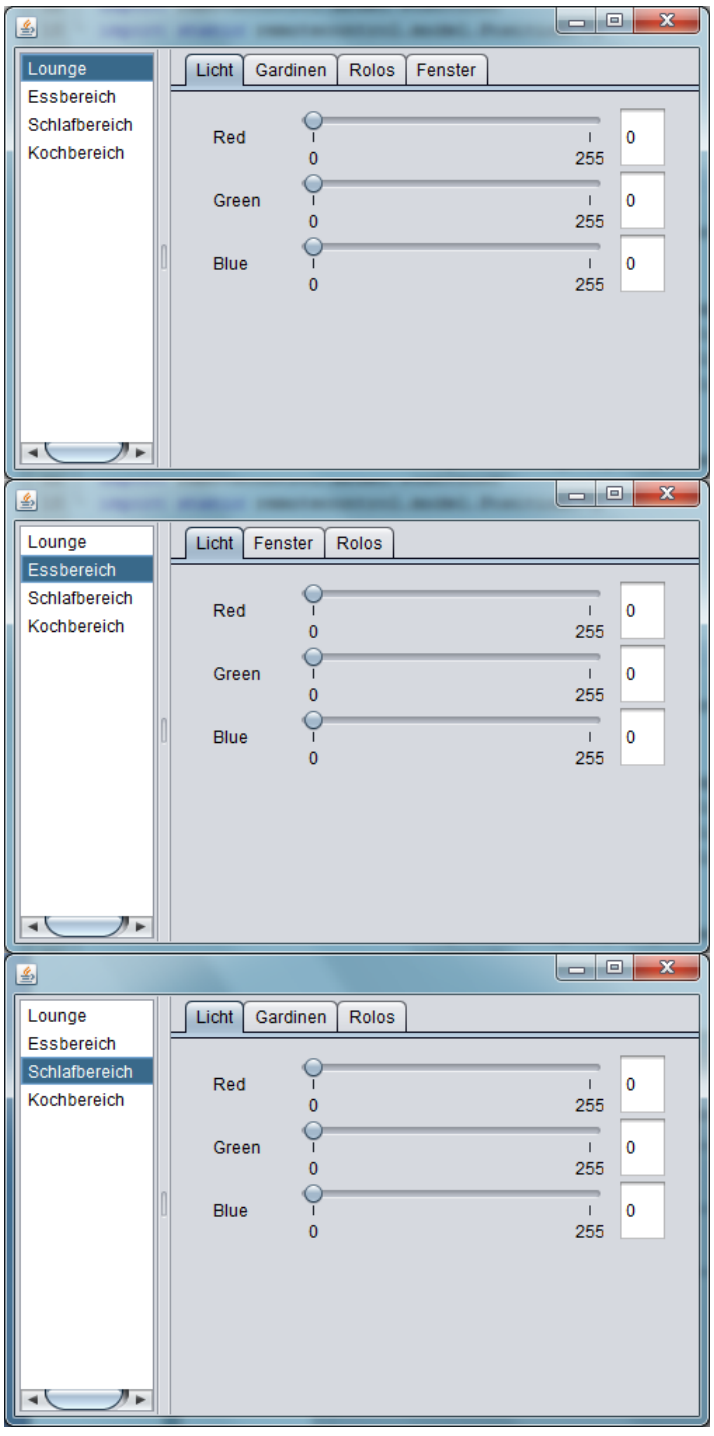
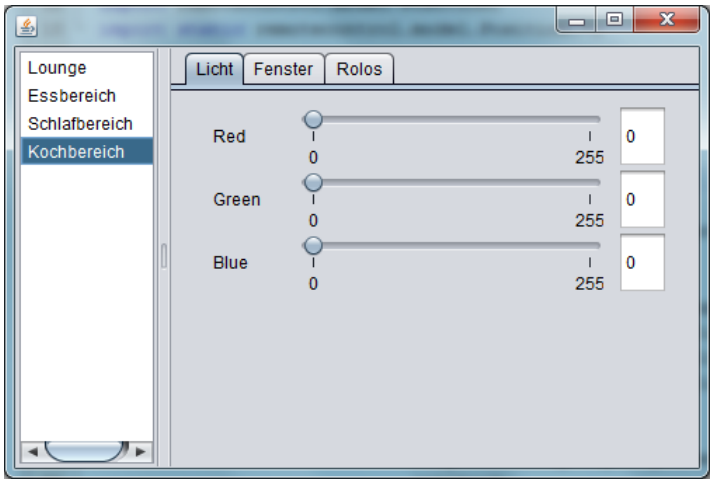
Aufgabe 4 Entwicklung eines GUI's für die Steuerung des Living Place

Sie sollen eine GUI entwickeln, dass Raum-abhängig die Steuerungselemente anbietet und mit geeigneten Listenern die Eingabe des Benutzers in Steuerungsbefehle für das Living Place umsetzt.

Die Aufbereitung der Steuernachrichten sowie das Aktivieren der Systeme im Living Place sind vorgegeben.

Das GUI hat folgenden Aufbau:





- In der Liste werden alle verfügbaren Räume angeboten.
- Selektion eines Raumes zeigt nur die Kontrollelemente an, die für diesen Raum zur Verfügung stehen.
- Die Kontrollelemente sind in einem JTabbedPane angeordnet. Jedes Kontrollelement belegt ein eigenes Tab.

Teil A:

1. Modellieren Sie Räume und Kontrollelemente als Enums.
    a. Die Namen der Enums sind vorgegeben und müssen exakt so geschrieben werden wie hier angegeben:
        i. Enum ***Room: LOUNGE, DINING, SLEEPING, KITCHEN***
        ii. Enum ***Control: LIGHT,BLIND,WINDOW,CURTAIN***

    Erzeugen Sie die Enums mit einem Namen, der als Ergebnis der ***toString()*** Methode zurückgeliefert wird, damit im GUI die Bezeichner oben angezeigt werden.
    b. Räume kennen ihre Controls. Room hat die abstrakte Methode
    public abstract Control[] getControls();
    die in jedem Enum Objekt implemeniert werden muss und ein Array der für diesen Raum definierten Controls zurückgibt.

2. Verwenden Sie für die linke Seite eine ***JList*** mit einem adaptierten ***ListModel***

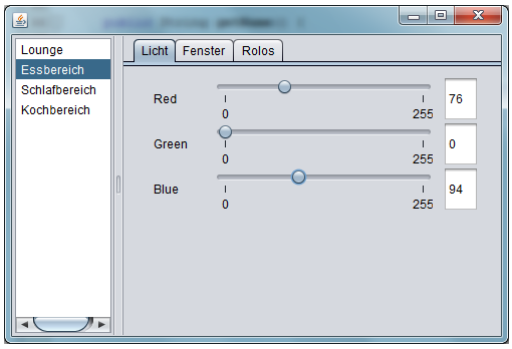
3. Registrieren Sie einen geeigneten Listener für die ***JList***, der in Abhängigkeit vom selektierten Raum den Inhalt des ***JTabbedPane*** aufbaut.
    a. ***JTabbedPane*** bietet dafür die Methoden
        i. ***removeAll:*** entfernt alle Tabs
        ii. ***addTab(<name_tab>, <a_JComponent>)***: fügt einen Tab mit Namen und ***name_tab*** hinzu, dessen Inhalt ***a_JComponent*** ist

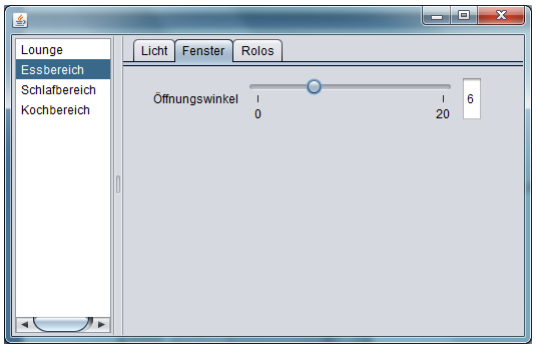
    Beginnen Sie einfach und verwenden zunächst eine TextView, die die Kombination aus Raum und Control anzeigt, um den Mechanismus zu überprüfen.

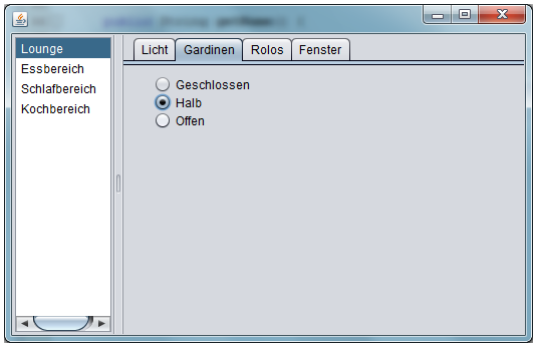
Teil B: Implementieren der Inhalte des Tabs

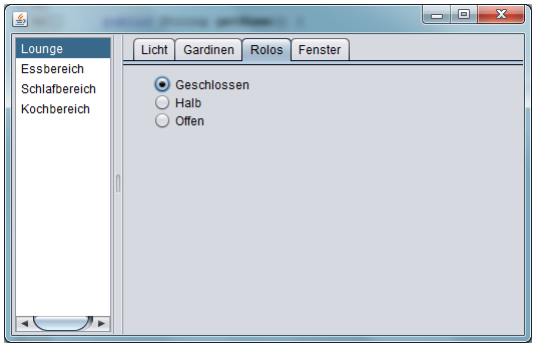
Wenn Sie sich die Elemente auf den einzelnen TabViews anschauen, dann stellen Sie fest, dass wir zwei verschiedene Aufbauten haben:

1. Label Slider TextView (Licht und Fenster)
2. RadioButton Gruppe (Rolos und Gardinen)









Das Ziel soll es sein die Einzelkomponenten für die unterschiedlichen Controls soweit wie möglich wiederzuverwenden und ggf. geeignet zu konfigurieren. Wir entwerfen eine Lösung, die ***ComponentGroup*** Objekte zu einer View komponiert.

Wir haben zwei Typen von Component Groups:

1. Die erste besteht aus einem ***JLabel***, einem ***JSlider*** und einem ***JFormattedTextfield***
2. Die zweite besteht aus einem ***JPanel***, das die Gruppe der ***JRadioButtons*** enthält.

Lösungsansatz:

1. Schreiben Sie eine Klasse ***ComponentGroupHelper***, die zwei statische Methoden hat:

   a. ***public static ComponentGroup getSliderViewGroup(***
   final ControlViewGroupCallback callback, final String label, int min, int max)

   Die Methode erzeugt eine Implementierung für das Interface ***ComponentGroup***, die aus einem Label, einen Slider und einem Textfeld mit den notwendigen Listenern besteht. Es bietet sich die Implementierung mit Hilfe einer anonymen inneren Klasse an.

b. ***public static ComponentGroup getRadioViewGroup(***
final ComponentGroupCallback callback,
final Position[] labels)

Die Methode erzeugt eine Implementierung für das Interface ***ComponentGroup***, die aus einem JPanel besteht, das die Button Gruppe und die RadioButtons enthält. ***Position*** ist ein Enum, das die möglichen Positionen für Gardinen und Rolos modelliert. Das Enum ist vorgegeben. Auch hier bietet sich eine Implementierung mit Hilfe einer anonymen inneren Klasse an.

c. Das Interface ***ComponentGroup*** verlangt die Implementierung der Methoden
   i. ***public Component[] getComponents()***: Hier wird das Array der GUI Komponente zurückgeliefert.
      1. Für die SliderViewGroup ist das das Array mit den 3 Komponenten Label, Slider, FormattedTextView.
      2. Für die RadioViewGroup ist das das Array mit einem Element, dem JPanel, das die Buttongruppe enthält.

   ii. ***public Object getValue()***: Hier wird der Wert der ComponentGroup zurückgeliefert.
      1. Für die SliderViewGroup ist das der letzte über den Slider eingestellte Wert.
      2. Für die RadioViewGroup das Enum Position, das dem selektierten Radiobuttion entspricht.

d. Um die Steuerung für farbiges Licht zu realisieren benötigen wir drei ***ComponentGroup*** Elemente bestehend aus Label Slider TextView, für die Steuerung der Fenster hingegen nur eine ***ComponentGroup***.

Da wir in der ComponentGroup also nichts über die Zusammenstellung der einzelnen Gruppen wissen, muss die Interpretation der Eingaben über kombinierte Gruppen an „höherer“ Stelle in einer einbettenden Komponente erfolgen.

Dazu übergeben wir in den beiden Erzeugermethoden eine Referenz auf ein Objekt, das das Interface ***ComponentGroupCallback*** implementiert. Über dieses Callback informiert die ***ComponentGroup*** die einbettende Komponente über Änderungen, indem die Methode ***valueChanged***() auf dem Callback Objekt aufgerufen wird.

Beim Aufruf der Erzeugermethoden von ***ComponentGroupHelper*** werden wir Referenzen auf Komponenten übergeben, die die ***ComponentGroup*** Elemente einbetten und das Interface ***ComponentGroupCallback*** implementieren**.** Über die Methode mit ***getValue***() können diese den Wert der ***ComponentGroup*** abfragen und in eine Steuernachricht übersetzen.

e. Bleibt noch die Komponenten zu schreiben, die die ComponentGroup Elemente zu den Control Views für die einzelnen Living Place Systeme zusammenstellen und die Nachrichten für die Steuerung generieren.

Teil B-2: Entwickeln der ControlView Komponenten

Sie finden in dem vorbereiteten Projekt die abstrakte Klasse ***ConfigurableControlView.*** Diese Klasse „implementiert" das Interface ***ComponentGroupCallback*** und gibt Implementierungen für

1. Das Layout der CompentGroup Elemente vor (Methode ***addComponents()***);

2. Das Erzeugen von ControlView Komponenten für alle Typen von ControlViews mit Hilfe der Factory-Methode
   public static ConfigurableControlView createInstance(Control control, Room room)

3. Sie müssen „nur" noch

   a. die konkreten Klassen ***ConfigurableLightView***, ***ConfigurableCurtainView*** , ***ConfigurableWindowView*** und ***ConfigurableBlindView*** implementieren und dabei die Interface-Methode ***valueChanged()*** realisieren.

   b. In der Methode valueChanged
      (i) werden die Werte aller ComponentGroup Elemente der View ausgelesen.
      (ii) Über den Raum (Instanzvariable ***room***) und das Control ein ***MessageHelper*** erzeugt (Methode ***getMessageHelper(room,control)***)
      (iii) Über das MessageHelper Objekt eine Liste von gültigen Nachrichten erzeugt (Methode ***getMessages(Room room, String... args)***) , die
      (iv) abschließend dem ***MagicPublisher*** zur Versendung übergeben wird.
      (v) Die ***args*** in ***getMessages*** sind die Werte der ComponentGroups

   c. eine Utility Klasse ***ControlViewHelper*** schreiben, deren Factory-Methode

      static ComponentGroup[] getComponentGroups(
      ComponentGroupCallback callback, Control control)

      für das jeweilige Control ein Array von passenden ComponentGroup Objekten zurückliefert

Anhang:

Vorgegebene Klassen / Interfaces / Enums (NICHT VERÄNDERN)

1. ***MessageHelper, Message, MessageFactory, MagicPublishe***r im Package ***remotecontrol.message***
2. ***ConfigurableControlView*** , ***ComponentGroup***, ***ComponentGroupCallback*** im Package ***remotecontrol.ui***
3. ***Position*** im Package ***remotecontrol.model***
4. ***Constants***, ***StringUtils*** im Package ***remotecontrol***.

Beim aktuellen Stand des MagicPublisher werden die Nachrichten nur ausgegeben. Für die Praktika-überprüfung wird dann ein voll funktionsfähiger Publisher integriert.