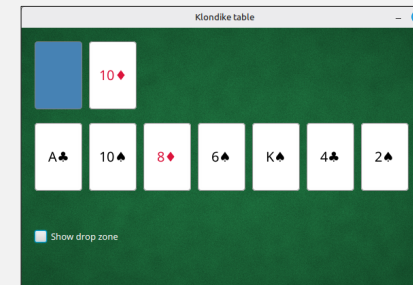


More JavaFX

- previously
 - which control to use
 - how to read its value
 - how to validate input
 - dialogs
 - a bit of layout
 - registering event handlers for nodes
 - choosing a layout container
- today
 - custom components
 - drag and drop

Demo



Shapes

- `javafx.scene.shape`
 - Rectangle, Circle, Line, Polygon
 - `setArcWidth()` / `setArcHeight()` round a rectangle's corners
 - Text
 - a shape that draws a string
 - every shape has a fill color (inside) and a stroke color (outline)
 - by default, a shape's outline is centered over its boundary
 - `setStrokeType(StrokeType.INSIDE)` keeps it fully inside the shape
- shapes are nodes too
 - add to the scene graph
 - parent layout positions them

Cards

```
Rectangle face = new Rectangle(70, 100); // width, height
face.setArcWidth(10);
face.setArcHeight(10);
face.setFill(Color.WHITE);
face.setStroke(Color.GRAY);
face.setStrokeType(StrokeType.INSIDE);
```

```
Text label = new Text("7♥");
label.setFill(Color.CRIMSON);
label.setFont(Font.font(20));
```

Color

- `javafx.scene.paint.Color`
 - be careful imports aren't `java.awt.Color` instead!
- specifying a color
 - named constants: `Color.WHITE`, `Color.CRIMSON`, `Color.DARKGREEN`, ...
 - exact color: `Color.rgb(21, 101, 52)` or `Color.web("#156534")`
- transparency
 - alpha (opacity) is fourth value (rgba) 0.0-1.0
 - e.g. `Color.rgb(255, 255, 255, 0.25)`
 - `Color.TRANSPARENT` – fully see-through but still there
 - null fill – no fill at all
 - alpha < 1.0 typically used for highlights, overlays

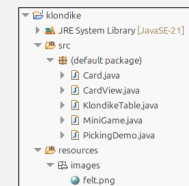


Images

- Image holds the picture
- ImageView is the node that shows it
 - size the view (display size) with `setFitWidth()` / `setFitHeight()`
 - scaling an image can distort and lose sharpness
 - prefer images that are the right size when details matter e.g. the image has fine detail or is used as a background for things drawn on top of
- load from the project's resources instead of a path

```
// project contains resources/images/felt.png, with resources added as
// a source folder
Image felt =
    new Image(getClass().getResource("/images/felt.png").
        toExternalForm());
ImageView background = new ImageView(felt);
```

- avoids problems with fragility of reference, packaging, portability, bad paths
- use resources for things the program ships with, file paths for things the user supplies



Custom Components

- a card on screen is made up of several nodes
 - a rounded rectangle
 - a label
 - (maybe) a back pattern
- create a *custom component* so the rest of the program can just deal with “a card”

name `CardView` reflects that this is just the display of the card – separate from the card state (`Card`)

```
public class CardView extends StackPane {
    public static final double WIDTH = 70;
    public static final double HEIGHT = 100;

    private Rectangle face_;
    private Text label_;

    public CardView(String text, boolean red) {
        face_ = new Rectangle(WIDTH, HEIGHT);
        face_.setArcWidth(10);
        face_.setArcHeight(10);
        face_.setStrokeType(StrokeType.INSIDE);
        face_.setStroke(Color.GRAY);

        label_ = new Text(text);
        label_.setFont(Font.font(20));
        if (red) {
            label_.setFill(Color.CRIMSON);
        } else {
            label_.setFill(Color.BLACK);
        }

        getChildren().addAll(face_, label_);
        setMinSize(WIDTH, HEIGHT);
        setPrefSize(WIDTH, HEIGHT);
        setMaxSize(WIDTH, HEIGHT);
        setFaceUp(true);
    }

    public void setFaceUp(boolean faceUp) {
        label_.setVisible(faceUp);
        if (faceUp) {
            face_.setFill(Color.WHITE);
        } else {
            face_.setFill(Color.STEELBLUE);
        }
    }

    public void setHighlighted(boolean on) {
        if (on) {
            face_.setStroke(Color.DODGERBLUE);
            face_.setStrokeWidth(4);
        } else {
            face_.setStroke(Color.GRAY);
            face_.setStrokeWidth(1);
        }
    }
}
```

CardView Details

- exact sizing
 - it is often correct to let the layout decide the size of components
 - with cards, the pieces need to line up with each other
 - some layouts stretch only in certain directions, so allowing resizing means things can become misaligned
 - keep outlines within the region
 - cards can sit next to each other without overlap
- why StackPane as a base?
 - suited for overlapping children that should line up with each other
- inheritance or composition?
 - common (and convenient) to extend a pane but violates LSP
 - outsiders can call StackPane methods like `card.getChildren().clear()`

```

public class CardView {
    private StackPane root_ = new StackPane();
    // ... same children and methods ...
    public Node getNode() { return root_; }
}

```

```

pile.getChildren().add(card.getNode());

```

18

Clicking Non-Buttons

- the hand cursor when the mouse is over the component is an *affordance*
 - no visual cue otherwise that the stock pile is clickable
- registering the handler
 - `setOnMouseClicked` – only controls have `setOnAction`
 - `e.getClickCount()` allows detection of double clicks

```

stock_.setCursor(Cursor.HAND); // hint that it's clickable

```

```

stock_.setOnMouseClicked(e -> drawFromStock());

```

CPSC 329: Software Development • Fall 2026

89

Drag and Drop

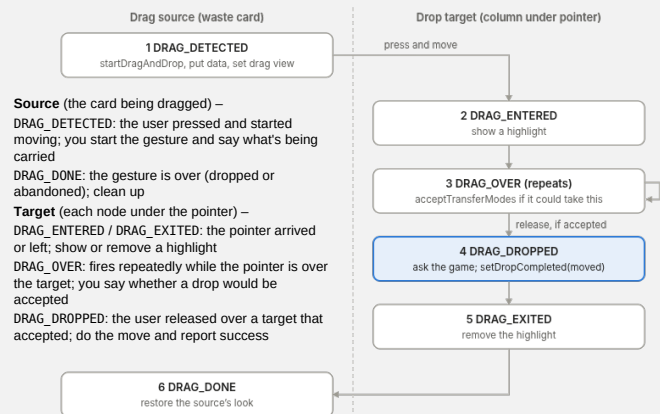
- two ways to drag

	move the node	platform drag and drop
what is it?	you move the node yourself using ordinary mouse events	utilize Java's built-in drag-and-drop gesture
mechanism	<code>setOnMousePressed / Dragged / Released</code> ; move the node with <code>setTranslateX/Y</code>	source starts a gesture; JavaFX tells each target the pointer passes over
what moves	the node	an image of the node; the node stays put
who finds the drop target	you, by checking coordinates	JavaFX, by picking the node under the pointer
targets can refuse	you code it	built in: a target accepts or doesn't
good for	sliders, freely positioned items, drawing	moving things onto something: cards to piles, files to folders

- does the item get dropped on something?
 - yes → platform drag and drop
 - no → move the node
 - when there's free placement (e.g. sticky notes), drag without drop (e.g. handles and controls, panning, drawing), other things following live (e.g. dragging a shape and having connector lines come along), speed and path matter

19

Drag and Drop Gesture – Events



CPSC 329: Software Development • Fall 2026

91

Drag Source

```
waste_.setOnDragDetected(e -> {
    Dragboard db = waste_.startDragAndDrop(TransferMode.MOVE);
    ClipboardContent content = new ClipboardContent();
    content.putString("waste"); // what is being carried
    db.setContent(content);

    SnapshotParameters params = new SnapshotParameters();
    params.setFill(Color.TRANSPARENT); // no white box around the card
    db.setDragView(waste_.snapshot(params, null),
        CardView.WIDTH / 2, CardView.HEIGHT / 2);

    waste_.setOpacity(0.4); // "lifted" look
    e.consume();
});

waste_.setOnDragDone(e -> {
    waste_.setOpacity(1.0);
    e.consume();
});
```

- Dragboard is a clipboard for the drag gesture
- the drag view is a picture that follows the pointer
 - snapshot is essentially a screengrab of the component's look
 - offsets put the pointer at the middle of the image
- dimming the source and restoring on DRAG_DONE gives the "picked up" feel
- consume events so that higher level containers don't respond

12

Drop Targets

```
for (int i = 0; i < 7; i++) {
    final int col = i; // the loop variable changes with each iteration so lambdas need their own copy
    StackPane target = columns_[i];
    target.setOnDragOver(e -> {
        if (e.getGestureSource() != target && e.getDragboard().hasString()) {
            e.acceptTransferModes(TransferMode.MOVE);
        }
        e.consume();
    });
    target.setOnDragEntered(e -> {
        setColumnHighlight(col, true);
        e.consume();
    });
    target.setOnDragExited(e -> {
        setColumnHighlight(col, false);
        e.consume();
    });
    target.setOnDragDropped(e -> {
        boolean moved = game_.moveWasteTo(col); // the game decides whether the drop is legal
        if (moved) {
            message_.setText("");
            refreshTable(); // redraw from the game state – don't shuffle nodes around by hand
        } else {
            message_.setText(game_.getLastErrorMessage());
        }
        e.setDropCompleted(moved); // tells the source whether the move happened – source can access in DRAG_DONE with e.isDropCompleted()
        e.consume();
    });
}
```

93

Allowing/Refusing the Drop: Who and When

- the game decides whether the move is legal
 - separation of concerns: the view handles the gesture, the game handles the rules
- refusal on drop
 - DRAG_OVER should only check "is this something I could take?"
 - it fires with every mouse movement – keep it cheap
 - allows an opportunity to explain a refusal
 - needs the make-the-move game method to report whether it worked and, if not, why
- refusal in DRAG_OVER
 - shows legality before the drop, but DRAG_DROPPED never fires on a refused target
 - the cursor may not show the refusal on every platform and never shows why
 - use your own cue in DRAG_ENTERED to do both (undo in DRAG_EXITED)
 - needs a game method that checks a move without making it

1

Common Gotchas



- forgetting `e.consume()` – the parent also reacts
- no content on the Dragboard – the gesture never starts
- expecting the node itself to move – only its image does
- forgetting that DRAG_DROPPED only reaches targets that accepted in DRAG_OVER
- updating the view by moving nodes yourself – the view and the game state drift apart
 - the game move the gesture invokes may change more than the source and target of the drag – the view only knows about the gesture, only the game knows all the consequences

Layering

- layering – about front vs back for overlapping nodes
 - the one drawn later is in front
 - a parent draws its children in order, first at the back and last at the front
- changing child order – among siblings
 - `node.toFront()` / `node.toBack()` move a node to the end or start of its parent's list
 - in `VBox`, `HBox`, `FlowPane`, `TilePane` list order is also position – changing node order changes the layout
 - in `StackPane`, `GridPane`, `BorderPane`, `Pane`, `Group` only front vs back changes – change node order doesn't affect layout
- changing view order – among siblings
 - `node.setViewOrder(n)` changes front vs back without touching the child list
 - lower values are in front – default is 0
 - safe in any pane – child order is not affected

96

Layers in a Window

- a `StackPane` draws its children in order
 - first child is at the bottom, the last is on top
- build the scene in layers, each with a job
 - common to have background, content, highlight (back to front)

```
StackPane (the table)
├─ ImageView      felt background
├─ VBox           the piles: stock, waste, 7 tableau columns
└─ Rectangle      "drop zone" band over the tableau row (translucent)
```

- `StackPane` centers each layer
 - make them all the same size so they line up

CPSC 329: Software Development • Fall 2026

97

Picking

- picking – which node gets the event?
 - the frontmost node that counts as “there” under the pointer
 - the event bubbles up from the picked node through its parents
- shapes are “there” when they are drawn
 - a transparent fill is “there”, a null fill is not (only the outline can be hit)
- panes and controls are “there” across their whole rectangle, even with no background
 - change with `setPickOnBounds(false)`
- a `mouseTransparent` node is never “there”
 - events pass through to what is behind
- hidden nodes (`setVisible(false)`) are never picked
- same rules apply for mouse events and drag events

CPSC 329: Software Development • Fall 2026

98

Picking

```
Rectangle hollow = new Rectangle(80, 80);
hollow.setFill(null);
hollow.setStroke(Color.BLACK);
hollow.setOnMouseClicked(e -> System.out.println("hollow"));
```

```
Rectangle clear = new Rectangle(80, 80);
clear.setFill(Color.TRANSPARENT);
clear.setStroke(Color.BLACK);
clear.setOnMouseClicked(e -> System.out.println("clear"));
```

- what happens?
 - “clear” prints
 - “hollow” only prints when you click exactly on the outline
- gotcha – a outline with a null fill looks like a drop target, but drops in its middle silently do nothing



CPSC 329: Software Development • Fall 2026

99

Overlays That Swallow Events

- the drop zone hint band sits on top of the tableau columns and has a fill (translucent white)
 - it is topmost and “there” – it gets drag events

- fix: mark decorations as invisible to the mouse

```
hint.setMouseTransparent(true); // drawn, but never picked
```

- guideline
 - highlights, overlays, backgrounds, and labels on top should be mouse-transparent
- debugging tip – add the following to print which node is actually being picked

```
scene.addEventFilter(MouseEvent.MOUSE_CLICKED,  
    e -> System.out.println(e.getTarget()));
```

- added to scene – the top level
- filter always runs even if a handler consumes the click

10

Common Mistakes

- setting only the preferred size, so a layout stretches a component out of line
- null fill on a shape that should take clicks or drops (use `Color.TRANSPARENT` instead)
- a higher or overlay that isn't mouseTransparent, swallowing events
- `snapshot(null, null)` for the drag image resulting in a white box around rounded shapes
- nothing put on the Dragboard, so the drag never starts
- accepting in `DRAW_OVER` but forgetting `setDropCompleted`
- game rules in the drop handler instead of in the game logic
- moving nodes by hand after a drop instead of redrawing from the game state
- using a loop variable directly inside a handler lambda