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)