PR2 – Formular für Lesenotizen

SS2021

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Lushaj	Detijon	1630149	23.05.21

Fehlerhafte Eingabe behandeln

Tooltip: kleines Pop-up-Fenster mit einer Beschreibung eines Dialogelementes.

```
public class Main extends Application {
    private TextField tfNumber;
    private Button btnCalc;
    EventHandler<ActionEvent> handler;
    Label lblStatus;

    @Override
    public void start(Stage primaryStage) {
       Label lbl = new Label("Number");
       tfNumber = new TextField();
       btnCalc = new Button("Calculate");
       btnCalc.setMnemonicParsing(true);       //Alt+C zum ausfuehren
       ...
       lblStatus = new Label("");              //fuer den BottomText

       BorderPane bp = new BorderPane();       //anstatt root
       bp.setTop(inp);
       bp.setCenter(sp);
       bp.setBottom(lblStatus);

       btnCalc.setOnAction(handler);           //gibt andere Eventhaelnder
       tfNumber.setOnAction(handler);          //im feld selbst
       tfNumber.setOnKeyReleased(ev -> checkInput() ); // lambdaausdruck Eventhaendler
       checkInput();           //fuer die erste pruefung damit keine eingabe ungueltig ist

       tfNumber.setTooltip(new Tooltip("Enter an int number pls"));
       btnCalc.setTooltip(new Tooltip("rechne aus"));

       Scene scene = new Scene(bp);            //anstatt root nun bp
       ...
    }
    private void checkInput() {                God method
        String eingabe = tfNumber.getText();
        if ( eingabe.matches("[0-9]+")) {      //ausdruecke von 0-9
            tfNumber.setStyle("");             //normale farbe wenn gueltig
            btnCalc.setDisable(false);         //man kann auf den button klicken
            tfNumber.setOnAction(handler);     //exception
            lblStatus.setText("");             // bottom text wird veraendert
        } else {
            tfNumber.setStyle("-fx-focus-color: red;");
            btnCalc.setDisable(true);
            tfNumber.setOnAction(null);
            lblStatus.setText("Integer number expected");
        }
```

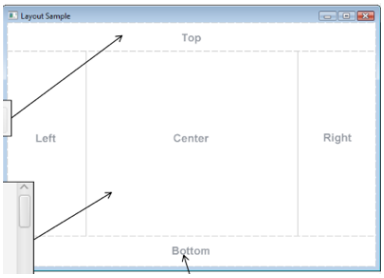
Icons

Vorgehen:

```
Image imageCalc = new Image(new File("calc.png").toURI().toString());
btnCalc= new Button("Calculate", new ImageView(imageCalc));
```

Position ändern:

```
btnCalc.setContentDisplay(ContentDisplay.TOP);
```



Properties

Properties in Java werden durch Methoden mit Namenskonvention realisiert. – zueinander passende getXxx- und setXxx-Methoden („alter Hut“)
– zzgl. xxxProperty-Methode (<- das ist neu!)

```
tfNumber.textProperty().addListener(
        (property, oldVal, newVal) -> {
            System.out.println(newVal) });  //wenn sich was verandert print das
tfNumber.setText("Hallo Welt");
tfNumber.textProperty().addListener(
        new ChangeListener<String>() {
            @Override public void changed(
                    ObservableValue<? extends String> property,
                    String oldVal,
                    String newVal) {
                System.out.println(newVal);
            }
        });
fNumber.setText("Hallo Welt");
```

- Vorteil: anwendungsnäher.
- Andere Eingabemöglichkeiten (Spracheingabe, Gestensteuerung, etc.) sind nun automatisch abgedeckt.

Nun ersetzen wir das hier:

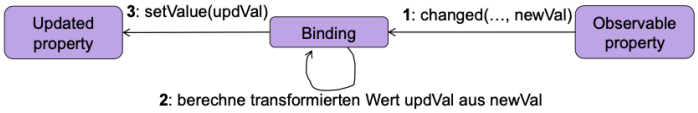
```
tfNumber.setOnKeyReleased( ev -> checkInput() );
```

- **Durch das hier:**

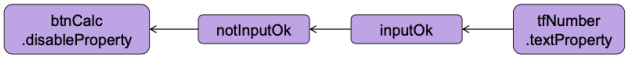
```
tfNumber.textProperty().addListener((obs,ov,nv) -> checkInput());
```

Bindings

- Ein Binding bindet ein beobachtetes Merkmal (rechts) an ein zu aktualisierendes anderes Merkmal (links)
- Das Binding-Objekt registriert sich rechts als ChangeListener und wird deshalb über Wertänderungen informiert.



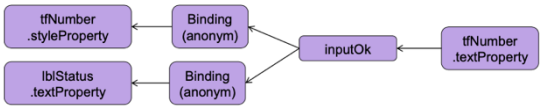

```
BooleanBinding inputOk = Bindings.createBooleanBinding(
        () -> tfNumber.getText().matches("[0-9]+"),
        tfNumber.textProperty());

//BooleanBinding notInputOk= Bindings.createBooleanBinding(
//        () -> !inputOk.getValue(),
//       inputOk);


BooleanBinding notInputOk = inputOk.not();
btnCalc.disableProperty().bind(notInputOk);


tfNumber.styleProperty().bind( Bindings.when(inputOk)
        .then("")
        .otherwise("-fx-focus-color: red;"));
lblStatus.textProperty().bind(
        Bindings.when(inputOk) .then("")
        .otherwise("Integer number expected"));
```



Gefahr: „Ravioli-Code“

- Es entstehen schnell hunderte kleiner Klassen und Objekte.

Wichtig also: Ordnung halten! Z. B. durch Projekt-Richtlinien, in welchen Klassen / Softwareschichten Bindings erstellt werden dürfen



– „God-Method“ + leichteres Debugging

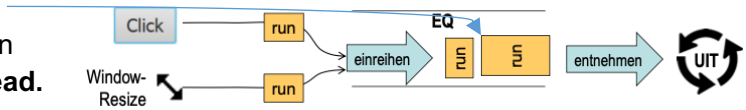
– „Ravioli-Code“. + verteilte Verantwortung

Threads in JavaFX

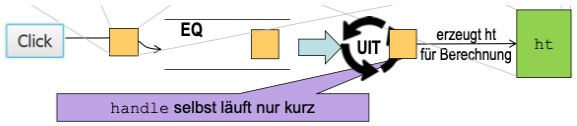
UI-Thread(UIT):

- Hier wird jegliche Ereignisbehandlung ausgeführt. (minimieren, clicken etc.)
- UIT entnimmt Runnable-Objekte (jeweils mit run-Methode) aus sog. **Event Queue (EQ)** und verarbeitet sie.

Langlaufende run-Methode blockiert nachfolgende Ereignisverarbeitungen
→ Wenn im Ereignis-Behandler aufwändige Berechnungen anstehen, **verlagern in einen eigenen Hintergrund-Thread.**



Um eine JavaFX-Anwendung mit gutem Antwortzeitverhalten zu erhalten, muss man aufwändige Berechnungen in Hintergrundthreads verlagern.



JavaFX ist nicht threadsicher

• Alle Methoden auf dem Scenegraph müssen im UI thread (UIT) aufgerufen werden.

Aufruf in den UIT zu verlagern:

- `Platform.runLater(runnable)`

reiht das Runnable-Objekt in die Event Queue ein.

- `Platform.isFxApplicationThread()`

liefert true, wenn der aufrufende Thread der UIT ist.

Aus dem Hintergrundthread heraus muss man alle Operationen mit JavaFX-Komponenten durch Platform.runLater...-Aufrufe in den UIT verlagern.

```
btnCalc.setOnAction(event -> {
    lbl.getScene().setCursor(Cursor.WAIT); //mauszeiger
    Thread ht= new Thread( () -> {                          //new Thread
        for (long i=0; i<10_000_000_000L; i++) {};
        count++;

        Platform.runLater( () -> {                          //1*
            lbl.setText(count + " Clicks");
            lbl.getScene().setCursor(Cursor.DEFAULT);
        });

    });
    ht.start();
});
```