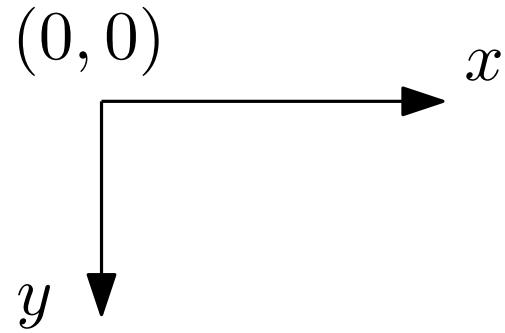
JavaFX
Koordinaten und Transformationen

Koordinaten

Jedes `Node`-Objekt hat sein eigenes Koordinatensystem.

In Container-Nodes beziehen sich Position und Größe der Kinder immer auf das Koordinatensystem der enthaltenden Node.



Koordinaten sind `double`-Werte.

Transformationen

Auf jedes Node-Objekt können Transformationen angewendet werden, um as Koordinatensystem zu verändern.

Beispiele:

- **Verschiebung**
  Der Ursprung des Koordinatensystem wird verschoben.
- **Rotation**
  Rotation des gesamten Koordinatensystems um einen Punkt.
- **Skalierung**
  Skalierung des gesamten Koordinatensystems auf einen Punkt.

Transformationen

Jedes Node-Objekt hat eine Liste von Transformationen, die nacheinander angewendet werden.

Damit können kompliziertere Transformationen leicht zusammengesetzt und verändert werden.

```
// 1. Verschieben um (50, 50)
node.getTransforms().add(new Translate(50, 50));

// 2. Skalieren um Faktor 1.5 mit Mittelpunkt (10, 10)
node.getTransforms().add(new Scale(1.5, 1.5, 10, 10));

// 3. Rotation um 30 Grad mit Mittelpunkt (50, 30)
node.getTransforms().add(new Rotate(30, 50, 30));
```

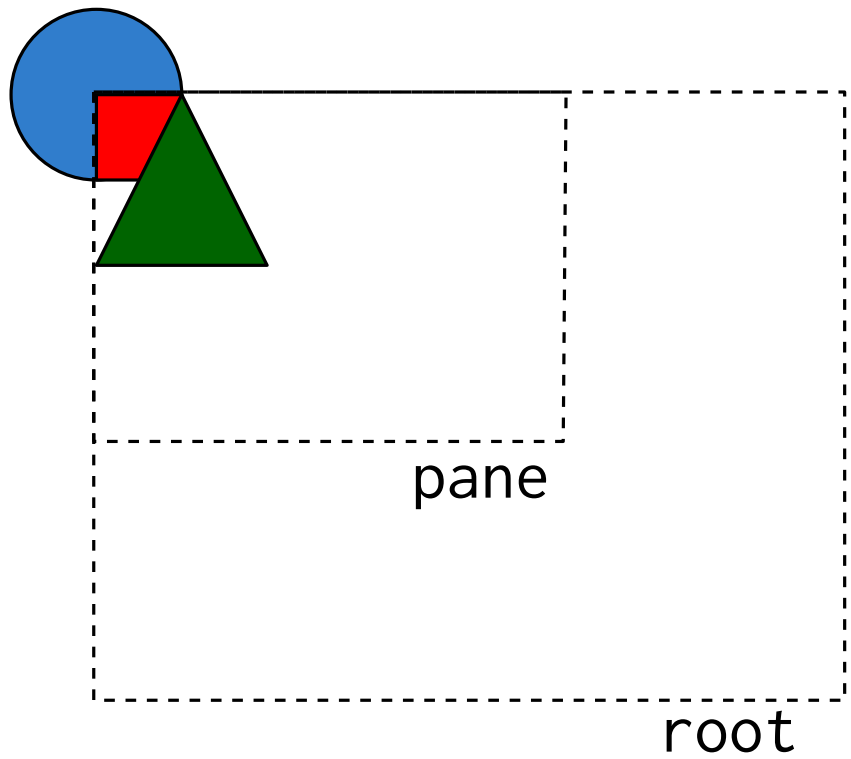
Transformationen

Beispiel: manuelles Layout in Pane

```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);
```



Transformationen

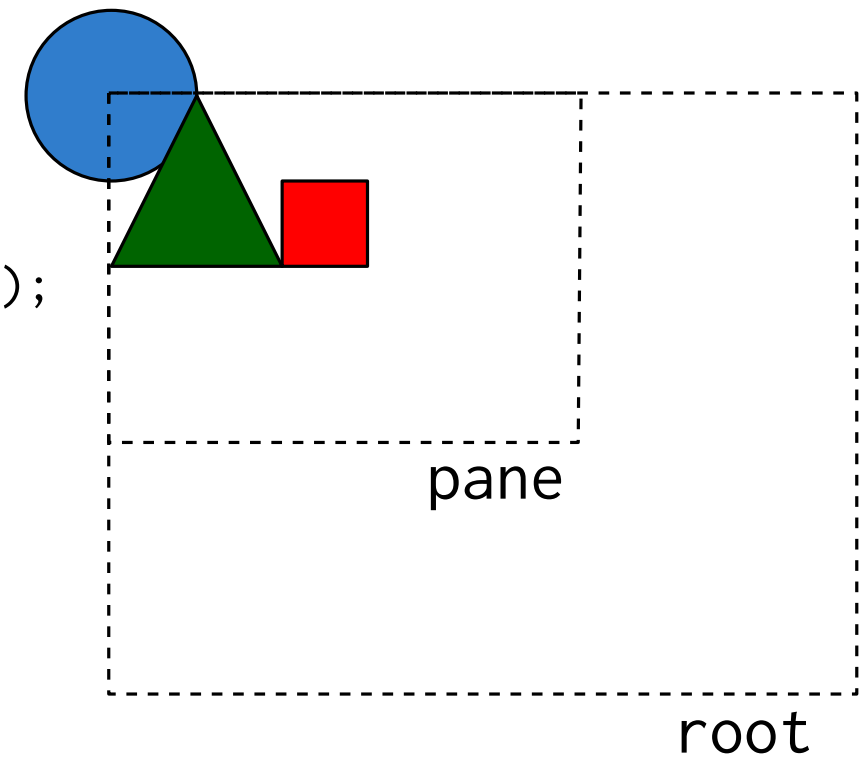
Beispiel: manuelles Layout in Pane

```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);

r.getTransforms().add(new Translate(40, 20));
```



Transformationen

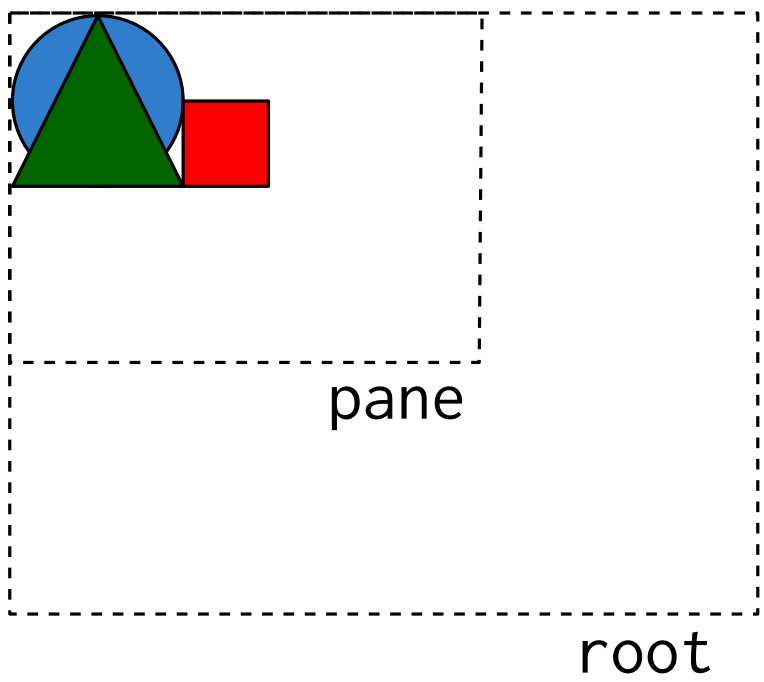
Beispiel: manuelles Layout in Pane

```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);

r.getTransforms().add(new Translate(40, 20));
c.getTransforms().add(new Translate(20, 20));
```



Transformationen

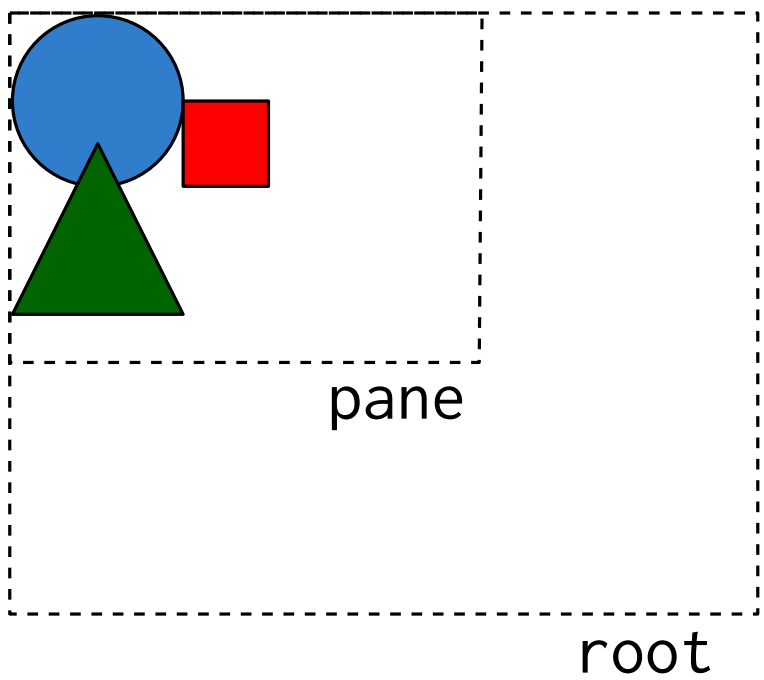
Beispiel: manuelles Layout in Pane

```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);

r.getTransforms().add(new Translate(40, 20));
c.getTransforms().add(new Translate(20, 20));
d.getTransforms().add(new Translate(0, 30));
```



Transformationen

Beispiel: manuelles Layout in Pane

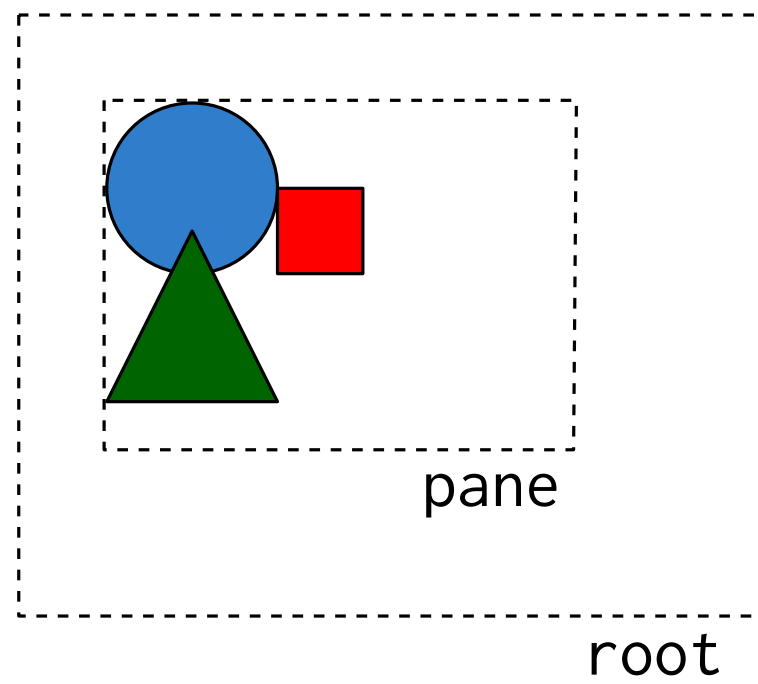
```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);

r.getTransforms().add(new Translate(40, 20));
c.getTransforms().add(new Translate(20, 20));
d.getTransforms().add(new Translate(0, 30));

pane.getTransforms.add(new Translate(20, 20));
```



Transformationen

Beispiel: manuelles Layout in Pane

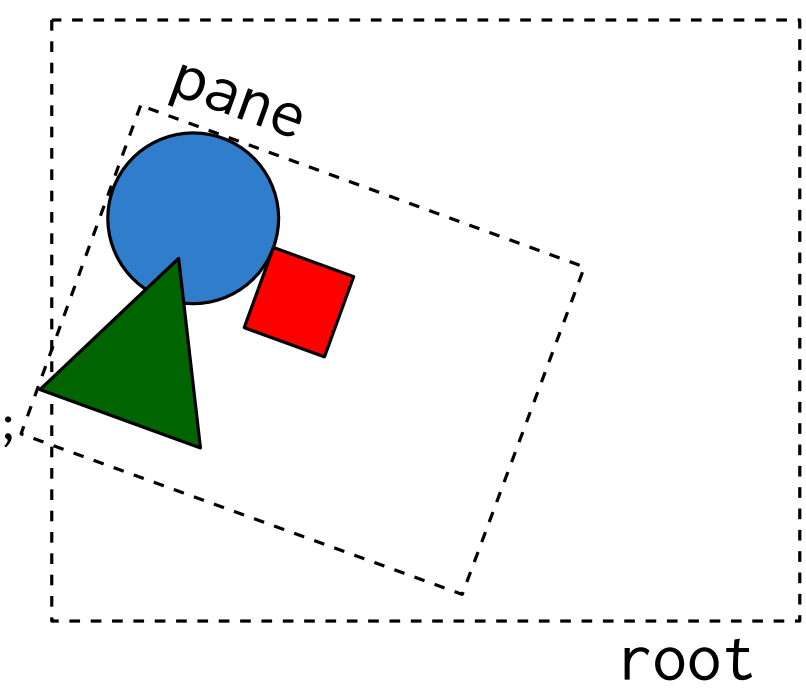
```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);

r.getTransforms().add(new Translate(40, 20));
c.getTransforms().add(new Translate(20, 20));
d.getTransforms().add(new Translate(0, 30));

pane.getTransforms.add(new Translate(20, 20));
pane.getTransforms.add(new Rotate(20, 0, 0));
```



Transformationen

Beispiel: manuelles Layout in Pane

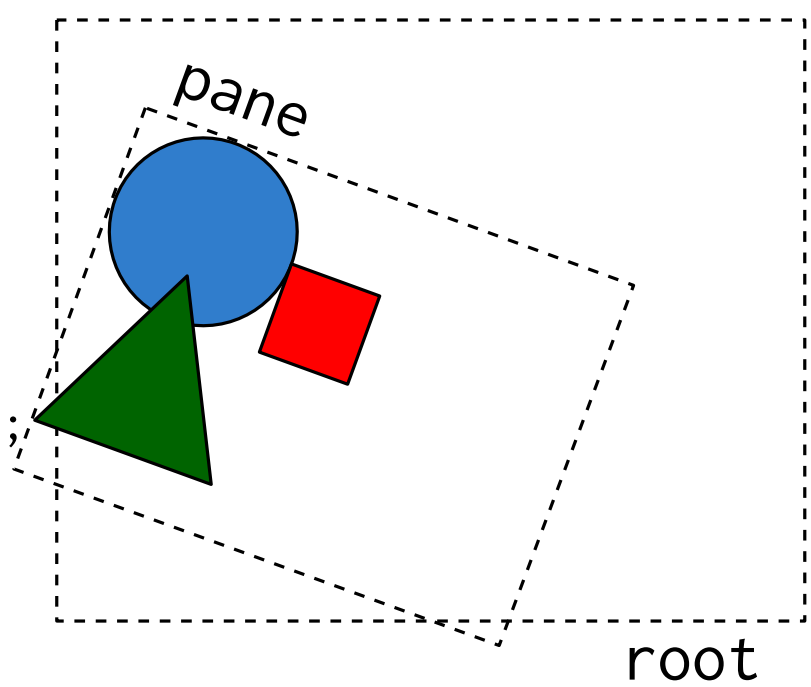
```
Pane root = new Pane();

Pane pane = new Pane();
Rectangle r = new Rectangle(20, 20);
Circle c = new Circle(20);
Polygon d = new Polygon(0, 40, 40, 40, 20, 0);
pane.getChildren().addAll(c, r, d);
root.getChildren().add(pane);

r.setFill(Color.RED);
c.setFill(Color.BLUE);
d.setFill(Color.GREEN);

r.getTransforms().add(new Translate(40, 20));
c.getTransforms().add(new Translate(20, 20));
d.getTransforms().add(new Translate(0, 30));

pane.getTransforms.add(new Translate(20, 20));
pane.getTransforms.add(new Rotate(20, 0, 0));
pane.getTransforms.add(new Scale(1.1, 1.1));
```



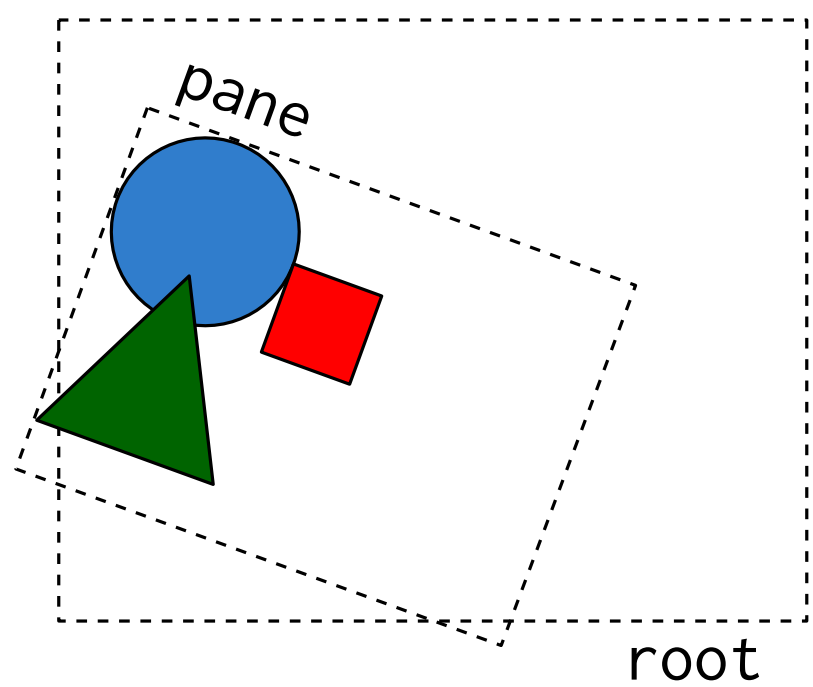
Transformationen

Beispiel: Wir wollen, dass die drei Figuren der Maus folgen.

```
... // Aufbau von root und pane wie eben

final Translate translate = new Translate(0, 0);
pane.getTransforms().add(translate);
pane.getTransforms().add(new Rotate(20, 0, 0));
pane.getTransforms().add(new Scale(1.1, 1.1));

root.setOnMouseMoved(event -> {
  translate.setX(event.getX());
  translate.setY(event.getY());
});
```

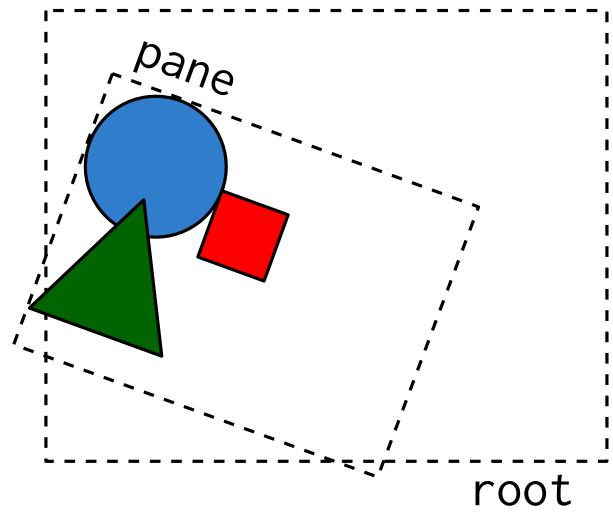


Umrechnen von Koordinaten

Beispiel: Wir wollen, dass die drei Figuren der Maus folgen und dass der Mittelpunkt des Kreises unter dem Mauszeiger zentriert ist.

Wir müssen dazu die Koordinaten des Kreismittelpunkts bezüglich des Koordinatensystems von `root` berechnen.

- Im Koordinatensystem von `pane` hat der Kreis den Mittelpunkt $(20, 20)$.
- Durch die Rotation und die Skalierung der `pane` wird die Umrechnung der Koordinaten kompliziert.
- JavaFX stellt Methoden zur Umrechnung von Koordinaten zur Verfügung.



Umrechnen von Koordinaten

Jedes `Node`-Objekt hat folgende Methoden:

- Umwandlung einer Koordinate vom Koordinatensystem der Node in das der ganzen Scene:

  ```
  Point2D localToScene(double x, double y)
  ```

- Umwandlung in die andere Richtung:

  ```
  Point2D sceneToLocal(double x, double y)
  ```

Berechnung des Mittelpunkt des Kreises bzgl. des Koordinatensystems von `root`:

```
// pane-Koordinate => Scene-Koordinate
Point2D ps = pane.localToScene(20, 20);
// Scene-Koordinate => root-Koordinate
Point2D pr = root.sceneToLocal(ps.getX(), ps.getY());
```

(Ergebnis: `Point2D [x = 17.930173873901367, y = 38.45138168334961]`)

Umrechnen von Koordinaten

Beispiel: Wir wollen, dass die drei Figuren der Maus folgen und dass der Mittelpunkt des Kreises soll unter dem Mauszeiger zentriert ist.

```
root.setOnMouseMoved(event -> {
  // Mittelpunkt des Kreises in root-Koordinaten berechnen
  Point2D circleCenterScene = pane.localToScene(20, 20);
  Point2D circleCenterRoot = root.sceneToLocal(circleCenterScene);

  // Position der Maus in root-Koordinaten berechnen
  Point2D mouseRoot = root.sceneToLocal(event.getSceneX(), event.getSceneY()

  // Verschiebung von pane um die Differenz
  // Am Anfang der Liste anfuegen: Verschiebung vor Rotation und Skalierung
  pane.getTransforms().add(0,
    new Translate(mouseRoot.getX() - circleCenterRoot.getX(),
                  mouseRoot.getY() - circleCenterRoot.getY()));
  });
```

Zoom und Rotation

Beispiel: Das bisherige Beispiel soll erweitert werden:

- Bei Linksklick soll auf den Mittelpunkt des Kreises hineingezoomt werden.
- Bei Rechtsklick soll $10°$ um den Mittelpunkt des Kreises rotiert werden.

```
root.setOnMouseClicked(event -> {
  if (event.getButton() == MouseButton.PRIMARY) {
    pane.getTransforms().add(new Scale(1.1, 1.1, 20, 20));
  } else if (event.getButton() == MouseButton.SECONDARY) {
    pane.getTransforms().add(new Rotate(10, 20, 20));
  }
});
```

Hier kann man den Punkt, auf den skaliert bzw. um den rotiert wird, direkt angeben. Oft würde man ihn mit `sceneToLocal`, `localToScene` und ähnlichen Methoden bestimmen.

Affine Transformationen

Problem: Die Liste der Transformationen `pane.getTransforms()` wird mit jeder Mausbewegung und jedem Klick länger (bis der Speicher irgendwann voll ist)!

Abhilfe: Jede Folge von Translationen, Rotationen und Skalierungen kann zu einer einzigen `Affine`-Transformation zusammengefasst werden.

Wir können eine einzige `Affine`-Transformation verwenden.

```
Affine affine = new Affine();
pane.getTransforms().add(affine);

affine.append(new Rotate(40));
affine.append(new Scale(1.5, 1.5));
```

Affine Transformationen

Translationen, Rotationen und Skalierung sind Spezialfälle *affiner Transformationen*.

Jede dreidimensionale affine Transformation ist durch 12 Zahlen $m_{xx}, m_{xy}, m_{xz}, t_x, m_{yx}, m_{yy}, m_{yz}, t_y, m_{zx}, m_{zy}, m_{zz}, t_z$ gegeben.

Diese Zahlen definieren, wie ein beliebiger gegebener Punkt zu transformieren ist.

Der Punkt (x, y, z) wird zu folgendem Punkt (x', y', z').

$$x' = m_{xx} \cdot x + m_{xy} \cdot y + m_{xz} \cdot z + t_x$$
$$y' = m_{yx} \cdot x + m_{yy} \cdot y + m_{yz} \cdot z + t_y$$
$$z' = m_{zx} \cdot x + m_{zy} \cdot y + m_{zz} \cdot z + t_z$$

Affine Transformationen

Affine Transformationen werden in JavaFX durch die Klasse `Affine` repräsentiert.

- Objekte dieser Klasse können genau wie `Transform`, `Scale` und `Rotate` verwendet werden.
- Es ist möglich, affine Transformation direkt durch Angabe der 12 Werte definieren.
- Einfacher ist die Verwendung von Hilfsmethoden.

Affine Transformationen

Beispiel: Konstruktion einer einzige affinen Transformation, die zuerst verschiebt, dann skaliert und dann dreht.

```
// Konstruktor erzeugt die Transformation,
// die nichts macht:
Affine affine = new Affine();

// Anhaengen einer Translation
affine.append(new Translate(50, 50));

// Anhaengen einer Skalierung
affine.append(new Scale(1.5, 1.5, 10, 10));

// Anhaengen einer Rotation
affine.append(new Rotate(30, 50, 30));

System.out.println(affine);
```

Affine Transformationen

Beispiel: Konstruktion einer einzige affinen Transformation, die zuerst verschiebt, dann skaliert und dann dreht.

```
// Konstruktor erzeugt die Transformation,
// die nichts macht:
Affine affine = new Affine();

// An
affin

// An
affin

// An
affin

Syste
```

`System.out.println(affine)` liefert:

```
Affine [
1.299038105676658, -0.7499999999999999, 0.0, 77.5480947161671
0.7499999999999999, 1.299038105676658, 0.0, 13.52885682970026
0.0, 0.0, 1.0, 0.0
]
```

Bedeutung:

```
Affine [
   mxx  mxy  mxz  tx
   myx  myy  myz  ty
   mzx  mzy  mzz  tz
]
```

Affine Transformationen

Erinnerung an Problem:

Die Liste der Transformationen `pane.getTransforms()` wird mit jeder Mausbewegung und jedem Klick länger, bis der Speicher voll ist!

Lösung:

Verwende eine einzige affine Transformation

```
Affine affine = new Affine();
pane.getTransforms().add(affine);
```

und nimm folgende Ersetzungen vor:

```
   pane.getTransforms().add(e);  ⟶  affine.append(e);
pane.getTransforms().add(0, e);  ⟶  affine.prepend(e);
```

(vollständiger Programmcode siehe Praktikumshomepage)

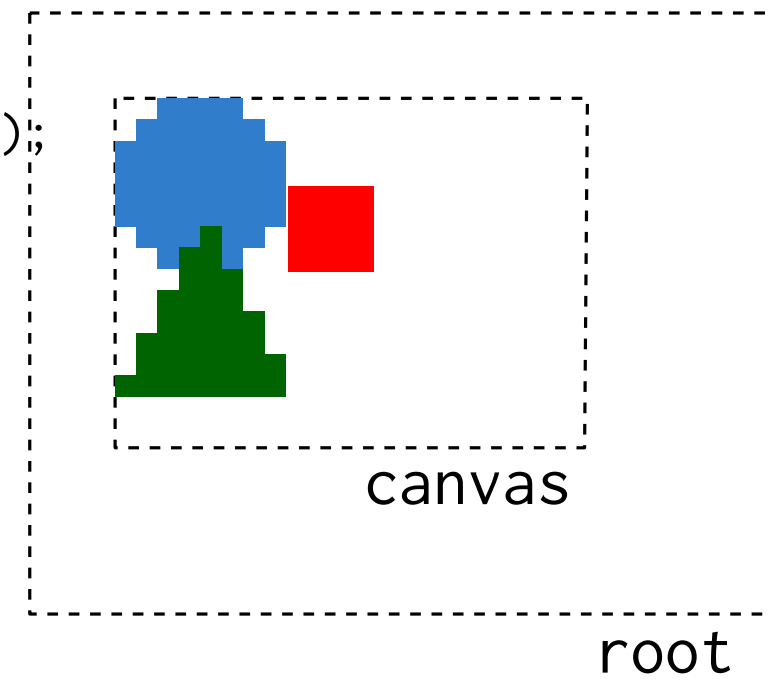
Transformationen und Canvas

Bei Benutzung von Canvas ist zu beachten, dass hier die Pixel des Canvas transformiert werden.

```
Pane root = new Pane();
Canvas canvas = new Canvas(200, 200);
root.getChildren().add(pane);

GraphicsContext gc = canvas.getGraphicsContext2D();
gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
gc.setFill(Color.RED);
gc.fillRect(40, 20, 20, 20);
... // Zeichnen von Kreis und Dreieck

canvas.getTransforms.add(new Translate(20, 20));
```



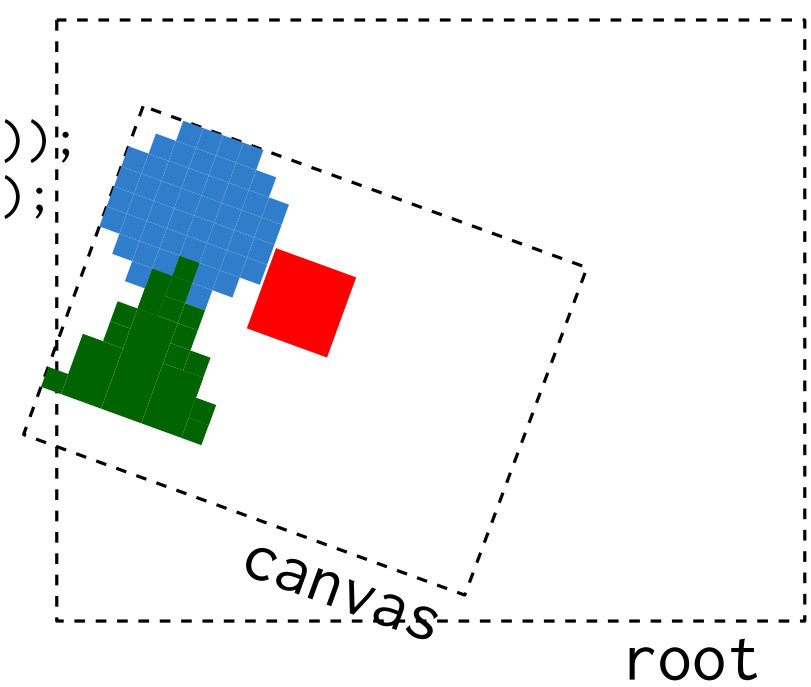
Transformationen und Canvas

Bei Benutzung von Canvas ist zu beachten, dass hier die Pixel des Canvas transformiert werden.

```
Pane root = new Pane();
Canvas canvas = new Canvas(200, 200);
root.getChildren().add(pane);

GraphicsContext gc = canvas.getGraphicsContext2D();
gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
gc.setFill(Color.RED);
gc.fillRect(40, 20, 20, 20);
... // Zeichnen von Kreis und Dreieck

canvas.getTransforms.add(new Translate(20, 20));
canvas.getTransforms.add(new Rotate(20, 0, 0));
```



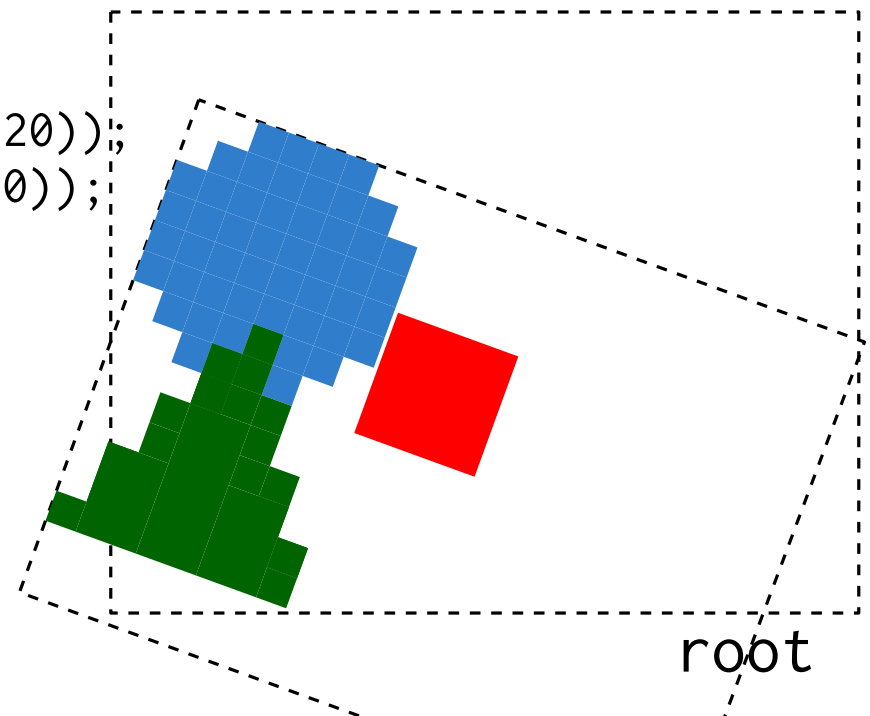
Transformationen und Canvas

Bei Benutzung von Canvas ist zu beachten, dass hier die Pixel des Canvas transformiert werden.

```
Pane root = new Pane();
Canvas canvas = new Canvas(200, 200);
root.getChildren().add(pane);

GraphicsContext gc = canvas.getGraphicsContext2D();
gc.clearRect(0, 0, canvas.getWidth(), canvas.getHeight());
gc.setFill(Color.RED);
gc.fillRect(40, 20, 20, 20);
... // Zeichnen von Kreis und Dreieck

canvas.getTransforms.add(new Translate(20, 20));
canvas.getTransforms.add(new Rotate(20, 0, 0));
canvas.getTransforms.add(new Scale(2, 2));
```



Transformationen und Canvas

Rotation und Skalierung einer Canvas-Node führt meist zu schlechten Ergebnissen.

Soll die Anzeige in einem Canvas skaliert werden, so müssen die Koordinaten beim Zeichnen im Canvas skaliert werden.

- Möglichkeit 1: manuelles Umrechnen der Koordinaten
- Möglichkeit 2: Canvas erlaubt die Umrechnung von Koordinaten *vor dem Zeichnen* mit einer affinen Transformation
  (Methode `setTransform` in `GraphicsContext`).

Transformationen und Canvas

Beispiel für `GraphicsContext.setTransform`:

Die Anweisungen

```
GraphicsContext gc = canvas.getGraphicsContext2D();
Affine affine = new Affine();
affine.append(new Translate(5, 10));
affine.append(new Scale(2, 3));
gc.setTransform(affine);
gc.fillRect(10, 20, 30, 40);
```

haben den gleichen Effekt wie

```
GraphicsContext gc = canvas.getGraphicsContext2D();
gc.fillRect(2*10 + 5, 3*20 + 10, 2*30 + 5, 3*40 + 10);
```

Transformationen können also das Umrechnen von Koordinaten erleichtern.