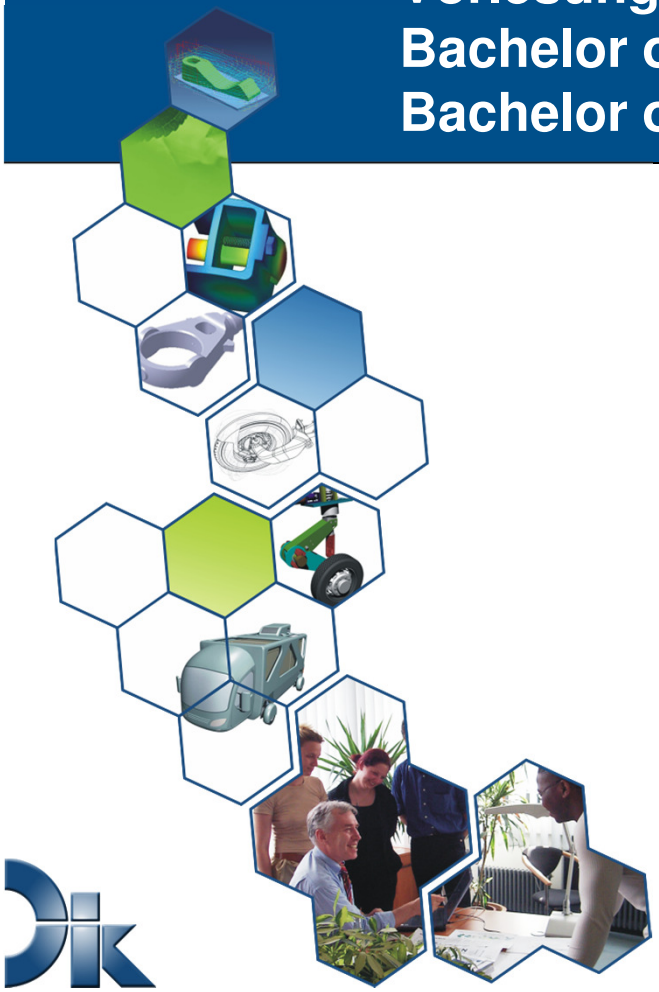
Grundlagen der elektronischen Datenverarbeitung

Vorlesung für Bachelor of Science MPE, Mechanik, Bachelor of Education

TECHNISCHE UNIVERSITÄT DARMSTADT



Die Vorlesung wird zusätzlich in den Raum S3|11-006 (1 Stockwerk tiefer) übertragen.

Einführung in den Maschinenbau (EMB)

TECHNISCHE UNIVERSITÄT DARMSTADT

Am 06.12.2010 wird die GeDV-Vorlesung aufgrund der EMB-Veranstaltungen ausfallen.

Am 10.12.2010 wird die PST-Vorlesung, die PST-Gruppenübungen und die PST-Tutorensprechstunde aus dem gleichen Grund ausfallen.

Gliederung 6. Vorlesungsstunde GeDV

TECHNISCHE UNIVERSITÄT DARMSTADT

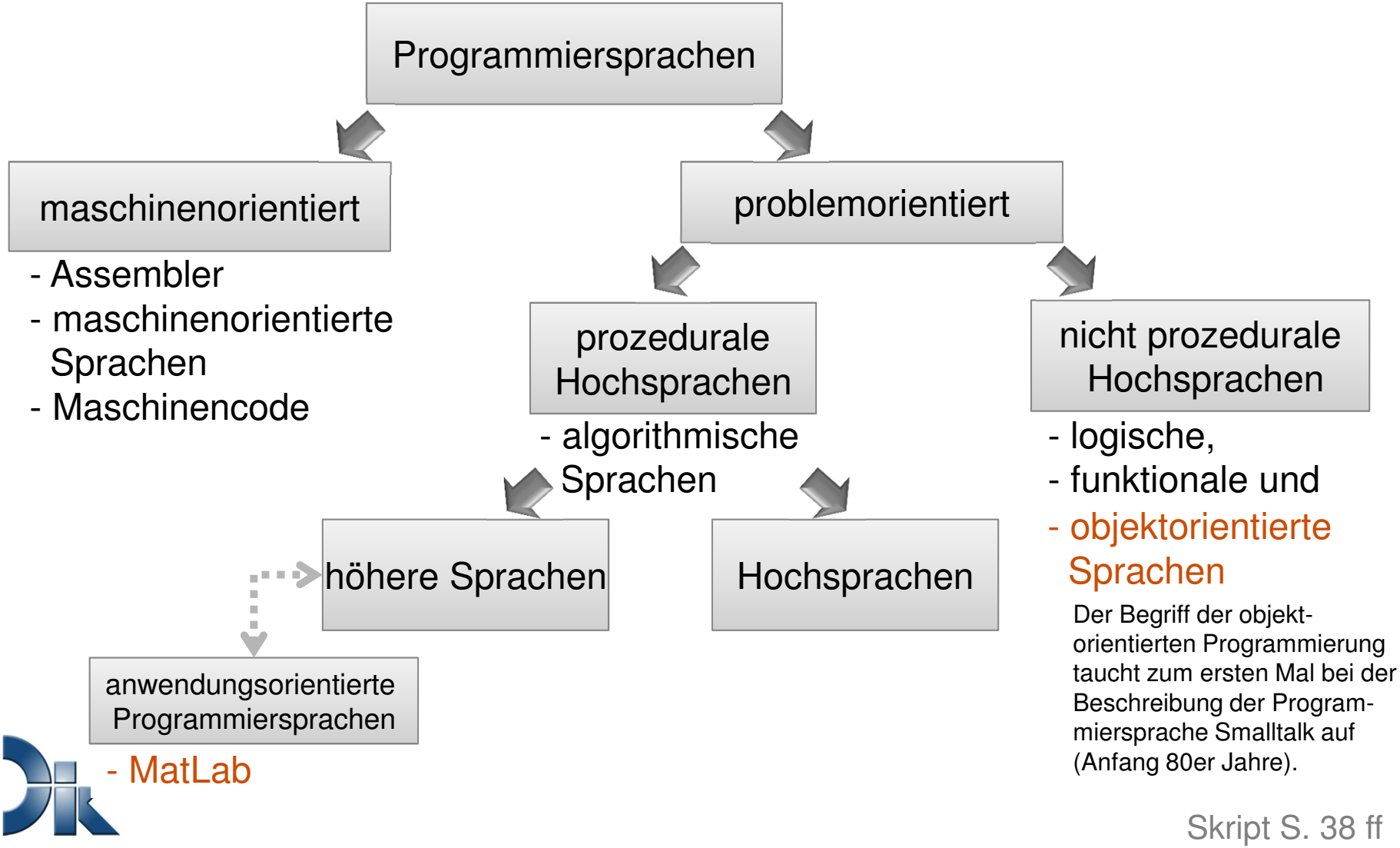
Programmiersprachen und Techniken

- Objektorientierung am Beispiel von Java
- Geschichte der Programmiersprachen

Datenstrukturen und Algorithmen

- Einfache Datentypen
- Strukturierte Datentypen
- Abstrakte Datentypen
- Graphen

Arten von Programmiersprachen



Der Begriff der objekt-orientierten Programmierung taucht zum ersten Mal bei der Beschreibung der Program-miersprache Smalltalk auf (Anfang 80er Jahre).

Skript S. 38 ff

Was ist Java?

TECHNISCHE UNIVERSITÄT DARMSTADT

- Java ist eine streng objektorientierte und streng typisierte Sprache (d.h. Variablentypen werden z.B. streng überprüft)
- Java ist entstanden im Jahre 1992 unter dem Name „Café“ mit dem Ziel, eine maschinenunabhängige Sprache für das Internet zu entwickeln
- Java ist unabhängig von der Rechnerplattform, sofern eine virtuelle Maschine (VM) für das Interpretieren von Java-Byte-Code existiert, der von einem Byte-Code-Compiler erzeugt wird,
- Java besitzt eine mit C++ vergleichbare Syntax, verzichtet jedoch auf einige Sprachkonstrukte von C++
- Java bietet umfangreiche Bibliotheken (Bsp.: java.io, java.applet, java.awt, ...)
- Gruppen von Java Programmen:
  - *applications*: in Java programmierte Anwendungen (Alternative zu anderen objektorientierten Sprachen)
  - *applets*: Java Programme die über einen „javafähigen“ Browser ausgeführt werden können; auch über das WWW (Vorteil zu anderen objektorientierten Sprachen)
  - zusätzlich noch: servlets, beans - hier nicht näher erwähnt

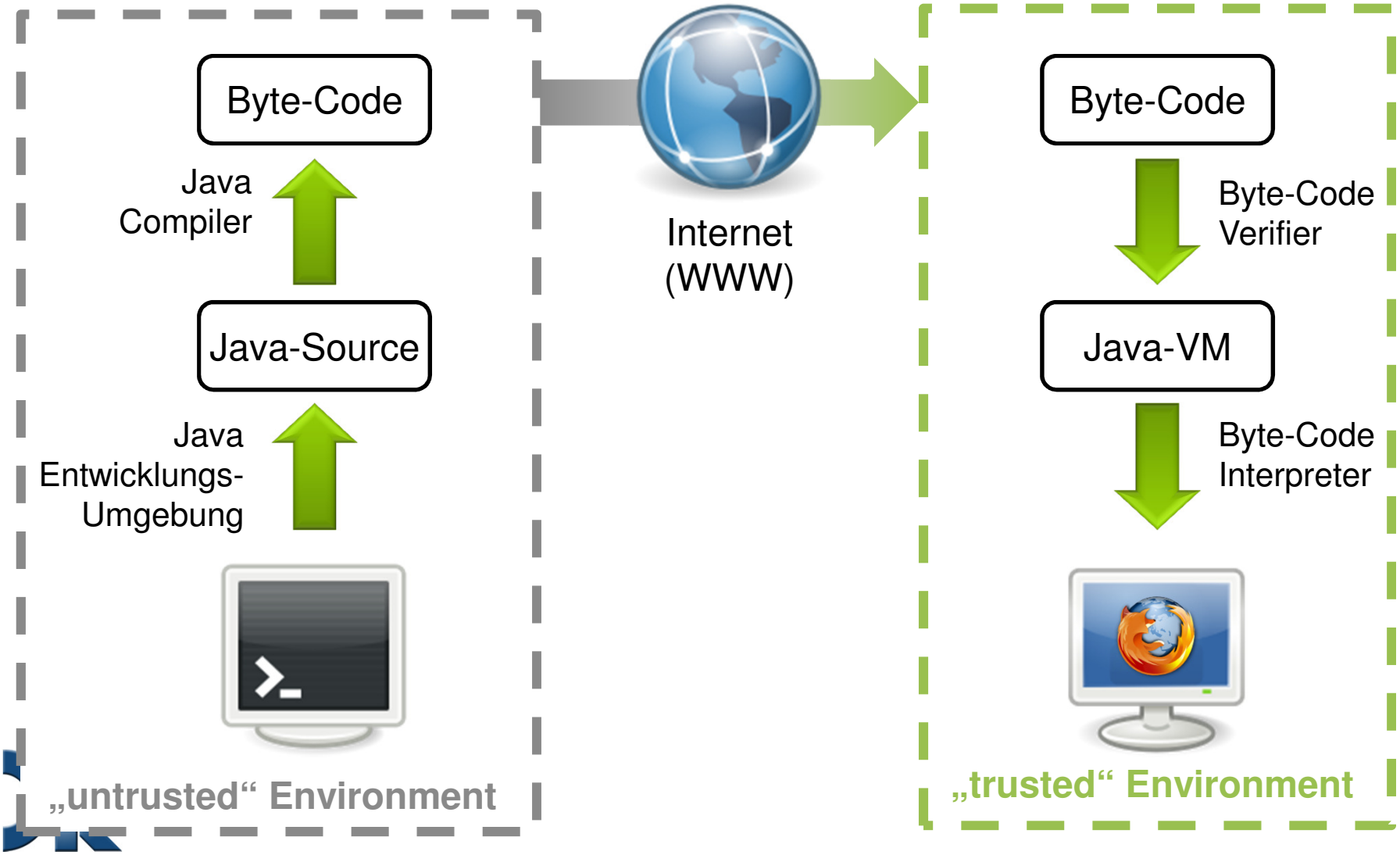
Skript S. 59 ff

Interpreter & Compiler

Skript S. 68 ff

Java Environment für Applets

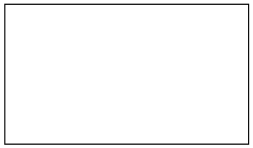
TECHNISCHE UNIVERSITÄT DARMSTADT



Beispiel: Programmbausteine in JAVA - Anweisungen

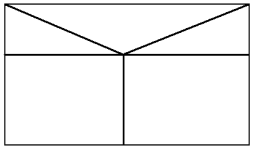
TECHNISCHE UNIVERSITÄT DARMSTADT

Sequenz



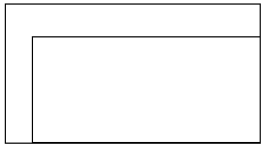
```
{
     Anweisung_1;
     ...
     Anweisung_n;
}
```

Alternativ



```
if ( Bedingung ){
     Anweisungen_1;
}
else{
     Anweisungen_2;
}
```

Wiederholung



```
for(Bedingung)
{
     Anweisung;
}
```

```
do
{
     Anweisung;
}
while(Bedingung)
```

```
while(Bedingung)
{
     Anweisung;
}
```

Ein einfaches Java-Applet „Hello World“

TECHNISCHE UNIVERSITÄT DARMSTADT

Datei `HelloWorld.java`

```
1  import java.applet.Applet
   // Das Programm verwendet die Klasse java.applet (notwendig),
2  import java.awt.*
   // sowie die Klasse Graphics des Java ´abstract  windowing toolkit (awt) zur graphischen Ausgabe.
3  public class HelloWorld extends Applet {
   // Die Klasse HelloWorld wird erzeugt. Sie ist von der Klasse  Applet abgeleitet, erweitert sie also.
4      public void init() {
       // Die von Applet geerbte Methode init() wird neu definiert
5          resize(200,25);


6      }


7      public void paint(Graphics g) {
       // Auch die geerbte Methode paint() wird neu definiert
8          g.drawString(„Hell World!“,50,25);
            // Die Methode drawString der übergebenen Klasse ‚g‘ erhält 3 Nachrichtenobjekte
9      }
10 }
```

Ein einfaches Java-Applet „Hello World“

TECHNISCHE UNIVERSITÄT DARMSTADT

Datei `HelloWorld.java`

```
1  import java.applet.Applet
   // Das Programm verwendet die Klasse java.applet (notwendig).
2  import java.awt.*
   // sowie die Klasse Graphics des Java "abstract windowing toolkit (awt)" zur graphischen Ausgabe.
3  public class HelloWorld extends Applet {
   // Die Klasse HelloWorld wird erzeugt. Sie ist von der Klasse Applet abgeleitet, erweitert sie also.
4      public void init() {
       // Die von Applet geerbte Methode init() wird neu definiert.
5          resize(200,25);

6      }

7      public void paint(Graphics g) {
       // Auch die geerbte Methode paint() wird neu definiert.
8          g.drawString("Hell World!",50,25);
           // Die Methode drawString der übergebenen Klasse g erhält 3 Nachrichtenobjekte.
9      }
10 }
```

Datei `HelloWorld.html`

```
<HTML>
    <HEAD>
        <TITLE>Ein Hello World Programm</TITLE>
    </HEAD>
    <BODY>
        <H1> Programm </H1>
            Hier ist die Ausgabe des Programms:
        <APPLET CODE="HelloWorld.class">
    </BODY>
</HTML>
```

Merkmale der objektorientierten Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Merkmale	Programmierung in Java
Objekt	`Kreis kreis1 = new Kreis();`

Objektattribute (Strukturen)	`public class Kreis {`
	`  private Koordinaten mittelpunkt;`
	`  private int radius;`
	`}`

Objektmethoden (Verhalten)	`Kreis kreis1 = new Kreis();`
	`kreis1.setMittelpunkt(new Koordinaten(0,0));`
	`kreis1.setRadius(5);`
	`kreis1.setFarbe(Color.blue);`
	`kreis1.setFarbe(Color.green);`

Skript S. 49 ff

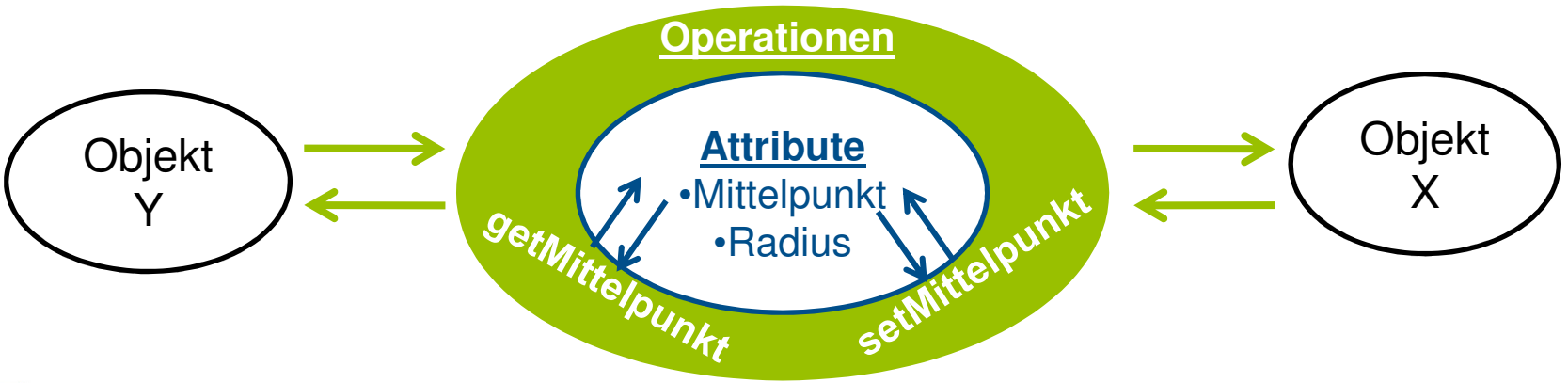
Merkmale der objektorientierten Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

Merkmale	Programmierung in Java
Kapselung	

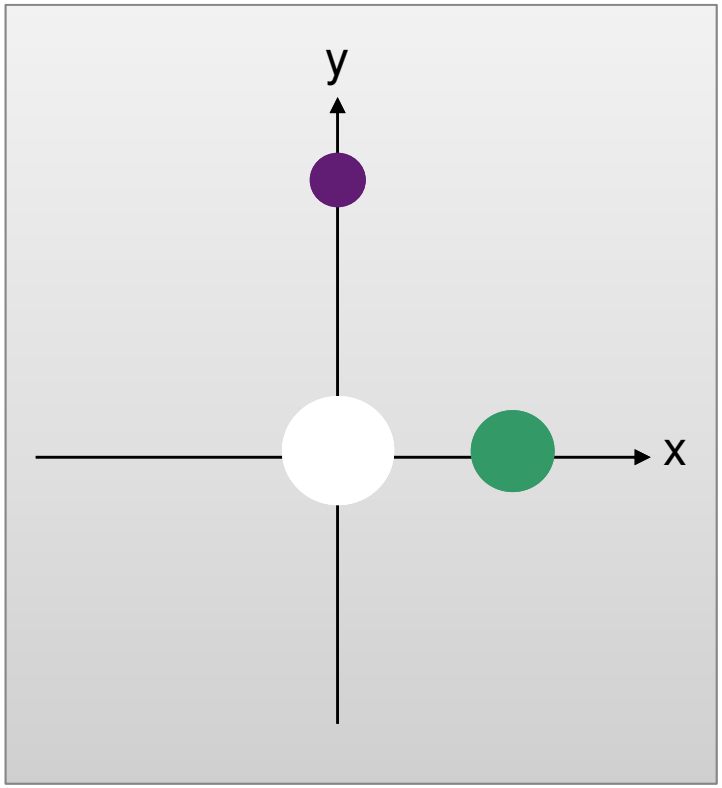
```
public class Kreis {
     private Koordinaten mittelpunkt=new Koordinaten(0,0);
     private int radius = 0;

     public Koordinaten getMittelpunkt() {
          return this.mittelpunkt;
     }
     public void setMittelpunkt(Koordinaten p_koordinaten){
          this.mittelpunkt = p_koordinaten;
     }
}
```



Skript S. 49 ff

Anwendungsbeispiel: Kreis




```
Kreis kreis1=new Kreis(new Koordinaten(0,0));
// Instanziierung eines Kreises „kreis1“ im Ursprung
```

```
kreis1.setMittelpunkt(new Koordinaten(0,5));
// Verschieben des Kreises nach x=0; y=5
```

```
kreis1.setRadius(1);
// Setzen des Radius auf 1
```

```
kreis1.setFarbe(Color.blue);
// Setzen der Farbe auf Blau
```

```
Kreis kreis2=new Kreis(new Koordinaten(0,0));
// Instanziierung eines Kreises „kreis2“ im Ursprung
```

```
kreis2.setMittelpunkt(new Koordinaten(4,0));
// Verschieben des Kreises nach x=4; y=0
```

```
kreis2.setRadius(2);
// Setzen des Radius auf 2
```

```
kreis2.setFarbe(Color.green);
// Setzen der Farbe auf Grün
```

Merkmale der objektorientierten Programmierung

TECHNISCHE UNIVERSITÄT DARMSTADT

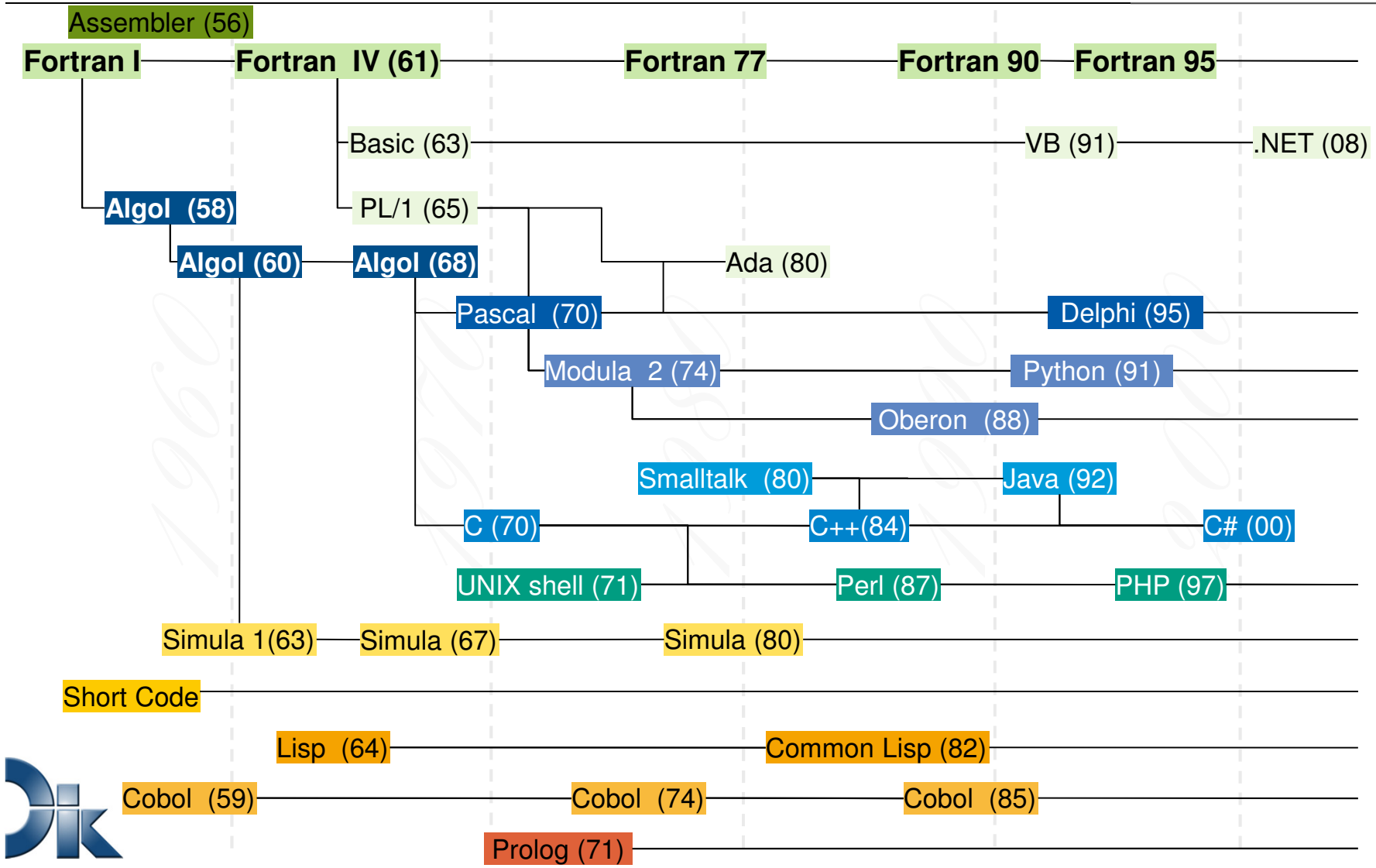
Merkmale	Programmierung in Java
Klassenkonzept	`public class Kreis { }`
Instanzen	`Kreis kreis1 = new Kreis();` `Kreis kreis2 = new Kreis();`
Generalisierung / Spezialisierung	`public class GefuellterKreis extends Kreis { }`

Polymorphie	`public class Kreis { public abstract void druckeInfo(); }`
	`public class GefuellterKreis extends Kreis { public void druckeInfo() { System.out.println(„Ein gefuellter Kreis“); } }`
	`public class LeererKreis extends Kreis { public void druckeInfo() { System.out.println(„Ein leerer Kreis“); } }`
	`public class Test { public static void main(String[] args) { Kreis kreis1 = new GefuellterKreis(); kreis1.druckInfo(); // Ein gefuellter Kreis kreis1 = new LeererKreis(); kreis1.druckInfo(); // Ein leerer Kreis } }`

Demo: MesswertPruefer in Java

Chronologische Entwicklung der Programmiersprachen

TECHNISCHE UNIVERSITÄT DARMSTADT



Fortran-Programmbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Fortran-Programm	Kommentar
`C  PROGRAMM: SUMME`	Kommentarzeile im Programm
`C`	Leerzeile
`  INTEGER S`	Vereinbarung: Variable S ganzzahlig
`  S = 0`	Anweisung: Wert-Zuweisung
`  DO 15 I=1,10` `  S=S+I*I`	DO-Anweisung: Schleife I (1 bis 10) über Anweisung bis Markierung 15
`15  CONTINUE`	Leeranweisung als Stilmittel
`  WRITE  (5,20) S`	Druck-Anweisung: Ausgabegerät Nr. 5
`20  FORMAT  (1H1,10X,I4)`	Format-Anweisung Nr. 20
`  STOP`	Programmausführung beenden
`  END`	Ende einer Einheit für den Compiler

Skript S. 42

Cobol-Programmbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Cobol-Programm

```
000010   IDENTIFICATION DIVISION.
000020   PROGRAMM-ID.SUMME.
000030*
000040   ENVIRONMENT DIVISION.
000050   CONFIGURATION SECTION.
000060   SOURCE-COMPUTER. SIEMENS-7880.
000070   OBJECT-COMPUTER. SIEMENS-7880.
000080*
000090   DATA DIVISION.
000100   WORKING-STORAGE SECTION.
000110   77   I PIC 99.
000120   77   S PIC 9(4) VALUE 0.
000130*
000140   PROCEDURE DIVISION.
000150   P1.
000160   PERFORM P2 VARYING I FROM
        1 BY  1 UNTIL I>10.

000170   DISPLAY S UPON PRINTER.
000180   STOP RUN.
000190   P2.
000200   COMPUTE S=S+I*I.
```

Kommentar

Teil 1: Programmkopf
Programmname: SUMME

Teil 2: Hardwarespezifische Angaben
Angaben zur Rechnerumgebung
Rechnertyp für Quellcode
Rechnertyp für Programmcode

Teil 3: Speicherbereichszuteilung
Variablenvereinbarungen
Variable I mit 2 Stellen
Variable S mit 4 Stellen, mit 0 initialisiert

Teil 4: Prozeduren
Prozedur P1
Erhöhe I von 1 bis 10 und führe bei jedem Schritt P2 aus

Ausgabe von S auf dem Drucker
Programmende
Prozedur P2
Berechne S=S+I*I

Skript S. 43

Basic-Programmbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Basic-Programm		Kommentar
10	`REM PROGRAMM: SUMME QUADRATE`	Dokumentierender Text
20	`REM`	Leerzeile
30	`S=0`	Summe-Anfangswert 0 setzen
40	`FOR I = 1 TO 10 STEP 1`	Schleifenbildung mit I = 1,2,... bis 10 um in S die I-Quadrate zu kumulieren
50	`    S = S + I * I`	
60	`    NEXT I`	
70	`PRINT "SUMME  IST "; S`	String und Ergebnis S drucken
80	`END`	Programmende

```
OK
RUN
SUMME  IST  358
OK
```

```
Interpreter bereit
Startet geladenes Programm
Ausdruck vom Ergebnis
Interpreter bereit
```

Skript S. 44

Pascal-Programmbeispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

Pascal-Programm	Kommentar
`Program SUM (INPUT,OUTPUT);`	Programmkopf
`VAR  I,S: INTEGER;`	Vereinbarung: Variablen I,S ganzzahlig
`BEGIN`	Programmblock-Beginn
`  S:=0`	Anweisung: Wert-Zuweisung
`  FOR I:= 1 TO 10 DO`	FOR-Schleife mit Zählvariable I (1 bis 10)
`    S:= S+SQR(I);`	Anweisung: Wert.Zuweisung
`    WRITELN (‘Ergebnis=‘,S);`	Ausgabe (Text, Variable)
`END`	Programmblock-Ende

Skript S. 45

Kapitel 4

Datenstrukturen und Algorithmen

Datentypen

Skript S. 91 ff

Einfache Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

integer `125`	Ganze Zahlen, der Bereich hängt von der Programmiersprache ab
real `-225,83`	Reelle Zahlen als Gleitkommadarstellung; auch hier ist der Bereich sprachenabhängig
boolean `true`	Zweiwertiger logischer Typ mit den Werten false und true
char `"z"`	Ein einzelnes Zeichen, meist ASCII-codiert; z.B. Buchstaben, Ziffern oder Sonderzeichen.

Skript S. 34

Einfache Datentypen in Java

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gleiche Typen unterscheiden sich nach Wertebereich und benötigten Speicher

Datentypen		Wertebereich	Speicher
Integer	byte	$-128=-2^7$ bis $127=2^{7-1}$	1 Byte
	short	$-32.768=-2^{15}$ bis $32.767=2^{15-1}$	2 Bytes
	int	-2^{31} bis $2^{31}-1$	4 Bytes
	long	-2^{63} bis $2^{63}-1$	8 Bytes
Real	float	(+/-) $3,4*10^{38}$	4 Bytes
	double	(+/-) $1,7*10^{38}$	8 Bytes
Boolean	boolean	true, false	1 Bit
Character	char	\u0000 bis \uFFFF (Unicode Zeichen)	2 Bytes

Skript S. 34

Einfache Datentypen in MATLAB

TECHNISCHE UNIVERSITÄT DARMSTADT

- Gleiche Typen unterscheiden sich nach Wertebereich und benötigten Speicher

Datentypen		Wertebereich	Speicher
Integer	int8	$-128=-2^7$ bis $127=2^{7-1}$	1 Byte
	int16	$-32.768=-2^{15}$ bis $32.767=2^{15-1}$	2 Bytes
	int32	-2^{31} bis $2^{31}-1$	4 Bytes
	int64	-2^{63} bis $2^{63}-1$	8 Bytes
Real	single	(+/-) $3,4*10^{38}$	4 Bytes
	double	(+/-) $1,7*10^{38}$	8 Bytes
Boolean	logical	true, false	1 Bit
Character	char	\u0000 bis \uFFFF (Unicode Zeichen)	2 Bytes

Skript S. 34

Strukturierte Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

array (Feld)

$$\begin{pmatrix} 1 & 1 & 2 \\ 2 & -4 & 7 \\ 3 & 12 & -3 \end{pmatrix}$$

Eine ein- oder mehrdimensionale Matrix deren Elemente alle denselben skalaren Datentyp haben. Angesprochen werden sie durch Indizierung der Variable.

Beispiel:

- um in der 5. Zeile und 3. Spalte einer Matrix den Wert 8 zuzuweisen, schreibt man „Matrix[5,3] = 8“,
- um einem anderen Feld mit dem Namen Matrixkopie alle Werte von Matrix zuzuordnen kann man einfach „Matrixkopie = Matrix“ schreiben.

record (Verbund)

```
Name
Vorname
Jahrgang
Ort
...
```

Ein hierarchischer Datentyp, dessen Bestandteile aus verschiedenen Datentypen aufgebaut sein können. Der Zugriff erfolgt durch Angabe des Weges durch die Hierarchie zum gewünschten Element.

Beispiel:

In der Programmiersprache PASCAL werden die Hierarchieebenen dabei durch Punkte getrennt, so dass die Adresse eines bestimmten Angestellten in der Personaldatei mit „Abteilung.Name.Adresse“ angesprochen werden würde.

file (Datei)

Eine Reihe von Daten gleichen Typs, wobei es sich dabei auch um Verbunde oder Felder handeln darf.

Zusätzlich zur Deklaration muss sie noch geöffnet und nach der Bearbeitung geschlossen werden. Programmiersprachen bieten dazu spezielle Befehle an, wie Öffnen, Schließen, Nächstes Element lesen und Zurücksetzen auf das erste Element.

Skript S. 34

Strukturierte Datentypen in Java

TECHNISCHE UNIVERSITÄT DARMSTADT

Beispiele:

array (Feld)	`char 'varname' [];`
	eindimensionales Charakterfeld (Darstellung von Text):
	`int 'varname' [][];`
	zweidimensionales Integer Feld (Darstellung 2-D Matrix)

record (Verbund)	methodenlose Klassendefinition mit Klassenhierarchie
file (Datei)	Handling über Klassendefinitionen im Paket java.io Bsp. Java.io.File -> Konstruktoren für Datei und Verzeichnisse

Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

- In nicht-objektorientierten Programmiersprachen können neue Datentypen auf der Basis der Elementardatentypen und der strukturierten Datentypen in der Typ-Deklaration festgelegt werden.

- Der Wertebereich eines Elementardatentyps ist durch den Typ implizit vorgegeben.

- Der Wertebereich eines strukturierten Datentyps ist das kartesisches Produkt der Wertebereiche seiner Komponententypen.

Skript S. 105 ff

Abstrakte Datentypen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Datenstrukturen komplexer Datenmodelle können oberhalb der Abstraktionsebene der in Programmiersprachen vorkommenden Datentypen liegen.
- Solche Datenstrukturen können in Klassen ähnlicher Strukturen zusammengefasst werden, die durch eine generische Datenstruktur beschrieben werden können.
- Eine generische Datenstruktur kann in einer Programmiersprache durch einen abstrakten Datentyp (ADT) realisiert werden.
- Sicht des ADT-Benutzers:
  Der abstrakte Datentyp besitzt eine Struktur, einen Wertebereich und eine Menge von Operationen.
- Sicht des ADT-Programmierers:
  Der abstrakte Datentyp wird auf der Basis schon vorhandener Datentypen und deren Operationen in der Programmiersprache beschrieben.

Skript S. 93 ff

Abstrakter Datentyp in Java

TECHNISCHE UNIVERSITÄT DARMSTADT

- Der Begriff abstrakter Datentyp entstand auch als Prototyp der Begriffswelt der objektorientierten Programmierung:
- Ein abstrakter Datentyp vereint die Repräsentation der Werte eines Typs mit den Operationen auf diesen Werten und verkapselt die Implementierung.
- Ein Programm, das einen abstrakten Datentyp benutzen möchte, muss nur die Definition diesen Datentyps kennen - nicht aber seine Implementierung.

→Man kann daher den abstrakter Datentyp in natürlicher Weise als Klassen einer OOP-Sprache wie z. B. Java auffassen.

Skript S. 93 ff

Abstrakter Datentyp „Vektor“

Skript S. 107

Abstrakter Datentyp: Graph

TECHNISCHE UNIVERSITÄT DARMSTADT

Ein **Graph G** ist ein Paar von zwei Mengen:

- einer Menge N von Knoten und
- einer Menge V von Kanten: G=(N,V)
- Eine Kante a aus V verbindet stets zwei Knoten A und B, beide aus N, miteinander: a={A,B}.

Verschiedene **Graphentypen** sind:

- Ungerichteter Graph:
  die Kanten besitzen keine Vorzugsrichtung, d.h. a ={A,B}={B,A}.
- Gerichteter Graph:
  die Kanten besitzen eine Vorzugsrichtung,
  d.h. a={A,B}, aber a´={B,A} mit einer neuen Kante a´.
- Gewichteter Graph:
  Die Kanten eines ungerichteten oder gerichteten Graphen werden mit Markierungen versehen und können in Operationen auf Graphen als Entscheidungsgrundlage dienen.

Skript S. 95 ff

Ungerichteter Graph

TECHNISCHE UNIVERSITÄT DARMSTADT

Skript S. 95 ff

Gerichteter Graph

Skript S. 95 ff

Doppelt gerichteter Graph

Skript S. 96 ff

Gewichteter Graph

Skript S. 95 ff

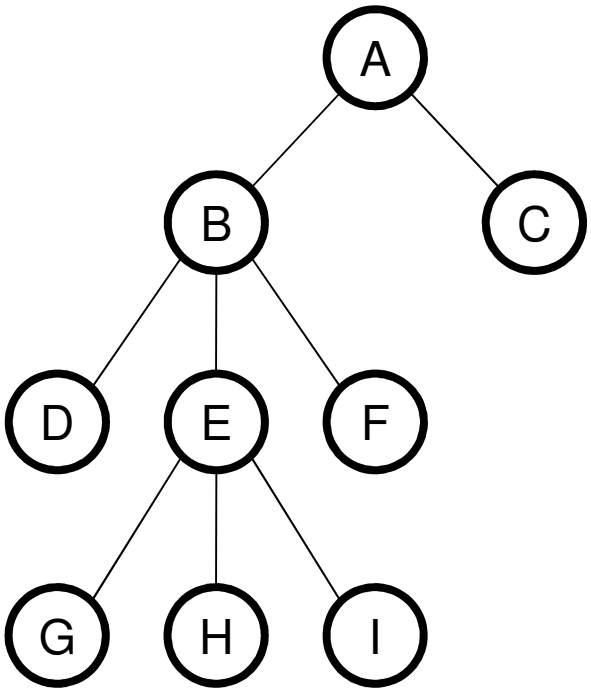
Abstrakter Datentyp: Graph

TECHNISCHE UNIVERSITÄT DARMSTADT

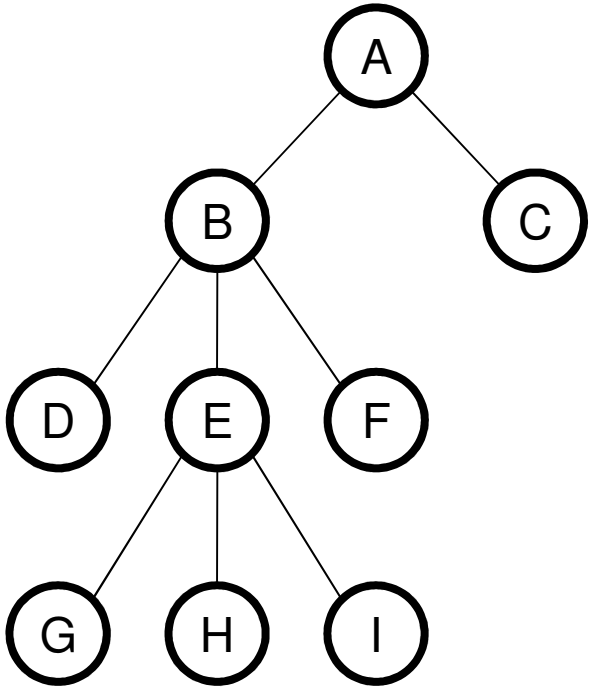
- Gebräuchliche **Operationen** auf Graphen sind:
  - Erzeugen von Knoten
  - Erzeugen von Kanten
  - Entfernen von Kanten und
  - Traversieren des Graphen, beginnend von einem bestimmten Knoten aus.
- Für das **Traversieren** eines Graphen gibt es zwei gebräuchliche Methoden:
  - Breitensuche:
    Nach dem Besuch des aktuellen Knotens werden alle Knoten besucht, die durch eine Kante mit dem aktuellen Knoten verbunden sind und noch nicht besucht wurden. Auf jeden einzelnen Knoten wird dann wieder die Breitensuche angewendet.
  - Tiefensuche:
    Nach dem Besuch des Knotens wird eine Kante gewählt, auf dessen verbundener Knoten wiederum die Tiefensuche angewendet wird. Steht keine weitere Kante zur Verfügung, oder führen die übrigen Kanten zu Knoten die schon besucht wurden, so wird die Kante zurückgewandert auf der der aktuelle Knoten aufgesucht wurde und dieser Knoten als aktueller Knoten für die Tiefensuche verwendet.

Skript S. 97 ff

Beispiel Breitensuche & Tiefensuche



Breitensuche



Breitensuche

vgl. Skript S. 98