PR2 – Formular für Lesenotizen

SS2021

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	14.05.21

7.h) Streams zur Anwendung von Operationen

- Formulierung eines Problems als Folge von Daten und Verarbeitungsoperationen darauf.

➔ Die Verarbeitungsoperationen können so auf mehrere Kerne verteilt werden.

- *Mechanismus, um mehrere Programme über Ein-/Ausgaben zu verketten*
- *Die Programme können dabei parallel laufen.*

eine Folge von Daten Ströme werden von Methoden verarbeitet und weiter gereicht.

➔ Die Methoden heißen hier auch Operationen.

Aufbau eines Streams

Initiale Operation: erzeugt den Strom

– Z. B. eine Folge von Zahlen or – Z. B. eine Collection:

```
List<String> c= Arrays.asList("Streams", "rock"); c.stream() // liefert String-Stream
```

Intermediäre Operation: transformiert den Strom und liefert wieder einen Strom

– Z. B. filter or – Z. B. map:

```
c.stream().map(s -> s.charAt(0)) // liefert char-Stream
```

Terminale Operation: liefert keinen Strom, sondern einen einfachen Wert oder Seiteneffekt

– Z. B. count or – Z. B. forEach:

```
c.stream().map(s->s.charAt(0)) .forEach(c->System.out.print(c)); // prints Sr
```

- Erst wenn die terminale Operation ausgeführt wird, werden die Elemente des davor definierten Streams Schritt für Schritt erzeugt und durchlaufen.
- Wenn die terminale Operation zurückkehrt, ist der Stream „verbraucht“.
- **Das „Schritt für Schritt“ kann auch parallel auf mehrere Prozessoren verteilt erfolgen, wenn man den Stream explizit als parallel markiert:**

```
public static long countFactorsStream(long n) {
  long cnt= LongStream.rangeClosed(1, n/2)
                                      .parallel()
                                      .filter( i -> isFactor(i, n) )
                                      .count();
  return cnt+1; }          //ermittelt alle teiler mittel der fnk isFactor boolean
```

```
int m= IntStream.rangeClosed(1,4).map(n->n*n).sum(); // 1+4+9+16=30
```

```
 int n= 7;
int fakultaet= IntStream.rangeClosed(2, n) .reduce(1, (a,b) -> a*b);
                                                  //Initialwert 1
             //Produkt-Verknüpfung des Initialwerts mit jedem weiteren Datenelement
```

```
List<String> list= Arrays.asList("Berti", "Conni", "Edi"); boolean b=
list.stream().allMatch(s->s.endsWith("i")); // true
 int m= list.stream()                    //Enden alle Elemente mit dem Buchstaben „i“?
             .mapToInt(String::length)
             .reduce(Integer::MIN_VALUE, Integer::max); // 5
             //Länge des längsten Strings
```

```
 /// teilaufgabe a
 class DoorBell implements ChangeListener {
      public void changed (ObservableProperty observable) {
         System.out.println(observable.getValue());
      }
 }// teilaufgabe b
 ChangeListener b = new ChangeListener() {
      public void changed (ObservableProperty observable) {
          System.out.println(observable.getValue());
      }
 } DoorBell b = new DoorBell();
//c
 ChangeListener b = (observable) -> System.out.println(observable.getValue());
```

```
public abstract class Medium implements Comparable<Medium> {

@Override
public int compareTo(Medium other) {
 return
Integer.valueOf(getjahr()).compareTo((Integer)other.getjahr());
}
```

```
@Override
public int compareTo(Medium other) {
    if (this.getjahr () > other.getErsjahr()) {
        return 1;
    } else if ( this.getjahr() < other.getjahr()) {
        return -1;
    } else {
        return 0;
    }
}
```

PR2 – Formular für Lesenotizen

SS2021

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	14.05.21

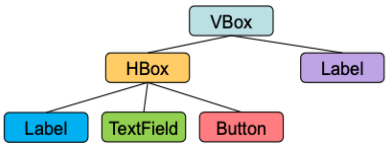
8. GUI – Graphische Benutzeroberflächen mit JavaFX

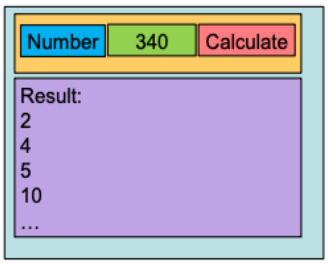
GUI Designer/GUI Builder: Entwicklungswerkzeug für die Zusammenstellung einer Benutzeroberfläche mittels Drag & Drop.

Ereignis(event): Zustandsänderung in der Außenwelt des Programms, z. B. Mausbewegung, Tastendruck..

Fokus: Mechanismus, der festlegt, welche Komponente eines Fensters Tastatureingaben empfängt

```
public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        //Alternatives Layout: FlowPane statt HBox: dann verschiebt sich nichts !
        //FlowPane inp= new FlowPane(Orientation.HORIZONTAL);

        Label lbl = new Label("Number");
        TextField tfNumber = new TextField();
        Button btnCalc = new Button("Calculate");
        HBox inp = new HBox();
        inp.getChildren().addAll(lbl, tfNumber, btnCalc);

        //Stylen
        inp.setSpacing(5); //5px abstand zwischen den Elementen
        inp.setAlignment(Pos.BASELINE_LEFT);    //zentral schrift
        inp.setPadding(new Insets(5,10,5,10)); //Innenabstände im Uhrzeigersinn
                                               //farbverlauf    vordefinierte farbssymbolde
        inp.setStyle("-fx-background-color:linear-gradient(to bottom,khaki,lavender);");
        HBox.setHgrow(tfNumber, Priority.ALWAYS);    //mitwachsen immer
        HBox.setHgrow(lbl, Priority.NEVER);          //mitwachsen nie
        HBox.setHgrow(btnCalc, Priority.NEVER);      //mitwachsen nie

        Label lblResult = new Label("Result: ");

        ScrollPane sp= new ScrollPane(lblResult); //man kann scrollen!!

        sp.setPrefSize(230, 70); //pref groesse (-1 -> min platz (-1,70)
        lblResult.setAlignment(Pos.TOP_LEFT);    //oben links zentieren
        sp.setStyle("-fx-border-color: blue;");  //farbe des rahmens

        VBox root = new VBox();                  //lblResult (ohne scrollen)
        root.getChildren().addAll(inp, sp);
        VBox.setVgrow(sp, Priority.ALWAYS); //platz ausnutzen

        //auto groesse (root,400,300)
        Scene scene = new Scene(root);

        primaryStage.setScene(scene);
        primaryStage.setTitle("Factors");
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```



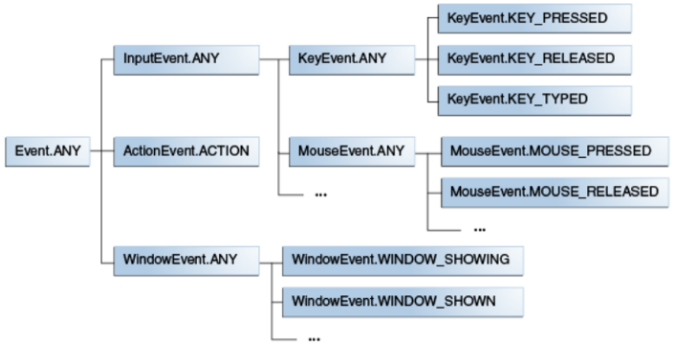



```
EventHandler<ActionEvent> handler=
new EventHandler<ActionEvent>() {
  @Override public void handle(ActionEvent event) {
     StringBuilder sb= new StringBuilder("Result:");
     long n= Long.parseLong(tfNumber.getText());
     for (long i=2; i<=n/2; i++) {
        if (n % i == 0) sb.append("\n").append(i);
     }
     lblResult.setText(sb.toString());
  }
};
btnCalc.setOnAction(handler);   //gibt andere EHaendler
tfNumber.setOnAction(handler);  //im feld selbst
```

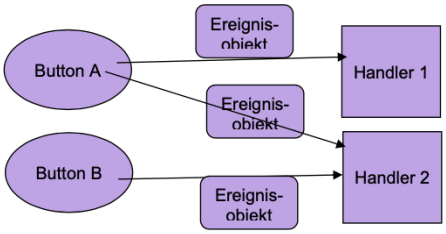
- `minWidth`, `minHeight`: minimale Größe
- `prefWidth`, `prefHeight`: bevorzugte Größe
- `maxWidth`, `maxHeight`: maximale Größe

L.8.3.1 Ereignisse

	Ereignis	Auslöser
Systemnahes Ereignis (low-level event)	Event `KeyEvent`	Taste drücken, loslassen
	Event `MouseEvent`	Maustaste drücken, loslassen, Maus bewegen
	Property `focused`	Komponente hat Fokus erhalten, verloren
	Event `WindowEvent`	Fenster aktiviert, deaktiviert, minimiert etc.
	`ListChangeListener` an `Node.getChildren()`	Komponente hinzugefügt, entfernt
Anwendungsereignis (high-level event)	`ActionEvent`	Buttonklick, Menüauswahl
	`ScrollEvent`	Ändern der Bildlaufleiste
	Property `value`	Auswahl aus Combo-Box
	Property `selectedItem`	Auswahl aus Liste
	Property `text`	Ändern eines Textes



L.8.3.3 B:setOnType vs. A:addEventHandler



`addEventHandler`:

```
public final <T extends Event>
void addEventHandler(EventType<T> eventType,
                     EventHandler<? super T> eventHandler)
```

Statt `btn.setOnAction(myHandler)` rufen wir dann auf:

`btn.addEventHandler(ActionEvent.ACTION, myHandler)`

Handler lassen sich übrigens mit der Methode **`removeEventHandler`** wieder von einer Ereignisquelle ablösen