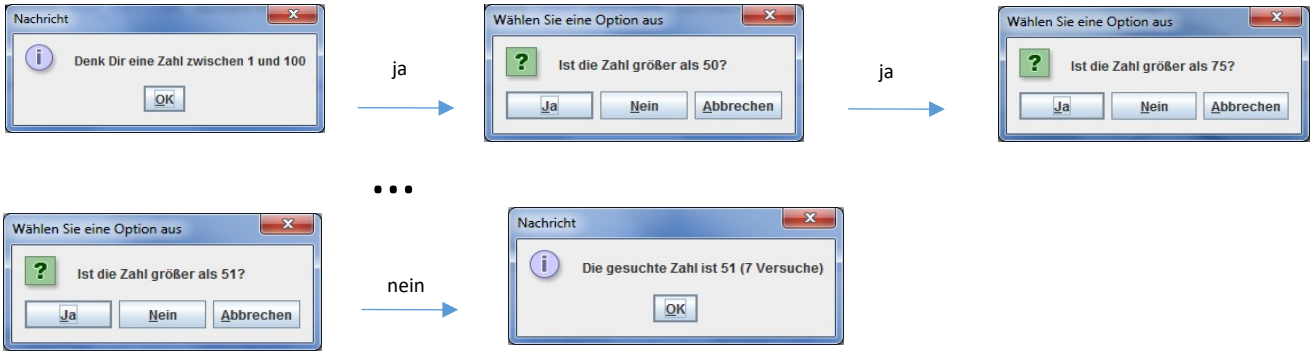
Bitte lösen Sie Aufgabe 4-6 bis zum 21.10.2020 **und geben Aufgabe 6 bis zum 25.10.2020 ab!**

Aufgabe 4

Binäre Suche (Schleife)

Schreiben Sie ein Programm, das eine Zahl zwischen 1 und 100 errät, die sich der Anwender ausgedacht hat. Das Programm stellt dabei nur Fragen, die mit ja oder nein zu beantworten sind.

Der Dialog soll etwa so aussehen:



Diese Anwendung ist nicht von JPanel abgeleitet und verwendet **nicht** *StandardFenster*, sie besteht nur aus den statischen Methoden ***main*** und ***binaereZahlenSuche***, welche *JOptionPane*-Methoden rufen.

Programmieren Sie in ***main*** zunächst folgende *do-while* Schleife, in der Sie den Anwender fragen, ob weitergemacht werden soll:

Aufgabe04
+ main(a: String[]): void + binaereSuche():void

```
do{
     binaereSuche();
} while(JOptionPane.showConfirmDialog(null,"Weiterspielen?")==JOptionPane.YES_OPTION);
```

In ***binaereSuche*** rufen Sie *JOptionPane.showMessageDialog* mit der Aufforderung, sich eine Zahl zu denken und definieren die int-Variablen *min=1*, *max=100* sowie *mitte.*

In einer *while*-Schleife, die solange läuft wie *min<max* ist, führen Sie die folgenden Schritte durch:

- Berechnen Sie *mitte* als *(min+max)/2*,
- fragen Sie mit *showConfirmDialog*, ob die Zahl größer ist als *mitte*, den Return-Wert speichern Sie in der int-Variable *antwort*.
- falls *antwort* gleich *JOptionPane.YES_OPTION* ist, setzen Sie *min=mitte+1,* bei *NO_OPTION* wird *max=mitte* gesetzt,
- falls keine der beiden Optionen gewählt wurde, beenden Sie die Methode mit *return*.

Nach der Schleife melden Sie die gefundene Zahl mit *JoptionPane.showMessageDialog()*.

Aufgabe 5

Binäre Suche (Rekursion)

Wiederholen Sie Aufgabe 4, aber verwenden Sie diesmal eine Rekursion statt der Schleife.

In *main* zeigen Sie wie in Aufgabe 4 in einer *do…while* Schleife den Dialog „Denk Dir eine Zahl zwischen 1 und 100“ und ruft dann die Methode *binaereSuche(1,100)*.

Aufgabe05
+ main(a: String[]): void + binaereSuche(min,max:int):void

Diese Methode prüft zunächst, ob *max<=min* ist – in diesem Fall meldet sie, dass die Zahl gefunden ist und die Rekursion wird mit *return* abgebrochen.

Für *max>min* wird der Mittelwert von *max* und *min* berechnet und der Dialog „Ist die Zahl größer als …“ gezeigt. Falls die Antwort gleich YES_OPTION ist, setzen sie die Suche rekursiv mit *binaereSuche(mitte+1,max)* fort, falls NO_OPTION gewählt wurde, wird die Suche rekursiv im Intervall *(min, mitte)* weitergeführt. Wurde keine der beiden Optionen gewählt, brechen Sie das Spiel mit *return* ab.

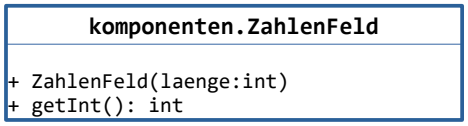
Vergleichen Sie beiden Lösungen (Rekursion und Schleife) miteinander.

Aufgabe 6

Rekursion, Swing, JTabbedPane

In dieser Aufgabe schreiben wir mehrere rekursive Algorithmen, die in einer gemeinsamen Swing-Anwendung zusammengefasst werden.

Erzeugen Sie das neue Paket ***komponenten*** und darin die von *JTextField* abgeleitete Klasse ***ZahlenFeld***.



Der **Konstruktor** gibt seinen Parameter an den Konstruktor der Basisklasse weiter rund ruft `this.setHorizontalAlignment(RIGHT)` damit die Eingabe rechtsbündig im Textfeld dargestellt wird.

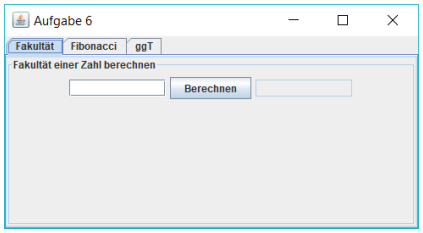
Die Methode ***getInt*** wandelt den Text des Zahlenfeldes mit *Integer.parseInt* in eine ganze Zahl und gibt sie zurück.

Die Klasse ***Aufgabe06*** wird von *StandardAnwendung* abgeleitet, in *main* rufen Sie *starteAnwendung*.

Im **Konstruktor** rufen Sie zunächst super(…). Dann legen sie das Layout als *BorderLayout* fest und erzeugen ein *JTabbedPane* Objekt, auf das Sie mehrere JPanel Objekte legen:

tabbedPane.*addTab(*`"Fakultät"`*, new JPanel());*
tabbedPane.*addTab(*`"Fibonacci"`*, new JPanel());*
…

Zum Schluss legen Sie das *JTabbedPane* Objekt auf *this*.



Jetzt haben Sie schon das Layout der Anwendung. Nach und nach ersetzen Sie jetzt die JPanel Objekte durch Objekte der folgenden von *JPanel* abgeleiteten Klassen:

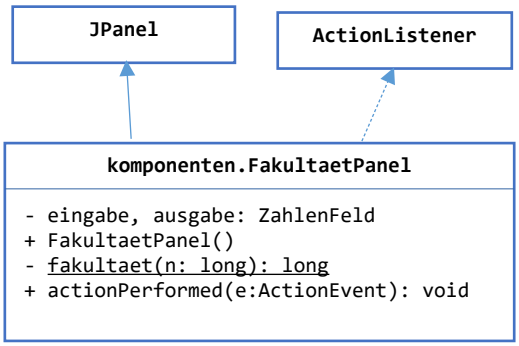
Klasse FakultaetPanel

Diese Klasse soll *ActionListener* implementieren und oben stehende Anordnung darstellen, welche die Fakultät einer Zahl berechnet. Geben Sie dem Panel eine *TitledBorder* mit dem Titel „Fakultät einer Zahl berechnen“. Lesen Sie dazu http://download.oracle.com/javase/tutorial/uiswing/components/border.html .

Sie können die Klasse FakultaetPanel am Ende von Aufgabe06.java anfügen (dann darf sie nicht public sein) oder in eine eigene Datei schreiben.

Eingabe- und Ausgabe werden als Instanzvariablen vom Typ ***ZahlenFeld*** der Größe 9 initialisiert, der Knopf wird in einer lokalen Variablen im Konstruktor gespeichert. Die Ausgabe wird durch Aufruf von *setEditable* als nicht-editierbar gesetzt.

Wenn Sie *die Klasse* als *ActionListener* sowohl für den Knopf als auch für das Eingabefeld registrieren, startet die Berechnung sowohl beim Klick auf den Button als auch beim Betätigen der Eingabetaste.



Warum ist der Return-Typ von *fakultaet long* und nicht *int*? Schlagen Sie nach, wie groß Werte dieser beiden Typen werden können. Sie werden feststellen, dass schon 21! nicht mehr als *long* dargestellt werden kann, stattdessen wird ein negativer Wert angezeigt.

Klasse *FibonacciPanel*

In der Fibonacci Reihe ist jede Zahl die Summe der vorhergehenden zwei Zahlen:

$$f_i = \begin{cases} a \text{ für } i=1 \\ b \text{ für } i=2 \\ f_{i-1}+f_{i-2} \text{ für } i>2 \end{cases}$$

Eine Fibonacci Reihe ist also durch die Werte ihrer ersten zwei Glieder a und b bestimmt.

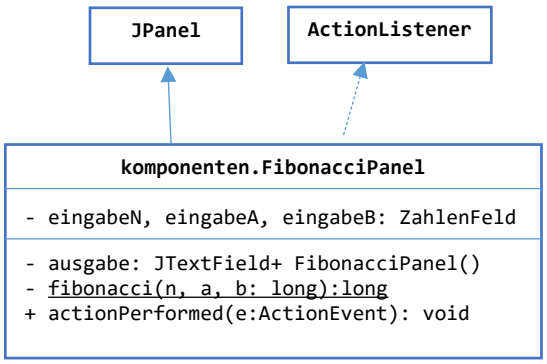
Einige Glieder der Fibonacci-Reihe mit a=b=1 sind:

f_3=2 f_4=3 f_5=5 f_{20}=6765

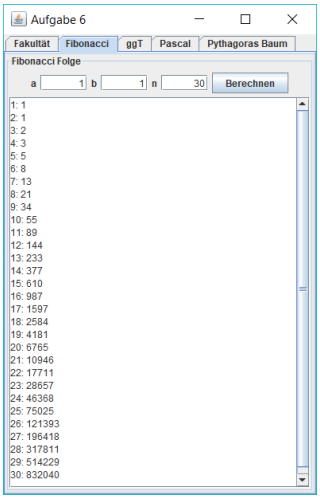
Berechnen Sie auf einem Blatt Papier zunächst f_n für n=3 und 5 für die Fibonacci-Reihe mit a=1 und b=3.

Beginnen Sie mit der **rekursiven** Methode ***fibonacci***, welche die Fibonacci Zahl $\mathbf{f_n}$ der Reihe mit den Startwerten a und b berechnet:

Für n=1 wird a zurückgegeben, für n=2 b, ansonsten gibt sie die Summe der letzten beiden Glieder zurück, wobei sich die Methode zweimal rekursiv aufruft.



Zum Testen der Methode erzeugen Sie nebenstehende Anordnung von Swing-Elementen, welche die Fibonacci Zahlen in einer *JTextArea* ausgibt. Die drei Zahlenfelder und die *TextArea* müssen Instanzvariablen sein, damit sie von der Methode *actionPerformed* gesehen werden. Das *JButton*- und die *JLabel* Objekte können lokale Variablen im Konstruktor sein.



Der **Konstruktor** legt das Layout als *BorderLayout* fest und definiert eine *TitledBorder* als Rand. Für die Eingabefelder und den Button wird ein eigenes Panel als lokale Variable definiert, welches mit

```
this.add(oberesPanel, BorderLayout.NORTH);
```

auf dem oberen Teil der Oberfläche platziert wird.

In die Mitte der Oberfläche legen Sie die Ausgabe mit Scrollbalken:

```
this.add(new JScrollPane(ausgabe), BorderLayout.CENTER));
```

Experimentieren Sie mit verschiedenen Anfangswerten von Fibonacci-Reihen.

Klasse GgtPanel

Der nebenstehende Algorithmus zur Berechnung des größten gemeinsamen Teilers (ggT) zweier Zahlen, die nicht beide 0 sind, ist ein schönes Beispiel für Rekursion.

Euklid hat diesen Algorithmus im 3. Jh. vor Christus in seinem Mathematischen Werk „*Die Elemente*“ beschrieben.
Er gilt als der älteste überlieferte Algorithmus, weil er im Gegensatz zu früheren Beschreibungen nicht anhand von Beispielen, sondern abstrakt definiert ist.

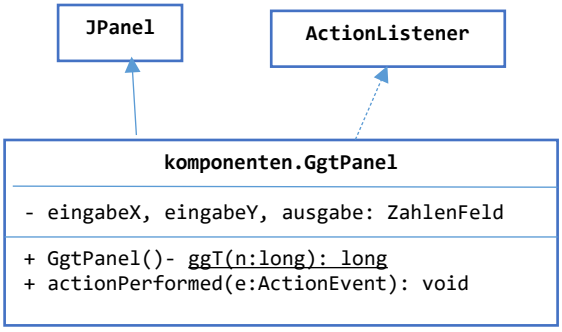
$$ggT(x,x) = x$$
$$ggt(x,y) = ggT(x-y,y) \quad \text{für } x>y$$
$$ggT(x,y) = ggT(x,y-x) \quad \text{für } y>x$$

Berechnen Sie zunächst von Hand den größten gemeinsamen Teiler von 54 und 30 nach diesem Schema, dann den von weiteren Zahlenpaaren, bis Sie die Arbeitsweise verstehen!

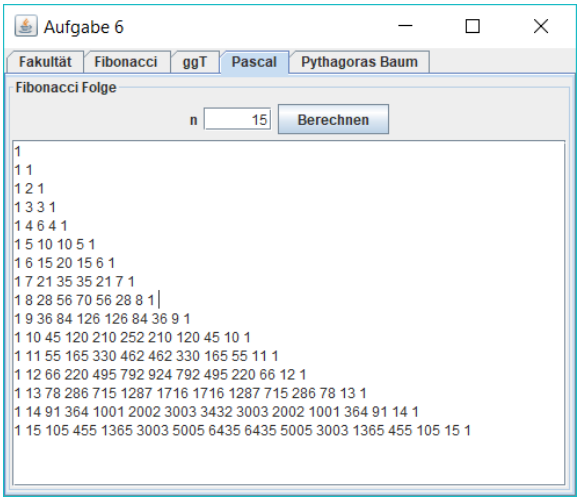
Erst danach programmieren und testen Sie folgende **rekursive** Java Methode zur Berechnung des größten gemeinsamen Teilers von x und y:

```
private static int ggT(int x, int y)
```

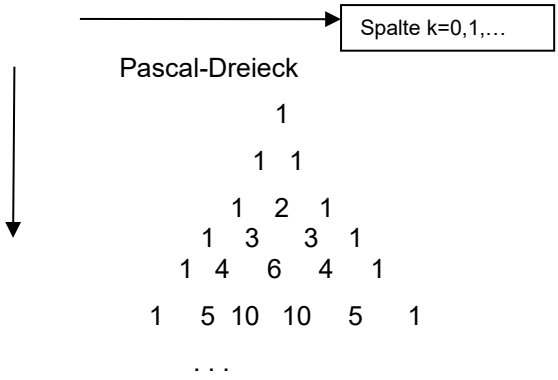
Schreiben Sie zum Testen die Klasse ***GgtPanel*** gemäß der nebenstehenden Abbildung.



Klasse *PascalPanel*



Die Pascal-Zahlen (Binominalkoeffizienten) sind ein weiteres schönes Beispiel für Rekursion:



Die Bildungsregel der Pascal-Zahlen lautet: $$pas(i,k)=\begin{cases} 1 \forall\ k=0 \, oder \, k=i \\ pas(i-1,k)+pas(i-1,k-1) \end{cases}$$

Schreiben Sie eine Java-Klasse, welche die Pascal-Zahlen in der gezeigten Form auf einer *JTextArea* ausgibt.

Programmieren Sie dazu die folgende **rekursive** Funktion:

```
static int pas(int i, int k)
```

Übrigens sind Binominalkoeffizienten und Fibonacci-Zahlen eng miteinander verbunden. Wenn Sie Spaß an mathematischen Fragestellungen haben, können Sie es z.B. hier nachlesen:

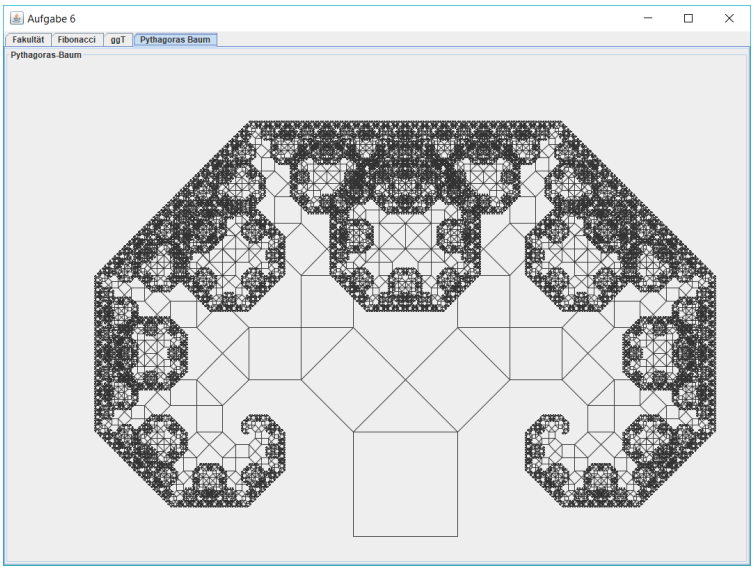
http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibmaths.html#pascal

Klasse PythagorasBaumPanel:

Diese Teilaufgabe muss **nicht** mit abgegeben werden!

Übernehmen Sie die Klasse *PythagorasBaumPanel* aus dem persönlichen Stundenplan im Ordner *komponenten*. Sie stellt einen Pythagoras-Baum dar.

Lesen Sie die Klasse aufmerksam durch und versuchen Sie sie anhand der Bildungsregel, die Sie z.B. in Wikipedia finden, das Programm zu verstehen.



Zusatzaufgabe für Experten: Ergänzen Sie die Anwendung um eine Klasse, die ein **Sierpinski Dreieck** zeichnet. Diese Teilaufgabe muss ebenfalls **nicht** abgegeben werden.

Das Wichtigste aus den ersten beiden Semesterwochen

- Bei der Definition einer Methode legt man ihren **Namen**, die **Sichtbarkeit**, den **Return-Typ**, und die **Parameter-Liste** fest und es wird definiert, ob es sich um eine Klassen- (static) oder Instanzmethode handelt.
- {Name, Return-Typ und Parameter-Liste} nennt man auch **Signatur** der Methode.
- Die Parameter, die an Methoden übergeben werden, sind **immer Kopien** der Original-Werte im rufenden Programm. Eine Methode kann also nicht aus Versehen einen Wert im rufenden Programm überschreiben.
- Wird eine Variable innerhalb einer Methode deklariert, nennt man sie eine **lokale Variable**. Sie gelten nur innerhalb der Methode, d.h. der gleiche Name kann in verschiedenen Methoden verwendet werden. Parameter sind ebenfalls lokale Variablen.
  Achtung: lokale Variablen verdecken Instanz- oder Klassenvariablen gleichen Namens, das ist eine häufige Ursache für eine *NullPointerException*.
- Ruft eine Methode sich selbst auf, spricht man von direkter oder indirekter **Rekursion**. Am Anfang einer rekursiven Methode muss in der Regel eine *if*-Anweisung stehen, welche die Rekursion beendet. Fehlerhafte Rekursionen führen zu einem *StackOverflowError*.
- Ein Algorithmus ist eine Vorschrift zur Lösung eines Problems in **endlich vielen** Schritten.
- Um den Aufwand von Algorithmen vergleichbar zu machen, gibt es die **Landau Notation** (**groß O Notation**), in der man die Ordnung der Komplexität eines Algorithmus angibt. Sie ist ein Maß dafür, wie der Aufwand des Algorithmus in Abhängigkeit der Menge der zu verarbeitenden Daten steigt. Die wichtigsten Komplexitäten sind O(1), O(log N), O(N), O(N log N), $O(N^2)$.
- Es gelten folgende Regeln für die O-Notation:
  - Zwei Komplexitäten, die sich durch einen Faktor unterscheiden, sind gleich
  - Ein Algorithmus, der eine Sequenz mehrerer Teilalgorithmen ist, hat die höchste Komplexität seiner Teilalgorithmen.
- Die **Lineare Suche** und die **binäre Suche** sind zwei wichtige Algorithmen, sie haben die Komplexität O(N) und O(log N). Die binäre Suche setzt einen sortierten Datenbestand voraus, sie halbiert in jedem Schritt die Menge der noch zu verarbeitenden Daten. Solche Algorithmen nennt man auch „*Divide and Conquer*“ Algorithmen.
- Ein Sortierverfahren ist **stabil**, wenn Elemente mit gleichem Schlüssel ihre ursprüngliche Reihenfolge behalten. **In-Situ** bedeuted, dass beim Sortieren die Daten nur einmal im Speicher gehalten wrerden müssen, **ex-Situ** Verfahren (out of place) brauchten den mehrfachen (meist doppelten) Speicher.