Einheit 2 Grundlegende Syntax und Semantik

Objektorientierte Programmierung

Lukas Wais

CODERS.BAY

Version: 24. September 2023

Inhaltsverzeichnis I

Kontrollfluss
- if Statements
- switch Anweisung

Benutzereingaben

Schleifen

Methoden und Funktionen
- Scope
- Überladen

Casting

Arrays
- Eindimensional
- Iteration
- Mehrdimensional
- Wert- und Referenzübergabe

Rekursion

Inhaltsverzeichnis II

Beispiele

Kontrollfluss

if-then

```
void applyBrakes() {
  // the "if" clause: bicycle must be moving
  if (isMoving){
    // the "then" clause: decrease current speed
    currentSpeed--;
  }
}
```

if-then-else

```
void applyBrakes() {
  if (isMoving) {
    currentSpeed--;
  } else {
    System.err.println("The bicycle has already stopped!");
  }
}
```

switch

```
int month = 8; String monthString;
switch (month) {
  case 1:  monthString = "January";
  break;

  case 2:  monthString = "February";
  break;

  case 3:
  case 4:  monthString = "March or April";
  break;

  default: monthString = "Invalid month";
  break;
}
```

Fall 3 und Fall 4 sind verodert.
Natürlich gibt es noch mehr Monate als Platz auf dieser Folie ☺.

Benutzereingaben

Scanner in Java

Um den newline character zu konsumieren ist es ratsam die gesamte Zeile zu lesen und dann entsprechend zu konvertieren.

```java
// create the scanner
Scanner scanner = new Scanner(System.in);
System.out.println("Gib dein Alter ein");
// Read in the number and directly convert it
int age = Integer.parseInt(scanner.nextLine());
```

Scanner in C#

Hier ist kein Scannerobjekt notwendig.

```
Console.WriteLine("Gib dein Alter ein");
// Read in the number and directly convert it
int age = Int32.Parse(Console.ReadLine());
```

Schleifen

while

```
while (condition) {
  // code block to be executed
}
```

```
int i = 0;
while (i < 3) {
  // code block to be executed
  i++;
}
```

for (Zählschleife)

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

```
for (int i = 0; i < 6; i++) {
  // code block to be executed
}
```

Für verschachtelte Schleifen ist es üblich die Zählvariable *i* für Iterator laut Alphabet zu erhöhen *j*, *k*, *l*, ...

foreach

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

```
for (int number : numbers) {
  // code block to be executed
}
```

Mit dieser Schleife kann man Arrays iterieren, es wird aber nur auf den Wert zugegriffen und nicht auf die Referenz. Ein Element in dem Array kann damit **nicht** verändert. Stichwort: *pass by value* und nicht *pass by reference*.

Methoden und Funktionen

Methoden und Funktionen

Eine Methode ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen wird. Man kann einer Methode Daten, sogenannte Parameter, übergeben und einen Wert, den Rückgabewert zurückgeben.

Methoden und Funktionen

Eine Methode ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen wird. Man kann einer Methode Daten, sogenannte Parameter, übergeben und einen Wert, den Rückgabewert zurückgeben.

- Methoden sind objektgebunden

Methoden und Funktionen

Eine Methode ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen wird. Man kann einer Methode Daten, sogenannte Parameter, übergeben und einen Wert, den Rückgabewert zurückgeben.

- Methoden sind objektgebunden
- Funktionen sind das nicht, in Java verwendet man dafür das Keyword *static*

Methoden und Funktionen

Eine Methode ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen wird. Man kann einer Methode Daten, sogenannte Parameter, übergeben und einen Wert, den Rückgabewert zurückgeben.

- Methoden sind objektgebunden
- Funktionen sind das nicht, in Java verwendet man dafür das Keyword *static*

Warum Methoden?

Methoden und Funktionen

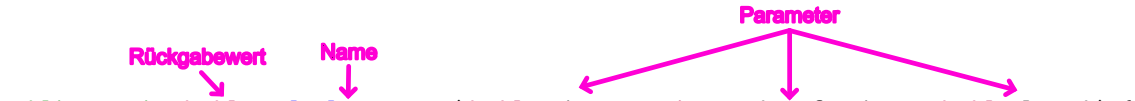
Eine Methode ist ein Codeblock, der nur ausgeführt wird, wenn er aufgerufen wird. Man kann einer Methode Daten, sogenannte Parameter, übergeben und einen Wert, den Rückgabewert zurückgeben.

- Methoden sind objektgebunden
- Funktionen sind das nicht, in Java verwendet man dafür das Keyword *static*

Warum Methoden?

- **Wiederverwendung von Code**. Einmal definieren und beliebig oft wiederverwenden.

Funktion


Parameter
Rückgabewert
Name
public static double calculateAnswer(double wingSpan, int numberOfEngines, double length) {
//do the calculation here
}

Scope

```java
public class Application {
    String name = "Julia";

    public static int calculation(int x, int y) {
        System.out.println(name);
        return x * y;
    }

    public static void printSureName(String surname) {
        System.out.println(name + " " + surname);
    }

    public static void printSureName() {
        System.out.println(name + " Mueller");
    }
}
```

Scope ist der Sichtbarkeitsbereich von Elementen (Variablen, Methoden, Klassen, ...).
Der Scope kann mit Keywords definiert werden.

Methoden mit gleichen Namen, aber unterschiedlichen Parametern werden überladen

```
public class DataArtist {
//      ...
    public void draw(String s) {
//      ...
    }
    public void draw(int i) {
//      ...
    }
    public void draw(double f) {
//      ...
    }
    public void draw(int i, double f) {
//      ...
    }
}
```

Casting

Typecasting

Der Datentyp wird zur Laufzeit verändert. In Java gibt es dafür zwei Möglichkeiten.

- Explizite Typumwandlung (Engl.: widening casting) → kleiner zu größer, keine Information geht verloren.

$$byte > short > char > int > long > float > double$$

- Implizite Typumwandlung (Engl.: narrowing casting) → größer zu kleiner, Informationen können verloren gehen.

$$double > float > long > int > char > short > byte$$

Beispiel Casting

```java
// widening cast
int myInt = 9;
// automatic casting: smaller to larger
double myDouble = myInt;

// narrowing casting
double myDouble = 3.14159d;
// manual casting: larger to smaller
int myInt = (int) myDouble;
// Output 3
System.out.println(myInt);
```

Arrays

Was ist ein Array

Definition

Ein Array ist eine Datenstruktur, die eine Sammlung von Werten des gleichen Typs abspeichert. Jeder Wert kann individuell über einen Index angesprochen werden. Sie sind die Basis für viele andere Datenstrukturen Arrays können für jeden Datentyp erstellt werden, aber nicht in der Größe verändert werden.

Deklarieren und Erzeugen

```java
// Declaration
int[] numbers;
// with the new keyword storage on the heap will be reserved
numbers = new int[5];
// most of the time those steps are combined
int[] numbers = new int[5]; // empty int array with 5 places
```

Die Inhalte eines Arrays

Die Elemente eines Arrays werden bei ihrer Erzeugung mit Standardwerten initialisiert:

Datentyp	Wert
Zahlen	0
boolean	false
char	\u0000
String	null

Befüllen eines Arrays

Um das vorherige Array zu befüllen kann man wie folgt vorgehen:

```
int[]numbers=new int[5];
numbers[0]=1;
numbers[1]=2;
...
```

Alternative kann ein Array auch direkt deklariert und initialisiert werden; wie reguläre Variablen.

```
int[] values = {1,2,3,4,5};
String[] friends = {"Max","Anna","Andi","Yvonne"};
```

Größe und Index

Die Größe eines Arrays wird im *length* Attribut gespeichert. Wenn wir an die numbers von vorhin denken hat `numbers.length` den Wert 5 und die Indizes gehen von $0, \ldots, 4$. Darum steht auch in der Schleife beim iterieren $i <$ *numbers.length*.

For-Schleife

Meist wird für die Iteration von Arrays eine reguläre For-Schleife verwendet. Vorteil: sie ist sehr kompakt und sie greift auf den Wert im Array zu, somit ist er veränderbar.

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

Selbstverständlich kann man auch mit allen anderen Schleifen Arrays durchlaufen.

Foreach-Schleife

Reicht es rein auf die Werte zugreifen zu können gibt es seit Java 1.5 die foreach-Schleife. Sie ist kompakter und erspart das explizite hinschreiben des Indexzugriffs.

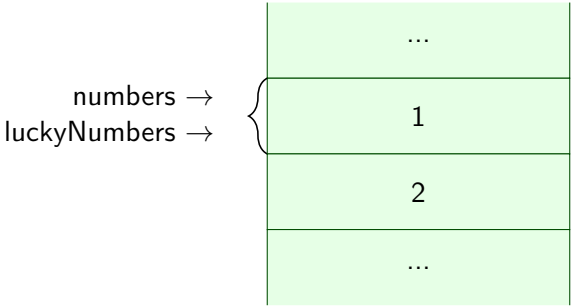
```
for (int number : numbers) {
  // number is only the value
  System.out.println(number);
}
```

Der Wert im Array kann in dieser Schleife **nicht** verändert werden. Darüber hinaus kann die Iterationsrichtung nicht verändert werden.

Kopieren von Arrays

Man kann eine Array-Variable in eine andere kopieren, aber dann verweisen beide Variablen auf das gleiche Array:

```
int[] luckyNumbers = numbers;
luckyNumbers[5] = 12; // numbers[5] is now also 12
```



Tatsächliche Kopie

Möchte man nur die Werte duplizieren, dann muss man die Arrays durchlaufen, oder man verwendet eine Funktion aus der *Arrays* Bibliothek, diese ist bereits in Java standardmäßig vorhanden.

```
// Arrays.copyOf(array to copy, length for new array);
int[] luckyNumbers = Arrays.copyOf(numbers, numbers.length);
```

Aufgabe 1

Schreibe ein Programm, das zuerst die gewünschte Größe des Arrays abfragt, und dann die einzelnen Kommazahlen einliest bis das Array vollständig befüllt ist. Anschließend sollen alle Elemente der Reihe nach auf der Konsole ausgeben werden.
Als Beispiel:

```
Enter the size of the array:
2
Enter number 1:
0,5
Enter number 2:
1,1
The 2 numbers are:
0.5, 1.1
```

2D Arrays

Mehrdimensionale Arrays in Java sind Arrays in Arrays.

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
int x = myNumbers[1][2];
System.out.println(x); // Outputs 7
```

Iteration von 2D Arrays

```
for (int i = 0; i < myNumbers.length; i++) {          // rows
  for (int j = 0; j < myNumbers[i].length; j++) {     // columns
    System.out.println(myNumbers[i][j]);
  }
}
```

Wertübergabe

Es wird der Wert einer Variable übergeben, sie selbst bleibt aber unverändert.

```
public static void main(String[] args) {
  int x = 3;
  incrementAndAdd(x);
  // x has still the value 3
  System.out.println(x);
}

static void incrementAndPrint(int x) {
  x = x + 1;
  System.out.println(x);
}
```

Referenzübergabe

Es wird die Referenz auf eine Variable übergeben, sie selbst wird verändert. In Java werden so Objekte und Arrays übergeben.

```
public static void main(String[] args) {
  int[] x = {1, 2, 3, 4};
  int index = 2;
  incrementAndPrint(x, index);
  // the value in the array has changed
  System.out.println(x[index]);
}

static void incrementAndPrint(int[] x, int index) {
  x[index] = x[index] + 1;
  System.out.println(x[index]);
}
```

Rekursion

Was ist Rekursion?

https://www.explainxkcd.com/wiki/index.php/Category:Recursion

Was ist Rekursion?

https://www.explainxkcd.com/wiki/index.php/Category:Recursion

Eine rekursive Funktion ruft sich immer wieder selbst auf, bis (hoffentlich) eine Abbruchbedingung erfüllt wird. Öfters kommt es vor, dass dadurch elegantere Lösungen möglich sind, als alternativen mit Schleifen. Aber Achtung Rekursion ist im Gegensatz zur Iteration teuer, weil eine Zuweisung eines neuen Stackframes notwendig ist.
Für interessierte eine Erläuterung auf stackoverflow.

Beispiel Summe

Rekursive Summenbildung von n Zahlen

```
private static int num(int n, int sum) {
  if (n == 0) {
    return sum;
  }

  sum += n;
  return num(n - 1, sum);
}
```

Fakultät

Eine Funktion, die einer natürlichen Zahl das Produkt aller natürlichen Zahlen (ohne Null) kleiner und gleich dieser Zahl zuordnet.

$$
\begin{aligned}
0! &= 1 = 1 \\
1! &= 1 = 1 \\
2! &= 1 \cdot 2 = 2 \\
3! &= 1 \cdot 2 \cdot 3 = 6 \\
4! &= 1 \cdot 2 \cdot 3 \cdot 4 = 24 \\
5! &= 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120
\end{aligned}
$$

Fakultät Formel

Die allgemein definierte Funktion lautet:

$$n! = \begin{cases} 1, & n = 0 \\ n \cdot (n-1)!, & n > 0 \end{cases}$$

Rekursive Fakultät in Java

Rekursive Fakultät in Java

```java
private static int factorial(int n) {
  if (n <= 1) {    // base case, termination condition
    return 1;
  }
  return n * factorial(n - 1);
}
```

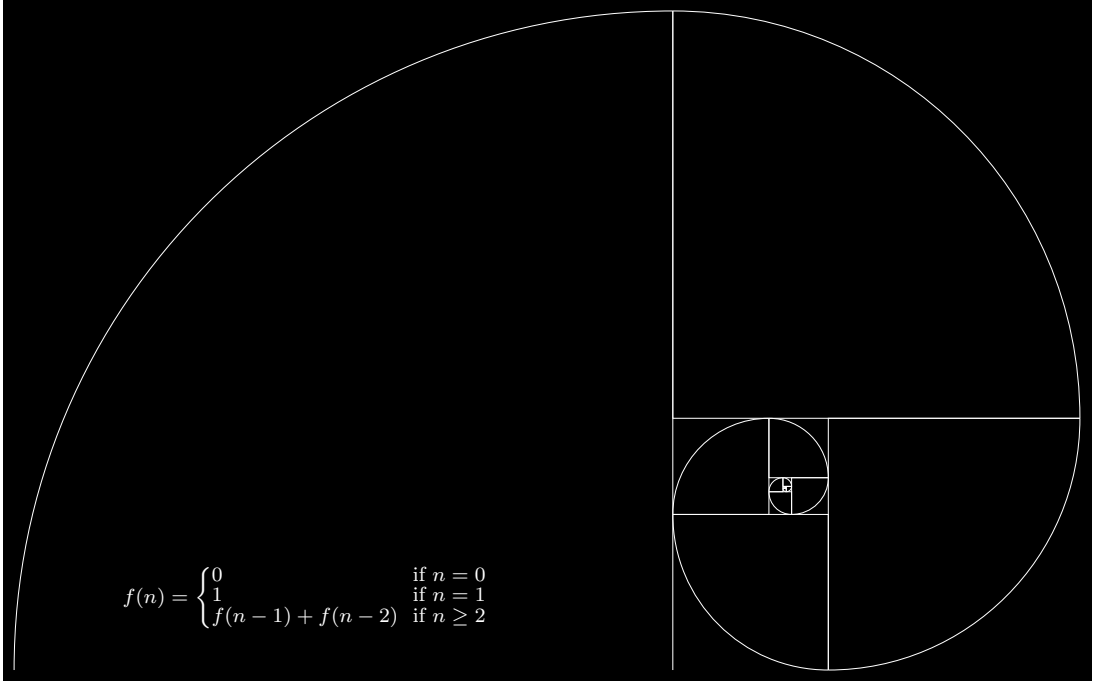
Beispiel 2 Primzahlen

```
public boolean isPrime(long n) {
  return isPrime(n, n - 1);
}

private boolean isPrime(long n, long m) {
  if (m == 1) {
    return true;
  } else if (n % m == 0) {
    return false;
  }
  return isPrime(n, m - 1);
}
```

Aufgabe 2

Berechne die n-te Fibonaccizahl rekursiv. Die Beschreibung der Folge ist auf der nächsten Folie.


$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{if } n \geq 2 \end{cases}$

Lösung in Java

```java
public int fibonacci(int n)  {
  if (n < 2) return n;
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```