Recap: Loops

- Repeating things

Repeat as long as something holds

Pattern

```
1 while (EXPRESSION) {
2   // some code
3 }
```

boolean

Example

```
1 String userInput;
2 while (userInput != "exit") {
3   // some code
4   // ask user for command
5   userInput = ...
6 }
```

Repeat for specified number of times

Pattern

```
1 for (INIT; CONDITION; UPDATE) {
2   // some code
3 }
```

Example

```
1 for (int i = 0; i < 17; i++) {
2   // some code
3 }
```

INIT COND UPDATE i=i+1

demo

Exercise 5

UNIVERSITÄT
ZU KÖLN

Session 6: Arrays and Strings

Softwaretechnologie: Java 1

Nils Reiter
nils.reiter@uni-koeln.de

November 22, 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 1

Arrays

Introduction

- So far: Single variables store single values
  - ```
    int i = 5; //one int value in one int variable
    ```
    ```

Introduction

- So far: Single variables store single values
  - `int i = 5; //one int value in one int variable`
- New: Array (German, rarely used: “Feld”)
  - Stores a collection of values – a data structure
  - Fixed number of values
  - All values have the same type

Introduction

- So far: Single variables store single values
  - `int i = 5; //one int value in one int variable`
- New: Array (German, rarely used: “Feld”)
  - Stores a collection of values – a data structure
  - Fixed number of values
  - All values have the same type
  - New syntactic element: square brackets `[]`

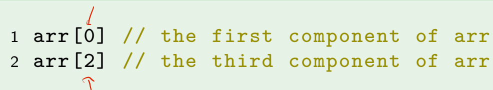
Using Arrays

- Array components are enumerated
  - `ARRAY_NAME [ INDEX ]` allows access to a specific component
  - 0-Base: The first component has the index number 0

Using Arrays

- Array components are enumerated
  - `ARRAY_NAME [ INDEX ]` allows access to a specific component
  - 0-Base: The first component has the index number 0

Example



```
1 arr[0] // the first component of arr
2 arr[2] // the third component of arr
```

Using Arrays

- Array components are enumerated
  - `ARRAY_NAME [ INDEX ]` allows access to a specific component
  - 0-Base: The first component has the index number 0

Example

```
1 arr[0] // the first component of arr
2 arr[2] // the third component of arr
```

- Array components can be used in expressions, similar to variable names

Example

```
1 arr[0] = 5;
2 int b = arr[2] + 4;
```

Array Creation

- With `new`
  - `int[] a = new int[5];`
  - Filled with 0s

Array Creation

- With `new`
  - `int[] a = new int[5];`
  - Filled with 0s
- As literal
  - `int[] a = new int[] {1, 2, 3};`
    - In this case, the type can be inferred, so we can skip `new int[3]`: `int[] a = {1, 2, 3};`
  - `someFunction(new int[] {1,2,3})` – literal array as argument

Array Length

- The number of components of an array is fixed at run-time

```
1 int a = 5;
2 a = a + (int) Math.random();
3 int[] arr = new int[a];
```

Array Length

- The number of components of an array is fixed at run-time

```
1 int a = 5;
2 a = a + (int) Math.random();
3 int[] arr = new int[a];
```

- There is no way to increase the length
  - ...except to create a new array and copy items from the old to the new

Array Length

- The number of components of an array is fixed at run-time

```
1 int a = 5;
2 a = a + (int) Math.random();
3 int[] arr = new int[a];
```

- There is no way to increase the length
  - ...except to create a new array and copy items from the old to the new
- Because the length is important, there is a way to access it: `arr.length`

demo

ArrayDemo

Array as a Type

- Array is not a type
- `int`-Array is a type
  - Type identified: `int[]`

Array as a Type

- Array is not a type
- `int`-Array is a type
  - Type identified: `int[]`
- Length is not part of the type
  - I.e., not known at compile time

Array as a Type

- Array is not a type
- `int`-Array is a type
  - Type identified: `int[]`
- Length is not part of the type
  - I.e., not known at compile time

```java
public static void main(String[] args) {
  // ...
}
```

- As `main` is a function, `args` is an argument of type `String[]`
  - ➔ A collection of character sequences

Arrays in Memory

	0	1	2	3	4	5	6	7	8	9
0x										
1x										
2x						0	0	0	0	0
3x	1	0	1							
4x										
5x										
6x										
7x										

17 + 1 · 8 = 25

```
1 byte[] bArray = new byte[3];
2 bArray[1] = (byte) 5; // green cells
```

Arrays in Memory

	0	1	2	3	4	5	6	7	8	9
0x										
1x										
2x						0	0	0	0	0
3x	1	0	1							
4x										
5x										
6x										
7x										

```
1 byte[] bArray = new byte[3];
2 bArray[1] = (byte) 5; // green cells
```

- Exact address of each component can be computed
- If starting address and length of each component are known
- “Direct access” (= fast)

Session 3
October 25

Primitive Data Types and Objects

Two kinds of types

- Primitive data types: Built into the language
  - Type names are reserved keywords in Java
    - I.e., it's not allowed to use them as an identifier
  - Convention: Lower cased
- Non-primitive data types ("reference types"): Established in the library
  - Type names are defined by library authors
  - Convention: Upper cased
  - Reference types can also be defined by us
    (in the form of classes, to be discussed later)

int[]

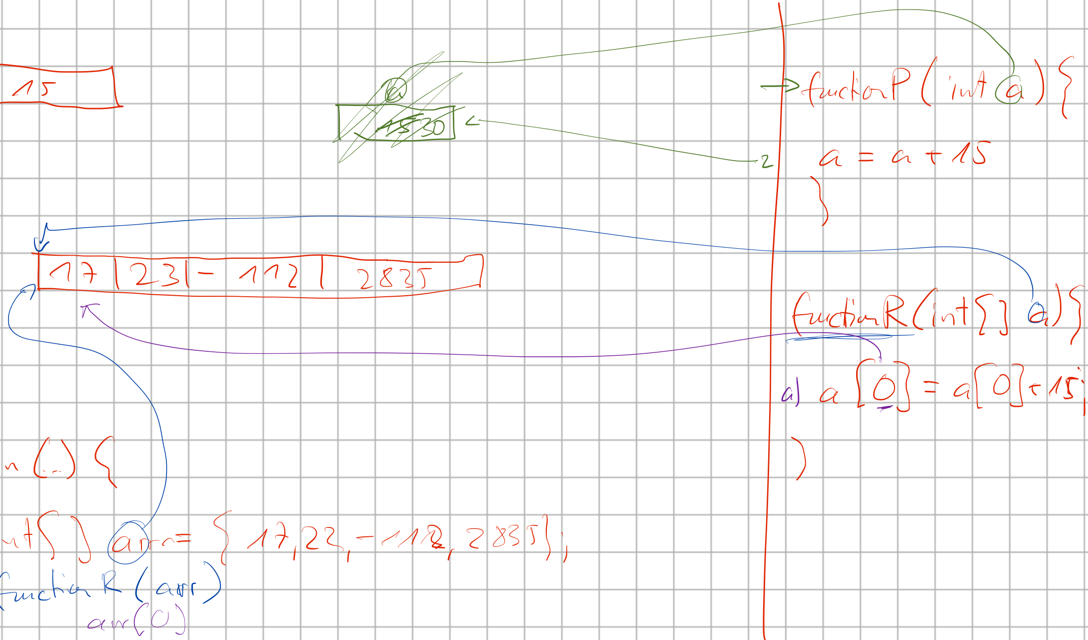
Array is a Reference Type

```
1 // Primitive type
2 int x = 5;
3 int y = x;
4 y = y + 2; // y now contains 7,
5            // x still 5
6
7 // Reference type
8 int[] a = {1,2,3};
9 int[] b = a;
10 a[0] = 0; // b[0] is now also 0!
11           // because a and b are
12           // references to the same
13           // memory region
```

- Primitive types: Values (of memory regions) are passed
- Reference types: References (to memory regions) are passed
  - If you change a reference type within a function, it's changed outside of the function
- Everything from now on is a reference type

demo

ReferenceTypeDemo


15
15 30
functionP (int a) {
a = a + 15
}
17 | 23 | -112 | 2835
functionR (int[] a) {
a) a[0] = a[0] + 15;
}
main (...) {
int[] arr = {17, 22, -112, 2835};
functionR (arr)
arr[0]
}

Comparing Reference Types

```
1 int[] a = {1,2,3};
2 int[] b = {1,2,3};
3
4 if (a == b) {
5   System.out.println("Arrays are equal");
6 } else {
7   System.out.println("Arrays are not equal");
8 }
```

- Which output do we get?

Comparing Reference Types

```
1 int[] a = {1,2,3};
2 int[] b = {1,2,3};
3
4 if (a == b) {
5   System.out.println("Arrays are equal");
6 } else {
7   System.out.println("Arrays are not equal");
8 }
```

- Which output do we get?
- If reference types are compared with `==` & co., we compare the memory location
  - Not the content
- To compare the content: `Arrays.equals(a1, a2)`

javadoc

Comparing Reference Types

```
1 int[] a = {1,2,3};
2 int[] b = {1,2,3};
3
4 if (a == b) {
5   System.out.println("Arrays are equal");
6 } else {
7   System.out.println("Arrays are not equal");
8 }
```

- Which output do we get?
- If reference types are compared with `==` & co., we compare the memory location
  - Not the content
- To compare the content: `Arrays.equals(a1, a2)` javadoc
  - ⚠ Using some functions requires importing them first
    - Eclipse suggestions are mostly correct, more on this next week

Array Copying

```
1 // Reference type
2 int[] a = {1,2,3};
3 int[] b = a; // does not create a copy of a
4 b[0] = 0;
5
6 int[] c = a.clone(); // creates a copy
7 c[2] = 10; // no change in a
```

- Copying an array: `someArray.clone()`
  - This is a method (note the parentheses)

Methods and Fields

- `length` is stored with an array
  - Calling `someArray.length` does not execute code, it's just a variable access
- `clone()` is a function associated with this array
  - Calling `someArray.clone()` runs this function in the context of this array
  - Method: A function with benefits

Methods and Fields

- `length` is stored with an array
  - Calling `someArray.length` does not execute code,
- `clone()` is a function associated with this array
  - Calling `someArray.clone()` runs this function in t
  - Method: A function with benefits
- Other methods are displayed by Eclipse

*ArrayDemo.java

```
1
2  public class ArrayDemo {
3
4      public static void main(String[] args) {
5          int[] a = {1,2,3};
6          int[] b = {1,2,3};
7
8          System.out.println(a == b);
9
10         a.
11     }
12
13 }
14
```

```
clone() : int[] - int[]
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
notify() : void - Object
notifyAll() : void - Object
toString() : String - Object
wait() : void - Object
wait(long timeoutMillis) : void - Object
wait(long timeoutMillis, int nanos) : void - Object
length : int - int[]
Press '^Space' to show Template Proposals
```

Array Patterns

Printing out an array:

```
1 System.out.println(Arrays.toString(array)); // Print out the array in readable form
```

Array Patterns

Printing out an array:

```
1 System.out.println(Arrays.toString(array)); // Print out the array in readable form
```

Iterating over an array:

```
1 for (int i = 0; i < array.length; i++) {
2   // access each array element with array[i]
3 }
```

Array Patterns

Printing out an array:

```
1 System.out.println(Arrays.toString(array)); // Print out the array in readable form
```

Iterating over an array:

```
1 for (int i = 0; i < array.length; i++) {
2   // access each array element with array[i]
3 }
```

Two-dimensional array (= a matrix or table):

```
1 int[][] matrix = new int[17][25];
2 int[0][0] = 15;
3 for (int i = 0; i < matrix.length; i++) {
4   for (int j = 0; j < matrix[i].length; j++) {
5     // cells can be accessed with matrix[i][j]
6   }
7 }
```

Section 2

Strings/Zeichenketten

Introduction

- Represents character sequences
- A reference type
- Internally: An array of `char`-values (mostly)

```
1 String s = "Hi there!"; // String literal with double quotes
```

String Operations

- Concatenation (“Aneinanderhängen”)

```
1 String s1 = "Hi";
2 String s2 = "there";
3 String s = s1 + s2; // s now contains "Hithere"
```

  - `+` is the only regular math operator you can use with strings

String Operations

- Concatenation (“Aneinanderhängen”)

```
1 String s1 = "Hi";
2 String s2 = "there";
3 String s = s1 + s2; // s now contains "Hithere"
```

  - `+` is the only regular math operator you can use with strings
- Length: `s.length() //returns 7 (as an int)`
  - Note the round brackets
  - Gives us the length in characters, not in bytes

String Operations

- Concatenation (“Aneinanderhängen”)

```
1 String s1 = "Hi";
2 String s2 = "there";
3 String s = s1 + s2; // s now contains "Hithere"
```

  - `+` is the only regular math operator you can use with strings
- Length: `s.length() //returns 7 (as an int)`
  - Note the round brackets
  - Gives us the length in characters, not in bytes
- Convert case
  - `s2.toLowerCase(); //returns "hi"`
  - `s2.toUpperCase(); //returns "HI"`

Strings and Other Types

- All primitive types can be converted into a string
  - `System.out.println()` does this automatically, as we have seen
- Conversion done implicitly:

```
1 int i = 2024;
2 String s = "Hallo";
3 System.out.println(s + i); // implicit conversion of i,
4                            // then concatenation
```

Strings and Other Types

- All primitive types can be converted into a string
  - `System.out.println()` does this automatically, as we have seen
- Conversion done implicitly:

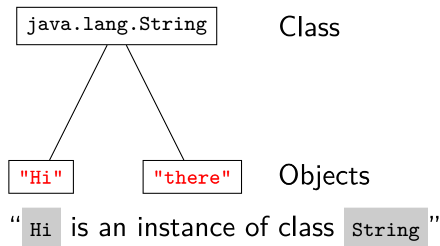
```
1 int i = 2024;
2 String s = "Hallo";
3 System.out.println(s + i); // implicit conversion of i,
4                            // then concatenation
```

- Explicit conversion
  - Many functions `String.valueOf(ARG)`
  - Take all primitive types as arguments

The class String

- `java.lang.String`: Our first class
- Classes and Objects:
  Object-oriented programming

More on classes and objects: Next week(s)



`main` Function

```java
public class MyProgram
  public static void main(String[] args) {
    // do stuff
  }
}
```

- Entry point for every Java program
- A regular function, with arguments

How to set the arguments?

`main` Function

```
1 public class MyProgram
2   public static void main(String[] args) {
3     // do stuff
4   }
5 }
```

- Entry point for every Java program
- A regular function, with arguments

How to set the arguments?

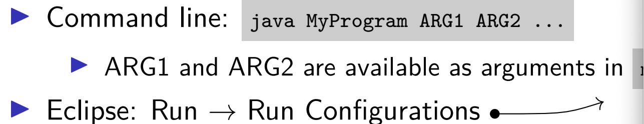
- Command line: `java MyProgram ARG1 ARG2 ...`
    - ARG1 and ARG2 are available as arguments in `main`

`main` Function

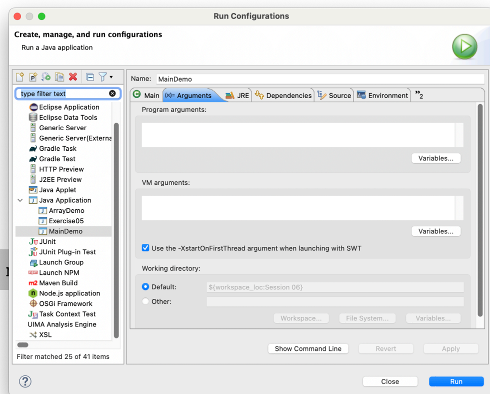
```
1 public class MyProgram
2   public static void main(String[] args) {
3     // do stuff
4   }
5 }
```

- Entry point for every Java program
- A regular function, with arguments

How to set the arguments?



- Command line: `java MyProgram ARG1 ARG2 ...`
  - ARG1 and ARG2 are available as arguments in
- Eclipse: Run → Run Configurations



demo

MainDemo

What can we do with Strings?

...and how do we find out?

java.lang.String

- Javadoc
  - `char charAt(int index);`
  - `int compareTo(String anotherString)`
  - `String concat(String str)`
  - `boolean endsWith(String suffix)`
  - `boolean isEmpty()`
  - `String substring(int beginIndex, int endIndex)`
  - ...

What can we do with Strings?

...and how do we find out?

- Javadoc

  java.lang.String

  - `char charAt(int index);`
  - `int compareTo(String anotherString)`
  - `String concat(String str)`
  - `boolean endsWith(String suffix)`
  - `boolean isEmpty()`
  - `String substring(int beginIndex, int endIndex)`
  - ...
- How to use them? `INSTANCE.METHOD(ARGUMENTS)`
  - Eclipse suggests possible methods/fields in a small window
  - Methods are associated with the specific instance before the `.`

Section 3

Exercise