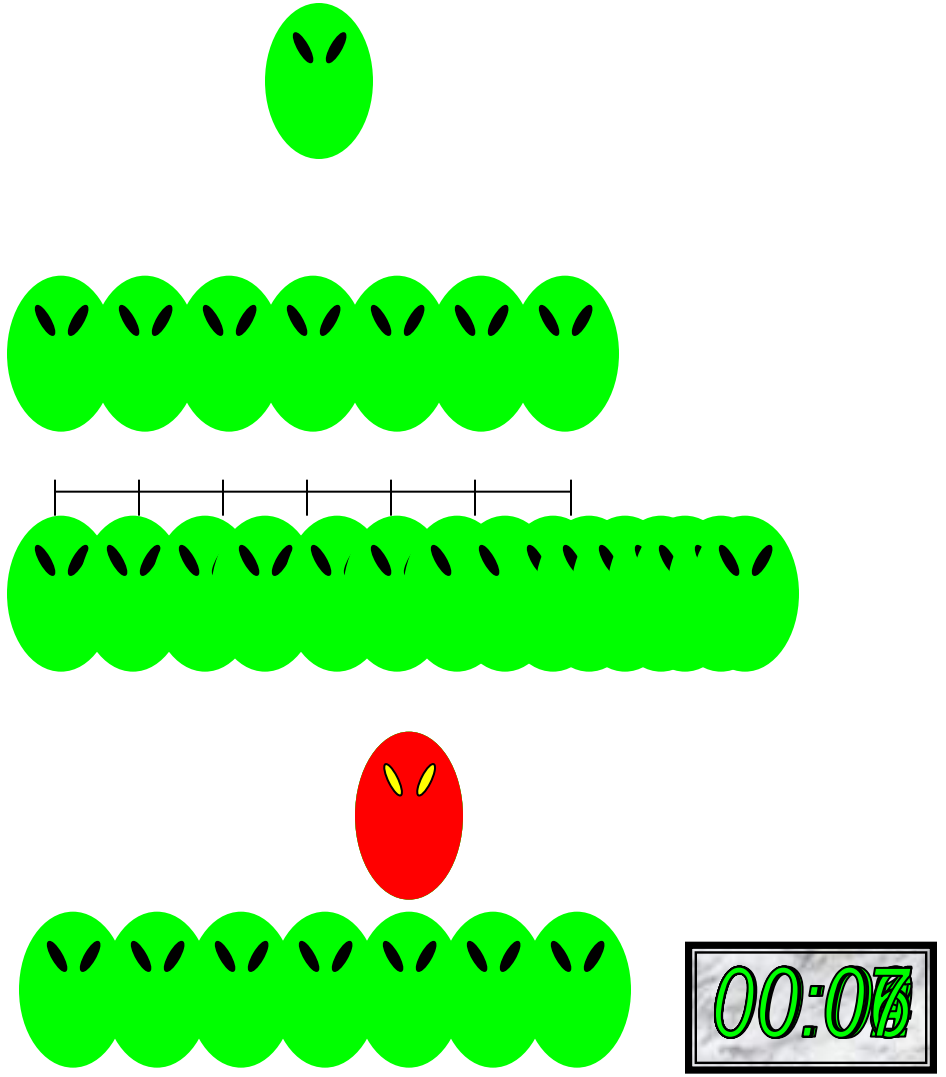
CSE201 Tutorial-6

JavaFX Animation

Animation – Change Over Time

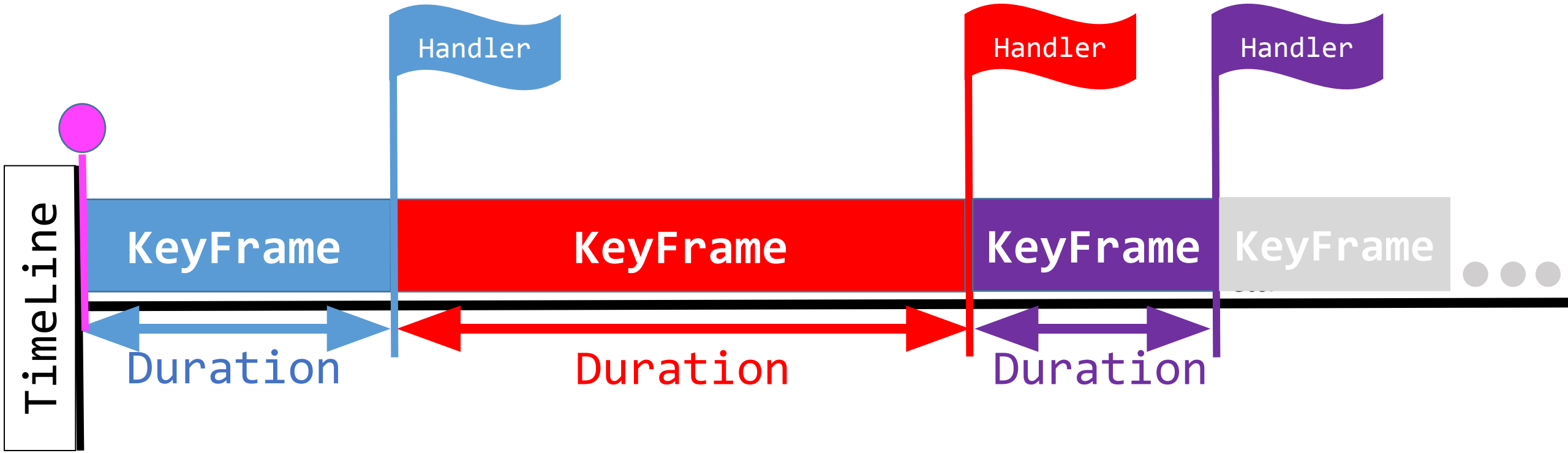
- Suppose we have an alien `Shape` we would like to **animate** (e.g. make it move across the screen)
- As in film and video animation, we can create **apparent motion** with many small changes in position
- If we move **fast enough** and in **small enough increments**, we get **smooth motion**
- Same goes for size, orientation, shape change, etc…
- How to orchestrate a sequence of incremental changes?
  - coordinate with a `Timeline` where change happens at defined instants



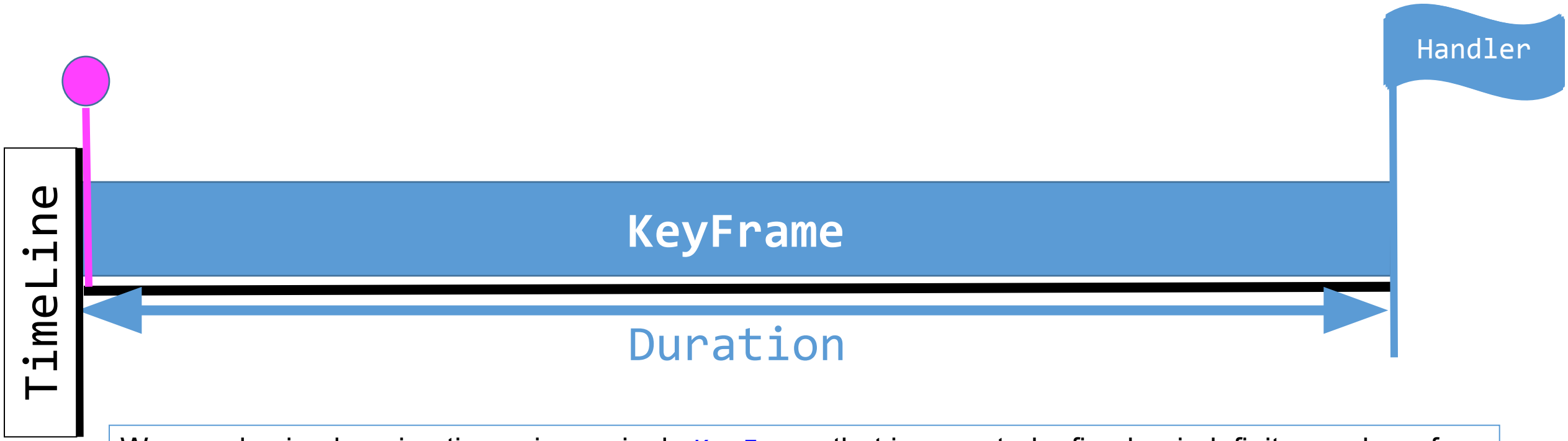
Introducing `Timelines` (1/3)

- The Timeline sequences one or more KeyFrames
  - each KeyFrame lasts for its entire Duration without making any changes
  - when the Duration ends, the EventHandler updates variables to affect the animation

Introducing Timelines (2/3)



Introducing Timelines (3/3)



We can do simple animation using a single KeyFrame that is repeated a fixed or indefinite number of times EventHandler is called, it makes incremental changes to time-varying variables (e.g., (x, y) position of a shape)

Using JavaFX `Timelines` (1/2)

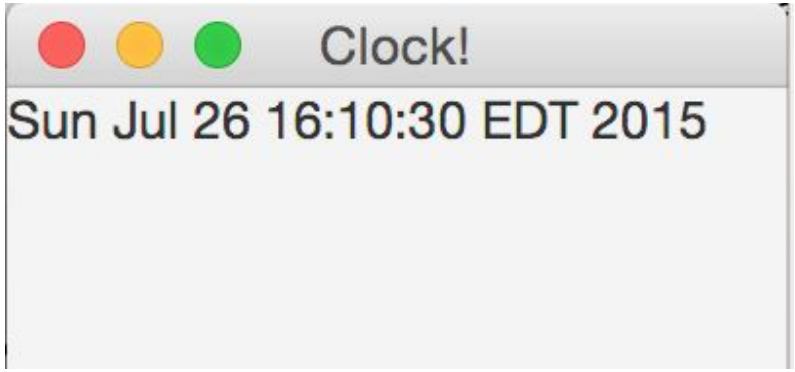
- `javafx.animation.Timeline` is used to sequence one or more `javafx.animation.KeyFrame`s, and optionally to run through them cyclically
  - each KeyFrame lasts for its entire duration without making any changes, until its time interval ends and EventHandler is called to make updates
- When we instantiate a `KeyFrame`, we pass it
  - a `Duration` (e.g. `Duration.seconds(0.3)` or `Duration.millis(300)`), which defines time that each `KeyFrame` lasts
  - an `EventHandler` that defines what should occur upon completion of each `KeyFrame`
- `KeyFrame` and `Timeline` work together to control the animation, but our application's `EventHandler` is the method that actually causes variables to change

Using JavaFX `Timelines` (2/2)

- We then pass our new `KeyFrame` into `Timeline`
- After we instantiate our `Timeline`, we must set its `CycleCount` property
  - this defines number of cycles in `Animation`
  - we will set cycle count to `Animation.INDEFINITE`, which will let `Timeline` run forever or until we explicitly stop it
- In order for `Timeline` to work, we must then call `Timeline.play();`

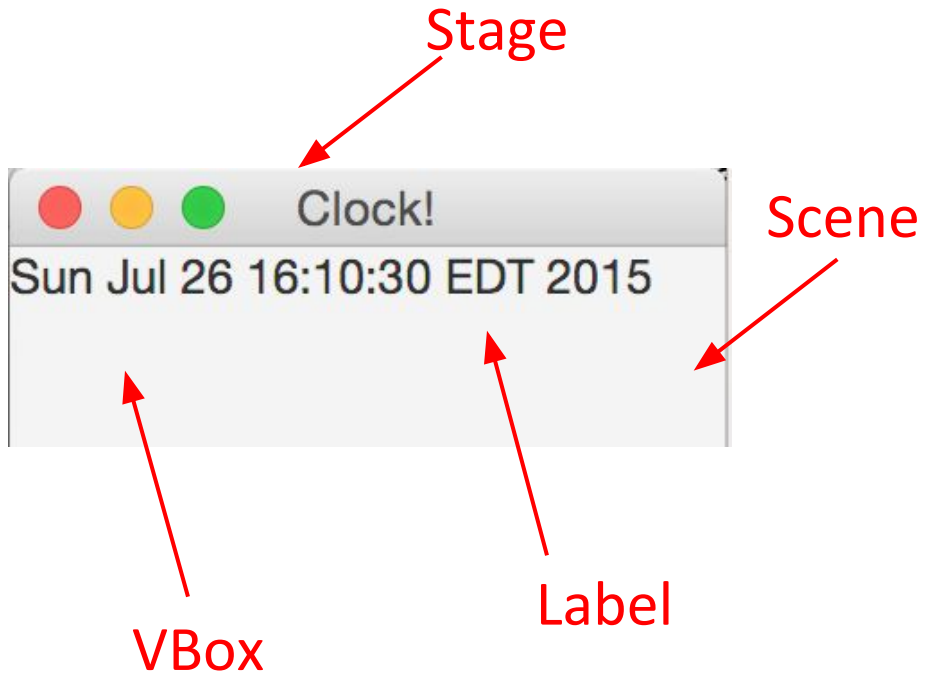
Another JavaFX App: `Clock`

- Simple example of discrete (non-smooth) animation
- Specifications: App should display current date and time, updating every second
- Useful classes:
  - `java.util.Date`
  - `javafx.util.Duration`
  - `javafx.animation.KeyFrame`
  - `javafx.animation.Timeline`



Process: `Clock`

1. **Write `App` class that extends `javafx.application.Application` and implements `start (Stage)`**
2. Write a `PaneOrganizer` class that instantiates root node and returns it in a public `getRoot()` method. Instantiate a `Label` and add it as root node's child. Factor out code for `Timeline` into its own method.
3. In our own `setupTimeline(),` instantiate a `KeyFrame` passing in a `Duration` and an instance of `TimeHandler` (defined later). Then instantiate `Timeline`, passing in our `KeyFrame`, and play `Timeline.`
4. Write private inner `TimeHandler` class that implements `EventHandler`-- it should know about a `Label` and update its text on every `ActionEvent`



Clock: App class (1/3)

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
    PaneOrganizer organizer = new PaneOrganizer();




    }

}
```

Clock: App class (2/3)

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

1b. Instantiate a Scene, passing in organizer.getRoot(), and desired width and height of Scene

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
    PaneOrganizer organizer = new PaneOrganizer();
    Scene scene =
        new Scene(organizer.getRoot(), 200, 200);



    }

}
```

Clock: App class (3/3)

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

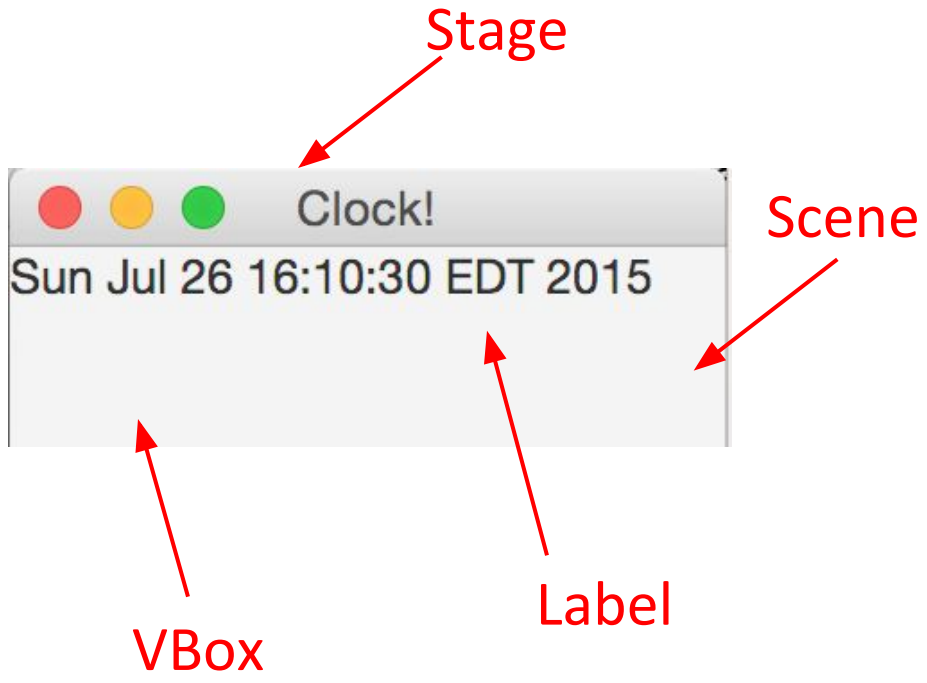
1b. Instantiate a Scene, passing in organizer.getRoot(), desired width and height of the Scene

1c. Set the Scene, set the Stage's title, and show the Stage!

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
    PaneOrganizer organizer = new PaneOrganizer();
    Scene scene =
        new Scene(organizer.getRoot(), 200, 200);

    stage.setScene(scene);
    stage.setTitle("Clock!");
    stage.show();
    }

}
```

Process: `Clock`

1. Write `App` class that extends `javafx.application.Application` and implements `start(Stage)`
2. **Write a `PaneOrganizer` class that instantiates root node and returns it in a public `getRoot()` method. Instantiate a `Label` and add it as root node's child. Factor out code for `Timeline` into its own method**
3. In our own `setupTimeline()`, instantiate a `KeyFrame` passing in a `Duration` and an instance of `TimeHandler` (defined later). Then instantiate a `Timeline`, passing in our `KeyFrame`, and play the `Timeline`
4. Write a private inner `TimeHandler` class that implements `EventHandler`-- it should know about a `Label` and update its text on every `ActionEvent`



Clock: PaneOrganizer Class (1/3)

2a. In the PaneOrganizer class' constructor, instantiate a root VBox and set it as the return value of a public getRoot() method

```
public class PaneOrganizer{
    private VBox _root;

    public PaneOrganizer(){
        _root = new Vbox();

    }

    public VBox getRoot() {
        return _root;
    }

}
```

Clock: PaneOrganizer Class (2/3)

2a. In the PaneOrganizer class' constructor, instantiate a root VBox and set it as the return value of a public getRoot() method

2b. Instantiate a Label and add it to the list of the root node's children

```
public class PaneOrganizer{
    private VBox _root;
    private Label _label;

    public PaneOrganizer(){
        _root = new Vbox();
        _label = new Label();
        _root.getChildren().add(_label);


    }

    public VBox getRoot() {
        return _root;
    }

}
```

Clock: PaneOrganizer Class (3/3)

2a. In the PaneOrganizer class' constructor, instantiate a root VBox and set it as the return value of a public getRoot() method

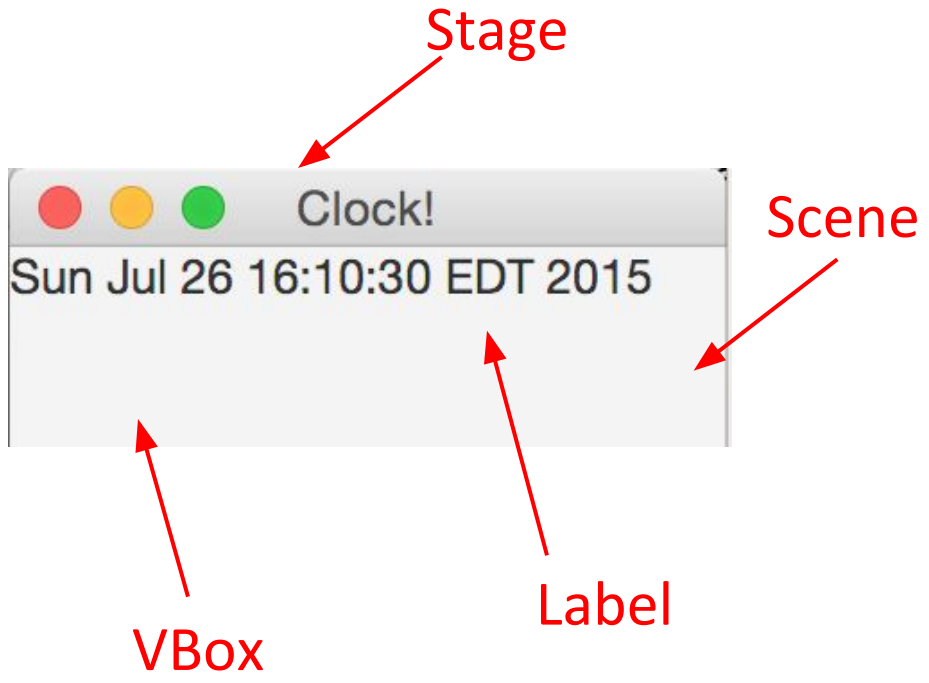
2b. Instantiate a Label and add it to the list of the root node's children

2c. Call setupTimeline(); we'll define this method next !

```
public class PaneOrganizer{
    private VBox _root;
    private Label _label;

    public PaneOrganizer(){
        _root = new Vbox();
        _label = new Label();
        _root.getChildren().add(_label);

        this.setupTimeline();
    }

    public VBox getRoot() {
        return _root;
    }

}
```

Process: `Clock`

1. Write an `App` class that extends `javafx.application.Application` and implements `start(Stage)`
2. Write a `PaneOrganizer` class that instantiates the root node and returns it in a public `getRoot()` method. Instantiate a `Label` and add it as the root node's child. Factor out code for `Timeline` into its own method.
3. **In `setupTimeline()`, instantiate a `KeyFrame`, passing in a `Duration` and an instance of `TimeHandler` (defined later). Then instantiate a `Timeline`, passing in our `KeyFrame`, and play the `Timeline`.**
4. Write a private inner `TimeHandler` class that implements `EventHandler` - it should know about a `Label` and update its text on every `ActionEvent`



Clock: PaneOrganizer class- setupTimeline() (1/4)

Within setupTimeline():

3a. Instantiate a KeyFrame, which takes two parameters

- want to update text of _label each second - therefore make Duration of the KeyFrame 1 second
- for the EventHandler parameter pass an instance of our TimeHandler class, to be created later

```
public class PaneOrganizer{
    //other code elided

    public void setupTimeline(){
        KeyFrame kf = new KeyFrame(
          Duration.seconds(1), //how long
          new TimeHandler()); //event handle




    }

}
```

Note: the JavaFX KeyFrame calls TimeHandler's handle() method, which changes the label text before the next 1 second cycle starts. `

Clock: PaneOrganizer class- setupTimeline() (2/4)

Within `setupTimeline()`:

3a. Instantiate a `KeyFrame`

3b. Instantiate a `Timeline`, passing in our new `KeyFrame`

```
public class PaneOrganizer{
    //other code elided

    public void setupTimeline(){
        KeyFrame kf = new KeyFrame(
          Duration.seconds(1),
          new TimeHandler());

     Timeline timeline = new Timeline(kf);



    }

}
```

Clock: PaneOrganizer class- setupTimeline() (3/4)

Within `setupTimeline()`:

3a. Instantiate a `KeyFrame`

3b. Instantiate a `Timeline`, passing in our new `KeyFrame`

3c. Set `CycleCount` to `INDEFINITE`

```
public class PaneOrganizer{
    //other code elided

    public void setupTimeline(){
        KeyFrame kf = new KeyFrame(
          Duration.seconds(1),
          new TimeHandler());

        Timeline timeline = new Timeline(kf);

     timeline.setCycleCount(
           Animation.INDEFINITE);

    }

}
```

Clock: PaneOrganizer class- setupTimeline() (4/4)

Within `setupTimeline()`:

3a. Instantiate a `KeyFrame`

3b. Instantiate a `Timeline`, passing in our new `KeyFrame`

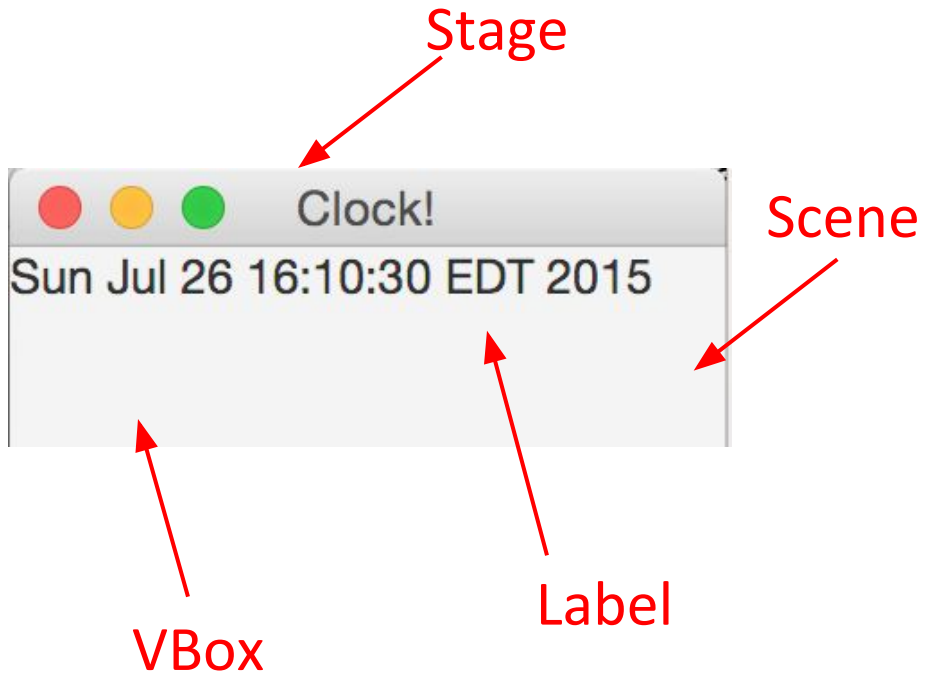
3c. Set `CycleCount` to `INDEFINITE`

3d. Play, i.e. start `Timeline`

```
public class PaneOrganizer{
    //other code elided

    public void setupTimeline(){
        KeyFrame kf = new KeyFrame(
          Duration.seconds(1),
          new TimeHandler());

        Timeline timeline = new Timeline(kf);

     timeline.setCycleCount(
           Animation.INDEFINITE);
     timeline.play();
    }

}
```

Process: `Clock`

1. Write an `App` class that extends `javafx.application.Application` and implements `start(Stage)`
2. Write a `PaneOrganizer` class that instantiates the root `Node` and returns it in public `getRoot()` method. Instantiate a `Label` and add it as root `node`'s child. Factor out code for `Timeline` into its own method.
3. In `setupTimeline()`, instantiate a `KeyFrame` passing in a `Duration` and an instance of `TimeHandler` (defined later). Then instantiate a `Timeline`, passing in our `KeyFrame`, and play the `Timeline`.
4. **Write a private inner `TimeHandler` class that implements `EventHandler` – it should know about a `Label` and update its text on every `ActionEvent`**



Clock: TimeHandler Private Inner Class (1/3)

4a. The last step is to create our TimeHandler and implement handle(), specifying what to occur **at the end** of each KeyFrame – called automatically by JFX

```
public class PaneOrganizer{
    //other code elided

    private class TimeHandler implements
EventHandler<ActionEvent>{

        public void handle(ActionEvent event){



        }

    } //end of private TimeHandler class

} //end of PaneOrganizer class
```

Clock: TimeHandler Private Inner Class (2/3)

4a. The last step is to create our TimeHandler and implement handle(), specifying what to occur **at the end** of each KeyFrame – called automatically by JFX

4b. java.util.Date represents a specific instant in time. Date is a representation of the time, to the nearest millisecond, at the moment the Date is instantiated

```
public class PaneOrganizer{
    //other code elided

    private class TimeHandler implements
EventHandler<ActionEvent>{

        public void handle(ActionEvent event){
        Date now = new Date();



        }

    } //end of private TimeHandler class

} //end of PaneOrganizer class
```

Clock: TimeHandler Private Inner Class (3/3)

4a. The last step is to create our TimeHandler and implement handle(), specifying what to occur **at** **the end** of each KeyFrame – called automatically by JFX

4b. java.util.Date represents a specific instant in time. Date is a representation of the time, to the nearest millisecond, at the moment the Date is instantiated

4c. Because our Timeline has a Duration of 1 second, each second a new Date will be generated, converted to a String, and set as the _label's text. This will appropriately update the Label with correct time every second!

```
public class PaneOrganizer{
    //other code elided

    private class TimeHandler implements
EventHandler<ActionEvent>{

        public void handle(ActionEvent event){
        Date now = new Date();
        //toString converts the Date into a
        //String with year, day, time etc.
_label.setText(now.toString());
        }

    } //end of private TimeHandler class

} //end of PaneOrganizer class
```

The Whole App: Clock

```
//App class imports
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.application.*;
// package includes Pane class and its subclasses
import javafx.scene.layout.*;
//package includes Label, Button classes
import javafx.scene.control.*;
//package includes ActionEvent, EventHandler classes
import javafx.event.*;
import javafx.util.Duration;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.Timeline;
import java.util.Date;
```

```
public class App extends Application {

    @Override
    public void start(Stage stage) {
    PaneOrganizer organizer = new PaneOrganizer();
    Scene scene = new Scene(organizer.getRoot(), 200, 200);

    stage.setScene(scene);
    stage.setTitle("Clock");
    stage.show();
    }

}
```

```
public class PaneOrganizer{
    private VBox _root;
    private Label _label;

    public PaneOrganizer(){
        _root = new VBox();
        _label = new Label();
        _root.getChildren().add(_label);
        this.setupTimeline();
    }

    public VBox getRoot() {
        return _root;
    }

    public void setupTimeline(){
        KeyFrame kf = new KeyFrame(Duration.seconds(1),
         new TimeHandler());
        Timeline timeline = new Timeline(kf);
            timeline.setCycleCount(Animation.INDEFINITE);
        timeline.play();
    }

    private class TimeHandler
    implements EventHandler<ActionEvent>{
        public void handle(ActionEvent event){
                Date now = new Date();
                _label.setText(now.toString());
        }
    }

}
```