*Praktikum Programmieren*

# Aufgabenblatt 2 - Entwurf

B-AI2 PMP SS 2018

Adrian Helberg, Gruppe 2

**Prüfer: Prof. Dr. Bernd Kahlbrandt**

17. April 2018

Listen und IO

Operatoren

# Listen und IO

Um die 1000 komplexen Zahlen nach vorgegebenen Regeln zu erstellen und in eine Textdatei zu schreiben gibt es folgende Strukturen:

- Einen *PrintWriter*, der den Namen der Textdatei entgegennimmt
- Ein Array der Größe 1000, welches die komplexen Zahlen hält
- Eine *for*-Kontrollstruktur, welche das deklarierte Array iteriert und mit komplexen Zahlen initialisiert
- Eine Anweisung *writer.println(...)*, welche die angegebene Datei zeilenweise beschreibt

# Listen und IO

Um komplexe Zahlen aus einer Textdatei zu lesen und zu speichern, gibt es folgende Strukturen:

- Einen *BufferedReader*, der die angegebene Textdatei liest
- Ein Array der Größe 1000, welches die komplexen Zahlen hält
- Eine Liste, welche die komplexen Zahlen hält
- Eine *while*-Kontrollstruktur, welche die Textdatei zeilenweise durchläuft
- Zwei Zuweisungen, welche die resultierenden, komplexen Zahlen in das Array und die Liste schreiben

# Listen und IO

Das Sortieren des Arrays bzw. der Liste wird über die Hilfsklasse *MathUtils* umgesetzt:

```
public static Complex[] sortByLength(Complex[] n);

public static List<Complex> sortByLength(List<Complex> n);
```

Um sie Sortierung zu ermöglichen wird eine interne Klasse *Tupel* verwendet, welche die Funktion *compareTo(...)* des Interfaces *Comparable<Tuple>* überschreibt:

```
public int compareTo(Tuple t) {
    return this.length < t.length
        ? -1 : (this.length > t.length
            ? 1 : 0);
}
```

# Operatoren

```
public static boolean isOdd(long i) {
    return i % 2 == 1;
}
```

- Ist die Methode korrekt?
  - *Nein*
- Wenn ja, warum, wenn nicht, warum?
  - *Sollte eine negative Zahl überprüft werden, liefert die Funktion ein falsches Ergebnis*[1]
- Was liefert sie und wie macht man das richtig?
  - Die Funktion liefert für *isOdd(-1L)* den Wert *false*.

```
public static boolean isOdd(long i) {
    return (i % 2 + 2) % 2 == 1;
}
```

[1] Das Ergebnis der Restoperation ist negativ, wenn die Division negativ, und positiv, wenn die Division positiv ist

# Operatoren

```
System.out.println((int) (char) (byte) -1);
```

- Welches Ergebnis erwarten Sie?
  - *Erwartet wird hier -1*
- Welches kommt tatsächlich heraus?
  - *Der Ausdruck liefert 65535*
- Was passiert genau?
  - *Zuerst wird das Byte "-1" via widening primitive conversion (§5.1.2), und dann der resultierende int via narrowing primitive conversion (§5.1.3) konvertiert. Anschließend wird das resultierende "?" in int konvertiert und ergibt 65535, da "?" der Hexzahl "FFFFFFFF" entspricht und damit des Maximalwerts des 16 bit integers*
  - siehe *Widening and Narrowing Primitive Conversion*[2]

[2] https://docs.oracle.com/javase/specs/jls/se8/html/jls-5.html

# Operatoren

```
int x = 1984;
int y = 2001;
x ^= y ^= x ^ y;
System.out.println("x = " + x + "; y = " + y);
```

- Was wird ausgegeben?
  - *x = 0; y = 1984*
  - Der Ausdruck wird von rechts nach links evaluiert

# Operatoren

Schritt 1:

```
11111000000      1984
11111010001      2001
-------------- XOR
00000010001        17
```

Schritt 2:

```
11111010001      2001
00000010001        17
-------------- XOR
11111000000      1984  (= y)
```

Schritt 3:

```
11111000000      1984
11111000000      1984
-------------- XOR
00000000000         0  (= x)
```

# Operatoren

```
public class Test {
    public static void main(String[] args) {
        System.out.print("Hell");
        System.out.println("o world");
    }
}
```

- Was schreibt das Programm auf die Konsole?
  - *Hello world*
  - *System.out.print()* schreibt in die Konsole
  - *System.out.println()* beendet nach dem Schreiben die aktuelle Zeile mit einem Zeilenumbruchzeichen (Betriebssystemabhängig)

# Operatoren

```
1 public class Increment {
2     public static void main(String[] args) {
3         int j = 0;
4         for (int i = 0; i < 100; i++)
5             j = j++;
6         System.out.println(j);
7     }
8 }
```

- Was gibt der folgende Code auf der Console aus?
  - *0*
  - Die Inkrementierung durch die "'++"-Operation wird erst nach der Zuweisungsoperation durchgeführt
  - Ersetzt man Zeile 5 mit *j = ++j;* funktioniert die gewollte Inkrementierung und der Code schreibt *100* in die Konsole