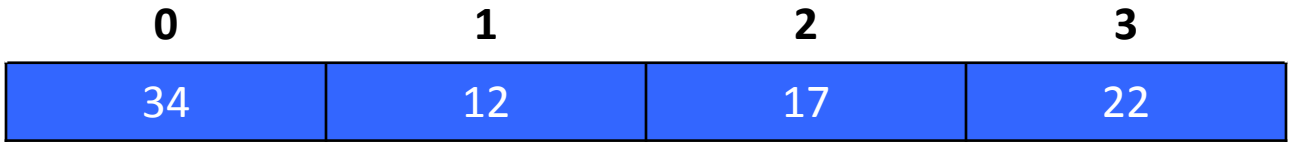
Javakurs FSS 2012

Lehrstuhl Stuckenschmidt

Tag 4 – ArrayList, PriorityQueue, HashSet und HashMap

Array List

- Dynamisches Array
  - `ArrayList` vertritt ein Array mit variabler Länge
  - Kapazität passt sich automatisch an
  - ArrayList wird selbstständig vergrößert



Array List

- `ArrayList<E>`
  - Generische Datentypen
  - Ermöglicht es unterschiedliche Datentypen zu verwenden
  - Implementierung findet Typ unabhängig statt

```
class Box<E>{
    E val;
    void setVal(E val){
        this.val = val;
    }
    E getVal(){
        return val;
    }
}
```

Array List

- `ArrayList<E>`
  - Generische Datentypen müssen Objekte sein
    - » `String`, `DeinObjekt`…
  - Primitive Datentypen **nicht** verwendbar
    - » `int`, `boolean`, `char`…

```
Box<String> stringBox = new Box<String>();
Box<DeinObjekt> deineOBox = new Box<DeinObjekt>();
```

Box für Zahlen? Boolsche Werte??

Wrapperklassen

- Objekte primitiver Datentypen
  - Wrapper-Objekte nehmen primitive Datentypen auf
  - Verfügen über zusätzliche Funktionen

Wrapperklasse	Primitver Datentyp
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Boolean	boolean
Character	char

Wrapperklassen

- Erzeugen von Wrapper-Objekten
  - Wrapper-Objekte sind `final`
  - Konstruktoren

```
Integer(int i)
Integer(String s)
```

  - Funktionen

```
toHexString(int i)
toDoubleValue()
```

  - Wert verändern

```
Integer i = new Integer(4);
i = new Integer(i.intValue()+1);
```

Autoboxing

- Boxing und Unboxing
  - Primitive Datentypen <-> Wrapper-Objekte
  - Boxing: Erstellen eines Wrapper-Objektes

```
Interger i = 42;
i = i + 15;                    //Unboxing & Boxing
```

  - Unboxing: Beziehen des Wertes aus einem Wrapper-Objekt

```
int k = i;
k = i + 3;
```

Autoboxing

- Vorsicht
  - Probleme mit ==
  - Erwartet keine primitiven Datentypen
  - Daher kein Unboxing
  - Referenzvergleich – `comparedTo, equals()`

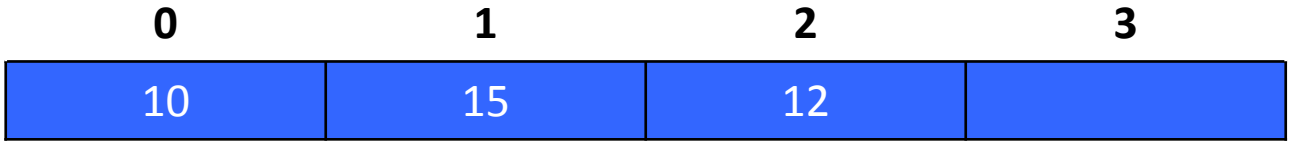
```
Integer i = 128;
Integer j = 128;
System.out.println( i >= j);
System.out.println( i <= j);
System.out.println( i == j);
```

```
true
true
false
```

Array List

- Erstellen einer ArrayList
  - Datentyp festlegen
  - Kapazität festlegen
  - Elemente einfügen

```
ArrayList<Integer> liste;
liste = new ArrayList<Integer>(4);
liste.add(10);
liste.add(3,15);
liste.add(12);
```



Array List

- Kapazität und Länge
  - Kapazität
    - » Felder die insgesamt zur Verfügung stehen
  - Länge
    - » Felder die Belegt werden
  - Kapazität ≥ Länge

Array List

- Kapazität
  - Viele Funktionen für die Kapazität

```
ArrayList(int initialCapacity)
ensureCapacity(int minCapacity)
trimToSize()
```

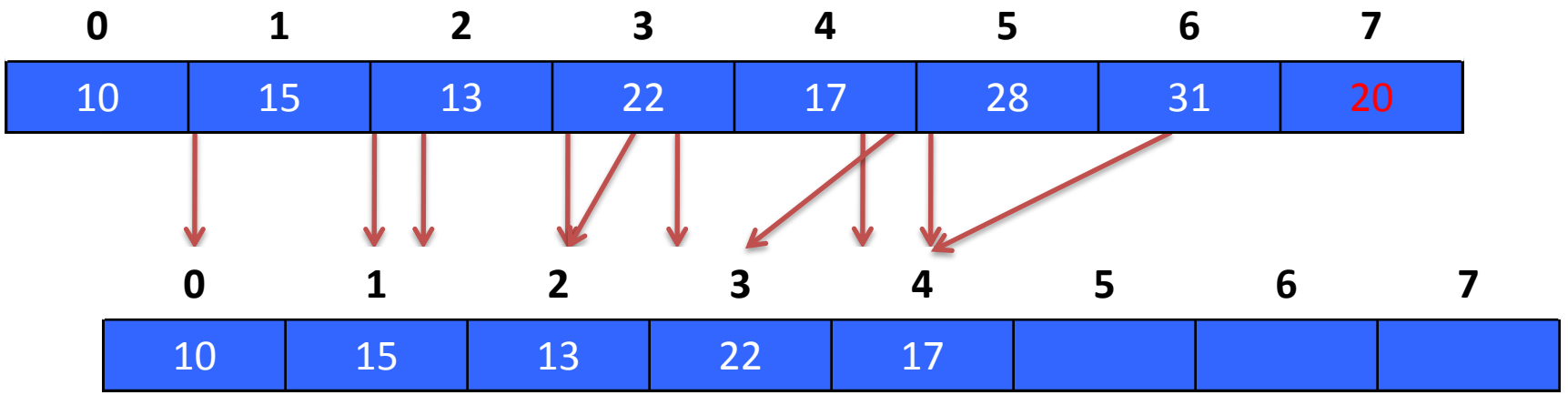
  - Kapazitätsmanagement beeinflusst Schnelligkeit
  - ArrayList wird um das 1,5 Fache vergrößert

Array List

- Einfügen von 8 Elementen

```
ArrayList<Integer> liste;
liste = new ArrayList<Integer>(2);
liste.add(10);
liste.add(15);
liste.add(13);
```

```
liste.add(22);
liste.add(17);
liste.add(28);
liste.add(31);
liste.add(20);
```

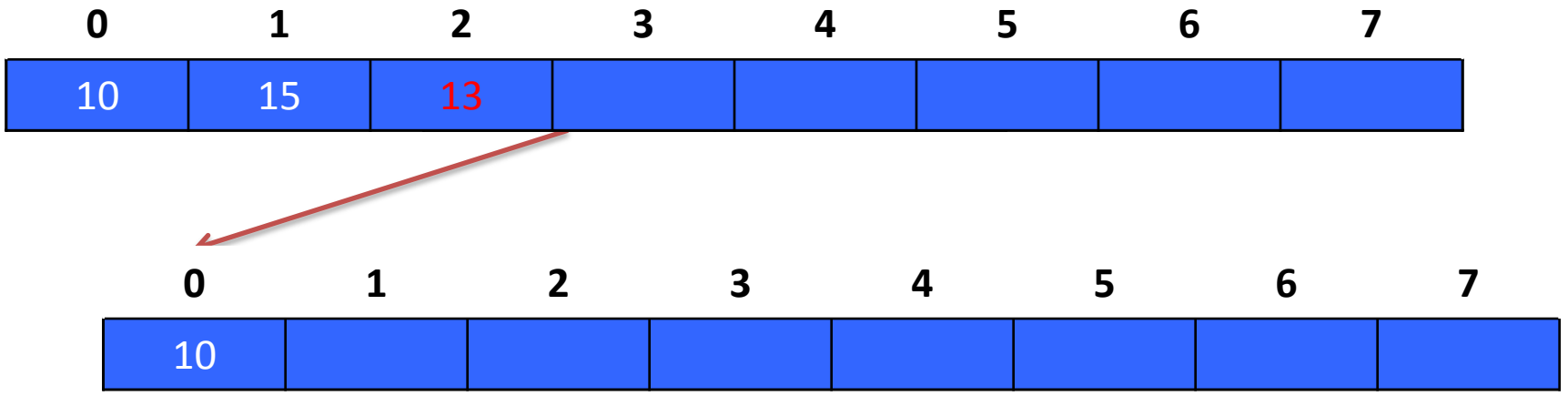


Array List

- Lösung: Einfügen von 8 Elementen

```
ArrayList<Integer> liste;
liste = new ArrayList<Integer>(2);
liste.add(10);
liste.ensureCapacity(8);
liste.add(15);
liste.add(13);
```

```
liste.add(22);
liste.add(17);
liste.add(28);
liste.add(31);
liste.add(20);
```



Array List

- Lösung: Einfügen von 8 Elementen

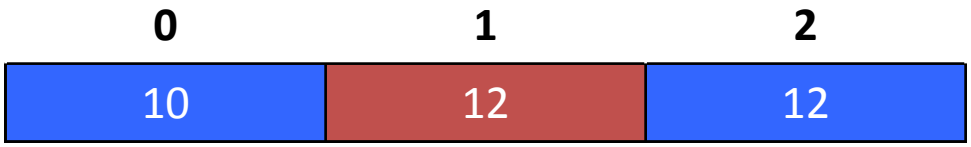
```
ArrayList<Integer> liste;
liste = new ArrayList<Integer>(8);
liste.add(10);
liste.add(15);
liste.add(13);
liste.add(22);
```

```
liste.add(22);
liste.add(17);
liste.add(28);
liste.add(31);
liste.add(20);
```

Array List

- Löschen von Elementen

```
ArrayList<Integer> liste;
liste = new ArrayList<Integer>();
liste.add(10);
liste.add(15);
liste.add(12);
liste.remove(1);
```



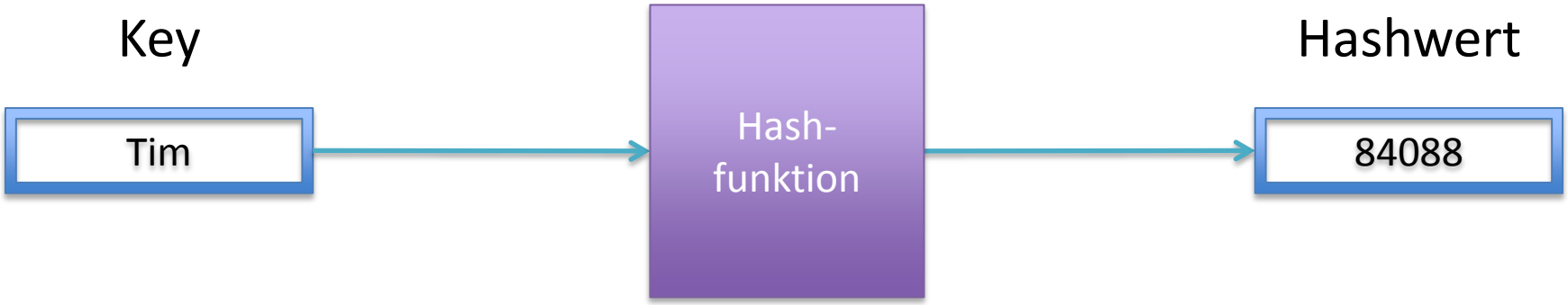
Array vs. ArrayList

- feste Länge
- primitive Datentypen
- weniger Funktionen
- ein einziger Datentyp

- variable Länge
- keine primitiven Datentypen
- mehr Funktionen
- mehrer Datentypen möglich

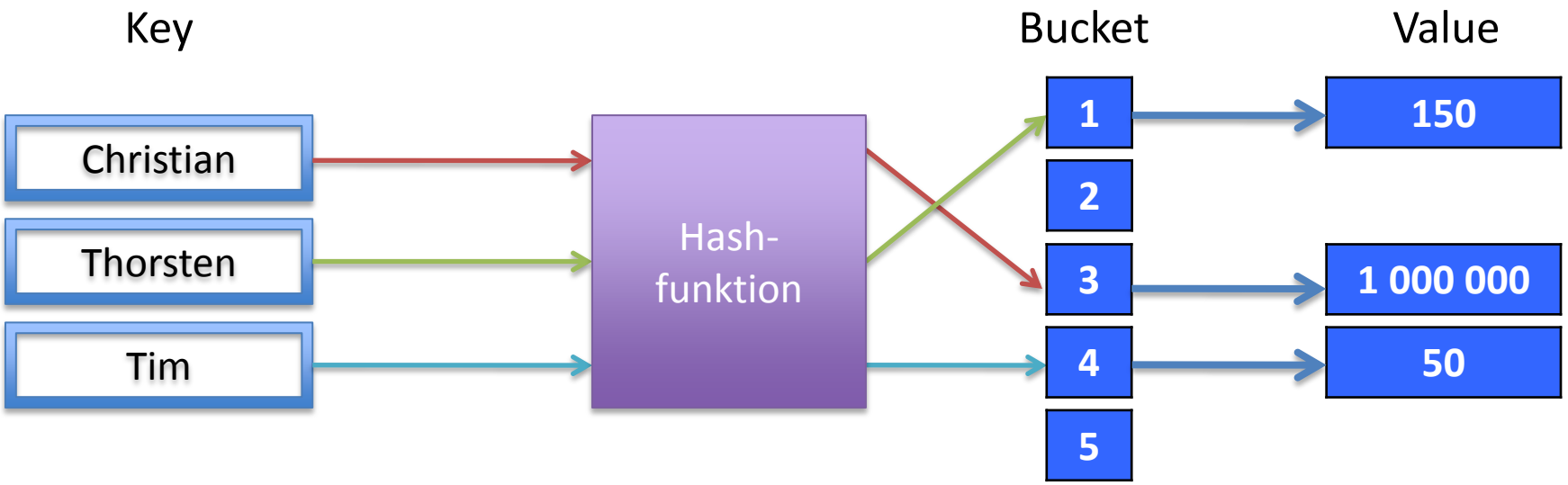
Hashing

- Hashverfahren
  - Algorithmus zum Suchen von Datenobjekten
  - Berechnung eines Hashwertes durch Hashfunktion
  - Hashwert fungiert als Index und „Fingerabdruck“
  - Jedes Objekt hat einen eigenen Hashwert



Hash Map

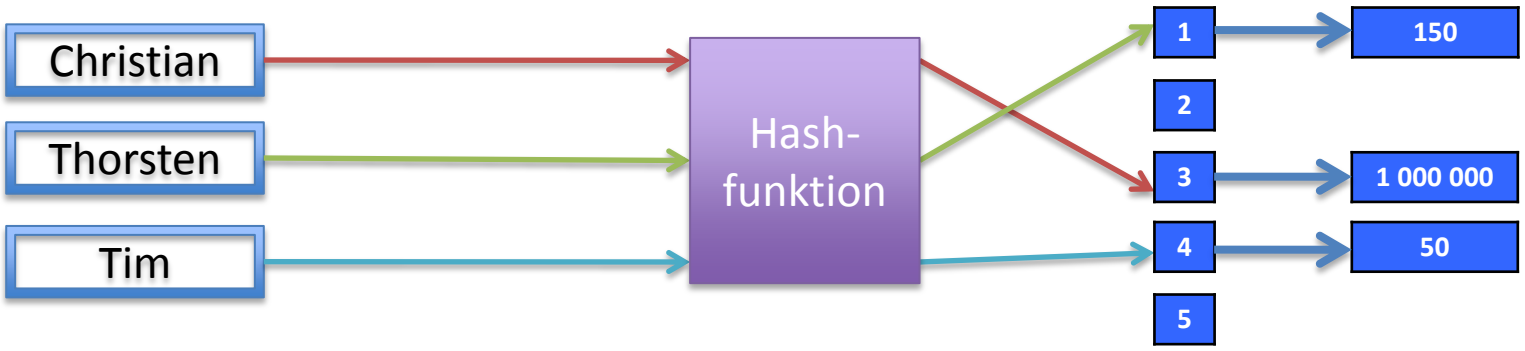
- HashMap<K,V>
  - Speichert Key-Value Paare
  - Der Wert wird unter dem Schlüssel abgelegt
  - Schlüssel müssen eindeutig sein
    - z.B. String, Integer (id)



Hash Map

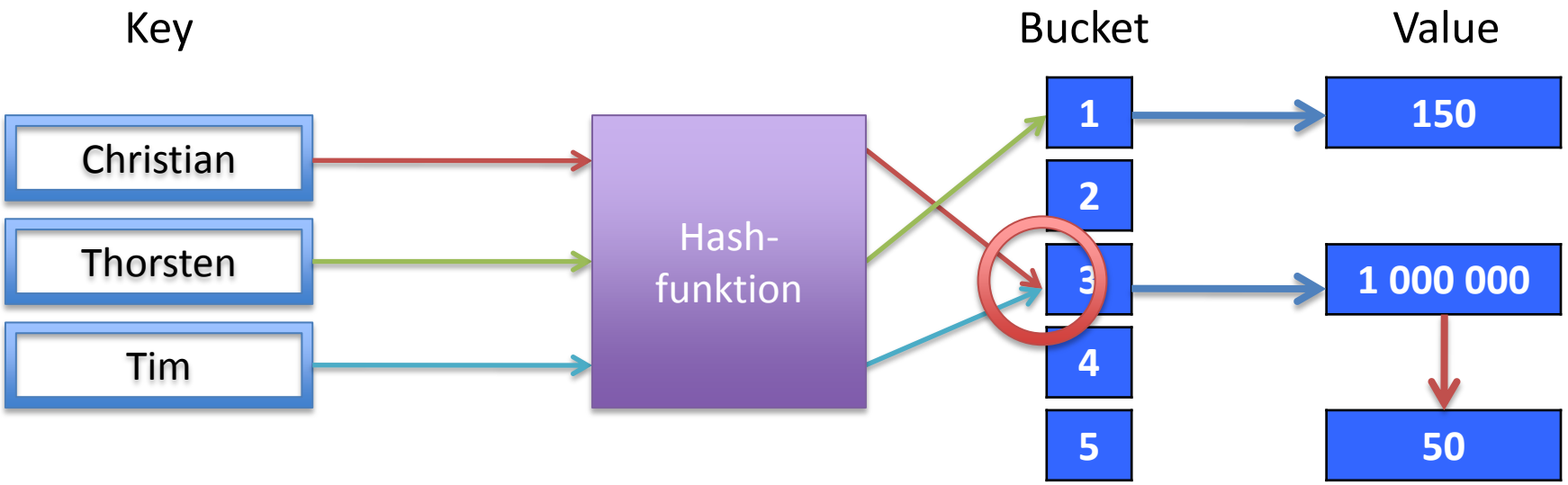
- Erstellen einer HashMap
  - K-V Typen festlegen
  - Paare einfügen mit `put(K,V)`

```
HashMap<String,Integer> kontos;
kontos = new HashMap<String,Integer>();
kontos.put("Christian",1000000);
kontos.put("Thorsten",150);
kontos.put("Tim",50);
```



Hash Map

- Kollision
  - Zwei unterschiedliche Schlüssel
  - Trotzdem selber Hashwert!
  - Kollisionsauflösung durch Verkettung



Hash Map

- Offenes Potential
  - Eigene Objekte als Value!
  - Bsp: Aktienobjekte mit Informationen
    - » Name, Kurs, Werte...

```
HashMap<String,Aktie> daxAktien;
daxAktien = new HashMap<String,Aktie>();
daxAktien.put("ADS.DE",adidas);
daxAktien.put("MEO.DE",metro);
daxAktien.put("HEI.DE",heidelbergCement);
...
```

Hash Map

- Objekte aufgepasst
  - Veränderungen wirken sich auf die Hash Map aus
  - Dateninkonsistenz

```
class MeinInt{
    int wert;
    public MeinInt(int i) {
        wert = i;
    }
}
```

```
MeinInt zahl = new MeinInt(13);
HashMap<String, MeinInt> map;
map = new HashMap<String, MeinInt>();
map.put("Tim", zahl);
zahl.wert = 42;
System.out.println(map.get("Tim").wert);
```

Hash Map

- Objekte aufgepasst
  - Bei Wrapperklassen unproblematisch
  - Neues Objekt anlegen!
    - » Hier durch Autoboxing

```
Integer zahl = new Integer(13);
HashMap<String, Integer> map;
map = new HashMap<String, Integer>();
map.put("Tim", zahl);
zahl = 42;
System.out.println(map.get("Tim").wert);
```

Hash Map

- Daten auslesen
  - Findet den Wert mit Hilfe des Schlüssels
  - Wert kann auch `null` sein
  - Niklas vorhanden ohne Konto?
    - » Nicht vorhanden?

```
konten.get("Tim");
konten.get("Niklas");
```

  - Konsolenausgabe

```
50
null
```

Hash Map

- Daten auslesen
  - `HashMap` erlaubt `null` als Schlüssel oder Wert
  - Unklar ob Eintrag vorhanden oder nicht

```
konten.containsKey(“Tim“);
konten.containsKey(“Niklas“);
```

  - Konsolenausgabe

```
true
false
```

Hash Map vs Hash Table

- Hash Table
  - `Synchronized`
  - Akzeptiert keine `null` Werte/Schlüssel

Hash Set

- Menge (eng. Set)
  - Jedes Element darf nur einmal vorkommen
  - Keine Methode zum Auslesen
    - Erweiterte for Schleife

```
HashSet<String> hSet = HashSet<String>();
System.out.println(hSet.add("Hallo"));
System.out.println(hSet.add("Hallo"));
```

  - Konsolenausgabe

```
true
false
```

Hash Set

- Mengenoperatoren
  - Vereinigung mit `addAll(Collection c)`
  - Schnittmenge mit `retainAll(Collection c)`
  - Teilmenge mit `containsAll(Collection c)`

```
menge1.add(1);
menge2.add(2);
menge2.add(1);
```

Hash Set

- Durchlaufen
  - Erweiterte for Schleife

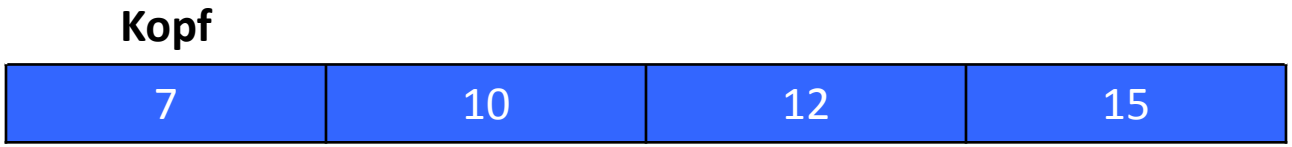
```
for(int i : hSet){
    System.out.println(i);
}
```

  - Durchläuft Objekte vom Typ `Iterable`
  - Vorsicht HashSet ist nicht sortiert!

Priority Queue

- Warteschlange
  - Sortiert
  - Generische Warteschlange

```
PriorityQueue<Integer> queue;
queue = new PriorityQueue<Integer>();
queue.add(10);
queue.add(15);
queue.add(12);
queue.add(7);
```



Priority Queue

- Vergleichen von Objekten
  - Comparable
    - » „Natürliche Sortierung“

```
PriorityQueue<Integer> queue;
queue = new PriorityQueue<Integer>();
```

  - Comparator
    - » Muss explizit angegeben werden

```
PriorityQueue<Integer> queue;
queue = new PriorityQueue<Integer>(11,myComparator);
```

Priority Queue

- Comparable
  - `compareTo(T o)` überschreiben
  - Objekt kann sich selbst mit anderen Objekten vergleichen
  - Umsetzung von nur einem Sortierkriterium
  - Muss in der selben Klasse sein

```
public int compareTo(MeinInt o) {
   if(this.wert < o.wert)
      return -1;
   else if(this.wert > o.wert)
      return 1;
   else
      return 0;
}
```

Priority Queue

- Comparator
  - `compare(T o1, T o2)` überschreiben
  - Objekt können verglichen werden
  - Mehrer Sortierkriterien möglich durch Extraklassen
  - Muss **nicht** in der selben Klasse sein!

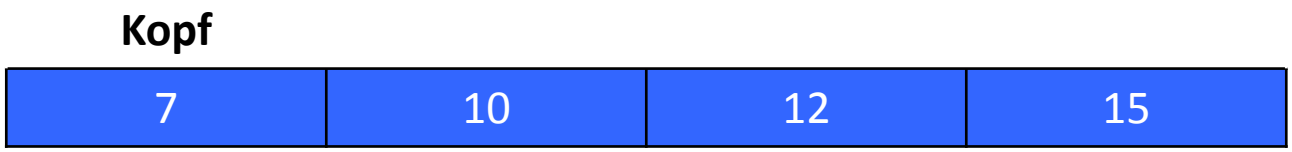
```
public int compare(MeinInt o1, MeinInt o2) {
   if(o1.wert < o2.wert)
      return -1;
   else if(o1.wert > o2.wert)
      return 1;
   else
      return 0;
}
```

Priority Queue

- Auf Elemente zugreifen
  - `peek()`
    - » Holt das oberste Element ohne es zu löschen
  - `poll()`
    - » Holt das oberste Element und löscht es

```
System.out.println(queue.peek());
System.out.println(queue.poll());
System.out.println(queue.poll());
```

```
7
7
10
```



Fragen?