Bitte lösen Sie Aufgaben 4-7 bis zum 26.04.2020 und **geben Sie Aufgabe 7 ab (1 Punkt).**

# Aufgabe 4: Vererbung und Konstruktor

Als nächstes lernen wir **Vererbung** kennen, eines der wichtigsten Konzepte der objektorientierten Programmierung. Vorher strukturieren wir unser Projekt indem wir anfangen, es in Pakete zu unterteilen.

Erzeugen dazu Sie in Eclipse eine neues Packet: File→new→Packege und nenne es ***aufgaben*** (Pakete werden klein geschrieben) und darin die von *JPanel* abgeleitete Klasse *Aufgabe4*:

```
package aufgaben;
public class Aufgabe4 extends JPanel{ …
```

Das bedeutet, dass Objekte vom Typ *Aufgabe4* alle Eigenschaften und Methoden von *JPanel* **erben**. Dann programmieren Sie den rechts stehenden ***Konstruktor*** von Aufgabe4.

Der **Konstruktor** ist eine Methode, die genau so heißt wie die Klasse (sie weicht also von der Regel ab, dass die Namen von Methoden mit einem Kleinbuchstaben beginnen).

Außerdem hat der Konstruktor keinen Return-Typ (das *void* vor *main*).

```
public Aufgabe4(){
   this.setBackground(Color.GREEN);
   JButton knopf = new JButton("Do not push!");
   this.add(knopf);
}
```

Der Konstruktor wird jedes Mal durchlaufen, wenn ein Objekt vom Typ *Aufgabe4* erzeugt wird. Dabei wird das neue Objekt, welches der Konstruktor initialisiert, mit ***this*** bezeichnet. Da ein *Aufgabe4* Objekt auch ein *JPanel* Objekt ist, erbt es die Methoden *setBackground* und *add* von *JPanel*.

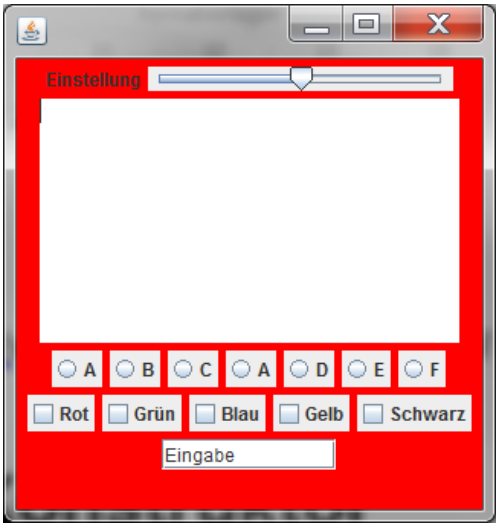
Als nächstes schreiben Sie die rechts gezeigte main-Methode. Diese legt ein Objekt vom Typ *Aufgabe4* (welches ja auch ein *JPanel* ist) auf ein JFrame-Fenster.

```
public static void main(String[] a){
 JFrame fenster = new JFrame("Aufgabe 4");
 fenster.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 fenster.setSize(500,400);
 Aufgabe4 panel = new Aufgabe4();
 fenster.add(panel);
 fenster.setVisible(true);
}
```

# Aufgabe 5: Verschiedene Swing Objekte

Schreiben Sie eine Java-Anwendung, welche gleich aufgebaut ist wie Aufgabe 4, also:

- Die Klasse *aufgaben.Aufgabe5* ist von *JPanel* abgeleitet,
- in *main* wird ein *JFrame* Objekt erzeugt, auf das ein neues Objekt vom Typ *Aufgabe5* gelegt wird,
- im Konstruktor von *Aufgabe5* werden verschiedene Objekte auf das Panel gelegt.




Experimentieren Sie, indem Sie im Konstruktor von *Aufgabe5* unterschiedliche Objekte auf das Panel legen, z.B. vom Typ *JButton*, *JLabel*, *JTextField*, *JTextArea*, *JRadioButton, JCheckBox* oder *JSlider*. Schauen Sie in der API Dokumentation nach, welche Konstruktoren die Klassen anbieten, um verschiedene Eigenschaften (Titel, Größe, u.s.w.) festzulegen.

Beobachten Sie, wie sich die Objekte auf der Oberfläche anordnen, wenn Sie die Größe des Fensters verändern. Man nennt dieses Verhalten ein *Flow-Layout*. Wir werden später andere Layouts für *JPanels* kennenlernen.

# Aufgabe 6

**Strukturierung, Vererbung**

Um nicht immer wieder die gleichen Programmzeilen zu wiederholen, übernehmen Sie die Klasse *StandardAnwendung*, aus dem online Kurs in das neue Paket tools. Erzeugen Sie deshalb zunächst in Ihrem Eclipse Projekt das Paket *tools* und darin die Klasse. Hier eine verkürzte Version:

```
package tools;
public abstract class StandardAnwendung extends JPanel {
   protected JFrame fenster;  //  Abgeleiteten Klassen koennen auf das Fenster der Anwendung zugreifen
         /**
          *  Startet einen Thread in dem ein Objekt der Hauptklasse erzeugt wird.
          */
     protected static void starteAnwendung(){
         StackTraceElement[] trace = Thread.currentThread().getStackTrace();
         String hauptKlasseName = trace[trace.length-1].getClassName();
         // Swing-Anwendungen müssen in einem eigenen Thread laufen
         SwingUtilities.invokeLater(new Runnable(){
          public void run(){
           try {
                  UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
                  Class hauptKlasse = Class.forName(hauptKlasseName);
                  StandardAnwendung app = (StandardAnwendung)hauptKlasse.getConstructor().newInstance();
            } catch (NoSuchMethodException e){
            JOptionPane.showMessageDialog(null, "<html>Ihre Klasse <strong>"+hauptKlasseName
                   +"</strong> hat keinen öffentlichen Standardkonstruktor,\n ",JOptionPane.ERROR_MESSAGE);
               e.printStackTrace();
           }catch (Exception e) {
                           e.printStackTrace();
                           }
                  }
         });

         protected StandardAnwendung(String titel, int b, int h){
                  this.fenster = new JFrame(titel);
                  this.fenster.setSize(b, h);
                  this.fenster.getContentPane().add(this);
                  this.fenster.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                  this.fenster.setVisible(true);
         }
         public void zeichne(Graphics2D g) {
                  // kann von abgeleiteten Klassen überschrieben werden um zu zeichnen
         }
         @Override
         public final void paintComponent(Graphics g) { // final verhindert das Überschreiben in abgeleiteten Klassen
                  super.paintComponent(g);
                  Graphics2D g2 = (Graphics2D)g;
                  g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                  this.zeichne(g2);
         }

         @Override
         public final void paint(Graphics g) {  // final verhindert, dass die Methode von abgeleiteten Klassen überschrieben wird,
                  super.paint(g);               // (damit nicht paint statt paintComponent gerufen wird).
         }
}
```

Diese Klasse vereinfacht unsere Übungen in Zukunft erheblich und sie startet unsere Anwendungen in einem eigenen Thread, was es uns später u.A. ermöglicht, Sound abzuspielen.

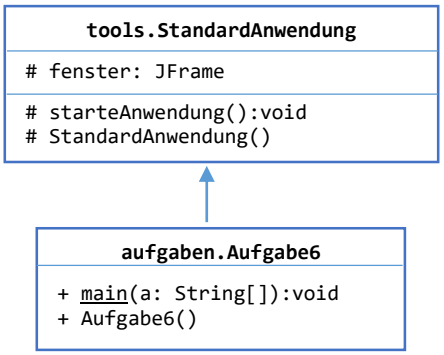
Unsere Aufgaben sehen in Zukunft wie folgt aus:

```
Package aufgaben;
public class Aufgabe6 extends StandardAnwendung {
  public static void main(String[] a){
    starteAnwendung();
  }

  public Aufgabe6(){
       super("Aufgabe 6", 500, 400);  // legt Titel und Größe des Fensters fest

       this.setBackground(Color.RED);
       this.add(new JLabel("Hier nicht klicken: "));
       JButton knopf = new JButton("Klick doch mal");
       this.add(knopf);
  }
}
```

**UML**




Die Struktur der Klassen und die Vererbungshierarchie wird oft in sogenannten **UML-Diagrammnen** dargestellt (**U**nified **M**odelling **L**anguage). In dieser Form kann man die Struktur von Software unabhängig von der verwendeten Sprache präsentieren.

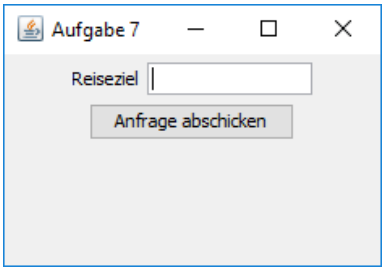
Im rechts gezeigten UML Diagramm werden folgende Elemente verwendet:

- Unter dem Namen der Klasse stehen zunächst ihre **Eigenschaften** (Felder, Variablen). Im Beispiel hat nur *StandardAnwendung* eine Eigenschaft: die Variable fenster vom Typ *JFrame*.
- Die **Sichtbarkeit** einer Variable oder Methode wird durch die vorangestellten Zeichen + und # dargestellt. Es gibt folgende Möglichkeiten, deren Bedeutung wir erst später genauer kennenlernen:
  - \+ public
  - \# protected
  - \- private
- Hinter einer Variablen oder Methode steht ein Doppelpunkt und dann deren **Typ**.
- **Statische** Variable oder Methode sind unterstrichen.

# Aufgabe 7

**Struktur, Vererbung, JTextField**

Schreiben Sie nebenstehende Java Anwendung der Größe 200*150 Pixel, die einen *Jlabel*, ein *JTextField* mit Platz für 12 Zeichen und einen *JButton* zeigt. Geben Sie Ihren Variablen verständliche Namen, z.B. *eingabeFeld* für das *JTextField*.



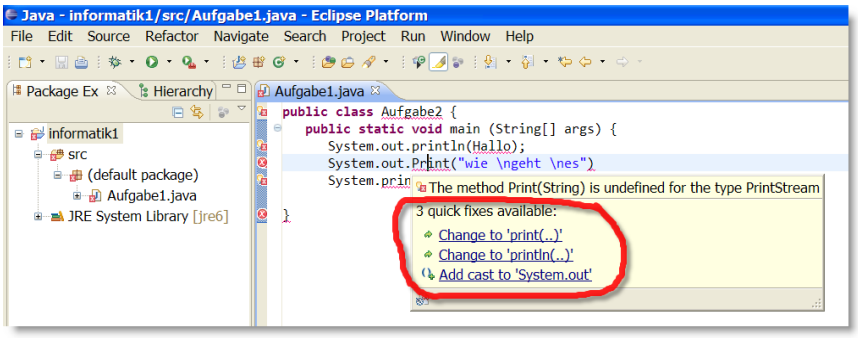

Strukturieren Sie Ihr Programm wie in Aufgabe 6 indem Sie *Aufgabe7* von *StandardAnwendung* ableiten.

Beobachten Sie, was mit der Anordnung der Objekte passiert, wenn Sie das Fenster vergrößern oder verkleinern. Ein JPanel ordnet seine Objekte standardmäßig in einem ***FlowLayout*** an.

# Eclipse-Unterstützung kennenlernen

Machen Sie sich mit folgenden „Zeitsparern“ vertraut:

- *Quick fixes*:
  Syntaxfehler werden nicht nur automatisch im Quelltext Ihres Programmes rot unterstrichen gekennzeichnet. In vielen Fällen macht Ihnen Eclipse sogar Vorschläge zur Problembehebung!



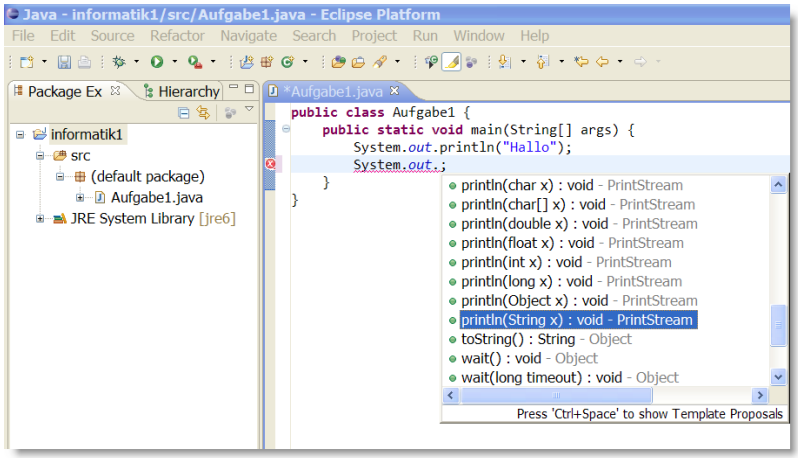

- *Eingebaute API-Dokumentation*: Wenn Sie mit dem Mauszeiger Klassen- und Methodennamen überstreichen, wird Ihnen die passende API-Dokumentation als Tooltip angezeigt.

  Ausprobieren: Überstreichen Sie eine Fehlerstelle. Wenn Eclipse eine Problemlösung anbieten kann, wird diese im Tooltip gezeigt. Bitte wählen Sie allerdings nicht aufs Geradewohl irgendeine Lösung aus, sondern versuchen zuerst das vorliegende Problem zu verstehen!

- *Formatter*: Mit `Strg+Umschalt+F` formatiert Eclipse Ihren Quelltext automatisch neu.
  Ausprobieren: Fügen Sie ein paar Leerzeilen in der Mitte Ihres Programmes ein und rücken Sie ein paar Zeilen wahllos mit zusätzlichen Leerzeichen ein. Drücken Sie dann `Strg+Umschalt+F`.

- *Content Assist*: Eclipse kann Ihnen bei der Eingabe von Methodennamen assistieren, in dem es anhand des Kontexts Ihres Programmes Vermutungen anstellt.

  Ausprobieren: Geben Sie in Ihrem Programm in einer neuen Zeile unterhalb des letzten `System.out.println(…)` folgendes ein: `System.out.`und drücken Sie dann `Strg+LEERTASTE`. Eclipse zeigt Ihnen dann eine Auswahl mit möglichen Methoden an, die für das Objekt *`out`* definiert sind. Wählen Sie mit den Pfeiltasten `println(String x)` aus.




- *Automatisches Eifügen der import-Anweisungen*: Mit `Strg+Umschalt+O` fügt Eclispe automatisch die notwendigen import-Anweisungen ein bzw. entfernt überflüssige Imports.

# Übersicht einiger Shortcuts in Eclipse

| | |
|---|---|
| **STRG + S** | **Speichern** |
| **STRG + Z** | Aktion rückgängig machen |
| **STRG + Pfeiltasten (Links, Rechts)** | Cursor Wörterweise verschieben |
| **STRG + Pfeiltasten (Hoch, Runter)** | Scrollen, ohne das der Cursor mit scrollt |
| **STRG + SHIFT + R** | Schnelles öffnen eines Item |
| **STRG + SHIFT + Pfeiltasten (Hoch, Runter)** | Zur nächsten Methode springen |
| **STRG + SHIFT + G** | Alle Referenzen zur ausgewählten Methode im gesamten Projekt suchen |
| **STRG + Q** | Rücksprung zur Stelle an der der Cursor zuletzt etwas verändert hat |
| **STRG + .** | Sprung zum nächsten Fehler |
| **ALT + Pfeiltasten (Hoch, Runter)** | Verschieben einer kompletten Zeile nach oben/unten |
| **STRG + ALT + Pfeiltasten (Hoch, Runter)** | Vervielfältigung einer bestimmten Zeile |
| **STRG + D** | Löscht aktuelle Zeile in der sich der Cursor befindet |
| **STRG + 7** | Schnelles auskommentieren |
| **F3** | Sprung zur Deklaration der ausgewählten Variable/Methode/Klasse |
| **STRG + SHIFT + F** | Automatische Formatierung |
| **ALT + SHIFT + M** | Lagert markierten Programm-Code in einer neuen Methode aus |
| **STRG + SHIFT + L** | Anzeigen aller Shortcuts |