GUIs: JavaFX (II)

Lecture 9 (26 April 2022)

Handling the mouse

mouse events

Mouse events are fired whenever a mouse button is

- pressed
- released
- clicked
- moved
- dragged

on a node or a scene, and when the cursor enters / exits a node.

Read the text on

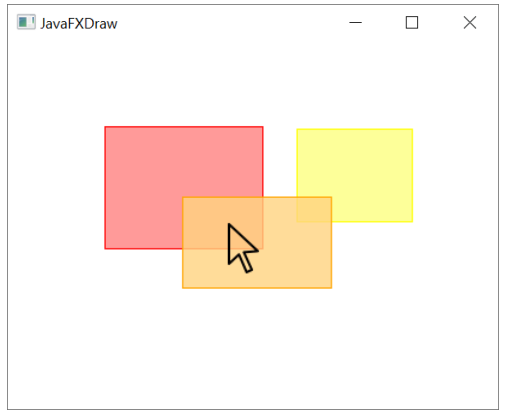
`https://docs.oracle.com/javase/8/javafx/api/javafx/scene/input/MouseEvent.html`

for details on mouse events.

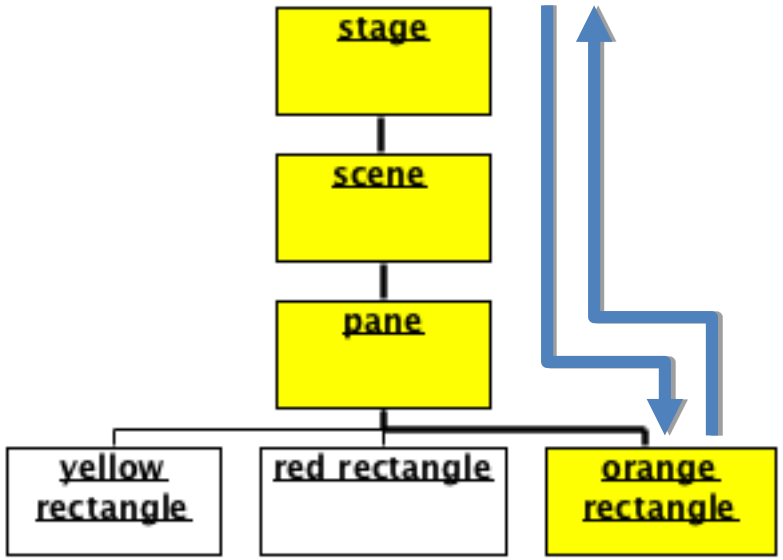
mouse handling

- There may be several nodes where the mouse pointer is located
  - each of them can have a mouse handler
- Who is handling the mouse event (first)?
- Target of event is selected when mouse button is pressed
- all subsequent events are delivered to the same target
  - until the button is released
- mouse pointer's location:
  - x, y: relative to the origin (0, 0) of the MouseEvent's node
  - `sceneX`, `sceneY`: relative to origin of the Scene that contains the node
  - `screenX`, `screenY`: relative to origin of screen that contains the mouse

JavaFX event delivery (“dispatching”)



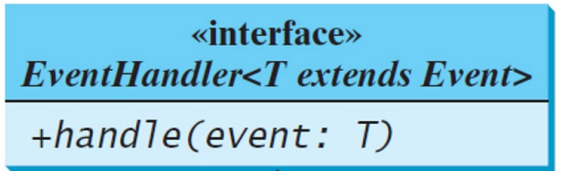
1. target selection
   - the target is the node at the location of the mouse
   - by default only the top node at this level receives the event
2. route construction (EventDispatchChain)
   - path from stage to node
3. event capturing
   - pass event top-down, apply filters on the path; filters can stop or redirect the event handling
4. event bubbling
   - invoke handlers bottom-up until it is consumed



mouse event handlers

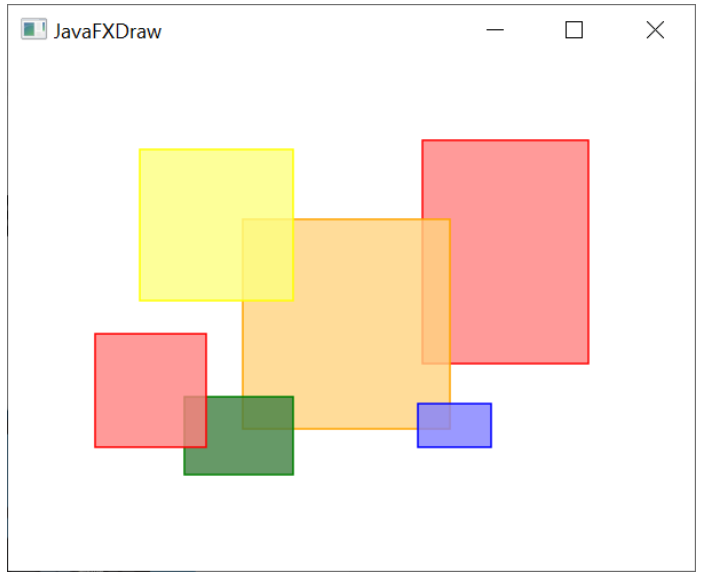
- Each component on the route can have its own handler(s)
- Without handler calling `event.consume()` the event is just passed further up
- There can be handlers for many gestures:
  - `setOnMouseClicked`, `setOnMousePressed`, `setOnMouseReleased`, `setOnMouseDragged`, `setOnMouseDragEntered`, `setOnMouseDragExited`, `setOnMouseDragOver`, `setOnMouseDragReleased`, `setOnMouseEntered`, `setOnMouseExited`, `setOnMouseMoved`
  - these setters associate a single handler to their event type
- can register multiple handlers to the same event

```
addEventHandler(EventType, EventHandler)
removeEventHandler(EventType, EventHandler)
```



mouse handling example: drawing rectangles

```
public class JavaFXDraw extends Application {
  private static final double MIN_SIZE = 5;
  private static final Color[] colors  {Color.RED,Color.ORANGE,Color.YELLOW,Color.GREEN,Color.BLUE};
  private int nextColorIx;

  private Rectangle currentRect;
  private double currentXOffset, currentYOffset;

  @Override
  public void start(Stage stage) {
    Pane pane = new Pane();
    pane.setOnMousePressed(e -> newRect(pane, e));
    pane.setOnMouseDragged(e -> resizeRect(e));
    pane.setOnMouseReleased(e -> finishRect(e));
    Scene scene = new Scene(pane, 400, 300);
    stage.setTitle(this.getClass().getSimpleName());
    stage.setScene(scene);
    stage.show();
  }
```



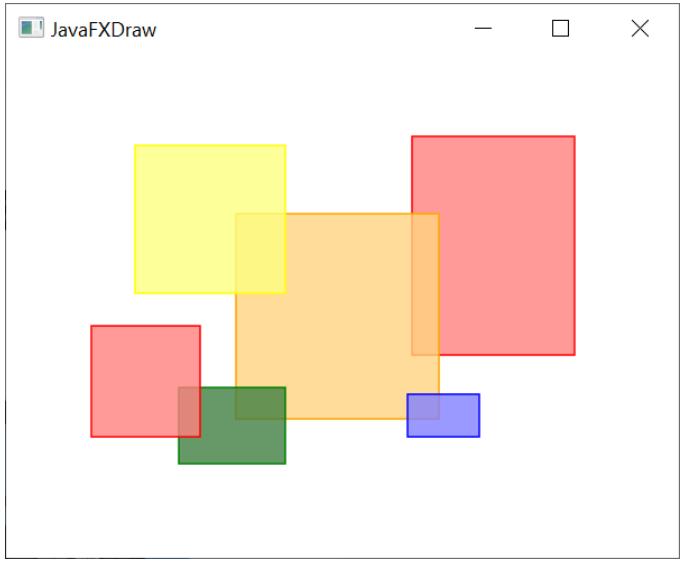
fields for rectangle creation/movement: fields for temporary storing new rectangle, and offset for dragging around

mouse handling example: creating the rectangle

- make a new rectangle at the position of the mouse
- field `currentRect` is used as a reference to this new object

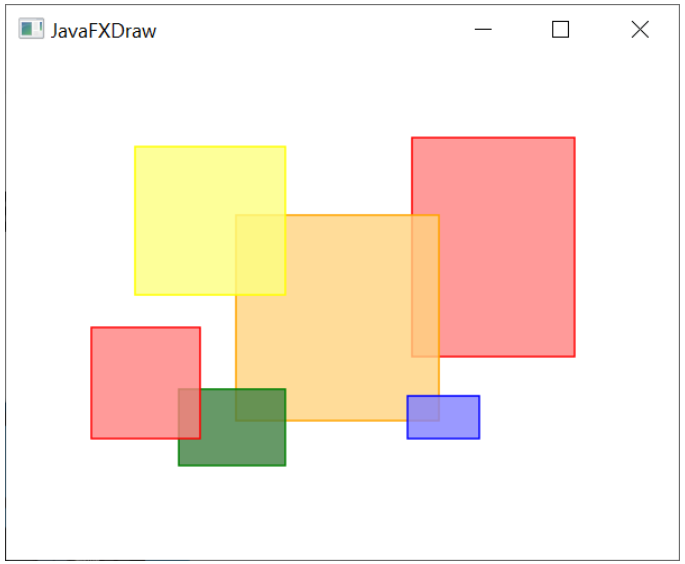
```
private void newRect(Pane pane, MouseEvent e) {
  currentRect = new Rectangle(e.getX(), e.getY(), MIN_SIZE, MIN_SIZE);
  Color nextColor = colors[nextColorIx];
  currentRect.setFill(nextColor.deriveColor(1, 0.5, 1, 0.8));
  currentRect.setStroke(nextColor);
  nextColorIx = (nextColorIx + 1) % colors.length;
  pane.getChildren().add(currentRect);
}
```

color for filling the interior of a rectangle is derived from the outline color



mouse handling example: resizing the rectangle

```
private void resizeRect(MouseEvent e) {
  double newWidth  = Math.max(e.getX() - currentRect.getX(), MIN_SIZE);
  double newHeight = Math.max(e.getY() - currentRect.getY(), MIN_SIZE);
  currentRect.setWidth(newWidth);
  currentRect.setHeight(newHeight);
}
```

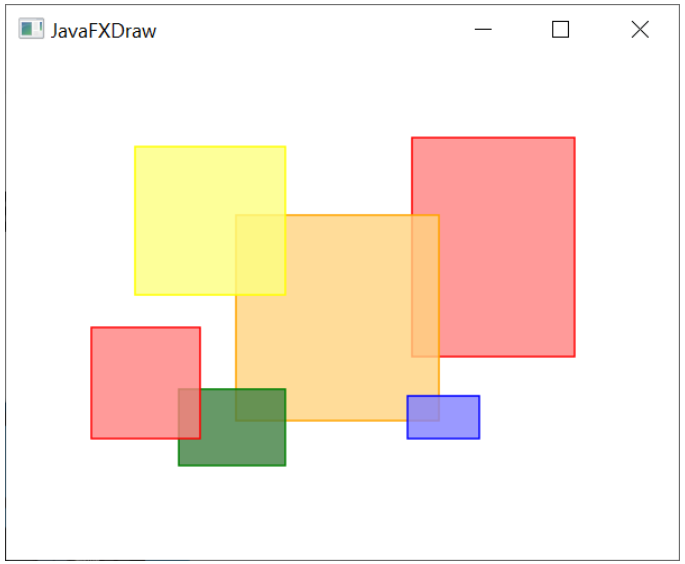


mouse handling example: finishing the rectangle

```
private void finishRect(MouseEvent e) {
  Rectangle thisRect = currentRect;
  thisRect.setOnMousePressed(e2 -> {
    if (e2.isShiftDown()) {
      thisRect.toFront();
    }
    currentXOffset = e2.getX() - thisRect.getX();
    currentYOffset = e2.getY() - thisRect.getY();
    e2.consume(); // stop propagation up
  });
  thisRect.setOnMouseDragged(e2 -> {
    thisRect.setX(e2.getX() - currentXOffset);
    thisRect.setY(e2.getY() - currentYOffset);
    e2.consume(); // stop propagation up
  });
}
```

listeners for the movement of the rectangle

check whether modifier key was down

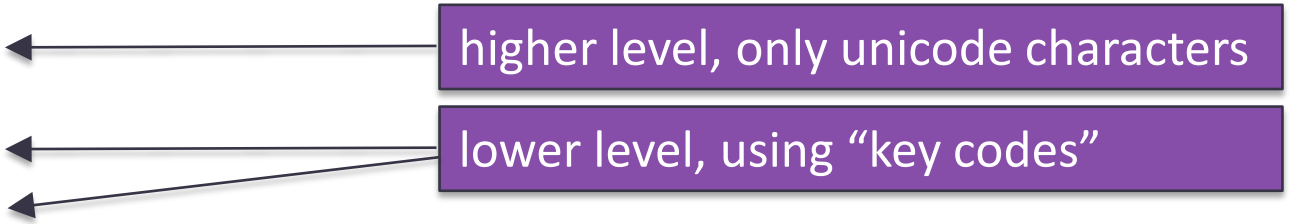


Handling the keyboard

keyboard handling

- KeyEvents are fired whenever a keyboard key is
  - typed
  - pressed
  - released

  on a focused node or scene.



Read the text on

`https://docs.oracle.com/javase/8/javafx/api/javafx/scene/input/KeyEvent.html`

`https://docs.oracle.com/javase/8/javafx/api/javafx/scene/input/KeyCode.html`

for details on key events/codes

key event handlers

No route like with MouseEvent, KeyEvent goes to focused element

- call Node method requestFocus() to request the focus
- there are restrictions, read the Javadoc of requestFocus()!

there can be handlers for many gestures:

- setOnKeyTyped, when a unicode character (e.g. s, r, y, å, ĉ, ş ...) is typed
- setOnKeyPressed, when a key is pressed down
- setOnKeyReleased, when a key is released

these setters associate a handler to their event type

Animations

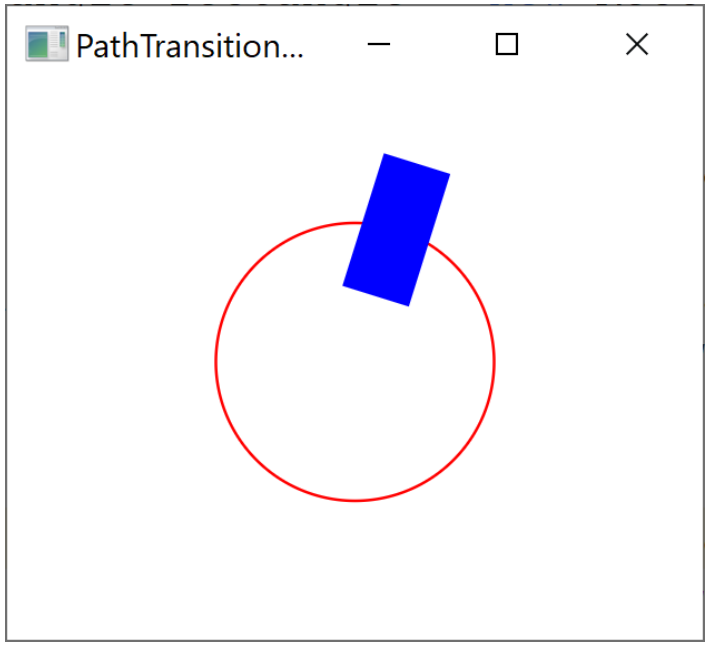
animations

- An animation: an illusion of motion when shapes change over time.
- various options in JavaFX:

1. Transitions: predefined Animations
   - path transition: node follows a path (defined by JavaFX Shape)
   - fade transition: fading a node
   - {rotate, fill, scale, stroke, translate, pause}transition
   - parallel transitions: transitions at the same time
   - sequential transitions: one after an other
2. Timeline: properties of nodes change in time
   - ‘any’ transformation, using KeyValues and KeyFrames
3. Timeline update handlers: explicitly set attributes after elapsed time

Transition

```
public void start(Stage stage) {
  Rectangle rectangle = new Rectangle(25, 50, Color.BLUE);
  Circle circle = new Circle(125, 100, 50, Color.WHITE);
  circle.setStroke(Color.RED);
  Pane pane = new Pane(circle, rectangle);

  PathTransition pt = new PathTransition();
  pt.setDuration(Duration.seconds(4));
  pt.setPath(circle);
  pt.setNode(rectangle);
  pt.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
  pt.setCycleCount(Timeline.INDEFINITE);
  pt.setAutoReverse(true);
  pt.play();

  pane.setOnMouseEntered(e -> pt.pause());
  pane.setOnMouseExited(e -> pt.play());

  stage.setTitle("PathTransitionDemo");
  stage.setScene(new Scene(pane, 250, 200));
  stage.show();
}
```



Timelines

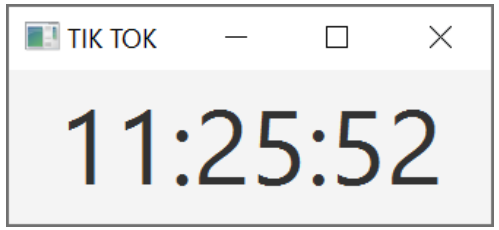
- pathTransition, fadeTransition, rotateTransition, etc. predefine (a limited set of) common transitions
- `TimeLine` can be used to define any transition
  - animation sequence based on KeyFrames
    - e.g. `new TimeLine(KeyFrame… keyFrames)`
  - keyFrames are executed sequentially
- KeyFrame determines an interval on a timeline. They often use node properties
- Generally, a KeyFrame consists of:
  - a `Duration:` length of the interval
  - a set of KeyValues
    - Each KeyValue consist of a Property and end value for that property.
  - e.g. `KeyFrame(Duration time, KeyValue... values)`

Example: moving a text using a Timeline

- *demo*
- text representation
  - `Text msg = new Text("OOP is pretty cool!");`
- Keyvalues:
  - `KeyValue initKeyValueX = new KeyValue(msg.translateXProperty(), sceneWidth);`
  - `KeyValue endKeyValueX  = new KeyValue(msg.translateXProperty(), 0);`
- Keyframe:
  - `KeyFrame initFrame = new KeyFrame(Duration.ZERO, initKeyValueX);`
  - `KeyFrame endFrame  = new KeyFrame(Duration.seconds(3), endKeyValueX);`
- Timeline:
  - `Timeline timeline = new Timeline(initFrame, endFrame);`

```
double sceneWidth  = scene.getWidth();
```

KeyFrame with a handler: a digital clock



no KeyValue but a handler

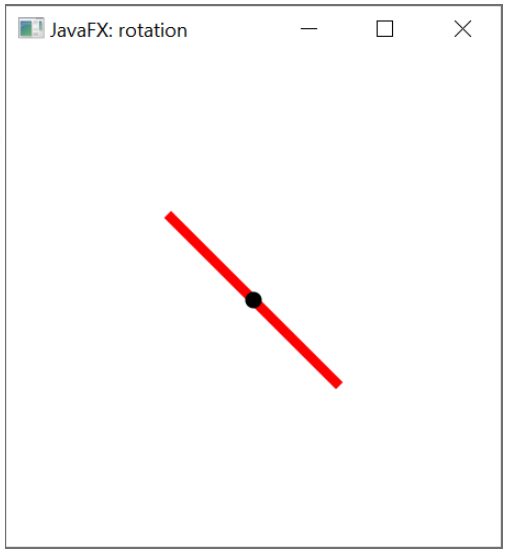
```
public void start(Stage stage) {
  Label timeLabel = new Label();
  timeLabel.setFont(Font.font(48));
  StackPane root = new StackPane(timeLabel);
  stage.setTitle("TIK TOK");
  stage.setScene(new Scene(root, 220, 70));
  stage.show();
  setTime(timeLabel);
  Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(1), e -> setTime(timeLabel)));
  timeline.setCycleCount(Timeline.INDEFINITE);
  timeline.play();
}

private void setTime(Label timeLabel) {
  LocalTime now = LocalTime.now();
  String time = String.format("%d:%02d:%02d",now.getHour(),now.getMinute(),now.getSecond());
  timeLabel.setText(time);
}
```

Intermezzo: transformations (I)

- JavaFX nodes have properties that allow simple transformations

```
public void start(Stage stage) {
  Line line  = new Line(100, 100, 200, 200);
  Circle dot = new Circle(150, 150, 5);
  line.setStroke(Color.RED);
  line.setStrokeWidth(6);

  Pane pane = new Pane(line,dot);
  pane.setOnMouseClicked(e -> line.setRotate(line.getRotate() + 15));

  Scene scene = new Scene(pane, 300, 300);
  stage.setTitle("JavaFX: rotation");
  stage.setScene(scene);
  stage.show();
}
```

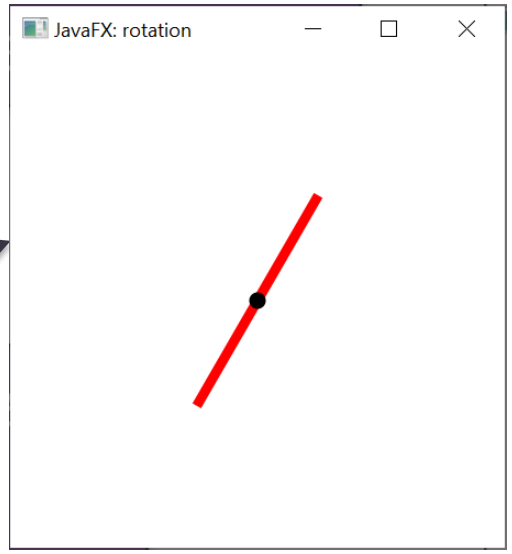
after startup



used to indicate the center of the line/rotation

change the rotate property

after 5 clicks

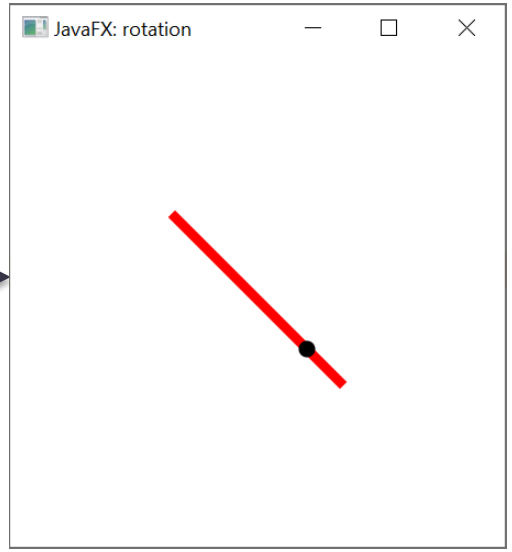


Intermezzo: transformations (II)

- for more complex transformations use package `javafx.scene.transform`

```
public void start(Stage stage) {
  Line line  = new Line(100, 100, 200, 200);
  Circle dot = new Circle(180, 180, 5);
  line.setStroke(Color.RED);
  line.setStrokeWidth(6);

  Rotate rotate = new Rotate(0, 180, 180);
  line.getTransforms().add(rotate);
  Pane pane = new Pane(line,dot);
  pane.setOnMouseClicked(e -> rotate.setAngle(rotate.getAngle() + 15));

  Scene scene = new Scene(pane, 300, 430);
  stage.setTitle("JavaFX: rotation");
  stage.setScene(scene);
  stage.show();
}
```

after startup

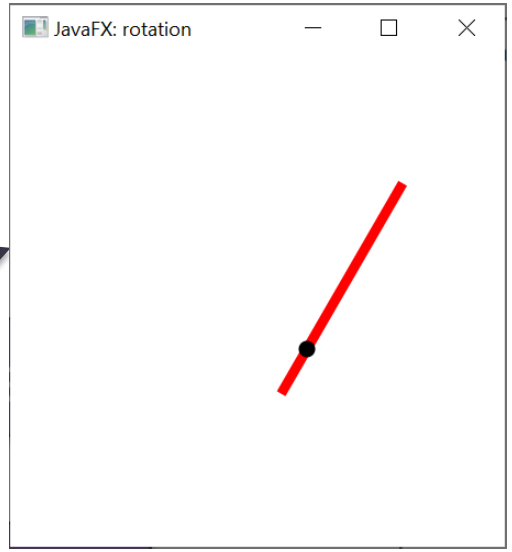


different center of rotation

installing the transfomation

change angle of rotation

after 5 clicks



TimeLine with transformations: an analog clock (I)

```
static final double RAD = 100, CENTRE_X = RAD + 10, CENTRE_Y = RAD + 10;
```

```
public void start(Stage stage) {
  Circle dial = new Circle(CENTRE_X, CENTRE_Y, RAD, Color.WHITE);
  dial.setStroke(Color.BLACK);

  Line hourHand = new Line(CENTRE_X, CENTRE_Y + 4, CENTRE_X, CENTRE_Y * 0.6);
  hourHand.setStrokeWidth(6);

  Line minHand = new Line(CENTRE_X, CENTRE_Y + 8, CENTRE_X, CENTRE_Y * 0.4);
  minHand.setStrokeWidth(4);

  Line secLine = new Line(CENTRE_X, CENTRE_Y + 14, CENTRE_X, CENTRE_Y * 0.2);
  secLine.setStrokeWidth(2);
  secLine.setStroke(Color.RED);
  Circle secCircle = new Circle(CENTRE_X, CENTRE_Y + 14, 8);
  secCircle.setFill(Color.RED);
  Group secHand = new Group(secLine, secCircle);
```

before the time is set

TimeLine with transformations: an analog clock (II)

```
  Rotate hourRot = new Rotate(0, CENTRE_X, CENTRE_Y);
  hourHand.getTransforms().add(hourRot);
  Rotate minRot  = new Rotate(0, CENTRE_X, CENTRE_Y);
  minHand.getTransforms().add(minRot);
  Rotate secRot  = new Rotate(0, CENTRE_X, CENTRE_Y);
  secHand.getTransforms().add(secRot);

  Pane pane = new Pane(dial, hourHand, minHand, secHand);
  pane.setStyle("-fx-background-color: grey;");
  stage.setTitle("Ring ...");
  stage.setScene(new Scene(pane, 2 * RAD + 20, 2 * RAD + 20));
  stage.show();

  setTime(hourRot, minRot, secRot);
  Timeline timeline = new Timeline(new KeyFrame(Duration.seconds(1),
                                                e -> setTime(hourRot, minRot, secRot)));
  timeline.setCycleCount(Animation.INDEFINITE);
  timeline.play();
}
```

TimeLine with transformations: an analog clock (III)

```
private void setTime(Rotate hourRot, Rotate minRot, Rotate secRot) {
  LocalTime now = LocalTime.now();
  hourRot.setAngle(now.getHour() * 30 + now.getMinute() / 2);
  minRot.setAngle(now.getMinute() * 6);
  secRot.setAngle(now.getSecond() * 6);
}
```

Object-Oriented Design: Separating GUI and data

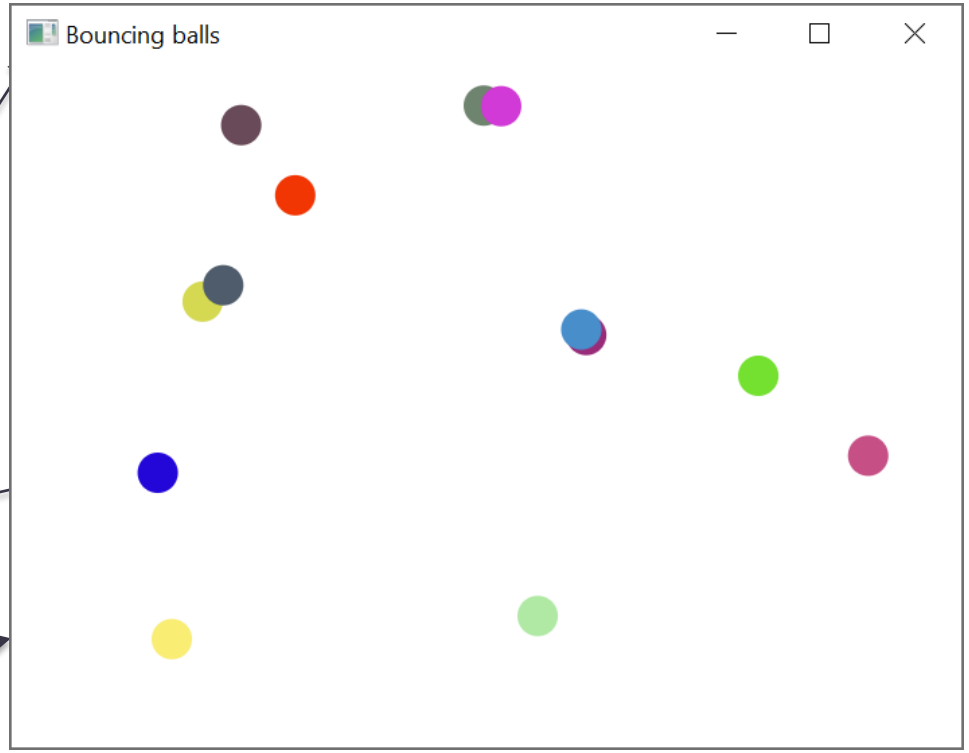
plain JavaFX animation: bouncing balls

- Tight coupling between world objects (Model) and JavaFX objects (View)
- keyboard:
  - 'n': new ball
  - other: start/stop animation
  - timeline: move balls

Application: holds timeline and key handler

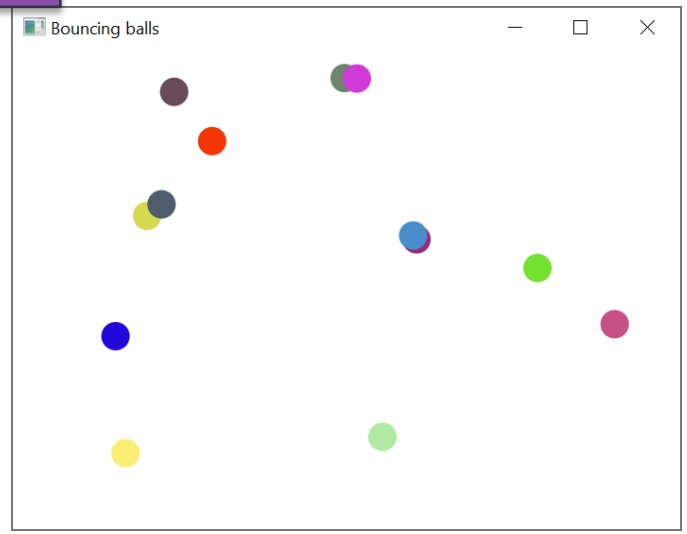
Ball: subclass of Circle

World: subclass of Pane



Ball (I)

Ball is also its own JavaFX representation

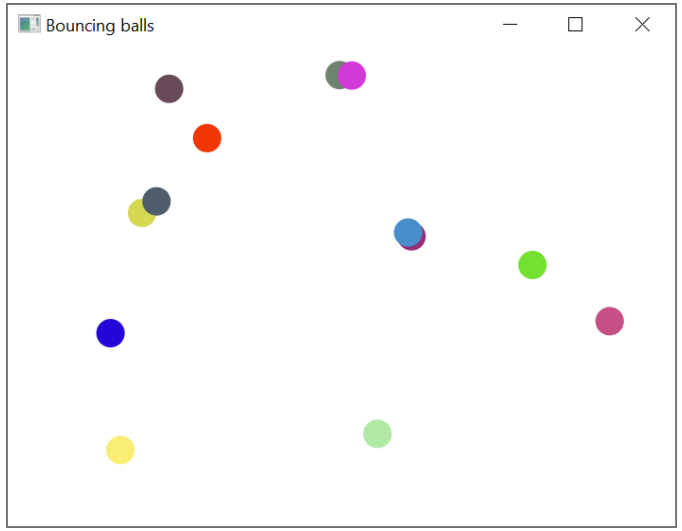



```
public class Ball extends Circle {

  private final static double RADIUS = 10;
  private double posX, posY;
  private double velX, velY;

  public Ball(double x, double y, double vx, double vy, Color color) {
    super(x, y, RADIUS, color);
    posX = x;
    posY = y;
    velX = vx;
    velY = vy;
  }
```

Circle constructor

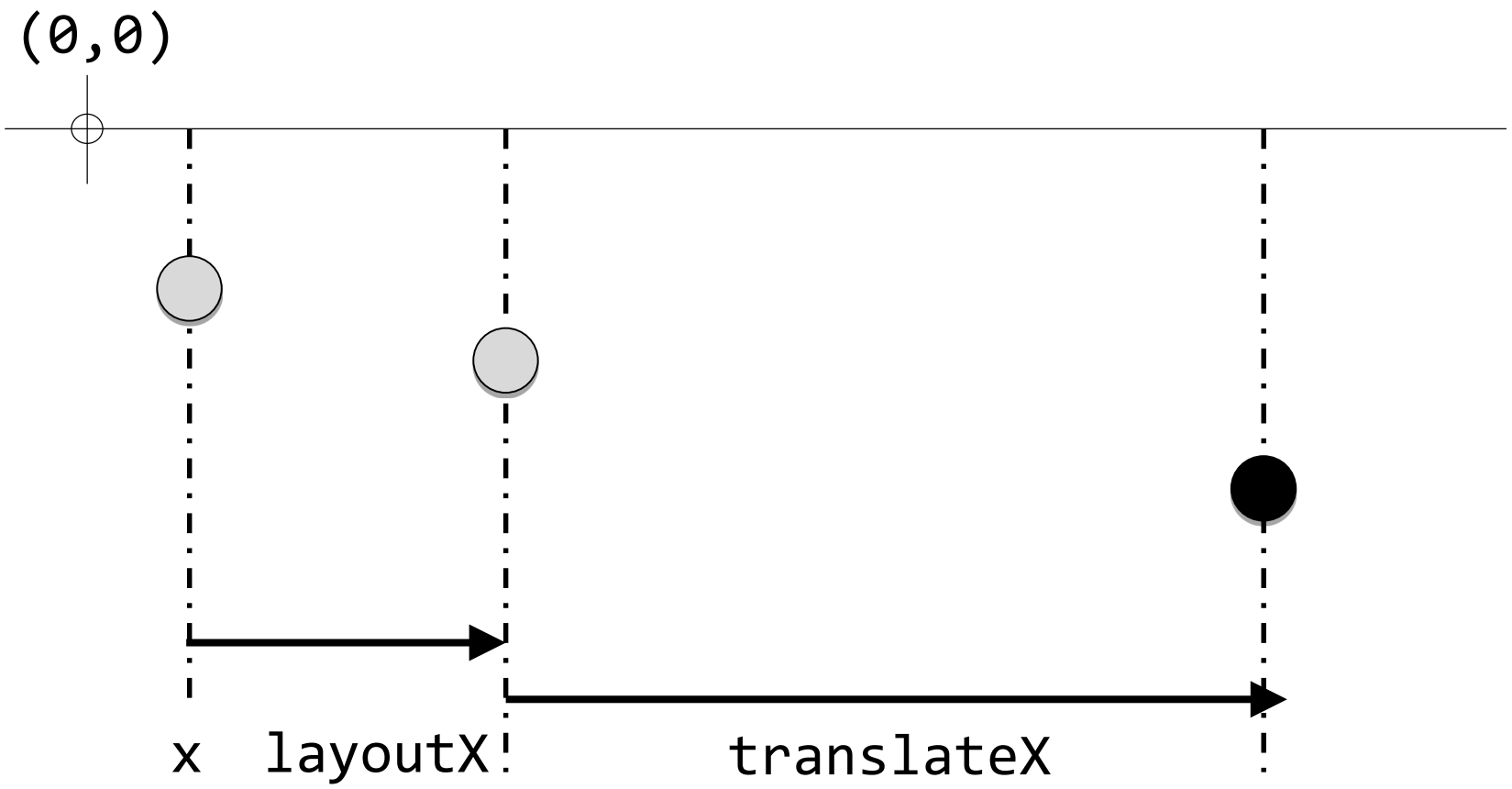
Ball (II)

```java
public void step(double maxX, double maxY) {
    posX += velX;
    posY += velY;
    if (posX < RADIUS || posX > maxX - RADIUS) {
        velX *= -1;
    }
    if (posY < RADIUS || posY > maxY - RADIUS ) {
        velY *= -1;
    }
    this.setTranslateX(posX - getCenterX());
    this.setTranslateY(posY - getCenterY());
}
```



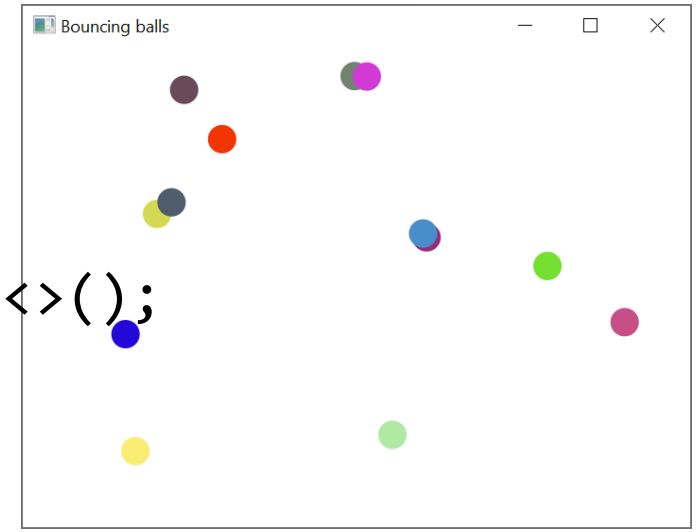
turn when the border is hit

Node position



World (I)

World is also its own JavaFX representation

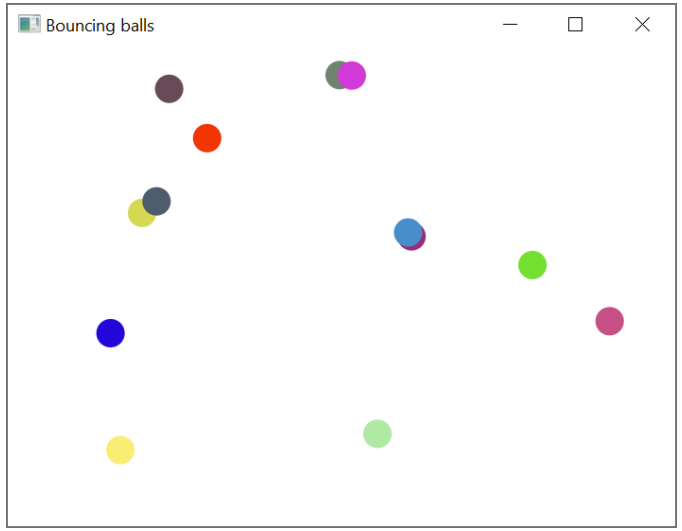



```
public class World extends Pane {

  private final List<Ball> balls      = new ArrayList<>();
  private static final Random random = new Random();

  public void newBall() {
    Ball ball = new Ball(random.nextDouble(this.getWidth()),
                         random.nextDouble(this.getHeight()),
                         random.nextDouble(1, 4), random.nextDouble(1, 4),
                         Color.rgb(random.nextInt(256),
                             random.nextInt(256), random.nextInt(256)));
    balls.add(ball);
    this.getChildren().add(ball);
  }
```

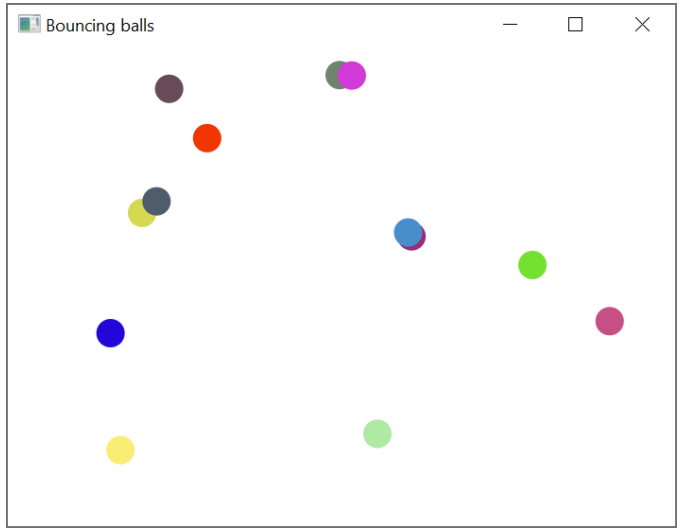
random color

World (II)

```
  public void tick() {
    for (Ball b : balls) {
      b.step(this.getWidth(), this.getHeight());
    }
  }
}
```



Main class: BallWorld (I)



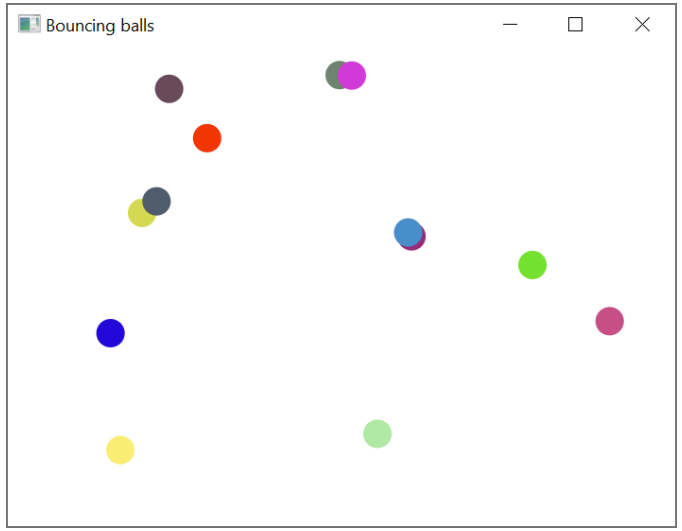

```
public class BallWorld extends Application {
```

timer delay in milliseconds

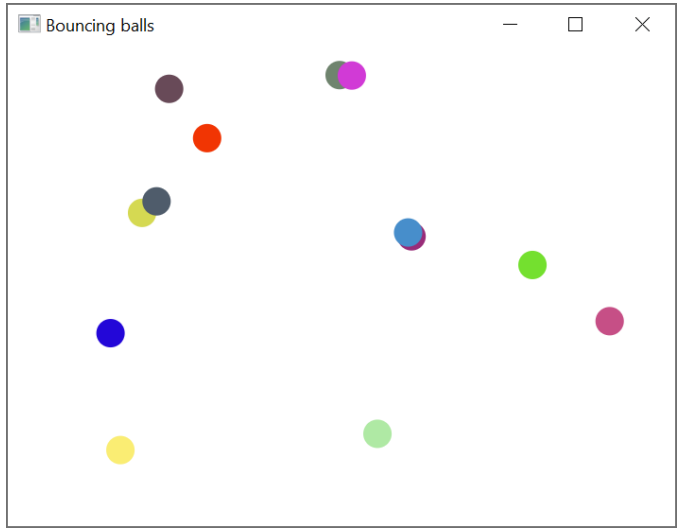
```
  private final static double DELAY = 20;
  private boolean running = true;

  public void start(Stage primaryStage) {
    World world = new World();
    Timeline timeline
            = new Timeline(new KeyFrame(Duration.millis(DELAY),
                                e -> world.tick()));
    timeline.setCycleCount(Animation.INDEFINITE);
    timeline.play();
```

Main class: BallWorld (II)

```
  Scene scene = new Scene(world, 300, 250);
  scene.setOnKeyPressed(event -> {
    if (event.getCode().equals(KeyCode.N)) {
      world.newBall();
    } else if (running) {
      timeline.pause();
      running = false;
    } else {
      timeline.play();
      running = true;
    }
  });
  primaryStage.setTitle("Bouncing balls");
  primaryStage.setScene(scene);
  primaryStage.show();
}
```



bouncing balls review



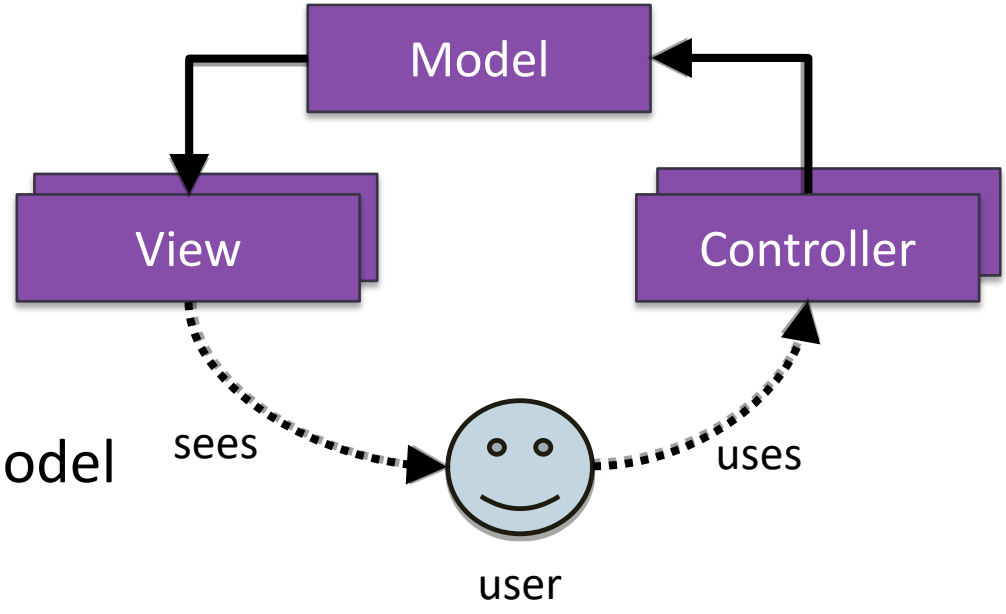
- Close coupling between state and its JavaFX view

```
public class Ball extends Circle { ..
public class World extends Pane { ..
```

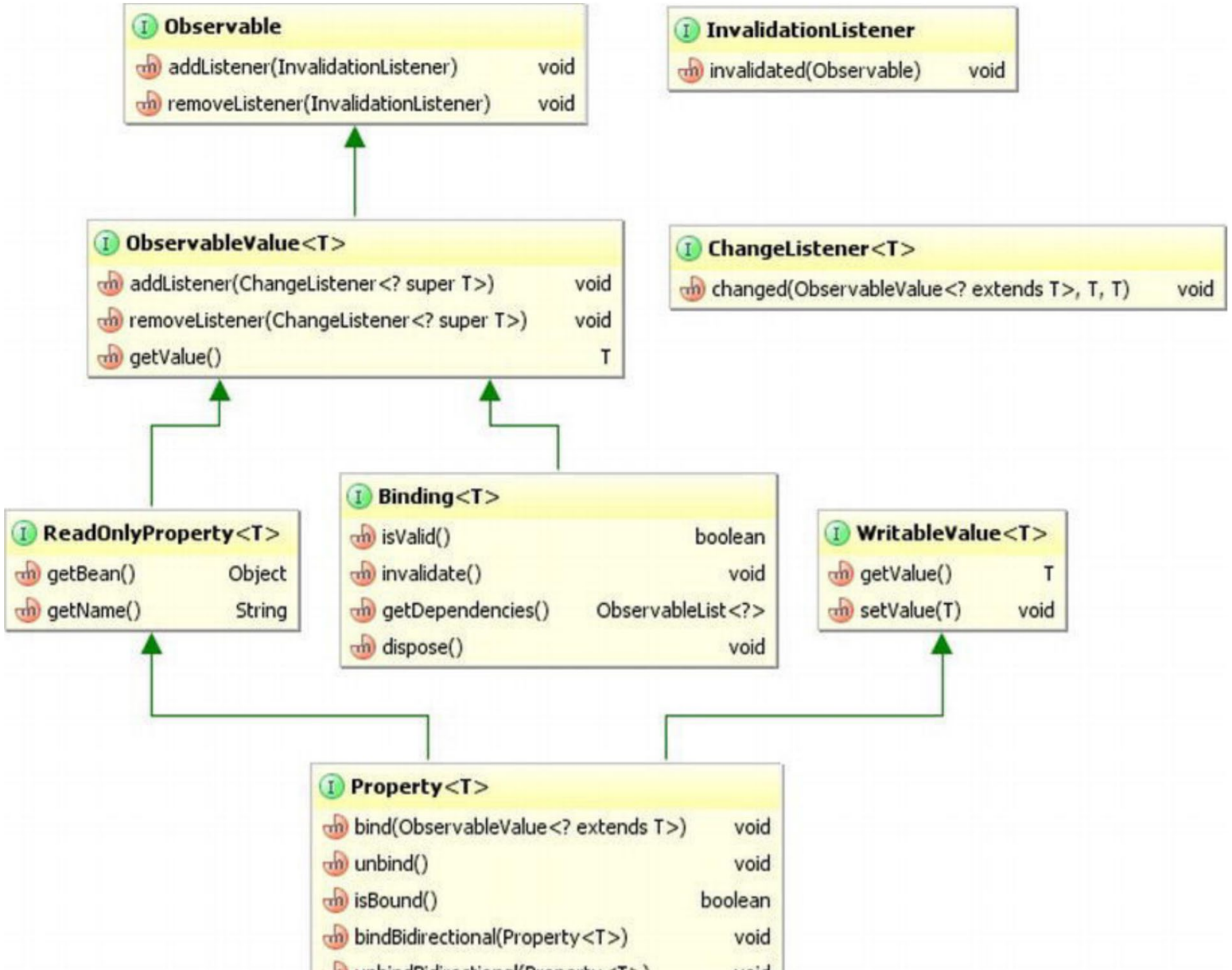
- This is fine if you are experiment
- Otherwise it is essential to separate the state and its JavaFX view better
  - like we told you in lecture 3/4
  - the MVC pattern is one of the ways to do this in with JavaFX

Model-View-Controller principle

- JavaFX can be structured according the MVC principle: separate state and GUI
  1. *Model*: state/core of program
     - has no idea of GUI (views)
  2. *View*: the JavaFX stages displaying a view on the model
     - unknown by model
  3. *Controller*: the handlers changing the model
- Properties are ObservableValues
  - they can be part of the model
- Our JavaFX examples focus on drawing (view) and handlers (controllers), there is usually no separate model



Properties, Observables, …



bouncing balls with separated MVC

Model

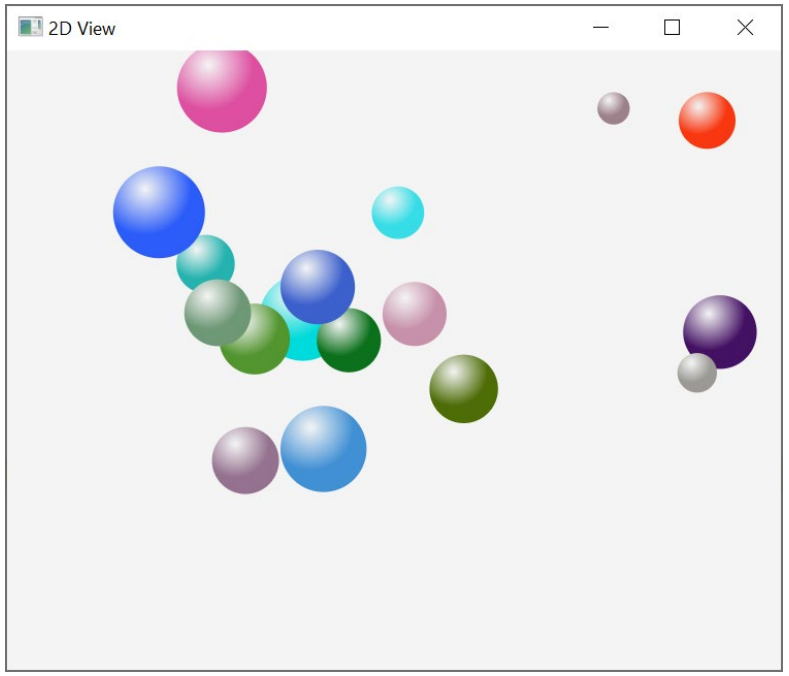
1. Ball: xPos, yPos, xVel, yVel, radius
2. World: list of balls

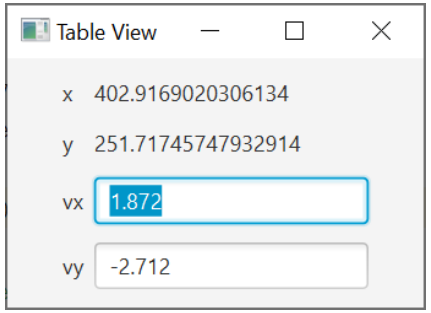
Views

1. balls on canvas
2. ball properties in table

controllers

1. timer
2. set velocity
3. creating balls
4. mouse handling





Model: Ball (I)

```
public class Ball {
  private final DoubleProperty radius;
  private final DoubleProperty xPos, yPos;
  private final DoubleProperty xVel, yVel;
  private World world;

  public Ball(double px, double py, double vx, double vy, double radius,
              World world) {
    this.world  = world;
    this.radius = new SimpleDoubleProperty(radius);
    this.xPos   = new SimpleDoubleProperty(px);
    this.yPos   = new SimpleDoubleProperty(py);
    this.xVel   = new SimpleDoubleProperty(vx);
    this.yVel   = new SimpleDoubleProperty(vy);
  }
  ...
```

getters for all Property fields (not shown here)

Model: Ball (II)

```
  private boolean touchesBorder(double coord, double maxCoord) {
    return coord < radius.doubleValue() || coord > maxCoord - radius.doubleValue();
  }

  public void move() {
    xPos.setValue(xPos.doubleValue() + xVel.doubleValue());
    if (touchesBorder(xPos.doubleValue(), world.getWorldWidth())) {
      xVel.set(-xVel.doubleValue());
    }
    yPos.setValue(yPos.doubleValue() + yVel.doubleValue());
    if (touchesBorder(yPos.doubleValue(), world.getWorldHeight())) {
      yVel.set(-yVel.doubleValue());
    }
  }
}
```

Model: World

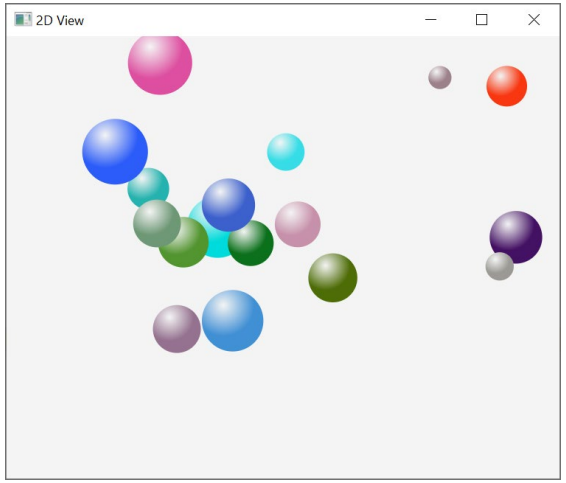
```
public class World {
  private double worldWidth, worldHeight;
  private final List<Ball> balls = new ArrayList<>();
  private static final Random random = new Random();

  public World(double worldWidth, double worldHeight) {
    this.worldWidth = worldWidth;
    this.worldHeight = worldHeight;
  }

  public Ball addBall(double x, double y) {
    Ball ball = new Ball( x, y, random.nextDouble(1, 4), random.nextDouble(1, 4),
                          random.nextDouble(10, 30), this);
    balls.add(ball);
    return ball;
  }

  public void kickBalls() {
    balls.forEach(b -> b.move());
  }
}
```

random velocity and radius

`forEach`: internal iteration, more about this in week 11

getters and setters for `worldWidth` and `worldHeight` (not shown here)

Controller/View: TwoDimVC



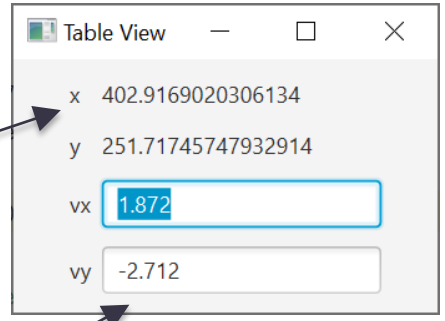

```
public class TwoDimVC extends Pane {

   public TwoDimVC(World world, TableVC tableView) {
      this.widthProperty().addListener((ov, ow, nw) -> world.setWorldWidth(nw.doubleValue()));
      this.heightProperty().addListener((ov, ow, nw) -> world.setWorldHeight(nw.doubleValue()));
      this.setOnMouseClicked(e -> newBall( e.getX(), e.getY(), world, tableView ));
   }

   private void newBall(double x, double y, World world, TableVC tableView) {
     Ball ball = world.addBall(x, y);
     Circle ballView = new Circle();
     RadialGradient rg = new RadialGradient(…);
     ballView.setFill(rg);
     ballView.radiusProperty().bind( ball.radiusProperty());
     ballView.translateXProperty().bind(ball.xPosProperty());
     ballView.translateYProperty().bind(ball.yPosProperty());
     ballView.setOnMouseClicked( e -> { tableView.showBall(ball); e.consume(); } );
     this.getChildren().add(ballView);
     tableView.showBall(ball);
   }
```

details of this have been omitted

Controller/View: **TableVC** (I)

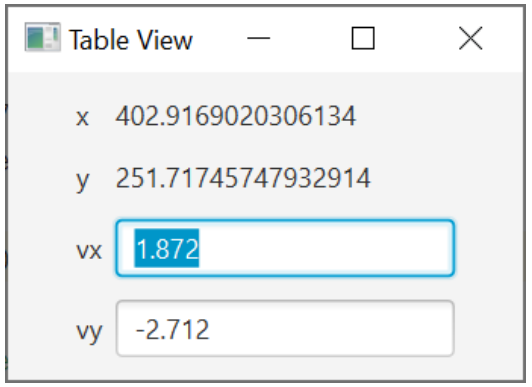
```
public class TableVC extends GridPane {
  private final Label xLabel, yLabel;
  private final TextField vxLabel, vyLabel;
  private Ball selectedBall = null;

  public TableVC( Driver driver ) {
    setAlignment(Pos.CENTER);
    setHgap(5);
    setVgap(10);
    add(new Label("x"), 0, 0);
    add(xLabel = new Label(""), 1, 0);
    add(new Label("y"), 0, 1);
    add(yLabel = new Label(""), 1, 1);
    add(new Label("vx"), 0, 2);
    add(vxLabel = new TextField(""), 1, 2);
    add(new Label("vy"), 0, 3);
    add(vyLabel = new TextField(""), 1, 3);
    setEditable(false);
    this.setVisible(false);
  }
```

TableVC is-a GridPane



Initially the Pane is not visible

Controller/View: **TableVC** (II)



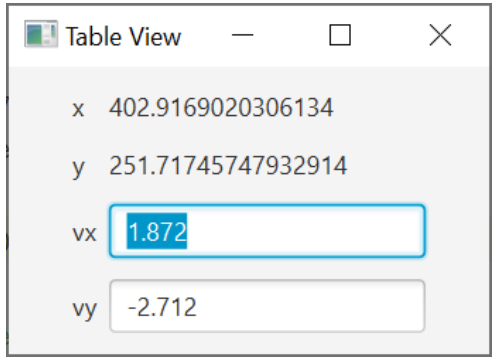

```
  public void setEditable(boolean edit) {
    vxLabel.setEditable(edit);
    vyLabel.setEditable(edit);
  }

  public void showBall(Ball ball) {
    this.setVisible(true);
    xLabel.textProperty().bind(ball.xPosProperty().asString());
    yLabel.textProperty().bind(ball.yPosProperty().asString());
    if ( selectedBall != null ) {
       vxLabel.textProperty().unbindBidirectional(selectedBall.xVelProperty());
       vyLabel.textProperty().unbindBidirectional(selectedBall.yVelProperty());
    }
    selectedBall = ball;
    vxLabel.textProperty().bindBidirectional(ball.xVelProperty(), new NumberStringConverter());
    vyLabel.textProperty().bindBidirectional(ball.yVelProperty(), new NumberStringConverter());
  }
}
```

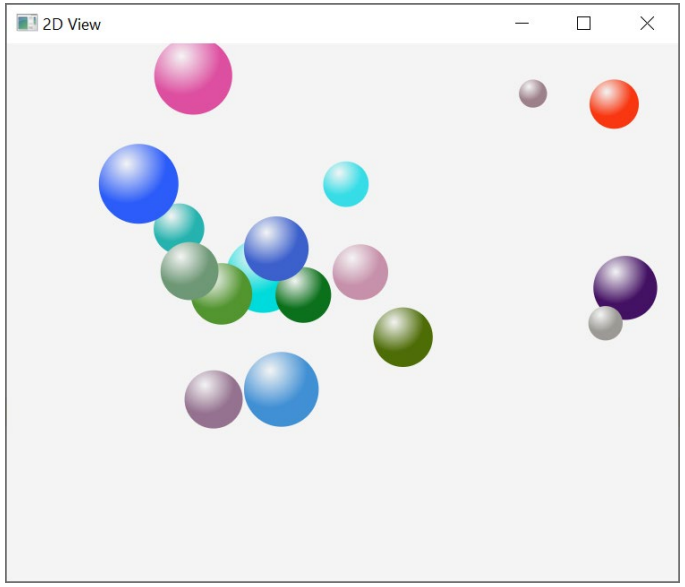
all updates are done through property binding

Timer controller: Driver

```
public class Driver {
  private final Timeline loop;

  public Driver(World world) {
    loop = new Timeline(new KeyFrame(Duration.millis(20), e -> world.kickBalls()));
    loop.setCycleCount(Timeline.INDEFINITE);
  }

  public void start() {
    loop.play();
  }

  public void pause() {
    loop.pause();
  }
}
```

Gluing everything together



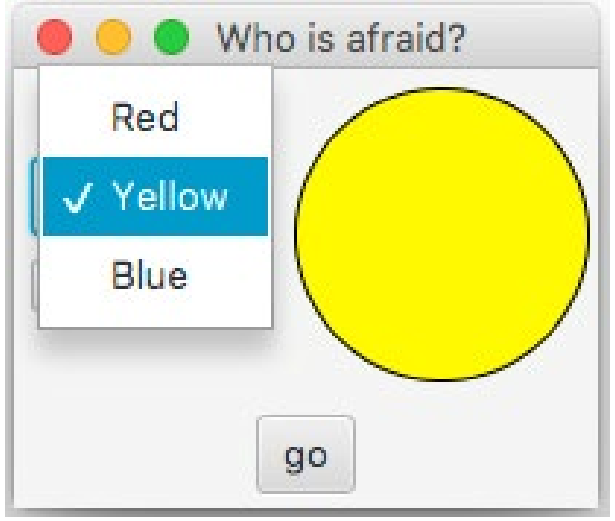



```
public void start(Stage primaryStage) {
  World world   = new World(600, 400);
  Driver driver = new Driver(world);
  TableVC tv    = new TableVC(driver);
  TwoDimVC tdv  = new TwoDimVC(world, tv);

  Stage secondStage = new Stage();
  secondStage.setTitle("Table View");
  secondStage.setScene(new Scene(tv));
  secondStage.show();

  primaryStage.setTitle("2D View");
  primaryStage.setScene(new Scene(tdv, world.getWorldWidth(), world.getWorldHeight()));
  primaryStage.show();
}
```

You should be more than capable of looking these up,
reading about them,
and experimenting with them yourselves:

Some more useful widgets

image, image view, radio buttons, choice boxes



Lecture 10: Design Patterms