



# Informatik I

## Übungsstunde 12

Autumn 2020

# Plan

- Kahoot
- Besprechung Bonusaufgabe 3
- Review Aufgaben 11: Dynamic Datastructures
- Repetition Generics, Datentypen
- Anwendung Generics (nur Erklären)
- Preview Aufgaben 12: Collections and Generics

# Bonusaufgabe "Music Player"

Verwendet die Datenstrukturen aus: `java.util`. Beispiel:  
`ArrayList`, `Queue`, `Stack` usw.

# Review Aufgaben 11

- Print a list backwards – gut
- Queue invariants – gut
- Queue: gut, aber Optimizations möglich
- Dictionary (noch zu korrigieren)

# Repetition Generics

*LinkedList<T>*



Wir wollen die Implementation von ArrayList, LinkedList, Stack, Queue, Set, Map für einen **beliebigen (generischen) Datentyp** wiederverwenden.

# Repetition Generics

Wir wollen die Implementation von ArrayList, LinkedList, Stack, Queue, Set, Map für einen **beliebigen (generischen) Datentyp** wiederverwenden.

Lösung: **Generics**.

# Generische Liste in Java: `java.util.List`

```
import java.util.LinkedList;
import java.util.List;
...
// Liste von Strings
List<String> list = new LinkedList<String>();

list.add("abc");
list.add("xyz");
Out.println(list.get(0)); // abc
```

# Generische Liste in Java: `java.util.List`

```
import java.util.LinkedList;
import java.util.List;
...
// Liste von Strings
List<String> list = new LinkedList<String>();

list.add("abc");
list.add("xyz");
Out.println(list.get(0)); // abc
```



# Generischer Listknoten

```
// ListNode mit generischem Werttyp T
class ListNode <T> {
    T value;
    ListNode<T> next;

    ListNode (T value, ListNode<T> next){
        this.value = value; this.next = next;
    }
}
```

---

Verwendung:

```
ListNode<String> n = new ListNode<String>("ETH", null);
```

# Code Beispiel

Code Beispiel (CodeExpert)  
Generic Stack



# Generischer Stack

```
public class Stack<T>{  
    private ListNode<T> top_node; // initialized with null  
    public void push(T value){  
        top_node = new ListNode<T>(value, top_node);  
    }  
    public T pop(){...}  
    public void output(){...}  
}
```


---

```
Stack<String> s = new Stack<String>();  
s.push("ETH");  
s.push("Hello");  
s.output(); // Hello ETH
```

# Generische *sortierte* Liste?

```
public class SortedList<T>{  
    private ListNode<T> head; // initialized with null  
    ...  
    // in a sorted way (sorted ascending by value)  
    public void insert(T value){  
        ListNode<T> n = head;  
        ListNode<T> prev = null;  
        while (n != null && value > n.value){  
            prev = n;  
            n = n.next;  
        }  
        ...  
    }  
}
```

error: bad operand types for binary operator '>'




# Code Beispiel

Code Beispiel (CodeExpert)  
Generic Sorted List



# Generische sortierte Liste!

```
public class SortedList <T extends Comparable<T>>{  
    private ListNode<T> head; // initialized with null  
    ...  
    // in a sorted way (sorted ascending by value)  
    public void insert(T value){  
        ListNode<T> n = head;  
        ListNode<T> prev = null;  
        while (n != null && value.compareTo(n.value)>0){  
            prev = n;  
            n = n.next;  
        }  
        ...  
    }  
}
```



extends Comparable<T> stellt sicher,  
dass die Methode T.compareTo existiert.

# Repetition Datentypen

## Datentypen

- Array
- List: ArrayList, LinkedList
- Stack (LIFO)
- Queue (FIFO)

## Neue Datentypen

- Set: Set, HashSet
- Map: Map, HashMap

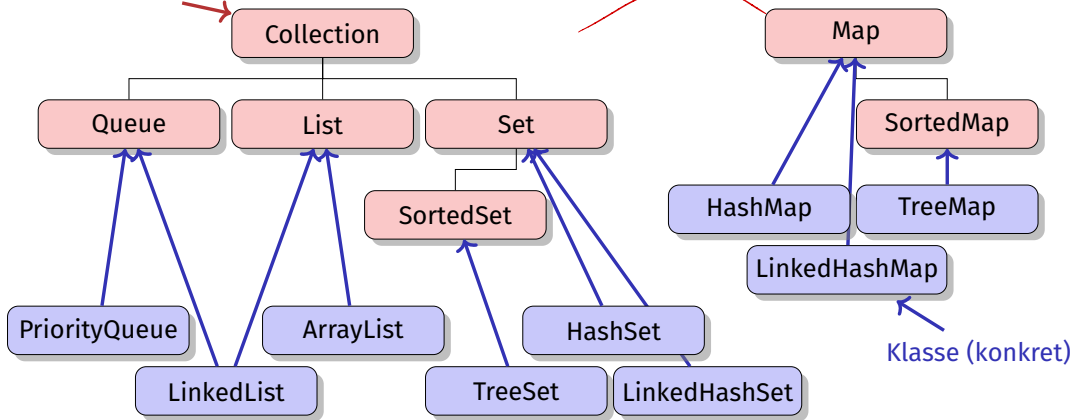
$$\{A, B, C\} \equiv \{C, B, A\}$$

Key  $\mapsto$  Value

$$\{ \text{"key1"} \mapsto 3, \text{"key2"} \mapsto 3, \dots \}$$

# Java Collections / Maps

Interface (abstrakt)





## Inventar - Produkte und Bestellungen

Erinnerung: Verfügbare Produkte, Preise und Kundenbestellungen speichern.

# Produkte und Bestellungen

Implementiere ein Bestellsungsverwaltungssystem, das verfügbare Produkte, Preise, Bestellungen (in welcher ein Produkt mehrmals vorkommen kann) und Preise speichert. Das System sollte automatisch den Gesamtpreis jeder Bestellung berechnen.

# Produkte und Bestellungen: Anwendung

Menge: Set

$\Leftrightarrow$  "wir haben einen Index"    key  $\mapsto$  Produkte.

Entwerfe eine Klasse so, dass in der Methode **main**:

- wir ein indiziertes Inventar von Produkten erstellen können (jedes mit einem Produktcode und einem Preis);
- wir eine Sammlung von Bestellungen erstellen können;
- wir Artikel zu Bestellungen hinzufügen können;
- wir Produkte und Bestellungen ausgeben können (und ihre Preise).

# Produkte und Bestellungen: Preise

→  $O(1) = c \cdot 1$  "die Laufzeit ist konstant"  $\Leftrightarrow$  "das Finden ist immer gleich schnell"

Beginne mit dem Code von Woche 8 und aktualisiere ihn, sodass die richtigen Java Collections verwendet werden.

Das Finden eines Inventarprodukts anhand seines Codes sollte in  $O(1)$  erfolgen.  $\Rightarrow$  Hash Map brauchen. ~~Arrays~~ ~~ArrayLists~~

Die Bestellungen sollten eine Menge ("set") sein und jeder enthält eine Menge ("set") von Artikeln.

# Preview Aufgaben 12