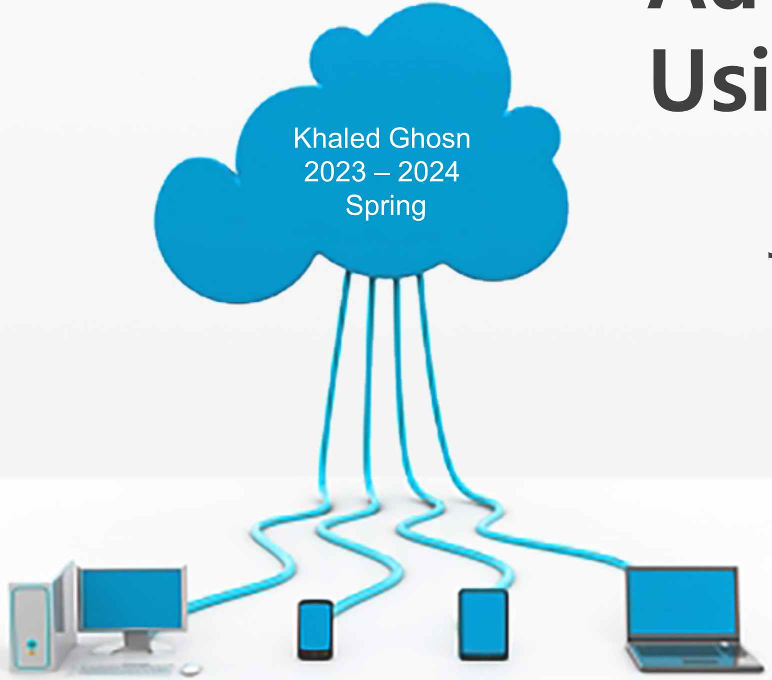
ARTS, SCIENCES & TECHNOLOGY UNIVERSITY IN LEBANON

AUL

CCE 417

# Advanced Programming Using Java


Khaled Ghosn
2023 – 2024
Spring

## JavaFx Animation

# JavaFx Animation

## Animation Basics

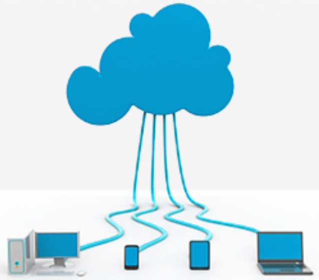
In general, the animation can be defined as the **transition** which creates the myth of motion for an object.

It is the set of transformations applied on an object over the specified **duration** **sequentially** so that the object can be shown as it is in motion.

This can be done by the rapid display of frames.

Animation in JavaFX can be divided into:

- ✓ **Transitions**
- ✓ **Timeline animation**



# JavaFx Animation

## Animation Basics

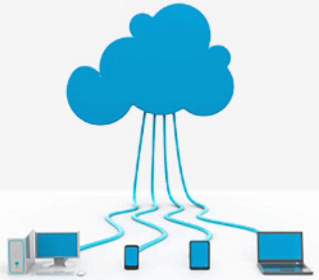
The JavaFX animation support enables you to animate JavaFX shapes and controls, e.g. moving or rotating them.

JavaFX provides easy to use animation API (javafx.animation package).

The package **javafx.animation** contains all the classes to apply the animations onto the nodes.

All the classes of this package extend the class

**javafx.animation.Animation**



# JavaFx Animation

## Animation Class

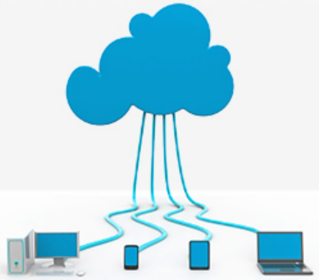
The class **Animation** provides the core functionality of all animations (used in the JavaFX runtime).

An animation can run in a loop by setting **cycleCount**.

To make an animation run back and forth while looping, set the **autoReverse** -flag.

Call **play ( )** or **playFromStart ( )** to play an Animation.

An Animation can be paused by calling **pause( )**, and the next **play( )** call will resume the Animation from where it was paused.



# JavaFx Animation

## Animation Class

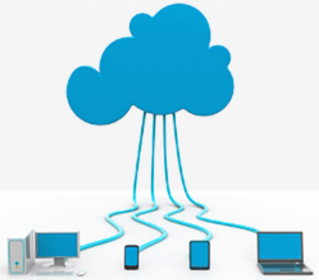
The Animation progresses in the direction and speed specified by **rate**, and stops when its **duration** is elapsed.

Inverting the value of **rate** toggles the play direction.

An Animation with **indefinite** duration (a **cycleCount** of **INDEFINITE**) runs repeatedly until the **stop( )** method is explicitly called, which will stop the running Animation and reset its play head to the initial position.

Animation in JavaFX can be divided into:

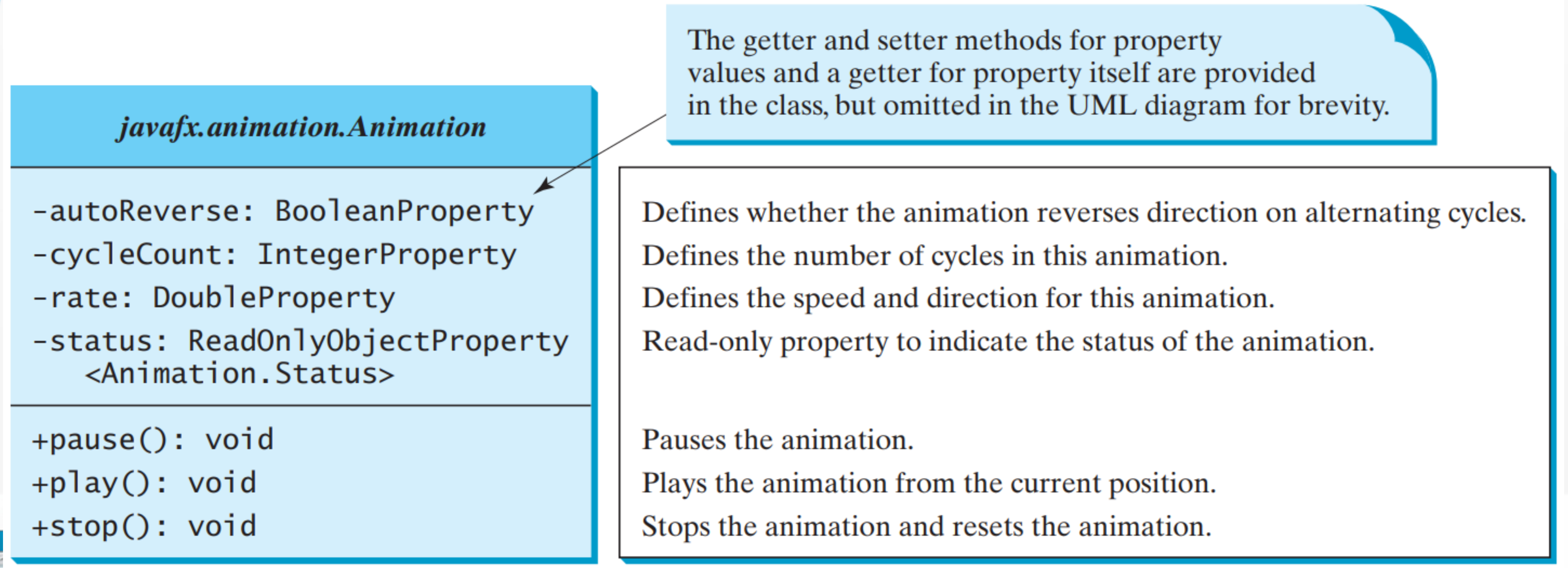
- ✓ **Transitions**
- ✓ **Timeline animation**



# JavaFx Animation

## Animation Class

The abstract **Animation** class is the root class for JavaFX animations



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

```
javafx.animation.Animation

-autoReverse: BooleanProperty
-cycleCount: IntegerProperty
-rate: DoubleProperty
-status: ReadOnlyObjectProperty
   <Animation.Status>

+pause(): void
+play(): void
+stop(): void
```

Defines whether the animation reverses direction on alternating cycles.
Defines the number of cycles in this animation.
Defines the speed and direction for this animation.
Read-only property to indicate the status of the animation.

Pauses the animation.
Plays the animation from the current position.
Stops the animation and resets the animation.

# JavaFx Animation

## Basic Transitions

| Transition | Description |
|---|---|
| **Fade Transition** | creates a fade effect animation that spans its duration |
| **Fill Transition** | creates an animation, that changes the filling of a shape over a duration |
| **Parallel Transition** | plays a list of animations in parallel |
| **Path Transition** | creates a path animation that spans its PathTransition.duration |
| **Pause Transition** | executes an Animation.onFinished at the end of its PauseTransition.duration |
| **Rotate Transition** | creates a rotation animation that spans its duration |
| **Scale Transition** | creates a scale animation that spans its ScaleTransition.duration |
| **Sequential Transition** | plays a list of Animations in sequential order |
| **Stroke Transition** | creates an animation, that changes the stroke color of a shape over a duration |
| **Translate Transition** | creates a move/translate animation that spans its TranslateTransition.duration |

# Transitions

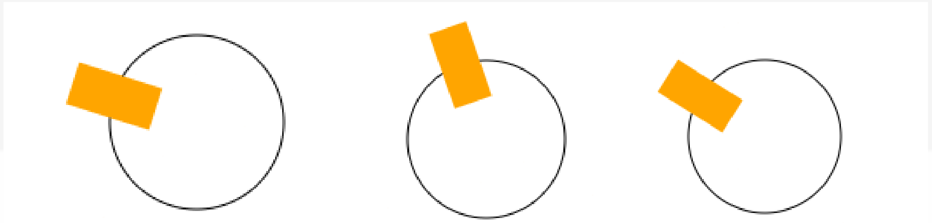
## Path Transition

The **PathTransition** class animates the moves of a node along a path from one end to the other over a given time

The **Duration** class defines a duration of time.

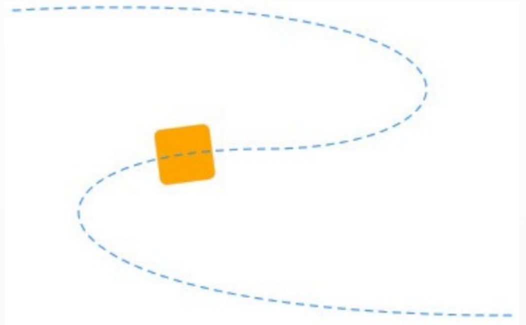
The class defines constants **INDEFINTE**, **ONE**, **UNKNOWN**, and **ZERO** to represent an indefinite duration, 1 milliseconds, unknow, and 0 duration.

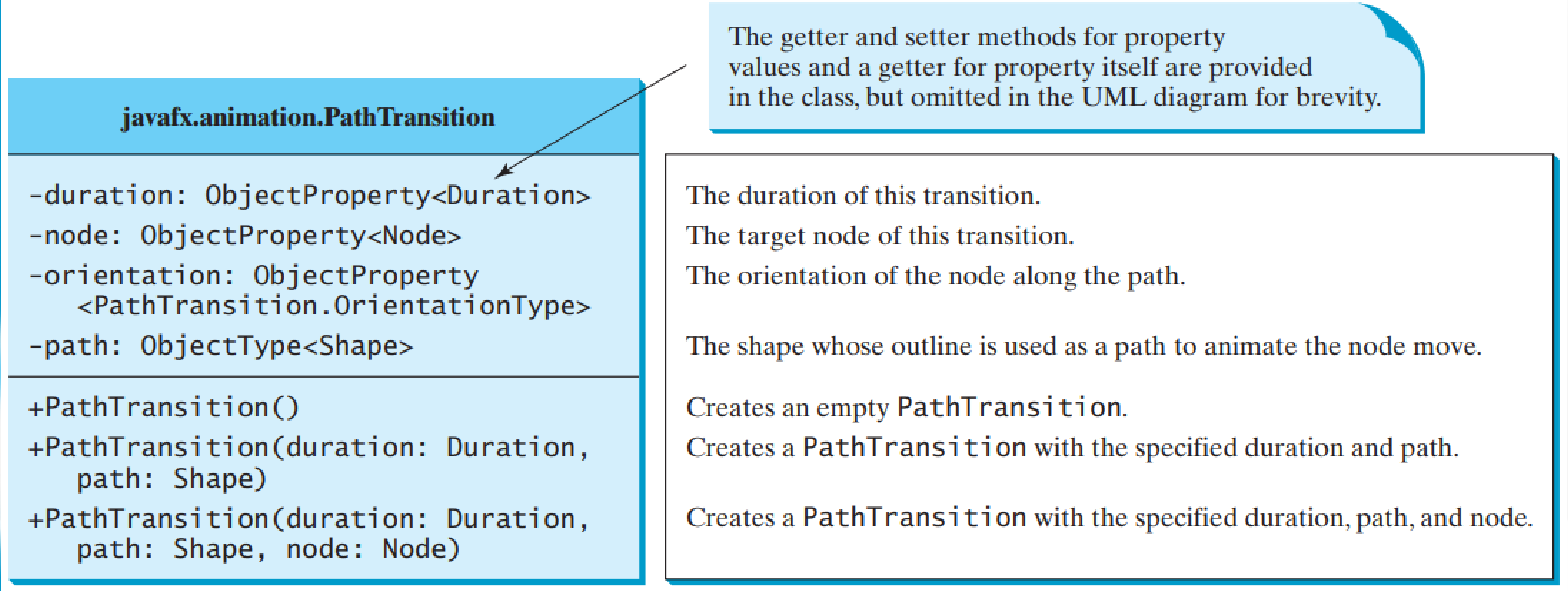
You can use new Duration(double millis) to create an instance of Duration



# Transitions

## Path Transition





The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

**javafx.animation.PathTransition**

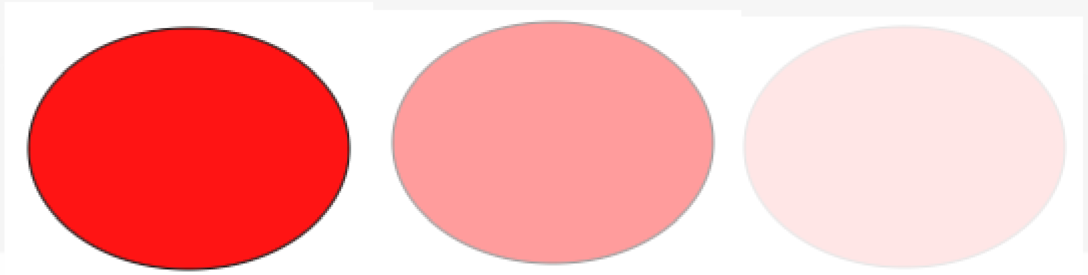
| | |
|---|---|
| `-duration: ObjectProperty<Duration>` | The duration of this transition. |
| `-node: ObjectProperty<Node>` | The target node of this transition. |
| `-orientation: ObjectProperty <PathTransition.OrientationType>` | The orientation of the node along the path. |
| `-path: ObjectType<Shape>` | The shape whose outline is used as a path to animate the node move. |
| `+PathTransition()` | Creates an empty `PathTransition`. |
| `+PathTransition(duration: Duration, path: Shape)` | Creates a `PathTransition` with the specified duration and path. |
| `+PathTransition(duration: Duration, path: Shape, node: Node)` | Creates a `PathTransition` with the specified duration, path, and node. |

## Transitions

### Fade Transition

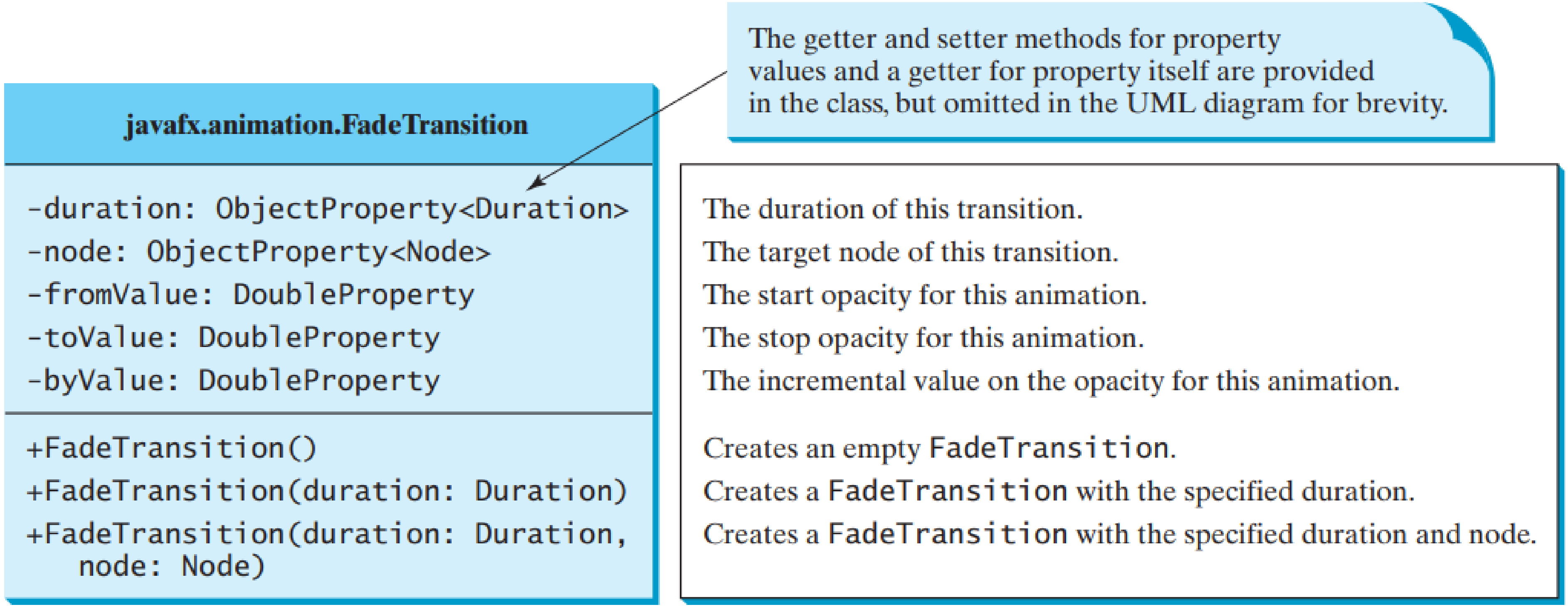
**FadeTransition** class animates the change of the opacity in a node over a given time.

**FadeTransition** creates a fade effect animation that spans its duration. This is done by updating the opacity variable of the node at regular interval.



## Transitions

### Fade Transition



The getter and setter methods for property values and a getter for property itself are provided in the class, but omitted in the UML diagram for brevity.

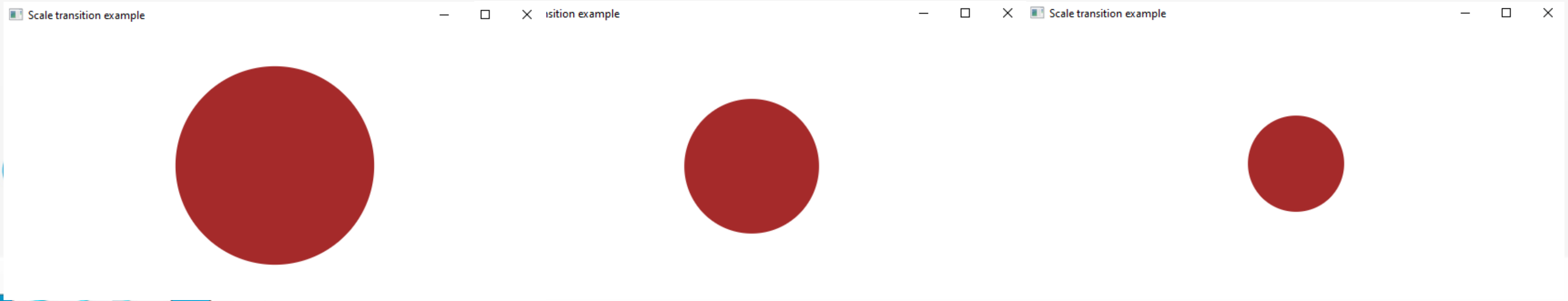
| javafx.animation.FadeTransition | |
|---|---|
| `-duration: ObjectProperty<Duration>` | The duration of this transition. |
| `-node: ObjectProperty<Node>` | The target node of this transition. |
| `-fromValue: DoubleProperty` | The start opacity for this animation. |
| `-toValue: DoubleProperty` | The stop opacity for this animation. |
| `-byValue: DoubleProperty` | The incremental value on the opacity for this animation. |
| `+FadeTransition()` | Creates an empty `FadeTransition`. |
| `+FadeTransition(duration: Duration)` | Creates a `FadeTransition` with the specified duration. |
| `+FadeTransition(duration: Duration, node: Node)` | Creates a `FadeTransition` with the specified duration and node. |

## Transitions

### Scale Transition

**Scale transition** is another JavaFX animation which can be used out of the box that allows to animate the scale / zoom of the given object

The object can be enlarged or minimized using this animation.

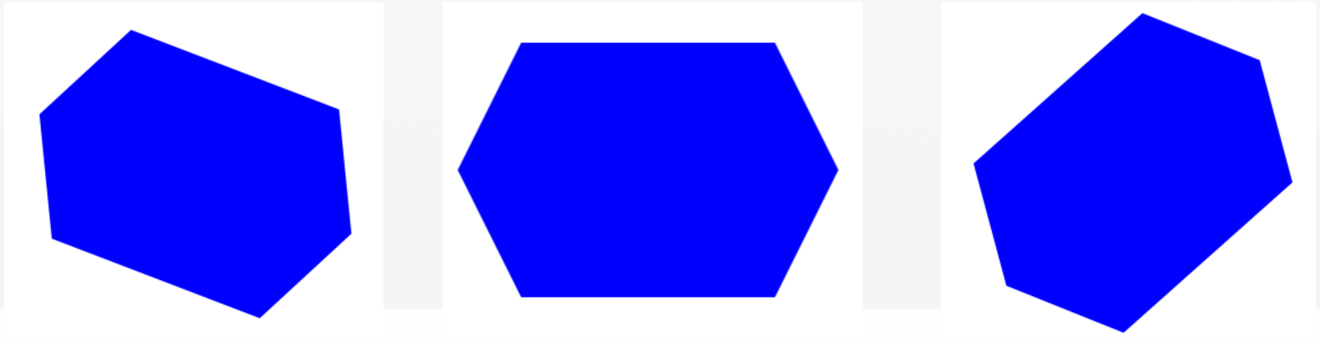


# Transitions

## RotateTransition

**Rotate transition** provides animation for rotating an object.

We can provide up to what angle the node should rotate by **toAngle**. Using **byAngle** we can specify how much it should rotate from current angle of rotation.



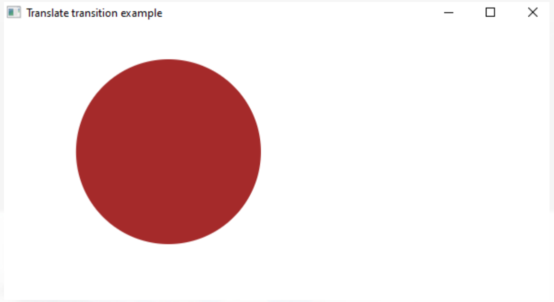
# Transitions

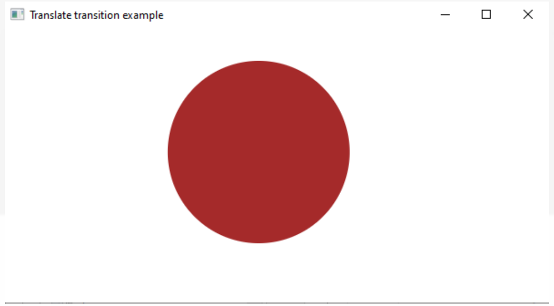
## Translate Transition

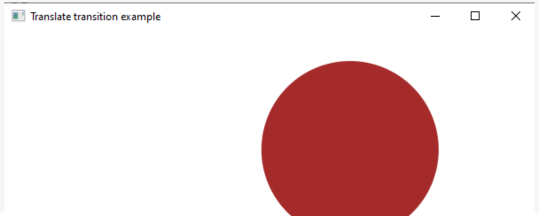
**Translate transition** allows to create movement animation from one point to another within a duration.

Using **TranslateTransition#setByX / TranslateTransition#setByY**, you can set how much it should move in x and y axis respectively.

It also possible to set precise destination by using **TranslateTransition#setToX / TranslateTransition#setToY**.





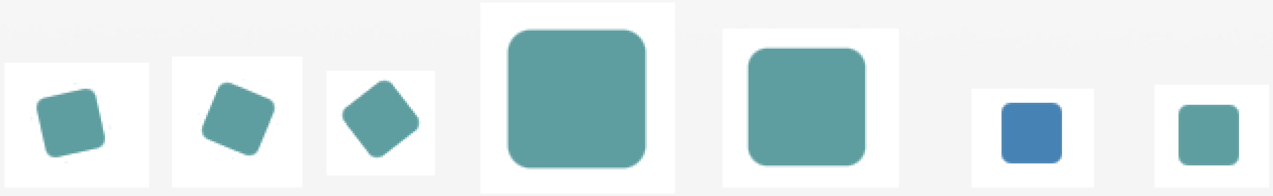


## Transitions

### Sequential Transition

A **sequential transition** executes several transitions one after another.

The following example shows the rotate, scale, and fill transitions applied to a rectangle.

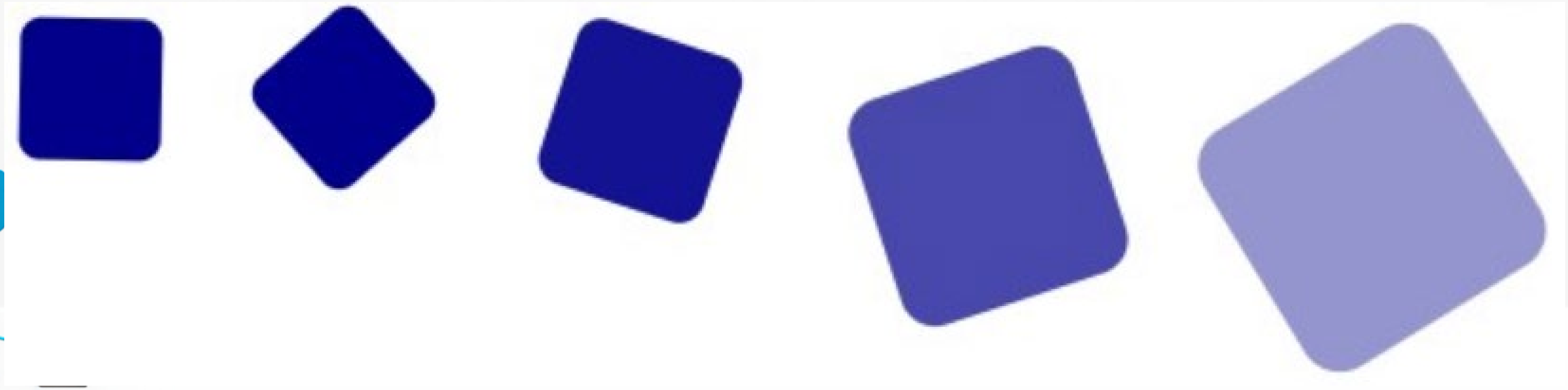


# Transitions

## Parallel Transition

A **parallel transition** executes several transitions simultaneously.

The following example shows the fade, translate, rotate, and scale transitions applied to a rectangle.



## Timeline Animation

A **Timeline** can be used to define a free form animation of any Writable Value

The **Timeline** class can be used to program any animation using one or more **KeyFrames** which contain the properties of nodes that change. These properties are encapsulated in KeyValues.

Each **KeyFrame** is executed sequentially at a specified time interval.

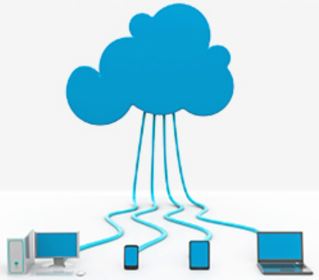
Timeline inherits from Animation.

You can construct a Timeline using the constructor

*new Timeline (KeyFrame...keyframes)*

A KeyFrame can be constructed using: *new KeyFrame (Duration duration, EventHandler<ActionEvent> onFinished)*

The handler **onFinished** is called when the duration for the key frame is elapsed.



## Timeline Animation

**Timeline** inherits from Animation.

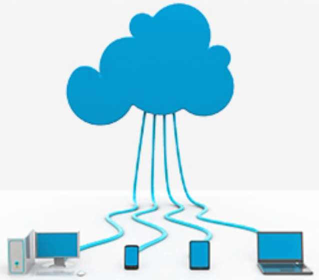
A **Timeline** can be used to define a free form animation of any Writable Value

The **Timeline** class can be used to program any animation using one or more **KeyFrames** which contain the properties of nodes that change. These properties are encapsulated in **KeyValues**.

**KeyFrame** defines target values at a specified point in time for a set of variables that are interpolated along a Timeline.

Each **KeyFrame** is executed sequentially at a specified time interval.

Timeline inherits from Animation.



## Timeline Animation

You can construct a Timeline using the constructor

*new Timeline (KeyFrame...keyframes)*

A **KeyFrame** can be constructed using: *new KeyFrame (**Duration** duration, EventHandler<ActionEvent> onFinished)*

The handler **onFinished** is called when the duration for the key frame is elapsed.