OOP2 01.04.16

**Applikation**

* Programmzyklus (life-cycle)

1. Erzeugen einer Instanz der spezifizierten Klasse
2. Aufruf der public void init(); Methode
3. Aufruf der Methode start(Stage);
4. Warten bis sich die Applikation beendet, g.d.w:
5. Aufruf von Platform.exit();
6. Das letzte Fenster wird geschlossen und das Attribut implicitExit der Platform enthält true
7. Aufruf der Methode stop();

* Übergabeparameter können mittels getParameter()-Methode in bzw. nach dem Aufruf von init() abgefragt werden.
* Nebenläufigkeit
* Erzeugen des Application-Thread (ausführen start-Methode, Ereignisverarbeitung, Animations-Zeitgeber)
* Erstellung von Scene und Stage-Objekten, sowie Modifikation von Szenegraphen muss hier erfolgen

Bemerkung: Scene und Stage-Objekte müssen im Application Thread erzeugt werden, da sie nur in dieser Umgebung existieren dürfen.

**Platform**

Exit() – Schließen der FX-Anwendung

Static boolean isFxApplicationThread() – liefert true, falls der aufrufende Thread der FxApplicationThread ist

Static bolean isImplicitExit() – liefert den Wert des implicitExit Attributes

Static Boolean isSupported(conditionalFeature f) – prüft, ob eine optionale Erweiterung von JavaFx in der Laufzeitumgebung verfügbar ist

Static void runLater(Runnable runnable) – übergibt ein ausführbares Objekt zur Ausführung durch den Application-Thread. Die Ausführungszeit ist unspezifiziert

Static void setImplicitExit(boolean implicitExit) – setzt den Wert des Attributes implicitExit

**Stage**

* Es können weitere Fenster in der Applikation erzeugt warden
* Stage-Objekte (Attribute) können durch die darunter liegende Platform – also extern - verändert werden => nur lesbar.
* Stage-Styles:
  + DECORATED: weißer Hintergrund mit System-Dekoration
  + UNDECORATED: weißer Hintergrund ohne Dekoration
  + TRANSPARENT: Kein Hintergrund und keine Dekoration
  + UTILITY: weißer Hintergrund und minimale Dekoration
* Optional: OwnerWindow(Parent)
  + Stage wird immer auf dem OwnerWindow angezeigt (im Window)
* Modalität (Blockierung der Ereigniskette):
  + NONE: keine Blockierung anderer Fenster
  + WINDOW\_MODAL: blockiert alle übergeordneten Fenster
  + APPLICATION\_MODAL: blockiert alle anderen Fenster der Anwendung
* Modale Dialoge können mittels
  + show() – nicht blockierend geöffnet werden
  + showAndWait() – blockierend geöffnet werden

08.04.16

**Attribute u. Methoden von Filechooser**

- Titel, initialer Ordner, Dateienamenserweiterung (Filter

- showOpenDialog() // Datei öffnen

- show SaveDialog() // Datei speichern

- showOpenMultiple // mehrere Dateien zum Öffnen auswählen

ShowOpen mit Owner = modal, mit null = eigenes Fenster

**Windows Events**

- ANY // stellvertretend für „alle“

WINDOW\_CLOSE\_REQUEST

WINDOW\_HIDDEN

WINDOW\_HIDING

WINDOW\_SHOWING

WINDOWS\_SHOWN

**Action Event**

- ANY ACTION

Subclass: MediaMarkerEvent

**Ereigniszustellung (Event Delivery Process)**

Die Ereigniszustellung durchläuft folgende Schritte:

1. Zielbestimmung
2. Wegbestimmung
3. Ereigniserfassung (Event capturing)
4. Event bubbling (sprudeln) = Ereignis läuft zurück, wo es herkam. -> Man weiß es wurde nicht verarbeitet

**Zielbestimmung**

Erfolgt nach internen Regeln:

Tastatureieignisse: Ziel ist der Knoten mit Focus

Mausereignisse: Ziel ist der Knoten an der Mausposition

Bem: Bei mehreren Knoten (übereinander) an der Mausposition, wird der oberste gewählt.

**Wegbestimmung**

Die gewählte (initiale) Route ist durch die Event-Dispatch-Chain des Targets bestimmt.

**Event Handling (Ereignisbehandlung)**

* Sowohl Filter als auch Handler werden durch das EventHandler-Interface implementiert. Der Unterschied liegt nur darin, wann (in welcher Phase) welcher ausgeführt wird.

**EventFilter**

Ein Filter an einem Parent-Node kann ein Event abfangen und verhindern, dass es seine Child-Nodes erreicht. Ein Node kann mehrere Filter haben, die Reihenfolge der Ausführung ist durch den Event-Typ vorgegeben:

Erst spezielle Events, dann generische z.B. MouseEvent.MOUSE\_PRESSED kommt vor InputEvent.ANY

**Event Handler**

Ein Knoten kann mehrere Handler besitzen.

Ein Handler im Parent-Node wird nur dann aufgerufen, wenn der Child-Node das Event nicht konsumiert.

Reihenfolge: erst spezielle Events, dann generische.

Ein Event wird konsumiert durch: event.consume();

15.04.16

Filter und Handler Unterschied: Preprocessing and Postprocessing. Die nachgelagerten oder vorgelagerten Events handlen. Siehe Beispiel Code (Consumer) am 15.04.16. Filter kommt zuerst, handler erst nach dem Durchlauf aller Knoten.

Faustregeln: Filter kommt von oben (root) nach unten. Handler kommt von unten. Man sollte immer handler benutzen. Also von unten kommen.

Source geht alle Objekte durch den Baum durch, bis er beim Target ist. Danach geht er den Baum wieder rückwärts durch. Target ist nur das Objekt das ich klicke.

**Convenience-methods** (Bequemlichkeit) zum registrieren eines Event-Handlers:

setOnEvent-Typ(EventHandler<? super event-class> valueName;

event-class kann spezielle Klasse sein, aber auch generelle Klasse (z.B. Event)

**Event-Filter:**

addEventFilter(EventType, EventHandler);

removeEventFilter(EventType, EventHandler);

**Event-Handler:**

addEventHandler und removeEventHandler (Parameter siehe oben)

**Mouse Event:**

* MouseButton getButton(); // MIDDLE, NONE, PRIMARY, SECONDARY (Maustasten)
* Int getClickCount()
* getSceneX(), getSceneY() // Koordinaten relative zur Wurzel
* getSceenX(), getScreenY() // Koordinaten relativ zum Bildschirm
* getX(), getY() // Koordinaten relativ zum Event-Target
* isPrimaryButtonDown(); Secondary, Middle
* isShiftDown();, Alt, Control, Meta, Shortcut
* isStillSincePressed() // ist die Maus innerhalb ihres Hysterese-Gebietes (Tolleranz-Gebiet)
* isSynthesized() // bei Touch-Events

**Koordinaten in Java:**

Ursprung ist oben links. Dann geht es positiv nach rechts (x) und positiv nach unten (y)

06.05.16

BoundingVolume = BoundingBox (mit Z = 0)

Lokale Bounding Box (in lokalen/objekteigenen Koordinaten) -> Breite und Höhe des Objektes

=> Fläche Objekt + Rand

Node:

getBoundsInLocal();

getBoundsInParent();

getLayoutBounds();

localToScene(); -> als Parameter z.B.: Point oder Bounds-Objekt => dadurch lässt sich Objekt nicht aus Fenster verschieben

Parent Bounds:

Fläche Objekt + Rand + Transformationen (z.B. Verschiebung)

Layout Bounds:

Stimmen mit lokalen Bounds überein, werden aber für Layout Berechnungen verwendet.

27.05.16

Menüs

* MenuBar, als Container für Aufklappmenüs.
* Menu, als aufklappbares Menü.
* MenuItems, als einzelne Menüeinträge

obj = new Menu(„Beschriftung“);

obj.getItems();

//Liefert Referenz auf Liste der Menüeinträge

// .add: hinzufügen von Einträgen

MenuBar.getMenus();

//Liefert Referenz auf Liste mit Menüs

Alle Controls verwenden Events zum triggern ihrer Funktionen

z.B. jedes MenuItem, jedes Menu

Toolbar

„Werkzeugleiste“

Items sind Nodes.

ChoiceBox

Auswahl-Steuerelement mit aufklappbarer Liste von Einträgen

SelectionModel:

z.B. „einfach Auswahl“

getSelectedIndex();

getSelectedItem();

//liefern Index bzw. Referenz des ausgewählten Objektes

ComboBox

Kombination aus ChoiceBox und Textfield. => Auswahlelement mit Möglichkeit der Dateneingabe.

Dateneingabe kann mittels: setEditable(true); aktiviert werden

SplitPane

Ist ein Control, das seine Kinder horizontal oder vertikal durch einen Separator trennt.

Wenn die Kinder in der SplitPane Layouts sind, lassen sich die Trenner verschieben. Sonst nicht.

Verlaufsanzeiger

ProgressBar: „Balkenanzeige“

ProgressIndicator: „Kreisförmige Verlaufsanzeige“

Unendliche Laufzeit wird mit Progressvalue == -1 erreicht

Der Stellwert der Verlaufsanzeige liegt im Wertebereich [0;1]

ScrollPane

Sobald der Inhalt nicht vollständig anzeigbar ist, fügt ScrollPane automatisch Scrollbalken ein.

ToggleButtons

Sind Schalter. „Es kann nur einer aktiv sein“

03.06.2016

**Effekte**

Blend = Überblendung

Bloom = Leuchten

BoxBlur = Unschärfe

MotionBlur = Bewegungsunschärfe

GaussianBlur = Gausscher Weichzeichner

Displacement = Pixelverschiebung

SepiaTone = Antike Fotografie

Als Füllfarbe einen Farbverlauf einsetzen: LinearGradient

Farben: Stop

Wiederholung: CycleMethod

**Bilddarstellung**

ImageView:

Größe ändern mit:

setFitWidth(double x);

setFitHeight(double y);

setPreserveRatio(Boolean); // Seitenverhältnis beibehalten

**Zwischenablage des Betriebssystems**

Clipboard

Typisch:

[STRG] + C // etwas ins Clipboard kopieren

[STRG] + V // etwas aus Clipboard entnehmen

**Transformationen**

Im Node:

Obj.getTransforms() //Liste von Transformationen

Obj.add(Transform

Rotate: Winkel, Drehpunkt

Scale: Zoomfaktor für jeweilige Achse, z.B. 50% = 0.5

Translate: Angabe eines Verschiebevektors (Versatz in jeweiliger Achsrichtung)

10.06.2016

**Java Collection Framework**

:= Sammlungen von Objekten

Hinzufügen: Boolean add(object);

Entfernen: Boolean remove(object);

Leeren: void clear();

Füllstand:

* Boolean isEmpty();
* Int size();

**Liste**

Indexzugriff:

* Object getElementAt(int index);
* setElementAt(int index, Object obj);

**Map**

“Mapping” = Abbildung

* Object get(key)
* set(Key, Object)

**Set**

“Menge” im mathematischen Sinne

**ListView**

setItems(ObservableList liste);

**Builder**

“Bauarbeiter” zum Erzeugen von Objekten.

-> Kontruktoraufrufe, sowie spezifische Einstellungen (Attributwerte) werden im Builder gespeichert.

Mittels Object create(); können dann beliebig viele gleichartige Objekte erzeugt werden.

anstelle von:

void setAttr1(int)

void setAttr2(int)

schreibe:

Object setAttr1(int)

Object setAttr2(int)

-> Object settAttr1 (int val){

this.val = val;

return this; }

Vorteil -> MyObject object = new MyObject().setAttr1(0).setAttr2(1);

17.06.2016

Anwender sieht View:

-z.B. Fenster auf Bildschirm

Anwender interagiert mit Controller

-z.B. durch Tastatur, Maus, Touch (Gesten) -> Ereignisse (events)

Controller:

-interpretiert/versteht die Ereignisse bzw. Ereignisfolge

-veranlasst ggf. Änderungen am Modell

Modell:

-Computeradäquates Datenmodell (objektorientiert, relational)

-z.B. mathematisches/physikalisches Simulationsmodell, relationales Datenmodell

-> Ausschnitt aus realer Welt

View und Controller oft schwer trennbar

->Änderungen an der View ziehen i.d.R. Änderungen am Controller nach sich

Aber:

-Modell und View gut trennbar

->Modell kann wiederverwendet werden bzw. View kann ausgetauscht werden

-> Bei Erweiterung d. Anwendung meist Anpassung aller Komponenten

zu a)

Vorteil:

Ermöglicht direktes Binden, Observer kann eingesetzt werden

Nachteil:

Artefakte aus UI (Observer) im Modell

=> Unabhängigkeit/Wiederverwendbarkeit von Modell kann dadurch eingeschränkt sein

(z. B. Abhängigkeit von Observer-Paket in JavaFX)

-> Einsatz von Schnittstellen/Verwendung weiterer Muster kann Abhilfe schaffen

Geeignet für: 1-Tier-Architektur (standalone Anwendung)

zu b)

Vorteil:

- Modell unabhängig von UI -> wiederverwendbar

- Bei Tausch der View nur Verbindung mit Controller implementieren

- Controller kann Objekttypen zwischen UI und Modell transkodieren

Nachteil:

- Mehrfachimplementierung von Funktionalität in Controller (Methodeninflation, Durchreichen von Daten beim Lesen von Modell

- Zusätzlicher Overhead durch transkodieren von Objekttypen

Bemerkung:

View und Controller bei Änderung der UI für gewöhnlich aneinander anpassen

-> Möglichkeit der Zusammenlegung

Wenn z.B. Geschäftslogik Teil des Modell

Geeignet: 3 Tier-Architektur

Tier (engl.) := Stufe, Rang, Reihe, Schicht

View:

- in FXML beschrieben

- Controller-Klasse wird root-Node zugeordnet (fx:controller)

- Ereignis-Handler werden mit Methoden d. Controller-Instanz verknüpft

Controller:

- Referenziert Model (bzw. benötigte Teile)

- Referenziert View Elemente (Annotation: @FXML)

- Verbindet ggf. Modell und View (Observer)

FXMLLoader:

- erzeugt Szenegraph

- erzeugt Controller-Instanz

- verbindet View-Elemente mit Controller

- verbindet Ereignis-Handler mit Methoden d. Controller-Instanz

Application:

- ruft FXMLLoader.load(…); auf

- setzt Szenegraph in Fenster