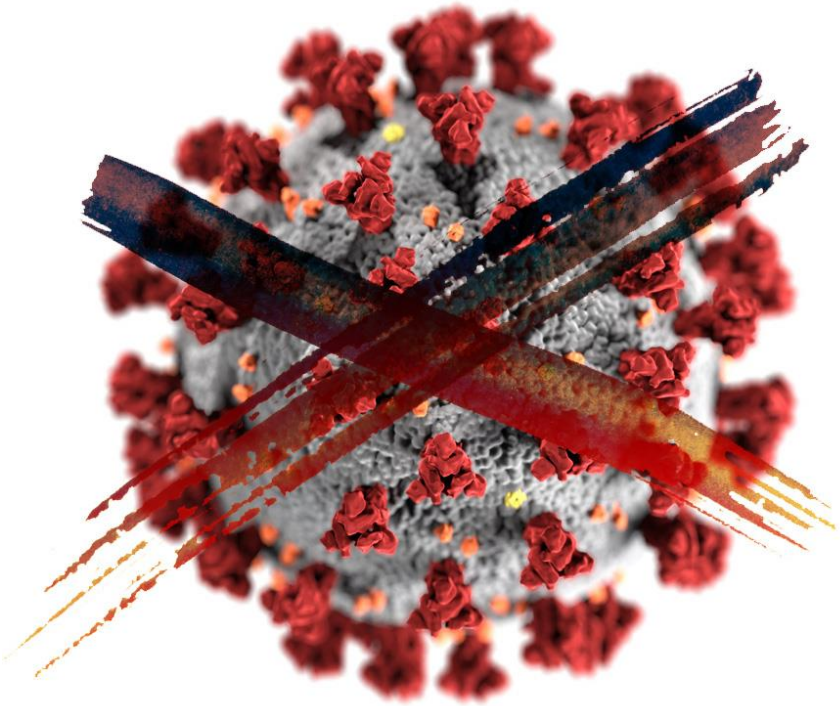
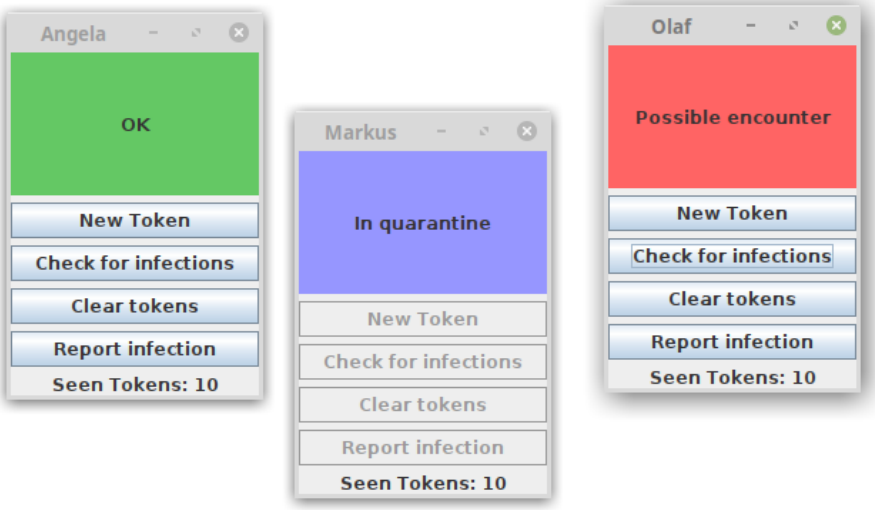
corona warn





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

09.10.2020

Bearbeitungszeit: 120 Minuten

Aufgabe

Wie aus maximal seriöser und glaubwürdiger Quelle[1] bekannt, wird aktuell am Update von COVID-19 auf COVID-20 gearbeitet. Zur Kompatibilität mit der neuen Version muss auch die Corona-Warn-App aktualisiert werden 😉.

Schreiben Sie daher eine grafische Java-Anwendung **`CoronaWarn`**, welche angelehnt an die Funktionalität der aktuellen App das Verfolgen von Infektionsketten auch nach dem COVID-20-Update ermöglichen soll. Auch unsere Variante soll weitgehend dem dezentralen, Token-basierten Ansatz folgen – auch wenn einige Aspekte zwecks vereinfachter Umsetzung „zentralisiert" wurden.

Ähnlich wie bei iOS- und Android-Geräten mit Google-Diensten wird für unsere „`JPhone`-Welt" eine API für das Verteilen und Abfragen der Tokens bereitgestellt, realisiert durch die Klasse **`CoronaWarnAPI`** (s. USB-Stick).

Teilaufgabe a) **[15%]**

Schreiben Sie eine Klasse **`JPhone`** zur Repräsentation eines Mobilgeräts, auf welchem unsere Anwendung läuft. Ein `JPhone`-Objekt kann sowohl eine ID (`id`) als auch einen Besitzer (`owner`) als Zeichenketten speichern. Sein Konstruktor nimmt für beide Attribute die Werte entgegen und initialisiert diese (vgl. Verwendung in Klasse `CoronaWarn`, siehe USB-Stick).

Für die Verfolgung von Infektionsketten sollen zufällig generierte, wechselnde Codes verwendet werden. Implementieren Sie hierfür die Klasse **`Token`**. Ein Token enthält eine Zeichenkette als Wert (`value`) sowie den Zeitpunkt der Generierung (`date`, vom Typ `java.util.Date`).

Die Klasse `Token` soll **zwei Konstruktoren** zur Verfügung stellen:

- Einen argumentlosen Konstruktor, der `value` mit einer zufälligen Zeichenkette und `date` mit dem aktuellen Zeitpunkt befüllt (beachten Sie hierzu die Hinweise unten!).
- Einen Konstruktor, der beide Attribute entgegennimmt und ihre Werte entsprechend den Instanz-Variablen zuweist (vgl. Verwendung in Methode `parseToken` der Klasse `CoronaWarn`, siehe USB-Stick).

Auch soll eine `toString`-Methode realisiert werden, die die Werte beider Attribute hintereinander hängt (vgl. Tooltip-Screenshot am Ende von *Teilaufgabe c)*), nutzen Sie dabei für die `Date`-Instanz einfach deren `toString`-Methode).

Darüber hinaus entwickeln Sie zur Repräsentation der möglichen Gefahrensituationen den *komplexen Aufzählungstyp* **`WarnStatus`**. Dieser speichert pro `WarnStatus`-Wert sowohl einen Statustext für die Ausgabe (`text`) als auch eine Farbe (`color`) für die spätere Anzeige. Folgende Status-Werte sind zu implementieren:

WarnStatus	Text	Color	WarnStatus	Text	Color
`UNKNOWN`	`Unknown`	`new Color(175,175,175)`	`ALARM`	`Possible encounter`	`new Color(255,100,100)`
`OK`	`Ok`	`new Color(100,200,100)`	`INFECTED`	`In quarantine`	`new Color(150,150,255)`

Hinweise: Ein `Date`-Objekt mit aktuellem Datum/Uhrzeit können Sie mit `new Date()` erzeugen. Eine zufällige Zeichenkette für den Wert des Tokens lässt sich mit `java.util.UUID.randomUUID().toString()` generieren.

Teilaufgabe b) **[5%]**

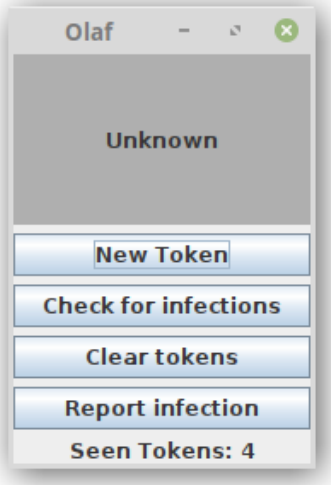
Um potenziell verschiedene Clients realisieren zu können, soll zunächst die *Java-**Schnittstelle*** **`CoronaWarnClient`** deklariert werden. Die Schnittstelle muss folgende Methoden für spätere Implementierungen definieren:

- `Token getCurrentToken()`: soll das aktuell für den Client gültige Token zurückliefern
- `List<Token> getAllTokens()`: soll die Liste sämtlicher Tokens liefern, die für diesen Client bisher verwendet wurden. Die Liste darf auch das aktuell gültige Token beinhalten, dies ist jedoch nicht notwendig.
- `List<Token> getAllSeenTokens()`: soll die Liste sämtlicher Tokens liefern, die dieser Client als „in der Nähe" bisher mitgeteilt bekam
- `void tokenReceived(Token token)`: dient dazu, dem aktuellen Client mitzuteilen, dass „in seiner Nähe" dieses Token ausgesendet wurde.

[1] https://www.der-postillon.com/2020/05/covid-20.html *(bei Interesse erst nach der Prüfung anschauen!)*

Teilaufgabe c) **[17%]**

Realisieren Sie nun eine Klasse **`CoronaWarnTerm`**, welche eine grafische Benutzeroberfläche zur Bedienung unserer Corona-Warn-App bereitstellt (vgl. Screenshot rechts). Selbstverständlich ist diese Klasse eine Implementierung der `CoronaWarnClient`-Schnittstelle und realisiert die Anforderungen an die jeweiligen Schnittstellen-Methoden wie in *Teilaufgabe b)* beschrieben.



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

- Eine Referenz auf die `JPhone`-Instanz für die Benutzeroberfläche.
- Eine Möglichkeit zum Speichern der aktuellen Gefahrensituation. Initial ist diese auf `WarnStatus.UNKNOWN` zu setzen.
- Eine Datenstruktur, um sämtliche Tokens speichern zu können, die der Klasse als „in der Nähe“ mitgeteilt werden (vgl. *Teilaufgabe b)*)
- Eine Datenstruktur zum Speichern der wechselnden, eigenen Tokens. Initialisieren Sie diese mit den Werten, die Ihnen `CoronaWarn.loadTokens` für die übergebene `JPhone`-Instanz liefert.
- Auch das zuletzt erzeugte, d.h. aktuell gültige Token muss verfügbar sein.

Der Konstruktor der Klasse nimmt die `JPhone`-Instanz für den `CoronaWarnTerm` entgegen (vgl. Verwendung in `main`-Methode der Klasse `CoronaWarn`, s. USB-Stick) und speichert sie in der Instanz-Variable. Der Titel des Fensters ist auf den Besitzer des Telefons (vgl. *Teilaufgabe a)*) zu setzen. Ebenso soll ein initiales, zufälliges Token erzeugt und in der entsprechenden Instanz-Variablen der Klasse sowie auch für das Telefon in einer Datei gespeichert werden (Aufruf der Methode `CoronaWarn.saveToken` mit den entsprechenden Parametern, vgl. *Teilaufgabe e)*).

Nachdem das Token erzeugt und entsprechend gespeichert wurde, ist der bereitgestellten `CoronaWarnAPI` (s. USB-Stick) mit dem Aufruf `CoronaWarnAPI.sendToken(this)` mitzuteilen, dass ein neues Token zu versenden ist.

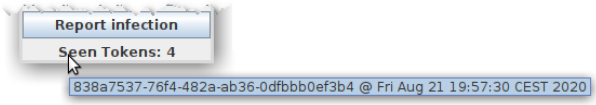
Hinweis: diese Funktionalität wird in Teilaufgabe d) und g) erweitert!

Für die Benutzeroberfläche ist im oberen Bereich ein Label vorzusehen, welches die aktuelle Gefahrensituation widerspiegeln soll, d.h. die Hintergrundfarbe und Beschriftung immer der Farbe und dem Statustext der gespeicherten Gefahrensituation entspricht.

Hinweise: Damit der Hintergrund eines Labels angezeigt wird, müssen Sie `mylabel.setOpaque(true)` aufrufen. Zum Festlegen einer Höhe, um wie im Screenshot oberhalb und unterhalb des Label-Texts noch einen farbigen Bereich zu haben, können Sie `mylabel.setPreferredSize(new java.awt.Dimension(0,100))` (für 100px Höhe) nutzen. Die zentrierte Darstellung innerhalb eines Labels erreicht man durch Aufruf von `mylabel.setHorizontalAlignment(JLabel.CENTER)`.

Darunter sind vier Buttons für alle Interaktionen, die unsere Applikation bereitstellen soll, vorzusehen (vgl. Screenshot und Beschreibung in *Teilaufgabe d)*). Interaktionen müssen hier noch nicht realisiert werden.

Am unteren Rand ist ein Label vorzusehen, welches anzeigt, wie viele Tokens dem Client bereits als „in der Nähe“ mitgeteilt wurden. Dieses soll auch als Tooltip (vgl. Screenshot) jeweils das aktuelle Token des Mobilgeräts anzeigen (nutzen Sie für den Tooltip-Text die `toString`-Methode, welche Sie in *Teilaufgabe a)* umgesetzt haben.).



Hinweis: Tooltips können Sie mit `mylabel.setToolTipText(..)` setzen.

Teilaufgabe d) **[15%]**

Die Buttons der grafischen Benutzeroberfläche sollen nun um Interaktivität erweitert werden:

- Der Button „New Token“ soll, wenn er gedrückt wird, ein neues Token erstellen. Das zuvor gültige Token muss natürlich in der Datenstruktur, die alle bisherigen Tokens speichert, abgelegt sein. Nachdem das neue Token erzeugt wurde ist dies zu speichern und der `CoronaWarnAPI` mitzuteilen (analog zur Erzeugung in *Teilaufgabe c)*)! Auch der Tooltip des Labels am unteren Rand ist zu aktualisieren.
  Hinweis: diese Funktionalität wird in Teilaufgabe g) nochmals erweitert!
- Der Button zum Überprüfen, ob es eine Begegnung mit als infiziert gemeldeten Tokens gab („Check for infections“), soll eine Überprüfung mittels `CoronaWarnAPI.checkInfection(this)` durchführen. Liefert der Aufruf **`true`** ist der aktuelle Gefahrenstatus auf `WarnStatus.ALARM` zu setzten, sonst auf `WarnStatus.OK`. Das Label im oberen Bereich ist entsprechend zu aktualisierten (sowohl Hintergrundfarbe als auch der Statustext, vgl. Deckblatt).

- Der Button zum „Vergessen“ sämtlicher Tokens („Clear Tokens“) soll
    - sämtliche Tokens aus Attributen/Datenstrukturen mit bisherigen eigenen Tokens entfernen
    - sämtliche Tokens aus der Datenstruktur für die als „in der Nähe“ gemeldeten Tokens entfernen
    - die Methode `CoronaWarn.clearTokenStore` mit der gespeicherten `JPhone`-Instanz aufrufen
    - zuletzt ein neues Token erzeugen, inkl. aller Aktionen wie beim „New Token“-Button beschrieben
- Der Button „Report infection“ soll der Warn-API ein eigenes, positives Test-Ergebnis mit einem Aufruf von `CoronaWarnAPI.reportInfection(this)` mitteilen, ebenso sämtliche Buttons deaktivieren und den eigenen Status auf `WarnStatus.INFECTED` setzen. Das Label im oberen Bereich ist entsprechend zu aktualisierten (Hintergrundfarbe & Statustext, vgl. Deckblatt).

Teilaufgabe e) **[8%]**

Erweitern Sie die Methode `saveToken` der Klasse `CoronaWarn` (s. USB-Stick) so, dass das übergebene Token für das übergebene `JPhone` gespeichert wird. Um die verschiedenen „Telefonspeicher“ zu simulieren, ist ein Dateiname auf Basis der ID der übergebenen `JPhone`-Instanz zu wählen, bspw. mit dem Muster *`$phoneId`*`-tokens.txt` (konkretes Bsp.: `0123-4567-tokens.txt`).

Die in der Datei zu speichernde Zeile (die für das spätere Einlesen mit der Methode `CoronaWarn.parseToken` kompatibel sein muss) ist in der vorgegebenen Variable `line` innerhalb von `CoronaWarn.saveToken` bereits basierend auf dem übergebenen `Token` für Sie vorbereitet. Sollte die Datei bereits existieren, ist eine neue Zeile an das Ende anzufügen. Sollten hierbei Fehler auftreten, zeigen Sie eine entsprechende Nachricht in einem Dialog an.

Teilaufgabe f) **[8%]**

Ändern Sie in der bereitgestellten Klasse `CoronaWarn` (s. USB-Stick) die Implementierung der Methode `loadTokens` so, dass die Tokens aus einer Datei eingelesen werden. Nutzen Sie denselben Dateinamen basierend auf der übergebenen `JPhone`-Instanz wie in *Teilaufgabe e)*!

Jede Zeile der Datei ist in ein `Token`-Objekt umzuwandeln und der Liste der Tokens hinzuzufügen. Nutzen Sie für das Umwandeln in `Token`-Objekte die bereitgestellte Methode `parseToken`, welche bereits für die Umwandlung der Beispieldaten genutzt wird.

Sollte die Datei noch nicht existieren oder während des Lesens ein Fehler auftreten, ist eine leere Liste zurückzugeben. Fehler während des Lesens sind abzufangen, müssen jedoch nicht weiter behandelt werden.

Erweitern Sie ebenfalls die Methode `CoronaWarn.clearTokenStore` so, dass die Datei, welche Sie in `loadTokens`/`saveToken` verwendet haben gelöscht oder mit leerem Inhalt überschrieben wird.

Teilaufgabe g) **[7%]**

Erweitern Sie die Klasse `CoronaWarnTerm` nun um eine *automatische* Erneuerung der Tokens!

Hierfür soll in jeder Instanz alle 5 Sekunden ein neues Token erzeugt werden, inkl. aller Aktionen wie beim „New Token“-Button (vgl. *Teilaufgabe d)*) beschrieben.

Diese Aktion soll *nebenläufig* durchgeführt werden. Die nebenläufige Ausführung endet für den aktuellen `CoronaWarnTerm`, wenn dieser eine Infektion meldet und somit der Zustand `WarnStatus.INFECTED` erreicht ist.

Allgemeine Hinweise

Starten
Starten Sie die Anwendung mit der gegebenen Klasse `CoronaWarn` (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.

`this`-Referenz in anonymer ActionListener-Instanz
Sollten Sie die `CoronaWarmAPI`-Aufrufe innerhalb von `CoronaWarnTerm` in einer anonymen `ActionListener`-Instanz tätigen, müssen Sie statt `this` die Referenz `CoronaWarnTerm.this` als Methoden-Parameter verwenden!