Informatik 2

Aufgabensammlung

Uwe Schulz, Simon Wiest

Organisation der Informatik Übungen

Die Note für das Modul Informatik 2 wird wie folgt ermittelt:

- Im Laufe des Semesters geben Sie jede Woche eine Aufgabe ab, die Abgabefrist endet jeweils am Sonntag um 24 Uhr. Für jede **fristgerecht** eingereichte Aufgabe erhalte Sie einen oder zwei Punkte, maximal 20 Punkte
- Eine Abgabe besteht aus einem selbst geschriebenen Java Programm. Bei den Java Programmieraufgaben mache ich Stichproben, ob es sich um eine Lösung der gestellten Aufgabe handelt, auch hier muss die Lösung nicht perfekt sein.
- Gegen Mitte des Semesters werden Sie zu einem Interview aufgefordert, bei dem ich Ihnen zu der zuletzt abgegebenen Aufgabe und zum bisherigen Stoff Fragen stelle. Für das Interview gibt es bis zu 20 Punkte.
- Nach Ende der Vorlesungen ist eine schriftliche Klausur geplant, die mit maximal 60 Punkten bewertet wird, also 60% der Note ausmacht. Falls im Februar aufgrund der Corona-Krise keine Klausur möglich ist, werde ich mit jedem nochmals ein ausführliches Interview durchführen.
- Die Informatik Note wird ergibt sich aus den erreichten Punkten von maximal 100 Punkten.
- **Wenn Sie an der Klausur nicht teilnehmen, gilt das Fach als nicht bestanden, falls Sie kein ärztliches Attest eingereicht haben.** Bei Vorlage eines ärztlichen Attests ist ein Nachtermin nach Vereinbarung möglich. Die Interviews sind Prüfungsleistungen, es sind keine Hilfsmittel erlaubt.
- Bitte vergessen Sie nicht, Informatik 2 als Prüfungsleistung anzumelden!

Bitte lösen Sie Aufgaben 1-3 bis zum 18.10.2020 **und geben Aufgabe 1 bis zum 18.10.2020 ab!**

Ziele des heutigen Übungstermins

- Sie gewöhnen sich wieder an den Umgang mit Eclipse.
- Sie richten das neue Projekt „Informatik2“ ein und strukturieren es.
- Sie bringen die Anwendung ‚BillardSpiel“ aus dem ersten Semester zum Laufen
- Sie schreiben eine kleine Swing Anwendungen und ein kleines Spiel.
- Sie lernen eine Anwendung als ausführbare Jar-Datei zu exportieren.

Aufgabe 1

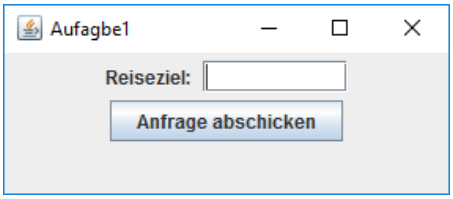
Zunächst richten Sie sich wieder eine Übungsumgebung ein, hier nochmal die wichtigsten Schritte:

1. Starten Sie Eclipse und klicken Sie die „*Welcome to Eclipse“*-Seite weg.
2. Legen Sie ein neues Java-Projekt an: auf ***File->New->Project*** klicken und dort ***Java Project*** auswählen. Wichtig: **keinen anderen Projekt-Typ nehmen**!
   Nennen Sie das Projekt z.B. **java_kurs_2**.
3. Laden Sie die Klasse ***StandardAnwendung*** aus dem online Kurs in das neue Paket ***tools***.
   Tipp: Wenn Sie den Text der einer Klasse mit STRG-A und STRG-C in die Zwischenablage kopiert haben, können Sie in Eclipse mit Rechtsklick auf das Paket *tools* die Klasse mit ‚*Paste*‘ einfügen.
4. Erzeugen Sie das neue Paket ***aufgaben***.
5. Erzeugen Sie die Klasse *aufgaben.Aufgabe01*, durch Klick auf ***File->New->Class*** und geben Sie als Name ***Aufgabe01*** ein und tippen das unten stehende Programm ein.
6. Führen Sie das Programm aus, entweder durch Klick auf die Schaltfläche mit dem weißen Dreieck im grünen Kreis oder mit Rechtsklick (Kontextmenü) und „*Run as → Java Application*“.

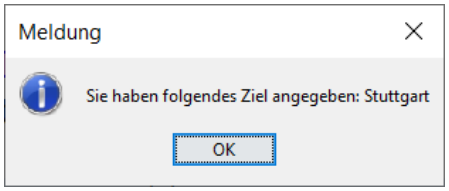
Lesen Sie die Klasse *StandardAnwendung* durch. Sie ist so strukturiert, dass im *main*-Programm einer abgeleiteten Klasse nur noch die Methode *starteAnwendung* gerufen werden muss:

```
package aufgaben;
public class Aufgabe01 extends StandardAnwendung {
  public static void main(String[] a){
    starteAnwendung();
  }

  public Aufgabe01(){
       super("Aufgabe 1", 300,150);
       this.add(new JLabel("Reiseziel: "));
       …
  }
}
```



Ergänzen Sie diese Klasse so, dass die oben gezeigte Anwendung *entsteht*, die einen *Jlabel*, ein *JTextField* der Länge 9 und einen *JButton* zeigt. Geben Sie Ihren Variablen verständliche Namen, z.B. *eingabe* für das *JTextField*.



Bei Klick auf den Button soll das eingegebene Reiseziel als Meldung ausgegeben werden, Z.B. „Sie haben folgendes Ziel eingegeben: Stuttgart“ (JOptionPane.showMessageDialog). Falls das Eingabefeld leer ist, melden Sie das als Fehler, z.B. „Leere Eingabe“.

Schreiben Sie dazu eine von JButton abgeleitete Klasse, die ActionListener implementiert und eine Instanzvariable vom Typ JTextField hat.

Aufgabe 2

Spiel, Steuerung mit Tastatur

Jetzt wollen wir wieder unsere Umgebung für die Programmierung von Spielen aus dem letzten Semester aktivieren und wir lernen, wie man Figuren mit den Tasten steuern kann. Wir schreiben ein kleines Spiel in dem zwei Figuren mit den Tasten gesteuert wird. Die Figuren stoppen, wenn sie den Rand berühren.

Übernehmen Sie zunächst die Klassen ***Spiel*** und ***Klang*** aus dem online Kurs in das Paket *tools* und die Klasse ***SpielFigur*** in das Paket *figuren*.

Wichtig: Die Klasse *SpielFigur* im online Kurs habe ich um die Methode *aktiviereTastenSteuerung* ergänzt, also nicht die SpielFigur aus dem letzten Semester nehmen!

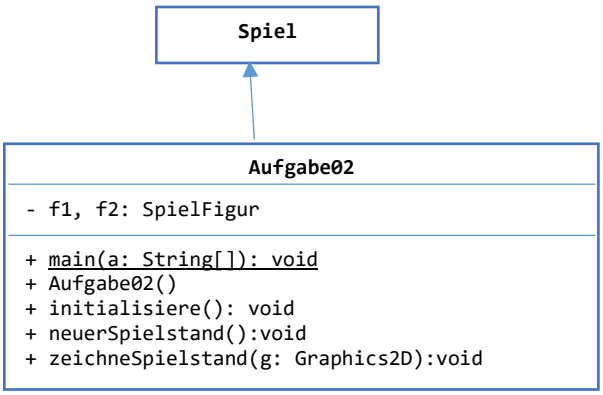
Falls bei der Übernahme von ***Klang*** folgender Fehler angezeigt wird:

Access Restriction: The Type ‚Media Player' ist not API

öffnen Sie Window→ Preferences→Java→Compiler→Errors/Warnings→Depricated and restricted API bei *Forbidden reference (access rules)* ändern Sie *Error* in *Warning* oder *Ignore*.

Nehmen Sie sich die Zeit und lesen Sie alle Klassen aufmerksam durch und versuchen Sie soweit wie möglich alle Zusammenhänge zu verstehen.

Erstellen Sie im Paket *aufgaben* die von *Spiel* abgeleitete Anwendung ***aufgaben.Aufgabe02***.



Der **Konstruktor** ruft den Basisklassenkonstruktor.

Die Methode ***initialisiere*** führt folgende Schritte durch:

- Erzeugen der Spielfiguren,
- Aktivieren der Pfeiltastensteuerung für die Spielfigur (Methode aktiviereTastenSteuerung), eine Figur soll mit den Tasten WASD, die andere mit den Pfeiltasten gesteuert werden,
- Setzen der Randreaktion für die Spielfiguren, welche die Figur stoppt, sobald die Figur den Rand berührt (Methode *setRandReaktion*)

In ***neuerSpielstand*** werden die Spielfiguren bewegt, in ***zeichneSpielstand*** werden sie dargestellt. Mit den Pfeiltasten bzw. den Tasten wasd können Sie jetzt zwei Spieler auf der gleichen Tastatur jeweils eine der Spielfiguren steuern.
Wenn Sie für die Spielfiguren eine Dämpfung von 0.9 definieren, bewegen sie sich nur solange eine Taste gedrückt ist – probieren Sie es aus!

Executable Jar erzeugen

Um eine Java Anwendung an Dritte weiterzugeben, muss sie als ausführbare Jar Datei exportiert werden. Eine *Jar*-Dateien (Java Archive) enthält alle Dateien zu einem Projekt, also sowohl die *.class* Dateien als auch Bilder, Töne, HTML-Dateien u.s.w. Sie ist ähnlich aufgebaut wie ein ZIP-Archiv und kann z.B. auch mit 7-zip geöffnet werden. Sie enthält in einer sogenannten Manifest-Datei die Information, welche Klasse die *main*-Methode enthält, die bei Doppelklick auf die Jar-Datei gestartet werden soll.

Um in Eclipse eine *Jar*-Datei zu erzeugen, Klicken Sie auf „File→Export" und wählen „Runnable Jar File" aus. Nach Klick auf „Next" wählen Sie unter „Launch configuration" Aufgabe02 und als „Export Destination" den Ort und Namen der Jar-Datei, z.B. *Desktop* und *Aufgabe02.jar*.

Mit einem Doppelklick auf die Jar-Datei können Sie Ihre Anwendung jetzt starten. Das funktioniert auf jedem Rechner, auf dem eine passende Java JRE installiert ist.

Aufgabe 3

Billard Spiel

In dieser Aufgabe müssen Sie nichts programmieren, sondern Sie installieren die Musterlösung des Billardspiels aus dem letzten Semester.

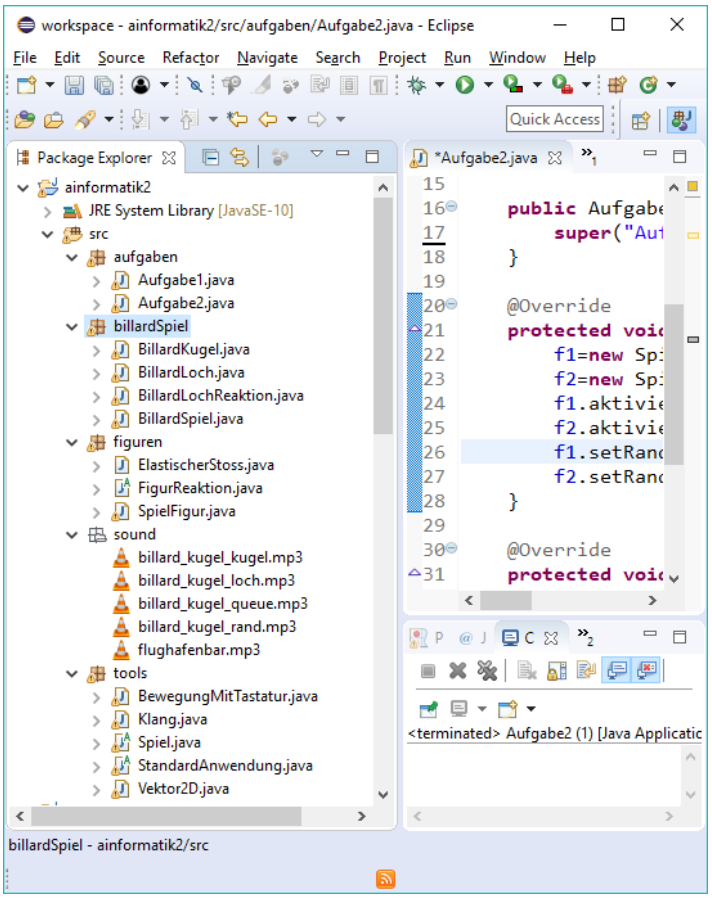
Übernehmen Sie die Klasse ***Vektor2D*** aus dem persönlichen Stundenplan in das Paket *tools* sowie die Klassen ***FigurReaktion*** und ***ElastischerStoss*** in das Paket *figuren*.

Übernehmen Sie in das Paket *billardSpiel* die Klassen ***BillardSpiel***, ***BillardKugel***, ***BillardLoch*** und ***BillardLochReaktion*** aus dem persönlichen Stundenplan.

In das Verzeichnis *sound* übernehmen Sie die Dateien aus dem entsprechenden Verzeichnis im persönlichen Stundenplan.
Tipp: Laden Sie zunächst alle Dateien nach *Downloads* oder *Desktop* herunter und importieren sie anschließend in Eclipse:

- Rechtsklick auf das Paket *sound*,
- *Import→General→File System*,
- Dateien auswählen und ‚*Finish*' klicken.

Ihr Projekt sollte jetzt die rechts gezeigte Struktur aufweisen.



Starten Sie ***BillardSpiel*** und lesen Sie die Klassen aufmerksam durch um möglichst viel zu verstehen. Im Laufe des Semesters werden wir das Spiel grafisch noch ein wenig verbessern.

Ein kurzer Überblick zu den Swing Klassen

Einen guter Einstieg in die Swing Klassen finden Sie hier:
https://de.wikibooks.org/wiki/Java_Standard:_Grafische_Oberfl%C3%A4chen_mit_Swing

Eine Swing-Anwendung besteht aus einem Objekt vom Typ ***JFrame***, das ist ein Fenster, auf das ein ***JPanel*** Objekt gelegt wird, auf dem die grafischen und interaktiven Elemente platziert werden.

Um die Anordnung der Elemente auf einem *JPanel* zu beeinflussen gibt man dem Panel einen Layout-Manager, z.B. `panel.setLayout(new BorderLayout)`. Wir verwenden bisher nur das Standard-Layout (***FlowLayout***), bei dem die Elemente zeilenweise von links nach rechts angeordnet werden sowie den ***BorderLayout***, der die fünf Bereiche NORTH, SOUTH,EAST,WEST und CENTER kennt, auf die man ein Element gezielt platzieren kann (`panel.add(new JButton(”Klick”), BorderLayout.NORTH)`. Ein wichtiger Nebeneffekt des *BorderLayout* ist, dass sich die darauf platzierten Elemente maximal ausdehnen, während sie beim *FlowLayout* nur den minimal benötigten Platz beanspruchen.

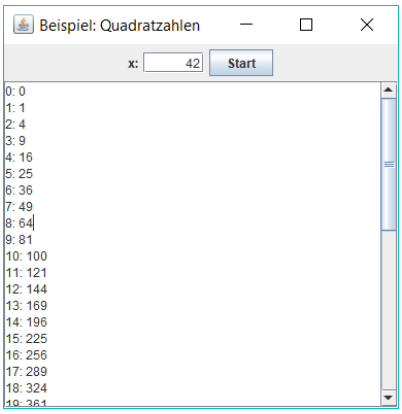
Die wichtigsten Swing Element sind:

- JLabel: ein nicht aktives Element mit einem Text oder Bild
- JButton: ein klickbares Element mit Text oder Bild, das mit *addActionListener* aktiviert wird.
- JTextField: ein einzeiliges Eingabefeld für Text
- JTextArea: ein mehrzeiliges Ein/Ausgabefeld für Text
- JScrollPane: Ein Feld mit Scroll Balken, das in der Regel um eine JTextArea gelegt wird (`panel.add(new JScrollPane(new JTextArea()))`
- JTabbedPane Ein Panel das mehrere Reiter mit eingebetteten Komponenten hat

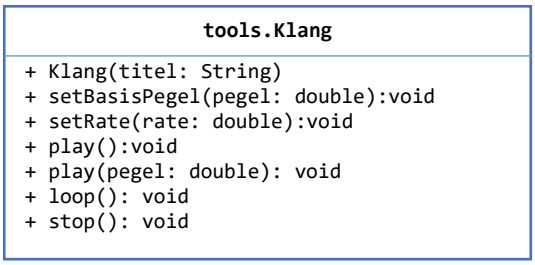
Im online Kurs finden Sie eine Kurzreferenz der wichtigsten Klassen.

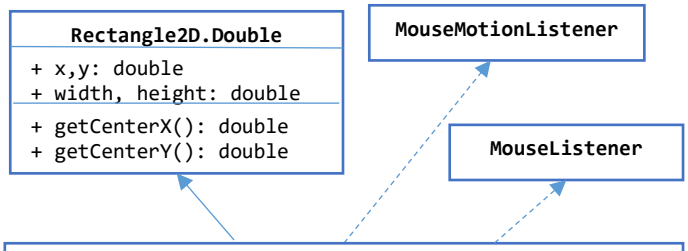
Um die Arbeit mit Swing Anwendungen zu erleichtern gibt es ein kleines Framework aus dem ersten Semester, das aus den Klassen ***StandardAnwendung***, ***Spiel***, ***Spielfigur*** und ***Klang*** besteht, eine Übersicht dieser Klassen mit Beispielen finden Sie auf der nächsten Seite.
Das folgende Beispiel ist eine komplette Anwendung:

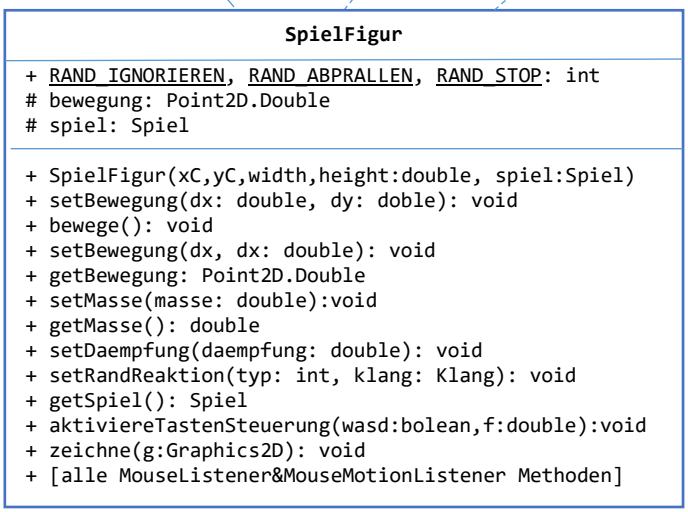
```
public class Quadrate extends StandardAnwendung implements ActionListener{
        private JTextField eingabe = new JTextField(5);
        private JTextArea ausgabe=new JTextArea();
        public static void main(String[] a) {
                starteAnwendung();
        }
        public Quadrate () {
                super("Beispiel: Quadratzahlen", 500,400);
                this.setLayout(new BorderLayout());
                JPanel oben = new JPanel();
                oben.add(new JLabel("x:"));
                oben.add(eingabe);
                JButton knopf = new JButton("Start");
                knopf.addActionListener(this);
                oben.add(knopf);
                this.add(oben, BorderLayout.NORTH);
                this.add(new JScrollPane(ausgabe), BorderLayout.CENTER);
        }
        @Override
        public void actionPerformed(ActionEvent arg0) {
                int n = Integer.parseInt(eingabe.getText());
                ausgabe.setText("");
                for(int i=0;i<n;i++) {
                        ausgabe.append(""+i+": "+(i*i)+"\n");
                }
        }
}
```

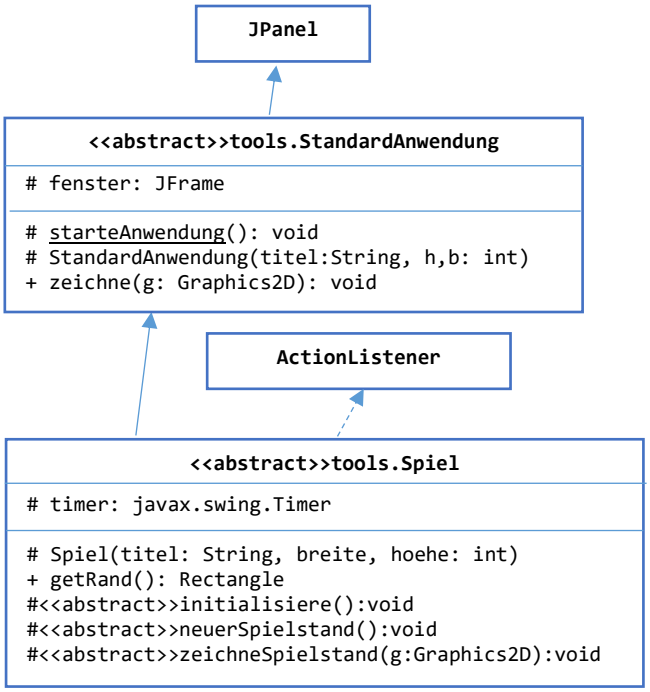


Das Framework für die Informatik-Übungen










```
//  Eine einfache Anwendung
//
public class Beispiel extends StandardAnwendung {
  public static void main(String[] a) {
    starteAnwendung();
  }

  public Beispiel() {
        super("Beispiel", 400, 400);
        this.add(new JLabel("Hallo Welt"));
  }
}
```

```
//  Ein einfaches Spiel
//
public class Beispiel extends Spiel{
 public static void main(String[] a) {
    starteAnwendung();
 }
 private SpielFigur figur;

 public Beispiel() {
    super("Beispiel", 600, 500);
 }
 @Override
 public void initialisiere() {
  figur=new SpielFigur(90, 90, 20, 30, this);
  figur.setBewegung(0.5, 0.2);
 }
 @Override
 public void neuerSpielstand() {
   figur.bewege();
 }
 @Override
 public void zeichneSpielstand(g: Graphics2D) {
   figur.zeichne(g);
 }
}
```

```
//  Ein einfache Spielfigur
//
public class MausOverEffektFigur extends SpielFigur {
 private boolean mausInFigur;

 public MausOverEffektFigur(double xC, double yC, Spiel s){
        super(xC, yC, 40, 40, s);
        this.setBewegung(1, 0);
 }
 @Override
 public void zeichne(Graphics2D g) {
        g.setColor(Color.BLUE);
        if(mausInFigur) g.setColor(Color.YELLOW);
        g.fill(this);
 }
 @Override
 public void mouseMoved(MouseEvent e) {
        mausInFigur = this.contains(e.getPoint());
  }
}
```