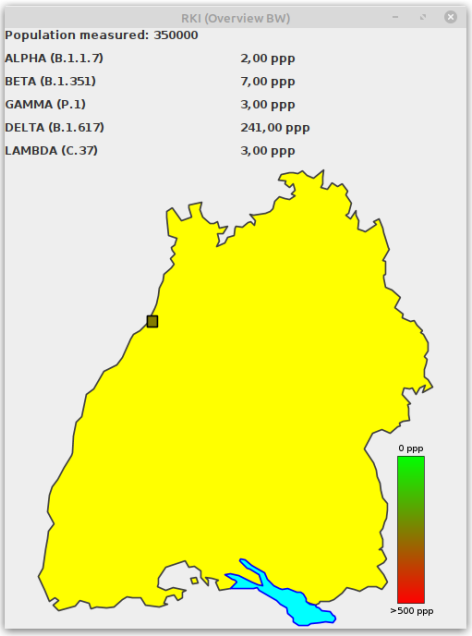
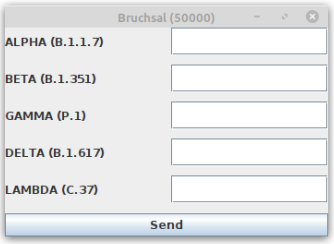
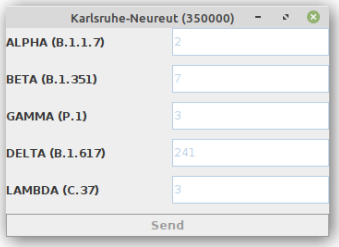
KLO COV-2

„Entscheidend ist, was hinten rauskommt.“ – H. Kohl







Die Grundidee dieser Prüfungsaufgabe basiert auf einem realen Forschungsprojekt.
Einige Detailaspekte sind jedoch frei erfunden 😉

GIB CORONA KEINE CHANCE

Hinweis zur Bewertung:

- 1/4 der Punkte (25%) wird nach Funktionstests Ihrer Lösung vergeben.
- 3/4 der Punkte (75%) werden entsprechend des in den Teilaufgaben angegebenen Schlüssels auf Basis des Quellcodes vergeben.

DHBW Karlsruhe, Vorlesung Programmieren I+II, Programmentwurf

15.10.2021

Bearbeitungszeit: 120 Minuten

Aufgabe

Um unabhängig von Bürgertests – deren Zahl mit steigender Impfquote ja immer mehr zurückgeht – einen Gesamtüberblick über das Infektionsgeschehen in Deutschland zu behalten soll im Rahmen eines Forschungsprojekts die Konzentration verschiedener Varianten von SARS-CoV-2 im Abwasser erfasst werden.

Zur Realisierung des Meldewesens für die Modellregion Baden-Württemberg sollen Sie daher eine grafische Java-Anwendung **`KloaCov2`** entwickeln, welche es den Betreibern von Kläranlagen ermöglicht, die Ergebnisse der Probenauswertung an das Robert-Koch-Institut (RKI) zu übermitteln.

Teilaufgabe a) **[5%]**

Um die verschiedenen Virusvarianten in unserem Programm unterscheiden zu können soll zunächst ein *komplexer Aufzählungstyp* **`Variant`** realisiert werden.

Der griechische Buchstabe, welcher der Variante zugewiesen wurde, ist hierbei der Name der Konstante. Darüber hinaus soll auch die korrekte wissenschaftliche Bezeichnung als Attribut (`designation`) erfasst werden.

Folgende fünf Varianten-Werte sind zu realisieren:

Variant	**Wiss. Bez.**	**Variant**	**Wiss. Bez.**	**Variant**	**Wiss. Bez.**
ALPHA	B.1.1.7	GAMMA	P.1	LAMBDA	C.37
BETA	B.1.351	DELTA	B.1.617		

Teilaufgabe b) **[5%]**

Damit im späteren Verlauf Elemente auch in einer Kartenansicht dargestellt werden können, soll eine Java-*Schnittstelle (Interface)* **`MapItem`** realisiert werden, welche für die Markierung von Karten-Elementen genutzt werden kann. Hierfür müssen folgende Methoden für spätere Implementierungen definiert werden:

- `String getTitle()`: soll den Titel des Karteneintrags zurückgeben
- `double getLongitude()`: soll den Längengrad des Karteneintrags zurückgeben
- `double getLatitude()`: soll den Breitengrad des Karteneintrags zurückgeben

Teilaufgabe c) **[12%]**

Zur Repräsentation eines Klärwerks soll die Klasse **`SewagePlant`** umgesetzt werden. Da die Klärwerke später auf einer Karte dargestellt werden sollen sind diese natürlich eine Ausprägung der Schnittstelle `MapItem`!

Ein Klärwerk soll seinen Namen (`name`), die (grobe) Anzahl der Personen, welche im Einzugsgebiet des Klärwerks leben, (`population`) sowie Längen- und Breitengrad (`longitude`/`latitude`) abbilden.

Ein zu definierender Konstruktor soll die Attribute in exakt dieser Reihenfolge entgegennehmen und initialisieren (vgl. Verwendung in bereitgestellter Klasse `KloaCov2`). Der Konstruktor soll auch überprüfen, ob die angegebenen Geo-Koordinaten in unserer Modellregion liegen. Falls sich der Längengrad (`longitude`) nicht zwischen 7 und 10 oder der Breitengrad (`latitude`) nicht zwischen 47 und 50 befindet soll eine *selbst zu schreibende* **`KloaCov2Exception`** geworfen werden, der als Nachricht die Information nach dem Schema „`Out of bounds: $longitute $latitude`“ übergeben wird (also bspw. „*Out of bounds: 12.5 51.4*“).

Als Titel des Karten-Eintrags (`getTitle`-Methode) soll der Name und die Personenzahl im Einzugsgebiet nach dem Muster *„`$name ($population people)`*“ zurückgegeben werden.

Teilaufgabe d) **[11%]**

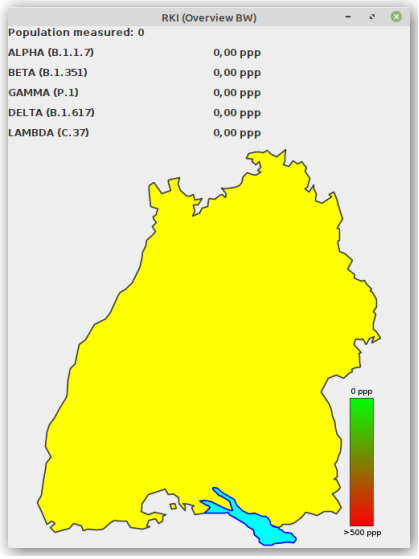
Natürlich sollen auch Messungen, welche in den Klärwerken vorgenommen werden, in unserer Applikation abgebildet werden können. Realisieren sie hierfür die Klasse **`VariantMeasurement`**, welche Messwerte für sämtliche Varianten von SARS-COV-2 (vgl. *Teilaufgabe a)*) aufnehmen kann.

Dafür sollen als Attribute das Klärwerk in welchem die Messung vorgenommen worden ist (`plant`), der Zeitpunkt der Messung (`date` vom Typ `java.util.Date`) sowie eine Datenstruktur zur Erfassung der Messwerte (`measurements`) abgebildet werden. Die Datenstruktur für die Messwerte soll Schlüssel-Wert-Paare speichern können, welche einer beliebigen Variante genau einen ganzzahligen Messwert zuweist. Die Messwerte werden in der (fiktiven) Einheit *parts per poo* (ppp) erfasst.

Definieren Sie einen geeigneten Konstruktor, welcher die jeweiligen Attribute entgegennimmt und/oder initialisiert.

Ebenso ist eine Methode `int getTotalValue()` zu definieren, welche die Summe der Messwerte für sämtliche Virus-Varianten dieser Messung berechnet und zurückgibt.

Teilaufgabe e) **[12%]**



Damit nach der Erfassung der Messwerte diese später auch übersichtlich dargestellt werden können, soll für das Robert-Koch-Institut als zuständige Behörde eine Art Dashboard bereitgestellt werden. Implementieren Sie hierfür die Klasse **`RKITerminal`**.

Das `RKITerminal` soll eine Schlüssel-Wert-Paar-Datenstruktur bereitstellen, die für jedes *Klärwerk* genau eine *Messung* (siehe *Teilaufgaben c)* und *d)*) speichern kann (`measurements`).

Der Titel ist entsprechend dem Screenshot zu setzen. Im oberen Bereich ist eine Anzeige vorzusehen, welche die Gesamtbevölkerung der mit Messwerten gespeicherten Klärwerke angibt.

Darüber hinaus ist für jede Virus-Variante (vgl. *Teilaufgabe a)*) ein Label mit Namen und wissenschaftlichem Bezeichner sowie der durchschnittliche Messwert anzugeben (siehe Hinweis 1!).

Ebenfalls ist eine Karte einzubinden. Die erforderliche Komponente wird Ihnen in Form der Klasse **`BWMap`** bereitgestellt (siehe USB-Stick).

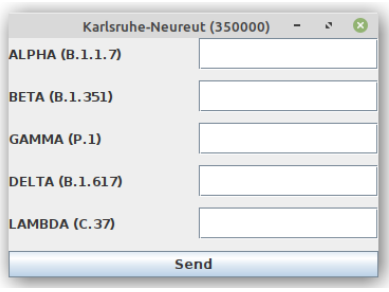
Damit Messungen erfasst werden können ist die Methode
`public void receiveMeasurement(VariantMeasurement meas)` zu realisieren, welche

- die übergebene Messung in der Schlüssel-Wert-Paar-Datenstruktur speichert (der Schlüssel ist das Klärwerk der Messung). Sollte für das Klärwerk bereits ein Wert existieren wird dieser einfach überschrieben.
- die neue Gesamtbevölkerung berechnet und das Label im oberen Bereich aktualisiert.
- die Durchschnittswerte für jede Virusvariante berechnet und die Anzeige entsprechend aktualisiert.
- der Karten-Komponente mit der Methode `setMapItem(MapItem item, int value)` den aktuellen Wert für das Klärwerk mitteilt. Zur Erinnerung: Klärwerke sind Ausprägungen der `MapItem`-Schnittstelle. Für den Messwert nutzen Sie die `getTotalValue`-Methode der Messung (vgl. *Teilaufgabe d)*).

Hinweis 1: Die gewählte Einheit (ppp) erlaubt es Ihnen, sämtliche Werte für jede Virus-Variante aufzusummieren und das Ergebnis durch die Anzahl der Klärwerke zu teilen. Es muss nicht nach Bevölkerung gewichtet werden!

Hinweis 2: Die Funktionalität der Methode `receiveMeasurement` wird in Teilaufgabe g) erweitert

Teilaufgabe f) **[15%]**



Realisieren Sie nun eine Klasse **`SewageTerminal`**, welche eine grafische Benutzeroberfläche zur Erfassung der Messwerte in einem Klärwerk bereitstellt (vgl. Screenshot rechts).

Die Klasse muss (mindestens) folgende Instanz-Variablen vorsehen:

- das Klärwerk (`plant`) für welches die Werte erfasst werden
- eine Referenz auf ein `RKITerminal` (`rki`), welchem die Messwerte später mitgeteilt werden

Der Konstruktor der Klasse nimmt die beiden Attribute in dieser Reihenfolge entgegen (vgl. Verwendung in bereitgestellter Klasse `KloaCov2`) und initialisiert die Instanzvariablen.

Für jede der Virusvarianten (vgl. *Teilaufgabe a)*) ist - wie auf dem Screenshot zu sehen - sowohl ein Label mit dem Namen und der wiss. Bezeichnung sowie ein Eingabefeld für den Messwert vorzusehen.

Darunter soll ein Button „`Send`“ zum Übermitteln der Werte vorgesehen werden. Beim Drücken des Buttons soll für jede Virusvariante der Messwert ausgelesen werden und in einer Schlüssel-Wert-Paar-Datenstruktur abgelegt werden.

Tritt hierbei ein Fehler auf (bspw. keine gütige Ganzzahl angegeben, das Feld ist leer, …) ist der Nutzer darüber in einem Dialog zu informieren. Es muss auch angegeben werden bei welcher der Virusvarianten der Eingabefehler erkannt wurde und wie der falsche Wert lautet (vgl. Screenshot unten). Es müssen jedoch nicht alle Fehler angezeigt werden, nach dem ersten darf die Operation angebrochen werden.

Konnten alle Werte erfolgreich ausgelesen werden ist basierend auf dem Klärwerk dieses Fensters, dem aktuellen Datum und Uhrzeit (siehe Hinweis 1) sowie den ausgelesenen Messwerten eine Instanz von `VariantMeasurement` zu erzeugen und dem `RKITerminal` mit der Methode `receiveMeasurement` zu übermitteln. Darüber hinaus sind sämtliche Eingabefelder sowie der Button zu deaktivieren (siehe Hinweis 2).

Hinweis 1: Eine Date-Instanz mit aktuellem Datum/Uhrzeit können Sie mit `new java.util.Date()` erzeugen

Hinweis 2: UI-Elemente können mit `setEnabled(false)` deaktiviert bzw. mit `setEnabled(true)` aktiviert werden

Hinweis 3: Die Funktionalität wird in Teilaufgabe i) erweitert

Teilaufgabe g) **[5%]**

Ändern Sie die Implementierung der `loadPlants`-Methode in der bereitgestellten Klasse `KloaCov2` (s. USB-Stick) nun so, dass die ebenfalls bereitgestellte Datei „`plants.txt`“ (s. USB-Stick) zeilenweise eingelesen wird. Fehler müssen zwar abgefangen aber nicht weiter behandelt werden.

Dabei ist jede Zeile der Datei in ein `SewagePlant`-Objekt umzuwandeln und anstatt der Beispieldaten der Liste der Klärwerke hinzuzufügen. Nutzen Sie für das Konvertieren in `SewagePlant` -Objekte die bereitgestellte Methode `parsePlant`, welche bereits für die Umwandlung der Beispieldaten genutzt wird. Aus der eingelesenen Gesamtliste sind (damit es nicht zu unübersichtlich wird 😊) *drei zufällig ausgewählte* Klärwerke zurückzugeben.

Teilaufgabe h) **[5%]**

Erweitern Sie die `receiveMeasurements`-Methode von `RKITerminal` (siehe *Teilaufgabe e)*) nun so, dass die Informationen der empfangenen Messung auch in eine Textdatei mit dem Namen „`ppp-logs.txt`“ gespeichert werden. Sollte die Textdatei bereits existieren, ist die Nachricht als neue Zeile an das Ende anzuhängen. Fehler müssen zwar abgefangen aber nicht weiter behandelt werden.

Es sollen der Name der Kläranlage, der Messzeitungpunkt sowie die Messwerte in der Zeile stehen. Für die `Date`-Instanz sowie für die Datenstruktur der Messwerte kann deren `toString`-Methode verwendet werden. Beispielzeile:

```
Karlsruhe-Neureut (Wed Sep 15 21:41:23 CEST 2021): {DELTA=36, ALPHA=2, BETA=5, LAMBDA=1, GAMMA=7}
```

Teilaufgabe i) **[5%]**

Erweitern Sie die Klasse `SewageTerminal` (vgl. *Teilaufgabe f)*) nun so, dass nach einer Wartezeit von 10 Sekunden der Button und die Textfelder wieder aktiviert werden (siehe *Hinweis 2, Teilaufgabe f)*). Der Inhalt der Textfelder ist zu leeren.

Diese Aktion soll *nebenläufig* durchgeführt werden. Die nebenläufige Ausführung endet, nachdem Buttons und Textfelder wieder aktiv sind.

Allgemeine Hinweise

Starten
Starten Sie die Anwendung mit der gegebenen Klasse `KloaCov2` (siehe USB-Stick).

Schließen eines Fensters
Beim Schließen eines Fensters soll die komplette Anwendung beendet werden.

Sichtbarkeit von Instanz-Attributen
Sämtliche Instanz-Attribute sind als privat zu definieren und von außerhalb der Klasse ggf. mittels Getter- und/oder Setter-Methoden zu verwenden.