Informatik II

Übung 5

FS 2019

Program Today

1. Feedback of last exercise
2. Repetition Lectures
3. Next Exercise

1. Feedback of last exercise

Environmental Data – Number Input

```
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
```

Environmental Data – CSV File

```
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. Repetition Lectures

Example: Reading from the Scanner

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

Exception Types

Runtime Exceptions

- Can happen anywhere
- *Can* be handled
- Cause: bug in the code

Checked Exceptions

- Must be declared
- *Must* be handled
- Cause: Unlikely but not impossible event

Example of a Checked Exception

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

Example of a Checked Exception

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

Compiler Error:

```
./Root/Main.java:9: error: unreported exception FileNotFoundException; must be caught or declared to be
    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();
                         ^
```

Handling Exceptions

```
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
    }
}
```

Protected scope

Measures to re-establis the normal situation

Gets executed in any case, at the end, always!

Try-With-Resources Statement

Specific syntax to close resources *automatically*:

```
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 get opened here

Resources get closed automatically here

3. Next Exercise

Generic Sorting with 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;
        ...
```

Generic Sorting with 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());
	}
}
```

Generic Sorting with 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));
Sort criteria? →Collections.sort(fruits);
                for (Fruit fruit : fruits)
                        System.out.println(fruit.getName());
        }
}
```

Generic Sorting: 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;
        ...
```

Generic Sorting: 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();
}
```

Generic Sorting: Comparison Method

negative value
⇔ "this < that"

positive value
⇔ "this > that"

```
// 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();
}
```

Questions / Suggestions?