**Bitte lösen Sie Aufgaben 11-13 bis zum 10.05.2020 und geben Sie Aufgabe 13 ab!**

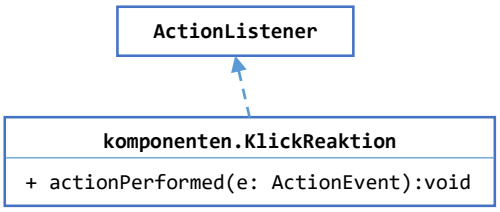
# Aufgabe 11

**Auf Klick reagieren, Interface**

In dieser Aufgabe Schreiben Sie zum ersten Mal eine Anwendung, die auf den Klick auf ein JButton Objekt reagiert. Legen Sie zunächst das **neue Paket *komponenten*** an, darin schreiben Sie die öffentliche Klasse *KlickReaktion*:


```
public class KlickReaktion implements ActionListener{
```

Wie das UML Diagramm zeigt, wird die Implementierung des Interface durch eine gestrichelte Linie dargestellt. Die vom Interface geforderte Methode *actionPerformed* wird vom Betriebssystem immer dann gerufen, wenn das Ereignis passiert, für das der *ActionListener* registriert ist:

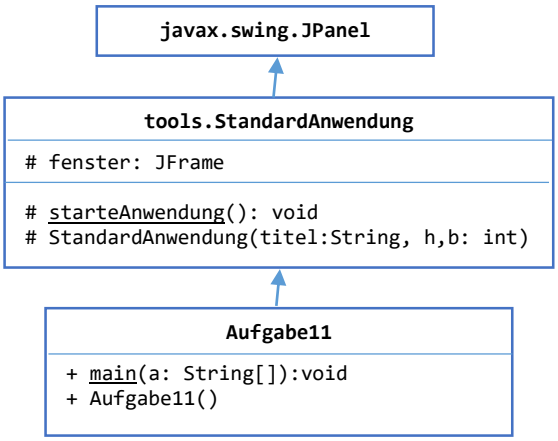



```
public void actionPerformed(ActionEvent e){
     JOptionPane.showMessageDialog(null, "Du sollst doch nicht Klicken!");
}
```

Dann schreiben Sie die Klasse ***Aufgabe11***. Der **Konstruktor** erzeugt ein Objekt vom Typ *JButton*, speichert es in der lokalen Variable *knopf* und legt es auf die Oberfläche. Dann registriert er ein neues Objekt vom Typ *KlickReaktion* als *ActionListener* auf den *JButton*:

```
knopf.addActionListener(new KlickReaktion());
```

Die Methode *addActionListener* meldet das neue anonyme Objekt vom Typ *KlickReaktion* als Ereignis-Handler für den Knopf an. Wenn das Ereignis - der Klick auf den Knopf – eintritt, wird dessen Methode *actionPerformed* aufgerufen.




Die Methode *JOptionPane.showMessageDialog* im Package *javax.swing* gibt es auch in mehreren überladenen Varianten, um Fenster für verschiedene Meldungen zu erzeugen, die Wichtigste ist:

```
showMessageDialog( p:Component, meldung,titel: String, typ: int): void
```

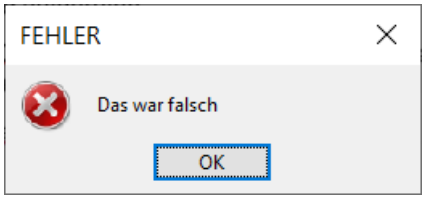
mit:

- *p:* eine Komponente, die steuert, wo das Dialogfenster geöffnet wird.:
  - Für *p=null* geht der Dialog in der Mitte des Hauptbildschirms auf.
  - Für *p≠null* erscheint der Dialog in der Nähe der angegebenen Komponente p. In den Übungen geben Sie meist *this* ein, wenn Sie den Dialog aus einer Komponente, einem *JPanel* u.s.w. heraus absetzen, dann geht das Fenster über oder neben der App auf. In der Klasse *KlickReaktion* geht das nicht, dort müssen Sie *null* angeben.
- *meldung*: Text der Meldung.
- *titel*: Titel, der im oberen Bereich des Meldungsfensters dargestellt wird
- typ: muss einer der folgenden Werte sein:
  - JOptionPane.INFORMATION_MESSAGE
  - JOptionPane.WARNING_MESSAGE
  - JOptionPane.ERROR_MESSAGE

*Beispiel:*

```
JOptionPane.showMessageDialog( null, "Das war falsch", "FEHLER", JOptionPane.ERROR_MESSAGE);
```

liefert das rechts gezeigten Meldungsfenster.




Ein mit *JoptionPane* erzeugter Dialoge sperret die Anwendung bis er geschlossen wird, man nennt das einen **modalen Dialog.**

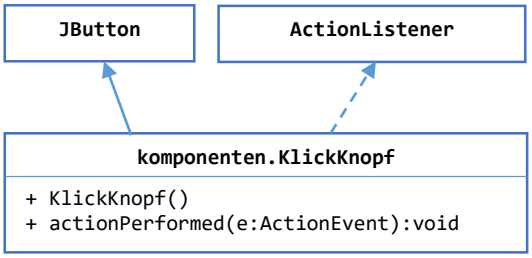
Experimentieren Sie in *actionPerformed* mit den anderen Meldungstypen.

# Aufgabe 12

**Andere Klassenaufteilung**

Wir schreiben die Anwendung von Aufgabe 11 nochmal, teilen die Klassen aber anders auf.

Wir beginnen mit der öffentliche Klasse ***komponenten.KlickKnopf***, die von *JButton* abgeleitet ist und *ActionListener* implementiert wie im UML Diagramm gezeigt. Sie ist keine App, denn sie hat keine *main* Methode. Programmieren Sie zunächst das Gerüst der Klasse anhand des UML Diagramms.




Die Methode ***actionPerformed*** macht das gleiche wie *actionPerformed* von *KlickReaktion* aus Aufgabe 11, mit einem kleinen Unterschied: der erste Parameter von *showMessageDialog* wird statt *null* zu *this*:

```
JOptionPane.showMessageDialog(this, "Du sollst doch nicht Klicken!");
```

damit das Fenster mit der Meldung innerhalb oder in der Nähe des Anwendungsfensters aufgeht.

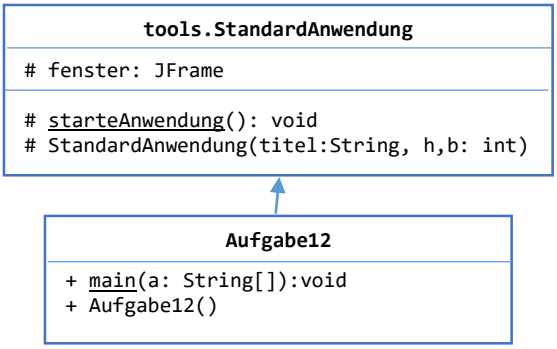
Der **Konstruktor** von *KlickKnopf* ruft zunächst super("Nicht klicken");

Somit hat jedes Objekt vom Typ *KlickKnopf* die gleiche Aufschrift. Dann registriert sich das Objekt als sein eigener *ActionListener*:

```
this.addActionListener(this);
```

Der Knopf reagiert also selbst auf den Klick. Die ganze Funktionalität des Knopfes ist jetzt in einer einzigen Klasse verpackt. Welche der beiden Aufteilungen (Aufgabe 11 oder 12) besser ist, hängt von der Anwendung ab.

Die Klasse ***Aufgabe12*** soll wie gewohnt von *StandardAnwendung* abgeleitet sein. Ihr **Konstruktor** legt ein Objekt vom Typ *KlickKnopf* auf sich selbst – das ist alles.




Unser Programm ist **Event-gesteuert**. Die Methode *actionPerformed* wird nicht von uns selbst gerufen, sondern erst wenn das Betriebssystem einen Klick auf den Knopf registriert.

Lesen Sie die Beschreibung des Interface *ActionListener* und der Methode *addActionListener* durch, um den Zusammenhang zwischen Event und Listener zu verstehen. Näheres finden Sie auch hier:

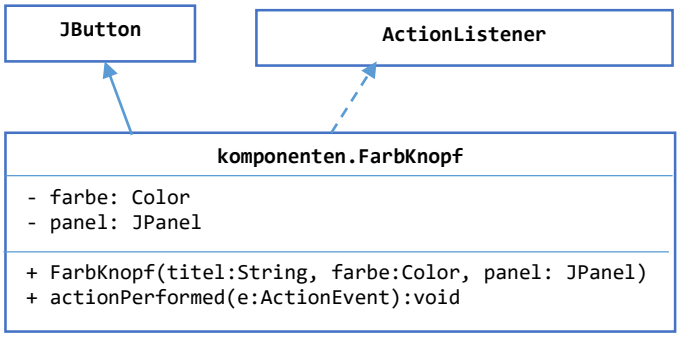
https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

# Aufgabe 13

**Instanzvariablen, ActionListener**

Schreiben Sie eine Java Anwendung, welche 6 Knöpfe zeigt, die den Hintergrund der Anwendung unterschiedlich einfärben.

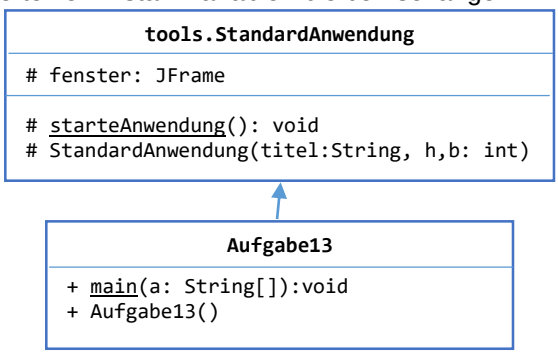
Erzeugen Sie Packet *komponenten* die von *JButton* abgeleitete Klasse ***FarbKnopf***, welche *ActionListener* implementiert. Bei Klick auf diesen Knopf soll ein Panel in einer bestimmten Farbe eingefärbt werden.




Der **Konstruktor** gibt *titel* an den Konstruktor der Basisklasse weiter, speichert die anderen beiden Parameter in den Instanzvariablen und registriert sich als sein eigener *ActionListener*. Zum Schluss können Sie den Hintergrund des Buttons mit *farbe* einfärben.

In ***actionPerformed*** wird das Panel eingefärbt.

Machen Sie sich klar, wozu die Instanzvariablen dienen: sie speichern ihre Werte über einen längeren Zeitraum, in diesem Fall zwischen dem Aufruf des Konstruktors beim Start der App und dem Klick auf den Knopf, der den Aufruf von *actionPerformed* auslöst. Die Werte von Instanzvariablen bleiben so lange erhalten, wie das Objekt existiert – in der Praxis meist so lange die Anwendung läuft.



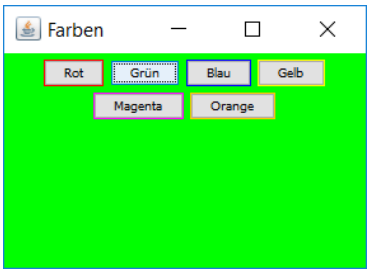

Die Anwendung ***Aufgabe13*** wird wie gewohnt von *StandardAnwendung* abgeleitet.

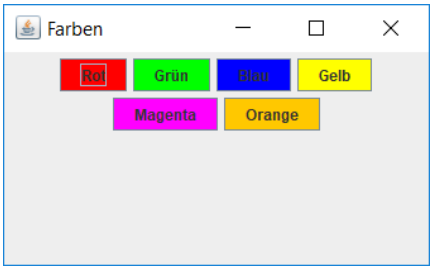
Der **Konstruktor** von ***Aufgabe13*** erzeugt 6 *FarbKnopf* Objekte und legt sie auf seine Oberfläche. Als Panel übergeben Sie dem Konstruktor von *FarbKnopf* jeweils *this*.

Um den Look- and-Feel zu ändern, fügen Sie im Konstruktor von Aufgabe13 hinter super( …) die folgenden Zeilen ein:

```
try{
   UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
}catch(Exception e){   }
```

Das Aussehen Ihrer App ist jetzt unabhängig vom Betriebssystem. Links sehen Sie den System Look-and-Feel von Windows, rechts den *Cross-Platform* Look-and-Feel, bei dem die Farben der *JButton* Objekte deutlicher sind. Am Ende gilt: über Geschmack lässt sich trefflich streiten…




# Das Wichtigste aus der aktuellen Vorlesungswoche:

- Wenn eine Klasse ein **Interface** implementiert, muss sie alle Methoden des Interfaces enthalten.
- Überall dort, wo etwas vom Typ eines Interface erwartet wird, kann man ein Objekt einer Klasse verwenden, welche dieses Interface implementiert.
- Die **elementaren** Datentypen ***int***, ***double*** und ***boolean*** sind fest in die Sprache eingebaut. Im Gegensatz dazu ist jede Klasse ein sogenannter ***Referenz***-Datentyp.
- Eine Variable vom Typ ***int*** speichert eine ganze Zahl, ***double*** eine gebrochen rationale Zahl und ***boolean*** einen logischen Wert.
- **Literale** nennt man Zeichketten, die einen festen Wert repräsentieren. Literale vom Typ ***int*** sind z.B. *1*, *-4711* oder *0*. Literale vom Typ ***double*** haben einen Dezimalpunkt, z.B. *1.* , *47.11* oder *-4.2*.
- Für den Typ ***boolean*** gibt es nur zwei Literale: *true* und *false*.
- Variablen, die innerhalb einer Methode definiert sind, nennt man **lokale Variablen**. Sie sind nur innerhalb der Methode sichtbar, in der sie definiert sind. Ihr Wert bleibt nur so lange erhalten, wie die Methode ausgeführt wird. Methodenparameter sind ebenfalls lokale Variablen.
- Variablen die außerhalb von Methode definiert sind und die **nicht static** deklariert sind nennt man **Instanzvariablen**. Von einer Instanzvariablen existieren so viele Kopien, wie es Objekte gibt, d.h. wenn noch kein Objekt einer Klasse erzeugt wurde, gibt es auch noch keine Kopie der Instanzvariablen.

**Mögliche Prüfungsaufgaben und Fragen:**

- Schreiben Sie eine Java Klasse anhand eines gegebenen UML Diagramms.
- Zählen Sie in den Klassen, die wir bisher geschrieben haben, alle Instanzvariablen, lokalen Variablen, Instanzmethoden sowie Klassenmethoden auf.
- Was versteht man unter überladene Methoden?
- Gegeben sei die nebenstehende Klasse Apfel:
  - Wie viele Kopien der Variable *farbe* existieren?
  - Von welchem Typ sind *this* und *super*?
  - Was bewirkt die Anweisung *super()*?
  - Gibt es überladene Methoden? Zählen sie diese ggf. auf.
  - Was bewirkt die Anweisung *this(Color.RED)*?
  - Wie unterscheidet sich ein Konstruktor von anderen Methoden?

```
public class Apfel extends Obst{
   public static void main(String[] a){
       Apfel t1 = new Apfel(Color.GREEN);
       Apfel t2 = new Apfel();
   }
   private Color farbe;
   public Apfel(Color f){
        super();
        this.farbe = f;
   }
   public Apfel(){
        this(Color.RED);
   }
 }
```