

SOFTWAREPRAKTIKUM

WWU MÜNSTER
INSTITUT FÜR INFORMATIK

DR. LUDGER BECKER
DOMINIK DREES

WS 19/20

AUFGABENSTELLUNG

17.02.2020

Hauptaufgabe

Entwickeln Sie in Gruppenarbeit einen Prototypen einer Biobanken-Software zur Verwaltung von Biomaterialproben aus medizinischen Studien.

Eine erste Beschreibung des Anwendungsgebietes finden Sie in diesem Dokument. Wenden Sie dabei den in der Vorlesung vorgestellten, objektorientierten Softwarekonstruktionsprozess an. Liefern Sie Ihre Zwischen- und Endergebnisse in dem in der Vorlesung bekannt gegebenen Umfang rechtzeitig zu den festgelegten Terminen im Learnweb ab. Stellen Sie die von Ihnen entwickelte Software und ihr Design in einer Präsentation (ca. 25 Minuten) vor.

Beschreibung des Anwendungsgebietes

Der von Ihnen entwickelte Prototyp soll verschiedene Gruppen von Benutzer*innen mit speziellen Anforderungen unterstützen. Hierbei soll es möglich sein, dass Benutzer*innen mehrere Rollen annehmen können.

*Biobank-Leiter*innen* administrieren die Biobank. Sie legen neue Studien an und verwalten diese. Zudem können sie neue Räume und Kühlschränke im System eintragen, modifizieren und ggf. aus diesem entfernen. Des Weiteren werden durch *Biobank-Leiter*innen* neue Biomaterialproben- und Behälter-Typen im System eingetragen.

Study Nurses überwachen die Ausführung medizinischer Studien. Sie tragen außerdem zu der Ausführung dieser Studien bei, indem sie medizinische Daten, die bei Visiten anfallen, im System eintragen.

*Medizinisch-Technische Assistenten*innen (MTAs)* sind im Labor tätig und interagieren im Zuge ihrer Arbeit direkt mit den Biomaterialproben. Zu ihren Aufgaben gehört es, in Visiten entnommene Biomaterialproben einzulagern und diese dabei auch in das System einzupflegen. Ebenso können sie Biomaterialproben aus der Lagerung (und damit dem System) entnehmen. Neue, durch Verarbeitung anderer Biomaterialproben entstandene, Biomaterialproben müssen ebenfalls im System erfasst werden.

*Leiter*innen der Personalabteilung* sind dafür verantwortlich, neue Benutzer*innen im System zu registrieren. Sie weisen diesen außerdem Rollen zu, die bestimmen, auf welche Funktionalität des Biobank-Verwaltungssystems sie zugreifen können. Benutzer*innen müssen dabei zu jedem Zeitpunkt mindestens eine Rolle haben.



(a) Biomaterialproben-Röhrchen



(b) Rack



(c) Schubladen



(d) Kühlschrank

Abbildung 1: Lagerung von Biomaterialproben aus medizinischen Studien

Anforderungen

Der zu entwickelnde Prototyp muss nicht unbedingt die gesamte gewünschte Funktionalität umfassen. Es müssen aber mindestens die Grundfunktionen realisiert werden. Bitte sprechen Sie den genauen Umfang Ihrer Software zu gegebener Zeit mit Ihrer*Ihrem Betreuer*in ab.

Grundfunktionen

Ihre Software muss alle der folgenden Funktionen unterstützen:

- Speicherung und Verwaltung von Biomaterialproben medizinischer Studien
 - Eine Biomaterialprobe entspricht einem Röhrchen mit (innerhalb einer Studie identifizierendem) Barcode (Abbildung 1a), das an einer bestimmten Position auf einem Rack gelagert wird (Abbildung 1b). Eine Biomaterialprobe hat neben dem Barcode ein Entnahmedatum, einen Materialtyp (Blut, Speichel, ...), eine Menge (2ml, 1g, ...), sowie einen Status (eingelagert, verbraucht, verschifft, verloren, ...).
 - Ein Rack ist dabei wie ein Feld beim Schiffversenken unterteilt. Es gibt Zeilen nummeriert mit Buchstaben und Spalten nummeriert mit Zahlen. Ein 10x10 Rack geht also beispielsweise von A1 bis J10. Außerdem haben Racks, wie Biomaterialproben, auch einen (innerhalb einer Studie eindeutigen) Barcode.
 - Racks können in Kühlschränken gelagert werden (Abbildung 1d). Dabei erhält ein Rack einen Stellplatz (grün) auf einer Schublade (rot). Mehrere Schubladen stecken übereinander in einem Gestell (gelb). Mehrere Gestelle bilden nebeneinander gruppiert ein Segment (blau). Ein Kühlschrank enthält mehrere Segmente.

- Kühlschränke stehen in Räumen.
- Anlegen und Verwalten von Benutzer*innen.
 - Die Identifikation von Benutzer*innen gegenüber dem System soll mit Name und Passwort erfolgen.
 - Neue Benutzer*innen mit ihren spezifischen Rollen sollen durch Personalleiter*innen registriert werden können. Nach der automatischen Vergabe eines initialen Passworts sollen die Benutzer*innen ihr Passwort selbstständig ändern können.
- Unterstützung der Biobank-Leiter*in-Funktionalität.
 - Kühlschränke und Räume im System müssen administriert werden. Es soll jeweils die Möglichkeit bestehen, sie hinzuzufügen, zu verändern (z.B. eingestellte Temperatur, Standort) und zu entfernen.
 - Biomaterialproben-Behältertypen (mit Volumen, Abmessungen, Deckel-Typ) und Biomaterialproben-Kategorien (Blut, Blut-Plasma, Speichel, ...) müssen definiert werden, bevor sie zur Erfassung von Biomaterialproben in entsprechenden Behältern verwendet werden können.
 - Studien sollen angelegt und verändert werden können. Hierzu gehört die Festsetzung des Namens der Studie (z.B. "Studie zur Feststellung eines statistischen Zusammenhangs zwischen Blutzuckerspiegel und Herzschlagfrequenz"), die gewünschte Anzahl der an der Studie Teilnehmenden, sowie die geplanten Visiten (i.d.R. zwischen 5 und 20). Zu jeder Visite sollte ein Name (z.B. "Zwischenvisite 3"), Informationen zu den zu erfassenden Daten ("Herzschlagfrequenz") und zu entnehmenden Biomaterialproben ("3x1ml Blut, 0.5ml Speichel") sowie der zeitliche Abstand im Bezug auf die anderen Visiten derselben Studie gespeichert werden.
- Unterstützung der Study Nurse-Funktionalität.
 - Patienten*innen müssen als Teilnehmer*innen an biomedizinischen Studien zunächst mit ihren persönlichen Daten (Name, Anschrift) erfasst werden.
 - Medizinische Daten, die während einer Visite erfasst werden (z.B. Blutdruck, Herzschlagfrequenz, ...), müssen in das System übertragen werden.
 - Der Status der Visiten muss überwacht werden, um beispielsweise Patientinnen und Patienten zu kontaktieren, die einen Visitentermin versäumt haben.
 - Study Nurses sollen dabei aus Datenschutzgründen nur Zugriff auf Daten erhalten, die zu Studien bzw. Visiten gehören, für die sie auch zuständig sind.
- Unterstützung der MTA-Funktionalität.
 - In Visiten entnommene Biomaterialproben müssen durch MTAs eingelagert werden. Dies beinhaltet den Platz für die Biomaterialproben abzufragen und zu belegen.
 - Es besteht die Möglichkeit, Biomaterialproben kurzfristig zu entnehmen, um sie zu testen und anschließend an den alten Platz zurückzustellen.

- Außerdem können Biomaterialproben dauerhaft entnommen werden. Der Platz kann dann ggf. für andere Biomaterialproben verwendet werden. Dies kann beispielsweise notwendig sein, wenn Biomaterialproben verarbeitet wurden oder wenn ein*e Patient*in die Einwilligung zur Teilnahme an einer Studie zurückgezogen hat und somit alle zugehörigen Proben entfernt werden müssen.
- Biomaterialproben, die durch Verarbeitung anderer Proben entstehen, müssen an einer neuen, sinnvollen Stelle (z.B. nahe der Ursprungsprobe) eingelagert werden. Dabei sollten Biomaterialproben derselben Studie gemeinsam z.B. in einem Segment oder (falls bspw. der Platz nicht ausreicht) in einem Kühlschrank gelagert werden. Ebenso sollten die Biomaterialproben eines*r Patienten*in soweit möglich auf demselben Rack eingeordnet werden.
- MTAs sollen aus Datenschutzgründen ausschließlich für sie relevante Informationen angezeigt bekommen und insbesondere niemals Zugriff auf Patientendaten (bspw. Name oder Adresse) haben.
- Speichern und vergleichen Sie Passwörter mit einem kryptographisch sicheren Verfahren:
 - Verwenden Sie den in der Java-Standard Library vorhanden kryptographischen Hashing-Algorithmus PBKDF2.
 - Verwenden Sie PBKDF2WithHmacSHA512 als aktuellen konkreten Hashing-Algorithmus.
 - Speichern Sie für jedes Passwort neben dem eigentlich Hash auch Salt, konkreten Hashing-Algorithmus und die Anzahl der Hashing-Rounds.
 - Als Hilfestellung können Sie die im Learnweb bereitgestellte Beispielapplikation (`PasswordHashing.java`) betrachten, modifizieren und Teile davon verwenden.

Zusätzliche Funktionen

Zusätzlich sollte Ihre Software ausgewählte weitere Funktionen unterstützen, die Ihnen sinnvoll erscheinen. Zum Beispiel:

- Möglichkeit den “Werdegang”/die “Chain of Custody” jeder Biomaterialprobe vollständig von Einlagerung über Verarbeitung bis Verbrauch/Entfernung nachzuvollziehen.
- Erweiterte Verwaltung von Biomaterialproben-Kategorien: Kategorien sollen hierarchisch organisiert sein und von der Biobankleitung dynamisch definiert werden können. Beispielsweise soll eine Oberkategorie “Blut” definiert werden können, zu der auch die spezifischere Unterkategorie “Plasma” gehören soll.
- Bieten Sie die Möglichkeit zusätzlich Anforderungen für Biomaterialproben (z.B. erlaubte Kühltemperatur, maximale Lagerdauer) zu erfassen und im System automatisch sicherzustellen. Außerdem sollte die Konsistenz des Lagersystems sichergestellt werden: Ein Kühlschrank mit einem Volumen von 100l kann nicht Biomaterialproben mit einem Gesamtvolumen von 200l beinhalten.

- Unterstützung der Wissenschaftler*innen-Funktionalität mit der Möglichkeit Anfragen nach Biomaterialproben für Forschungstätigkeiten zu stellen: Beispielsweise könnte eine Wissenschaftlerin besonders an Blutplasmaproben, die von Schlaganfallpatienten unter dem 50. Lebensjahr entnommen wurden, interessiert sein.
- Grafische Darstellung der Biomaterialproben/Rack/Kühlschrank-Organisation für die Nutzung durch MTAs.
- Intelligente Biomaterialproben-Zugriffspläne (“Picking-Listen”) für MTAs: Sinnvolle Sortierung der Liste von Biomaterialproben vor Entnahme. Beispielsweise sollten Biomaterialproben, die auf dem gleichen Rack gelagert sind, direkt nacheinander auf der Liste stehen. Dies kann beispielsweise hilfreich sein, wenn Biomaterialproben an ein externes Labor geschickt werden sollen.
- Unterstützung der Lager-Mitarbeiter*innen-Funktionalität mit der Möglichkeit Biomaterialproben zu reorganisieren, um Platz zu schaffen, beispielsweise: Biomaterialproben von mehreren bezüglich der Abmessungen kompatiblen wenig gefüllten Racks können auf einem Rack zusammengeführt werden. Ebenso können Racks aus verschiedenen kompatiblen Kühlschränken zusammengeführt werden, um Kühlschrank auszuschalten und Strom zu sparen. Diese Funktionalität kann insbesondere bei besonders großen Biobanken nützlich sein. Wird diese Funktionalität umgesetzt, kann die Beschränkung, dass Kühlschränke/Fächer immer einer Studie zugeordnet sind, aufgehoben werden.
- Simulation von Hardware-Interaktion mit Barcode-/QR-Code-Scanner z.B. über Webcam bei der Verwaltung der Biomaterialproben.
- Anzeige globaler Statistiken für Leiter*innen der Biobank: Anzahl freier Racks bzw. Kühlschränke, Anzahl Racks mit Belegung über bzw. unter einem bestimmten Grad. Ggf. automatischer Reorganisationsvorschlag. Die Statistiken sollen sowohl global als auch bezogen auf eine Studie einsehbar sein.
- Verschlüsselung der auf der Festplatte gespeicherten Daten.

Außerdem können Sie gerne weitere sinnvolle Funktionen realisieren, für die Sie Ideen haben.

Allgemeine Anforderungen

Neben den funktionalen Anforderungen gibt es weitere Anforderungen an das System:

- Verwenden Sie das in der Vorlesung vorgestellte Vorgehensmodell und beachten Sie die dort festgelegten Richtlinien für Dokumentation, Testen, Implementierung etc.
- Geben Sie die geforderten Zwischenabgaben zu den festgesetzten Terminen ab.
- Ihre Software soll in Java 13 geschrieben sein.
- Ihre Software soll eine graphische Benutzeroberfläche (GUI) haben.

- Die wichtigen Daten sollen zwischen dem Beenden und einem erneuten Start Ihrer Applikation persistent gehalten werden.
- Die Anwendung soll plattformunabhängig entworfen werden. Das Kompilieren, Deployment und vor allem Ausführen der JUnit-Tests und der Anwendung selbst soll *von Beginn an* von der Kommandozeile aus mit dem Tool “Gradle” funktionieren. Nachdem ein erster Prototyp Ihrer Applikation erstellt wurde, sollte eine Datei `README.md` im Wurzelverzeichnis Ihres `git`-Repositorys beschreiben (welche auch in der GitLab-Weboberfläche angezeigt wird), welche Kommandos hierfür jeweils nötig sind. Testen Sie dies auf den Linux-Rechnern des Fachbereichs. Achten Sie dabei insbesondere darauf, dass sich alle benötigten Bibliotheken **entweder** im Repository befinden **oder** in der `gradle.build`-Datei definiert sind. Insbesondere sollten Sie keine zusätzlichen (nur) von der IDE automatisch bereitgestellten Bibliotheken verwenden.

Ihr Projekt sollte letztendlich folgende Gradle-Tasks unterstützen unabhängig von der IDE unterstützen.

1. `compileJava`: Kompiliert die Anwendung
 2. `run`: Führt die Anwendung aus
 3. `compileTestJava`: Kompiliert die Tests
 4. `test`: Führt die Tests aus (Bericht in `build/reports/tests/test/index.html`)
 5. `build`: Führt einen vollständigen Build inklusive Tests aus
 6. `jar`: Erstellt eine jar in `build/libs`
 7. `assemble`: Erstellt einer ausführbaren Anwendung in `build/distributions`
- Die JUnit-Tests müssen, damit sie nach dem Einchecken einer neuen Version auf dem Server automatisch ausgeführt werden können, unabhängig von GUI-Komponenten sein. Achten Sie daher unbedingt darauf, die “Geschäftslogik“ ihrer Anwendung unabhängig von der View-Seite zu halten, damit sie erstere ausführlich automatisiert testen können.

Einschränkungen

Im Rahmen eines vierwöchigen Softwarepraktikums kann man keine vollständig ausgereifte Software mit „allen Schikanen“ entwickeln. Folgende Einschränkungen, die den Entwicklungsaufwand verringern, sollten für den zu entwickelnden Prototyp der Software gemacht werden. Falls Sie eine dieser Einschränkungen nicht machen wollen, halten Sie vorher bitte Rücksprache mit Ihren Betreuern.

- Die GUI soll ausschließlich mit JavaFX entworfen werden. Die GUI soll dabei von Hand, d.h. ohne die Verwendung eines GUI-Editors, erstellt werden.
- Für die Realisierung der Persistenz sollte der Serialisierungsmechanismus von Java oder die XStream-Bibliothek verwendet werden. Wenn Sie dennoch gerne eine Datenbank verwenden möchten, empfehlen wir OrmLite in Verbindung mit SQLite.

- Die Anwendung soll als Einzelplatzanwendung entworfen werden, d.h. der Zugriff erfolgt von einem einzelnen Terminal.

Hinweise zu den genannten Technologien finden Sie im Learnweb oder auf der entsprechenden Webseite des Softwarepraktikums ([http://www.uni-muenster.de/Informatik.
Becker/LiteraturSopra.html](http://www.uni-muenster.de/Informatik.Becker/LiteraturSopra.html)).

Warm-Up-Aufgabe

Diese Aufgabe dient zum Aufwärmen der Java-Kenntnisse. Sie soll von jedem Teilnehmer eigenständig bearbeitet werden.

Mitarbeiter eines konkurrierendes Softwareunternehmen haben bereits eine erste Version der Software geschrieben, bevor dem Unternehmen der Auftrag entzogen wurde. Es handelt sich um eine Softwarekomponente einer Java-Anwendung. Die Applikation soll dabei den Zustand eines Racks mit belegten und unbelegten Slots anzeigen können und es erlauben, programmatisch Röhrchen dem Rack hinzuzufügen.

Teil 1: Die erste Version der Software besteht aus einem `gradle`-Projekt mit einzigen Quellcode-Datei (`RackVerwaltung.java`).

- Clonen Sie Ihr persönliches Warmup-git-Repository. Dabei ist <Gruppe> ein Platzhalter für ihren Gruppennamen (gruppe01, gruppe02, ...) und <Kennung> ein Platzhalter für ihre Nutzerkennung (bspw. m_must01):

```
git clone "git@zivgitlab.uni-muenster.de:info-becker/\  
sopra_19_20_<Gruppe>/warmup_<Kennung>.git"
```

- Richten Sie das Build-Tool “Gradle” für ihr Warm-Up-Projekt ein:

- Richten Sie Gradle ein. Dafür wählen Sie **eine** von zwei Optionen:

- * Führen Sie das Script `gradlew` (Unix) bzw. `gradlew.bat` (Windows) einmal aus, um automatisch eine aktuelle Gradle Version herunterzuladen. **Achtung:** In diesem Fall ersetzt das Script den `gradle`-Befehl in allen folgenden Anweisungen. D.h. Sie führen beispielsweise `./gradlew test` bzw. `gradlew.bat test` anstatt `gradle test` aus, um die Anwendung auszuführen.
 - * (Optional:) Installieren Sie Gradle (etwa über den Paketmanager oder wie auf der Projektwebseite beschrieben).

- Das Projekt hat bereits folgende Tasks, die Sie mit `gradle <task>` aufrufen können. Testen Sie diese und machen Sie sich generell mit der Verwendung von Gradle vertraut.

- * `compileJava`, das ihr Projekt kompilieren soll.
 - * `run`: Ausführen der Anwendung
 - * `compileTestJava`: Kompilieren der Tests
 - * `test`: Ausführen der Tests (Bericht in `build/reports/tests/test/index.html`)
 - * `build`: Vollständiger Build inklusive Tests
 - * `jar`: Erstellen einer jar in `build/libs`
 - * `assemble`: Erstellen einer ausführbaren Anwendung in `build/distributions`
 - * `coverage`: Überprüft die Testabdeckung.

- Üben Sie den Umgang mit `git` und dem Webinterface der gitlab-Instanz.
- * Fügen Sie der `.gitignore`-Datei den `build` Ordner hinzu, damit build-Artefakte (die nämlich immer aus dem vorhandenen Quellcode generiert werden können) nicht dem Repository hinzugefügt werden.

- * Ergänzen die `.gitignore`-Datei um weitere temporäre Dateien, die ihr IDE während der Nutzung im Projektordner erstellt.
- * Der master-branch ihres persönlichen Repositories ist *protected*, d.h. es kann nicht direkt auf ihn gepusht werden. Erstellen Sie deshalb über das Web-Interface des ZIV-Gitlab-Servers (https://zivgitlab.uni-muenster.de/info-becker/sopra_19_20_<Gruppe>/warmup_<Kennung>/issues) ein *Issue*, das das Problem der fehlenden `.gitignore`-Einträge beschreibt. Erstellen Sie anschließend zu diesem Issue einen Merge-Request ebenfalls über das Web-Interface des ZIV-Gitlab-Servers und checken Sie den für Ihre Merge-Request erstellten Branch lokal aus:
`git checkout <branchname>`
- * Checken Sie Ihre Änderung an der `.gitignore`-Datei ein:
`git add .gitignore`
`git commit`
- * Laden Sie Ihre lokale Änderungen in den Branch im Repository hoch.
`git push origin <branchname>`
- * Mergen Sie Ihren Branch nach erfolgreichem Durchlauf der Tests über das Webinterface in den `master`-branch. Entfernen Sie dafür zunächst den "WIP-Status" im Webinterface und führen Sie den Merge durch, wenn die Test-Pipeline erfolgreich durchgelaufen ist.
Achtung: Im Gruppenprojekt soll das Mergen selbst durch eine/n anderen Teilnehmer*in nach Überprüfung durchgeführt werden. Da Sie die Warm-Up-Aufgabe aber selbstständig bearbeiten, dürfen und müssen Sie hier natürlich ausnahmsweise auch die Tester-/Code-Reviewer-Rolle übernehmen und Ihre Änderungen auch selbstständig mergen.

Teil 2: Leider ist der vorhandene Quellcode sehr unübersichtlich und fehlerhaft. Führen Sie zur Fehlerkorrektur und Anpassung an neue Anforderungen die folgenden Arbeitsschritte durch:

- Bearbeiten Sie diesen Teil der Aufgabe in mindestens einem separaten Issue korrespondierender Merge-Request, die sie nach Fertigstellung schließen bzw. in den `master`-branch mergen können.
- Verschieben Sie die Klassen `ProbenRoehrchenTyp` und `ProbenRoehrchen` in eigene Dateien. Stellen Sie sicher, dass das Projekt weiterhin erfolgreich kompiliert.
- Erstellen Sie JUnit-Tests (im Ordner `src/test/java` Ihres Gradle-Projekts), um die Fehler zu finden. Die erstellten Tests sollen für die fehlerhafte Funktionalitäten fehlschlagen. Ein erster, aber nur begrenzt hilfreicher Test wurde in `RackVerwaltungTest.java` bereits angelegt. **Achtung:** Die Klassennamen ggf. neu angelegter Testklassen müssen in "Test" oder "Tests" enden, damit sie von Gradle als solche erkannt werden.
- Kommentieren Sie den Code, sodass spätere Mitarbeiter nicht die gleichen Probleme haben wie Sie. Geben Sie dabei gemäß den Vorgaben aus der Vorlesung auch Pre- und Postconditions sowie Invarianten für die Methoden bzw. Klassen an.

- Beseitigen Sie die gefundenen Fehler und verbessern Sie gegebenenfalls die Implementierung. Sie können dabei die Methoden, Attribute und Variablen gerne umbenennen, doch soll der Code nur verbessert und NICHT neu geschrieben werden. Behalten Sie also die grobe Struktur der Klassen bei.
- Bereiten Sie die Verwendung der Klassen in einer größeren Applikation vor, indem Sie durch Methoden das Auslesen wichtiger Eigenschaften aller Klassen ermöglichen.

Teil 3: Die Anwendung soll weiterentwickelt werden. Bearbeiten Sie dazu folgende teils optionalen Aufgaben:

- Bearbeiten Sie auch diesen Teil der Aufgabe in mindestens einem separaten Issue und einem korrespondierender Merge-Request, den sie nach Fertigstellung schließen bzw. in den `master`-branch mergen können.
- Erstellen Sie eine grafische Oberfläche wie z.B. in Abbildung 2.
- Die Anwendung soll das interaktive Löschen und Hinzufügen von Probenrörchen in dem verwalteten Rack zulassen.
- Der Zustand des Racks soll beim Beenden der Anwendung nicht verloren gehen. Benutzen Sie einen Persistenzmechanismus, um diese Einstellungen zu speichern. Dabei genügt es, wenn die Daten beim Starten der Anwendung aus einer festen Datei geladen und beim Beenden der Anwendung in diese Datei zurückgeschrieben werden.
- Zusatz 1: Achten Sie auf gute Bedienbarkeit und Fehlerfreiheit. Zum Beispiel sollen bei der Spezifikation einer Menge nur sinnvolle Mengenangaben und bei der Angabe der Probenkategorie nur eine aus einer statischen Liste von Kategorien ausgewählt werden können.
- Zusatz 2: Ermöglichen Sie die Verwaltung mehrerer Racks. Dabei muss beachtet werden, dass jedes Rack einen unterschiedlichen Typus von Probenrörchen aufnehmen kann.

Hinweise

Die GUI soll ohne die Verwendung eines GUI-Editors erstellt werden. Wenn Sie nicht genau wissen, wie Sie bei der Realisierung der GUI vorgehen sollen, beachten Sie die folgenden Anregungen:

- Um die Anwendung in eine JavaFX (GUI-Anwendung) zu verwandeln, müssen sie Änderungen am Einstiegspunkt des Programms vornehmen. Ungewöhnlich ist in diesem Fall insbesondere, dass der Einstiegspunkt der Applikation nicht mehr eine statische `main`-Methode ist. Leiten Sie daher die Klasse, die die Haupt-Applikationslogik ihrer Anwendung enthält von `Application` ab. Überschreiben sie dort die Methode `public void start(Stage primaryStage)` als Einstiegspunkt für ihr Programm.

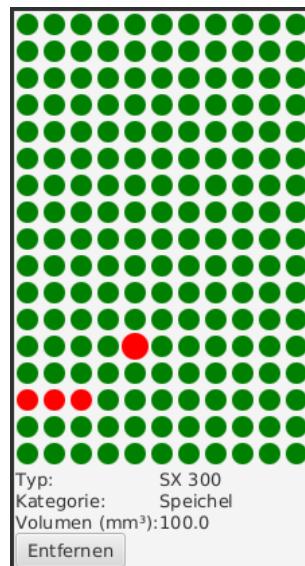


Abbildung 2: Beispiel einer Lösung der Warm-Up-Aufgabe.

- Die Datei ‘build.gradle‘ in ihrem Repository sollte bereits alle nötigen Einträge zur Verwendung von JavaFX aufweisen. Sollten Sie dennoch weitere Änderungen vornehmen wollen, beachten Sie bitte die folgenden Hinweise: <https://openjfx.io/openjfx-docs/#gradle>.
- Verwenden Sie **GridPanes** für die Darstellung der Probenröhrchen-Matrix.
- Verwenden Sie für die Definition eines Dialogs zum hinzufügen neuer Röhrchen einen **Dialog<...>** als Basis.
- Um das Speichern beim Beenden der Anwendung zu realisieren, müssen Sie die **stop** Methode ihrer von **Application** abgeleiteten Klasse überschreiben. Achten Sie darauf, dass Sie wirklich nur das Rack serialisieren und nicht (ungewollt) auch Teile der Komponentenstruktur der Anwendung. Dies kann passieren, wenn Sie das Datenmodell Referenzen auf andere Teile, beispielsweise der GUI-Komponente ihrer Anwendung hat.

Weitere Hinweise

- Die Warm-Up-Aufgabe soll plattformunabhängig gelöst werden, d.h. das Programm soll mindestens auf Linux, Mac und Windows lauffähig sein. Das häufigste Problem in diesem Zusammenhang ist die Verwendung von plattformspezifischen Pfadangaben, z.B. *C:\TEMP\DATA.SER*. Eine portable Entsprechung würde stattdessen z.B. auf die System-Property *java.io.tmpdir* Bezug nehmen oder relativ zum Ausführungsordner ./ Dateien ablegen.

Material

Das folgende Material kann bei der Bearbeitung der Warm-Up-Aufgabe hilfreich sein:

- Ein gutes Buch zur Java-Programmierung, z.B.:
 - *Handbuch der Java-Programmierung. Guido Krüger, Heiko Hansen.* Eine HTML-Version kann unter <http://javabuch.de/download.html> heruntergeladen werden. Es behandelt Java 8 und kann zu Java 13 Unterschiede enthalten.
 - *Java ist auch eine Insel. Ullensboom.* Eine Online-Version ist verfügbar unter: <http://openbook.rheinwerk-verlag.de/javainsel/>. Es behandelt Java 8 und kann zu Java 13 Unterschiede enthalten.
- JavaFX-Tutorial: <https://www.javatpoint.com/javafx-tutorial>
- Java-API Spezifikation: <http://docs.oracle.com/javase/13/docs/api/>

Zusätzliche Links finden Sie auf der Webseite des Softwarepraktikums bzw. im Learnweb.

Zeitplan

| | |
|--|---|
| Fr, 21.02.2020, 17:00 Uhr | Abgabe des Analysedokuments (Pflichtenheft) |
| Di, 25.02.2020 und ggf. Mi, 26.02.2020 | Präsentation und Besprechung des Analysedokuments |
| Di, 25.02.2020, 17:00 Uhr | Abgabe der Warm-Up-Aufgabe |
| Di, 03.03.2020, 17:00 Uhr | Erste Abgabe des Entwurfsdokuments und der Implementierung |
| Do, 05.03.2020, ab 9:00 Uhr | Feedback-Runde durch die Betreuer |
| Di, 10.03.2020, 17:00 Uhr | Abgabe des Überarbeiteten Entwurfsdokuments und der Implementierung, zudem Abgabe des Handbuchs |
| Mi+Do, 11+12.03.2020 | Vorbereitung der Abschlusspräsentation, Prüfungsgespräche und Einzelvorführungen in den Gruppen |
| Fr, 13.03.2020 | Abschlusspräsentation und Besuch der Abschlusspräsentationen anderer Gruppen |
| nach Absprache | Finale Feedback-Runde durch die Betreuer |

Weitere Informationen zur Zeitplanung finden Sie auf den Vorlesungsfolien. Die genaue Zeitplanung der Einzelvorführungen und Abschlusspräsentationen wird rechtzeitig auf der Webseite des Softwarepraktikums bzw. im Learnweb bekannt gegeben.