

JavaFX für CMF 3.0

Das für Visualisierung und Event-Handling eingesetzte Framework für CMF 3.0 ist JavaFX (11 oder höher) - siehe <https://openjfx.io>

Nachfolgend sind einige Grundlagen aus JavaFX zusammengefasst und jeweils auf die Anwendung in CMF 3.0 bezogen.

Basics

Die wichtigsten Konzepte in JavaFX sind

- Stage - <https://openjfx.io/javadoc/11/javafx.graphics/javafx/stage/Stage.html>
- Scene - <https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/Scene.html>
- Node - <https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/Node.html>
 - Group - <https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/Group.html>

Eine Stage ist im Prinzip ein Fenster, das als „Bühne“ für unseren Inhalt dient. In der Stage enthalten ist ein Objekt der Klasse Scene.

- Das Stage-Objekt wird von der JavaFX-Runtime selbst erzeugt - nachdem in CommunityMirror.main() über Application.launch(args) die Kontrolle an die Runtime übergeben worden ist. Das Stage-Objekt wird als Parameter stage in der Funktion CommunityMirror.start() übergeben, die nach Bedienung der Initialisierung der JavaFX-Runtime aufgerufen wird.

Die Scene („Szene“) enthält alle Objekte, die in der Stage gerendert werden sollen, sowie interne Werte, auf die zugegriffen werden kann.

- In CMF3 wird das Scene-Objekt im View erzeugt und in CommunityMirror.createMainView() vom View ausgelesen und der Stage zugeordnet (mittels stage.setScene()).
- Konkret erfolgt die Erzeugung des Szene-Objektes im Konstruktor der View-Klasse: this.scene = new Scene(this, width, height);
- Die Klasse vom Typ View ist von javafx.scene.Group abgeleitet und dient als Wurzel-Node der Scene (des Scene-Graphs)
- "Gefüllt" wird die Scene in der onInit-Methode der View - d.h. es werden Kind-Objekte in die View-Group eingehängt.

Die Objekte, die in der Stage gerendert werden sollen, sind alle vom Typ Node abgeleitet.

- Die Wurzel-Node der Scene ist das View-Objekt (vom Typ javafx.scene.Group)
- in FlowView.onInit() werden unterhalb dieses Nodes eingefügt:
 - weitere javafx.scene.Group-Objekte für background, flow, area, graph, front
 - In diese Gruppen werden dann die InformationComponents eingefügt

Das ganze heisst auch "Scene-Graph"

TBD

TBD: Background image - wie passt das da rein

Wir verwenden keine LayoutManger - die Objekte werden direkt gesetzt ... TBD

Nodes

Nodes sind primitive Objekte in JavaFX

Nodes können nicht nur gerendert werden, alle Nodes lassen sich auch bewegen, skalieren, drehen oder scheren. Außerdem lassen sich auf Nodes visuelle Effekte anwenden.

Siehe: <https://openjfx.io/javadoc/11/javafx.graphics/javafx/scene/Node.html>

Koordinatensystem

In einem Node wird ein klassisches lokales Koordinatensystem genutzt - mit einer X-Achse, die nach rechts wächst und einer Y-Achse, die nach unten wächst.

Auf Nodes können Transformations angewendet werden (rotation, translation and scaling) - diese wirken innerhalb des lokalen Koordinatensystems.

Deshalb

- Node.getBoundsInLocal() liefert LayoutX, LayoutY und alle angewandte Effekte (dazu später) - "Each Node has a read-only `boundsInLocal` variable which specifies the bounding rectangle of the Node in untransformed local coordinates. `boundsInLocal` includes the Node's shape geometry, including any space required for a non-zero stroke that may fall outside the local position/size variables, and its `clip` and `effect` variables."
- Node.getBoundsInParent() liefert LayoutX, LayoutY + Effekt + Transformationen - "Each Node also has a read-only `boundsInParent` variable which specifies the bounding rectangle of the Node after all transformations have been applied, including those set in `transforms`, `scaleX/scaleY`, `rotate`, `translateX/translateY`, and `layoutX/layoutY`. It is called "boundsInParent" because the rectangle will be relative to the parent's coordinate system. This is the 'visual' bounds of the node."

"The node's final translation will be computed as layoutX + translateX, where layoutX establishes the node's stable position and translateX optionally makes dynamic adjustments to that position."

Es gibt folglich zwei Möglichkeiten, einen Node zu platzieren / zu verschieben:

- Änderung von layoutX, layoutY (z.B. mit der Funktion relocate()) - Änderung im Koordinatensystem des ParentNodes (Group) - wird normalerweise von LayoutManagern benutzt
 - Defines the x coordinate of the translation that is added to this Node's transform for the purpose of layout. The value should be computed as the offset required to adjust the position of the node from its current layoutBounds minX position (which might not be 0) to the desired location.
For example, if textnode should be positioned at finalX - textnode.setLayoutX(finalX - textnode.getLayoutBounds().getMinX());

Failure to subtract layoutBounds minX may result in misplacement of the node. The relocate(x, y) method will automatically do the correct computation and should generally be used over setting layoutX directly.
- Änderung von translationX, translationY - Änderung im lokalen Koordinatensystem des Nodes (Offset)

Für die VisualComponents ergibt sich folgendes:

- boundsInLocal are a Node's bounds in its own coordinate space (and before transformations like scale and rotate are applied): boundsInLocal: BoundingBox [minX:-7.0, minY:-7.0, minZ:0.0, width:214.0, height:214.0, depth:0.0, maxX:207.0, maxY:207.0, maxZ:0.0]
- boundsInParent are a Node's bounds in its Parent's coordinate space (and after transformations are applied) boundsInParent: BoundingBox [minX:27.584625244140625, minY:128.5409698486328, minZ:0.0, width:214.0, height:213.99998474121094, depth:0.0, maxX:241.58462524414062, maxY:342.54095458984375, maxZ:0.0]
- layoutBounds: BoundingBox [minX:-7.0, minY:-7.0, minZ:0.0, width:214.0, height:214.0, depth:0.0, maxX:207.0, maxY:207.0, maxZ:0.0]
- layoutX/Y: 34.58462470559226, 135.54096727565263
- translateX/Y: 0.0, 0.0
- scaleX/Y: 1.0, 1.0
- (0,0) localToScene: Point2D [x = 34.584625244140625, y = 135.5409698486328]
- (0,0) localToScreen: Point2D [x = 34.58462470559225, y = 180.54096727565263]

CSS

The Node class contains id, styleClass, and style variables that are used in styling this node from CSS. The id and styleClass variables are used in CSS style sheets to identify nodes to which styles should be applied. The style variable contains style properties and values that are applied directly to this node.

See

- <https://www.callicoder.com/javafx-css-tutorial/>
- <https://docs.oracle.com/javase/8/javafx/api/javafx/scene/doc-files/cssref.html>

CSS is currently used in the CMF3 to style designs in FXML files - see there.

Groups

A Group node contains an ObservableList of children that are rendered in order whenever this node is rendered.

A Group will take on the collective bounds of its children and is not directly resizable.

Any transform, effect, or state applied to a Group will be applied to all children of that group. Such transforms and effects will NOT be included in this Group's layout bounds, however if transforms and effects are set directly on children of this Group, those will be included in this Group's layout bounds.

By default, a Group will "auto-size" its managed resizable children to their preferred sizes during the layout pass to ensure that Regions and Controls are sized properly as their state changes. If an application needs to disable this auto-sizing behavior, then it should set `autoSizeChildren` to false and understand that if the preferred size of the children change, they will not automatically resize (so buyer beware!).

Animations

JavaFX unterstützt Animationen - dabei wird eine Strategie vorgegeben und dann über die vorgegebene Zeit **per Transformations** auf den Node angewandt.

z.B.

```

double fromX = visualComponent.getLayoutBounds().getMinX() + visualComponent.getLayoutBounds().getWidth() / 2;
double fromY = visualComponent.getLayoutBounds().getMinY() + visualComponent.getLayoutBounds().getHeight() / 2;
fromX += visualComponent.getTranslateX();
fromY += visualComponent.getTranslateY();
Path path = new Path();
path.getElements().add(new MoveTo(fromX, fromY));
path.getElements().add(new LineTo(fromX+diffX, fromY+diffY));
PathTransition pathTransition = new PathTransition();
pathTransition.setDuration(Duration.millis(500));
pathTransition.setPath(path);
pathTransition.setNode(visualComponent);
pathTransition.setOrientation(PathTransition.OrientationType.NONE);
pathTransition.setCycleCount(1);
pathTransition.setAutoReverse(true);
pathTransition.play();

```

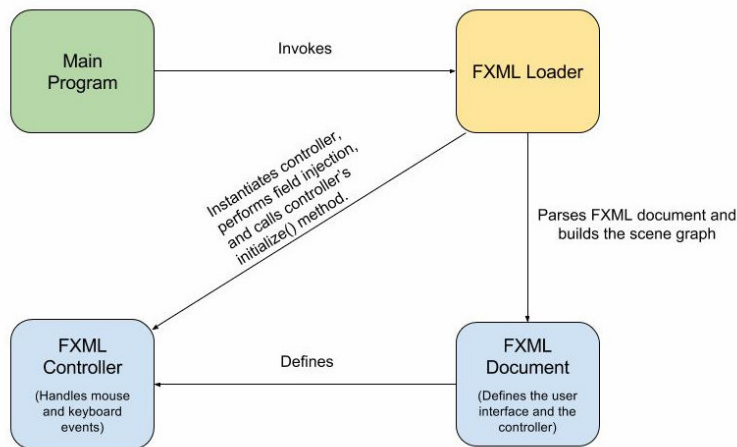
Events

MouseEvent#getX() returns the x-coordinate relative to the source node. MouseEvent#getSceneX() returns the x-coordinate relative to the source node's scene. And MouseEvent#getScreenX() returns the x-coordinate relative to the entire screen

FXML

Die **Model-View-Controller-Idee** wird folgendermaßen umgesetzt:

- Das Model sind die Informationspartikel (also die Item-Objekte von CommunityMashup)
- Die View sind die Instanzen der FlowVisualItem-Objekte, deren konkretes Layout in FXML-Dateien definiert ist (die Auswahl der FXML-Datei erfolgt in FXMLVisualItem).
- Der Controller ist das FXMLController-Objekt, das in der init()-Methode des FlowVisualItem erzeugt wird (bzw. genauer in der von der init()-Methoden aufgerufenen Methode loadViewAndController()) - dieses Objekt erhält eine Referenz auf das Model und versorgt die View mit Daten - und wird aufgerufen, wenn die View Interaktionen handhabt



Hinweis: Die MVC-Umsetzung im CMF3 ist nicht "ganz sauber", denn die Klasse FlowVisualItem definiert Callbacks für Touch und MouseEvents etc - übernimmt also einige Aufgaben eines Controllers.

CSS in FXML

CSS (see earlier) can be used on JavaFX nodes to style them in FXML files.

In CMF3 we load a default CSS file - additional CSS files from the theme can be loaded via `stylesheets="@../flow.css"` in the FXML file.

Threads

Die JavaFX-Node-Klassen sind nicht threadsafe - d.h. sie können nur von einem Thread - dem JavaFX Application Thread genutzt werden.

Wenn nun (z.B. in Event-Händlern) länger dauernde Aktivitäten benötigt werden, dann ist dafür ein eigener Thread zu starten - Problem: In dem Thread können keine Node-Objekte genutzt werden - dafür ist es notwendig, die Update-Aktivitäten wieder in den Application-Thread zu schieben:

```
Platform.runLater(new Runnable() {
    @Override
    public void run() {
        addTeaserComponent(item, finalUrl);
    }
});
```

Beispiel einer komplexeren Nutzung:

```
Task task = new Task<Void>() {
    @Override public void call() {
        // load image to cache
        ImageCache.fetchImageImmediately(finalUrl);
        // and now the stuff that has to be executed in the JavaFX thread
        Platform.runLater(new Runnable() {
            @Override
            public void run() {
                addTeaserComponent(item, finalUrl);
            }
        });
    }
    return null;
};
new Thread(task).start();
```

Any computation intensive tasks must be decoupled from the JavaFX's main application thread by using separate worker threads. JavaFX provides a complete package to deal with the issues of multithreading and concurrency. There is an interface called Worker, an abstract class called Task, and ScheduledService for this purpose. The Task is basically a Worker implementation, ideal for implementing long running computation. The Task class extends FutureTask, and, as a result, supports Runnable, Future, and RunnableFuture interfaces as well. Due to its legacy/inheritance, this class can be used in various ways.

<https://docs.oracle.com/javafx/2/threads/jfxpub-threads.htm>

Andere Nutzung von Threads in ImageCache:

```
this.executor = Executors.newFixedThreadPool(MAX_THREADS, runnable -> {
    Thread t = new Thread(runnable);
    t.setDaemon(true);
    return t ;
});
public Future<Image> doFetchImage(String identifier) {
    Task<Image> task = new Task<Image>() {
        @Override
        protected Image call() throws Exception {
            updateTitle("Image Cache Loader");
            //Check if matching Image is in cache
            Image image = getCachedImage(identifier);
            if(image != null) {
                return image;
            }
            //Create new Image
            image = new Image(identifier, false);
            addToCache(identifier, image);
            return image;
        }
    };
    executor.execute(task);
    return task;
}
```

Erweiterungen

- <http://jfxtras.org/overview.html>
- <https://github.com/controlsfx/controlsfx/wiki/ControlsFX-Features>

Physics Library / Engine für JavaFX

Noch nicht näher betrachtet ...

<http://www.jbox2d.org> (<https://github.com/jbox2d/jbox2d>) - wird benutzt in <https://github.com/AlmasB/FXGL>

<https://netopyr.com/2013/03/06/javafx-library-for-inverse-kinematics-2-0/>

Veraltet?

<http://www.cokeandcode.com/phys2d/>