Recap: Arrays

- Data structure to store multiple values
- Syntax

```java
// create a new array with 4 components
int[] arr = new int[4];   ← arr[3]

// access 2nd component
System.out.println(arr[1]);
arr[1] = 7;
```

- Properties
  - Length is fixed
  - All components are of the same type (e.g., `int`)
  - A reference type

Section 1

Exercise 6

– Audiotestpause –

UNIVERSITÄT ZU KÖLN

Session 7: Strings and Comments, Ascii Art 2.0

Softwaretechnologie: Java 1

Nils Reiter
nils.reiter@uni-koeln.de

December 6, 2023

INSTITUT FÜR DIGITAL HUMANITIES UNIVERSITÄT ZU KÖLN

Section 2

Comments and Javadoc

Comments

- Ignored by the compiler
- Information for us humans

Comments

- Ignored by the compiler
- Information for us humans

Two and a Half Types

```
// This comment ends when the line ends

/* This comments ends with */

/*
We can include text that spans
multiple lines

+ a variant of this (see below)
*/
```

Comments

Example

```
public class Example {

  public static void main(String[] args) {
    // stores how much users want to withdraw
    int amount = 1500;

    /* the next lines are supposed to calculate
       the third root of amount, I took the idea from
       http://www...
    */
    int temp = 3;
    amount = amount / temp;
    // TODO: Implement me!
  }
}
```

Commenting

- No fixed rules what to comment
- Helpful: Your intentions, complex expressions, non-trivial functions
- Avoid commenting trivial things
- Keep comments up to date

Javadoc

- Comments, so far: `/* ... */` and `// ...`
  - Implementation comments about your code

Javadoc

- Comments, so far: `/* ... */` and `// ...`
  - Implementation comments about your code
- New comment type: `/** ... */`
  - API comment for other programmers about a function/class/method
  - Not about specific lines, but the entire function

Javadoc

- Comments, so far: `/* ... */` and `// ...`
    - Implementation comments about your code
- New comment type: `/** ... */`
    - API comment for other programmers about a function/class/method
    - Not about specific lines, but the entire function
- API comments can be extracted to an HTML page
    - All Java classes/functions/methods have such a documentation
    - Javadoc: Math.random()
    - Reading documentation is an integral part of programming – get used to it

Javadoc

Javadoc

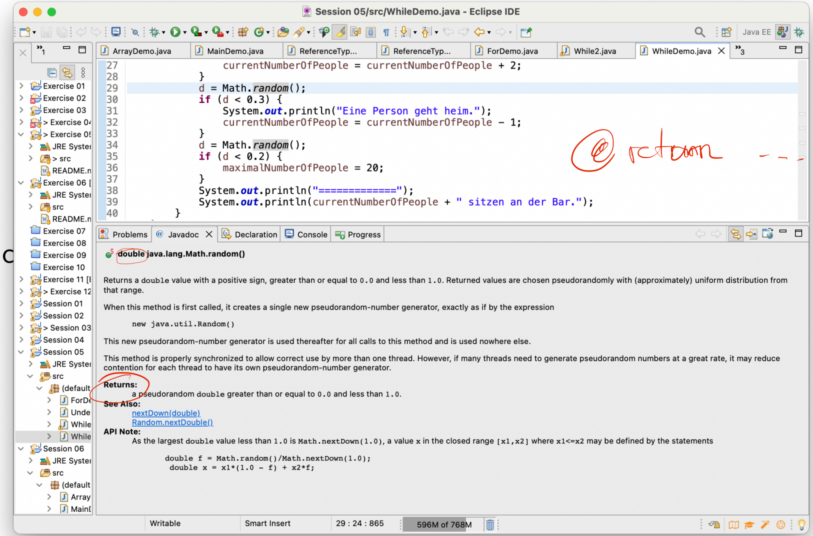
Eclipse

- Javadoc comments directly displayed by Eclipse

Javadoc

Eclipse

- Javadoc c



```java
                currentNumberOfPeople = currentNumberOfPeople + 2;
            }
            d = Math.random();
            if (d < 0.3) {
                System.out.println("Eine Person geht heim.");
                currentNumberOfPeople = currentNumberOfPeople - 1;
            }
            d = Math.random();
            if (d < 0.2) {
                maximalNumberOfPeople = 20;
            }
            System.out.println("============");
            System.out.println(currentNumberOfPeople + " sitzen an der Bar.");
        }
```

@ return - - -

double java.lang.Math.random()

Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`. Returned values are chosen pseudorandomly with (approximately) uniform distribution from that range.

When this method is first called, it creates a single new pseudorandom-number generator, exactly as if by the expression

```
new java.util.Random()
```

This new pseudorandom-number generator is used thereafter for all calls to this method and is used nowhere else.

This method is properly synchronized to allow correct use by more than one thread. However, if many threads need to generate pseudorandom numbers at a great rate, it may reduce contention for each thread to have its own pseudorandom-number generator.

Returns:
a pseudorandom `double` greater than or equal to `0.0` and less than `1.0`.

See Also:
nextDown(double)
Random.nextDouble()

API Note:
As the largest `double` value less than `1.0` is `Math.nextDown(1.0)`, a value `x` in the closed range `[x1,x2]` where `x1<=x2` may be defined by the statements

```
double f = Math.random()/Math.nextDown(1.0);
double x = x1*(1.0 - f) + x2*f;
```

Javadoc

Eclipse

- Javadoc comments directly displayed by Eclipse
- Eclipse can generate Javadoc HTML files
  - Menu > Project > Generate Javadoc ...

Section 3

Strings/Zeichenketten

Introduction

$\text{String} = \text{char}[\,]$

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

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

String Operations

- Concatenation (“Aneinanderhängen”)

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

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

String Operations

- Concatenation ("Aneinanderhängen")

```
String s1 = "Hi";
String s2 = "there";
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")

  ```
  String s1 = "Hi";
  String s2 = "there";
  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:

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

Strings and Other Types

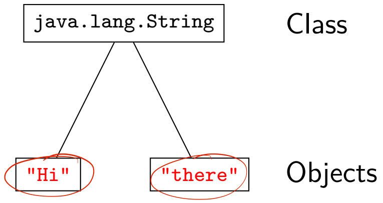
- All primitive types can be converted into a string
  - `System.out.println()` does this automatically, as we have seen
- Conversion done implicitly:

```java
int i = 2024;
String s = "Hallo";
System.out.println(s + i); // implicit conversion of i,
                           // 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



"`Hi` is an instance of class `String`"

`main` Function

```
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

```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?

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

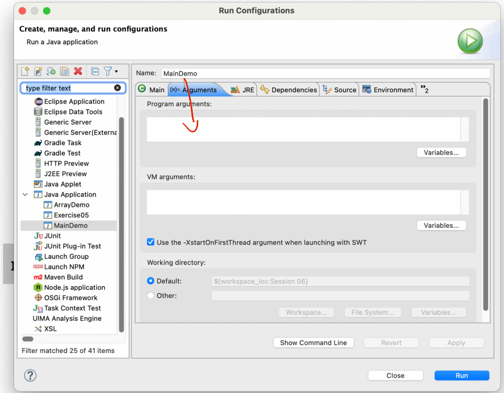
`main` Function

```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?

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



Name: MainDemo

Program arguments:

VM arguments:

[✓] Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

[✓ Default:] ${workspace_loc:Session 06}

Other:

demo

MainDemo

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)`
  - ...

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 4

ASCII Art 2.0

ASCII Art 2.0

- So far: All functions print out lines of the image directly
- Next version: Should be possible to manipulate the image as a whole (e.g., invert it)
- To do
  - Change all functions such that they return a string instead of printing one
  - Invert the image

demo

AsciiArt

Section 5

Exercise