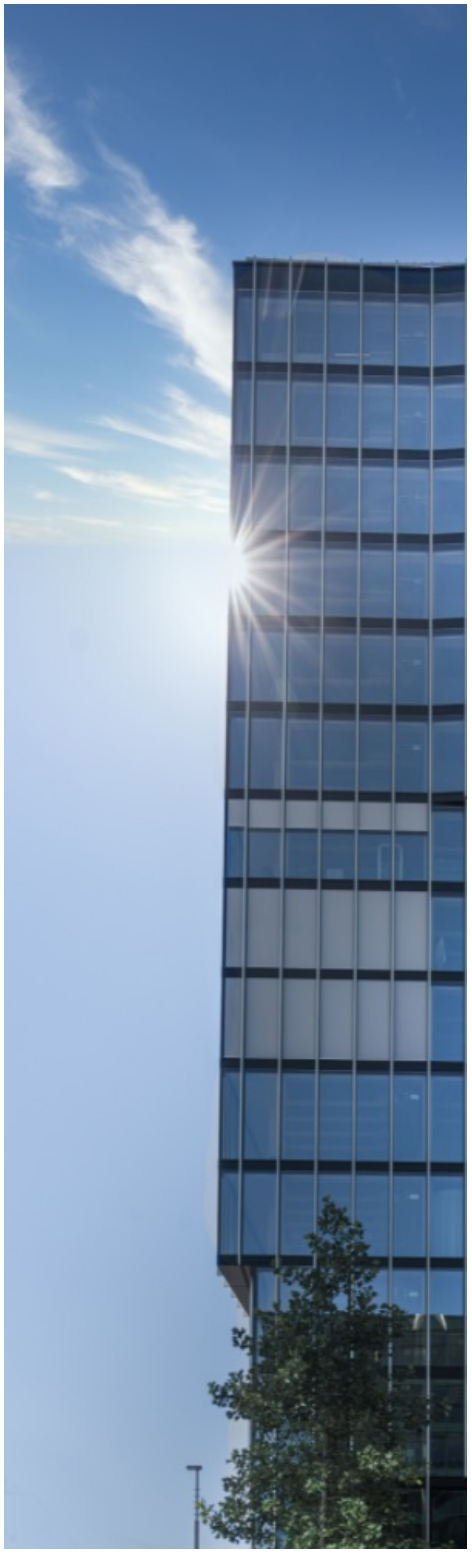
HSLU Hochschule Luzern

Algorithmen und Datenstrukturen

Datenstrukturen: Collections - Refresher

Roland Gisler / Ingmar Baetge

FH Zentralschweiz



Inhalt

- Was sind Datenstrukturen?
- Eigenschaften von Datenstrukturen
- Java Collection Framework
- Beispiel: **`List`** und **`ArrayList`**
- Speicherverwaltung
- Zusammenfassung

Was sind Datenstrukturen?

Was sind Datenstrukturen?

- Sehr häufig haben wir es in der Programmierung mit Mengen von Daten bzw. Objekten zu tun.
  Beispiele:
  - Messwerte (z.B. **`Temperatur`**)
  - Bankkonten (mit Besitzer und Kontostand)
  - Formen (wie z.B. **`Circle`** oder **`Rectangle`**)
  - Personen (z.B. Studierende)

- Datenstrukturen werden verwendet, um **Mengen** von Daten bzw. Objekten **effizient zu speichern** und **verarbeiten** zu können.
- Unter Verarbeiten versteht man Funktionen, wie z.B. alle enthaltenen Daten/Objekte einzeln zu bearbeiten, nach bestimmten Objekten zu suchen, zu zählen, zu filtern oder nach unterschiedlichen Kriterien zu sortieren etc.

Verschiedene Arten von Datenstrukturen

- Es gibt verschiedene Arten von Datenstrukturen. Beispiele:
  - **Array** – indexierte Reihung (typisch als Sprachbestandteil).
  - **Tree** – hierarchisch geordnete Daten (Baumstruktur).
  - **List** – einfache Reihung.
  - **Map** – Zuordnung zwischen Schlüssel und Wert(-paaren).
  - **Queue** – (Warte-)Schlange, FIFO.
  - **Set** – Sammlung ohne Duplikate.
  - **Stack** – Stapel- oder Kellerspeicher, FILO.
- Java bietet für jede Art wiederum verschiedene Implementationen dieser Datenstrukturen an, um den unterschiedlichen Anforderungen gerecht zu werden ➔Java Collections Framework.
- Hinweis: Eine universelle Datenstruktur, die **alles** perfekt kann, gibt es (leider) nicht, wir müssen fallspezifisch auswählen!

Eigenschaften von Datenstrukturen

- Die verschiedenen Datenstrukturen unterscheiden sich in vielerlei Hinsicht, zum Beispiel:
  - **Grösse**: Dynamisch oder statisch.
  - **Zugriff**: Direktzugriff oder indirekt / sequenziell.
  - **Sortierung**: Sortiert oder unsortiert, mit/ohne Ordnung.
  - **Suche**: Beschleunigte Suche (z.B. binär oder über Hashwert).
  - **Geschwindigkeit**: Grundlegende Operationen wie Suchen, Einfügen, Anhängen, Entfernen, Verschieben von einzelnen Elementen an verschiedenen Positionen.
- Anhand dieser **Eigenschaften**, und den teilweise damit eng verbundenen Algorithmen, sollten wir für jeden Anwendungsfall gezielt die am besten geeignetste Datenstruktur auswählen!

Wichtige Ordnungsfunktionen

Aufgelistet gemäss Wachstumsrate:

▪ konstant	$O(1)$	Hashing
▪ logarithmisch	$O(\ln n)$	binäres Suchen
▪ linear	$O(n)$	Suchen in Texten
▪ n log n	$O(n \cdot \ln n)$	raffiniertes Sortieren
▪ polynomiell ①	$O(n^k)$	naives Sortieren
▪ exponentiell	$O(a^n)$	Optimierungen
▪ Fakultät	$O(n!)$	Permutationen, TSP②

① In der Praxis anwendbare Algorithmen haben typischerweise Ordnungsfunktionen $\leq n^2$ (quadratisch) oder n^3 (kubisch).

② TSP = «Traveling-Salesman-Problem»

Java Collection Framework

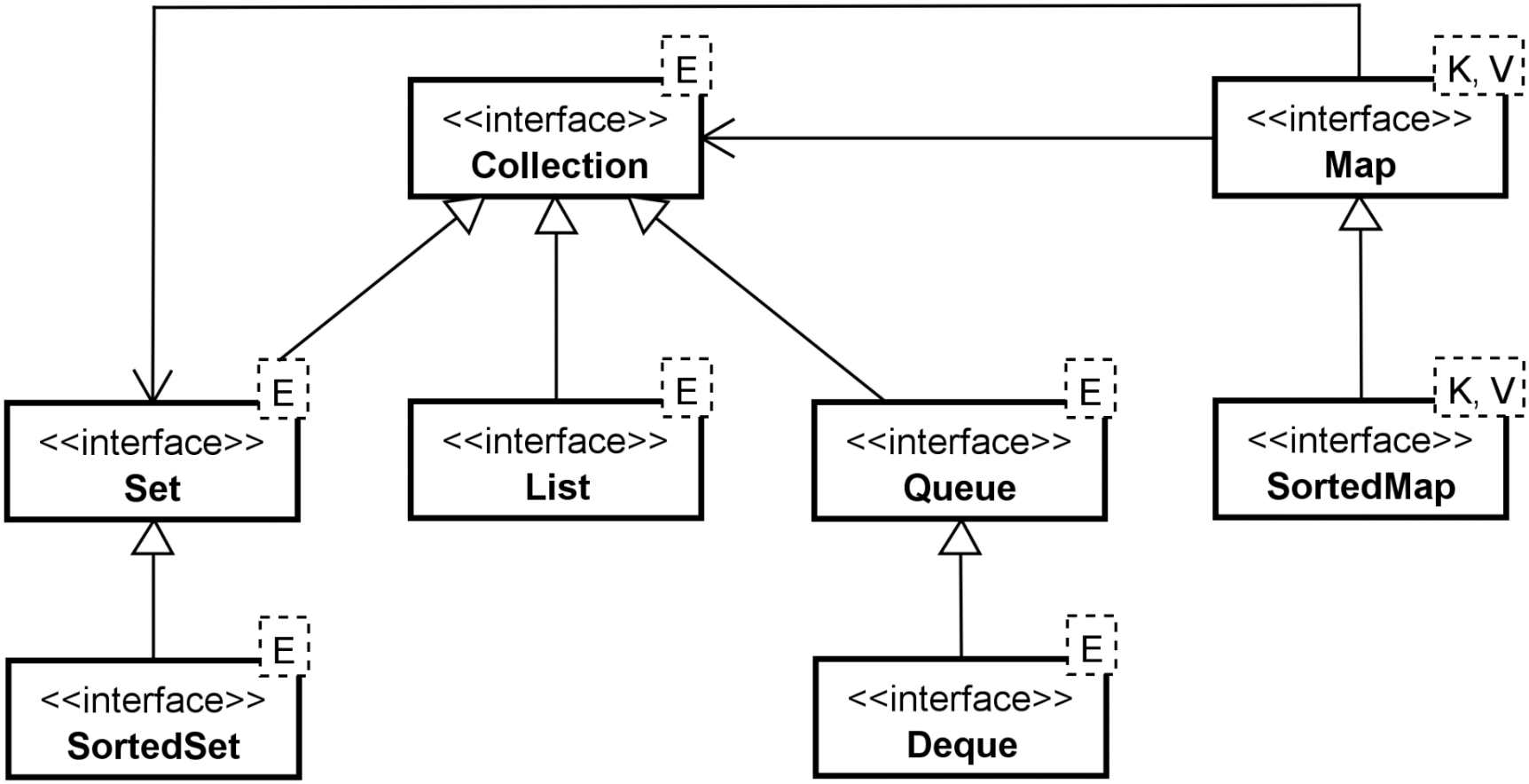
Java Collection Framework - Übersicht

Das Collection Framework besteht aus drei wesentlichen Inhalten:

- **Interfaces**: Abstrakte Datentypen, welche verschiedenartige Datenstrukturarten repräsentieren.
  - `List<E>` abstrahiert Listen, `Map<K,V>` abstrahiert Map etc.
- **Implementationen**: Konkrete, wiederverwendbare Implementationen der durch die Schnittstellen definierten Datenstrukturen.
  - `LinkedList<E>` und `ArrayList<E>` sind zum Beispiel zwei Implementationen von `List<E>`.
- **Algorithmen**: Methoden zur Behandlung von Datenstrukturen. Meist polymorph implementiert.
  - `iterator()` ermöglicht sequenziellen Zugriff auf alle Elemente, unabhängig von der konkreten Implementation.
  - `sort(List<E>)` sortiert beliebige `List`-Implementationen.

Java Collection Framework – Interfaces - Übersicht

- Auszug aus den vorhandenen Schnittstellen (nicht vollständig):



Java Collection Framework - Funktionsbasis

- Damit die polymorphen Algorithmen und Datenstrukturen des Collection Frameworks mit **beliebigen** Elementen funktionieren können, benötigen wir ein paar grundlegende Funktionen:
  - Für die **Suche** von Objekten (oder Verhindern von Duplikaten) müssen wir Objekte als **«gleich»** erkennen können.
  - Für die **Sortierung** müssen wir Objekte vergleichen und als **kleiner** (<), **gleich** (==) oder **grösser** (>) beurteilen können.
- Daraus resultiert, dass
  - die Klasse **`Object`** (**die** Basisklasse) einige dafür notwendige Methoden definiert: **`equals()`** und **`hashcode()`**
  - es zusätzliche Interfaces gibt, die wir ggf. implementieren müssen: **`Comparable<T>`** oder **`Comparator<T>`**
- siehe OOP Input ➔ **O09_IP_ObjectEqualsCompare.pdf**

Beispiel: ArrayList im Einsatz

```
final List<Temperatur> verlauf  = new ArrayList<>();

verlauf.add(new Temperatur(10.2));
verlauf.add(new Temperatur(15.8));
verlauf.add(new Temperatur(21.3));

System.out.println("Anzahl Messwerte: " + verlauf.size());
```

```
Anzahl Messwerte: 3
```

```
verlauf.set(1, new Temperatur(18.5));

final Temperatur t2 = verlauf.get(1);
System.out.println("Zweiter Messwert: " + t2.getCelsius());
```

```
Zweiter Messwert: 18.5
```

Speicherverwaltung

Speicherverwaltung - Grundlagen

- Wie verwaltet das Betriebssystem eigentlich Speicher?
- Stark vereinfachte Annahme: Der Speicher steht mit einem linearen Adressraum ab Adresse `0` zur Verfügung.
  - Beispiel: Ab Adresse `$00` bis `$0F` (also gerade mal 16 Bytes)

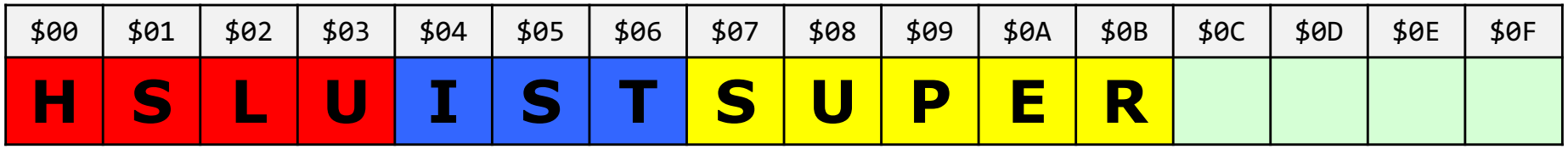
$00	$01	$02	$03	$04	$05	$06	$07	$08	$09	$0A	$0B	$0C	$0D	$0E	$0F

- Speicher wird in Blöcken angefordert (**m**emory **alloc**ation).
  - Beispiele:

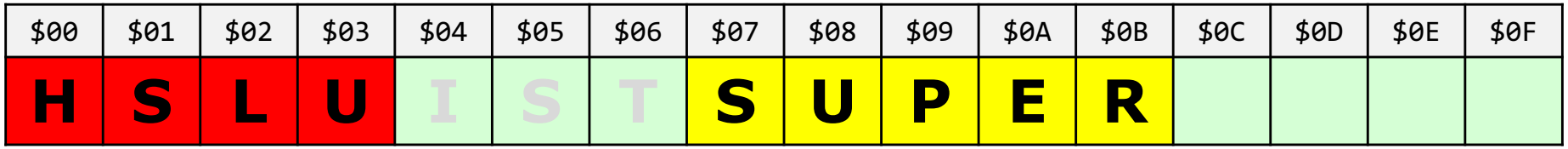
```
red = malloc(4); red.write("HSLU");
blue = malloc(3); blue.write("IST");
yellow = malloc(5); yellow.write("SUPER");
```

$00	$01	$02	$03	$04	$05	$06	$07	$08	$09	$0A	$0B	$0C	$0D	$0E	$0F
H	S	L	U	I	S	T	S	U	P	E	R				

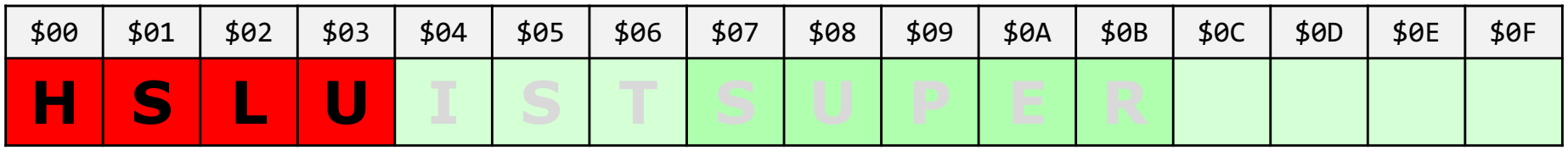
Speicherverwaltung - Fragmentierung



- Speicher wird (netterweise) auch wieder freigegeben (free).
  - Beispiel: `free(blue); // kein Java!`



- Was passiert nun, wenn man versucht einen Block mit `malloc(7)` anzufordern?
- Was passiert, wenn man auch noch `free(yellow)` freigibt?



Speicherverwaltung - Anforderungen

- Verwaltung soll zuverlässig sein, wertvoller Speicherplatz soll nicht verloren gehen (keine «memory leaks»).
- Allokationen sollen **schnell** sein: Keine lange Suche nach einem in der Grösse passenden Block.
- Freigabe soll ebenfalls **schnell** sein.
- Mit der Zeit entstehen Lücken sowohl durch belegten als auch freien Speicher (Fragmentierung).
- Liegen freie Lücken **nebeneinander**, sollten diese in der Datenstruktur wieder zusammengefasst werden!
- Das Betriebssystem muss Speicher an Prozesse vergeben und diese Speicherbereiche voneinander isolieren (Sicherheit)
- **Speicherverwaltung ist komplex, und wird in Teilen vom Betriebssystem, in Teilen von der jeweiligen Programmiersprache übernommen.**

Speicherverwaltung

- In Hochsprachen wie Java, C# oder Python können wir nicht direkt mit dem Speicher interagieren (vgl. C/Assembler) – Speicher wird automatisch bereitgestellt, wenn wir Objekte instanziieren (**new**)
- Wir haben keinen direkten Zugriff auf Speicheradressen (*Referenzen* statt *Pointer*)
- Die Freigabe von nicht mehr benutztem Speicher übernimmt der *Garbage Collector*
- Damit können viele Fehler vermieden werden, aber wir haben weniger/indirekte Steuerung von Speicherverbrauch
- bei vielen Datenstrukturen werden sich die Datenelemente zufällig im Speicher anordnen, abhängig von der Speichersituation beim Erzeugen der Element-Objekte (Ausnahme: Array)

Zusammenfassung

- Datenstrukturen dienen zur Verarbeitung von Mengen von Daten bzw. Objekten.
- Es gibt verschiedene Datenstrukturen mit unterschiedlichen Fähigkeiten und Eigenschaften.
- Das Java Collections Framework enthält Schnittstellen, Algorithmen und konkrete Implementationen von und zu Datenstrukturen.
- **`ArrayList`** ist ein Beispiel einer sehr häufig eingesetzten, konkreten (Listen-)Datenstruktur in Java.
- Die Auswahl der geeigneten Datenstruktur erfolgt aufgrund der konkreten Anforderungen.
  - Aufgrund der Eigenschaften wie Zugriffsart, Art und Häufigkeit der Operationen, Datenmenge und –struktur.

HSLU Hochschule Luzern

Fragen?