

# PROCESSING

JAVAFX

Created by Michael Kirsch & Beat Rossmly

# INHALT

## 1. Imports

1. Imports
2. Node Graph

## 2. AnimationTimer

1. Wie können wir einen Loop erzeugen?

## 3. Anonymous Classes

1. Was sind Anonyme Klassen?
2. Wie können wir nun den Kreis bewegen?

## 4. Key Input

1. Wie können wir auf Tasteneingaben reagieren?

IMPORTS

# IMPORTS

Imports binden Funktionalitäten von Entwicklern ein.

```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class Main extends Application {
    Circle ball;

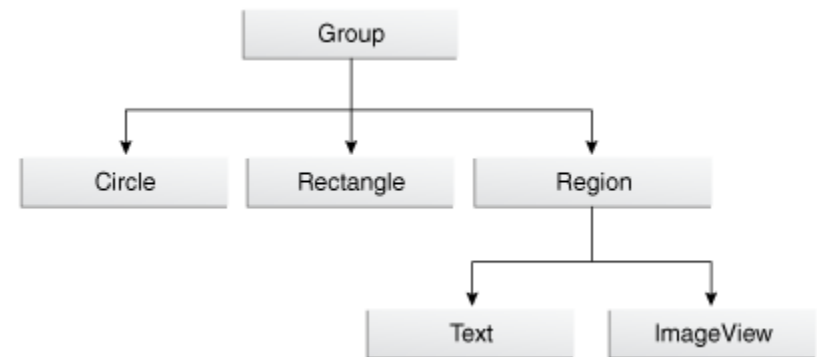
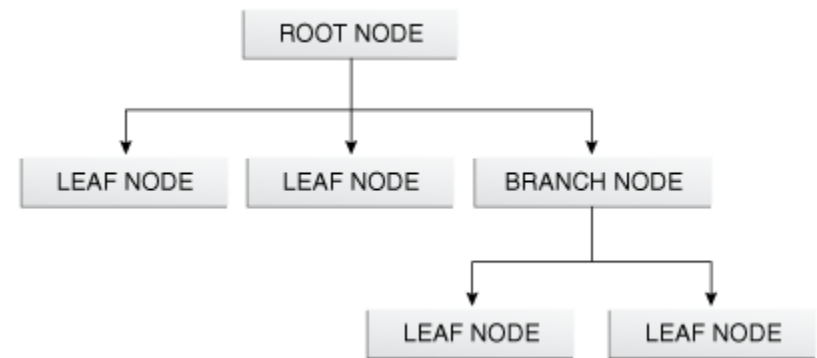
    public void start(Stage primaryStage) throws Exception{
        Group root = new Group();
        ball = new Circle (100,100,50);
        root.getChildren().add(ball);

        primaryStage.setTitle("Pong");
        primaryStage.setScene(new Scene(root, 300, 275));
        primaryStage.show();
    }

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

# NODE GRAPH

- Der Node Graph enthält alle UI Elemente.
- Die "Leafs" des Graphen sind dabei die sichtbaren Elemente.
- Die "Branches" stellen strukturierende Elemente dar, die wiederum Nodes als Kinder haben.



# NODE GRAPH

- Der Befehl **getChildren()** returned die Liste aller Kind-Knoten eines Nodes.
- Durch **add** können wir ein neues Kind zu dieser Liste hinzufügen.
- Auf diese Weise bauen wir den Node-Graph manuell auf.

```
//...  
Group root = new Group();  
ball = new Circle (100,100,50);  
root.getChildren().add(ball);  
//...
```

ANIMATIONTIMER

# WIE KÖNNEN WIR EINEN LOOP ERZEUGEN?

Betrachten wir den bisherigen Code können wir bereits in **start**,  
ähnlich wie in **setup** unsere Applikation initialisieren.

Wo führen wir nun aber unseren **draw** Loop aus?



```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

public class Main extends Application {
    Circle ball;

    public void start(Stage primaryStage) throws Exception{
        // setup
        Group root = new Group();
        ball = new Circle (100,100,50);
        root.getChildren().add(ball);
        primaryStage.setTitle("Pong");
        primaryStage.setScene(new Scene(root, 300, 275));
        primaryStage.show();

        // draw
        while(true) { /* hier könnte unser Code stehen */ }
    }

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

- Die auf der vorherigen Folie demonstrierte Methode über eine nicht endene **while** Schleife einen Loop zu erzeugen, in dem wir unsere Animationen zeichnen können ist möglich, bietet aber einige Nachteile.
- Besser ist es auf von JavaFX vorgefertigte Methodiken zurückzugreifen.
- Für diesen Fall bietet JavaFX die Klasse **AnimationTimer**, die Code in einem gleichmäßigen Zeitabstand wiederholt.
- Dieser wird typischerweise wie folgend eingebunden.

```
//...

public void start(Stage primaryStage) throws Exception{
    // setup
    Group root = new Group();
    ball = new Circle (100,100,50);
    root.getChildren().add(ball);
    primaryStage.setTitle("Pong");
    primaryStage.setScene(new Scene(root, 300, 275));
    primaryStage.show();

    // draw
    new AnimationTimer () {
        public void handle (long currentNanoTime) {
            /* hier könnte unser Code stehen */
        }
    }.start();
}

//...
}
```

Was passiert hier?

ANONYMOUS CLASSES

# WAS SIND ANONYME KLASSEN?

- Anonyme Klassen erlauben es uns Variationen von Klassen während der Initialisierung eines Objekts zu erzeugen.

```
public class XXX {
    int x;

    public XXX () {}

    void doSomething () {
        System.out.println("do something");
    }
}

public class Main {
    public static void main (String [] args) {
        XXX test = new XXX () {
            @Override
            void doSomething() {
                System.out.println("do something else");
            }
        };

        test.doSomething(); // -> "do something else"
    }
}
```

- Beim Konstruktoraufruf haben wir die Möglichkeit die verwendete Klasse zu überschreiben.
- Das bedeutet bereits angelegte Funktionalität kann abgeändert werden, aber keine neue hinzugefügt.
- Das liegt daran, dass der Datentyp der Elternklasse diese Funktionalität nicht enthält und daher nicht auf diese verwiesen werden kann.
- Der Code kann noch weiter verknapppt werden.



```
public class XXX {
    int x;

    public XXX () {}

    void doSomething () {
        System.out.println("do something");
    }
}

public class Main {
    public static void main (String [] args) {
        new XXX () {
            @Override
            void doSomething() {
                System.out.println("do something else");
            }
        }.doSomething(); // -> "do something else"
    }
}
```

- Nun wird direkt ein Objekt ohne Ziel erzeugt.
- Durch den `.` Operator kann sofort auf dem erzeugten Objekt eine Methode aufgerufen werden.
- Dies sehen wir auch beim `AnimationTimer`.

```
//...

public void start(Stage primaryStage) throws Exception{
    // setup
    Group root = new Group();
    ball = new Circle (100,100,50);
    root.getChildren().add(ball);
    primaryStage.setTitle("Pong");
    primaryStage.setScene(new Scene(root, 300, 275));
    primaryStage.show();

    // draw
    new AnimationTimer () {
        public void handle (long currentNanoTime) {
            /* hier könnte unser Code stehen */
        }
    }.start();
}

//...
}
```

WIE KÖNNEN WIR NUN DEN KREIS  
BEWEGEN?

```
//...

public void start(Stage primaryStage) throws Exception{
    // setup
    Group root = new Group();
    ball = new Circle (100,100,50);
    root.getChildren().add(ball);
    primaryStage.setTitle("Pong");
    primaryStage.setScene(new Scene(root, 300, 275));
    primaryStage.show();

    int x = 0;

    // draw
    new AnimationTimer () {
        public void handle (long currentTime) {
            x++;
            ball.setCenterX(x);
        }
    }.start();
}

//...
}
```

KEY INPUT

WIE KÖNNEN WIR AUF TASTENEINGABEN  
REAGIEREN?

```

//...
public void start(Stage primaryStage) throws Exception{
    Group root = new Group();
    ball = new Circle (100,100,50);
    root.getChildren().add(ball);
    primaryStage.setTitle("Pong");
    primaryStage.setScene(new Scene(root, 300, 275));
    primaryStage.show();

    new AnimationTimer () {
        public void handle (long currentTime) {
            /* hier könnte unser Code stehen */
        }
    }.start();

    // Tasten Eingabe
    scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
        @Override
        public void handle(KeyEvent event) { /* Tastendruck */}
    });
    scene.setOnKeyReleased(new EventHandler<KeyEvent>() {
        @Override
        public void handle(KeyEvent event) { /* Loslassen der Tasten */}
    });
}
//...

```



- Auch hier sehen wir wieder die Verwendung anonymer Klassen.
- Die Klasse **EventHandler** wird beim Übergeben an **scene** überschrieben.

# QUELLEN

- <https://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm>