DHBW
Duale Hochschule
Baden-Württemberg

AIRBUS
DEFENCE & SPACE

Java GUI Grundlagen

Lukas Abelt
lukas.abelt@airbus.com

DHBW Ravensburg
Wirtschaftsinformatik

Ravensburg
15. Mai 2019

Inhalt

Inhalt

Java GUI Anwendungen

Die Historie (Vgl. [10])

- Anfangs bestanden Computerprogramme in der Regel aus Kommandozeilenanweisungen
- Ab den 1970er Jahren wurden grafische Anwendungen immer wichtiger
- Gründe hierfür unter anderem:
  - 1973 - Release des Alto personal PC's von Xerox
  - 1984 - Erster Macintosh
  - 1985 - Release des Amiga mit GUI Oberfläche
  - 1985 - Release von Windows 1.0
  - Kurz: Der „Personal Computer“ fand immer mehr Verbreitung
- Dadurch Nutzerverschiebung
  - Von „Experten“ zu „unwissenden“ Nutzern

Historie

Nachteile von Konsolenanwendungen

- Kommandozeilenanwendungen sind für „Normalnutzer“ nicht sinnvoll
- Begrenzte Interaktionsmöglichkeiten $\rightarrow$ Nur Texteingabe
- Bedienung nicht intuitiv $\rightarrow$ Kann nicht ohne spezielles Wissen bedient werden
- Sieht nicht ansprechend aus
- GUIs lösen diese Probleme (theoretisch)
  - Der Nutzer hat eine Vielzahl an Interaktionsmöglichkeiten
  - Gut designete GUI benötigt keine Anleitung, der Nutzen lässt sich ableiten
  - GUIs können ansprechend designed werden (zB. über Nutzung von Bildern und Icons)

GUI Anwendungen

Die technischen Implikationen

- Jedes Betriebssystem implementiert ggf. verschiedene GUI-Komponenten und -Logik
- GUI musste also für jedes OS neu implementiert werden
- Dadurch Nachteile:
  - Massiver Mehraufwand für Cross-Platform-Development
  - Kein einheitliches Aussehen auf verschiedenen Plattformen
  - Nicht jedes OS unterstützt jedes GUI Feature $\rightarrow$ Dadurch wieder eingeschränkte Funktionalität

Java to the rescue!

Eine GUI sie zu knechten(?)

- Java wollte eine plattformunabhängige GUI schaffen
- Ergebnis war 1995 das **Abstract Window Toolkit (AWT)**
- Durch die Kopplung zum JRE waren diese Plattformunabhängig
- Intern nutzt AWT jedoch direkt Betriebssystemkomponenten
  - Dadurch sind AWT Komponenten sehr effizient (Geringe Abstraktion)
  - Ist aber auch Grund für die Probleme von AWT

AWT

Der Fall

- Probleme an AWT:
  - Durch enge Betriebssystembindung musste der gemeinsame Nenner aller Betriebssysteme gefunden werden
  - Dadurch auf wenige Komponenten beschränkt
  - Begrenzter Umfang (zB Darstellen von Icons auf Komponenten nicht möglich)
  - AWT war im Grunde „quick and dirty“ zusammengepfuscht um „erstmal zu funktionieren“ (Siehe [9] S. 778)
  - Alternative Namen:
    - *Awkward Window Toolkit*
    - *Annoying Window Toolkit*

AWT

Meinungen (Siehe [8] S. 778)

Quelle: [1]

„The AWT was something we put together in six weeks to run on as many platforms as we could, and its goal was really just to work. So we came out with this very simple, lowest-common-denominator thing that actually worked quite well. But we knew at the time we were doing it that it was really limited. After that was out, we started doing the Swing thing, and that involved working with Netscape and IBM and folks from all over the place.“

—James Gosling, „Vater“ von Java

Komponenten in AWT

Auflistung (Vgl. [8] S. 1070)

- Das AWT besteht aus nur acht Komponenten:
  - Button
  - Checkbox
  - Choice
  - Label
  - List
  - Scrollbar
  - TextArea
  - TextField
- Man nennt diese auch die *Peer-Klassen*

Die Welt nach AWT

Die Geburt von Swing

- AWT wurde schnell erneuert
- Initial begann NetScape mit einer eigenen AWT Erweiterung: Die *Internet Foundation Classes (IFC)*
- Später arbeiten Sun (Heute Oracle), NetScape und IBM zusammen
- 1998 entstanden daraus die *Java Foundation Classes (JFC)*
- Kernelement waren hierbei die neuen *Swing* Komponenten

Grundsätzliche Bestandteile I

Der JFC

- **GUI-Komponenten**
  - Die *Swing* Komponenten bringen ein neues, versatiles, Set an neuen Funktionen. Diese sind, anders als die alten AWT Komponenten, komplett durch Java implementiert und verwaltet.
- **Pluggable Look&Feel**
  - Die Komponenten können zur Laufzeit im Aussehen verändert werden ohne, dass ein Neustart der Anwendung nötig ist. Alle *Swing*-Elemente bringen diese Fähigkeit mit
- **Java 2D API**

Grundsätzliche Bestandteile II

Der JFC

- Die neue 2D Grafik-API bildet automatisch über Objektbeschreibungen Objekte, die auf dem Bildschirm dargestellt werden. Komplexe Objekte können über PFade gebildet werden und darauf Bewegungs- und Verschiebeoperationen ausgeführt werden

- **Drag&Drop**
  - Daten können mittels Drag and Drop zwischen verschiedenen Applikationen übertragen werden. So können Java-Programme auch Daten nutzen, die aus Nicht-Java Anwendungen stammen

- **Accessibility**
  - Die neue API erlaubt es, mehr Interaktionstechniken für körperlich eingeschränkte Nutzer anzubieten. Dies sind zum Beispiel die Vorlesefunktion, eine Spracherkennung oder eine Bildschirmlupe

Swing

Noch etwas trivia

- Swing bietet schlussendlich viel von dem was AWT sein sollte
- Einheitliches Look&Feel auf allen Plattformen
- Hohe konfigurierbarkeit
- Ausgereifter und erweiterbarer Funktionsumfang
- Einziges Problem: Swing wurde mit der Zeit nicht weiterentwickelt
- Dadurch fehlen immer mehr Features von modernen GUIs

JavaFX

Der neue heiße Scheiß!

- Moderne Features vermisst man in Swing, wie:
  - Animationen
  - Medienunterstützung
- Daher begann die Entwicklung von JavaFX
- Dies sollte, anders als bei dem Wechsel von AWT zu Swing, komplett losgelöst vom bisherigen GUI Stack sein
- Bildet einen kompletten modernen Media-Stack ab
- Kann für 3D Anwendungen direkt auf Grafikkartenfunktionen zugreifen

JavaFX

Historie im Überblick

- **2007** - Release von JavaFX 1.0
- **2008** - JavaFX 2.0, entfernen von Java Script und Entscheidung zur puren Java-API
- **2012** - JavaFX 2.2 wird Teil der Standard JRE/JDK (Version 7 Update 6)
- **2018** - Java 11 wird released, jetzt wieder ohne integriertes JavaFX (JavaFX 11)
- **Heute** - JavaFX wird unabhängig von Java weiterentwickelt und ist Open-Source als *OpenJFX*

Java GUI

Die Zukunft

- Ob JFX sich durchsetzen wird bleibt fraglich
- Anwendungsfälle entwickeln und verändern sich rasant
- Neue Peripherie hat ganz andere Anforderungen:
  - Smartphones und -Watches
  - Virtual Reality
  - Mixed Reality (Bspw.: Hololens)
- Meine (persönliche) Meinung: Auch OpenJFX wird langfristig untergehen

Inhalt

Swing

Grundlegendes

- AWT Komponenten erzeugen Komponenten über Betriebssystemaufrufe
- Dadurch Speicher nicht über Java verwaltet
- Swing erzeugt Komponenten über direkte Low-Level Calls
- Jede Komponente wird auf den Bildschirm gezeichnet
- Dadurch komplette Speicherverwaltung in Java
- Overhead führt ggf. zu Performanceverlust
- Aber mehr Kontrolle über die Features

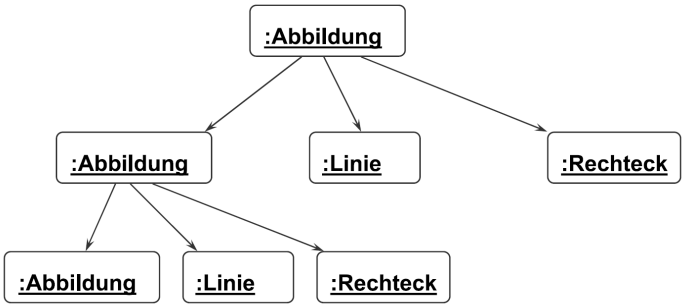
Swing

Konzepte

- Swing nutzt das Entwurfstmuster des *Kompositums*
- Kompositum beschreibt im Grunde eine Art Baumstruktur
- Objekte werden zu Gruppen zusammengefasst und Gruppen wiederum zu größeren Gruppen
- Dadurch kann eine einheitliche Behandlung von Objekten und Aggregaten erreicht werden
- Gemeinsame Eigenschaften von Objekt und Aggregat werden in einer Oberklasse isoliert

Beispiel für Kompositum

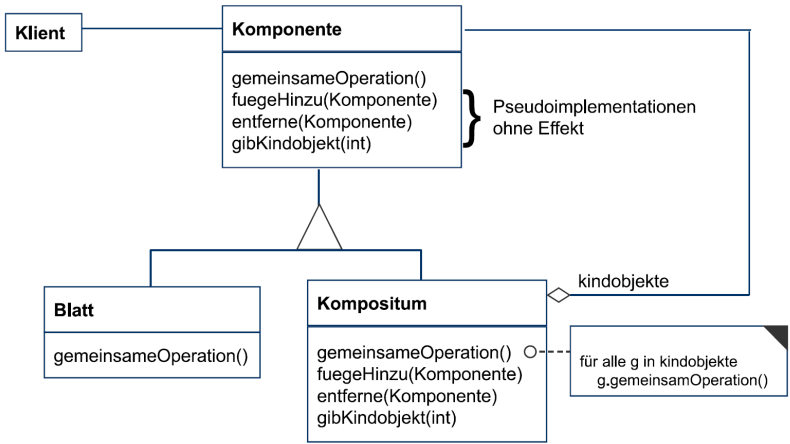
Zusammengesetzte Grapfik-Objekte (Vgl. [2] S. 26)



Gemeinsame Operationen: `zeichne()`, `verschiebe()`, `lösche()`, `skaliere()`

Kompositum

Beispielhaftes Klassendiagramm



Quelle: [2]

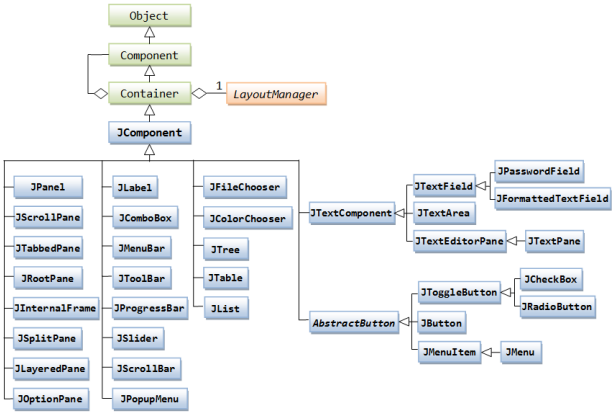
Kompositum

Umsetzung in Swing

- Die Klassenhierarchie stellt ein Kompositum dar
- Swing Objekte unterteilen sich hierbei in zwei Klassen:
  - `Component`
  - `Container`
- `Container` fassen `Components` zusammen
- Jeder Container ist für die Anordnung seiner Komponenten zuständig
- Jeder `Container` ist auch immer eine `Component`

Klassendiagramm

Der Swing Bibliothek



Swing Fenster

JFrame Klasse

- Basis eines Swing Fensters ist die `JFrame` Klasse
- Basiert auf der AWT Klasse `Frame`
- Lässt sich manipulieren z.B. in bezug auf
  - Sichtbarkeit
  - Größe und Position
  - Operation beim schließen
- Leeres Fenster wird erzeugt über den `JFrame()` Konstruktor

Swing Fenster

Ganz simpel

```
1 public static void main(String[] args){
2   JFrame window = new JFrame();
3   window.setDefaultCloseOperation(WindowConstants.
        ↪ EXIT_ON_CLOSE);
4   window.setVisible(true);
5 }
```

JFrame

Ändern der Größe des Fensters

- Größe wird über `setSize()` verändert
- Hier gibt es zwei Überladungen:
  - Als Parameter ein `Dimension` Objekt
  - Zwei `int` Werte als Parameter

Swing Fenster

Weitere Eigenschaften

- Bildet einen *Top-Level-Container*
- `JFrame` wird immer mit Titel und Menüleiste erzeugt
- Erscheinung des Frames kann geändert werden
  - Man spricht in der Regel von „Decorations“
  - Ändern der Form des Fensters (In begrenztem Rahmen)
  - Wählen verschiedener Farbschema
  - (Teil-)transparente Fenster
  - usw

Weitere Fenster in Swing

- `JWindow` – Fenster ohne Menüleiste
- `JDialog` – Zum erstellen von (modalen) Dialogfeldern

Inhalt

Komponenten

Grundlegendes

- Komponenten in Swing sind frei erweiterbar
- Swing Komponenten i.D.R. über Namensgebung von AWT Komponenten unterscheidbar
- Für viele Anwendungsfälle sind jedoch die Standardkomponenten schon ausreichen
- Neue Komponenten werden über
- add() Methode zu einem beliebigen Container hinzugefügt
- Komponenten können eine bevorzugte oder minimale Größe definieren
- Position wird i.d.R. automatisch bestimmt (Über den Layoutmanager)

Grundlegende Komponenten I

Übersicht

- `JPanel` – Einfacher Bereich ohne spezielle Besonderheiten. Ist ein `Container` und wird in der Regel genutzt um Elemente zu gruppieren und ggf. anzuordnen
- `JButton` – Einfacher Button für den eine Funktion programmiert werden kann(Zum Beispiel das Starten einer Berechung)
- `JCheckBox` – Auswahlbox für eine binäre Eingabe
- `JLabel` – Simples Feld zum Anzeigen von Texten. Kann nicht durch den Nutzer bearbeitet werden
- `JTextField` – Textfeld zum Anzeigen oder Bearbeiten von Texten (Einzelzeilen). Kann durch den Nutzer verändert werden
- `JSlider` – Schieberegler

Grundlegende Komponenten II

Übersicht

- `JRadioButton` – Gibt die Möglichkeit zur Auswahl von einer Option aus mehreren gegebenen Möglichkeiten. Es kann pro Gruppe an Optionen immer nur genau eine gleichzeitig aktiviert sein. Um RadioButtons in Gruppen zusammenzufassen, ist die Nutzung eines `ButtonGroup` Objects notwendig
- `JComboBox` – Element, das die Auswahl eines Eintrags aus einer Liste erlaubt oder auch einen eigenen Eintrag ermöglicht
- `JEditorPane` – Ermöglicht die Eingabe von mehrzeiligen Texten

Inhalt

GUI Layout

Anordnen von Elementen

- Bei Nutzen von `add()` werden Komponenten automatisch platziert
- Dies geschieht durch den *Layout Manager*
- Manager werden pro Container definiert
- Je nach Layout sind ggf. feste „Slots“ im Container vorhanden
- Um Komponente an bestimmter Position zu platzieren wird Überladung von `add()` verwendet:
  - `add(Component comp, Object constraint)`
  - `constraint` definiert Position im Container

GUI Layout

Größe von Komponenten

- Komponenten können absolute, minimale, maximale und bevorzugte Größe definieren
  - `setSize(int height, int width)`
  - `setMinimumSize(Dimension d)`
  - `setMaximumSize(Dimension d)`
  - `setPreferredSize(Dimension d)`
- Layout Manager *versucht* diese zu beachten
- *Müssen* dies jedoch nicht
- Insbesondere absolute Größen sind meist gegen das Prinzip von Layout Managern

GUI Layout

Layout Manager

- Vorteil von Layout Managern:
  - Relative Positionierung von Komponenten kann definiert werden
  - Vereinfachen erstellen von GUIs
  - Größe und Position von Komponenten werden automatisch angepasst (Zum Beispiel bei Ändern der Fenstergröße)

BorderLayout

Grundlegendes (Vgl. [3])

- Teil den Container in fünf Bereiche
- Diese sind über Konstanten in der `BorderLayout` Klasse definiert:
  - `PAGE_START` – Oberer Bereich (Kopfzeile)
  - `PAGE_END` – Unterer Bereich (Fußzeile)
  - `LINE_START` – Linker Bereich zwischen Kopf und Fuß
  - `LINE_END` – Rechter Bereich zwischen Kopf und Fuß
  - `CENTER` – Mittlerer (Haupt-)bereich
- Beispiel: Hinzufügen im `PAGE_START` Bereich:
  - `add(button, BorderLayout.PAGE_START)`

BorderLayout

Weitere Eigenschaften (Vgl. [3])

- Fokus liegt auf den `CENTER` Bereich
- Dieser nutzt maximal verfügbaren Platz
- Alle anderen Bereiche nutzen nur den benötigten Platz
- Nicht alle Bereiche müssen genutzt werden
- `PAGE_START` und `PAGE_END` nutzen (sofern vorhanden) immer die volle Breite
- Freiräume zwischen Bereichen lassen sich definieren:
  - `setHGap(int)`
  - `setVGap(int)`
- Standard-Layoutmanager von `JFrame`

BorderLayout

Aufgabe

BorderLayout Demo

Implementiert eine simple Fensteranwendung, um das Verhalten des `BorderLayout` kennen zu lernen. Platziert dazu zunächst jeweils einen `JButton` in jedem Bereich.
Untersucht, wie sich die Größe und Anordnung der Buttons ändert, wenn ihr eine Größe für die Buttons definiert oder wenn ihr einzelne Bereiche im Layout ungenutzt lasst.

BoxLayout

Grundlegendes(Vgl. [4])

- Ordnet Komponenten entweder vertikal oder horizontal angeordnet
- Dafür zwei Konstanten in der `BoxLayout` Klasse
  - `PAGE_AXIS` – Vertikale Anordnung (Als eine „Spalte“)
  - `LINE_AXIS` – Horizontale Anordnung (Als eine „Zeile“)
- Komponenten werden dann gemäß `ComponentAlignment` platziert (Bspw. Linksbündig/Zentriert/Rechtsbündig)
- Definierte maximale Größe von Komponenten wird im `BoxLayout` *immer* respektiert

BoxLayout

Aufgabe

Entwerft eine Beispielandwendung, in der mehrere Buttons in einem Frame mit dem `BoxLayout` als Layout Manager. Nutzt folgende Basis um den Layout Manager im `JFrame()` anzupassen:

```
1 JFrame frame = new JFrame();
2 frame.setSize(300,600);
3 frame.setLayout(new BoxLayout(window, BoxLayout.PAGE_AXIS))
    ↪ ;
4
5 /*Buttons hinzufügen*/
6
7 frame.setVisible(true);
```

Experimentiert, was passiert, wenn ihr die Orientierung der einzelnen Komponenten anpasst.

FlowLayout

(Vgl. [5])

- Ist das Standard-Layout von `JPanel`
- Komponenten werden in einer Reihe nebeneinander platziert
- Hierbei wird die bevorzugte Größe der Komponenten berücksichtigt
- Ist in einer Reihe nicht genug Platz verbleibend für die neue Komponente, so wird eine neue Reihe eingefügt
- Positionierung der Reihen kann definiert werden:
  - `BoxLayout.LEFT` – Linksbündig
  - `BoxLayout.RIGHT` – Rechtsbündig
  - `BoxLayout.CENTER` – Zentriert
  - `BoxLayout.LEADING` – Bündig mit dem Beginn der Container-Orientierung (zB. Links bei Links-nach-Rechts Orientierung)
  - `BoxLayout.TRAILING` – Bündig mit dem Ende der Container-Orientierung (zB. Links bei Rechts-nach-Links Orientierung)

FlowLayout

Aufgabe

Entwerft eine simple Fensteranwendung, die über das Hinzufügen mehrerer Buttons mit verschiedener Größe die Funktionsweise des `FlowLayouts` demonstriert. Ändert hierzu entweder den LayoutManager des erzeugten `JFrame` Objekts(Siehe Aufgaben zum BoxLayout) oder fügt ein `JPanel` hinzu, dem ihr anschließend die Buttons hinzufügt.
Experimentiert mit dem Verhalten des FlowLayouts bei ändern der Fenstergröße.

GridLayout

Grundlegendes (Vgl. [7])

- Rastert den Container in Zeilen und Spalten
- Jede „Zelle“ ist gleichgroß
- Komponenten in einer Zelle nehmen die gesamte Zelle ein
- Komponenten werden automatisch in die „nächste“ Zelle hinzugefügt
- Initialisierung:
  - `GridLayout()` – Initialisiert den Container mit einem 1x1 Grid
  - `GridLayout(int rows, int cols)` – Initialisiert das Grid mit der definierten Anzahl an Zeilen und Spalten
- Zeilen *oder* Spalten können auf 0 gesetzt werden → Heißt, dass die Anzahl der Zeilen/Spalten dynamisch aus Anzahl der Komponenten ermittelt wird

GridLayout

Aufgabe

Entwerft eine Simple Fensteranwendung in der ihr mehrere Komponenten mit dem GridLayout anordnet. Um den Layout Manager zu ändern geht vor wie im Beispielcode zur BoxLayout Aufgabe.
Untersucht, wie sich Anordnung der Komponenten verhält, wenn ihr die Anzahl der Zeilen oder Spalten mit 0 definiert.

GridBagLayout

Grundlegendes (Vgl. [6])

- Ähnlich zum `GridLayout`
- Jedoch flexibler in der Positionierung
- Dadurch aber auch komplexer in der Anwendung
- Komponenten können in beliebigen Zellen platziert werden
- Komponenten können mehrere Zellen einnehmen
- Relative Positionierung von Komponenten in den Zellen möglich

GridBagLayout

Visualisiert

[6]

GridBagLayout

Constraints (Vgl. [6])

- Platzierung der Komponenten im GridBag wird über `GridBagConstraints` Objekt definiert
- Diese müssen beim hinzufügen von Komponenten mit übergeben werden:
  - `add(Component,GridBagConstraints)`
- Für die Constraints lassen sich definieren:
- `gridx, gridy` – Zeile und Spalte der hinzuzufügenden Komponente. Kann auch als relativ zur zuletzt hinzugefügten Komponente definiert werden
- `gridwidth, gridheight` – definiert, wie viele Zeilen bzw. Spalten die Komponente einnehmen soll

GridBagLayout

Constraints (Vgl. [6])

- `fill` – Definiert inwiefern die Komponente den verfügbaren Platz nutzen soll. Kann genutzt werden, um Komponenten horizontal bzw. vertikal auf die Größe der Zelle zu skalieren
- `ipadx, ipady, insets` – Kann genutzt werden um padding der Komponente zum Rand zu definieren
- `anchor` – Definiert die Referenzposition in der Zelle (z.B. links unten, rechts oben etc.)
- `weightx, weighty` – Wird genutzt um zu bestimmen, wie der gesamt verfügbare Platz auf die einzelnen Zeilen/Spalten verteilt wird. Insbesondere wichtig, wenn die Fenstergröße verändert wird.

GridBagLayout

Aufgabe

Experimentiert (wenn ihr wollt) mit dem GridBagLayout und den entsprechenden Constraints. Zur Orientierung könnt die Oracle Tutorial Seite nutzen(Siehe [6])

Layout Manager

Kombinieren von Layout Managern

- Komplexe GUIs erfordern nicht zwingend einen komplexen LayoutManager
- Da jeder Container Layout Manager selbst definiert können diese kombiniert werden
- Heißt: Ein Container wird einem übergeordneten Layout hinzugefügt, verwendet aber für seine eigenen Komponenten einen anderen Manager
- Als Basis-Container wird meist ein `JPanel` verwendet

Layout Manager

Aufgabe

Entwerft eine simple GUI Anwendung mit einem BorderLayout. Platziert beliebige Komponenten in den äußeren Bereichen. Platziert im Center Bereich ein `JPanel`, in dem Komponenten im `BoxLayout` angeordnet sind.

GUI-Builder

- Alternativ zum „ausprogrammieren“ der GUI können auch GUI-Builder verwendet werden
- Hier werden die Komponenten in einer grafischen Oberfläche platziert
- Struktur wird dann meist in XML Format (o.Ä.) übersetzt
- Spezieller Layout Manager übernimmt Übersetzung in Code
- In IntelliJ integriert
- Für Eclipse als Plug-In verfügbar

Inhalt

Eigene Fenster

Erstellen von Fenstern

- Unsere bisherigen Fenster haben wir uns in der `main` zusammengebastelt
- Dies ist in der Regel unpraktikabel
- Code ist nicht wiederverwendbar
- Und ggf. schwierig wartbar
- Angestrebt wird eine Trennung von Darstellung und Logik und Daten
- Daher: Kapselung von GUI in eigene Klasse
  - Diese erbt von `JFrame`
  - Fenster wird im Konstruktor „zusammengebaut“
  - Hinzufügen aller Komponenten, Layouten von diesen, setzen von Attributen wie Größe etc.
  - In der Main Klasse wird dann lediglich ein Objekt von unserer neuen `JFrame` Unterklasse erzeugt

Eigene Komponenten

Grundlegendes

- Erben vom `JComponent` Typ
- Lassen sich vielseitig konfigurieren z.B. in:
  - Aussehen
  - Verhalten
- Nützlich kann es zum Beispiel sein die `getXxxSize` Methode zu überschreiben, wenn eine Komponente immer eine bestimmte Größe haben soll
- Fokus heute: Zeichnen von eigenen Komponenten

Zeichnen von Komponenten

Schritte im `paint()` Prozess

- Jede Komponente besitzt eine `paint()` Methode
- In dieser wird das Aussehen der Komponente auf den Bildschirm gezeichnet
- `paint()` ruft drei Methoden (in dieser Reihenfolge) auf:
  - `paintComponent(Graphics g)`
  - `paintBorder(Graphics g)`
  - `paintChildren(Graphics g)`
- In der Regel reicht es aus `paintComponent()` zu überschreiben
- Wir rufen `paint()` bzw. `paintComponent()` **nie** direkt auf

Zeichnen von Komponenten

Die `paint()` Methode

- `paint()` Wird durch das System automatisch aufgerufen bei zB.:
  - Verschieben des Fensters/Von Komponenten
  - Ändern der Fenstergröße
  - Minimieren/Maximieren
- Wenn programmatisch neu gezeichnet werden muss (Zum beispiel weil sich Daten geändert haben) wird die `repaint()` Methode aufgerufen
- `repaint()` akzeptiert Argumente, sodass nur ein bestimmter Bereich neu gezeichnet wird

Das `Graphics` Objekt

Zeichnen von Komponenten

- Den `paintXXX()` Methoden wird ein Objekt vom Typ *Graphic* übergeben
- Dieses fasst verschiedene Aspekte zum Zeichnen zusammen:
  - Aktuelle Komponente auf die gezeichnet wird
  - Die Aktuelle Farbe
  - Die aktuelle Font
  - Weitere Aspekte die für die aktuelle Betrachtung nicht von Relevanz sind
- Die Graphics Klasse bietet einige Funktionen zum Zeichnen von primitiven Formen an

Das `Graphics` Objekt

Zeichenoperationen

- `drawString` – Zeichnet einen gegebenen String an der angegebenen Position (Mehrere Überladungen)
- `drawImage` – Zeichnet ein Bild an einer bestimmten Stelle (Mehrere Überladungen)
- `drawArc` – Zeichnet einen Ellipsenbogen
- `drawLine` – Zeichnet eine Linie zwischen zwei Punkten
- `drawOval` – Zeichnet ein Oval
- `drawRect` – Zeichnet ein Rechteck
- `drawRoundRect` – Zeichnet ein abgerundetes Rechteck

Zeichnen von Komponenten

Weitere nützliche Operationen

- Über `getWidth()` bzw. `getHeight()` kann die aktuelle Größe der Komponenten bestimmt werden
- Für viele der `drawXXX` Methoden existieren analoge `fillXXX` Methoden
- Aktuelle Zeichenfarbe lässt sich ändern über `setColor`
- Analog lässt sich `setFont` verwenden um die Schriftart zu ändern

Hinweise

Beim Entwerfen von View-Komponenten

- Die Komponenten sollten nur für die Darstellung verantwortlich sein
- In ihnen werden in der Regel keine Applikationsdaten gespeichert
- Auch die Logik ist in der Regel entkoppelt von der Darstellung
- Man spricht hierbei auch vom *Model-View-Controller (MVC)* Konzept:
  - Model – Tatsächlich verwendetes Datenmodell. Speichert die Applikationsdaten (Datenschicht)
  - View – Anzeigen/rendern von Daten. Dient lediglich zur Darstellung (Präsentationsschicht)
  - Controller – Realisiert die Steuerung in Interaktion des Nutzers. Bildet in der Regel die Kopplung zwischen Model und View (Interaktionsschicht)

Inhalt

Aufgabe 1

Simple Komponente

Entwickelt eine simple Komponente als Unterklasse von `JComponent`. Überschreibt die `paintComponent` Methode und experimentiert mit den verschiedenen Methoden des `Grpahics` Objektes

Aufgabe 2

Sinuskurve

Entwickelt eine Komponente, die eine Sinuskurve zeichnet. Die Sinuskurve sollte bei vergrößern/verkleinern des Fensters entsprechend weiter gezeichnet werden. Die x-Achse sollte in der Mitte der Komponenten liegen und die maximale Amplitude noch im Bereich der Komponenten liegen.
Hinweis: Um den Sinus einer bestimmten Zahl zu bestimmen könnt ihr die Math.sin(double) Funktion verwenden. Beachtet allerdings, dass diese einen Wert im Intervall $[-1; 1]$ zurück gibt, den ihr ggf. für die Größe skalieren müsst

Quellen I

[1] Wikimedia Commons. *File:James Gosling 2008.jpg — Wikimedia Commons, the free media repository*. 2015. URL: https://commons.wikimedia.org/w/index.php?title=File:James_Gosling_2008.jpg&oldid=149207971 (besucht am 05. 02. 2019).

[2] Prof. Dr. Andreas Judt. *Software Engineering 2. Entwurfsmuster*. 2016.

[3] Oracle. *How to Use BorderLayout*. 2017. URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/border.html (besucht am 07. 05. 2019).

Quellen II

[4] Oracle. *How to Use BoxLayout*. 2017. URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/box.html (besucht am 07. 05. 2019).

[5] Oracle. *How to Use FlowLayout*. 2017. URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html (besucht am 07. 05. 2019).

[6] Oracle. *How to Use GridBagLayout*. 2017. URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html (besucht am 07. 05. 2019).

Quellen III

[7] Oracle. *How to Use GridLayout*. 2017. URL: https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html (besucht am 07. 05. 2019).

[8] C. Ullenboom. *Java ist auch eine Insel: Das umfassende Handbuch*. Rheinwerk Computing, 2018. ISBN: 978-3-8362-5869-2.

[9] C. Ullenboom. *Java SE 8 Standard-Bibliothek: das Handbuch für Java-Entwickler ; [Nebenläufigkeit, String-Verarbeitung, Datenstrukturen und Algorithmen, XML, RMI, JDBC, Reflection, JavaFX, Swing, Grafik- und Netzwerkprogrammierung ; JNI, Sicherheit]*. Galileo Computing. Galileo Press, 2014. ISBN: 9783836228749. URL: https://books.google.de/books?id=D3jSnQEACAAJ.

Quellen IV

[10] Wikipedia contributors. *History of the graphical user interface — Wikipedia, The Free Encyclopedia*. 2019. URL: https://en.wikipedia.org/w/index.php?title=History_of_the_graphical_user_interface&oldid=882638996 (besucht am 05. 02. 2019).

Kontakt

- E-Mail: `lukas.abelt@airbus.com`
- GitHub: `https://www.github.com/LuAbelt`
- GitLab: `https://www.gitlab.com/LuAbelt`
- Telefon(Firma): 07545 - 8 8895
- Telegram: LuAbelt