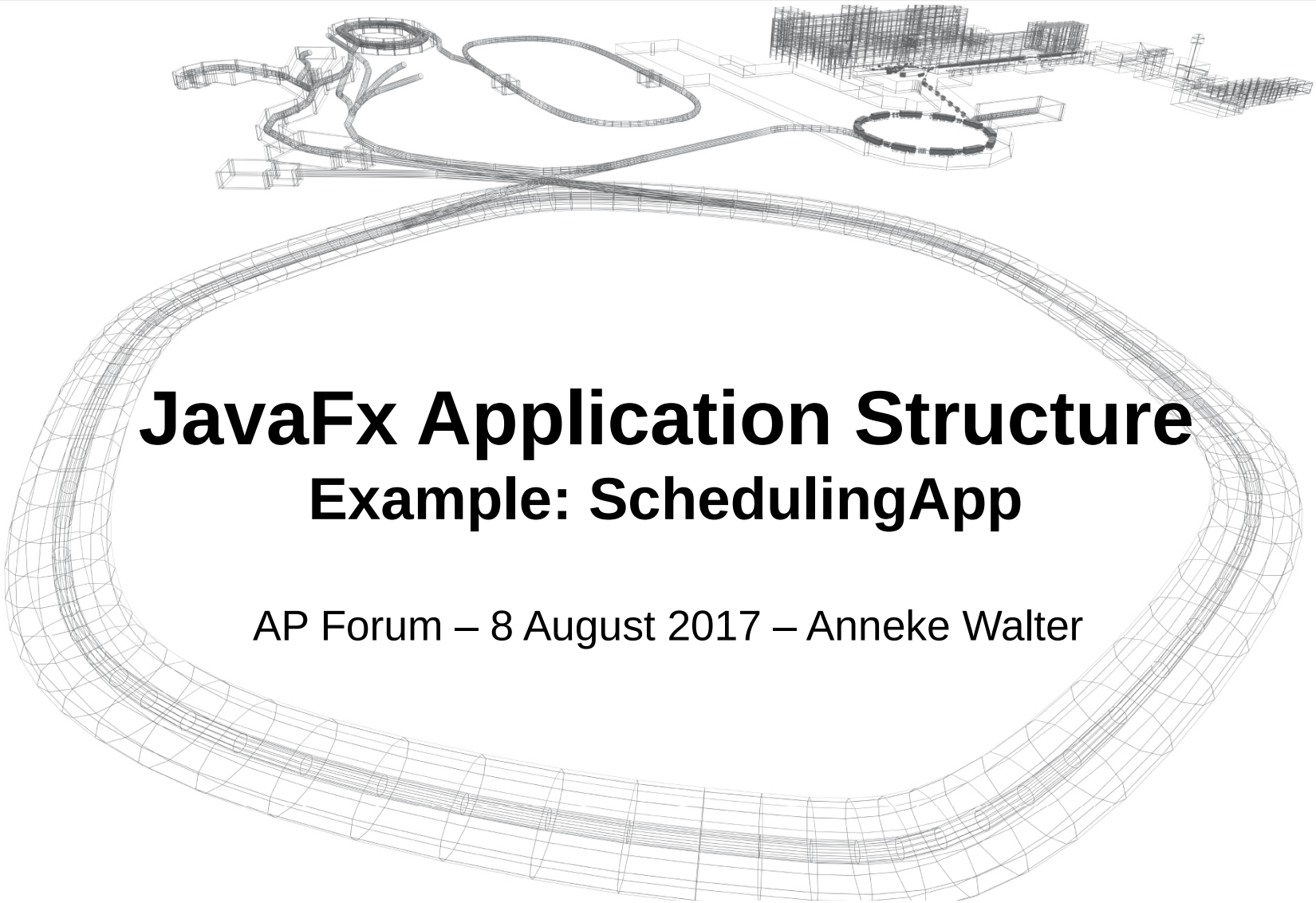


JavaFx Application Structure

Example: SchedulingApp

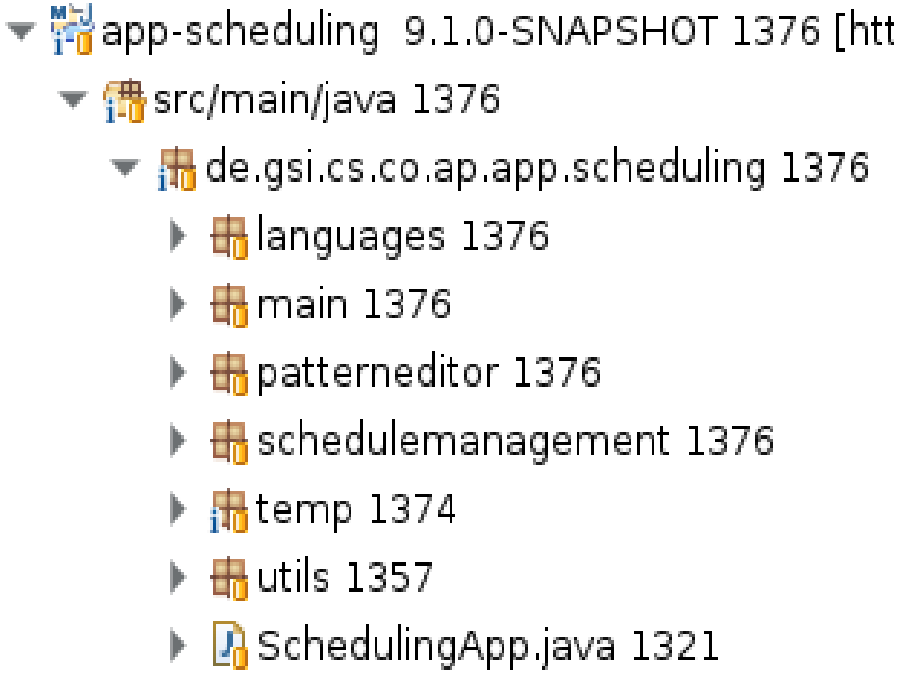
AP Forum – 8 August 2017 – Anneke Walter

Agenda

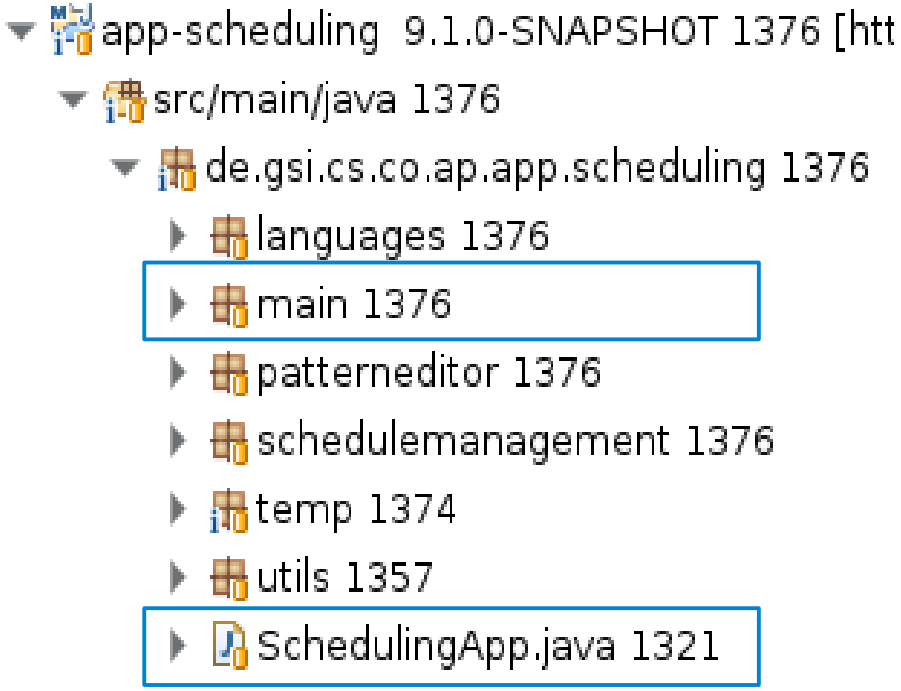
- Mini demo
- Architecture
  - Package structure
  - Model-View-Controller
- Changing the UI language
- Using the SceneBuilder
- Questions & Discussion

Mini demo

Architecture: Package structure

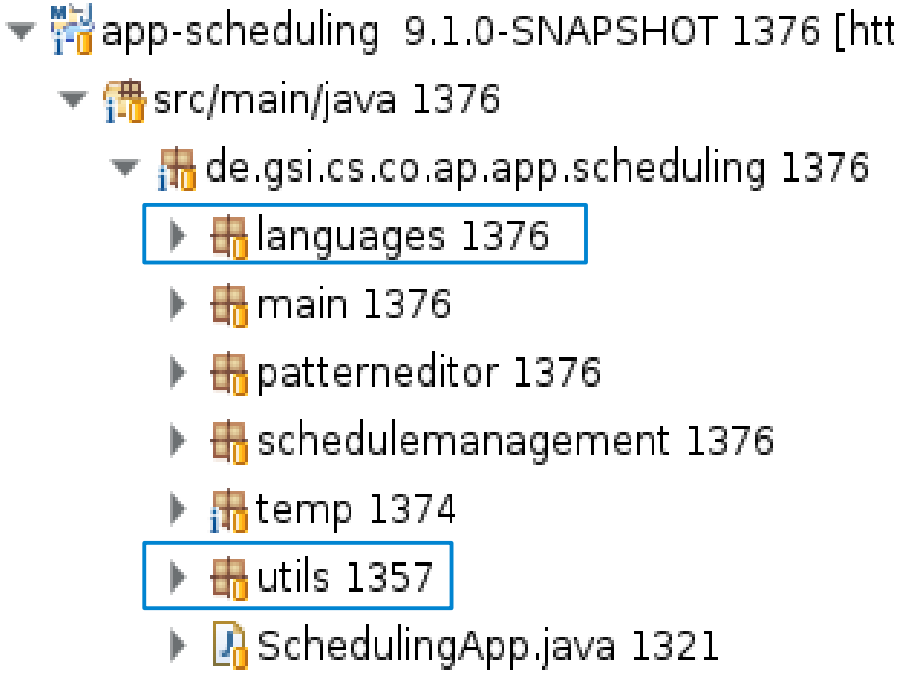


Architecture: Package structure



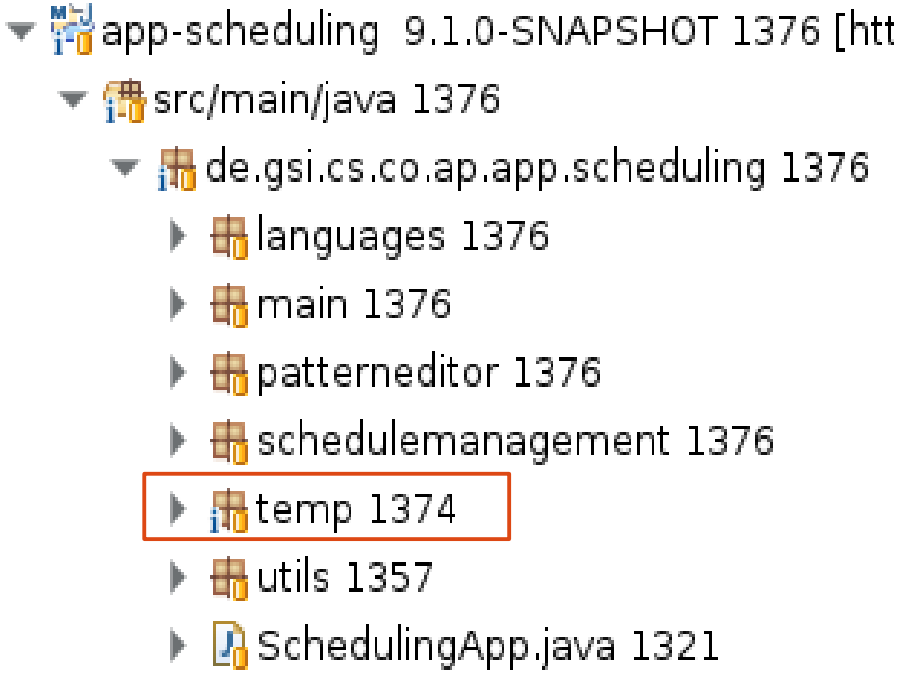
- directly in app package: only „main class“
  - only starts application!
- „main“ package for highest-level app logic

Architecture: Package structure



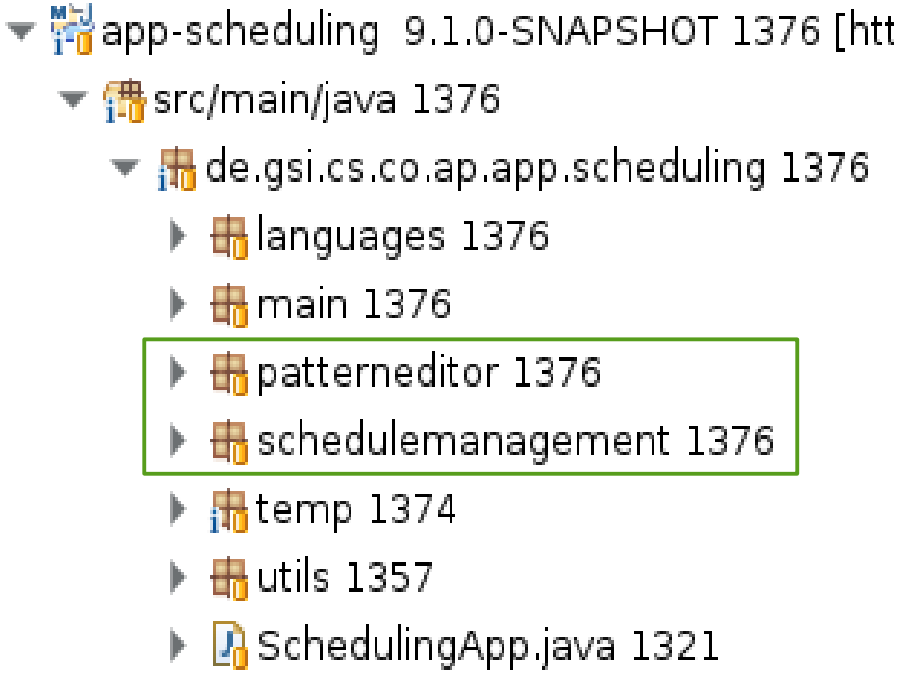
- typically useful packages for AP applications

Architecture: Package structure



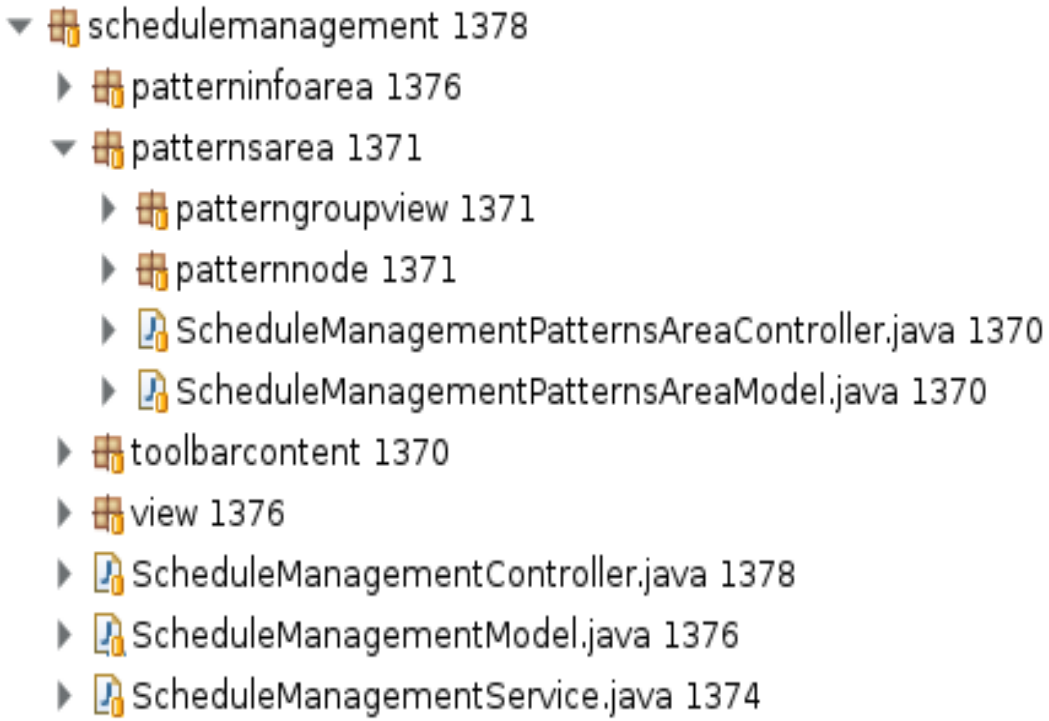
- „temporary“ classes
  - dummy data
  - classes to move to different project
  - classes to be replaced
  - ...

Architecture: Package structure



- one package per app component
  - may use data or shared functionality from main
  - same-level or sub-level package?

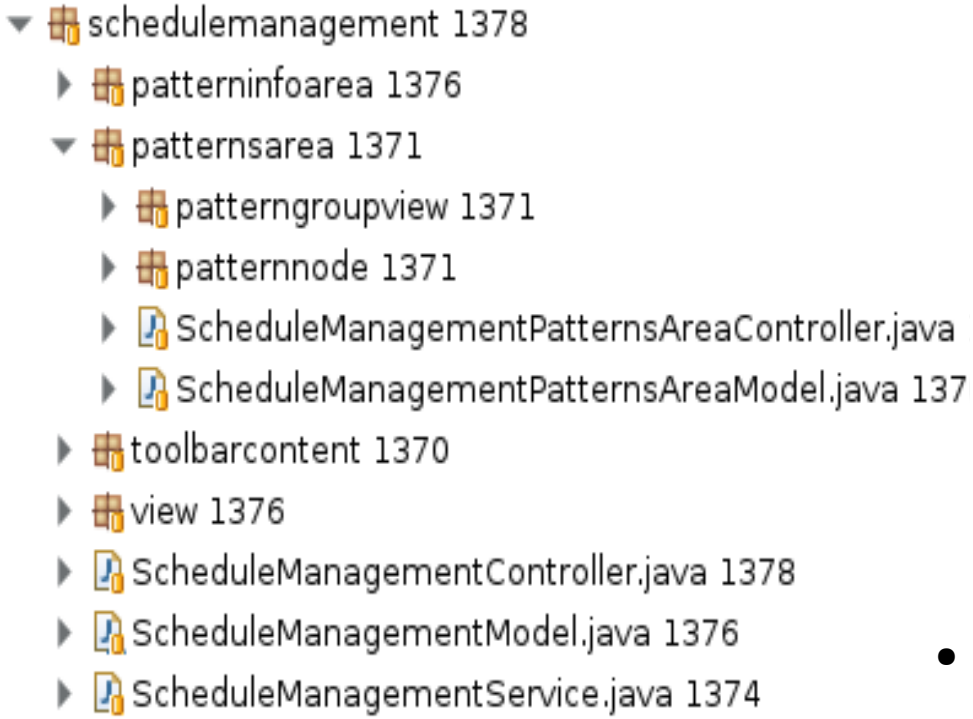
Architecture: Package structure



- „recursive“:
  sub-packages for
  app components
  use similar structure
  - one package
    per sub-controller!

Architecture: Package structure

- src/main/java

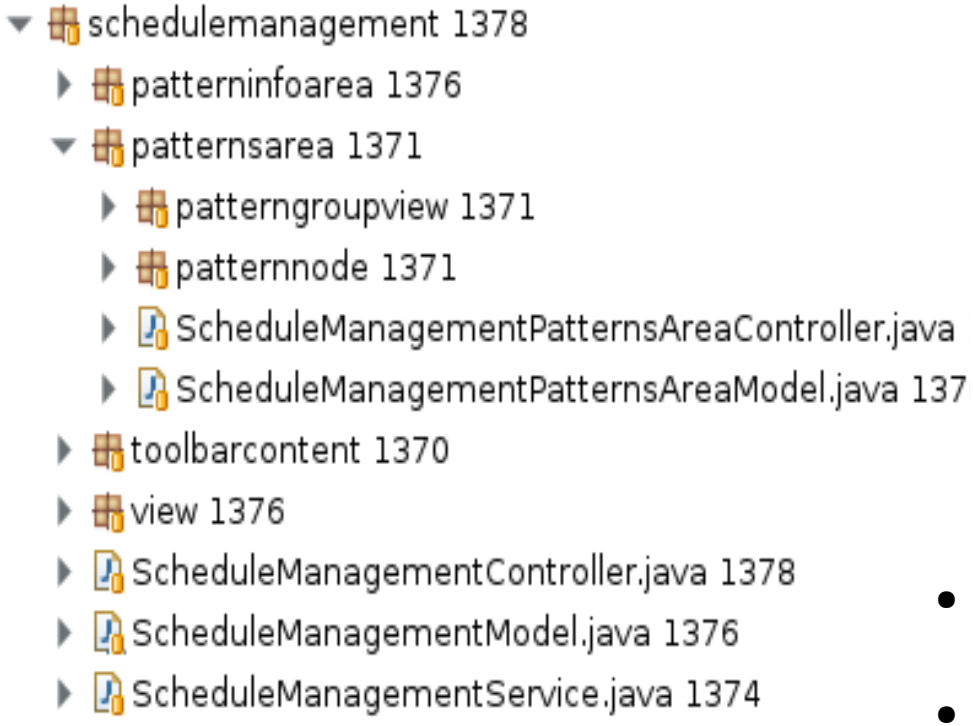


- src/main/resources

- same package structure in /java and /resources

Architecture: Model-View-Controller

- src/main/java



- src/main/resources

- **Model**: _Model
- **View**: _.fxml, _.css
- **Controller**: _Controller

Architecture: Model-View-Controller

View

- Structure/Layout: 1 _.fxml file per (sub-)component
  - define Controller class with fx:controller
- Styling: use CSS instead whenever possible
  - makes changing style at multiple places easy
  - in .css file (preferred) or in-line in FXML
  - discouraged: styling per FXML attributes or Java code

Architecture: Model-View-Controller

Controller

- one _Controller class per _.fxml file
- view logic ⇒ „view controller“ ~ M**V**C
  - e.g. wire controls together: „ButtonB=enabled when CheckboxC=checked“
  - e.g. wire controls to model: „ButtonB=enabled when Model.getList=empty“
  - e.g. set up handlers for GUI events
- application logic ⇒ application controller ~ MV**C**
  - e.g. „when handleDelete is called, delete all contexts from the database“
- no solution for clean/complete separation yet :-(

Architecture: Model-View-Controller

Controller

- „top“ controller class of a component acts as *both* application controller *and* view controller!
  - application logic: only intent „if X happens, we do Y“ - but *no* details!!

Excursus: Helper classes for application logic

- Keep details of application logic out of Controllers!
  - NO: huge controller classes that must be changed every time internal details change
  - YES: simple controller classes that show *intent* of application logic
- Strategy for development (?)
  - initially: temporary helper class like _Service or _Helper for _Controller
  - later: get rid of temporary helper classes by refactoring!
    - move helper methods into other, precisely named helper classes
    - move helper methods into existing helper classes in common libs etc.

Architecture: Model-View-Controller

Controller

- „top“ controller class of a component acts as *both* application controller *and* view controller!
  - application logic: only intent „if X happens, we do Y“ - but *no* details!!
  - view logic (try to keep minimal!)
    - open dialogs to ask user if they really want to do X etc.
    - deactivate UI parts until a long-running task is finished
    - …
- sub-controllers should be pure view controllers

Architecture: Model-View-Controller

Controller

- communication between controllers
  - super-controller knows sub-controllers
  - sub-controllers do not know each other or super-controller
  - delegate action-handling from sub-controller to super-controller
    - sub-controller offers methods to register action handlers
    - super-controller registers handlers
  - communication between controllers: separate, but wired models!

Architecture: Model-View-Controller

Model

- usually: one _Model class per _Controller class
  - if _Model *extremely* simple: optionally just use model object directly
- Model provides data as JavaFx observables
  - properties (+ getters, setters where applicable), observable collections
- Models for sub-controllers of components often only define „empty“ observables, do not retrieve data
  - super-controller sets up bindings between its own model and the sub-models of its sub-controllers!

Changing the UI language

- JavaFx StringBindings, updated automatically whenever language changed via LanguageSelectorImpl!
- using helper class **LanguageDependentBindings** in common-language-translator

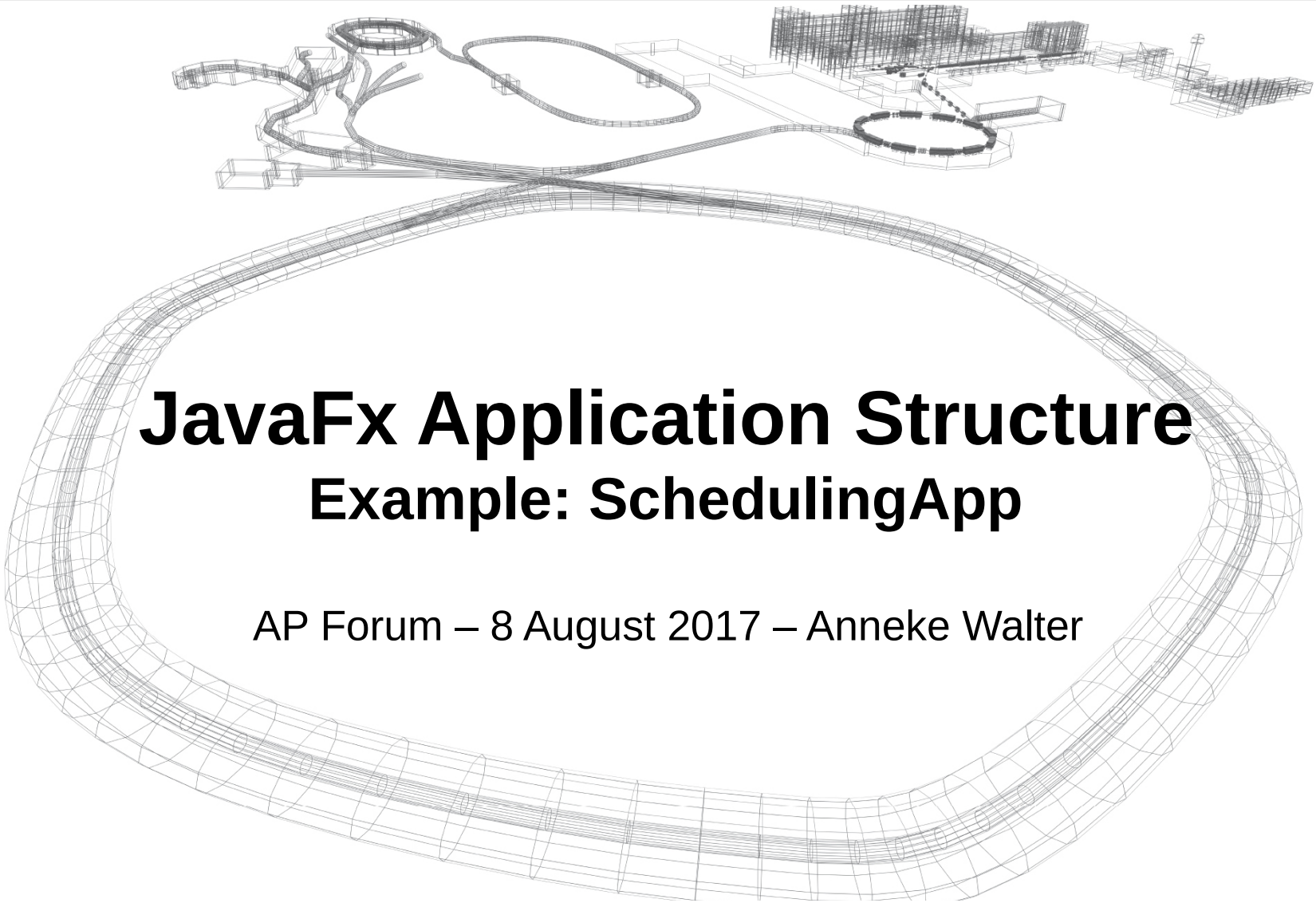
```
myLabel.textProperty().bind(
    LanguageDependentBindings.createLanguageBinding(
                    MyResourceBundle.LANG_TEXT_HELLO));
```

Using the Scene Builder

- `javafxscenebuilder`
- useful WYSIWYG tool for creating FXML files
  - use it! unless you prefer plain FXML markup
- support for in-development custom controls impractical
  - consider adding custom controls to view in Java
- some unusual but valid FXML constructs not supported
  - ask yourself: really needed?

Questions and Discussion

- Covered Topics
  - Architecture
    - Package structure: recursive, package-per-controller
    - Model-View-Controller
      - View: FXML, CSS
      - Controller: view/application controllers, helper classes for app logic details
      - Model: model class (usually), observables, wired sub-models
  - Changing the UI language: LanguageDependentBindings
  - Using the SceneBuilder



JavaFx Application Structure

Example: SchedulingApp

AP Forum – 8 August 2017 – Anneke Walter