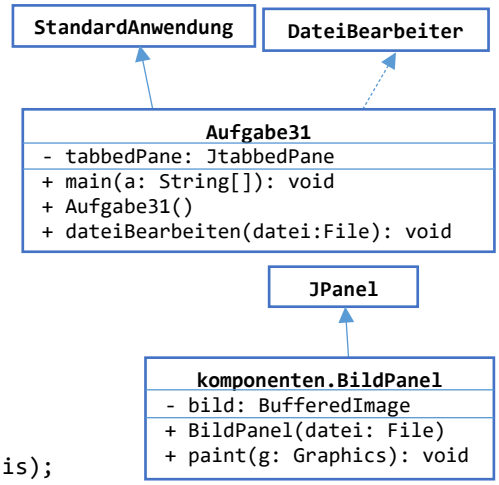
Bitte geben Sie Bildpanel.java und Aufgabe32.java bis zum 24.01.2021 ab!

Aufgabe 31

Bilder Einlesen und anzeigen



Erzeugen Sie eine Kopie der Klasse *Aufgabe29* und nennen sie ***Aufgabe31***.

Die einzige Änderung –abgesehen vom Titel der Anwendung- ist in der Methode ***dateiBearbeiten***: dort erzeugen Sie statt eines *EditPanel* Objekts ein Objekt vom Typ ***BildPanel***.

Die Klasse ***BildPanel*** soll ein Bild öffnen und es darstellen:

Im **Konstruktor** lesen wir das Bild aus der Datei *datei* in die Instanzvariable *bild*:

```
bild = ImageIO.read(datei);
```

dabei müssen IO Fehler mit try/catch abgefangen werden.

In ***paint*** stellen Sie das Bild dar mit

```
g.drawImage(bild, 0, 0, this.getWidth(), this.getHeight(), this);
```

Wir können jetzt Bild-Dateien öffnen und die Bilder werden in unserer Anwendung dargestellt. Allerdings sind die Bilder so an das Fensters angepasst, dass sie verzerrt werden wie rechts gezeigt.

Seitenverhältnis erhalten

Damit das Bild optimal in das Fenster passt, gehen Sie in ***paint*** wie folgt vor:

- Speichern Sie die Größe des Panels in den lokalen float-Variablen ***b*** und ***h*** ,
- berechnen Sie Die Seitenverhältnisse des Panels und des Bildes und als float Variablen.

```
float svPanel=b/h;
float svBild=(float)bild.getWidth()/(float)bild.getHeight();
```

- falls das Seitenverhältnis des Bildes größer ist als das des Panels, berechnen Sie einen neuen Wert für ***h*** aus dem Verhältnis von Panel-Breite zu Bild-Breite:

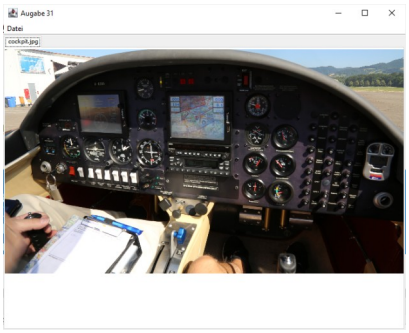
```
h = bild.getHeight()*b/bild.getWidth();
```

- falls das Seitenverhältnis des Bildes kleiner ist als das des Panels, berechnen Sie einen neuen Wert von ***b*** aus dem Verhältnis von Panel-Höhe zu Bild-Höhe:

```
b = bild.getWidth()*h/bild.getHeight();
```

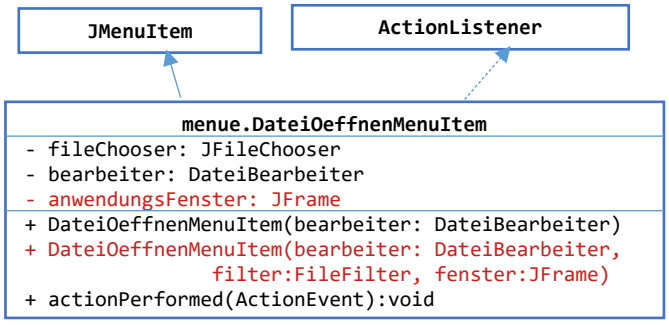
- zeichnen Sie das Bild wie folgt:

```
g.drawImage(bild, 0,0, (int)b, (int)h, this);
```



Versuchen Sie zu verstehen, warum bei den Berechnungen die Typumwandlungen (float) bzw. (int) notwendig sind!

Verbesserter Datei-Öffnen Menüeintrag



Wir wollen noch eine verbesserte Version des Datei-Menüs erstellen:

- Der überladene **Konstruktor** speichert *fenster* in der neuen Instanzvariable, dann prüft er ob filter null ist, wenn nicht ruft er `fileChooser.setFileFilter(filter)`.
- Der alte Konstruktor ruft `this(bearbeiter, null, null)` um zu vermeiden, dass wir Software duplizieren.
- In ***actionPerormed*** übergeben Sie *anwendungsFenster* als ersten Parameter an *showOpenDialog*.

Der Dialog geht jetzt über der Anwendung auf und nicht irgendwo auf dem Desktop. Das ist besonders wertvoll wenn man mit mehreren Bildschirmen arbeitet.

Zusätzlich können wir in ***Aufgabe31*** einen Filter definieren, der nur bestimmte Datei-Endungen zulässt:

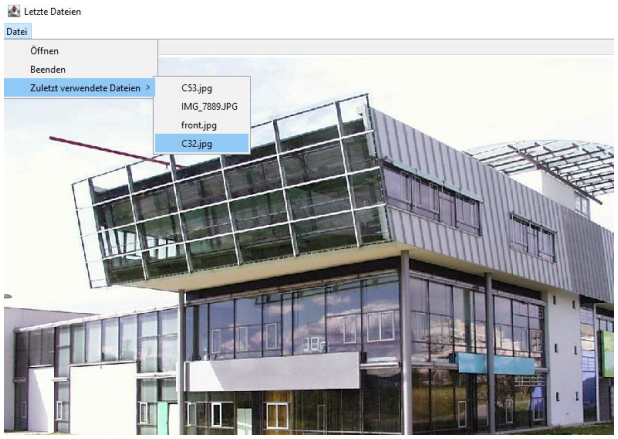
```
FileNameExtensionFilter filter = new FileNameExtensionFilter("Bilder","png","jpg","jpeg","gif");
dateiMenu.add(new DateiOeffnenMenuItem(this, filter, this.fenster));
```

Weitere erlaubte Datei-Endungen könnte z.B. *bmp, tiff, GIF, PNG oder JPG* sein.

Aufgabe 32

Unsere Anwendungen werden sehr viel benutzerfreundlicher wenn wir die Liste der zuletzt geöffneten Dateien speichern wie rechts im Bild gezeigt.

Menü mit den zuletzt geöffneten Dateien



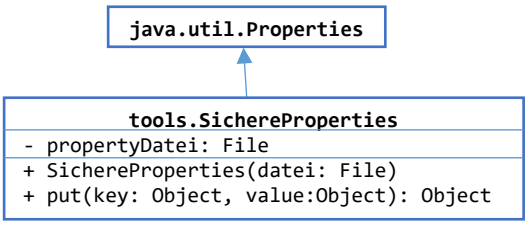
Dazu schreiben wir zunächst die folgende Klasse.

Die Klasse *SichereProperties*

Diese Klasse implementiert eine Liste von Eigenschaften, die permanent, d.h. bei jeder Änderung auf der Festplatte gespeichert werden. Dazu erweitern wir die Klasse *Properties* wie im UML Diagramm gezeigt.

Zur Erinnerung: die Basisklasse *Properties* implementiert eine Map mit den beiden Methoden:

```
put(key;Object, value:Object): Object
getProperty(key: String): String
```



Der **Konstruktor** von *SichereProperties* speichert seinen Parameter in der Instanzvariable, prüft ob wir Lesezugriff auf die angegebene Datei haben (canRead) und ruft dann

```
this.loadFromXML(new FileInputStream(datei));
```

um den letzten Zustand von der Festplatte zu holen. Im Fehlerfall geben Sie nur den Stack-Trace aus.

Die **Methode put** ist überladen (@Override nicht vergessen!). Sie ruft zunächst *super.put* und speichert dann den aktuellen Stand mit

```
this.storeToXML(new FileOutputStream(propertyDatei), "Stand "+ new Date());
```

Auch hier reicht im Fehlerfall ein *printStackTrace*.

Um die Klasse zu testen können Sie folgende main-Methode einfügen:

```
public static void main(String[] args) {
  String dateiname = System.getProperty("user.home")+"/test.prop";
  System.out.println(dateiname);
  SichereProperties p = new SichereProperties(new File(dateiname));
  p.put("Antwort", "42");
  System.out.println(p.getProperty("Antwort"));

}
```

Nach der Ausführung finden Sie in Ihrem Benutzerverzeichnis die rechts gezeigte Datei *test.prop*.

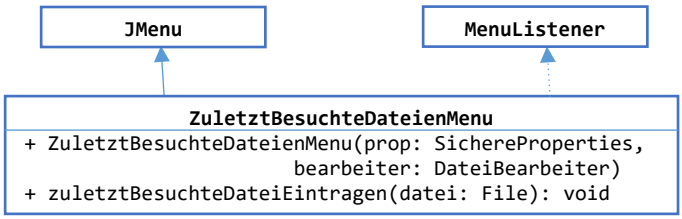
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Letzter Stand Tue Jan 12 16:57:44 CET 2021</comment>
<entry key="Antwort">42</entry>
</properties>
```

Die Klasse *ZuletztBesuchteDateienMenu*

Diese Klasse habe ich Ihnen in den online Kurs gelegt, bitte laden Sie sie von dort herunter.

Ein Objekt der Klasse ist ein Untermenü, das Sie in Ihr Datei-Menü einfügen können wie im Bild weiter oben gezeigt. Es sorgt dafür, dass die zuletzt besuchte Datei immer vorne in der Liste steht. Intern werden die Dateipfade als Eigenschaften mit den Namen *LetzteDatei_0*, *LetzteDatei_0*, u.s.w. in ein *SichereProperties* Objekt eingetragen.

Zur besseren Übersicht finden Sie rechts nochmal die Klasse als UML Diagramm, darin sind nur die beiden öffentlichen Methoden aufgeführt, die man zur Anwendung braucht.



Die Klasse *Aufgabe32*

Um das Menü in Ihre Anwendung einzubauen laden Sie die Klasse ***ZuletztBesuchteDateienMenu*** aus dem Online-Kurs ins Paket *menu*, dann erstellen Sie eine Kopie von *Aufgabe31* und nenn sie *Aufgabe32.*

Im **Konstruktor** erstellen Sie ein SichereProperties Objekt, als Dateinamen nehmen Sie z.B.

```
System.getProperty("user.home")+"/Aufgabe32.prop"
```

Dieses Objekt kann nicht nur zum speichern der zuletzt geöffneten Dateien dienen, sonder Sie können auch andere Eigenschaften damit verwalten und sie permanent speichern.

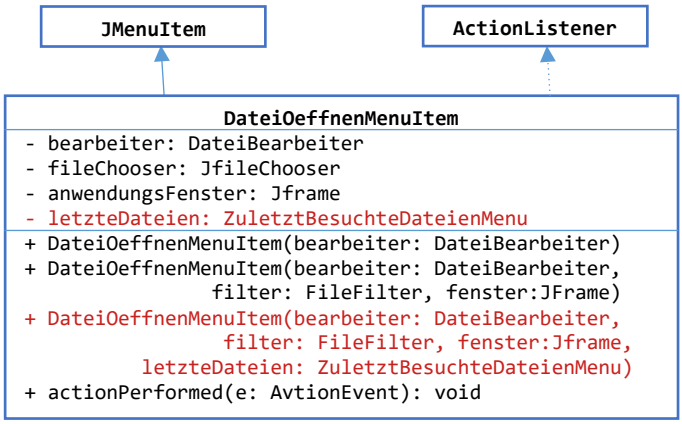
Dann erstellen Sie ein ***ZuletztBesuchteDateienMenu***, das Sie mit *add* zu Ihrem Datei-Menü hinzufügen.

In der Klasse ***DateiOeffnenMenuItem*** müssen wir noch folgende Erweiterungen vornehmen:

Ein **neuer überladener Konstruktor** bekommt einen weiteren Parameter vom Typ *ZuletztBesuchteDateienMenu*, den er in der neuen Instanzvariable speichert. Am besten rufen Sie in den beiden anderen Konstruktoren wieder diesen Konstruktor *this*, um doppelten Code zu vermeiden.

In ***actionPerformed*** fragen Sie ab, ob *letzteDateien* nicht *null* ist und rufen *zuletztBesuchteDateiEintragen*, damit die aktuelle Datei in die Liste der letzten Dateien eingetragen und berücksichtigt wird.

Jetzt können Sie im Konstruktor von ***Aufgabe32*** den neuen Konstruktor von *DateiOeffnenMenuItem* rufen, und das ganze sollte funktionieren.



Hinweise zur Prüfung:

Programmieren sollten Sie folgendes können:

- Swing Anwendungen mit unserer Klasse *StandardAnwendung*
- Array-Methoden
- Rekursionen
- Selection-Sort und Bubble-Sort
- Stack, Queue
- Binärbaum
- Suchbaum
- Heap (kommt noch)

Und **theoretische** Fragen zu allen Kapiteln beantworten können sowie Suchbäume, AVL-Bäume und Heap auf Papier konstruieren und für einen Baum die unterschiedlichen Ordnungen angeben.