

JavaFX Controls

- last time
 - event loop
 - handlers
 - scene graph
 - don't block the UI thread
- today
 - which control to use
 - how to read its value
 - how to validate input
 - dialogs
 - a bit of layout

Controls

- each control is a Node in the scene graph
- controls are in the package `javafx.scene.control`
- four steps for every control –
 - create it
 - configure it
 - connect its handler
 - add it to a layout, then add the layout to the scene

```
Button start = new Button("Start Game"); // 1. create
start.setDefaultButton(true);           // 2. configure
start.setOnAction(e -> startGame());    // 3. connect handler
VBox root = new VBox(10, start);        // 4. add to a layout
stage.setScene(new Scene(root, 300, 200));
stage.show();
```

- a particular instance of a node can only appear once in the scene graph
 - can't add the same node to two parents

Choosing Controls

- choose controls by interaction need rather than appearance
 - ask “what does the user need to do?” before “which control looks right?”

need	interaction	controls
display	show information (no interaction)	Label, ImageView, ProgressBar, Separator, Tooltip
input	user supplies a value	TextField, PasswordField, TextArea, Spinner, Slider
action	user triggers something	Button, MenuItem, Hyperlink
selection	user chooses among options	CheckBox, RadioButton, ComboBox, ChoiceBox, ListView

- constrain, don't validate
 - if you can list the valid values, use a selection control rather than a text field

Display Controls

- Label – text for display and the caption for input fields
- ImageView, ProgressBar, Separator – pictures, progress, grouping
- Tooltip – extra help on hover
 - not present on all devices and easy to miss
 - supplemental only – should never be the only place critical information appears

Action Controls

- **Button** – one clear action per button
 - if a button commits an action, label it with a descriptive verb
 - people don't read dialogs – “OK” forces them back to the message to figure out what “OK” means
 - OK/cancel get ambiguous fast – dialog asks “Cancel your order?”, what do “OK” and “Cancel” mean then?
 - specific labels make consequences visible – “Delete”, “Overwrite”, “Send now” signals import more than “OK”
 - “Submit” describes the system, not the user's goal – they aren't submitting, they are placing an order, creating an account, ...
 - if it only dismisses information, “OK” is fine
 - e.g. “File saved” or error message with only a single button
 - `setDefaultButton(true)` – Enter activates it
 - `setCancelButton(true)` – Esc activates it
- **MenuBar, Menu, MenuItem** – for commands that don't need to be visible at all times
- **HyperLink** – link-like actions

31

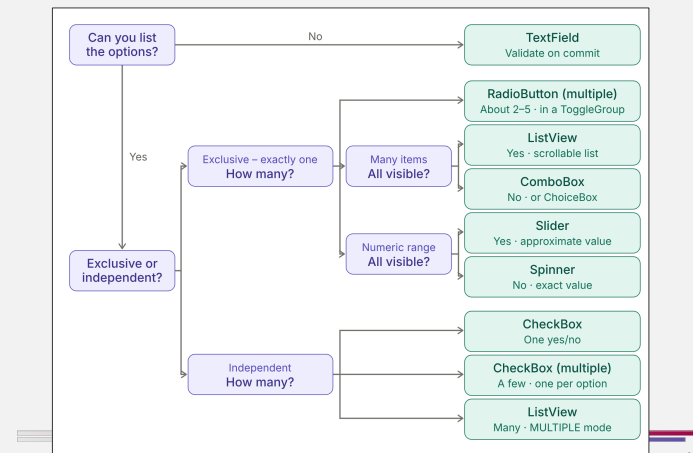
Text Input Controls

- **TextField** – one line
- **PasswordField** – one line, masks characters
- **TextArea** – multiple lines
 - `setWrapText(true)` – whether text autowraps
- `getText()` always returns a `String`, even for numbers – parse it yourself
- `trim()` before using the text
 - decide what an empty field means e.g. select the default, not just a missing value
- pressing Enter in a `TextField` fires its `setOnAction` handler
- `setPromptText(“e.g. 12345”)` is a hint that vanishes once typing starts
 - not a substitute for a `Label`!

CPSC 329: Software Development • Fall 2026

32

Selection Controls



CPSC 329: Software Development • Fall 2026

33

Selection Controls

- **ComboBox vs ChoiceBox**
 - `ComboBox` is more flexible – can be editable, supports custom display
 - `ChoiceBox` is the minimal version
 - fine to always use `ComboBox`
- radio buttons only make sense inside a `ToggleGroup`
 - without one, each button acts independently
- always set a default selection
 - otherwise “nothing selected” becomes a state to handle

CPSC 329: Software Development • Fall 2026

34

Reading Selection Values

- the handler reads state out of controls
 - the code pulls values from the controls and doesn't track them itself
- read values when the user commits (e.g. presses Start), not on every keystroke or click
 - reacting to every change needs listeners and bindings

CheckBox	isSelected()
RadioButton	isSelected() or toggleGroup.getSelectedToggle()
ComboBox<T>	getValue()
ListView<T>	getSelectionModel().getSelectedItem()
Spinner<T>	getValue()
Slider	getValue()

- watch for null – nothing selected results in null for ComboBox and ListView

35

Controls Holding Objects

- ComboBox<T> displays each (non-null) item's toString() unless otherwise specified
 - for text – a StringConverter tells the control how to display an item – “here's how to turn an item into text”
 - for pictures, colors, or a custom layout in the drop-down list – setCellFactory(...) hands the control a factory for the cells it draws – “here's how to make the things that draw items”
 - the closed ComboBox still uses the converter or toString()

CPSC 329: Software Development • Fall 2026

36

Controls Holding Objects

```
ComboBox<CardBack> backs = new ComboBox<>();
backs.getItems().addAll(CardBack.values());
backs.setConverter(new StringConverter<CardBack>() {
    public String toString(CardBack b) { return b.getDisplayName(); }
    public CardBack fromString(String s) { return null; } // not editable
});
```

```
ComboBox<CardBack> backs = new ComboBox<>();
backs.getItems().addAll(CardBack.values());
backs.setCellFactory(list -> new ListCell<CardBack>() {
    protected void updateItem(CardBack b, boolean empty) {
        super.updateItem(b, empty);
        if (empty || b == null) { // reused cell with no item
            setText(null);
            setGraphic(null);
        } else {
            setText(b.getDisplayName());
            setGraphic(new Rectangle(16, 16, b.getColor()));
        }
    }
});
// same look when closed
backs.setButtonCell(backs.getCellFactory().call(null));
```

CPSC 329: Software Development • Fall 2026

37

Which control?

- choose Draw-1 or Draw-3
- turn the timer on or off
- choose a card back from 12 designs
- enter a deal number
- pick one saved game to resume from a long list
- confirm abandoning the current game
- show the current move count

CPSC 329: Software Development • Fall 2026

38

Validating Text Input

- prefer constraint over validation
- preference order
 - constrain – use Spinner, Slider, ComboBox so invalid input is impossible
 - filter keystrokes – TextFormatter silently blocks invalid characters
 - bad input just vanishes with no feedback to the user on what is wrong
 - validate on commit and explain – parse, catch the error, say what's wrong next to the field

```
TextField seed = new TextField();

// accept edit only if the resulting text is all digits
seed.setTextFormatter(new TextFormatter<String>(change ->
    change.getControlNewText().matches("\\d*") ? change : null));
```

Validating Text Input

- good error messages include what is wrong and an example
 - “Seed must be a whole number e.g. 12345”
 - combine filtering with commit validation with a label or prompt text stating the rule
- don't rely on color alone to show an error
- put the parsing logic in a plain Java method rather than the handler
 - so it is unit-testable without JavaFX

Disabling vs Hiding

- `setDisable(true)` – visible but grayed out and unresponsive
 - still takes up space
 - disabling a container disables everything in inside it
- `setVisible(false)` – invisible but still takes up space
 - children aren't drawn if parent is invisible
- `setManaged(false)` – removed from the layout so neighbors move to fill the space
 - container takes no space so everything inside also takes none

Disabling vs Hiding

- disable when the option exists but doesn't apply right now
 - layout stays put and users can see the option exists
- hide (invisible and unmanaged) only what never applies in this context
 - decided once when the window is built so nothing jumps while the user works
- when you disable something, make the reason visible nearby
 - a disabled control with no explanation is frustrating
 - OK if there's an obvious nearby answer e.g. adjacent checkbox is unticked
 - display a short label or message
 - tooltips can add detail but shouldn't be the only explanation
- don't use disabling to silently enforce validation
 - showing why an input is wrong is usually better just having a grayed-out button reflecting that *something* is wrong

Dialogs

- a modal dialog blocks interaction with its owner window until it is closed
- use (only) for decisions that must be made or messages that require acknowledgement
 - e.g. confirm before something destructive
 - use sparingly – dialogs disrupt
- confirm everything trains users to click through without reading
 - put an inline message in the window for routine feedback

Dialogs

- Alert – display INFORMATION, WARNING, CONFIRMATION, ERROR
- TextInputDialog, ChoiceDialog<T> – ask for one value
- FileChooser / DirectoryChooser – return a File (null if cancelled)
- Dialog<R> – custom dialogs
- showAndWait() and related showX() methods – returns an Optional holding the result
 - Optional<T> is a container that holds either one value of type T or nothing
 - meant for return values – not to be used for instance variables, parameters, or collections
 - user can cancel or close the window, so there's always a “nothing chosen” case to handle
- give the dialog an owner so it appears over its window
 - initOwner(window) – typically the stage but can be any window

```
Alert confirm = new Alert(Alert.AlertType.CONFIRMATION,
    "Abandon the current game?", ButtonType.YES, ButtonType.NO);
confirm.setTitle("New Game");
confirm.initOwner(stage);
Optional<ButtonType> result = confirm.showAndWait();
if (result.isPresent() && result.get() == ButtonType.YES) {
    startNewGame();
}

File file = new FileChooser().showOpenDialog(stage); // null if the
user cancels
```

VBox and HBox

- layout panes
 - VBox stacks the children vertically
 - HBox places the children in a row
 - nest to get rows in a column
- new VBox(10, a, b, c)
 - first argument is the spacing between the children, followed by the children
- setPadding(new Insets(15))
 - space between the container's edge and its contents
- setAlignment(Pos.CENTER_RIGHT)
 - where the children sit inside the container (vertical and horizontal)