CAMPUS 02 GRAZ
FACHHOCHSCHULE DER WIRTSCHAFT

SORTIERUNG VON DATEN

Bernhard Fuchs

basierend auf Folien von Christian Hofer

TERMINE

0004VD1004 PROGRAMMIEREN 2 (51UE IL, SS 2024/25)

Gruppe

Tag	Datum	von	bis	Ort	Ereignis	Termintyp	Lerneinheit	Vortragende*r	Anmerkung
Standardgruppe									
Fr	21.03.2025	08:15	12:15	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Do	27.03.2025	08:15	15:30	CZ004	Abhaltung	fix		Jandl, Silke	
Di	01.04.2025	08:15	16:00	CZ004	Abhaltung	fix		Jandl, Silke	
Mo	07.04.2025	12:30	16:00	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Di	08.04.2025	13:30	15:30	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Do	10.04.2025	08:15	16:00	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Mo	28.04.2025	12:30	16:00	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Mi	30.04.2025	08:15	11:45	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Mi	30.04.2025	12:30	16:00	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Mi	14.05.2025	08:15	16:00	CZ004	Abhaltung	fix		Fuchs, Bernhard, Dipl.-Ing.	
Mo	26.05.2025	08:15	12:15	CZ004	Prüfungstermin	fix			Teilergebnisse: "Klausurarbeit"

https://interface.campus02.at/lv/detail?type=all&lang=de&uniqueUuid=eb872862-db8d-45c4-ba96-76a582df703e;dac5c0f5-4884-4f6e-be31-453a58eed22d;1

INHALTE & ZIELE

- Sortieren von Daten
- Daten die in Arrays od. Collections vorliegen nach unterschiedlichen Kriterien sortieren können
  - Unterschied zw. den Interfaces Comparable und Comparator erklären können
  - Comparable und Comparator nach diversen Vorgaben eigenständig implementieren und anwenden können

SORTIEREN VON ARRAYS

- Arrays:
  - um Daten zu sortieren die in Arrays vorliegen bietet Java die “Hilfsklasse” **java.util.Arrays**
  - ja nach Datentyp gibt es entsprechende Überladungen einer Methode zum Sortieren **static void sort(type[] a)**
  - type: byte, char, double, float, short, int, long, Object
- **WICHTIG**: kein Rückgabewert d.h. sortiert direkt das übergebene Array

SORTIEREN VON ARRAYS

- **Arrays.sort(type[] a) Algorithmen**
  - verwendet für primitive Typen eine **spezielle Variante von QuickSort**
  - verwendet für Object[] **eine spezielle Kombination von MergeSort und InsertionSort**

⇒Details zu den Algorithmen QuickSort und MergeSort werden in anderen LVs vorgestellt

SORTIEREN VON ARRAYS

- Arrays.sort(type[] a)
  - die **Reihenfolge** für die Sortierung ergibt sich aus der sog. "**natürlichen Ordnung**" des jeweiligen Typ
  - bei numerischen Typen in aufsteigender numerischer Reihenfolgen z.B. int[] {3,1,9,5} -> {1,3,5,9}
  - bei char[] aufsteigend nach Ascii Tabelleneintrag

Was passiert bei Object[] a => Welche natürliche Ordnung macht hier Sinn?

SORTIEREN VON ARRAYS

- Arrays.sort(Object[] a)
  - wollen wir z.B. ein String[] sortieren lassen funktioniert das erwartungsgemäß
  - übergeben wir jedoch einen **selbst erstellten Referenztyp** z.B. ein Person[] wird eine **ClassCastException** ausgelöst – Warum?

=> Java signalisiert damit, dass **keine Möglichkeit besteht "ohne Zusatzinformation" die Sortierung durchzuführen**!

SORTIEREN VON ARRAYS

- **aus der Java API Dokumentation zu**

```
public static void sort(Object[] a)
```

Sorts the specified array of objects **into ascending order, according to the natural ordering of its elements. All elements in the array must implement the Comparable interface. …**

COMPARABLE

- **Java API definiert generisches Comparable Interface (= eine „Schnittstelle“)**
  - dieses Interface sollte bei Klassen implementiert werden, für deren Instanzen Vergleiche hinsichtlich einer Ordnungsrelation benötigt werden

```
public interface Comparable <T> {
   public int compareTo(T o);
}
```

COMPARABLE INTERFACE

- **compareTo Methode von Comparable**
  - nimmt Instanz des entsprechenden Typs entgegen und macht zwischen dem **aktuellen (this) und dem übergebenen Objekt (other) den Vergleich**
  - Methode **liefert negativen Wert (< 0) wenn aktuelles Objekt vorher in Sortierung sein soll**
  - Methode **liefert positiven Wert (> 0) wenn aktuelles Objekt nachher in Sortierung sein soll**
  - Methode **liefert 0 zurück, sofern beide Objekte gleichwertig bezogen auf Sortierung sind**

COMPARABLE INTERFACE

- **compareTo Methode von Comparable**

```
public class Person implements Comparable<Person> {
    private int id;
    private String firstName;
    private String lastName;

    @Override
    public int compareTo(Person o) {
        //NOTE: ascending according to id
        if{this.id < o.id) return -1;
        if(this.id > o.id) return 1;
        return 0;
    }
}
```

SORTIEREN VON ARRAYS

```
Person[] people =
   {new Person(4321,"Max", „Mustermann"),
     new Person(3456,"Silvia", „Musterfrau");


    //uses compareTo Method of Comparable Interface
   Arrays.sort(people);
    for(Person p : people) {
          System.out.printl(p);
   }
```

SORTIEREN VON COLLECTIONS

- Collections
  - um Daten zu sortieren die in Collections vorliegen bietet Java die “Hilfsklasse” java.util.Collections
  - es gibt dazu eine generische Methode zum Sortieren von Listen welche Referenztypen beinhalten

    static void sort(List<T> list)

    WICHTIG: kein Rückgabewert d.h. sortiert direkt die übergebene Collection

SORTIEREN VON COLLECTIONS

- **Sortieren mit Comparable**

```
List<Person> people = new ArrayList<>();
people.add(new Person(4321,"Max",„Mustermann")),
people.add(new Person(3456,"Silvia","Musterfrau"));

//uses compareTo Method of Comparable Interface
Collections.sort(people);

for(Person p: people) {
    System.out.println(p);
}
```

COMPARABLE INTERFACE

- **equals() und hashCode()**
  …kein Zwang aber EMPFEHLUNG!
  - die von Object geerbten **equals() sowie hashCode() sollten ebenso implementiert bzw. überschrieben werden,** falls eine Klasse Comparable<T> implementiert
  - **auf Konsistenz mit compareTo Methode achten**

```
equals==true  <==> compareTo==0
equals==false <==> compareTo!=0
```

BEISPIEL: HASEN

- Hasen sollen Alter halten und nach dem Alter aufsteigend sortiert werden
- Kleines Testprogramm dazu:
  - Ein paar Hasen in einer ArrayList sortieren
  - Lässt sich damit auch ein Osterhase sortieren?

HASEN SORTIEREN ERWEITERN

- Wir möchten nach zwei Kriterien sortieren
  - Zuerst nach Alter und als zweites Kriterium nach Anzahl der Karotten
- Testen
  - In ArrayList sortieren

COMPARABLE INTERFACE

- **Problem:** Für eine Klassen werden unterschiedliche Sortierreihenfolgen benötigt
  - Wir möchten Hasen manchmal
    - nach Alter und Karotten sortieren
    - wie viel Urlaubstage sie haben
    - (oder anderen Kriterien die uns letztens eingefallen sind)
- Das Interface Comparable können wir in einer Klasse jedoch nur einmal implementieren und damit ist die gewünschte Sortierreihenfolge vorgegeben…

STATTDESSEN: COMPARATOR

- Lösung:
  - die “Hilfsklassen“ *java.util.Arrays* sowie *java.util.Collections* bieten **Überladungen der sort Methoden** an welche einen **Comparator** zur Sortierung übernehmen

=> durch geeignete Comparator Klassen lassen sich Arrays als auch Collections beliebig sortieren

COMPARATOR INTERFACE

- Java API definiert generisches Comparator Interface
  - Comparator Klassen implementieren dieses Interface und können dann für benutzerdefinierte Sortierreihenfolgen verwendet werden

```
public interface Comparator<T> {
    public int compare(T o1, T o2);
}
```

COMPARATOR INTERFACE

- **compare Methode von Comparator**
  - **nimmt zwei Instanzen des entsprechenden Typs** entgegen und **macht den Vergleich**
  - Methode **liefert negativen Wert (< 0) wenn Objekt 1 vorher in Sortierung sein soll**
  - Methode **liefert positiven Wert (> 0) wenn Objekt 1 nachher in Sortierung sein soll**
  - Methode **liefert 0 zurück, sofern beide Objekte gleichwertig bezogen auf Sortierung sind**

COMPARATOR PERSON

- **Bsp. Personen nach Id sortieren**

```
public class IdComparatorASC
   implements Comparator<Person> {
   @Override
   public int compare(Person o1, Person o2) {
        //NOTE: ascending according to id
        if(o1.getId() < o2.getId()) return -1;
        if(o1.getId() > o2.getId()) return 1;
        return 0;
   }
}
```

COMPARATOR PERSON

- **Bsp. Personen nach Vornamen sortieren**

```
public class FirstNameComparator
   implements Comparator<Person> {
   @Override
   public int compare(Person o1, Person 02) {
    //NOTE: ascending according to id
        return o1.getFirstName()
       .compareTo(o2.getFirstName());
   }
}
```

SORTIEREN VON COLLECTIONS

- **Sortieren mit Comparator**

```
List<Person> people = new ArrayList<>();
people.add(new Person(4321,"Max","Mustermann")),
people.add(new Person(3456,"Silvia","Musterfrau"));

//uses compareTo Method of Comparable Interface
Collections.sort(people, new IdComparatorAsc());

for(Person p: people) {
    System.out.println(p);
}
```

- **Analoge Verwendung bei Arrays**

HASEN SORTIEREN ERWEITERN

- Comparator um nach Urlaubstagen (absteigend) zu sortieren

ANONYMOUS COMPARATOR (FALLS NOCH ZEIT IST)

- **kürzere Schreibweise durch Comparator als anonyme innere Klasse (falls wir wirklich nur einmal brauchen)**

```
//ordering according to comparator
//given as anonymous inner class
//using last name
Collections.sort(lp, new Comparator<Person>() {
 @override
 public int compare(Person o1, Person o2) {
        return o1.getLastName()
        .compareTo(o2.getLastName());
    }
});
```