Contents

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

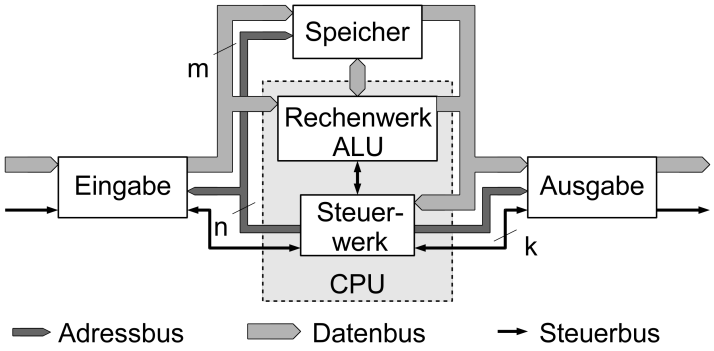

Figure 1: Bildunterschrift

- CPU (Central Processing Unit)
  - CU (Control Unit)
    - Koordiniert die Ausführung von Befehle
  - ALU (Arithmetic Logical Unit) - **Rechenwerk**
    - Das sind die Schaltkreise um die wirklichen Berechnung durchzuführen
  - Clock (Taktgeber)
    - Sendet regelmäßig impulse aus, um einen Takt vorzugeben
    - Ein Quwarz der mit Strom (CEmos Batterie) in Schwingung versetzt wird
  - PC (Program counter - Programmzähler)
    - Adressiert diejenige Zelle im Hauptspeicher (Memory) bei dem 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, jenachdem 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: Tastartur, 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 weißt 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

- Teile globale Variablen zwischen Exemplaren einer Klasse.

```
object Studi:
        private var mattrikel_zaehler: Int = 0

class Studi(var name: String, var fach: String):
        var mattrikel_nr = Studi.mattrikel_zaehler
        Studi.mattrikel_zaehler = Studi.mattrikel_zaehler + 1
        def getMat(): Int = mattrikel_nr

@main
def test(): Unit =
        var s1: Studi = new Studi("Max", "Info")
        var s2: Studi = new Studi("Katharina", "Binfo")
        var s3: Studi = new Studi("Günther", "Winfo")
        println(s1.getMat())
        println(s2.getMat())
        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**

```
class Person(var name: String, private var age: Int):
        def mature(): Unit =
                age = age + 1
        def getAge(): Int = age
        def work(): Unit =
                println("werkel werkel")
```

- Inklusionspolymorphie:
  - Objekte der Unterklasse sind typekompotibel 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 Class:CastException als Error von Scala.

5.7 späte Bindung/Überschreiben

```
class Studi(name: String, private var age: Int, var fach: String) extends Person(name, age):
        def getZurMensa(): Unit = println("Mjam, schlürf")
        override def work(): Unit =
                println("studier studier")

var s1: Studi = new Studi("Max", 12, "Data Science")
var s1: Studi = Studi@6870f51a

scala> s1.work()
studier studier

var p1: Person = s1
var p1: Person = Studi@6870f52a

scala> p1.work()
studier studier

# zweite person
scala> var p2: Person = new Person("Moritz", 10)
var p2: Person = Person@1107891a

scala> p1 = p2
p1: Person = Person@234234a

scala> p1.work()
werkle werkle

# beispiel mit Array von Personen wird gezeigt...das schreibe ich jetzt nicht ab
scala> var ps: Array[Person] = Array(p2, s1)

for p <- ps do
        p.work()

#ausgabe von scala
werkel werkel
studier studier
```

Beispiel-Titel

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