Vorlesung Programmieren

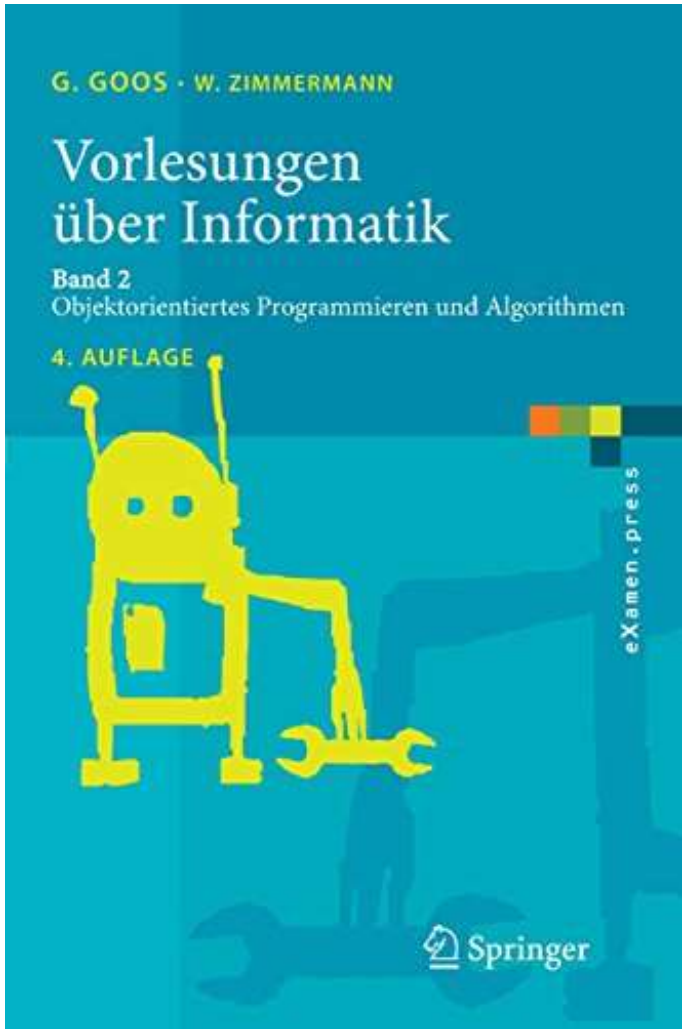
18. Vom Programm zur Maschine

PD Dr. rer. nat. Robert Heinrich

VOM PROGRAMM ZUR MASCHINE

Literaturhinweis – Weiterlesen

- Gerhard Goos, Wolf Zimmermann: *Vorlesungen über Informatik – Band 2: Objektorientiertes Programmieren und Algorithmen*, Springer Verlag Berlin/Heidelberg, 4. Auflage, 2006



Online verfügbar:
https://link.springer.com/book/10.1007%2F3-540-34778-X

Kapitel 11: *Vom Programm zur Maschine*

Lernziele

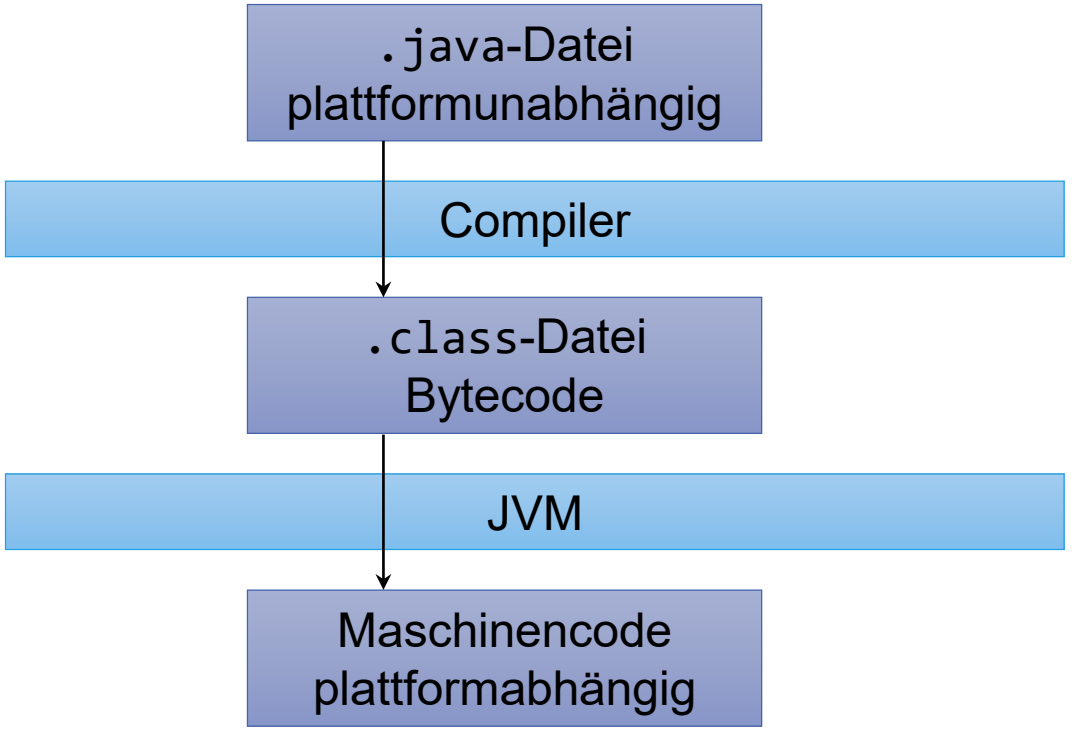
- Sie verstehen die Funktionsweise der **Halde** (engl. *heap*) und des **Kellerspeichers** (engl. *stack*).
- Sie können den Einsatz der Datenstrukturen bei der **Speichereinteilung** erläutern.
- Sie können den Einsatz der Datenstrukturen bei unterschiedlichen **Sprüngen** im Programmcode erklären.
- Sie können die Funktionsweise und den Einsatz der Datenstrukturen beim **Methodenaufruf** und **Verlassen einer Methode** erläutern.

Quelle: http://phdcomics.com

Motivation

Hintergrund: Von der **Java-Datei** über den plattformunabhängigen **Bytecode** hin zum plattformabhängigen **Maschinencode**

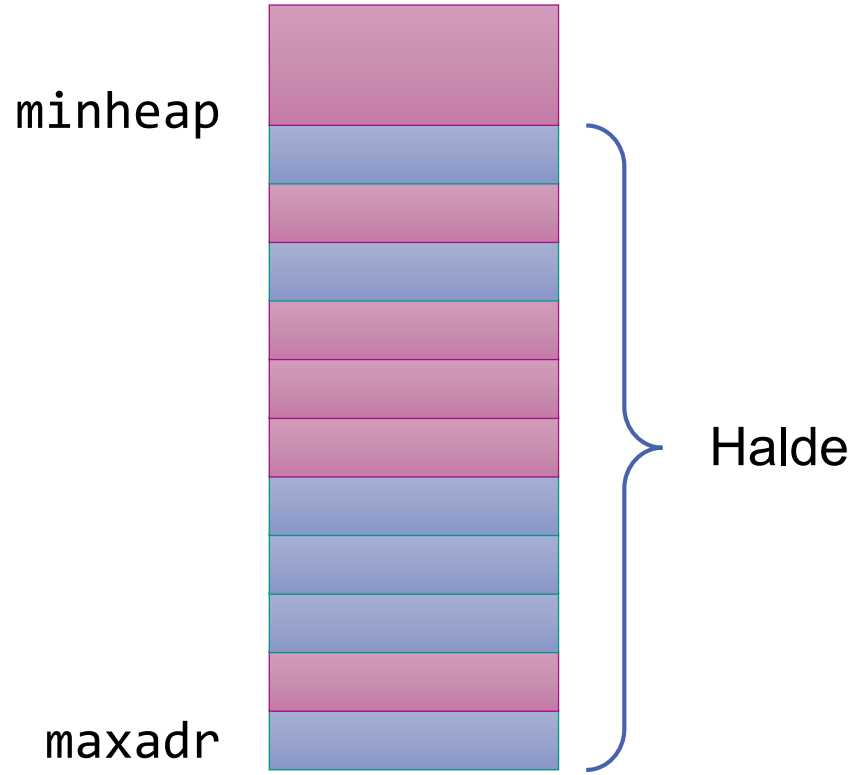
- Wie ist der Speicher aufgebaut?
- Was passiert bei einem Methodenaufruf?



Halde

- Die **Halde** (engl. *heap*) ist ein reservierter Speicher zur Speicherung von Elementen zur Laufzeit bei Bedarf
- Speichern der Daten in einer beliebigen Anordnung
- Freigabe des allokierten Speichers durch Aufhebung des Referenzierens
- Beispiel: Java Heap zur Speicherung von Objekten in Java

Dateneinteilung in der Halde



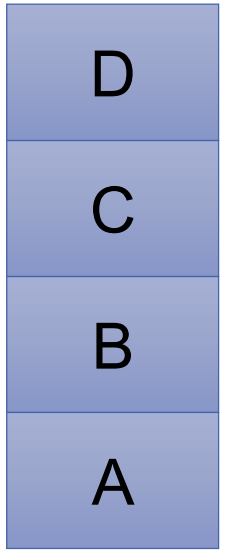
Kellerspeicher

Kellerspeicher (engl. *stack*), auch bekannt als Stapelspeicher, ist eine Datenstruktur.

- Arbeitsweise: Last-In-First-Out (LIFO)
- Push-Operation: Das neue Objekt wird auf den Keller gelegt (gestapelt)
- Pop-Operation: Das neueste Objekt vom Keller heruntergenommen

Befehlsreihenfolge:

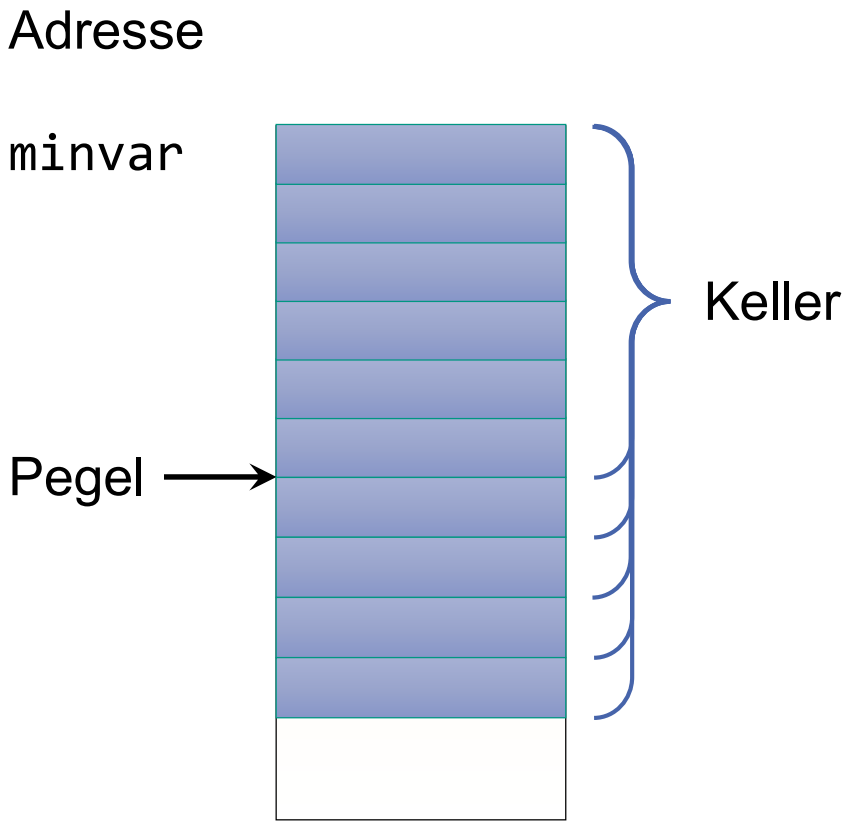
- Push(A)
- Push(B)
- Push(C)
- Push(D)
- D=Pop
- C=Pop
- B=Pop



Einsatz des Kellerspeichers

- Ein Kellerspeicher kann dynamisch wachsen und schrumpfen
- Allokation des Speichers zur Laufzeit
- Freigabe des allokierten Speichers nach Abarbeitung des Codes
- Einsatz in Prozessoren für Aufrufe von Unterprogrammen (Methoden)

Dateneinteilung im Keller



Wiederholung: Variablen und Speicher

x	y	z	a	b	c	d	e	f
0	3	12	44	0	20	3.4	0	-2
int	int	int	int	int	int	float	int	int
01	02	03	04	05	06	07	08	09

- Der Hauptspeicher besteht aus einer Reihung von Speicherplätzen
- Die Speicherplätze sind durchnummeriert (»Speicher-Adressen«)

Wiederholung: Variablen und Speicher

- Wir haben hier eine abstrakte Sicht auf den Speicher angenommen
- Üblicherweise wird an jeder Speicheradresse nur 1 Oktett (8 Bit, »Byte«) gespeichert.
  - Größere Datentypen werden auf mehrere aufeinanderfolgende Adressen verteilt
  - z.B. benötigt ein Java-`int` vier aufeinanderfolgende Adressen
- Für unsere Zwecke ist die abstrakte Sicht jedoch ausreichend

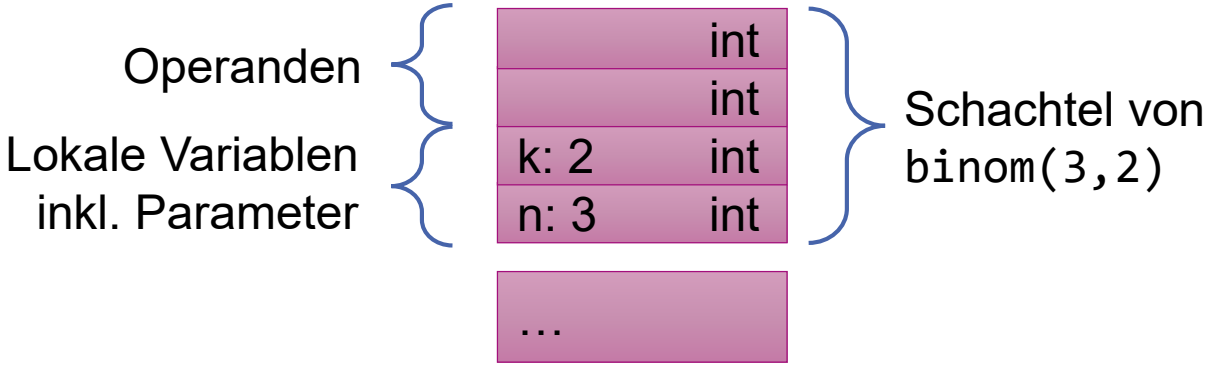
(aus Foliensatz 02: Typen und Variablen)

Wiederholung: Methoden im Aufrufstapel

Lokale Variable (und Parameter) einer Methode werden im **Aufrufstapel** (*call stack* oder **Laufzeitkeller**) abgelegt

Für jede Methode wird ein neuer Speicherbereich (**Schachtel** bzw. *frame*) auf dem Aufrufstapel angelegt mit u.a.

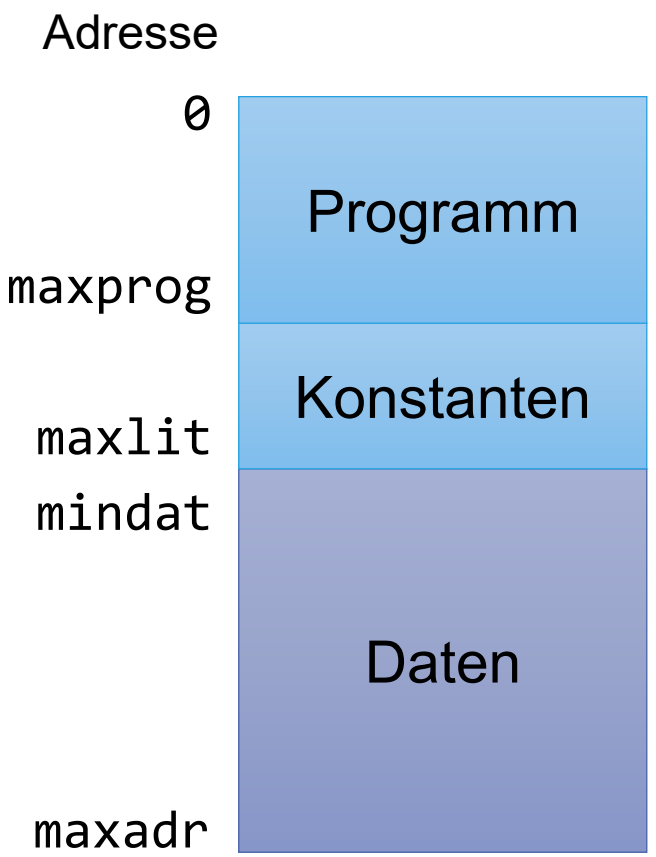
- Lokalen Variablen (inkl. Parameter)
- Operanden



(aus Foliensatz 11: Rekursion)

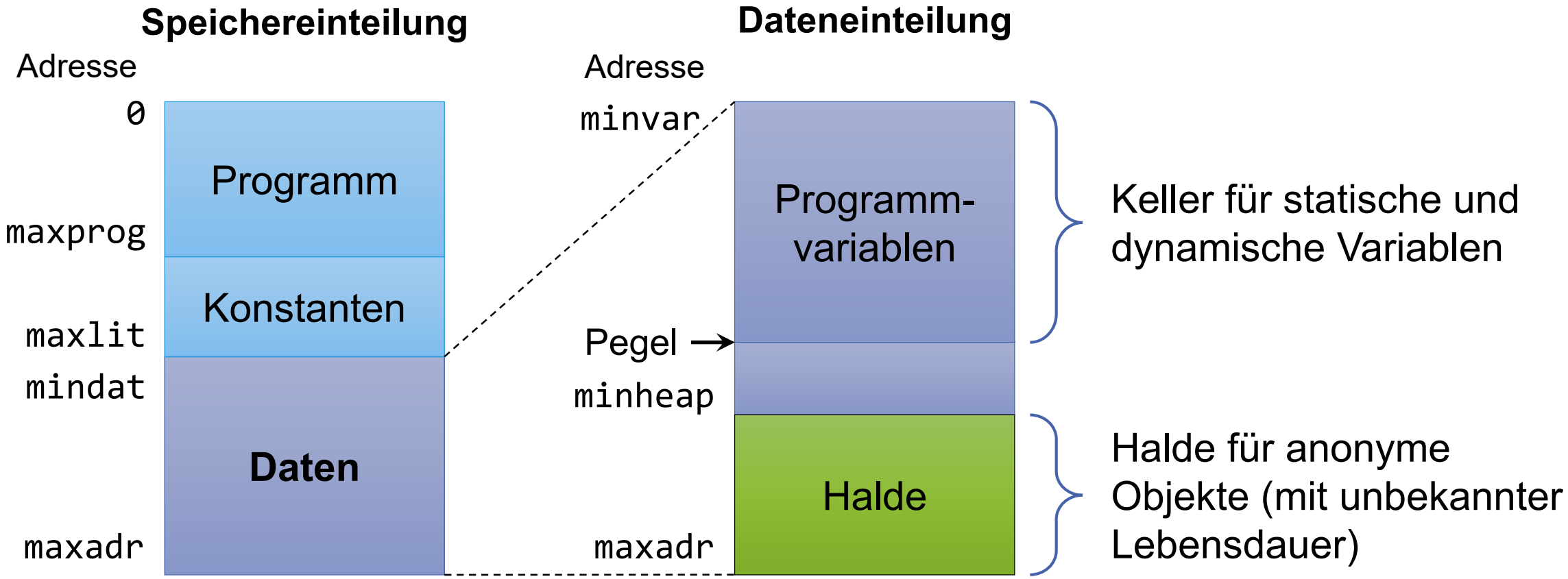
Speichereinteilung

- Ein Speicher besitzt einen Adressraum: `[0, maxadr]`.
- Die Elemente des Speichers `Speicher[i]` heißen **Speicherzellen**
- Der Wert einer Speicherzelle oder einer Variable heißt **Inhalt**
- Der Speicher des Programms besteht aus:
  - Programmbereich für Befehle
  - Konstantenbereich für alle Konstanten (Literale) des Programms
  - Datenbereich für alle Variablen



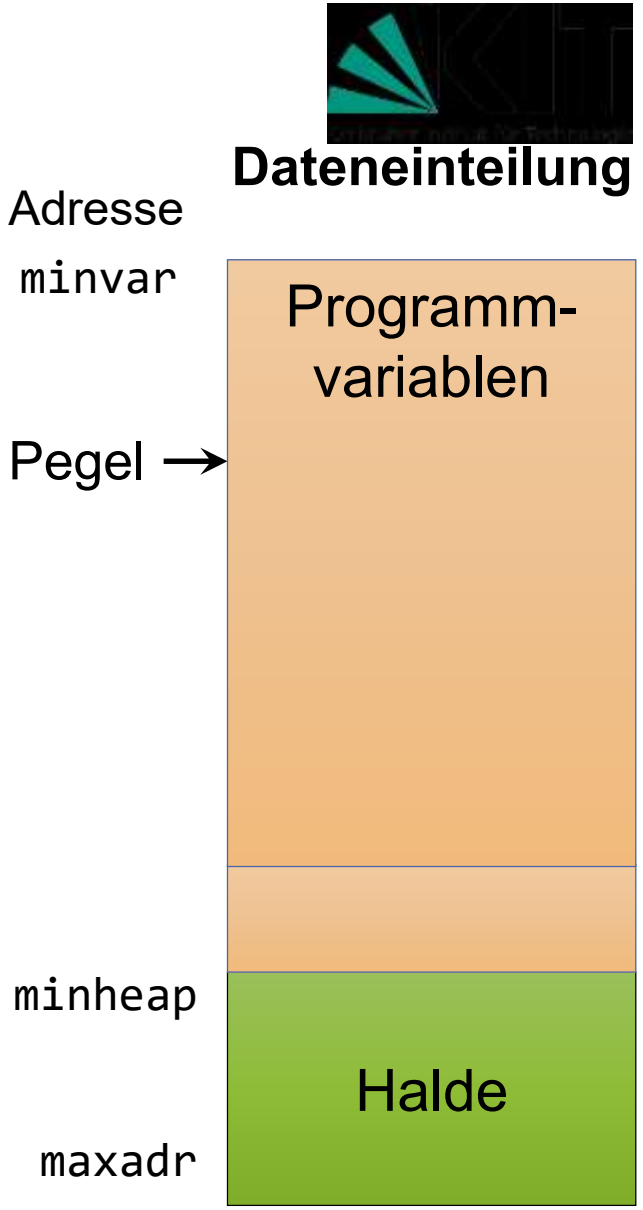
Einteilung des Datenbereichs

- Grund der Aufteilung des Speichers: Anzahl der Referenzobjekte a priori nicht bekannt
- Keller wächst nach unten



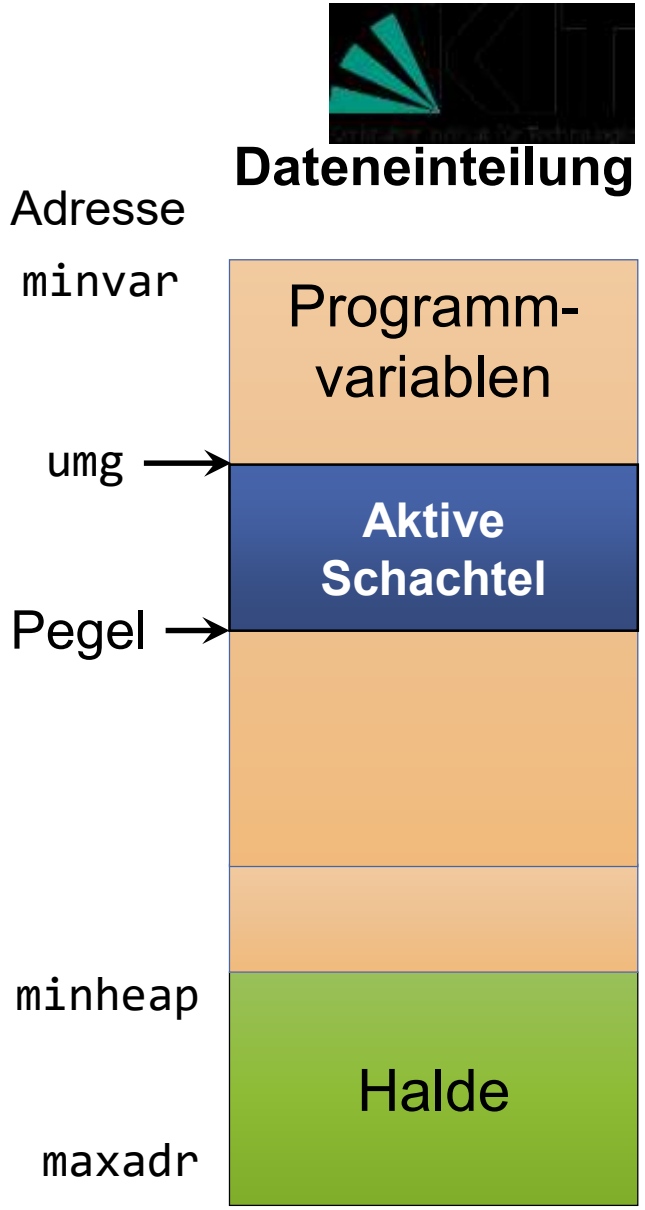
Laufzeitkeller

- Zur Speicherung statischer und dynamischer Programmvariablen
- Grund für Kellerspeicher: Geschachtelte Unterprogrammaufrufe
- Kellerspeicher besteht daher aus Einheiten, bekannt als **Schachteln** (engl. *stack frame, activation record*)
- Unterste Schachtel reserviert für statische Variablen
- Schachtel beinhaltet:
  - lokale Variablen
  - Rückkehradresse
  - etc.



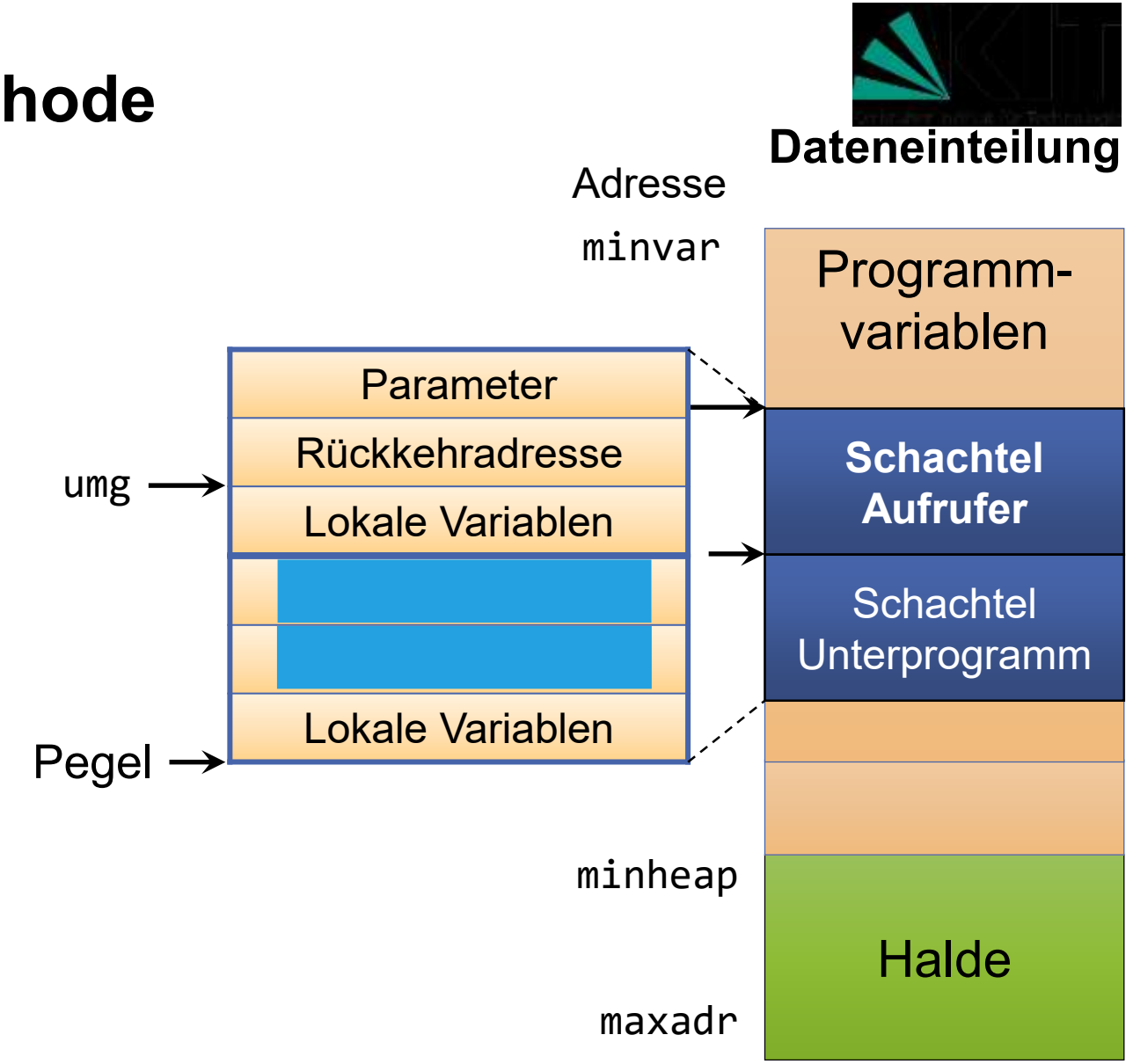
Adressierung durch drei Basisadressen

- `minvar`: die **Anfangsadresse** des Speicherbereichs statischer Programmvariablen
- **Umgebungszeiger** (`umg`): Die Basisadresse für die dynamischen Variablen (lokalen Variablen) in der Schachtel des laufenden Unterprogramms
- Pegel: **Kellerpegel**
  - Adresse des obersten Elementes im Laufzeitkeller
  - Basisadresse für Zwischenergebnisse
  - wird bei jedem Unterprogrammaufruf um Größe der Schachtel erhöht
- Adresse von Variable v mit **relativer Adresse** r_v:
  - bei statischer Variable: `minvar` $+ r_v$
  - bei dynamischer Variable: `umg` $+ r_v$



Schritte beim Aufruf einer Methode

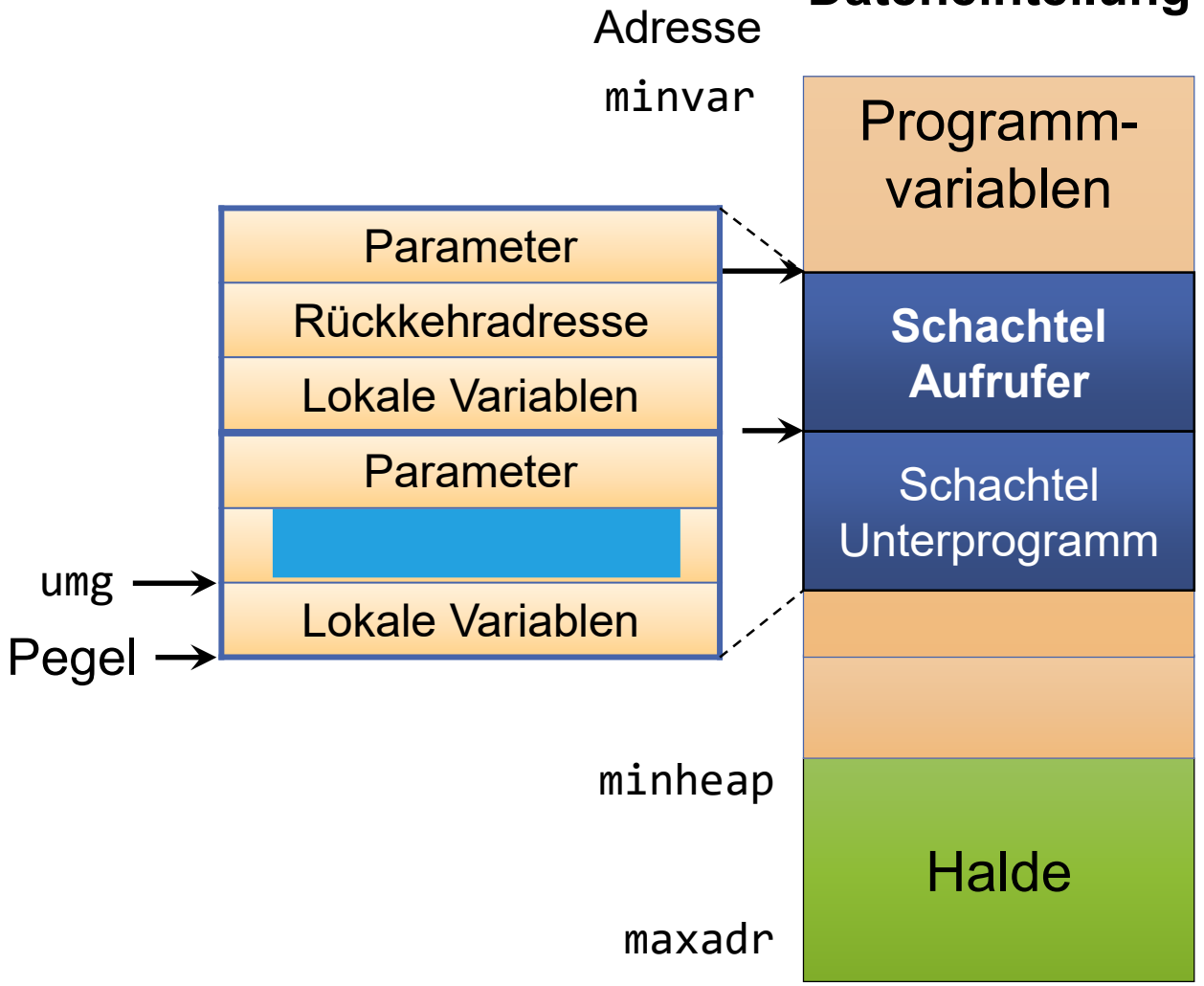
1. Erhöhung des Kellerpegels um den Umfang der Schachtel
2. Zuweisung der Werte der aktuellen Parameter in die Schachtel
3. Eintrag der Rückkehradresse in die Schachtel
4. Setzen des Umgebungszeigers auf die Schachtel
5. Methodenaufruf: Sprung auf die erste Adresse der Methode



Schritte beim Verlassen einer Methode

1. Zuweisung eines eventuellen Ergebnisses
2. Zuweisung der Rückkehradresse an ein Register R
3. Rücksetzen des Umgebungszeigers
4. Erniedrigung des Kellerpegels zum Aufbau des Kellers
5. Unbedingter Sprung auf die Rückkehradresse

Dateneinteilung



Zusammenfassung

- **Halde** (engl. *Heap*) ist ein reservierter Speicher zur Speicherung von Elementen zur Laufzeit bei Bedarf in beliebiger Reihenfolge
- **Kellerspeicher** (engl. *Stack*) ist eine Datenstruktur nach Last-In-First-Out-Prinzip
  - Allokation des Speichers zur Laufzeit
  - Freigabe des allokierten Speichers nach Abarbeitung des Codes
  - Häufiger Einsatz in Prozessoren
- Der Speicher des Programms besteht aus:
  - Programmbereich für Befehle
  - Konstantenbereich für alle Konstanten (Literale) des Programms
  - Datenbereich für alle Variablen
- Aufruf einer Methode und Verlassen einer Methode

AUSBLICK

Wie geht es weiter?

Übungsschein nicht bestanden?

- Übungsblätter, Präsenzübung, Abschlussaufgaben auch im SoSe 24 angeboten, keine Vorlesung, Präsenztutorien (freiwillig) (*)

Für Informatikstudenten/-innen:

- 2. Semester: Vorlesung Softwaretechnik
- 3. Semester: Praxis der Software-Entwicklung
- Mehr Veranstaltungen zur Softwaretechnik unter https://sdqweb.ipd.kit.edu/wiki/Lehre

Und wie verbessern Sie Ihre Programmierkenntnisse?

- Lesen Sie Programme!
- Schreiben Sie Programme!
- Nutzen Sie Informationsquellen im Internet (z.B. http://stackoverflow.com)
- Nehmen Sie an Veranstaltungen teil (z.B. Hackathon Karlsruhe)

(*) https://sdqweb.ipd.kit.edu/wiki/Übung_Programmieren_SS2024

Wie geht es weiter?
Ein nicht vollständiger Java-Überblick

- Strukturierung von Programmen:
  - Ab Java 9: Module
- Dependency-Management
  - Beim Bauen
    - Maven
    - Gradle
  - Zur Laufzeit
    - Dependency Injection z.B. mit Spring
    - OSGI
- Server-Programmierung
  - Spring
  - Jakarta EE
- Multi-Thread
  - Threads, Executors,…
- GUI-Programmierung
  - Swing, SWT, JavaFX
- Android-Programmierung
- Einbindung externer Datenquellen
  - JDBC
  - ObjectMapper z.B. Hibernate, Jackson …
  - Java NIO
- Neue Java-Funktionen (11 vs 21)
  - Records
  - Sealed Classes
  - …

Viel Erfolg in Ihrem weiteren Studium!
Viel Spaß beim Programmieren!