Table of Contents

Controller

```
public class DataViewerController {
    Stage stage;

    @FXML
    private Label lName;

    @FXML
    private TextField tfFilename;

    @FXML
    private Button btSelectFile;

    @FXML
    private ChoiceBox<String> cbGemeinde;

    @FXML
    private ProgressBar pbProgress;


    @FXML
    private AreaChart<?, ?> chart;

    @FXML
    private Label lStatus;

    Service<List<String>> svImportService;
    Service svLoadDataService;
    Service svLoadGesamtService;
    Service svImportIntoDb;

    FileChooser fileChooser = new FileChooser();

    File file;
    String gemeinde;
    String srcFile;
```

```
    @FXML
    public void initialize() {
        lName.setText("Peter Klose");

        chart.getXAxis().setLabel("Jahr");
        ((NumberAxis)chart.getXAxis()).setTickLabelFormatter(new
FormatStringConverter<>(new DecimalFormat("0000")));
        chart.getYAxis().setLabel("Bewohner");

        svImportService = new Service() {
            @Override
            protected Task createTask() {
                Task task = new ImportTask(file);
                return task;
            }
        };

        svLoadDataService = new Service() {
            @Override
            protected Task createTask() {
                Task task = new LoadDataTask(gemeinde,srcFile);
                return task;
            }
        };
        svImportIntoDb = new Service() {
            @Override
            protected Task createTask() {
                Task task = new ImportIntoDbTask(srcFile);
                return task;
            }
        };
        svLoadGesamtService = new Service() {
            @Override
            protected Task createTask() {
                Task task = new LoadGesamtTask(srcFile);
                return task;
            }
        };
    }


    @FXML
    void importAction(ActionEvent event) {
        cbGemeinde.getItems().clear();
        file = fileChooser.showOpenDialog(null);

        pbProgress.progressProperty().bind(svImportService.progressProperty());
        statusAction("Importiere...");
```

```
        svImportService.setOnSucceeded(workerStateEvent -> {
            List<String> retunVal = svImportService.getValue();
            if(retunVal != null){
                cbGemeinde.getItems().add("Gesamt");
                cbGemeinde.getItems().addAll(retunVal);
                statusOk("Gemeindeliste wurde aktualisiert!");
            }else {
                statusError("Gemeindeliste wurde NICHT aktualisiert!");
            }
        });

        svImportService.restart();
    }


    @FXML
    void redrawAction(ActionEvent event) {

        statusAction("Aufbereiten der Daten...");
        gemeinde = cbGemeinde.getValue();

        chart.getData().clear();
        XYChart.Series dataSeries = new XYChart.Series();
        dataSeries.setName(gemeinde);

        if(!gemeinde.equals("Gesamt")){

        pbProgress.progressProperty().bind(svLoadDataService.progressProperty());
        svLoadDataService.restart();

        svLoadDataService.setOnSucceeded(workerStateEvent -> {
            // Task<XVChart.Series<Integer,Integer>>
            List<Integer> vals = (List<Integer>) svLoadDataService.getValue();
            if(vals != null){
                for (int i = 0; i < vals.size(); i+=2) {
                    dataSeries.getData().add(new XYChart.Data( vals.get(i), vals.get(
i+1)));
                }
                statusOk("Chart aktualisiert!");
                chart.getData().add(dataSeries);

                addChartToolTips();
            }else {
                statusError("Chart NICHT aktualisiert!");
            }
        });
        }else {
            System.out.println("GESAMT");
            pbProgress.progressProperty().bind(svLoadGesamtService.progressProperty()
);
```

```
            svLoadGesamtService.restart();
        }

        // TODO: Aufgabe 5 - Gesamteinwohner für OÖ anzeigen

        // statusAction("Aufbereitung der Daten...");
        // statusOk("Chart aktualisiert!");

    }

    @FXML
    void dbImportAction(ActionEvent event) {

        statusAction("Datenbank-Import läuft...");

        pbProgress.progressProperty().bind(svImportIntoDb.progressProperty());
        svImportIntoDb.restart();

        svImportIntoDb.setOnSucceeded(workerStateEvent -> {
            List<Boolean> vals = (List<Boolean>) svImportIntoDb.getValue();
            if(vals != null){
                statusOk("DB-Import abgeschlossen!");
            }else {
                statusError("DB-Import NICHT aktualisiert!");
            }
        });

        // statusAction("Datenbank-Import läuft....");
        // statusOk("DB-Import abgeschlossen");
    }

    void addChartToolTips() {
        chart.getData().stream().forEach(series -> {
            series.getData().stream().forEach(data -> {
                Tooltip.install(data.getNode(), new Tooltip("Jahr " + data.getXValue()
+ ": " + data.getYValue() + " Einwohner"));
                data.getNode().setOnMouseEntered(mouseEvent -> data.getNode()
.getStyleClass().add("onHover"));
                data.getNode().setOnMouseExited(mouseEvent -> data.getNode()
.getStyleClass().remove("onHover"));
            });
        });

    }

    //<editor-fold desc="//Ready methods.... no need to change...">
    @FXML
    void selectFileAction(ActionEvent event) {
        FileChooser fc = new FileChooser();
        fc.setTitle("CSV-Datei auswählen...");
        fc.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV-Datei",
```

```
"*.csv"));
        fc.setInitialDirectory(new File("."));
        File file = fc.showOpenDialog(stage);
        if (file != null)
            srcFile = file.getAbsolutePath();
            tfFilename.setText(srcFile);
    }

    //<editor-fold desc="// Helper Methods for Status-Messages">
    void statusOk(String text) {
        lStatus.setTextFill(Color.GREEN);
        lStatus.setText(text);
    }

    void statusError(String text) {
        lStatus.setTextFill(Color.RED);
        lStatus.setText(text);
    }

    void statusAction(String text) {
        lStatus.setTextFill(Color.BLACK);
        lStatus.setText(text);
    }
}
```

ImportTask

```
public class ImportTask extends Task<List<String>> {
    String src;

    public ImportTask(File file) {
        this.src = file.getAbsolutePath();
    }

    @Override
    protected List<String> call() throws Exception {
        //System.out.println(src);
        long zeilen = Files.lines(Path.of(src)).count() - 1;
        AtomicInteger i = new AtomicInteger(0);
        List<String> mylist = Files.lines(Path.of(src))
                .skip(1)
                .map(s -> s.split(";"))
                .map(strings -> {
                    updateProgress(i.incrementAndGet(),zeilen);
                    return strings[2];
                })

                .sorted()
                .distinct()
                .toList();

        return mylist;
    }
}
```

LoadDataTask

```
public class LoadDataTask extends Task {
    String gemeinde;
    String src;
    public LoadDataTask(String gemeinde,String src) {
        this.gemeinde = gemeinde;
        this.src = src;
    }

    @Override
    protected List<Integer> call() throws Exception {
        if (src == null){
            return null;
        }
        long zeilen = Files.lines(Path.of(src)).count();
        AtomicInteger i = new AtomicInteger(0);
        return Files.lines(Path.of(src))
                .filter(s -> s.contains(gemeinde))
                .map(s -> s.split(";"))
                .flatMap(strings -> {
                    updateProgress(i.incrementAndGet(),zeilen);
                    Stream<Integer> s= Stream.of(Integer.valueOf(strings[3]),Integer
.valueOf(strings[4]));
                    return s;
                }).toList();
    }
}
```

Import Into DB Task

```
public class ImportIntoDbTask extends Task {
String src;
DataRepoo repo= new DataRepoo();
public ImportIntoDbTask(String srcFile) {
this.src = srcFile;
}

    @Override
    protected Object call() throws Exception {
        if(src == null){
            return null;
        }
        long zeilen = Files.lines(Path.of(src)).count();
        AtomicInteger i = new AtomicInteger(0);
        System.out.println("test");
        List<Boolean> mylist = Files.lines(Path.of(src))
                .skip(1)
                .map(s -> s.split(";"))
                .flatMap(strings -> {
                    System.out.println("InsertClient");
                    updateProgress(i.incrementAndGet(),zeilen);
                    Stream<Boolean> l= Stream.of(repo.insert(Integer.valueOf(strings[
1]),strings[2],Integer.valueOf(strings[3]),Integer.valueOf(strings[4])));
                    return l;
                }).toList();
        repo.commit();

        return mylist;
    }
}
```

DataRepo

```
public class DataRepoo {
Connection conn;

    public DataRepoo() {
        try {
            this.conn = DriverManager.getConnection(
"jdbc:derby://localhost:1527/db;create=true","app","app");
            this.conn.setAutoCommit(false);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public boolean insert(int gemid,String gemeinde,int jahr,int einwohner){
        System.out.println("INSERT");
        try {
            PreparedStatement pstmt = conn.prepareStatement("insert into
population(gemid,gemeinde,jahr,einwohner) values (?,?,?,?)");
            pstmt.setInt(1,gemid);
            pstmt.setString(2,gemeinde);
            pstmt.setInt(3,jahr);
            pstmt.setInt(4,einwohner);

            int res = pstmt.executeUpdate();

            System.out.println(res);
            return true;

        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    public void commit() throws SQLException {
        this.conn.commit();
    }
}
```

Load Gesamt Task

```
public class LoadGesamtTask extends Task<List<Stream<Integer>>> {
    String src;

    @Override
    protected List<Stream<Integer>> call() throws Exception {
        if (src == null){
            return null;
        }

        long zeilen = Files.lines(Path.of(src)).count();
        AtomicInteger i = new AtomicInteger(0);
        return Files.lines(Path.of(src))
                .skip(1)
                .map(s -> s.split(";"))
                .collect(Collectors.groupingBy(strings -> strings[3]))
                .entrySet().stream()
                .map(stringListEntry -> {
                    String year = stringListEntry.getKey();
                    int population = stringListEntry.getValue().stream()
                            .map(strings -> Integer.valueOf(strings[4]))
                            .collect(Collectors.summingInt(Integer::intValue));
                    Stream<Integer> s= Stream.of(Integer.valueOf(year),Integer.
valueOf(population));
                    return s;
                }).toList();
    }

    public LoadGesamtTask(String srcFile) {
        this.src = srcFile;
    }
}
```