

# Informatik II

## Übung 5

FS 2019

# Heutiges Programm

- 1 Feedback letzte Übung
- 2 Wiederholung Vorlesung
- 3 Zur aktuellen Übung

# 1. Feedback letzte Übung

# Umweltdaten – Zahleneingabe

```
def inputNumber(prompt, minIncl, maxIncl):  
    """Read a number in the interval [minIncl..maxIncl]"""  
    while True:  
        try:  
            str = input(prompt)  
            index = int(str)  
            if index in range(minIncl,maxIncl + 1):  
                return index  
        except ValueError:  
            pass
```

# Umweltdaten – CSV Datei

```
for i, v in enumerate(values):  
    value = float(v)  
    if not.isnan(value):  
        if units[i] == '' and not.isnan(value):  
            value = int(value) # convert to int if unit is '1'  
        data[stations[i]][datum][fields[i]] = value
```

## **2. Wiederholung Vorlesung**

# Beispiel: Lesen vom Scanner

```
Scanner scanner = new Scanner(System.in);
int number;
while (scanner.hasNextInt()) {
    number = scanner.nextInt();
    ...
}
```

# Arten von Ausnahmen

## Systemausnahmen

(runtime exceptions)

- Können überall auftreten
- *Können* behandelt werden
- Ursache: Bug im Programm

## Benutzerausnahmen

(checked exceptions)

- Müssen deklariert werden
- *Müssen* behandelt werden
- Ursache: Unwahrscheinliches, aber prinzipiell mögliches Ereignis



# Beispiel einer Benutzerausnahme

```
private static String[] readFile(String filename){  
    FileReader fr = new FileReader(filename);  
    BufferedReader bufr = new BufferedReader(fr);  
    ...  
    line = bufr.readLine();  
    ...  
}
```

# Beispiel einer Benutzerausnahme

```
private static String[] readFile(String filename){
    FileReader fr = new FileReader(filename);
    BufferedReader bufr = new BufferedReader(fr);
    ...
    line = bufr.readLine();
    ...
}
```

## *Compiler* Fehler:

```
./Root/Main.java:9: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
    FileReader fr = new FileReader(filename);
                    ^
```

```
./Root/Main.java:11: error: unreported exception IOException; must be caught or declared to be thrown
    String line = bufr.readLine();
                    ^
```

# Behandeln von Ausnahmen

```
private static String[] readFile(String filename){
    try{
        FileReader fr = new FileReader(filename);
        BufferedReader bufr = new BufferedReader(fr);
        ...
        line = bufr.readLine();
        ...
    } catch (IOException e){
        // do some recovery handling
    } finally {
        // close resources
    }
}
```

*Geschützter Bereich*

*Massnahmen zur Wiederherstellung der Normalsituation*

*Wird in jedem Fall am Schluss ausgeführt, immer!*

# Try-With-Resources Anweisung

Spezifische Syntax, um Ressourcen *automatisch* zu schliessen:

```
private static String[] readFile(String filename){  
    try (  
        FileReader fr = new FileReader(filename);  
        BufferedReader bufr = new BufferedReader(fr)) {  
        ...  
        line = bufr.readLine();  
        ...  
    } catch (IOException e){  
        // do some recovery handling  
    }  
}
```

*Resources werden hier geöffnet*

*Resources werden hier automatisch geschlossen*

## **3. Zur aktuellen Übung**

# Generisches Sortieren mit Java

```
public class Fruit {  
    private String name;           // Attributes  
    private int taste;  
    private int calories;  
    Fruit(String name, int taste, int calories) {  
        this.name = name;        // Constructor  
        this.taste = taste;  
        this.calories = calories;  
    }  
    public String getName() {      // Name Getter  
        return name;  
    }  
    public int getTaste() {        // Taste Getter  
        return taste;  
    }  
    ...  
}
```

# Generisches Sortieren mit Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collections;
public class Main {
    public static void main(String[] args) {
        List<Fruit> fruits = new ArrayList<Fruit>();
        fruits.add(new Fruit("apple", 5, 52));
        fruits.add(new Fruit("pear", 5, 57));
        fruits.add(new Fruit("banana", 6, 89));
        fruits.add(new Fruit("kiwi", 6, 61));
        Collections.sort(fruits);
        for (Fruit fruit : fruits)
            System.out.println(fruit.getName());
    }
}
```

# Generisches Sortieren mit Java

```
import java.util.ArrayList;
import java.util.List;
import java.util.Collections;
public class Main {
    public static void main(String[] args) {
        List<Fruit> fruits = new ArrayList<Fruit>();
        fruits.add(new Fruit("apple", 5, 52));
        fruits.add(new Fruit("pear", 5, 57));
        fruits.add(new Fruit("banana", 6, 89));
        fruits.add(new Fruit("kiwi", 6, 61));
        Wie sortieren? → Collections.sort(fruits);
        for (Fruit fruit : fruits)
            System.out.println(fruit.getName());
    }
}
```



# Generisches Sortieren: Comparable Interface

```
public class Fruit implements Comparable<Fruit> {  
    private String name;           // Attributes  
    private int taste;  
    private int calories;  
    Fruit(String name, int taste, int calories) {  
        this.name = name;         // Constructor  
        this.taste = taste;  
        this.calories = calories;  
    }  
    public String getName() {      // Name Getter  
        return name;  
    }  
    public int getTaste() {        // Taste Getter  
        return taste;  
    }  
    ...  
}
```

# Generisches Sortieren: Comparison Method

```
// Comparison Method for Comparable Interface:
public int compareTo(Fruit that) {
    int BEFORE = -1; int EQUAL = 0; int AFTER = 1;
    // Check if same fruit
    if (this == that)
        return EQUAL;
    // DESCENDING by Taste
    if (this.getTaste() > that.getTaste())
        return BEFORE;
    else if (this.getTaste() < that.getTaste())
        return AFTER;
    else // Same Taste -> ASCENDING by Calories
        return this.getCalories()-that.getCalories();
}
```

# Generisches Sortieren: Comparison Method

```
// Comparison Method for Comparable Interface:  
public int compareTo(Fruit that) {  
    int BEFORE = -1; int EQUAL = 0; int AFTER = 1;  
    // Check if same fruit  
    if (this == that)  
        return EQUAL;  
    // DESCENDING by Taste  
    if (this.getTaste() > that.getTaste())  
        return BEFORE;  
    else if (this.getTaste() < that.getTaste())  
        return AFTER;  
    else // Same Taste -> ASCENDING by Calories  
        return this.getCalories() - that.getCalories();  
}
```

*negative Zahl*  
⇔ "this < that"

*positive Zahl*  
⇔ "this > that"

Fragen oder Anregungen?