CMF3 - Entwicklungsumgebung

in Eclipse

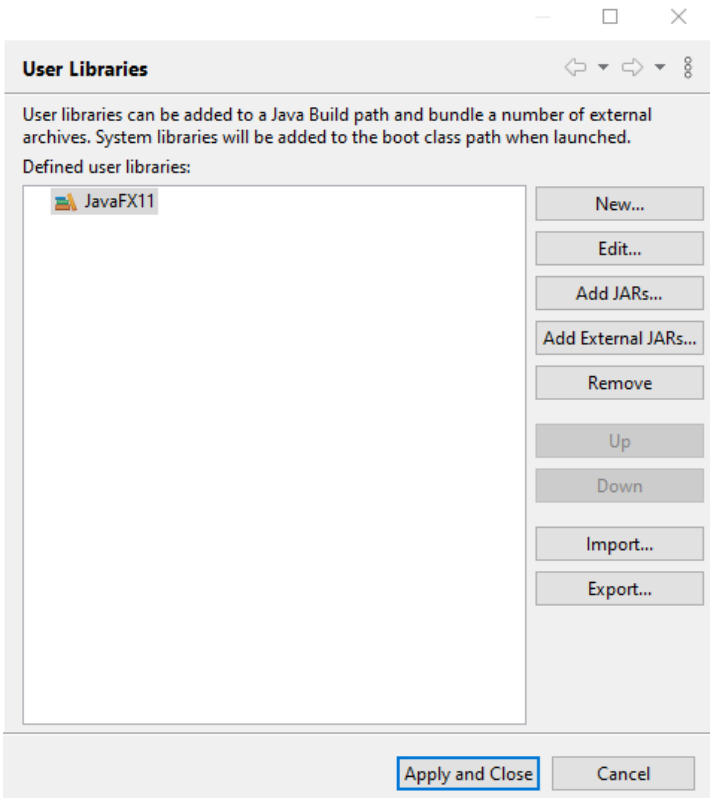
JDK 11 installieren - z.B. von http://jdk.java.net

Eclipse installieren

- eventuell "Add Java 11 as an installed JRE to Eclipse": Eclipse -> Window -> Preferences -> Java -> Installed JREs -> Add.

JavaFX installieren

- Package von https://openjfx.io holen und lokal ablegen, z.B. unter /Users/XXXX/Development/lib/javafx-sdk-11.0.2 (auf einem Mac)
- User Library "JavaFX11" (Name muss exakt übereinstimmen) anlegen: Eclipse -> Window -> Preferences -> Java -> Build Path -> User Libraries -> New JARs im lib Unterordner hinzufügen (alle jar-Dateien im bin-Unterodner):
  - New… -> „JavaFX11“
  - JavaFX11 auswählen -> Add External JARs… -> Navigieren zum fx Ordner … \Development\lib\javafx-sdk-11.0.2**\lib** und Auswahl der jar-Dateien
  - Beispiel aus Eclipse 2020-03 Version für Windows:



Projekt aus GIT importieren

- https://athene2.informatik.unibw-muenchen.de/CM/communitymirrorframework3.git
- Sicherstellen, dass Java 11 als JDK ausgewählt ist.
- User Library JavaFX11 im Modulpfad des Projektes hinzufügen (kontrollieren, ob die User Library hinzugefügt ist)

Run Configuration "Java Application" anlegen

- Main Class: org.sociotech.cmf3.CommunityMirror
- Program Arguments: mirror.properties
- Wichtig: Sicherstellen, dass die Option "Use the -XstartOnFirstThread argument when launching with SWT" NICHT ausgewählt ist. [Diese Option wird bei Windows Eclipse Version 2020-03 nicht angezeigt.]

Damit sollte der Demonstrator mit den Konfigurationen in resources/mirror.properties laufen.

Um für den eigenen Test Konfigurationen zu ändern kann eine der folgenden beiden Wege eingeschlagen werden:

1) In Eclipse: Kopieren der Datei resources/mirror.properties - z.B. nach resources/mirror.properties.local - die dann nicht unter GIT-Verwaltung steht und lokal editiert werden kann - und setzen der neuen Datei bei Program Arguments in der Run Configuration

2) In Eclipse: Die zu ändernden Konfigurationsattribute in der Run Configuration unter Environment definieren - also dort z.B. "mashup.server" als neue Variable setzen und einen Wert zuweisen.

Logging

Für das (Console-)Logging wird das SLF4J/Logback-Logging-Framework verwendet - http://logback.qos.ch/index.html.

Die Konfiguration des Loggings erfolgt in der Datei resources/logback.xml - Neben dem Logging für Debugging-Zwecke auf der Konsole ist ein Logging in die Datei log/communitymirror.log definiert. Weiterhin ist hier auch der org.sociotech.cmf3.log.LoggingFrameworkAppender definiert, der WARN und ERROR-Meldungen in das CMF3-interne Logging-Framework einspeist.

Das Logging auf die Konsole und in die Log-Datei kann über Kommandozeilenparameter konfiguriert werden:

- -Dlogback.console.threshold=DEBUG (Default ist DEBUG, Alternativen sind TRACE, INFO, WARN, ERROR, ALL oder OFF)
- -Dlogback.file.threshold=INFO (Default ist INFO, Alternativen sind TRACE, DEBUG, WARN, ERROR, ALL oder OFF)

Entwicklung unter MacOS

... klappt eigentlich bestens (wie oben beschrieben).

Einziges Problem ist, dass Macs normalerweise keinen Touch-Screen haben - Es ist also schwierig, die Touch-Interaktion zu testen.

Ich dachte erst, dass es eine Möglichkeit wäre Apples Sidecar zu nutzen und ein iPad als "second (touch) screen" für den Mac verwenden. Siehe https://support.apple.com/de-de/HT210380 für weitere Information. Geht leider nicht - "second screen" ja, nicht aber Touch - denn das unterstützt Apple nicht.

Es funktioniert allerdings mit der kommerziellen App "Duet".