Praktikum Programmieren

Aufgabenblatt 3 - Entwurf

B-AI2 PMP SS 2018

Adrian Helberg, Rodrigo Ehlers , Gruppe 2

Prüfer: Prof. Dr. Bernd Kahlbrandt

4. Mai 2018

Datum und Erzeugen von Streams

Funktionale Interfaces

Verarbeitung von Streams

Datum und Erzeugen von Streams

Erster Mai: Typ *LocalDate*, da

- keine Zeitzoneninformation gebraucht wird
- Schaltjahre implementiert sind (*"[...] today's rules for leap years are applied for all time"*) [1]

[1] Java Dokumentation LocalDate

Datum und Erzeugen von Streams

Erzeugen eines Streams mittels *iterate* mit Begrenzung und Filter:

```
List<Integer> dates = Stream
    .iterate(workers_day, d -> d.plusYears(1))

    .limit(ChronoUnit.YEARS.between(workers_day, until))

    .filter(d -> d.getDayOfWeek()
        .toString().equals("TUESDAY"))

    .mapToInt(LocalDate::getYear)

    .boxed() // Primitive stream

    .collect(Collectors.toList());
```

Datum und Erzeugen von Streams

Ermitteln von Abständen und formatierte Ausgabe

```
public String getFormattedDistanceString() {

      return "Days between the two MAY 1st: "
         + DAYS.between(first, second)
         + "";

}
```

Datum und Erzeugen von Streams

Nächster Sonntag, nächstes Schaltjahr, Datum 42 Tage später:

```
final LocalDate date = LocalDate.of(2018, Month.MAY, 1);
final LocalDate nextSunday = date
    .with(next(DayOfWeek.SUNDAY));

final int nextLeapYear = nextLeapYear(date);

final LocalDate after42Days = date.plusDays(42);
```

Funktionale Interfaces

Variable Anzahl von *Predicates* entgegennehmen und mit logischem *AND* verknüpfen:

```
public static <T> Predicate<T> andAll
                        (Predicate<T>[] predicates) {

    return Arrays
        .stream(predicates)
        .reduce(x -> true, Predicate::and);

}
```

Funktionale Interfaces

Variable Anzahl von *Predicates* entgegennehmen und mit logischem *OR* verknüpfen:

```
public static <T> Predicate<T> orAny
                            (Predicate<T>[] predicates) {

    return Arrays
        .stream(predicates)
        .reduce(x -> false, Predicate::or);

}
```

Funktionale Interfaces

Test *andAll*:

```
@Test
public void andAllTest() {

    List<String> values = new ArrayList<>();
    values.add("Automat");
    values.add("Baum");
    values.add("Auto");
    values.add("Laut");

    Predicate<String>[] predicates = new Predicate[3];
    predicates[0] = c -> c.startsWith("A");
    predicates[1] = c -> c.length() == 4;
    predicates[2] = c -> c.contains("o");

    [...]
```

Funktionale Interfaces

Test *andAll*:

```
    [...]

    Predicate<String> p = FormatUtils.andAll(predicates);

    List<String> filteredList = values
        .stream()
        .filter(p)
        .collect(Collectors.toList());

    List<String> expected = new ArrayList<>();
    expected.add("Auto");

    assertEquals(expected, filteredList);

}
```

Funktionale Interfaces

Test *orAny*:

```
@Test
public void orAnyTest() {

    List<String> values = new ArrayList<>();
    values.add("Automat");
    values.add("Baum");
    values.add("Auto");
    values.add("Laut");

    Predicate<String>[] predicates = new Predicate[3];
    predicates[0] = c -> c.startsWith("A");
    predicates[1] = c -> c.length() == 4;
    predicates[2] = c -> c.contains("o");

    [...]
```

Funktionale Interfaces

Test *andAll*:

```
    [...]

    Predicate<String> p = FormatUtils.orAny(predicates);

    List<String> filteredList = values
        .stream()
        .filter(p)
        .collect(Collectors.toList());

    List<String> expected = new ArrayList<>();
    expected.add("Automat");
    expected.add("Baum");
    expected.add("Auto");
    expected.add("Laut");

    assertEquals(expected, filteredList);

}
```

Verarbeitung von Streams

Alle Palindrome mit sechs Buchstaben:

```
List<String> palindromes = Files

    .lines(Paths.get("scrabble.txt"))

    .filter(s ->
        s.length() == 6 &&
        s.equals(new StringBuilder(s)
            .reverse().toString()))

    .collect(Collectors.toList());
```

Verarbeitung von Streams

alle Worte, die an der 2., 4. und 6. Stelle ein “e” haben:

```
List<String> eee = Files

    .lines(Paths.get("scrabble.txt"))

    .filter(s ->
        s.length() > 5 &&
        s.charAt(1) == 'e' &&
        s.charAt(3) == 'e' &&
        s.charAt(5) == 'e')

    .collect(Collectors.toList());
```

Verarbeitung von Streams

alle Worte, die Paare von Vokalen enthalten:

```
List<String> vowels = Files
    .lines(Paths.get("scrabble.txt"))

    .filter(s ->
        s.contains("aa") ||
        s.contains("ee") ||
        s.contains("ii") ||
        s.contains("oo") ||
        s.contains("uu"))

    .collect(Collectors.toList());
```

Verarbeitung von Streams

Worte, die in beiden Dateien vorkommen:

```
List<String> sgb = Files
    .lines(Paths.get("sgb-words.txt"))
    .collect(Collectors.toList());
List<String> common = Files
    .lines(Paths.get("scrabble.txt"))
    .filter(sgb::contains)
    .collect(Collectors.toList());
```