

Laborpraktikum Software

Wintersemester 2019/20

A	0. Organisatorisches 1. Refresher Java / UML 2. Vererbung, Interfaces und Polymorphie	✓
B	3. Wert-/Verweis-Typen / -Parameter 4. Exceptions 5. Streams, Dateien, Kodierungen	✓
C	6. Objektgeflechte 7. Collections 8. Binärstreams, Serialisierung	✓
D	9. XML 10. WebRequest / -Response	✓
E	11. Events 12. Einführung in die gängigsten grafischen Dialogelemente 13. Ergonomie	

Teile der Folien basieren auf Material von Thomas Wilhelm, M. Schneider, Herhard Weber, M. Löberbauer und W. Goerigk
Sowie Dr. Ledgards University of Toledo

Wahlpflicht-Modul SMSB

- Im 4ten Fachsemester ist ein WPF zu belegen
- Angeboten werden
 - Block Chain (Dr. Pieper)
 - Kryptografische Protokolle (Prof. Noack)
 - KI (zusammen mit SMIB, Lehrbeauftragter)
 - Industrielle Kommunikationsprotokolle (Prof. Büchau)
 - Big Data (Hr. Plotz, Prof. Bunse)
 - Graphische Datenverarbeitung (Prof. Ehricke)
- Auswahl von 2 Favoriten unter:
 - https://ilias.hochschule-stralsund.de/ilias/goto.php?target=cat_74315&client_id=ecs-ilias

Quellen

- Wenn nicht gesondert angegeben:
 - Programmieren spielend gelernt mit dem Java-Hamster-Modell; D. Boles
 - „Java Developer“: <http://www.oracle.com/technetwork/java/index-jsp-135888.html>
- „gute“ Nachschlagewerke
 - Java ist auch eine Insel von Christian Ullenboom
 - Java: How to Program; Deitel & Deitel

Interaktionsmodell

- Jede Interaktion eines Nutzers mit einer Applikation löst sog. „Events“ aus
 - Tastatur (KeyEvent)
 - Maus – Bewegung oder Klick (MouseEvent)
 - ...
- Jedes Java-Objekt kann über auftretende Events informiert werden. Dazu muss es
 - Das entsprechende Interface implementieren, und
 - Als Listener bei der entsprechenden Quelle angemeldet werden
 - Vgl. Konzept der Exception

Beispiele für GUI-Events

- Klick auf Button oder Enter -> ActionListener
- Größe eines Fensters ändern -> ChangeListener
- Maus-Button -> MouseListener
- Maus-Bewegung -> MouseMotionListener
- ...

Interaktionsmodell

- Wichtige Elemente
 - Event source - GUI mit der interagiert wird
 - Event listener – Object das Events empfängt und Aktionen auslöst
- Ihre Aufgabe
 - Registrieren des Listener an der Quelle
 - Implementierung eines Event Handler
- Nutzen von
 - Klassen
 - Anonymen Klassen
 - Lambda Ausdrücken

Grundsätzliche Realisierung

```
new EventHandler() {  
    @Override //  
    public void handle(ActionEvent event) {  
        System.out.print("Hello World !"); } } }
```

- Nutzen einer anonymen inneren Klasse zur Implementierung des Event-Handler

Eventhandling für GUI-Elemente

```
...
textField.addEventHandler(KeyEvent.KEY_PRESSED,
    new EventHandler() {
        public void handle(KeyEvent keyEvent) {

            System.out.println("Taste getdrückt " +
                               keyEvent.getCode());
            keyEvent.consume();
        }
    });
...
```


Einschub: Innere Klassen

- Innere Klassen
 - werden innerhalb einer anderen Klasse definiert
 - lassen sich in der definierenden Klasse verbergen
 - haben Zugriff auf Daten und Methoden der definierenden Klasse
 - können „anonym“, d.h. ad hoc und ohne Klassennamen, definiert werden
- Es gibt vier Typen von inneren Klassen
 - Element Klassen
 - Geschachtelte Klassen
 - Lokale
 - **Anonyme Klassen**

Einschub: Innere Klassen II

- Anonyme Klassen
 - Lokale Klasse ohne Namen
 - werden mit einem Operator definiert, der das Objekt anlegt
 - Klasse für ein einziges Objekt
 - Anwendung:
 - für nur ein Objekt benötigt
 - für kleinen Adapter-Code
 - insbesondere für die Realisierung von Listenern beim Ereigniskonzept

Einschub: Innere Klassen III

```
@Override
public void start(Stage primaryStage) {
    rect.setFill(Color.BLUE);

    rect.setOnMouseClicked(new EventHandler<MouseEvent>()
    {
        @Override
        public void handle(MouseEvent t) {
            rect.setFill(Color.RED);
        }
    });

    StackPane root = new StackPane();
    root.getChildren().add(rect);

    Scene scene = new Scene(root, 300, 250);

    primaryStage.setTitle("Hello World!");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

Einschub: Innere Klassen III

- Adapter-Klassen
 - Klassen, die die Verwendung von Interfaces vereinfachen
 - Erlauben Zugriff auf gerade benötigten Methoden

```
public abstract class PerAdapter implements Persistency
{
    @Override
    void save(Zettelkasten zk, String dateiname) {}
    @Override
    Zettelkasten load(String dateiname) {}
}

public static void main(String[] args) {
    new PerAdapter() {
        public void save(Zettelkasten zk, String dateiname) {
            System.out.println("Mach was mit dem Adapter");
        }
    }.save(ZetKa, „Test.txt“);
}
```

GUI ENTWICKLUNG MIT JAVA FX

Graphische Benutzeroberflächen

- **“Graphical User Interface” (GUI) Applikationen basieren auf dem Prinzip der Objektorientierung (im Ggs. Zu Konsolen-Applikationen)**
- **JavaFX ist ein Framework zur Entwicklung von GUI Applikationen**
- **Die JavaFX API ist ein gutes Bsp. für die Anwendung von OO-Prinzipien in der Softwareentwicklung**

Allgemein

- Dieser Teil der Vorlesung bezieht sich auf die Erstellung von einfachen User-Interfaces.
- Zur Entwicklung von Verhalten (Interaktionen, etc.) sind Events und deren Bearbeitung notwendig

JavaFX vs Swing vs AWT

- Java nutzte ursprünglich das Abstract Windows Toolkit (AWT)
- AWT nutzt Elemente des jeweiligen Betriebssystems und ist dessen Regeln unterworfen
- Nachfolger ist Swing mit dem Ziel von robusten, plattformunabhängigen Applikationen

JavaFX vs Swing vs AWT

- Swing wurde für Desktopapplikationen (nicht Web) entwickelt
- Swing wurde inzwischen durch eine neue GUI Bibliothek (Java FX) ersetzt
- Swing wird nicht weiterentwickelt
- Java FX wird für die kommenden Jahre der Java GUI Standard sein
 - Oracle stellt den Support für JavaFX voraussichtlich 2022 ein
 - Weitere Entwicklung als separates Open-Source-Modul
 - Seit Java 11 getrennte Bibliothek

JavaFX vs Swing vs AWT

- JavaFX erlaubt die Entwicklung von RIAs (Rich Internet Applications) die auf dem Desktop und im Browser (IE / FireFox / Chrome / Safari) ausführbar sind
- Java FX \neq Applets

Java FX – Grundlegenden Struktur

- (Wdhlg.) Abstrakte Klassen dienen der Vererbung und werden niemals instantiiert.
- JavaFX Programme leiten sich von der abstrakten Klasse *javafx.application.Application* ab

Java FX – Grundlegenden Struktur

```
public class MyProgram  
{  
    // Implementierung  
}
```

Wird Zu

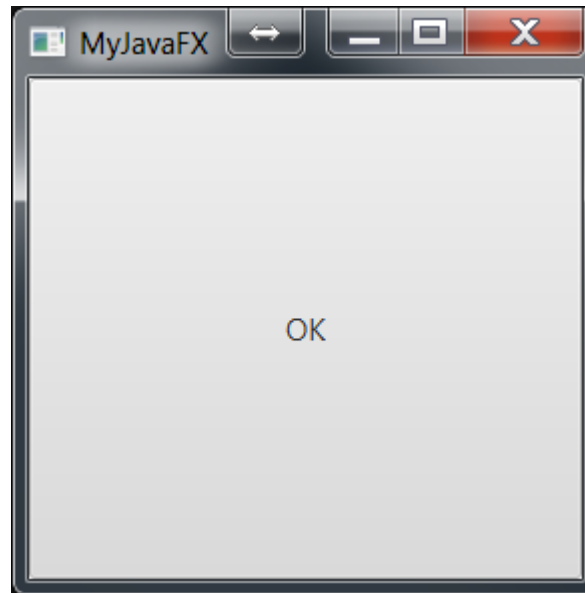
```
import javafx.application.Application;  
  
...  
public class MyProgram extends Application  
{  
    // Implementierung  
}
```

Java FX: IDE und Scenebuilder

- Installation
 - JavaFX
 - Ab JDK11 kein Teil des JDK.
 - Download Open-Source JavaFX SDK.
 - Scenebuilder – Separates Werkzeug
 - IDE
 - In IntelliJ bereits inkludiert
 - e(fx)clipse – Eclipse Plugin
 - Kommunikation via fXML-Datei
- Anleitung: ILIAS

Java FX – Die erste Applikation

- Unser erstes JavaFX Programm öffnet ein Fenster mit dem Titel “MyJavaFX”. Im Fenster existiert ein Button der mit “OK” beschrieben ist
- Achtung: Scenebuilder wird hier nicht verwendet, sondern nur der Java Code betrachtet



Java FX – Die erste Applikation

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;

public class MyJavaFX extends Application
{
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage)
    {
        // Create a scene and place a button in the scene

        Button btOK = new Button("OK"); // create a button
        Scene scene = new Scene(btOK, 200, 250); // create a scene WITH the button
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    /**
     * The main method is only needed for the IDE with limited
     * JavaFX support. Not needed for running from the command line.
     */
    public static void main(String[] args)
    {
        Application.launch(args);
    }
}
```

Java FX – Die erste Applikation

- JavaFX Programme nutzen die Analogie der “Bühne” (stage)
- Auf der Bühne werden Szenen (scenes) dargestellt die wiederum aus anderen Komponenten bestehen
- In JavaFX erstellen wir Komponenten, fügen diese zu Szenen hinzu und bringen diese auf die Bühne

Java FX – Die erste Applikation

- In JavaFX entspricht die Bühne dem Fenster unserer Applikation
- Applikationen können mehr als eine Bühne haben. Allerdings muss es eine Hauptbühne (primary stage) geben



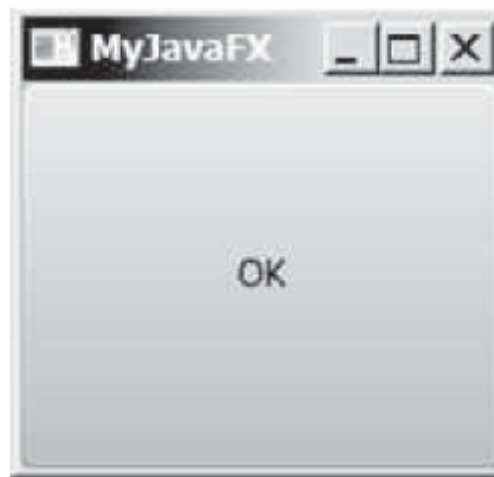
Java FX – Die erste Applikation

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class MultipleStageDemo extends Application
{
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage)
    {
        // Create a scene and place a button in the scene
        Scene scene = new Scene(new Button("OK"), 200, 250);
        primaryStage.setTitle("MyJavaFX"); // Set the stage title
        primaryStage.setScene(scene);      // Put the scene in the stage
        primaryStage.show();               // Display the stage

        Stage stage = new Stage();        // Create a new stage
        stage.setTitle("Second Stage");    // Set the stage title
        // Set a scene with a button in the stage
        stage.setScene(new Scene(new Button("New Stage"), 100, 100));
        stage.show();                     // Display the stage
    }
}
```

Java FX – Die erste Applikation

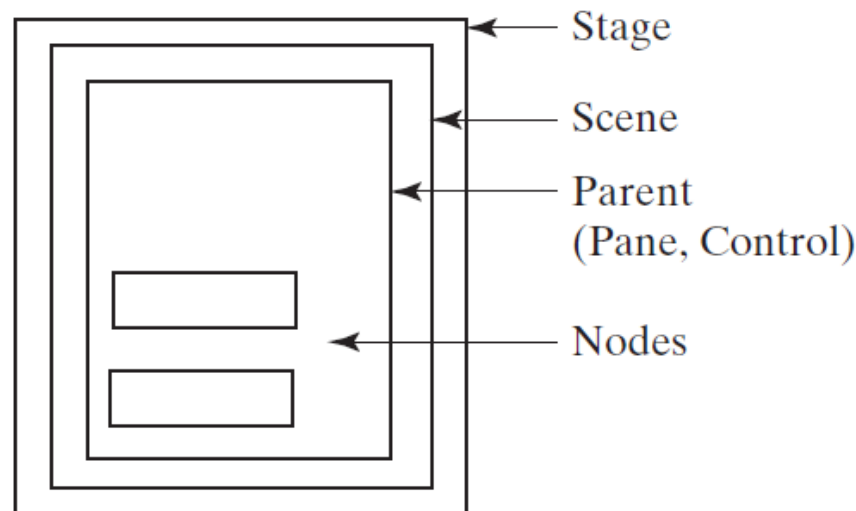
- Fenster sind in ihrer Größe veränderbar
 - Minimize und Maximize Buttons



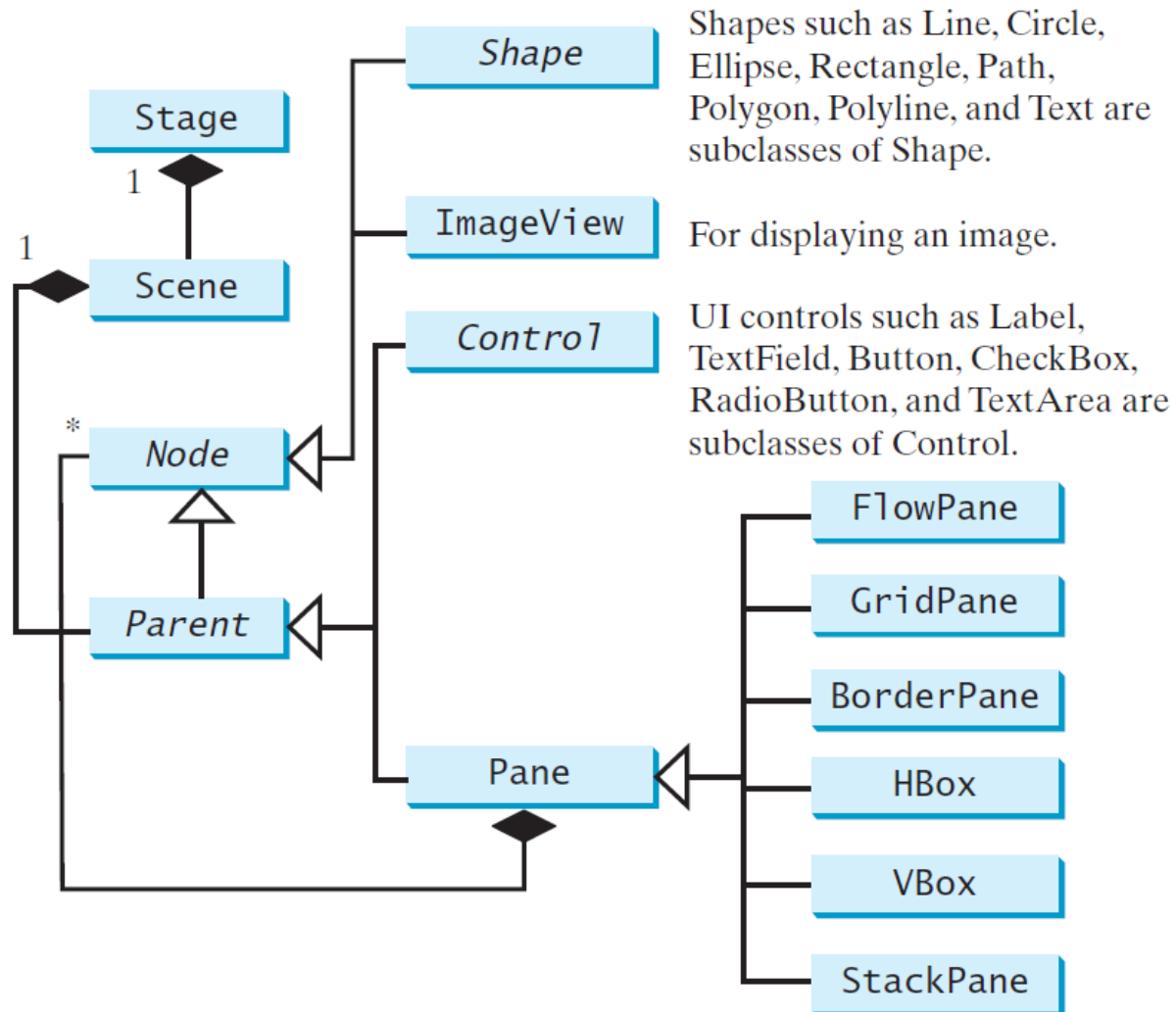
- Feste Größe via: `stage.setResizable(false)`

Panes, UI Controls, und Shapes

- Bisher: Der Button wurde direkt in der Szene platziert und beansprucht die maximale Größe
- Alternative: Spezifikation von Größe und Position
- Java FX: Elemente (nodes) werden in Container (panes) verpackt und zur Szene hinzugefügt



Panes, UI Controls, und Shapes



Quelle: University of Toledo

Panes, UI Controls, und Shapes

- Ziel: Erstellen unserer ersten Applikation unter Nutzung von „Panes“
- Hier: Einsatz einer sog. “StackPane”
- „Panes“ verhalten sich (Hinzufügen, Löschen, etc.) wie eine ArrayList
- Neue Elemente werden als “Kinder” der Liste der “Pane Children” hinzugefügt

Panes, UI Controls, und Shapes

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;

public class ButtonInPane extends Application
{
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage)
    {
        StackPane pane = new StackPane(); // Make a pane to work with

        // create a new button, and add it to the pane's list of children
        pane.getChildren().add(new Button("OK"));

        // Make a new scene, containing the pane
        Scene scene = new Scene(pane, 200, 50);
        primaryStage.setTitle("Button in a pane"); // Set the stage title
        primaryStage.setScene(scene); // Put scene in the stage
        primaryStage.show(); // Display the stage
    }
}
```

Layout Panes

- Bisher: Nodes werden zu einer „Pane“ hinzugefügt, diese zu einer „Scene“ und diese wiederum zu einer „Stage“
- Jetzt: Anordnung der Nodes (Layout) in einer „Pane“
- Java bietet hierzu eine Reihe verschiedener „Panes“ an die wir im folgenden betrachten werden

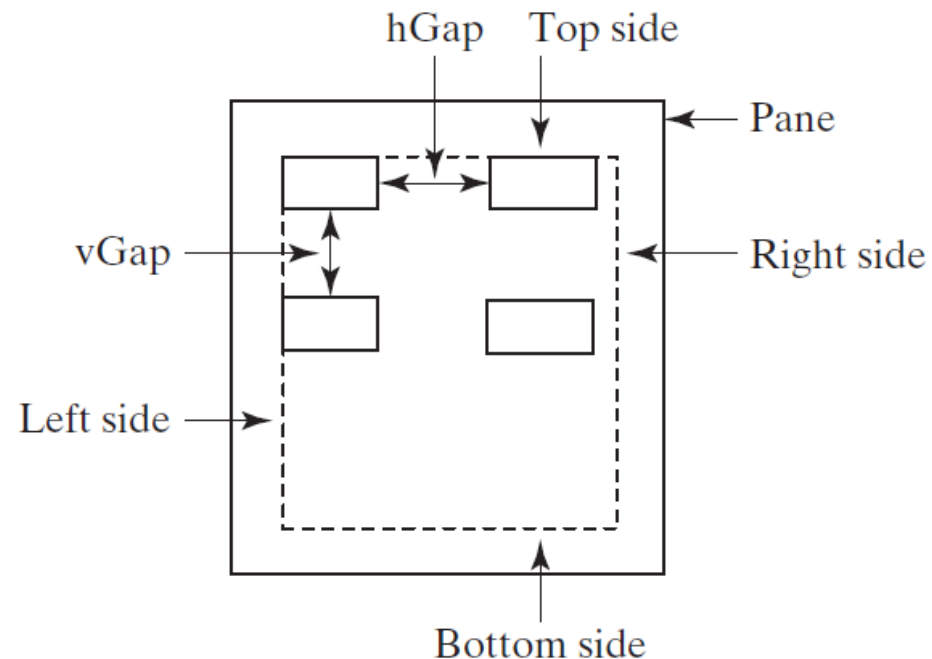
Layout Panes

Name	Description
Pane	Base class for layout panes. Use its <u>getChildren()</u> method to return the list of nodes on the pane (or add to that list) Provides no particular layout capabilities – it's a "blank canvas" typically used to draw shapes on
<u>StackPane</u>	Places nodes on top of each other in the center of the pane
<u>FlowPane</u>	Places the nodes row-by-row horizontally or column-by-column vertically (reading order)
<u>GridPane</u>	Provide a 2-D grid of cells, into which we can place nodes
<u>BorderPane</u>	Divides pane into top, bottom, left, right, and center regions
<u>HBox</u>	Places nodes in a single (horizontal) row
<u>VBox</u>	Places nodes in a single (vertical) column

Quelle: University of Toledo

Flow Pane

- FlowPane: Anordnung von Nodes in sequentieller Ordnung (Orientation.HORIZONTAL oder Orientation.VERTICAL)
- Abstand zwischen Nodes wird in Pixel definiert



Quelle: University of Toledo

Flow Pane

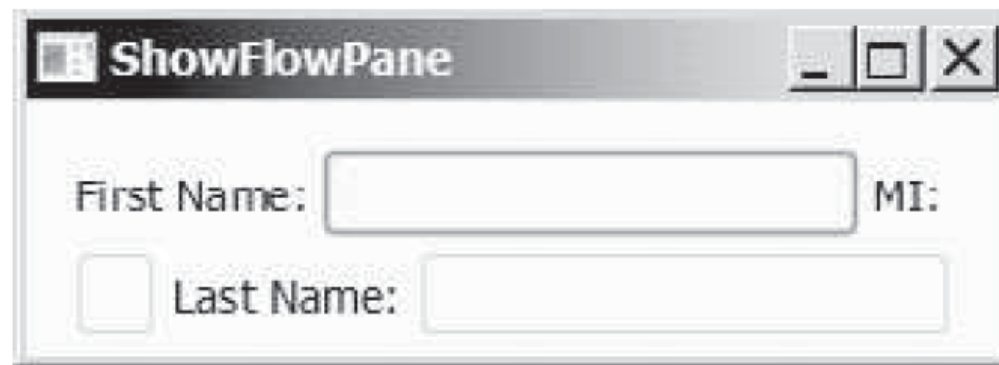
```
public void start(Stage primaryStage) // From Listing 14.10 (p. 553)
{
    // Create a pane and set its properties
    FlowPane pane = new FlowPane();
    pane.setPadding(new Insets(11, 12, 13, 14));
    pane.setHgap(5);
    pane.setVgap(5);

    // Place nodes in the pane
    pane.getChildren().addAll(new Label("First Name:"),
    new TextField(), new Label("MI:"));
    TextField tfMi = new TextField();
    tfMi.setPrefColumnCount(1);
    pane.getChildren().addAll(tfMi, new Label("Last Name:"),
    new TextField());

    // Create a scene and place it in the stage
    Scene scene = new Scene(pane, 200, 250);
    primaryStage.setTitle("ShowFlowPane"); // Set the stage title
    primaryStage.setScene(scene);         // Put scene in stage
    primaryStage.show();                   // Display the stage
}
```

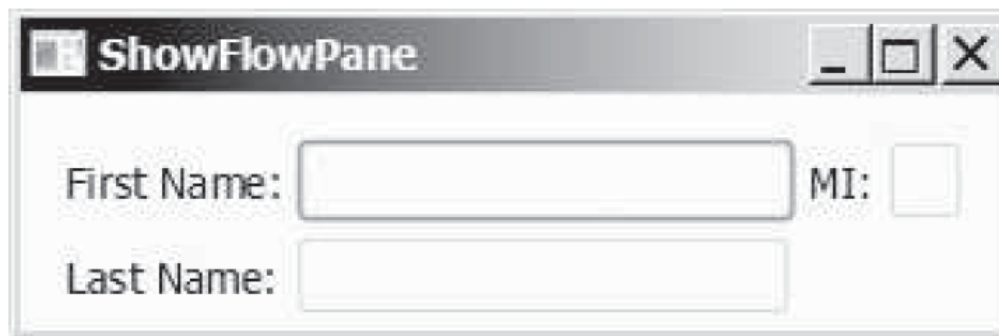
Flow Pane

- Problem: Größenänderung



First Name: MI:

☐ Last Name:



First Name: MI: ☐

Last Name:

Grid Pane

- Eine GridPane nutzt das Konzept einer Tabelle (Reihen und Spalten) zur Anordnung von “Panee”



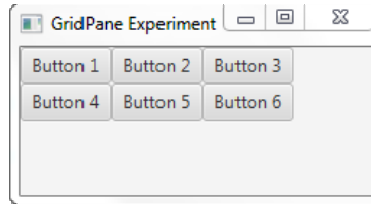
The screenshot shows a Java Swing window titled "ShowGridPane". Inside the window, there is a form with three text input fields arranged vertically. The labels "First Name:", "MI:", and "Last Name:" are positioned to the left of their respective input fields. Below the input fields, there is a button labeled "Add Name". The window has a standard title bar with minimize, maximize, and close buttons.



The screenshot shows a Java Swing window titled "ShowGridPane". Inside the window, there is a form with three text input fields arranged vertically. The labels "First Name:", "MI:", and "Last Name:" are positioned to the left of their respective input fields. Below the input fields, there is a button labeled "Add Name". The window has a standard title bar with minimize, maximize, and close buttons.

GridPane

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.stage.Stage;
```



```
public class GridPaneExperiments extends Application {
```

```
    @Override
```

```
    public void start(Stage primaryStage) throws
    Exception {
```

```
        primaryStage.setTitle("GridPane Experiment");
```

```
        Button button1 = new Button("Button 1");
```

```
        Button button2 = new Button("Button 2");
```

```
        Button button3 = new Button("Button 3");
```

```
        Button button4 = new Button("Button 4");
```

```
        Button button5 = new Button("Button 5");
```

```
        Button button6 = new Button("Button 6");
```

```
        GridPane gridPane = new GridPane();
```

```
        gridPane.add(button1, 0, 0, 1, 1);
```

```
        gridPane.add(button2, 1, 0, 1, 1);
```

```
        gridPane.add(button3, 2, 0, 1, 1);
```

```
        gridPane.add(button4, 0, 1, 1, 1);
```

```
        gridPane.add(button5, 1, 1, 1, 1);
```

```
        gridPane.add(button6, 2, 1, 1, 1);
```

```
        Scene scene = new Scene(gridPane,
240, 100);
```

```
        primaryStage.setScene(scene);
```

```
        primaryStage.show();
```

```
    }
```

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

```
        Application.launch(args);
```

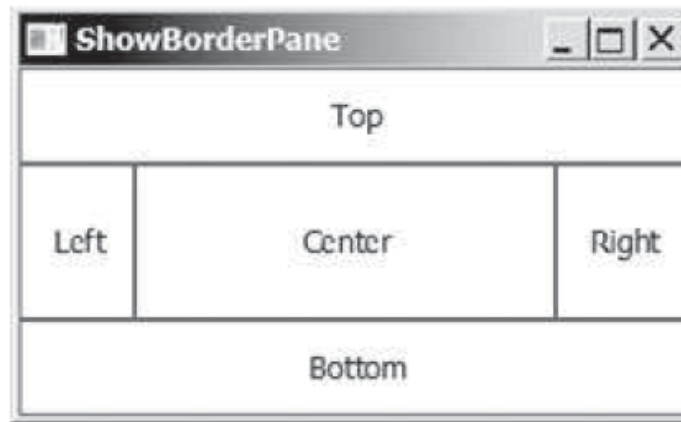
```
    }
```

```
}
```

Quelle <http://tutorials.jenkov.com>

Border Pane

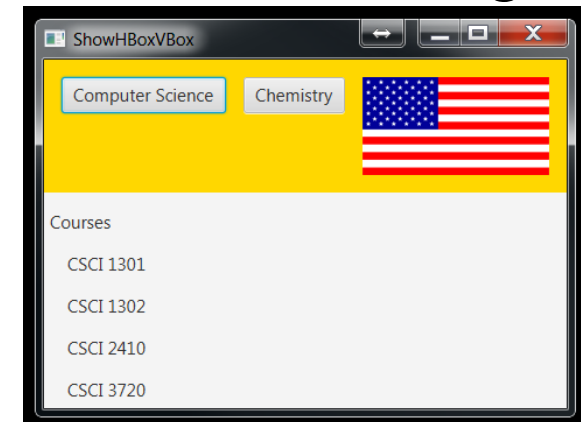
- Eine „BorderPane“ unterteilt eine „Pane“ in fünf Regionen



- Beachte: Eine „Pane“ ist auch eine „Node“ und kann daher weitere „Panee“ enthalten
- Eine leere Region wird nicht angezeigt
- “Löschen” einer Region mittels `set<region>(null)`

Hbox und VBox

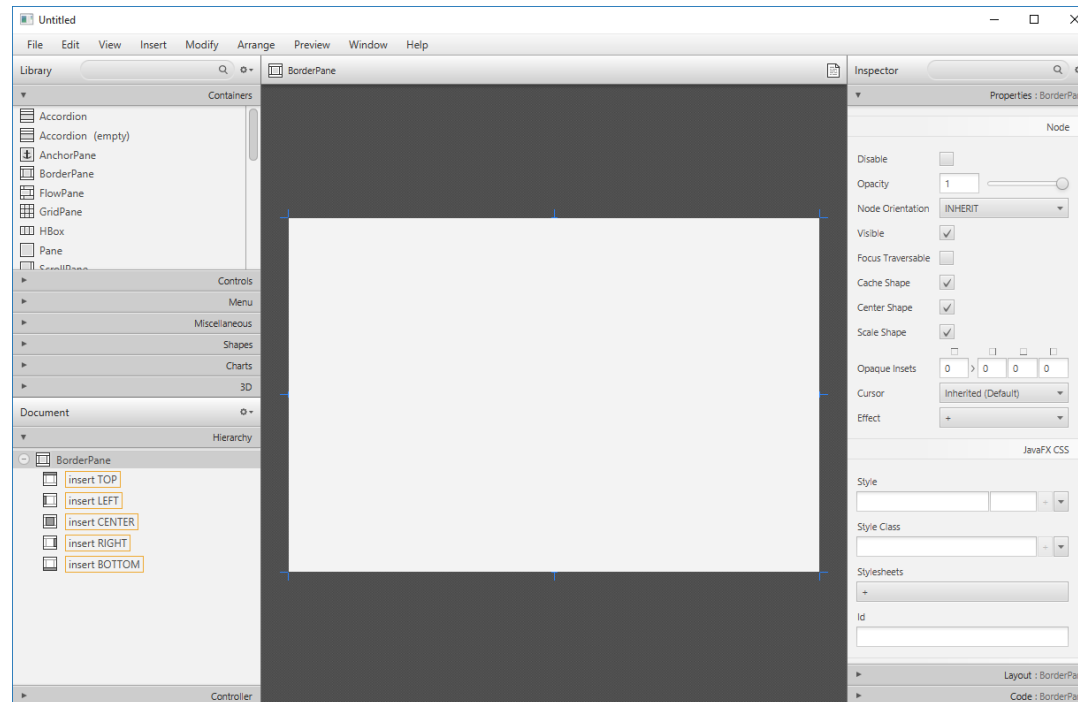
- Eine Flow Pane nutzt die sequentielle Darstellung von Elementen
- HBox und VBox panes nutzen zur Darstellung eine singuläre Reihe oder Spalte



- Das Beispiel nutzt: BorderPane, Hbox und VBox
- BorderPane: Top und BottomRegion
- Top: Hbox mit zwei Buttons und ImageView
- Bottom: VBox mit 5 Labels

Scene Builder

- Wer das manuelle Programmieren der Oberflächen nicht mag, nimmt einen interaktiven GUI-Builder wie den Scene Builder:



Weitere Elemente

- Common Properties
- Fonts
- Shapes
- Images
- Siehe <https://jaxenter.de/java-tutorial-javafx-53878>

Ereignisbehandlung

Grundlagen

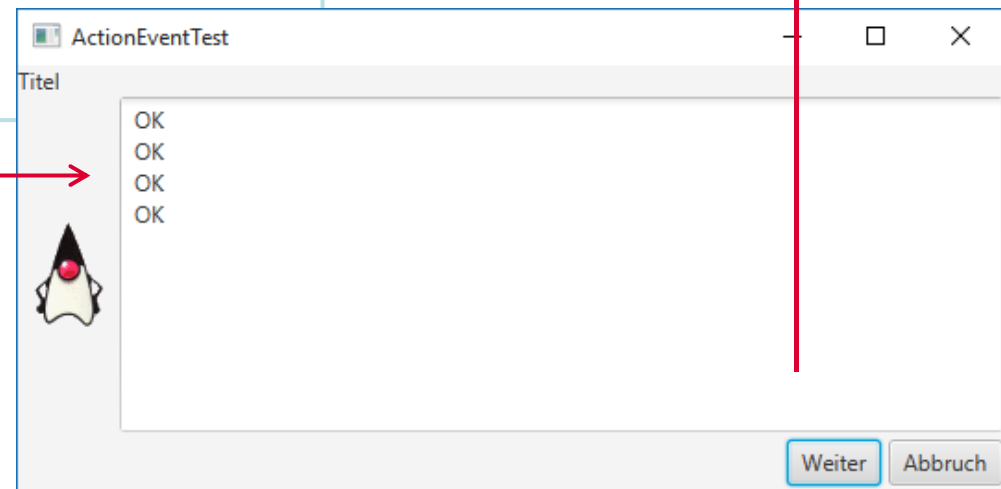
- In JavaFX werden unterschiedliche Arten von Ereignissen unterstützt.
- Als wichtiges Ereignis soll hier das **ActionEvent** näher betrachtet werden.
- Buttons lösen ein Action-Event aus, nachdem der Button gedrückt und wieder losgelassen wurde.
- Die Ereignisbehandlung basiert auf einem modifizierten Beobachter-Entwurfsmuster (Listener = Beobachter)

Ereignisbehandlung

Grundlagen

- Beispiel:

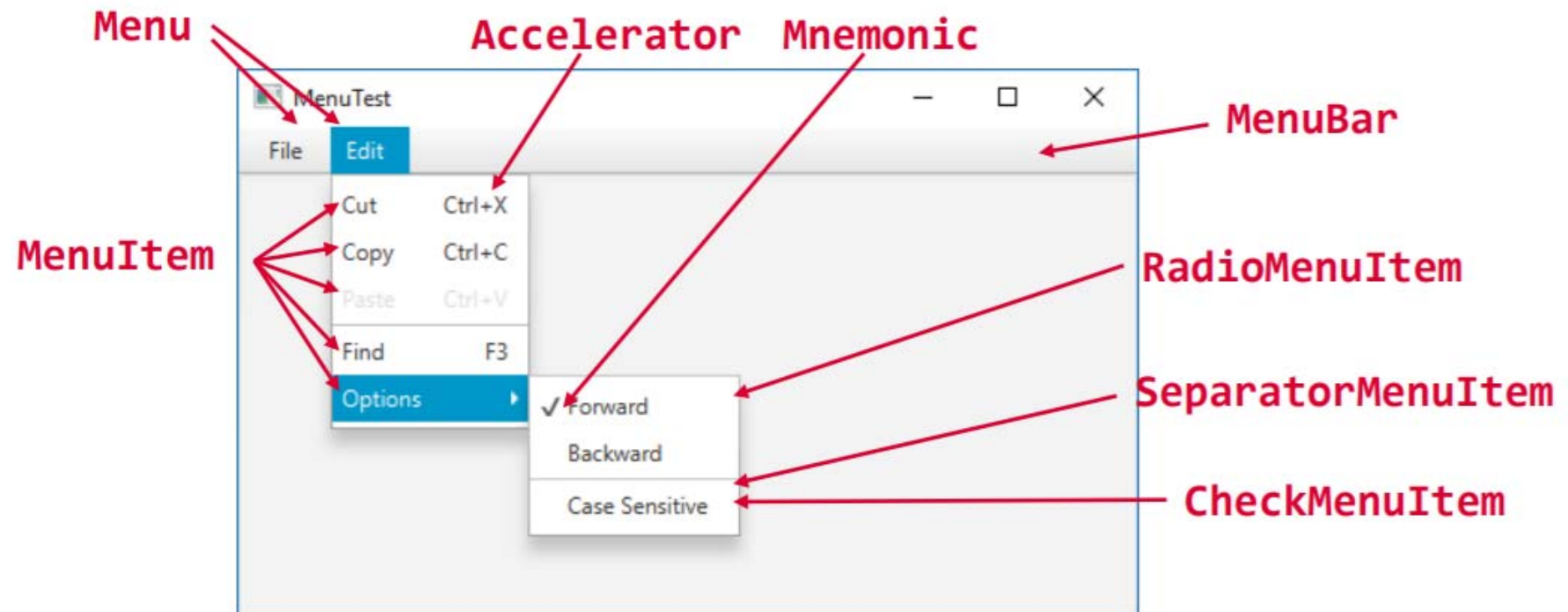
```
...  
final TextArea centerText = new TextArea("");  
borderPane.setCenter(centerText);  
  
...  
Button okButton = new Button("Weiter");  
okButton.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent e) {  
        centerText.appendText("OK\n");  
    }  
});  
...
```



Quelle: Holger Vogelsang holger.vogelsang@hs-karlsruhe.de

Menüstrukturen

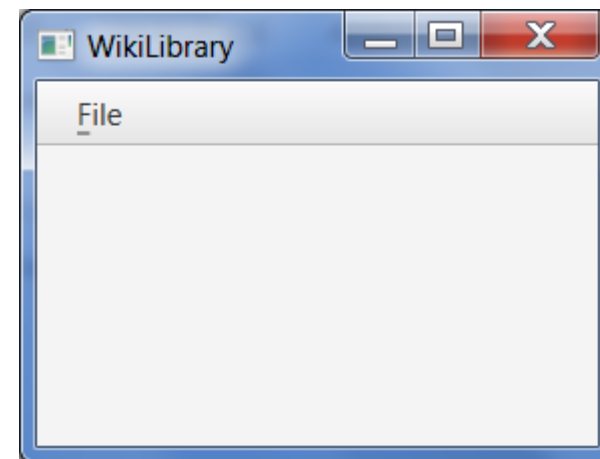
Aufbau einer Menüstruktur



Quelle: Holger Vogelsang holger.vogelsang@hs-karlsruhe.de

Menüstrukturen

```
VBox mroot = new VBox();  
MenuBar menuBar = new MenuBar();  
  
Menu fileMenu = new Menu("_File");  
MenuItem newNotebookMenuItem = new MenuItem("New Notebook...");  
newNotebookMenuItem.setAccelerator(KeyCombination.keyCombination("Meta+N"));  
newNotebookMenuItem.setOnAction(event -> { System.out.println("Action fired"); });  
  
fileMenu.getItems().add(newNotebookMenuItem);  
menuBar.getMenus().add(fileMenu);  
  
mroot.getChildren().add(menuBar);  
Scene scene = new Scene(mroot, 1270, 890);
```



Menüstrukturen

Accelerator

- Einstellen mittels *KeyCombination*
- Registration über das *MenuItem*
- `newNotebookMenuItem.setAccelerator(KeyCombination.keyCombination("Meta+N"));
fileMenu.getItems().add(newNotebookMenuItem);`
- Parameter
 - Schlüssel: A, B, C, ...
 - Modifier: Ctrl, Meta, Shift, Alt

Mnemonic

- Einschalten (Menu oder MenuItem)
 - `fileMenu.setMnemonicParsing(true);`

Einschub: Lambda Notation

- Bisher
 - `Button.setOnAction(new EventHandler<ActionEvent>() { ... })`
- Neu
 - Nutzung der Java (ab Java 8) Lambda-Notation zur Code-Reduktion

Einschub: Lambda Notation

```
button.setOnAction((ActionEvent event)-> {  
    ToggleButton source = (ToggleButton) event.getSource();  
    if (source.isSelected()) {  
        btnText.set("Clicked!");  
    } else {  
        btnText.set("Click!");  
    }  
});
```

- Verkürzt (Dank Compiler Optimierung)
 - `button.setOnAction(event -> { ... });`

Einschub: Lambda Notation

Lambda Ausdruck

```
Button btn = new Button();  
btn.setOnAction((event)->  
    System.out.println("Action");  
});
```

Parameter

Lambda Funktionsblock

- Lambda-Ausdrücke sind anonym (Namenslos)
- Parameter müssen nicht typisiert werden
- Typen werden vom Compiler hergeleitet

Beispiel

- Im folgenden Beispiel wird eine **Integer-Liste absteigend sortiert** mit Hilfe eines **Comparator-Objekts**.
- Das Comparator-Objekt wird neu erzeugt und implementiert das Interface Comparator und ist damit Instanz einer anonymen, inneren Klasse.

```
List<Integer> intList = Arrays.asList(5, 2, 7, 8, 9, 1, 4, 3, 6, 10);  
intList.sort( new Comparator<Integer>(){  
    public int compare(Integer x, Integer y) {  
        return y.compareTo(x);  
    }  
});
```


- Es muss ein **großer syntaktischer Aufwand** betrieben werden, um das Sortierverfahren mit der gewünschten Vergleichsmethode zu parameterisieren.
- Beachte: seit Java 8 bietet das Interface List<E> auch eine Sortiermethode (stabiles Sortierverfahren) an:

void sort(Comparator<? **super** E> c)

Beispiel

- Mit einem **Lambda-Ausdruck** geht es **prägnanter**.

```
List<Integer> intList = Arrays.asList(5, 2, 7, 8, 9, 1, 4, 3, 6, 10);  
intList.sort( (x,y) -> y.compareTo(x) );
```



Lambda-Ausdruck

- Beachte: hier hat der Lambda-Ausdruck zwei Parameter x, y. Beide Parameter müssen nicht typisiert werden. Der Parametertyp wird vom Java-Compiler hergeleitet.

Lambda Ausdrücke

- Lambda-Ausdrücke haben die allgemeine Bauart:

(Parameterliste) -> Funktionsblock

- Die Parameterliste kann leer sein.
- Hat die Parameterliste nur einen (nicht typisierten) Parameter, dann kann die Klammer entfallen.
- Die Parameter können typisiert werden (in manchen Situationen ist das auch erforderlich). Die Klammer muss dann geschrieben werden.

```
( ) -> System.out.println("Hallo");
```

```
(x) -> x+1
```

```
x -> x+1
```

```
(x, y) -> x+y
```

```
(String s) -> s + "!"
```

```
(int x, int y) -> x + y
```

Lambda Ausdrücke

- Der Funktionsblock bei Lambda-Termen folgt den gleichen Regeln wie bei Methoden.
- Wird ein Rückgabewert erwartet, dann muss ein return erfolgen (Kurzschreibweise möglich: siehe unten). Erfolgt kein Rückgabewert, dann kann return entfallen.

```
(int n) -> {  
    int p = 1;  
    for (int i = 1; i <= n; i++)  
        p *= i;  
    return p;  
}
```

```
(int n) -> {  
    for (int i = 1; i <= n; i++)  
        System.out.println();  
}
```

- Besteht der Funktionsblock nur aus einer return-Anweisung oder einen Funktionsaufruf, dann gibt es folgende Kurzschreibweise:

```
(int n) -> n + 1
```

```
() -> System.out.println("Hallo")
```

statt

```
(int n) -> {  
    return n + 1;  
}
```

```
() -> {  
    System.out.println("Hallo");  
}
```

ERGONOMIE

Definition - HCI

“Human-Computer Interaction (HCI) is a discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them.”
[ACM, SIGCHI]

HCI untersucht

1. Kontext von Computern
2. Fähigkeiten des Menschen
3. Entwicklungsprozess
4. Architektur der Schnittstellen

Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Kontext von Computern

- Arbeitsplatz
- Restaurant
- Haus
- Point of Sales
- Öffentliche Plätze



Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Fähigkeiten von Menschen

- Menschliche Informationsverarbeitung
 - visual overload
 - motorische Fertigkeiten
 - Kinder, behinderte und ältere Benutzer
- Kommunikation
 - sozial
- Ergonomie
 - individuelles Vermögen und Grenzen

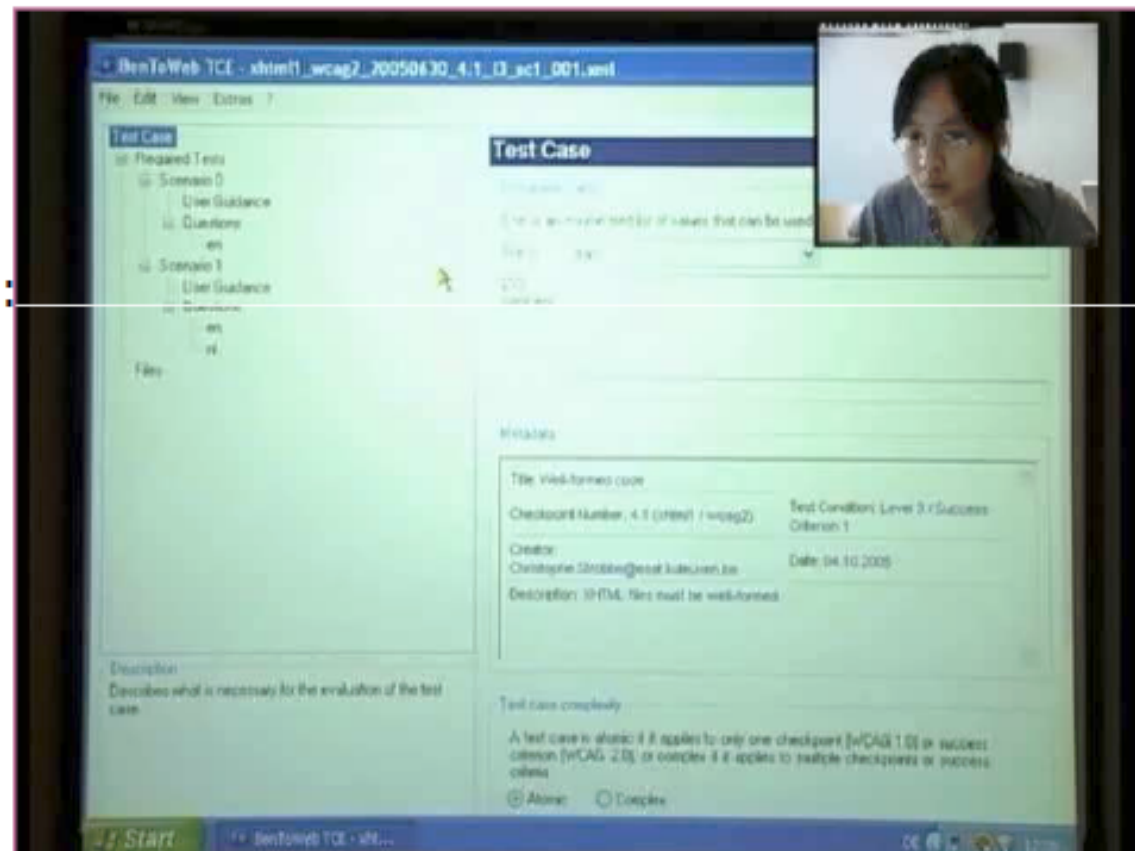
Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Evaluation

analysieren der
Problem der
Benutzer mit
Software

Kriterien

- Sicht der Users:
ISO 9241 (11)
- Expertensicht:
ISO 9241 (10)



Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

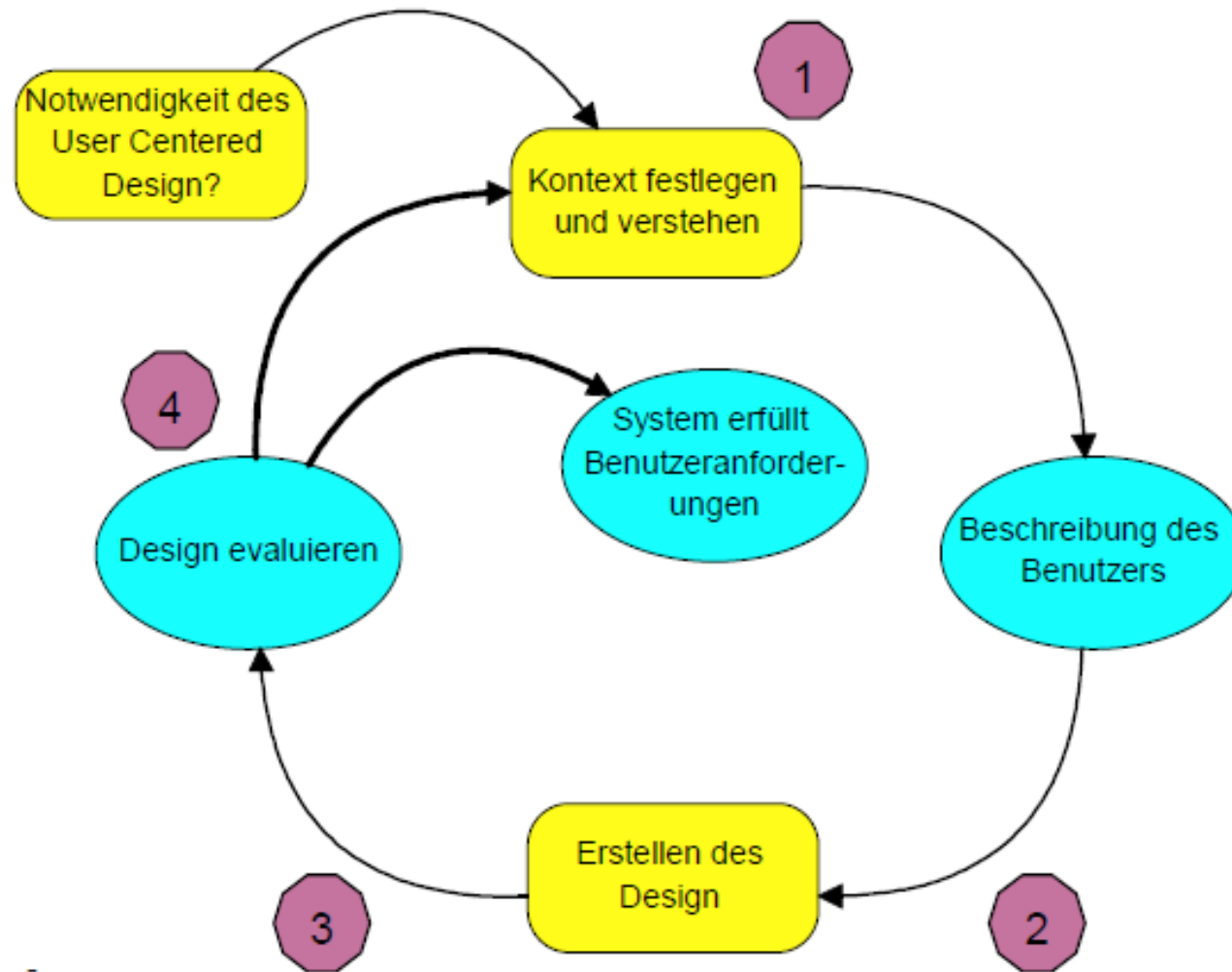
Evaluation

- Graphische Benutzungsoberfläche (GUI)
 - Maus-basierte Interaktion
 - Desktop Computer
 - Web-based
- Multimediale UI
 - Games (3D)
 - Broadcasting (iTV)
 - Remote control
 - Mobile
 - voice and pen
 - Ubiquitous
 - location based
- *Multimodal user interface*



Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Prozess



Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Prozess

Ziel: Bestellung im Restaurant

Aktionen:

1. Tisch festlegen
2. Getränke
3. Speisen
 1. Vorspeisen
 2. Hauptgang
4. Nachtisch
5. Bezahlvorgang
 1. Rechnungslegung
 1. Gemeinsame Rechnung
 2. Einzelrechnungen
 2. Kassiervorgang
 1. Bargeld
 2. Kreditkarte
 3. Trinkgeld festlegen

Menü?
Formular?
Wizard?
Funktionstasten?

Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Kriterien – ISO 9241

1. Aufgabenangemessenheit
2. Selbstbeschreibungsfähigkeit
3. Steuerbarkeit
4. Erwartungskonsistenz
5. Fehlerrobustheit
6. Lernförderlichkeit
7. Individualisierbarkeit

Quelle: http://st.inf.tu-dresden.de/files/teaching/ws11/swp/GUI_Tutorial.pdf

Aufgabenangemessenheit

- Primäraufgabe (Arbeitsaufgaben) vor Sekundäraufgabe (Bedienung): People aren' t trying to use computers - they are trying to get their jobs done. (Apple Computer)
- Nützlichkeit: Sinnvolle und ausreichende Funktionalität
- Komfort:
 - Keine unnötige Belastung mit systemspezifischen Eingaben
 - sinnvolle Defaultwerte
 - Cursorpositionierung im richtige Feld
 - ...
- Ziel: effiziente Aufgabenbewältigung

Quelle: <http://www.mmk.ei.tum.de/lehre/ebof/vlss12/vorl03.pdf>

Selbstbeschreibungsfähigkeit

- Jederzeit muß klar sein:
 - Wo bin ich?
 - Wie kam ich hierher, wohin kann ich noch gehen?
 - Was kann ich hier tun?
- Intuitiv verständliche Begriffe und Icons
- Sprechen Sie die Sprache der Benutzer
- Geben Sie ausreichende Informationen
 - Statuszeile, Hinweise
- Kontextsensitive Hilfe
- Abbildung der Realität, Metaphern

Quelle: <http://www.mmk.ei.tum.de/lehre/ebof/vlss12/vorl03.pdf>

Erwartungskonformität

- Prinzip der geringsten Verwunderung
- Konsistenz
 - konsistente Begriffe, Vorgehensweisen, Prinzipien und konsistentes Layout
 - innerhalb einer Anwendung und anwendungsübergreifend
 - Einhaltung von Systemstandards
- Feedback
 - jederzeit im Bilde über den aktuellen Systemzustand
 - Bestätigung von Benutzereingaben

Quelle: <http://www.mmk.ei.tum.de/lehre/ebof/vlss12/vorl03.pdf>

Steuerbarkeit

- Benutzer wollen das System im Griff haben, nicht umgekehrt
- Kontrolle und Initiative beim Benutzer
- Abbruchmöglichkeit an jedem Punkt
- Shortcuts für geübte Benutzer

Quelle: <http://www.mmk.ei.tum.de/lehre/ebof/vlss12/vorl03.pdf>

Fehlertoleranz

- Erster Schritt: Fehlervermeidung /Prävention
- Nicht alle Fehler sind vermeidbar --> Fehlermanagement
- Gute Fehlermeldungen sind
 - sichtbar
 - informativ
 - verständlich
 - nicht beleidigend
- Gute Fehlermeldungen helfen das Problem zu lösen
 - Ansatzpunkte, Strategien, mögliche Ursachen
- Aus Fehlern sollte man lernen können

Quelle: <http://www.mmk.ei.tum.de/lehre/ebof/vlss12/vorl03.pdf>

Evaluation

J. Nielsen: 10 Designregeln der heuristischen Evaluation

- Sichtbarkeit und Transparenz des Systemzustands
- Konformität mit der Alltagswelt
- Steuerbarkeit und Freiheitsgrade
- Konsistenz und Standards
- Fehlervermeidung
- Wiedererkennen statt Erinnern
- Flexibilität und Effizienz
- Ästhetisches und schlankes Design
- Fehlererkennung, -diagnose und -behebung unterstützen
- Online-Hilfe und Dokumentation

Quelle: <http://www.mmk.ei.tum.de/lehre/ebof/vlss12/vorl03.pdf>

Rückblick

Wintersemester 2019/20

A	<ul style="list-style-type: none"> 0. Organisatorisches 1. Refresher Java / UML 2. Vererbung, Interfaces und Polymorphie 	
B	<ul style="list-style-type: none"> 3. Wert-/Verweis-Typen / -Parameter 4. Exceptions 5. Streams, Dateien, Kodierungen 	
C	<ul style="list-style-type: none"> 6. Objektgeflechte 7. Collections 8. Binärstreams, Serialisierung 	
D	<ul style="list-style-type: none"> 9. XML 10. WebRequest / -Response 	
E	<ul style="list-style-type: none"> 11. Events 12. Einführung in die gängigsten grafischen Dialogelemente 13. Ergonomie 	