



# GUI Übung 8 – Architektur

## Beschreibung

Ziel der Übung ist es, die bekannte Anwendung in eine Schichtenarchitektur zu überführen. Die Schichten werden dabei als eigene Komponenten umgesetzt. Technisch erfolgt dies als Multi-Projekt-Gradle-Build.

**Hinweis:** Eine typische Projektstruktur für Gradle sieht wie folgt aus:

```
<Wurzelprojekt>
| - build.gradle
| - settings.gradle
| - <Subprojekt 1>
|   | - build.gradle
|   | - src
|     | - main
|       | - java           //Java Klasse
|       | - resources      //sonstige Dateien, z.B.: FXML, ...
| - <Subprojekt 2>
|   | - build.gradle
|   | - src
|     | - main
|       | - java
|       | - resources
```

## Vorbereitung – Startprojekt

Holen Sie sich das Starter-Projekt von

<https://github.com/dominikhaas/Vorlesung-GUI-2020/tree/master/codebase/e08-architecture-starter> und öffnen Sie dieses in IntelliJ.

Starten Sie im Gradle-Fenster auf der rechten Seite einen Build.

Führen Sie den Gradle-Task `run` aus.

**Ergebnis:** Die Anwendung startet und die bekannte Oberfläche wird angezeigt.

## Aufgabe 1 – Schichtenarchitektur

Zerlegen Sie das existierende Projekt in Schichten. Gehen Sie dabei wie folgt vor:

- 1.) Bestimmen Sie die drei typischen Schichten aus der Vorlesung. Legen Sie für jede noch fehlende Schicht ein entsprechend benanntes Modul vom Typ „Gradle -> Java“ an. Das sind ihre Subprojekte.  
**Hinweis:** Zum Beispiel über das Kontextmenu auf dem Projekt -> New -> Module
- 2.) Verschieben Sie die Klassen aus dem Package `unsorted` in die jeweilige Schicht.  
**Hinweis:** Wenn Sie auf Klassen aus einem anderen Projekt zugreifen wollen, so müssen Sie die Abhängigkeit (dependency) hinzufügen. Sie können dies über den Eintrag  
`„compile project (':<Subprojekt>')“` im Abschnitt `dependencies` tun.
- 3.) Bauen Sie das Projekt und führen Sie es aus.

## Aufgabe 2 – Komponente

Erzeugen Sie in Ihrer Schicht „Anwendungskern“ eine neue saubere Komponente zur Protokollierung mit folgendem Interface:

```
public interface ProtocolService {  
    void writeProtocol(ProtocolEntry entry);  
}
```

Gehen Sie dabei wie folgt vor:

- 1.) Erzeugen Sie ein neues Package.
- 2.) Legen Sie das Interface an.
- 3.) Denken Sie sich eine Klasse `ProtocolEntry` aus, mit mindestens zwei Feldern.
- 4.) Schreiben Sie eine Dummy-Implementierung für den Service
- 5.) Teilen Sie die Komponente in Schnittstelle und Implementierung auf.
- 6.) Verwenden Sie den neuen Service im Service `EventServiceImpl`
- 7.) Prüfen Sie, mit den Folien der Vorlesung, ob Sie alle Regeln zum Thema „Legale und illegale Abhängigkeiten“ einhalten.

## Aufgabe 4 – Java Modul System (optional & fortgeschritten)

Verwenden Sie das Java Modul System um Sichtbarkeiten zu steuern und für den Service-Lookup. Gehen Sie dabei wie folgt vor:

- 1.) Ergänzen Sie die build.gradle Datei der Java-Module (nicht JavaFX-, nicht GUI-Modul) um den unten angeführten Abschnitt.
- 2.) Fügen Sie in jedes Subprojekt unter src/main/java eine Datei module-info.java ein.
- 3.) Steuern Sie die Sichtbarkeiten so, dass die API sichtbar (exports) ist und die Implementierung nicht.
- 4.) Registrieren Sie die Implementierung des Service  
(zum Beispiel: provides EventService with EventServiceImpl;).
- 5.) Passen Sie die modul-info.java der GUI so an, dass alle Pakete erreichbar sind (open) und signalisieren Sie die Verwendung des Service. Siehe Listing unten
- 6.) Entfernen Sie die direkte Verwendung des Services DataServiceServiceImpl in der GUI. Verwenden Sie stattdessen:  
EventService service =  
ServiceLoader.load(EventService.class).findFirst().get();

Ergänzung für den Modul-Build (nur in Java-Modulen, nicht JavaFX, nicht GUI):

```
ext.moduleName = '<module name>'

compileJava {
    inputs.property("moduleName", moduleName)
    doFirst {
        options.compilerArgs = [
            '--module-path', classpath.asPath,
        ]
        classpath = files()
    }
}
```

Module-info.java für das GUI-Projekt (JavaFX)

```
module de.throsenheim.gui.events.gui {
    requires javafx.controls;
    requires javafx.fxml;
    requires org.slf4j;

    requires de.throsenheim.gui.events.business;

    opens de.throsenheim.gui.events;
    opens de.throsenheim.gui.events.eventform;
    opens de.throsenheim.gui.events.status;
    opens de.throsenheim.gui.events.event;

    uses EventService;
}
```