R2S
Resource Software Solution

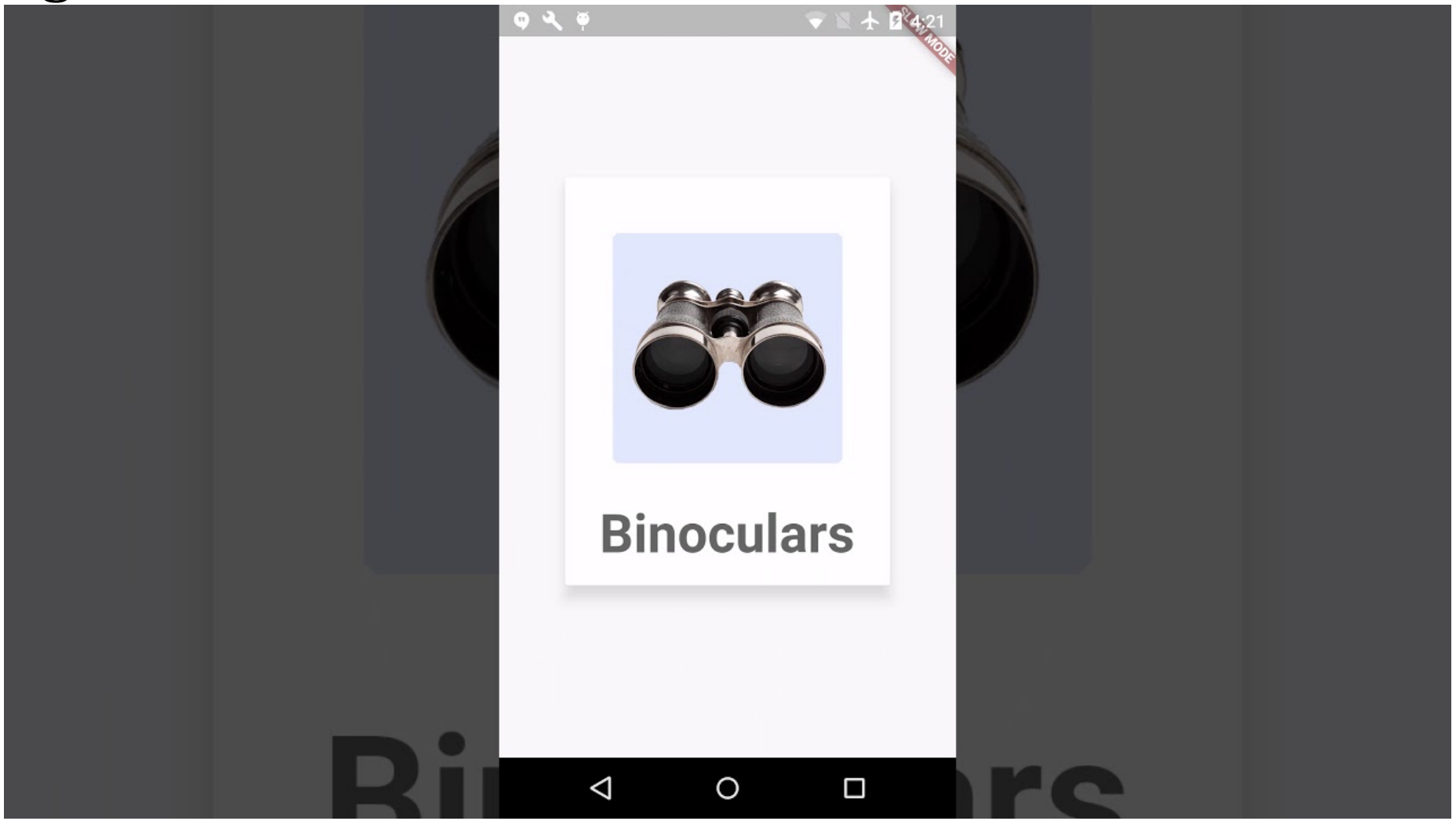
Animations

Contents

- Introduction to animations
- Types of Animations

Introduction

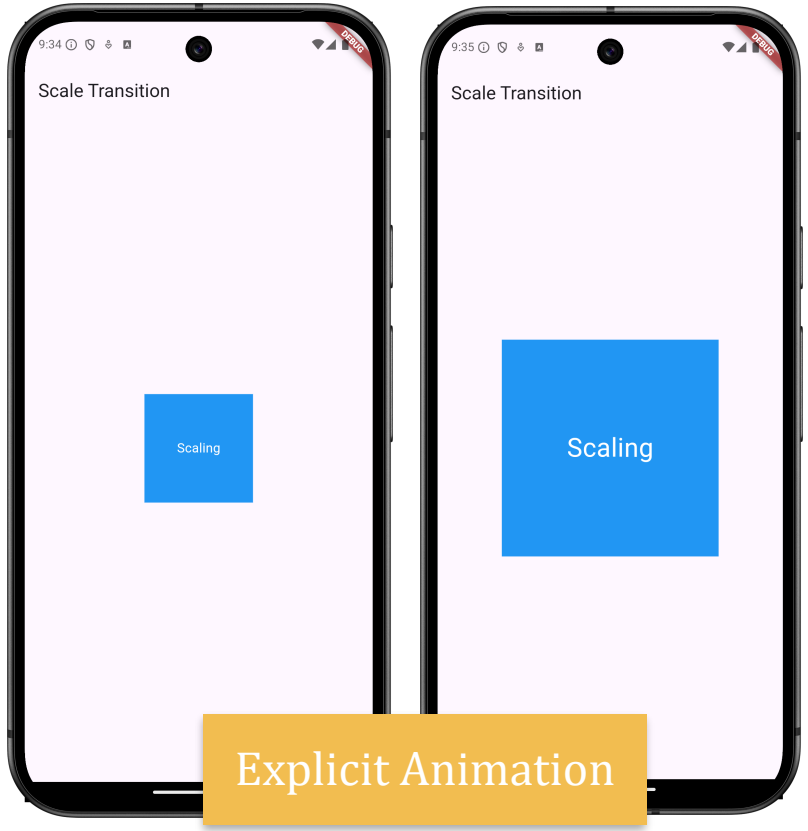
- Well-designed animations make a **UI feel more intuitive**



Types of Animations

- **Implicit** Animations: It automatically animates changes to its properties like **width, height, color, padding**, **opacity**, etc., over a given **duration**.
- **Explicit** Animations are useful for more complex animations where fine control over the animation sequence is required.

Implicit Animation



Explicit Animation

Implicit Animations (1)

- Here are some of the most commonly used **implicit animation widgets**

1. AnimatedContainer

Animates changes to properties like **width**, **height**, **color**, **padding**, **alignment**, **decoration**, etc.

```
// Example: Changing the size and color of a container over time.
AnimatedContainer(
  duration: Duration(seconds: 1),
  width: _width,
  height: _height,
  color: _color,
  child: ...
);
```

Implicit Animations (2)

- Here are some of the most commonly used **implicit animation widgets**

2. AnimatedOpacity — Animates the transition of a widget's opacity (fades in or fades out)

```
// Example: Fading a widget in and out by changing its opacity.
AnimatedOpacity(
  opacity: _opacity,
  duration: Duration(seconds: 1),
  child: ...
);
```

Implicit Animations (3)

- Here are some of the most commonly used **implicit animation widgets**

3. AnimatedAlign

Animates the alignment of a widget inside its parent.

```
// Example: Moving a widget from one part of the screen to another smoothly
AnimatedAlign(
  alignment: _alignment,
  duration: Duration(seconds: 1),
  child: ...
);
```

Implicit Animations (4)

- Here are some of the most commonly used **implicit animation widgets**

4. AnimatedPositioned — Used inside a **Stack** to animate changes in the position of a widget.

```
// Example: Changing the top, left, right, or bottom properties to smoothly move a widget.
Stack(
  children: [
    AnimatedPositioned(
      top: _top,
      left: _left,
      duration: Duration(seconds: 1),
      child: ...
    )
  ]
);
```

Implicit Animations (5)

- Here are some of the most commonly used **implicit animation widgets**

5. AnimatedTheme Animates changes to the app's theme.

```
// Example: Smoothly transitioning between light and dark themes.
AnimatedTheme(
  data: _themeData,
  duration: Duration(seconds: 1),
  child: ...
);
```

Implicit Animations (6)

- Here are some of the most commonly used **implicit animation widgets**

6. AnimatedListState — A list that animates the addition and removal of items.

```
AnimatedList(
  key: _listKey,
  initialItemCount: _items.length,
  itemBuilder: (context, index, animation) {
    return FadeTransition(
        opacity: animation,
        child: _buildItem(index)
    );
  },
);
```

AnimatedPadding, AnimatedCrossFade, AnimatedDefaultTextStyle, AnimatedSwitcher, AnimatedSize, AnimatedIcon, AnimatedRotation, AnimatedScale, AnimatedContainer, AnimatedPhysicalModel

Explicit Animations (1)

- Core Components of Explicit Animations:

1. AnimationController

Central to explicit animations, the AnimationController is used to manage the animation's duration and state (start, stop, reverse, etc.).

```
AnimationController _controller = AnimationController(
  vsync: this,
  duration: const Duration(seconds: 2),
);
```

2. Tween<T>

Defines a **range of values** for the animation to interpolate between.

```
Tween<double>(begin: 0.0, end: 100.0).animate(_controller);
```

Explicit Animations (2)

- Core Components of Explicit Animations:

3. CurvedAnimation

Applies easing curves to an animation, such as linear, ease-in, ease-out, etc.

```
CurvedAnimation(
  parent: _controller,
  curve: Curves.easeIn,
);
```

- Types of Easing Curves:
  1. **EaseIn**: Starts slowly and then accelerates towards the end. This is useful for animations where you want to mimic the acceleration of an object.
  2. **EaseOut**: Starts quickly and then decelerates towards the end.
  3. **EaseInOut**: Combines both **easeIn** and **easeOut,** starting slowly, accelerating in the middle, and then decelerating towards the end.
  4. **Bounce**: Causes the animation to bounce at the end, simulating a bouncy effect.
  5. etc

Explicit Animations (3)

- Core Components of Explicit Animations:

4. Animation<T>

Represents the value of the animation at any given time, which is updated by the **AnimationController**.

```
Animation<double> _animation = Tween<double>(begin: 0.0, end: 100.0).animate(_controller);
```

5. AnimatedBuilder

A widget that listens to an animation and rebuilds only the parts of the widget tree that depend on the animation.

```
AnimatedBuilder(
 animation: _animation,
 builder: (context, child) {
  return Container(
   width: _animation.value,
   height: _animation.value,
   color: Colors.blue,
  );
 },
);
```

Explicit Animations (4)

- Advanced Explicit Animation Widgets and Techniques:

PageRouteBuilder

Custom animations for screen transitions using PageRouteBuilder with an AnimationController.

```
Navigator.push(
  context,
  PageRouteBuilder(
    pageBuilder: (context, animation, secondaryAnimation) => SecondPage(),
    transitionsBuilder: (context, animation, secondaryAnimation, child) {
      return FadeTransition(
        opacity: animation,
        child: child,
      );
    },
  ),
);
```

Explicit Animations (5)

- Advanced Explicit Animation Widgets and Techniques:

FadeTransition Animates the opacity of a widget.

```
FadeTransition(
  opacity: _animation, // Animation<double>
  child: _child,
);
```

ScaleTransition Animates the scaling of a widget

```
ScaleTransition(
  scale: _animation, // Animation<double>
  child: _child,
);
```

Explicit Animations (6)

- Advanced Explicit Animation Widgets and Techniques:

RotationTransition Animates the rotation of a widget.

```
RotationTransition(
  turns: _animation, // Animation<double>
  child: _child,
);
```

SlideTransition Animates the position of a widget

```
SlideTransition(
  position: _animation, // Animation<Offset>
  child: _child,
);
```

SlideTransition (1)

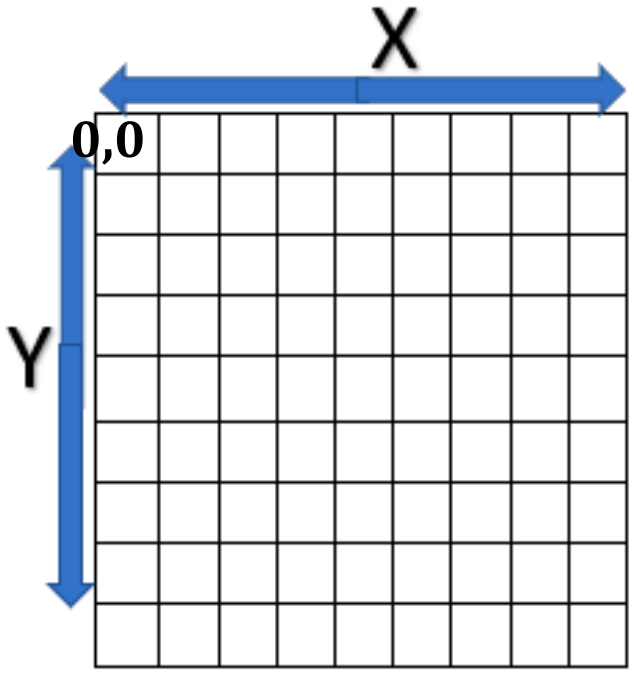
- Animates the **position of a widget** relative to its **normal position (0, 0)**.

```
return PageRouteBuilder(
   pageBuilder: (context, animation, secondaryAnimation) => newPage,
   transitionsBuilder: (context, animation, secondaryAnimation, child) {
     const begin = Offset(1.0, 0.0);
     const end = Offset(0.0, 0.0);
     const curve = Curves.easeInOut;
     var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));

     return SlideTransition(
       position: animation.drive(tween),
       child: child, // This is the newPage widget being transitioned
     );
   },
 );
```

SlideTransition (2)

- **Offset** class: Simply it is a **data class to store X and Y coordinates** and pass that class data to other classes or functions.

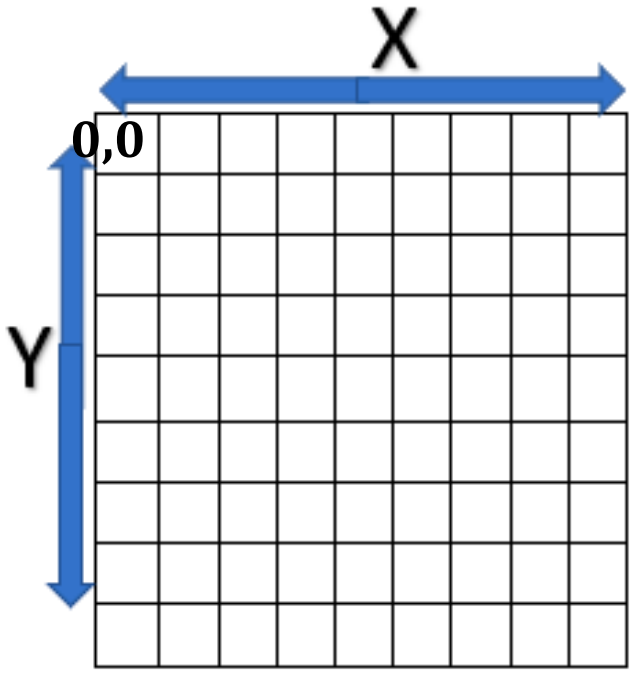



```
//The animation starts from the right.
const begin = Offset(1.0, 0.0);

//The animation ends at the original position.
const end = Offset(0.0, 0.0);
```

SlideTransition (3)

- **Offset** class: Simply it is a **data class to store X and Y coordinates** and pass that class data to other classes or functions.



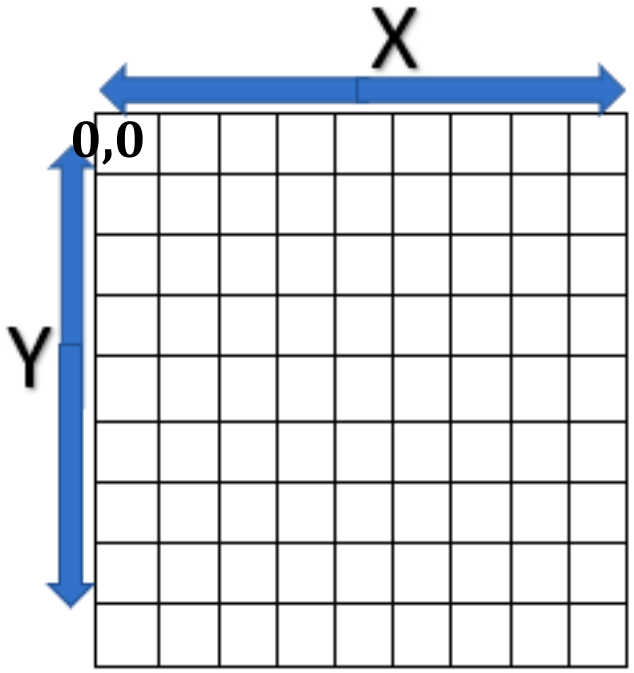
//The animation starts from the bottom

```
//The animation starts from the bottom
const begin = Offset(0.0, 1.0);

//The animation ends at the original position.
const end = Offset(0.0, 0.0);
```

SlideTransition (4)

- **Offset** class: Simply it is a **data class to store X and Y coordinates** and pass that class data to other classes or functions.



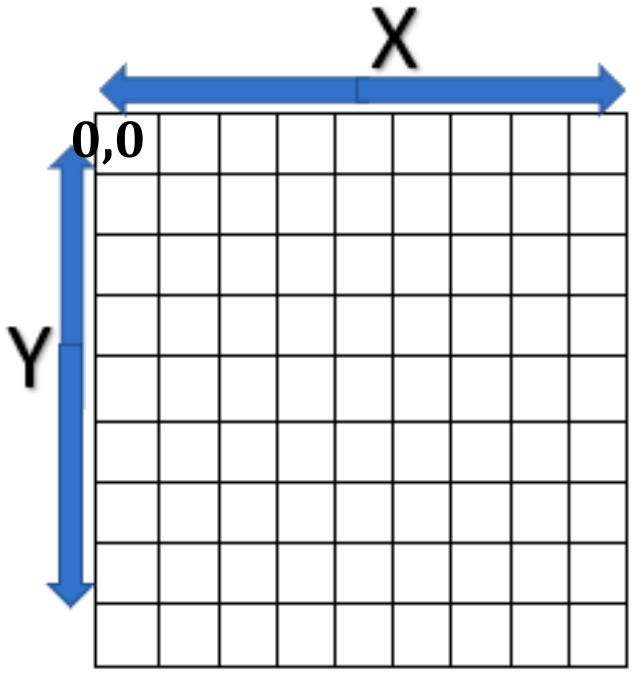
//The animation starts from the left

```
//The animation starts from the left
const begin = Offset(-1.0, 0.0);

//The animation ends at the original position.
const end = Offset(0.0, 0.0);
```

SlideTransition (5)

- **Offset** class: Simply it is a **data class to store X and Y coordinates** and pass that class data to other classes or functions.



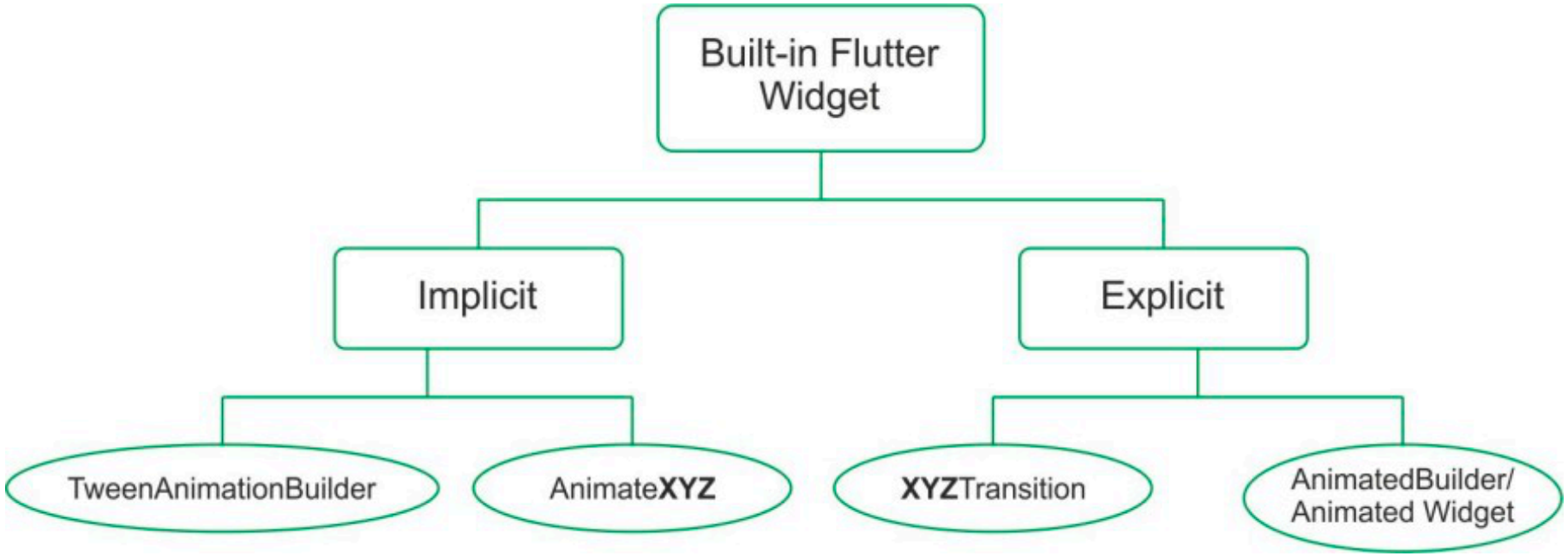
//The animation starts from the top

```
//The animation starts from the top
const begin = Offset(0.0, -1.0);

//The animation ends at the original position.
const end = Offset(0.0, 0.0);
```

Summary

- Implicit animations use **AnimatedWidgets** to **automatically** animate when a value is changed and a rebuild occurs (i.e. via *setState*).
- With explicit animations, you **can control** the direction of the animation, repeat it, and stop it at any point in time.



*Keeping up those **inspiration** and the **enthusiasm** in the **learning path**.*
*Let confidence to bring it into **your career path** for getting gain the **success** as your expectation.*

Thank you

Contact

- **Name: R2S Academy**
- **Email: daotao@r2s.edu.vn**
- **Phone/Zalo: 0919 365 363**
- **FB: https://www.facebook.com/r2s.tuyendung**
- **Website: www.r2s.edu.vn**

Questions and Answers