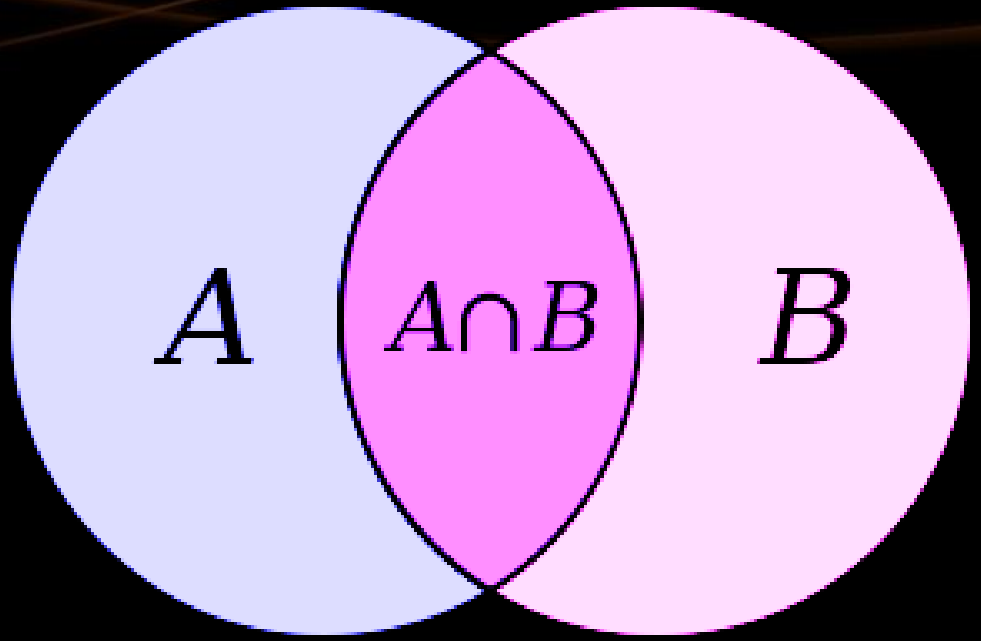

```
import java.util.regex.*;
…

String regex = "\\+\\d{1,3}([ -]*[0-9]+)+";
System.out.println("+359 2 981-981".matches(regex)); // true
System.out.println("invalid number".matches(regex)); // false
System.out.println("+359 123-".matches(regex)); // false
System.out.println("+359 (2) 981 981".matches(regex)); // false
System.out.println("+44 280 11 11".matches(regex)); // true
System.out.println("++44 280 11 11".matches(regex)); // false
System.out.println("(+49) 325 908 44".matches(regex)); // false
System.out.println("+49 325 908-40-40".matches(regex)); // true
```

Find Matches by Pattern – Example


```
import java.util.regex.*;
…
String text =
    "Hello, my number in Sofia is +359 894 11 22 33, " +
    "but in Munich my number is +49 89 975-99222.";
Pattern phonePattern = Pattern.compile(
    "\\+\\d{1,3}([ -]*([0-9]+))+");
Matcher matcher = phonePattern.matcher(text);
while (matcher.find()) {
    System.out.println(matcher.group());
}
// +359 894 11 22 33
// +49 89 975-99222
```

Strings

Live Demos



Sets

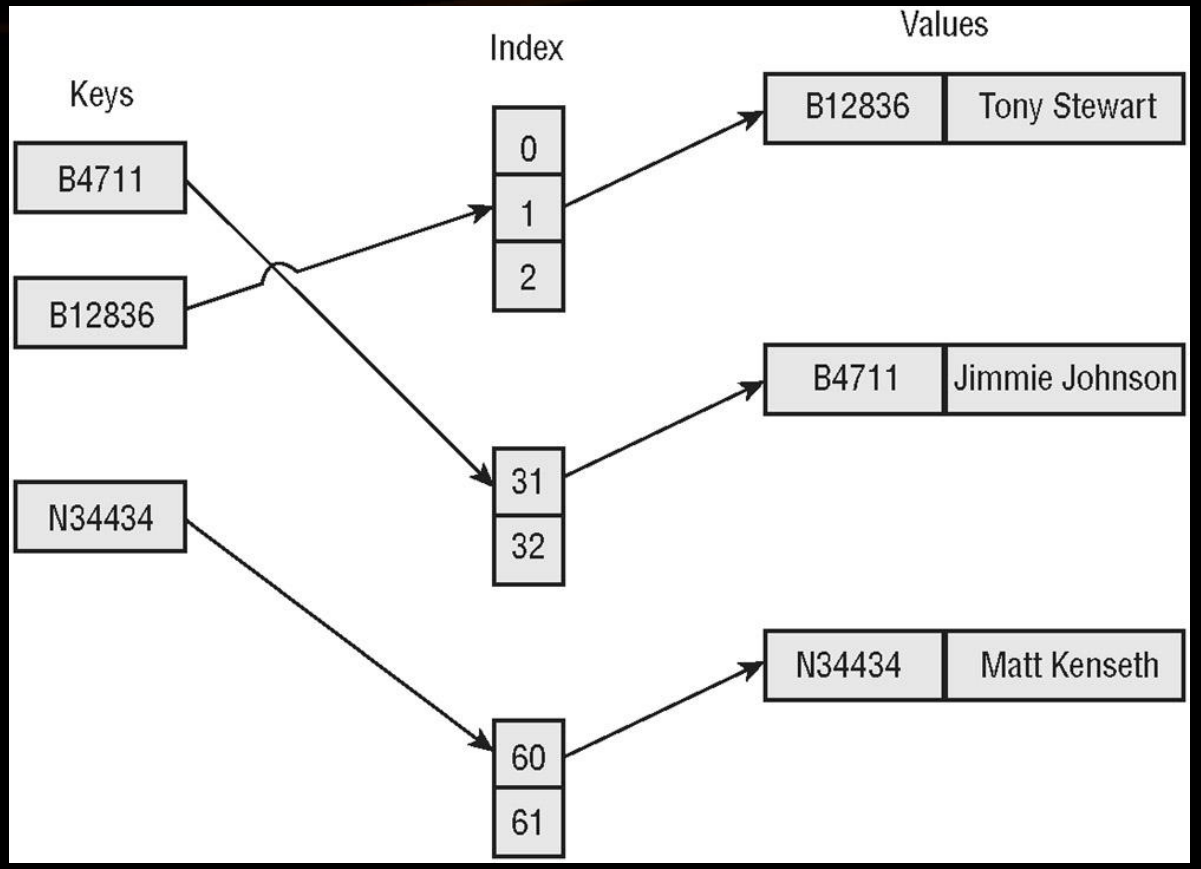
HashSet<E> and TreeSet<E>

Sets in Java

- Sets in Java keep unique elements
  - Like lists but duplicated elements are stored only once
- **`HashSet<E>`**
  - Keeps a set of elements in a hash-tables
  - The elements are randomly ordered (by their hash code)
- **`TreeSet<E>`**
  - Keeps a set of elements in a red-black ordered search tree
  - The elements are ordered incrementally

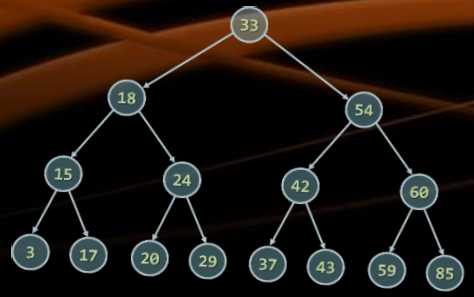
How hash sets are stored in the memory

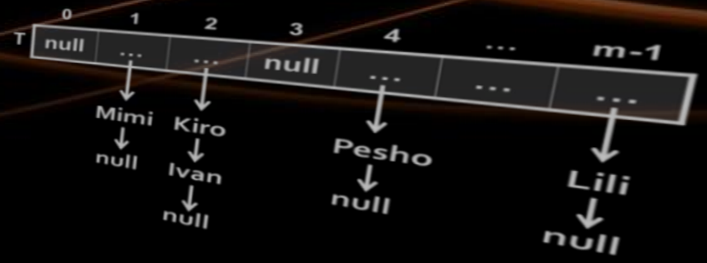
- Reference data type
  - Variable “names” holds hash indexes that point to Objects in the heap as values.
  - Each Object has an address that points to a value in the memory.

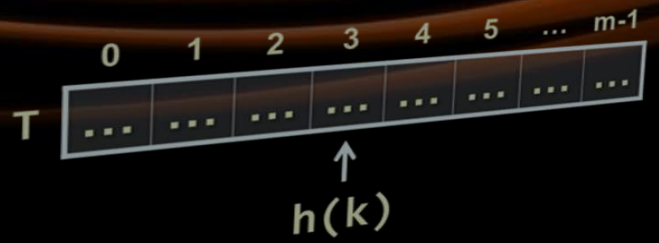


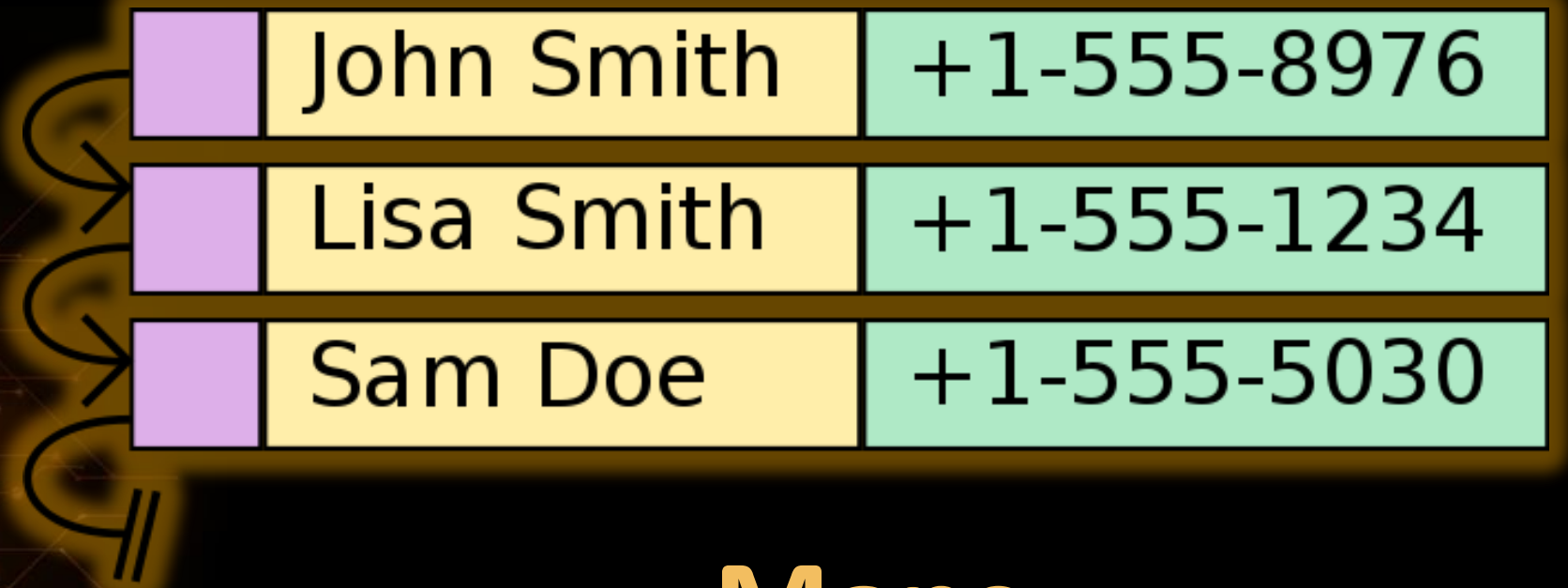
HashSet<E> and TreeSet<E> – Examples


```
Set<String> set = new TreeSet<String>();
set.add("Pesho");
set.add("Tosho");
set.add("Pesho");
set.add("Gosho");
set.add("Maria");
set.add("Alice");
set.remove("Pesho");
System.out.println(set); // [Alice, Gosho, Maria, Tosho]
```


33
18
54
15
24
42
60
3
17
20
29
37
43
59
85


T
0
1
2
3
4
...
m-1
null
null
Mimi
null
Kiro
Ivan
null
Pesho
null
Lili
null


T
0
1
2
3
4
5
...
m-1
h(k)


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

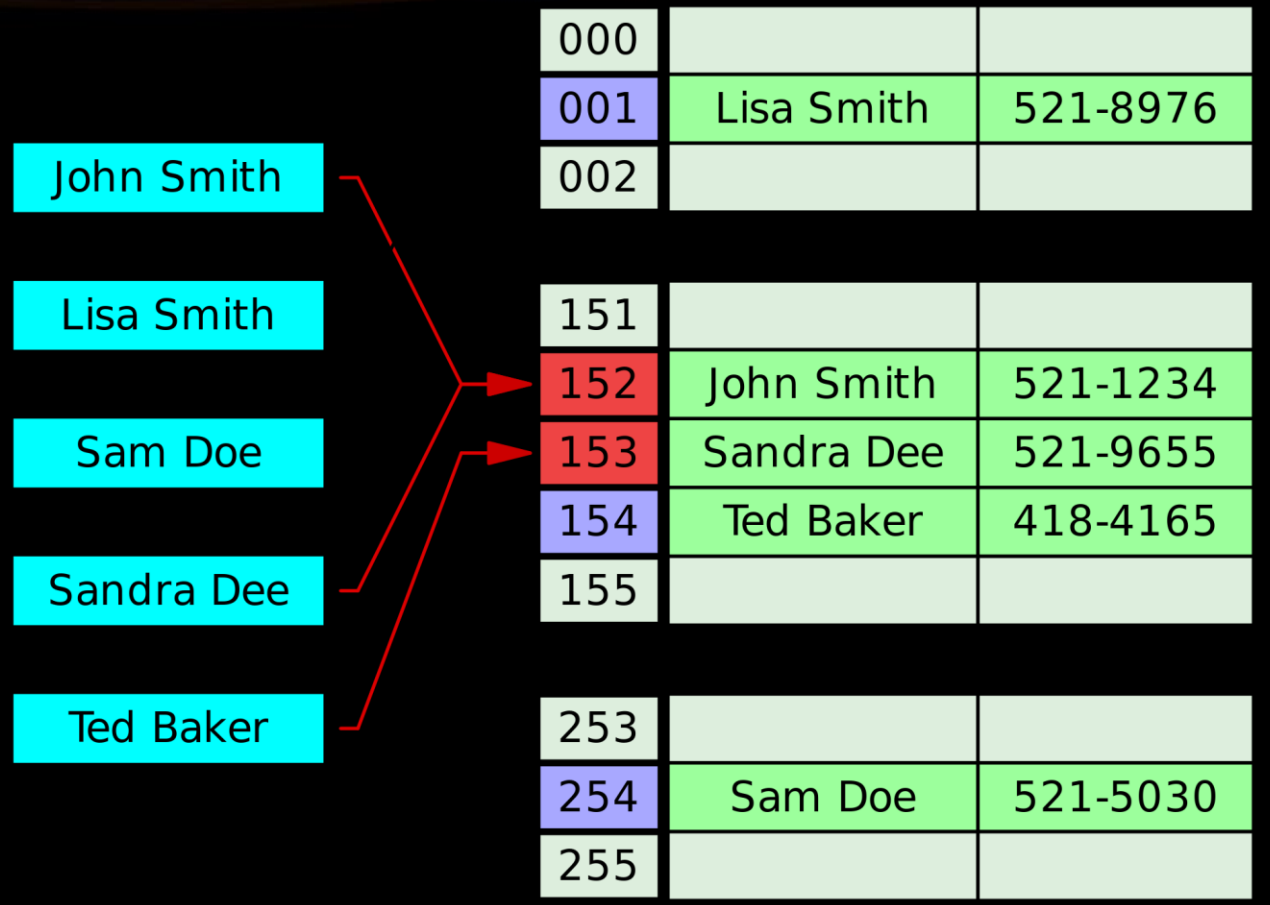
Maps

Maps in Java

- Maps in Java keep unique <key, value> pairs
- **`HashMap<K, V>`**
  - Keeps a map of elements in a hash-table
  - The elements are randomly ordered (by their hash code)
- **`TreeMap<K, V>`**
  - Keeps a set of elements in a red-black ordered search tree
  - The elements are ordered incrementally by their key

How hash maps are stored in the memory

- Reference data type
  - Variable “phonebook” holds hash indexes that point to keys in the heap as values.
  - Each key points to a value in the memory.



HashMap<K, V> – Examples

- Counting words occurrences in a list:

```
String[] words = { "yes", "hi", "hello", "hi", "welcome",
  "yes", "yes", "welcome", "hi", "yes", "hello", "yes" };

Map<String, Integer> wordsCount = new HashMap<String, Integer>();
for (String word : words) {
  Integer count = wordsCount.get(word);
  if (count == null) {
    count = 0;
  }
  wordsCount.put(word, count+1);
}

System.out.println(wordsCount); // {hi=3, yes=5, hello=2, welcome=2}
```

TreeMap<K, V> – Examples

- Students and their grades

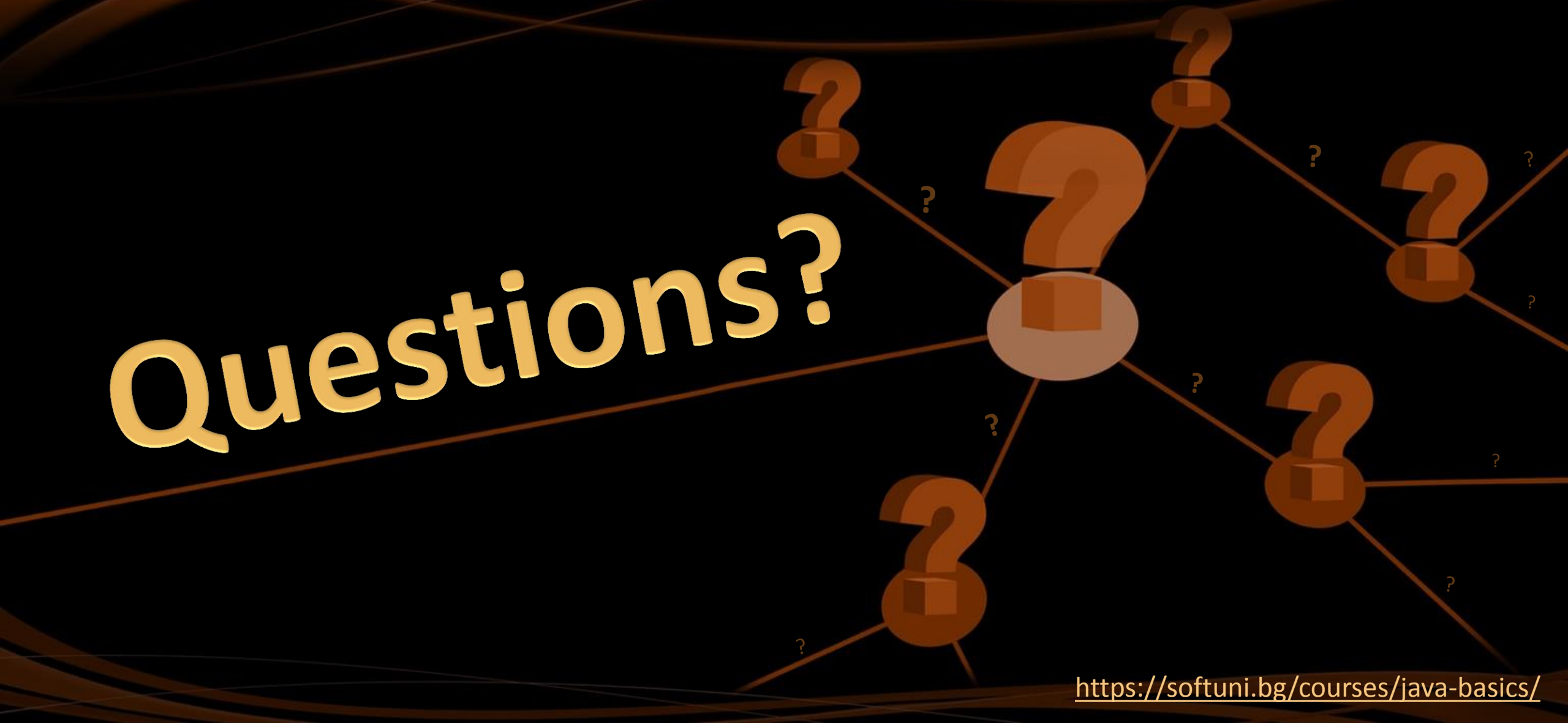
```
HashMap<String, ArrayList<Integer>> grades = new HashMap<>();
grades.put("Peter", new ArrayList<>(Arrays.asList(5)));
grades.put("George", new ArrayList<>(Arrays.asList(5, 5, 6)));
grades.put("Maria", new ArrayList<>(Arrays.asList(5, 4, 4)));
grades.get("Peter").add(6);
grades.get("George").add(6);

for (String key : grades.keySet()) {
  System.out.println("" + key + " -> " + grades.get(key));
}
```

Summary

- Arrays, Strings and Collections:
  1. Arrays: `int[]`, `String[]`, etc.
  2. Strings: `String str = "Hello";`
  3. Lists: `ArrayList<E>`
  4. Sets: `HashSet<E>`, `TreeSet<E>`
  5. Maps: `HashMap<K, V>`, `TreeMap<K, V>`

Java Collections Basics

SOFTWARE UNIVERSITY FOUNDATION



https://softuni.bg/courses/java-basics/

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
  - "Fundamentals of Computer Programming with Java" book by Svetlin Nakov & Co. under CC-BY-SA license
  - "C# Basics" course by Software University 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 @ YouTube
  - youtube.com/SoftwareUniversity
- Software University Forums – forum.softuni.bg