

Contents

1	Einleitung	2
2	Imperative Programmierung	2
2.1	Grundlagen der Imperativen Programmierung	2
2.1.1	Von-Neumann-Architektur	2
2.1.2	Programmiersprachen	5
2.1.3	Ausdrücke	5
2.1.4	Imperative Programmierung	5
2.2	Datentypen und Variablen	5
2.2.1	Primitive Datentypen	5
2.2.2	Zusammengesetzte Datentypen	5
2.2.3	Darstellung von Daten?	5
2.2.4	Die fünf Facetten von Variablen	5
2.3	Unterprogramme und Funktionen	5
2.3.1	Fundamentale Unterprogramme	5
2.3.2	Parameter Übergabe Strategien	5
2.3.3	Rekursion	5
2.3.4	Fünf schritte für die Implementierung von einer Funktion	5
2.3.5	Case Study ???	5
3	Algorithmen	5
3.1	Algorithmische Probleme	5
3.2	Sortieren	5
3.3	Laufzeit	5
3.4	Korrektheit	5
4	Funktionelle Programmierung	5
4.1	Funktionelle Programmierung in Scala	5
4.2	Listen und Hochrangige Funktionen	5
4.3	Dateitypen II	5
5	Objekt Orientierte Programmierung	5
5.1	Objekt Orientierte Programmierung	5
5.2	Queues and Stacks	5
5.3	Priority Queue	5
5.4	Dictionaries and Binary Search Trees	5
5.5	Begleitobjekte	5
5.6	Vererbung	6
5.7	späte Bindung/Überschreiben	7

1 Einleitung

2 Imperative Programmierung

2.1 Grundlagen der Imperativen Programmierung

2.1.1 Von-Neumann-Architektur

- Ein Rechner besteht aus vier Komponenten/Bestandteilen:
 - CPU (Central Processing Unit) - **Steuerwerk**
 - Memory (Speicher/RAM)
 - Ein/Ausgabe (I/O)
 - BUS - verbindet alle Komponenten

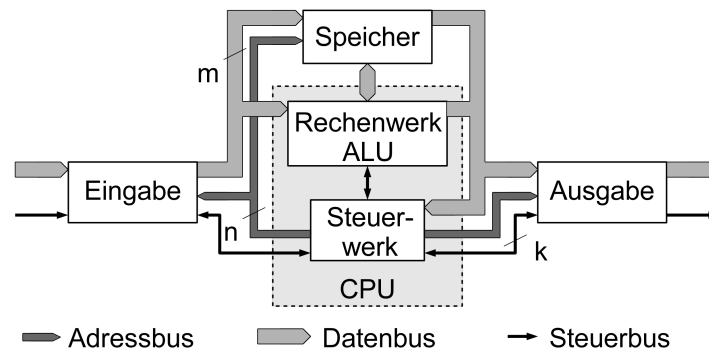


Figure 1: Bildunterschrift

- CPU (Central Processing Unit)
 - CU (Control Unit)
 - * Koordiniert die Ausführung von Befehlen
 - ALU (Arithmetic Logical Unit) - **Rechenwerk**
 - * Das sind die Schaltkreise, um die wirklichen Berechnungen durchzuführen
 - Clock (Taktgeber)
 - * Sendet regelmäßig Impulse aus, um einen Takt vorzugeben
 - * Ein Quarz, der mit Strom (CEmos Batterie) in Schwingung versetzt wird
 - PC (Program counter - Programmzähler)
 - * Adressiert diejenige Zelle im Hauptspeicher (Memory), bei der die nächste Anweisung beginnt

- Speicher (RAM)
 - Speichert die Daten und Programme
 - RAM (Random Access Memory)
 - * Speicher besteht aus Zellen
 - * Zellen sind durchnummeriert, z.B. von 0 bis n-1
 - * Alle Zellen sind gleich groß, zwischen 0 und x (Hardware abhängig)
 - * Jede Zelle speichert eine Zahl, repräsentiert in Bits
 - * 8 Bits/16 Bits/32 Bits/64 Bits - sind mögliche Variationen, je nachdem wie das System aufgebaut ist
- **Funktionalität** des Speichers: Lesen und Schreiben.
 - Lese die Zelle 100 oder schreibe in die Zelle 100
 - schreibe zahl z in Speicherzelle h
 - wird in beliebiger Reihenfolge unterstützt
- ROM (Read only Memory)
- Ein/Ausgabe (I/O)
 - Sind Physikalische Geräte, die erlauben mit dem Computer von außerhalb zu Kommunizieren wie: Tastatur, Maus, Drucker, Kamera, Bildschirm, Netzwerkinterface, USB-Anschluss, ...usw.
- BUS
 - sind Kabel/Leitungen/Drähte, die alle Komponenten miteinander verbinden

Funktionsweise Solange der Rechner eingeschaltet ist, führt er die folgende Schritte immer wieder aus.

- FETCH - CU weiß den Memory über den Bus an, den Inhalt der Speicherzelle mit Adresse PC zu liefern
- DECODE - (Dekodiere/Interpretiere) CU schaut nach um welche Anweisung es sich handelt und holt gegebenenfalls weitere Bestandteile der Anweisung aus dem Memory
- EXECUTE - CU sorgt dafür, das Anweisung ausgeführt wird, indem sie die andere Komponente Koordiniert
- Aktualisiere PC
- REPEAT - Wiederholt den Durchlauf

- 2.1.2 Programmiersprachen
- 2.1.3 Ausdrücke
- 2.1.4 Imperative Programmierung
- 2.2 Datentypen und Variablen
 - 2.2.1 Primitive Datentypen
 - 2.2.2 Zusammengesetzte Datentypen
 - 2.2.3 Darstellung von Daten?
 - 2.2.4 Die fünf Facetten von Variablen
- 2.3 Unterprogramme und Funktionen
 - 2.3.1 Fundamentale Unterprogramme
 - 2.3.2 Parameter Übergabe Strategien
 - 2.3.3 Rekursion
 - 2.3.4 Fünf schritte für die Implementierung von einer Funktion
 - 2.3.5 Case Study ???

3 Algorithmen

- 3.1 Algorithmische Probleme
- 3.2 Sortieren
- 3.3 Laufzeit
- 3.4 Korrektheit

4 Funktionelle Programmierung

- 4.1 Funktionelle Programmierung in Scala
- 4.2 Listen und Hochrangige Funktionen
- 4.3 Datentypen II

5 Objekt Orientierte Programmierung

- 5.1 Objekt Orientierte Programmierung
- 5.2 Queues and Stacks
- 5.3 Priority Queue
- 5.4 Dictionaries and Binary Search Trees
- 5.5 Begleitobjekte

5

- Teile globale Variablen zwischen Exemplaren einer Klasse.

```

1 object Studi:
2     private var matrikel_zaeher: Int = 0
3
4 class Studi(var name: String, var fach: String):
5     var matrikel_nr = Studi.matrikel_zaeher
6     Studi.matrikel_zaeher = Studi.matrikel_zaeher + 1
7     def getMat(): Int = matrikel_nr
8
9 @main
10 def test(): Unit =
11     var s1: Studi = new Studi("Max", "Info")
12     var s2: Studi = new Studi("Katharina", "Binfo")
13     var s3: Studi = new Studi("Günther", "Winfo")
14     println(s1.getMat())
15     println(s2.getMat())
16     println(s3.getMat())

```

5.6 Vererbung

- Soll Ziele der Wiederverwendbarkeit & Erweiterbarkeit unterstützen.
- Können Beziehungen zwischen Klassen herstellen
- Können neue Klassen zu **Unterklassen** von bestehenden Klassen machen!
- **Unterklasse** übernimmt damit automatisch alle Methoden und Attribute der **Oberklasse**

```

1 class Person(var name: String, private var age: Int):
2     def mature(): Unit =
3         age = age + 1
4     def getAge(): Int = age
5     def work(): Unit =
6         println("werkel werkel")

```

- Inklusionspolymorphie:
 - Objekte der Unterklasse sind typekompatibel mit der Oberklasse
- Wir unterscheiden:
 - **Statischer** Typ einer Variable:
 - * Typ aus Variablendeklaration (ändert sich nicht - statisch)
 - * legt fest auf welche Attribute und Methoden zugegriffen werden kann
 - **Dynamischen** Typ einer Variable:
 - * Typ des Objekts, auf das die Variable aktuell verweist

Wir können den statischen Typ einer Variable mit einem Cast(Typumwandlung - nur wenn der dynamische Typ es erlaubt) ändern(asInstanceOf). Wenn er es nicht erlaubt, sonst bekommen wir eine ClassCastException als Error von Scala.

5.7 späte Bindung/Überschreiben

```
1 class Studi(name: String, private var age: Int, var fach: String) extends Person(name, age):
2     def getZurMensa(): Unit = println("Mjam, schlürf")
3     override def work(): Unit =
4         println("studier studier")
5
6 var s1: Studi = new Studi("Max", 12, "Data Science")
7 var s1: Studi = Studi@6870f51a
8
9 scala> s1.work()
10 studier studier
11
12 var p1: Person = s1
13 var p1: Person = Studi@6870f52a
14
15 scala> p1.work()
16 studier studier
17
18 # zweite person
19 scala> var p2: Person = new Person("Moritz", 10)
20 var p2: Person = Person@1107891a
21
22 scala> p1 = p2
23 p1: Person = Person@234234a
24
25 scala> p1.work()
26 werkle werkle
27
28 # beispiel mit Array von Personen wird gezeigt...das schreibe ich jetzt nicht ab
29 scala> var ps: Array[Person] = Array(p2, s1)
30
31 for p <- ps do
32     p.work()
33
34 #ausgabe von scala
35 werkel werkel
36 studier studier
```

Beispiel-Titel

Dies ist eine farbige Textbox mit einem Titel, einem blauen Hintergrund und einem Rahmen.