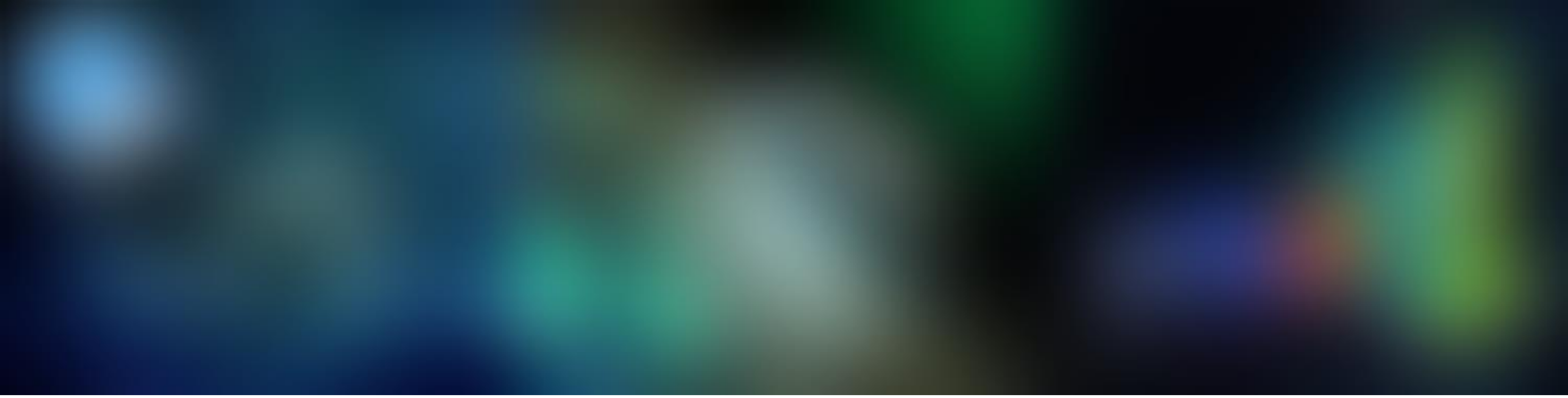
Object-Oriented Programming In Mechatronic Systems

Summer School 2018

Module 2 – Basics of the Java Programming Language

Aachen, Germany

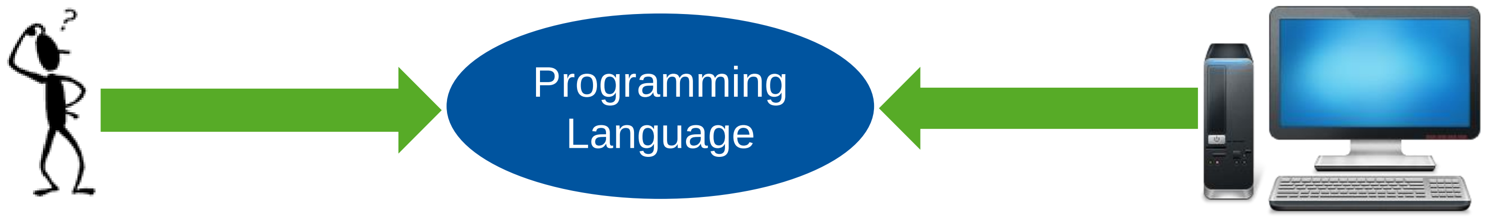
Cybernetics Lab IMA & IfU
Faculty of Mechanical Engineering
RWTH Aachen University



Recap

Recap Module 1

Interface between Human and Computer



Still, both have different requirements:

Human:

- Natural language
- Legibility
- Expressiveness

Computer

- Simple translation into machine code
- Efficiency of the generated code

Recap Module 1

Module 1 was about the basics of Java

- The structure of a Java program, e.g. a class definition
- Variables and how to define them (e.g. instance variables)
- Primitive datatypes like `int` or `char`
- The first method called `main`

… and how to start developing using the Eclipse IDE

- Creating a new project in Eclipse
- ... creating a class in Eclipse
- … compile and execute applications in Eclipse

Module 2 will be about control flow statements and arrays!

Recap Module 1

What are variables?

- A container, a box or a cup. It contains something.
- They come in different kinds
- They got a name

Examples

- `short numberOfEngines = 5;`
- `double temperature = 23.7;`
- `boolean engineStarted = true;`
- `char c = 'e';`
- `int depth = -343535;`

Recap Module 1

Four Primitive Data Types in Java

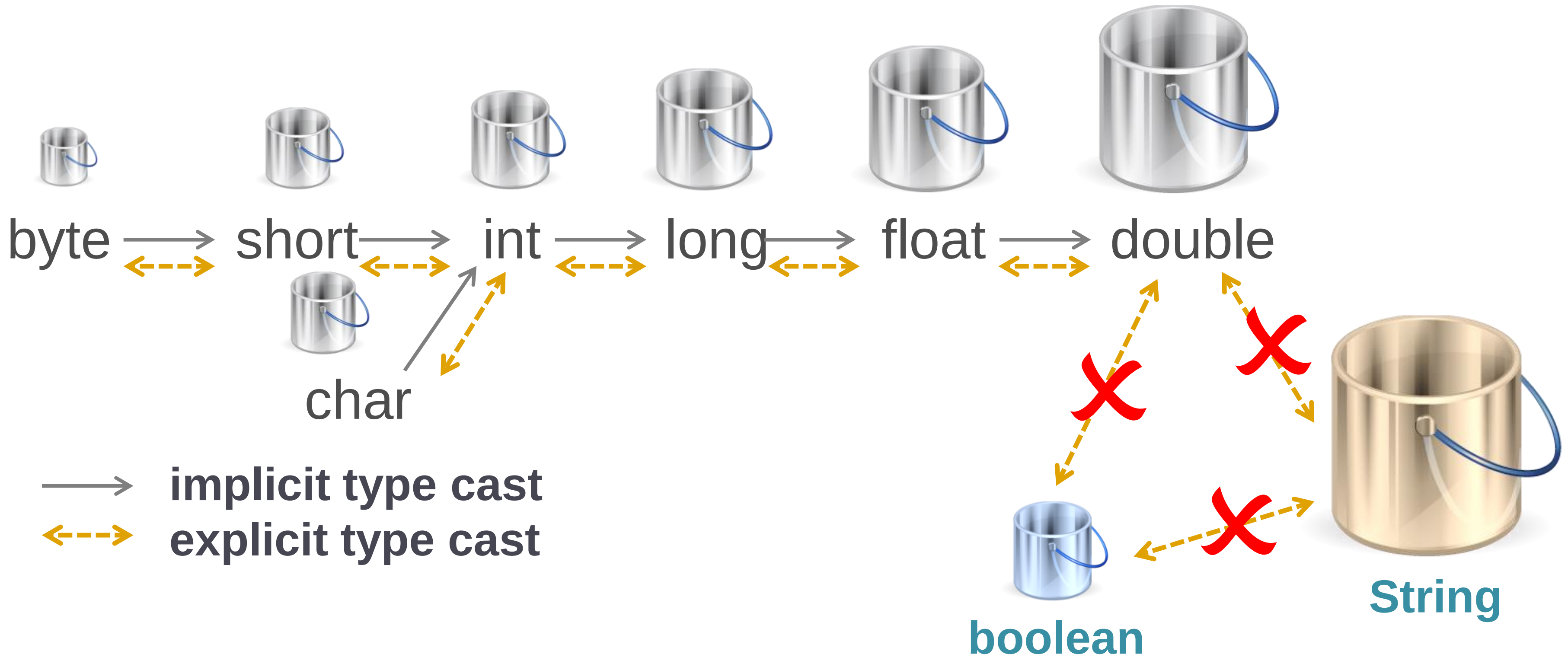
- boolean, char, integer and floating point
- They got a default value
- They only hold one value

Data Type	Example	Keyword
Logical value	true, false	`boolean`
Single character	a, b, …	`char`
Whole number	1, -3, 87, …	`byte, short, int, long`
Real number	-2.6, 9.4, …	`float, double`

For details (e.g. max or min values) see:
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Recap Module 1

Type cast overview



Control Flow Statements

Control Flow Statements

Control flow

- Statements are generally executed from top to bottom
- Control flow statements break up the flow
- Enable that particular blocks of code are executed conditionally

Control flow statements in Java

- There are three types of control flow statements in Java
- Decision-making statements (`if-then, if-then-else, switch`)
- Looping statements (`for, while, do-while`)
- Branching statements (`break, continue, return`)

Control Flow Statements

if-then(-else) statement (simple)

- Certain section of code is only executed if test evaluates to `true`
- If test section evaluates to `false`, else block is executed
- Nesting possible
- Else block is optional

Structure of a simple if-then-(-else) statement

```
if (<condition>){
  statement(s)
}
else {
  statements(s)
}
```

Control Flow Statements

A simple if-then(-else) example

```
public void break() {
    if (carIsMoving) {
        speed = 0;
    }
    else {
        System.out.println("Car has already stopped!");
    }
}
```

If the car is moving then set its speed to zero. Otherwise print a message to the command line which says that the car has already stopped.

Control Flow Statements

A complex if-then(-else) statement

```
if (<condition 1>){
     statement(s)
}
else if (<condition 2>){
     statements(s)
}
…
else if (<condition N>)
     statements(s)
}
else {
     statements(s)
}
```

Control Flow Statements

A complex example

```
public class IfElseDemo {
      public static void main(String[] args) {
        int testscore = 76;
        char grade;
        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

Control Flow Statements

Switch statement I

- Arbitrary number of execution paths
- Only discrete values are allowed
- Variable types: `byte, short, int, char`
- Tests expression based on a single integer or character

Structure of a switch statement

```
switch(<variable>){
        case <value1>:          //instruction
        case <value2>:          //instruction
        case <value3>:          //instruction
        …
        default:                //instruction
}
```

Control Flow Statements

Switch statement II

- `Case` translates to “search for match and then execute every following instruction” (aka *fall through*)
- `Break` terminates enclosing switch statement
- `Default` handles values not handled by case sections

Example of a switch statement with break

```
int gear = 2;
String gearString;
switch (gear) {
      case 1:  gearString = "low"; break;
      case 2:  gearString = "medium"; break;
      case 3:  gearString = "high"; break
      default: gearString = "undefined‟; break;
}
System.out.println(gearString);
```

What’s the output?

Output: medium

Control Flow Statements

Switch statement II

- `Case` translates to “search for match and then execute every following instruction” (aka *fall through*)
- `Break` terminates enclosing switch statement
- `Default` handles values not handled by case sections

Example of a switch statement without break

```
int gear = 2;
String gearString;
switch (gear) {
      case 1:   gearString = "low";
      case 2:   gearString = "medium";
      case 3:   gearString = "high";
      default: gearString = "undefined";
}
System.out.println(gearString);
```

What’s the output?

Output: undefined

Control Flow Statements

for statement

- Aka the "*for loop*"
- Provides a way to iterate over a range of values
- Terminates if a certain condition applies

Structure of a for statement

```
for (initialization; termination; increment) {
     statement(s)
}
```

- Initialization expression initializes the loop; executed once.
- Loop terminates if termination evaluates to `false`
- The increment is invoked after each iteration. Can also be a decrement.

Control Flow Statements

Example of a for statement I

```
public class LoopDemo{

    public static void main(String[] args) {

    for (int i = 0; i < 15; i++){

            System.out.println("Loop "+i);

    }

}
```

```
Output:
Loop 0
Loop 1
…
Loop 14
```

Control Flow Statements

Example of a for statement II

```
public class LoopDemo{

    public static void main(String[] args) {

    for (int i = 10; i > 0; i--){

          System.out.println("Countdown "+i);

    }

}
```

Output:
Countdown 10
Countdown 9
…
Countdown 1

Control Flow Statements

Example of an (odd) for statement III

```
public class LoopDemo{

    public static void main(String[] args) {

    for ( ; ; ){

        System.out.println("Loop");

    }

}
```

The three expressions are optional!

Infinite Output:
Loop
Loop
…
Loop

Control Flow Statements

while statement

- Executes statements while particular expression is `true`
- If expression evaluates to `false` the execution stops
- The expression is evaluated **before** every execution

Structure of a while statement

```
while(expression) {
     statement(s)
}
```

Control Flow Statements

Example of a while statement

```
public class WhileDemo {

    public static void main(String[] args){

        int count = 1;

        while (count < 11) {
            System.out.println("Count: " + count);
            count++;
        }
    }
}
```

Output:
Count 1
Count 2
…
Count 10

Control Flow Statements

do-while statement

- Executes statements while particular expression is `true`
- If expression evaluates to `false` the execution stops
- The expression is evaluated **after** every execution. It always executes at least once!
- Notice the “**;**” after the while statement!

Structure of a do-while statement

```
do {
     statement(s)
}
while(expression);
```

Control Flow Statements

Example of a do-while statement

```
public class WhileDemo {

    public static void main(String[] args){

        int count = 1;

        do {
            System.out.println("Count: " + count);
        }
        while (count < 1);
    }
}
```

Control Flow Statements

Three branching statements

- `break`: Instantly terminates a `switch`, `for`, `while` or `do-while` execution
- `continue`: Skips the current iteration of a `for`, `while` or `do-while`
- `return`: Exits from the current method. Used to return a value in case of non void methods.

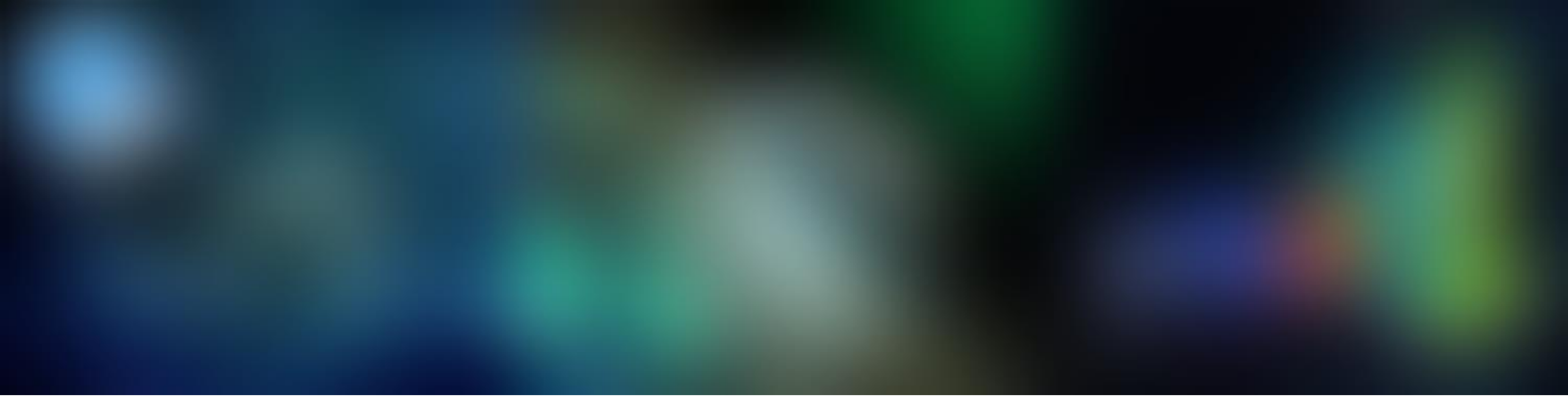
Example of continue statement

```
while (readNext(line)) {
  if (line.isEmpty() || line.isComment())
    continue;
  // More code here
}
```

Control Flow Statements

Summary of control flow statements

Statement	Features
`if-then-else`	Executes section of code if test evaluates to `true`. If test evaluates to `false` else branch is executed.
`switch`	Arbitrary number of execution paths are possible.
`while`	Continually executes a block of code while condition is true. Evaluates at top.
`while-do`	Evaluates at bottom. Runs at least once.
`for`	Loops over a range of values.
`break, continue, return`	Instantly terminates flow or continues the next iteration. Exits the current method.



Arrays

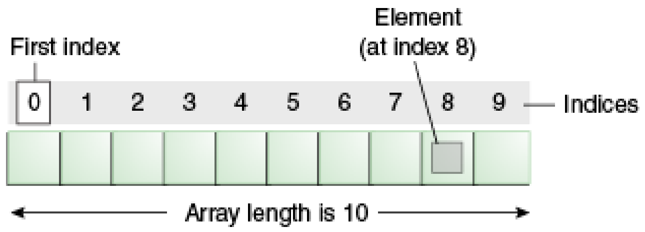
Arrays

Recap and Motivation

- Primitive data types (e.g. `int`) can only hold a single value
- E.g. `int val = 17;`

Array Features

- Arrays can hold multiple values (or *elements*)!
- Can only hold one data type, i.e. no mixture of data types (e.g. `int` **and** `char`)
- Length is established upon creation
- After that it’s fixed!
- Access to elements via *index*
- Index starts with 0. That is, the first array element has the index 0:



Arrays

Two ways of array creation (Examples)

- Initialize with values (e.g. six): `int[] array1 = {1,2,3,4,5,6};`
- Declaration (e.g. length nine): `int[] array2 = new int[9];`

Access to elements I

```
public static void main(String[] args) {
        int[] ar = new int[3];
        ar[0] = 100;
        ar[1] = 200;
        ar[2] = 300;
        System.out.println("Array value on pos 1:" +ar[0]);
        System.out.println("Array value on pos 2:" +ar[1]);
        System.out.println("Array value on pos 3:" +ar[2]);
}
```

Arrays

Access to elements II

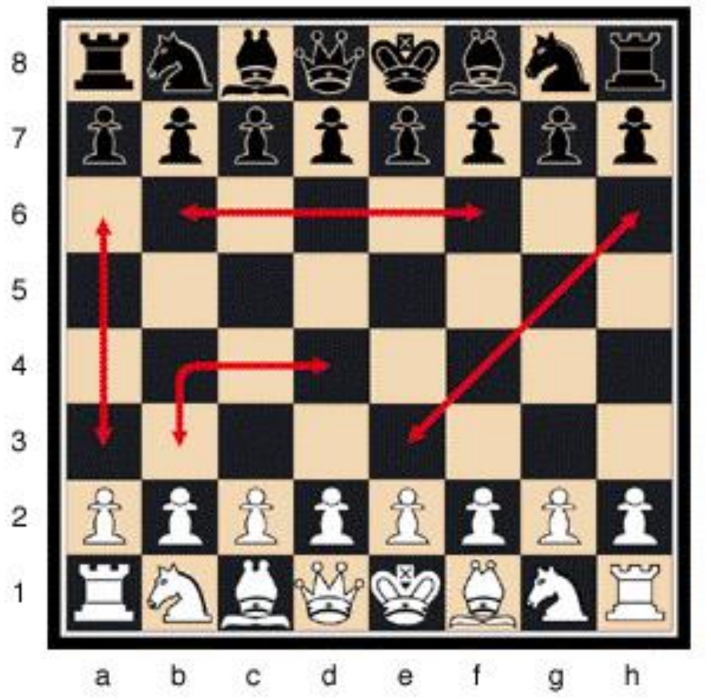
```
public static void main(String[] args) {

        int[] ar = new int[3];
        ar[0] = 100;
        ar[1] = 200;
        ar[2] = 300;

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

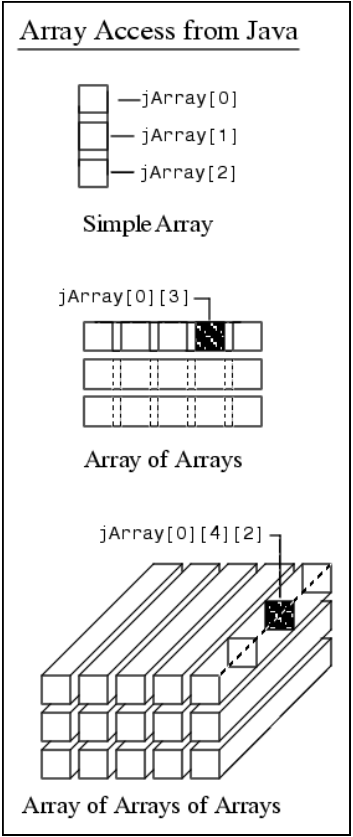
Determines the size of the array

Arrays

Multi dimensional arrays are possible

- Example: Positions on a chessboard or a Matrix are realized with arrays of arrays
- `int[][] chessboard = new int[8][8];`

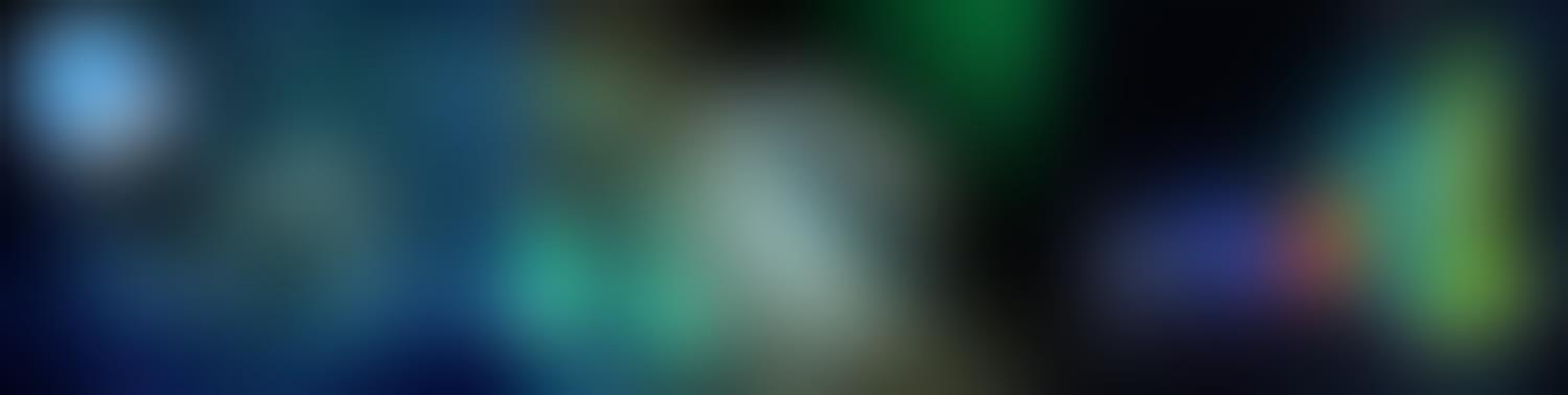




Arrays

For the curious mind … You can declare the following arrays:

- `byte[] anArrayOfBytes;`
- `short[] anArrayOfShorts;`
- `long[] anArrayOfLongs;`
- `float[] anArrayOfFloats;`
- `double[] anArrayOfDoubles;`
- `boolean[] anArrayOfBooleans;`
- `char[] anArrayOfChars;`
- `String[] anArrayOfStrings;`



Strings

Strings

Recap and Motivation

- Single characters can be presented as `char`
- E.g. `char c = 'd';`
- How can names, passwords etc. be presented?
- Naïve approach: As `char` arrays. Drawbacks (arrays are fixed length)

String features

- Keyword `String`
- Strings are denoted by quotation marks, e.g. `"A String"`
- Example: `String name = "RWTH";`

Thank you very much!