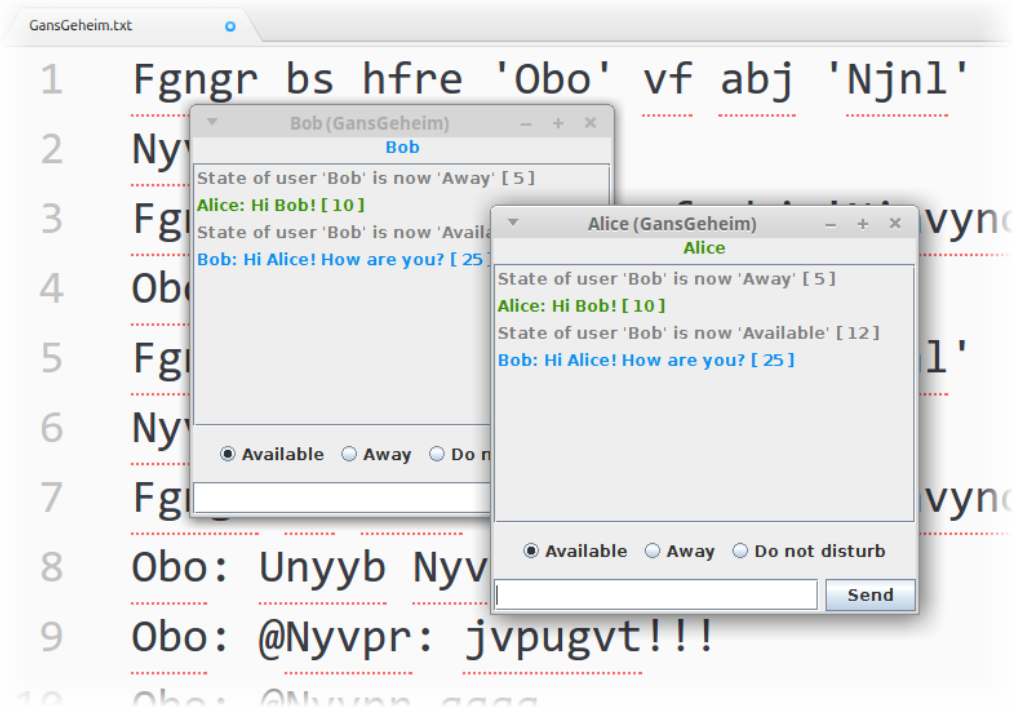


Download Zusatzdateien:

`https://www.iai.kit.edu/javavl/pe/uebung2021.zip`

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

Übungsprogrammentwurf Oktober 2021

Bearbeitungszeit: 120 Minuten

Aufgabe

Schreiben Sie eine Java-Anwendung „*SnatChat*“, die einen Messenger für „geheime“ Kurznachrichten implementiert!

Nachrichten bei *SnatChat* zeichnen sich dadurch aus, dass sie für die Empfänger nur für kurze Zeit lesbar sind und sich danach – ähnlich wie in *Mission: Impossible* – „selbst zerstören“.

Der Nachrichtenaustausch ist über eine spezielle, Java-basierte Nutzeroberfläche „*SnatChatWindow“* möglich. Im Hintergrund nutzen alle *SnatChatWindow*s zur Kommunikation einen *SnatChatRoom* um bspw. Nachrichten an alle Nutzer zu verteilen.

Teilaufgabe a) **[5%]**

Durch den absehbaren, enormen Erfolg von *SnatChat* (Kurznachrichtendienste sind ein kaum besetztes Geschäftsfeld!) soll zunächst die Funktionalität für Frontends (wie *SnatChatWindow*) über das Java-*Interface* **`SnatChatFrontend`** definiert werden. Das Interface hat die folgenden Methoden:

- **`public void receiveMessage(Message msg)`**
- **`public void receiveMessage(String text)`**
- **`public Account getAccount()`**

Teilaufgabe b) **[10%]**

Als Datenobjekte nutzt *SnatChat* Benutzerkonten (Klasse **`Account`**) sowie Nachrichten (Klasse **`Message`**).

Schreiben Sie zunächst einen *komplexen Aufzählungstyp* **`State`**, mit den möglichen Werten **`AVAILABLE`**, **`AWAY`** und **`DND`** sowie den passenden, sprechenden Labels „Available“, „Away“ und „Do not disturb“!

Schreiben Sie eine Klasse `Account`, die zu jedem Konto den Benutzernamen (`String name`), den aktuellen Status (`State state`) und eine Farbe für dessen Nachrichten (`Color color`) speichert! Dabei soll der Name im Konstruktor übergeben und die Farbe für den Account *zufällig und nicht zu hell (s. Hinweis)* erzeugt werden. Initial ist jeder Account im Zustand `State.AVAILABLE`.

Schreiben Sie eine Klasse **`Message`**, die zu jeder Nachricht deren Inhalt (`text` vom Typ `String`) sowie eine Referenz auf den Absender (`Account`-Instanz) speichert! Der Konstruktor von `Message` soll das Setzen beider Attribute ermöglichen.

Machen Sie die Attribute beider Klassen „privat“ und über Getter/Setter-Methoden verfügbar!

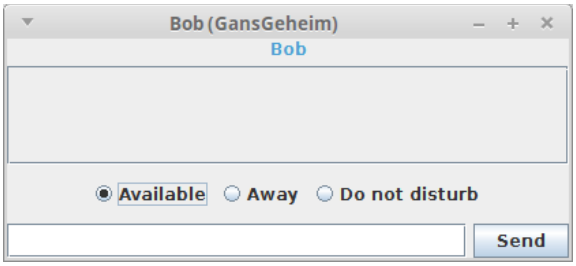
Teilaufgabe c) **[10%]**

Entwickeln Sie eine Klasse **`SnatChatRoom`** mit (mindestens) folgenden Methoden:

- **`public String getRoomName()`** ermöglicht das Auslesen des Raum-Namens. Der Name des Raums soll in einer Instanz-Variablen (`String roomName`) gespeichert und im Konstruktor von `SnatChatRoom` übergeben werden!
- **`public void register(SnatChatFrontend s)`** ermöglicht die Anmeldung eines Frontends (welches das Interface `SnatChatFrontend` implementiert).
- **`public void unregister(SnatChatFrontend s)`** ermöglicht das Abmelden eines Frontends.
- **`public void sendMessage(Message msg)`** schickt eine Nachricht `msg` an alle angemeldeten Frontends.
- **`public void sendMessage(String text)`** schickt den Nachrichtentext `text` an alle angemeldeten Frontends.

Teilaufgabe d) **[10%]**

Schreiben Sie eine Klasse **`SnatChatWindow`**, welche das Interface `SnatChatFrontend` implementiert und eine grafische Benutzeroberfläche in einem eigenen Fenster erzeugt!



In einem Konstruktor soll dem `SnatChatWindow` ein `SnatChatRoom`-Objekt sowie ein `Account` übergeben werden. *(vgl. Beispiel-Code zum Starten / Hinweise)*

Die Benutzeroberfläche eines `SnatChatWindow`s besteht aus der Titelzeile (Format: „*Benutzername (Raumname)*“), dem Benutzernamen in der entsprechenden Farbe des Accounts innerhalb des Fensters, einer `ChatMessagesComponent` (Klasse bereitgestellt auf USB-Stick) für die Anzeige der Nachrichten, Radio-Buttons für die Auswahl des aktuellen Status (der richtige Status ist passend zu der übergebenen Account-Instanz zu selektieren), einem einzeiligen Feld zur Eingabe von Nachrichten („Eingabe“, zunächst leer) und einem Button „Send“ zum Absenden der Nachrichten (zunächst keine Aktion, wird später ergänzt).

Mit Hilfe der bereitgestellten `ChatMessagesComponent` sollen beim Aufruf von…

- … **`receiveMessage(String)`** eine Nachricht mit dem übergebenen Text und der Textfarbe Grau (`java.awt.Color.GRAY`) hinzugefügt werden.
- … **`receiveMessage(Message)`** eine Nachricht in der Form „Absender: Nachrichtentext“ mit der Farbe des Absenders als Textfarbe hinzugefügt werden.

Hinweis: In Teilaufgabe g) wird diese Funktionalität erweitert!

Teilaufgabe e) **[10%]**

Erweitern Sie die Klasse `SnatChatWindow` so, dass sie auf Benutzerinteraktionen reagiert:

- Bei der Auswahl eines RadioButtons soll eine einfache Text-Nachricht in der Form „State of user *'username'* is now *'statelabel'*“ verschickt werden (vgl. Abbildung Vorderseite) sowie der Status im `Account`-Objekt aktualisiert werden.
- Beim Drücken des Buttons „Send“ soll…
  - … bei leerem Nachrichtenfeld ein Dialog erscheinen, der den aktuellen Benutzer darauf hinweist, dass eine Nachricht eingeben muss. *(vgl. Abbildung unten)*
  - … bei gefülltem Nachrichtenfeld ein Nachrichten-Objekt mit dem Inhalt des Textfelds als Nachrichtentext und dem `Account` der `SnatChatWindow`-Instanz als Absender generiert und verschickt werden. Im Anschluss soll das Textfeld geleert werden.
- **1 Zusatzpunkt**: Auch das Drücken der Enter-Taste im Textfeld löst das Abschicken aus.

Teilaufgabe f) **[10%]**

Erweitern Sie die Klasse `SnatChatRoom` so, dass alle Nachrichten, die über den Raum verschickt werden, auch in eine Textdatei geloggt werden. Hierfür soll jede Nachricht in eine Zeile der Datei „*`raumname`*`.txt`“ geschrieben werden (also „`GansGeheim.txt`“ für den Raumnamen aus der bereitgestellten main-Methode). Besteht die Datei bereits, sollen weitere Nachrichten angehängt werden.

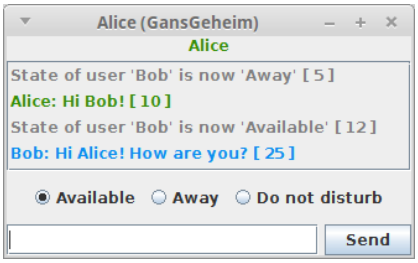
Darüber hinaus soll beim Registrieren einer `SnatChatFrontend`-Instanz die Textdatei eingelesen werden und jede Nachricht als einfache Text-Nachricht direkt an das Frontend geschickt werden.

Erweitern Sie die Einlese-Routine für die Log-Datei anschließend so, dass nur maximal die letzten 10 Nachrichten/Zeilen an das Frontend geschickt werden, nicht die komplette Historie.

Teilaufgabe g) **[10%]**

Der Nachrichtendienst *SnatChat* legt großen Wert auf Vertraulichkeit. Erweitern Sie die Klasse `SnatChatWindow` daher so, dass eingehende Nachrichten nach 30 Sekunden automatisch aus der Nachrichtenliste entfernt werden.

Für jede Nachricht soll rechts des Nachrichtentextes ein Countdown heruntergezählt werden, wie viele Sekunden die Nachricht noch zu sehen ist.



Teilaufgabe h) **[10%]**

Vertraulichkeit ist der Markenkern von *SnatChat* – daher sollen die Log-Dateien nicht im Klartext gespeichert, sondern die Nachrichten verschlüsselt abgelegt werden. Hierfür wurde der Algorithmus „`rot13`“ ausgewählt. Implementieren Sie hierfür in der Klasse `Message` eine statische Methode

```
public static String rot13(String message)
```

welche die Zeichenkette `message` durchläuft und auf jedes Zeichen folgende Regeln anwendet und das Ergebnis zurückgibt:

- Ersetzt werden nur Zeichen von `A-Z` und `a-z`, alle anderen Zeichen bleiben gleich
- Alle Zeichen von `A-M` und `a-m` werden durch das Zeichen 13 Buchstaben *später* im Alphabet ersetzt
- Alle Zeichen von `N-Z` und `n-z` wird das Zeichen 13 Buchstaben *früher* im Alphabet ersetzt

Sorgen Sie dafür, dass vor dem Schreiben der Zeile in die Datei die „Verschlüsselung“ auf die zu schreibende Zeichenkette angewandt wird!

Der große Vorteil von `rot13` ist, dass durch doppeltes Anwenden alles wieder lesbar wird. Sorgen Sie dafür, dass beim Lesen aus der Log-Datei auf jede eingelesene Zeile wiederum der rot13-Algorithmus angewandt wird um die Zeilen wieder lesbar zu machen.

Hinweise:

Starten:

Starten Sie die Anwendung mit einer Klasse `SnatChat` (s. USB-Stick), etwa

```
public class SnatChat {
   public static void main(String[] args) {
      SnatChatRoom room = new SnatChatRoom("GansGeheim");
      room.register( new SnatChatWindow(room, new Account("Bob") ) );
      room.register( new SnatChatWindow(room, new Account("Alice") ) );
   }
}
```

Erzeugen einer zufälligen, nicht zu hellen Farbe:

Nutzen Sie hierfür `new java.awt.Color(int r, int g, int b)`. Für `r`, `g` und `b` sind jeweils **zufällige Werte von 0 bis 200 (beide inklusive)** zu erzeugen.

Verwendung der bereitgestellten Klasse `ChatMessagesComponent`:

Anstatt eines `JPanels` für das Hinzufügen der Nachrichten können Sie die Klasse `ChatMessagesComponent` verwenden, welche sich um das Layouten (inkl. Scroll-Balken) der Nachrichten kümmert. Die Verwendung kann beispielhaft so aussehen:

```
ChatMessagesComponent chatMessages = new ChatMessagesComponent();
JLabel myLabel = new JLabel("A Message for Bob");  // Label mit neuer Nachricht

chatMessages.add(myLabel); // Hinzufügen einer Nachricht
chatMessages.remove(myLabel); // Entfernen einer Nachricht

frame.add(chatMessages);
```