Labor
Software Engineering Praktikum
mukB/MI/Units im SoSe25

Prof. Dr. Katharina Mehner-Heindl
Medien und Informationswesen, Hochschule Offenburg

Katharina.Mehner-Heindl@hs-offenburg.de

Modulhandbuch u. Stupo(s)

Modul Software Engineering mukB/MI/temporär Units:
Labor Praktikum Software Engineering **1 SWS**

Arbeitsaufwand

- 15 Stunden Anwesenheit während der Laborzeit
- 30 Stunden Vor-/ Nachbereitung bzw. Arbeit am Rechner außerhalb der pflichtgemäßen Anwesenheitszeit

Studien- und Prüfungsleistungen

- Laborarbeit LA unbenotet
- Anwesenheitspflicht ab dem 1. Termin, Details siehe Moodle
- Gesamtnote des Moduls berechnet sich nur aus der Klausur
- Das Modul ist nur bestanden, wenn alle Veranstaltungen bestanden sind
- Credits nur für das gesamte Modul

Verpflichtende Laboranmeldung im Hochschulportal

- Wer die Frist verpasst, kann das Labor dieses Semester nicht durchführen
- Wer sich nicht abmeldet aber nicht mitarbeitet/nicht besteht, bekommt ein NB

Ablauf des Labors

- Ziel: Bearbeitung verschiedener Aufgaben bei der Softwareentwicklung
  - Jeweils kurze Einführung durch Dozenten
  - Umfangreiche Eigenleistungen
  - Programmiersprache Java
  - Verwendung Eclipse oder IntelliJ
  - Finale Abgabe per **Edugit**
  - Zweiergruppen
    - Eigenleistung, Abschreiben gilt als Plagiarismus, auch Abschreiben aus dem Internet etc.
- Abgaben (Meilensteine) laut Terminen in Moodle
- Aufgabenstellung: Computerspiel

Wasser-Sortier-Puzzle

- 4 farbige Flüssigkeiten aufgeteilt auf 4 Gläser
  - Je Farbe 3 Portionen mit Volumen von ¼ Glas
  - Zufällige Verteilung (auf 3 od. 4 Gläser ist offen, kann festgelegt sein)
  - 2 Portionen von gleicher Farbe müssen zusammenbleiben
- Ziel: Durch Umgießen die Farben trennen!
  - Farbportion/Glas durch Mausklick (o.ä.) Wählen, durch zweiten Klick (o.ä.) Ziel-Glas wählen
  - Timer läuft, eventuell Grenze setzen, Ergebnis bekannt geben (gelöst/ungelöst/nicht mehr lösbar, Zeit)
  - Bei Units kann grafische Oberfläche durch Kommandozeile ersetzt werden.
  - Gestalterische Abänderung möglich, z.B. horizontal Ställe mit je 4 Tieren sortieren…

Implementierungshinweise

- Verwenden Sie als logische Datenstruktur einen zweidimensionalen Array:

```
int[ ][ ] area = {      {0,0,0,0,0,0,0,0,0,0},
                        {0,0,0,0,0,0,0,0,0,0},
                        {0,0,0,0,0,0,0,0,0,0},
                        {0,0,0,0,0,0,0,0,0,0},  };   /*0 für leer, andere Zahlen für Elemente*/
```

- Timer in Java für unabhängige Ausführung

```
javax.swing.Timer timer =   new javax.swing.Timer(1000, new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                if(/* Stop Bedingung */) { timer.stop(); }
                                /* weitere Anweisungen */ }
                });
                timer.start();
```

Meilensteine (Termine siehe Moodle)

- Pflichtenheft & UI Design für Software-Spiel (Emailabgabe)
  - Schreiben Sie ca. **8 Anforderungen** an Ihre Software auf (z.B. Funktion, Gestaltung, Computerstrategie, etc.)
  - UI-Prototyp als Java Eclipse Projekt, d.h. erste Version Startbildschirm mit Spielbrett, **ohne Reaktion auf Mausklicks und ohne sonstige Funktionalität**
- Use Case Diagramm des Software-Spiels (Vorzeigen)
  - 1 Systemkasten mit 1 Aktor und minimal 4 Use Cases, 1x include, 1x extend. 1 textuelle Beschreibung eines Use Case wie in der Vorlesung
- Klassendiagramme (Vorzeigen)
  - 1 zusammenhängendes Klassendiagramm: z.B. Spieler, Spielfeld, Spielfeldzustand… Klassen, Assoziationen, Attribute mit einfachen Typen, Methoden (ohne Parameterlisten), Kardinalitäten, Rollen oder Assoziationsnamen
- Sequenzdiagramme u. Zustandsdiagramm (Vorzeigen)
  - 1 Sequenzdiagr. passend zum Klassendiagramm, 1 Option (od. Loop/Verwzweigung)
  - 1 Zustandsdiagr.: Spielablauf, Spielzug, Gewinner…
- Testen (Vorzeigen)
  - Whitebox-Testen des eigenen Programms mit JUnit (1 Testfall programmieren)
  - Blackbox-Testen eines eigenen/fremden Programms (1 Testfall dokumentieren)
- Vollständiges Programm (Abgabe über **Edugit**)
  - Lauffähige Implementierung als Eclipse/IntelliJ-Export (.zip)

Erfordernisse Pflichtenheft

- **Funktionale** Anforderungen beschreiben Aktionen, Abläufe, Möglichkeiten, z.B.:
  - F1: Die Anwendung erlaubt das Spielen von …
    F2: Es sollen zwei menschliche Spieler gegeneinander spielen können.
    F2.1: Jeder Spieler kann das Spiel jederzeit beenden.
    ...
- **Nicht-funktionale** Anforderungen weitere Eigenschaften und Qualität, z.B.:
  - N1: User Interface nutzt Maus.
    N2: Kontrastreiche Farbpalette und hohe Auflösung von Icons.
    N2.1: maximal 7 Farben
    N3: Start-Up-Zeit unter 1 sec, Antwortszeit unter 0,5 sec.
    N4: Verfügbarkeit von 99,9 %.
    N5: Die Anwendung speichert keine personenbezogenen Daten.
    ...
- **Technische** Anforderungen beziehen sich auf die Umsetzung (der ganzen Anwendung oder von Teilen), z.B. :
  - T1 Die Anwendung soll in Java realisiert werden
    T1.1 Für die GUI werden die AWT/Swing Bibliotheken genutzt.
    T2 Die Anwendung soll nicht mehr als x MB auf der Festplatte belegen
    ...
- Bitte von der Struktur genauso wie hier aufbauen!
- Bitte inhaltlich nicht einfach hiervon abschreiben, sondern abwandeln/ergänzen!
- Bitte **widerspruchsfrei** und **überprüfbar**, z.b. mit ja/nein!
- Bitte mit fortlaufender **Nummerierung/Untergliederungen**

Erfordernisse bei UML

- Use Cases
  - Namen sind Vorgänge, keine Dinge oder Zustände
  - <include> bedeutet: das ein Use-Case das Verhalten des anderen **immer** importiert
  - <extend> bedeutet: das Verhalten eines Use-Cases wird **optional** erweitert
  - zu einem <extend> gehört immer ein Extension Point und eine Bedingung für die optionale Erweiterung. Beides wird in den Use Case gezeichnet.
- Klassendiagramm
  - Namen sind Personen oder Dinge
  - Nicht das ganze System in wenige Klassen packen, sondern logisch trennen.
- Sequenzdiagramme
  - Konsistenz mit Klassendiagrammen, d.h. Instanzen haben Klassen
- Zustandsdiagramme
  - Namen sind Zustände, seltener Vorgänge, niemals Personen oder Dinge
  - Möglich für das gesamte System (und zusätzlich bei großen Systemen für Teilsysteme) und möglich für jede Klasse (mit Zustandsänderungen) unter Angabe der Methoden
  - fassen Abläufe zusammen, d.h. nicht nur einen konkreten Ablauf angeben und nicht für jeden Ablauf eigene Zustände erfinden sondern abstrahieren von mehreren möglichen Abläufen durch Zusammenlegen von Zuständen
- Die korrekte Verwendung der Notation und die Konsistenz zwischen den Diagrammen ist wichtiger, als umfangreiche Diagramme zu erstellen. Probieren Sie möglichst viele syntaktische Elemente aus.

Einsatz der UML Diagramme

1. *Objektorientierte Analyse*, d.h. für eine Umsetzung der Anforderungen in Strukturen und Abläufe: Hier wird die Frage beantwortet, *was* macht die Software, aber nicht, *wie* macht sie es. Zur Analyse gehört, welche Klassen und Objekte in der Software mit welchen Eigenschaften repräsentiert werden müssen, aber nicht die genaue Auswahl einer optimalen Datenstruktur.
   - Z.B. wird modelliert, dass die Software überprüfen muss, ob ein das Spiel gewonnen wurde, aber nicht in Abhängigkeit von einer zu wählenden Implementierung der Datenstruktur.
   - Z.B. werden die Eigenschaften von Spieler und Spielfeld modelliert.
   - Z.B. werden die Spielregeln modelliert sowie Spielstart, Spielablauf und Spielende.
   - Z.B. wird die Schnittstelle zwischen Nutzer und Software modelliert
2. *Objektorientiertes Design*, d.h. für ein genaues Vorbild bzw. Abbild der Umsetzung in der Software: Hier werden zur Analyse Details hinzugefügt, die für die Implementierung wichtig sind, z.B. Datenstrukturen, Optimierung von Algorithmen oder Verwendung von Design Patterns.

Erfordernisse Präsentation

- Präsentation:
  - Das eigene Spiel demonstrieren
  - Ein Detail der Implementierung erklären, z.B. die Klassenstruktur, die Gewinnerermittlung oder einen anderen interessanten Aspekt
- Ca. 5 Minuten
- Vorher testen und üben ;-)

Entwicklungsumgebung

- Eclipse mit Java Standard Edition
  - Alternativen wie IntelliJ möglich, aber kein Support
- Java Swing/AWT für das GUI
  - **Beispiel in Moodle (zip-Datei mit Eclipse importieren)**
  - Andere Alternativen zulässig, aber kein Support
- UML Modelle als Handzeichnung
  - Es macht wenig Sinn, die Diagramme aus Java-Code zu generieren. Dabei lernt man nichts und der Detailierungsgrad passt nicht.
  - Manche Werkzeuge verwenden falsche Notation.
  - Gute Übung für die Klausur
- JUnit Test mit Eclipse für die Durchführung des Whitbox-Tests (keine Alternative)
- GIT-Repository

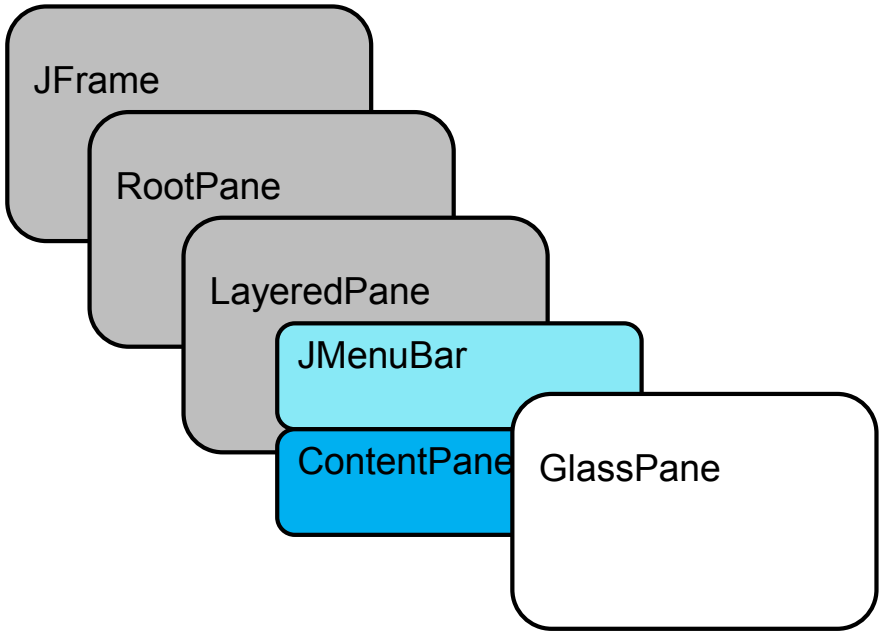
Hinweise Java Programmierung

- Verwenden Sie die Objektorientierung richtig
  - Definieren Sie mehr als eine Klasse!
  - **Erzeugen Sie Instanzen!**
  - Rufen Sie Methoden auf!
- Verwenden Sie Vererbung sinnvoll
- Verwenden Sie ***static*** nur wenn Sie sicher sind
- Verwenden Sie Model-View-Controler

UI mit Swing

- GUI Bibliothek
  - Bestandteil der Java Foundation Classes
  - Nicht threadsicher
  - z.B. http://openbook.galileocomputing.de/javainsel9/
- Prinzip: Container mit Layoutmanagern für Komponenten.
- Fenster, Dialoge, etc. mit
  - Menüs
  - Schaltflächen
  - Text und Grafiken
  - Direkte Grafikausgabe mittels JPanel und überschreiben der paintComponent() Methode
  - Action- und MouseListener für die Interaktion

Aufbau des JFrame



JFrame - Fenster erweitern

```
import javax.swing.*;

public class BeispielFrame extends JFrame {
  public BeispielFrame)                                    // Konstruktor
  {
    super(„Beispiel Titel");                               // Konstruktor der parent-Klasse
    this.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    this.setSize( 600, 200 );
    this.setResizable(false);                              // verhindert Fenster-Resize
    JLabel lb = new  JLabel( “Copyright by HS Offenburg!“);
    f.add(lb);                                             // früher: f.getContentPane().add(lb);
    f.pack();                                              // minimale Größe für Inhalte
    f.setVisible( true );
  }
}
```

Fenstererzeugung thread-sicher machen

```
public static void main( String[] args ) {

                    // Thread-sicherer start des Fensters.
                    javax.swing.SwingUtilities.invokeLater (
                                        new Runnable() {
                                            public void run() {
                                                    try {
                                                        BeispielFrame frame = new BeispielFrame();
                                                        frame.setVisible(true);
                                                    } catch (Exception e) {
                                                        e.printStackTrace();
                                                    }
                                            }
                                        }
                    );
          }
```

JFrame ohne LayoutManager (absolute Positionierung von Komponenten)

```
// im Konstruktor der eignen JFrame
…
this.setLayout(null);
…
spielbrett = new MyJPanel();
spielbrett.setSize(200, 200);
spielbrett.setLocation(10, 20);
this.add(spielbrett);
…
```

Typische Komponenten

- JMenuBar und JMenu
- JButton
  - setText, wiederholt aufrufbar
  - setIcon, wiederholt aufrufbar
  - setOpaque(false), erzeugt Transparenz
- JLabel
  - setText, wiederholt aufrufbar

Beispiel: Label mit MouseListener

```
label.addMouseListener( new MouseAdapter() {
      @Override public void mouseClicked( MouseEvent e ) {
        if ( e.getClickCount() > 1 )
          System.exit( 0 );
      }
    } );
```

Ereignisbehandlung

Anonymen Listener erzeugen (oder WindowListener implementieren)

```
addWindowListener( new WindowAdapter() {
    @Override public void windowClosing(
WindowEvent e ) {
      System.exit( 0 );
    }
  } );
```

Beispiel: Button mit ActionListener & actionPerformed()

```
ActionListener al = new ActionListener() {
  @Override public void actionPerformed( ActionEvent e ) {
    button1.setIcon( icon2 );
  }
};
button1.addActionListener( al );
//button1.removeActionListener( al ) ist möglich



button2.addActionListener( new ActionListener() {
  public void actionPerformed( ActionEvent e ) {
    System.exit( 0 );
  }
} );
```

Beispiel: Direkte Grafikausgabe implementieren

```
MyJPanel extends JPanel {
        // Konstruktor
        …
        @Override                                   // Überschreiben
        public void paintComponent(Graphics g) {
                super.paintComponent(g);  // Hintergrund übermalen
                                                    // Graphics2D Cast für größere
                                                    //  Auswahl an Zeichenmethoden
                Graphics2D g2 = (Graphics2D) g;
                g2.setStroke(new BasicStroke(3));
        …
```

Der Aufruf von repaint() forciert eine Aktualisierung des Bildschirminhaltes. Der Aufruf erfolgt auf der zu aktualisierenden Komponente. z.B. Instanz des JFrames oder nur eines JPanels.

Grafikprogrammierung: paint()

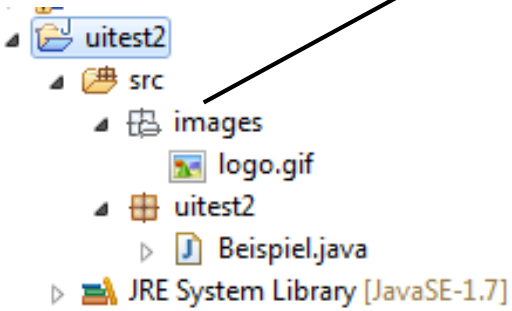
- paint() ruft auf paintComponent(), paintBorder() und paintChildren()
- wird aufgerufen, wenn die Komponente neu gezeichnet werden muss.
- repaint() von außen aufgerufen, erzwingt Neuzeichnen.
- Unterklassen von Swing-Komponenten überschreiben im Regelfall nicht paint(), sondern paintComponent().
- @Override paintComponent(Graphics g){} //automatically called
- Die Methode paintComponent() besitzt in der Oberklasse die Sichtbarkeit protected

Image/Icon mit getResource()

```
URL imgURL = this.getClass().getResource( "/images/logo.gif" );
Icon i = new ImageIcon(imgURL);
…
Image j = Toolkit.getDefaultToolkit().createImage(imgURL);
prepareImage(grafik, this);        // Grafik vorbereiten für die Darstellung

JButton b = new JButton();
b.setIcon(i);
```

Erzeugt durch:
New -> Folder
Bilder können per Drag&Drop kopiert werden



Hintergrundgrafik – Ähnlich „Direkte Grafikausgabe“

```
contentPane = new JBackgroundPanel();
setContentPane(contentPane);

public class JBackgroundPanel extends JPanel
{
        private BufferedImage img;
         public JBackgroundPanel() {
                  img = ImageIO.read(getClass().getClassLoader()
                   .getResourceAsStream("images/image.gif"));
                }
         }

         @Override protected void paintComponent(Graphics g) {
           super.paintComponent(g);
           g.drawImage(img, 0, 0, getWidth(), getHeight(), this);
         }
}
```

Layout verändern

- JFrame.setLayout(…)
- Default: BorderLayout nach Himmelsrichtungen, z.B. CENTER
- GridLayout: Gleichmässige Verteilung auf Zeilen und Spalten
- GridBagLayout: Elemente lassen sich durch Constraints auch über mehrere Zellen ausdehnen