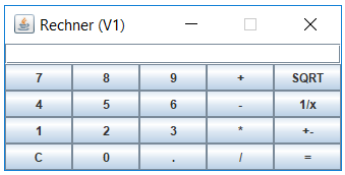
Bitte lösen Sie Aufgaben 14-16 bis zum 17.05.2020 und geben Sie Aufgabe 16 ab!

Aufgabe 14

Layout

Schreiben Sie eine Java Anwendung, welche nebenstehendes Layout für einen Taschenrechner darstellt.



Für diese Aufgabe erzeugen Sie bitte das Package *aufgabe14*, dann können wir jeweils das ganze Paket kopieren, wenn wir später am Taschenrechner weiterarbeiten. Darin erzeugen Sie die von *StandardAnwendung* abgeleitete Klasse *Taschenrechner*.

Für den Taschenrechner habe ich auch den *CrossPlatformLookAndFeel* gewählt – wie gesagt, das ist Geschmackssache - probieren Sie es aus.

Im Konstruktor von *Taschenrechner* definieren Sie zunächst ein `BorderLayout` für das Haupt-Panel:

```
this.setLayout(new BorderLayout());
```

Erzeugen Sie das *JTextField*-Objekt ‚anzeige' und legen es auf den oberen Bereich des Hauptpanels:

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

Dann erzeugen Sie das *JPanel*-Objekt *tastenFeld*, diesem geben Sie ein *GridLayout* mit einem Raster von 4*5 Zeilen/Spalten:

```
tastenFeld.setLayout(new GridLayout(4,5));
```

Danach legen Sie 20 *JButton*-Objekte mit der Aufschrift "7", "8", "9", "+","SQRT", u.s.w. auf das Tastenfeld.

Zum Schluss legen Sie das Tastenfeld in die Mitte des Hauptpanels:

```
this.add(tastenFeld, Borderlayout.CENTER)
```

Durch Aufruf von *fenster.pack()* passen Sie die Größe des Fensters an den Inhalt an, mit *fenster.setResizable(false)* verhindern Sie, dass seine Größe verändert werden kann. Wenn Sie diesen Aufruf auskommentieren, können Sie den Effekt von *BorderLayout* und *GridLayout* beobachten, dass sich alle Objekte maximal ausdehnen.

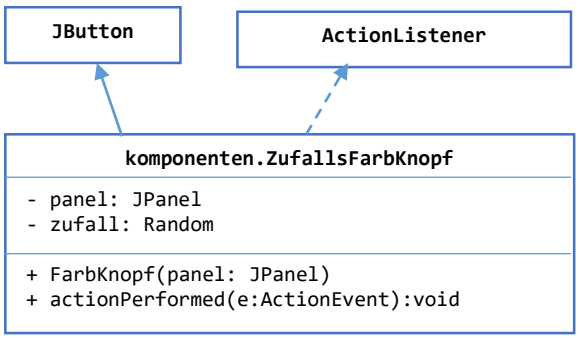
Im Laufe des Semesters werden wir den Taschenrechner komplett fertig programmieren.

Aufgabe 15

Random: zufällige Zahlen

Zufallszahlen sind ein wichtiges Element, um Anwendungen interessanter zu gestalten. In dieser Aufgabe Schreiben Sie eine Java Anwendung, welche einen *JButton* zeigt, der die Hintergrundfarbe bei jedem Klick zufällig verändert.

Legen Sie zunächst im Paket *komponenten* die von *JButton* abgeleitete Klasse ***ZufallsFarbKnopf*** an, welche *ActionListener* implementiert. Bei Klick auf diesen Knopf soll ein Panel in einer zufälligen Farbe eingefärbt werden. Programmieren Sie das Gerüst der Klasse anhand des UML Diagramms.



Der **Konstruktor** setzt den Titel auf „Zufällige Farbe", speichert den Parameter in der Instanzvariablen und registriert sich als sein eigener *ActionListener*. Dann erzeugt er ein *Random*-Objekt und speichert es in der Instanzvariable *zufall*.

In ***actionPerformed*** wird das Panel eingefärbt. Dazu erzeugen Sie zunächst drei zufällige ganze Zahlen zwischen 0 und 255:

```
int rot = zufall.nextInt(256);
    …
```

daraus machen Sie eine zufällige Farbe:

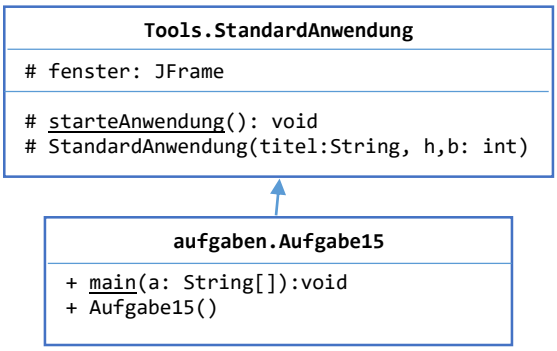
```
Color farbe = new Color(rot, gruen, blau);
```

und färben *panel* damit ein. Lesen Sie die Beschreibung von *Color* und *Random* aufmerksam durch.

Die Anwendung ***Aufgabe15*** wird wie gewohnt von *StandardAnwendung* abgeleitet.

Der **Konstruktor** von ***Aufgabe15*** erzeugt ein *ZufallsFarbKnopf* Objekt und legt es auf seine Oberfläche.

Stellen Sie sicher, dass sie die Aufgabe komplett verstehen! Es geht vor allem um die Struktur der Klassen.



Aufgabe 16

Instanzvariablen, *zeichne* Methode

Schreiben Sie die von *StandardAnwendung* abgeleitete Klasse *Aufgabe16* und färben Sie sie rot, indem Sie im Konstruktor *this.setBackground* aufrufen.

Jetzt fügen Sie die Instanzvariable *xPos* vom Typ *int* in die Klasse *Aufgabe16* ein und geben ihr den Anfangswert 10. Der Typ *int* speichert ganze Zahlen (englisch integer).

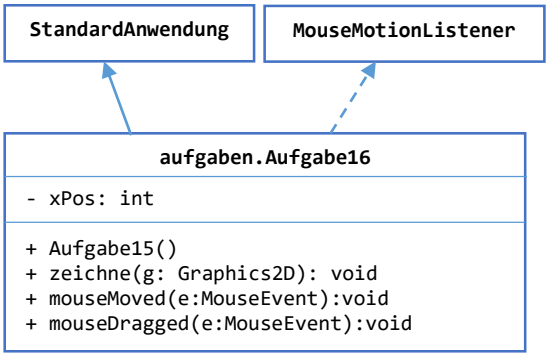
Als nächstes schreiben wir die Methode *zeichne*:

```
@Override
public void zeichne(Graphics2D g){
    g.fillOval(xPos, 40, 30,30);
 }
```

Die Annotation *‚@Override'* zeigt an, dass wir die Methode aus der Basisklasse **überschreiben**.

Das Ergebnis entspricht dem Bild: es erscheint eine 30 Pixel großen Kreis an die Position (10, 40).

Eine kleine Animation



Ist Ihnen aufgefallen, dass wir die Methode *zeichne* nirgendwo aufrufen? Sie wird nämlich von der Methode *paintComponent* der Basisklasse *StandardAnwendung* gerufen. Diese wiederum wird vom Display-Manager des Betriebssystems immer dann gerufen, wenn das Panel neu dargestellt werden muss, z.B. weil sich seine Größe verändert hat.

Wir machen das sichtbar, indem wir jedes Mal, wenn die Maus sich bewegt, die Position des Kreises verändern. Dazu lassen wir zunächst die Klasse *Aufgabe16* das Interface *MouseMotionListener* implementieren und fügen die beiden Methoden *mouseMoved* und *mouseDragged* hinzu wie im UML Diagramm dargestellt.
Tipp: Eclipse fügt die fehlenden Methoden automatisch hinzu, wenn Sie mit der rechten Maustaste auf die Fehlermeldung klicken und ‚*Quick Fix→add unimplemented Methods*' wählen.

Dann rgistriert sich *Aufgabe16* im **Konstruktor** als sein eigener *MouseMotionListener*:

```
this.addMouseMotionListener(this);
```

In der Methode *mouseMoved* fügen Sie jetzt folgende Zeilen ein:

```
xPos = xPos+1;
this.repaint();
```

Die Methode *repaint* weist den Display-Manager an, das Panel sobald wie möglich neu darzustellen.

Wenn die Maus bewegt wird, wandert der Kreis von links nach rechts, verschwindet aber irgendwann rechts aus dem Bildschirm. Fügen Sie zu *zeichne* folgende Zeile hinzu:

```
xPos = (xPos+1) % this.getWidth();
```

Der % Operator heißt **Modulo-Operator**, er berechnet den ganzzahligen Rest bei der Division des linken durch den rechten Operanden. So ergibt z.B. 7%5→2, 9/4→1, 4%5→4.

In unserem Fall sorgt der Modulo-Operator dafür, dass die X-Position immer zwischen 0 und dem rechten Rand des Panels liegt, denn die Methode *getWidth* liefert die aktuelle Breite des *JPanels* in Pixeln.

Erzeugen Sie das neue Paket ***figuren*** und fügen Sie darin folgende Klasse hinzu:

```
package figuren;
public class PlaySymbol {
    private Polygon dreieck = new Polygon();  // java.awt.Polygon
    public PlaySymbol() {
           dreieck.addPoint(40, 15);
           dreieck.addPoint(60, 25);
           dreieck.addPoint(40, 35);
    }

    public void zeichne(Graphics2D g, int x, int y) {
           g.translate(x,y);
           g.setColor(Color.BLACK);
           g.fillRoundRect(0, 0, 90, 50, 25, 25);
           g.setColor(Color.WHITE);
           g.fillPolygon(dreieck);
    }
}
```

Dann ergänzen Sie Klasse *Aufgabe16* um die folgende Instanzvariable:

```
private PlaySymbol play = new PlaySymbol();
```

Und in der Methode *zeichne* in *Aufgabe16* stellen sie es dar:

```
play.zeichne(g, xPos, 130);
```

Versuchen Sie, das Symbol von links oben nach rechts unten über den Bildschirm wandern zu lassen (neue Instanzvariable *yPos*, *getHeight* statt *getWidth* verwenden).

Versuchen Sie, selbst eine oder mehrere Klassen zu schreiben, welche ein Icon auf den Bildschirm zeichnet und animieren Sie es. Der Fantasie sind dabei keine Grenzen gesetzt.

Das Wichtigste aus der aktuellen Vorlesungswoche:

- Es gibt ***elementare Datentypen*** und ***Referenz-Datentypen***.
- Elementare Datentypen gibt es nur wenige, sie werden klein geschrieben. Für uns sind im Moment nur ***int*** (ganze Zahlen), ***double*** (gebrochene Zahlen) sowie ***boolean*** (true, false) wichtig.
- Alle **Klassen sind Referenz-Datentypen**.
- ***Variablen*** sind Namen für eine Speicherzelle, in der Daten eines bestimmten Typs aufgehoben werden können. Um eine Variable zu definieren, schreibt man ihren Datentyp und dahinter den Namen der Variablen. Variablen werden klein geschrieben.
- **Variablen werden** in Java bei der Definition **initialisiert**. Variablen eines elementaren Datentyps erhalten dabei den Wert 0 bzw. false, Variablen eines Referenz-Datentyps den Wert ***null***.
- Bei Variablen eines elementaren Datentyps wir der Wert direkt in der Variable gespeichert.
- Variablen eines Referenz-Datentyps speichern keine Objekte, sondern die **Referenz** auf ein Objekt. Haben sie den Wert *null*, zeigen sie auf kein Objekt.
- Variablen, die innerhalb einer Methode definiert sind, nennt man ***lokale Variablen***. Sie können nur innerhalb der Methode verwendet werden.
- Das Zeichen **=** nennt man **Zuweisungsoperator**. Es speichert den Wert, der rechts von ihm steht in der Variablen, die links davon steht.
- Die gleiche Variable kann sowohl rechts als auch links vom Zuweisungsoperator stehen, z.B
  ```
  zahl = zahl + 1;
  ```
- ***this*** ist immer vom Typ der eigenen Klasse, ***super*** vom Typ der Basisklasse.
- Hat eine abgeleitete Klasse eine Methode, die den gleichen Namen und die gleiche Parameterliste hat wie eine Methode in der Basisklasse, spricht man von einer **überschriebenen Methode**.

- Kommunikation ist der Austausch von Information zwischen Sender und Empfänger. Um die abstrakte Information auszutauschen, muss eine konkrete **Nachricht** verschickt werden, sie besteht aus einem **physilkalischen Signal**, einem **Kanal** zur Übetragung und einer **Interpretationsvorschrift**. Bei ungestörter Kommunikation ist fir Information, die beim Empfänger ankommt die gleiche, welche der Sender geschickt hat.
- Es gibt Raumkanäle (Kabel, Funkstrecken, …) und Zeitkanäle (Speicher: Festplatten, Grabsteine, USB Speicher, …).
- Computer verarbeiten Zeichen, keine Zahlen. Ein Zeichen ist ein Phänomen aus einer **endlichen** Menge von unterscheidbaren physikalischen Phänomenen.
- Eine endliche Menge von Zeichen nennt man **Zeichenvorrat**, die Anzahl der Zeichen nennt man seine **Mächtigkeit**., sie wird durch Betragsstriche angegeben, Beispiel: Z={0,1,2,3,…,9}, |Z|=10.
- Ein Zeichenvorrat, dessen Zeichen geordnet sind, nennt man **Alphabet**.
- Hängt man Zeichen aus einem Zeichenvorrat aneinander, bildet man **Zeichenketten** oder **Worte**. Die Zahl der Zeichen in einer Zeichenkette ist ihre **Länge**, die Position eines Zeichens nennt man **Stelle**.
- Worte über einem Urvorrat kann man wiederum als die Zeichen eines neuen Zeichenvorrates begreifen. Beispiel: aus den Buchstaben unserer Sprache bilden wir Worte. Alle Worte im Duden kann man als Zeichenvorrat ansehen, aus dem Sätze gebildet werden, die wiederum ein Zeichenvorrat sind (die Zahl der möglichen deutschen Sätze ist endlich!). Das Bilden eines neuen Zeichenvorrates **Z** durch Wortbildung über einem Urvorrat **U** nennt man **Abstraktion**. In der Regel gilt |Z|>>|U|.