CISC 3120

C30c: Model-View-Controller and Writing Larger JavaFX Apps

Hui Chen

Department of Computer & Information Science

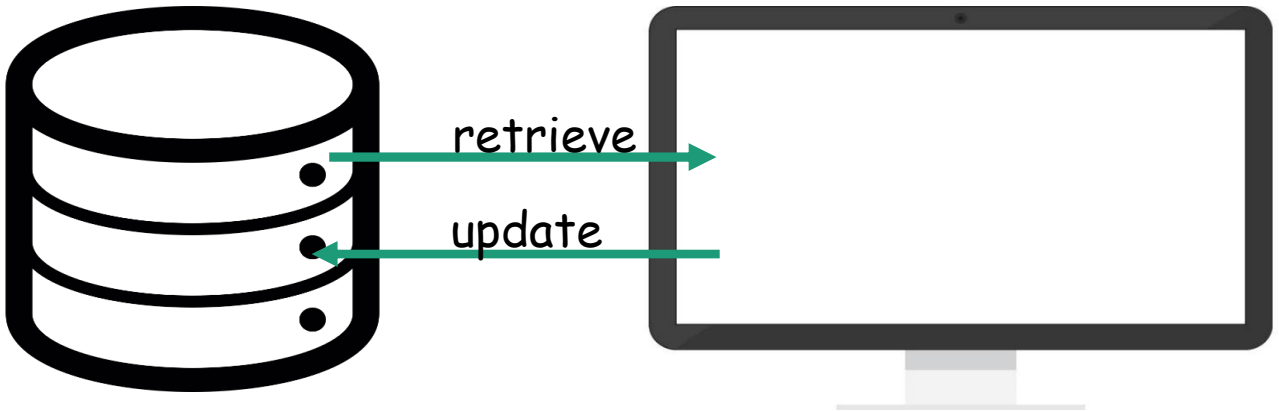
CUNY Brooklyn College

Outline

- Model-View-Controller pattern
- Writing larger JavaFX application

Applications: Data and View

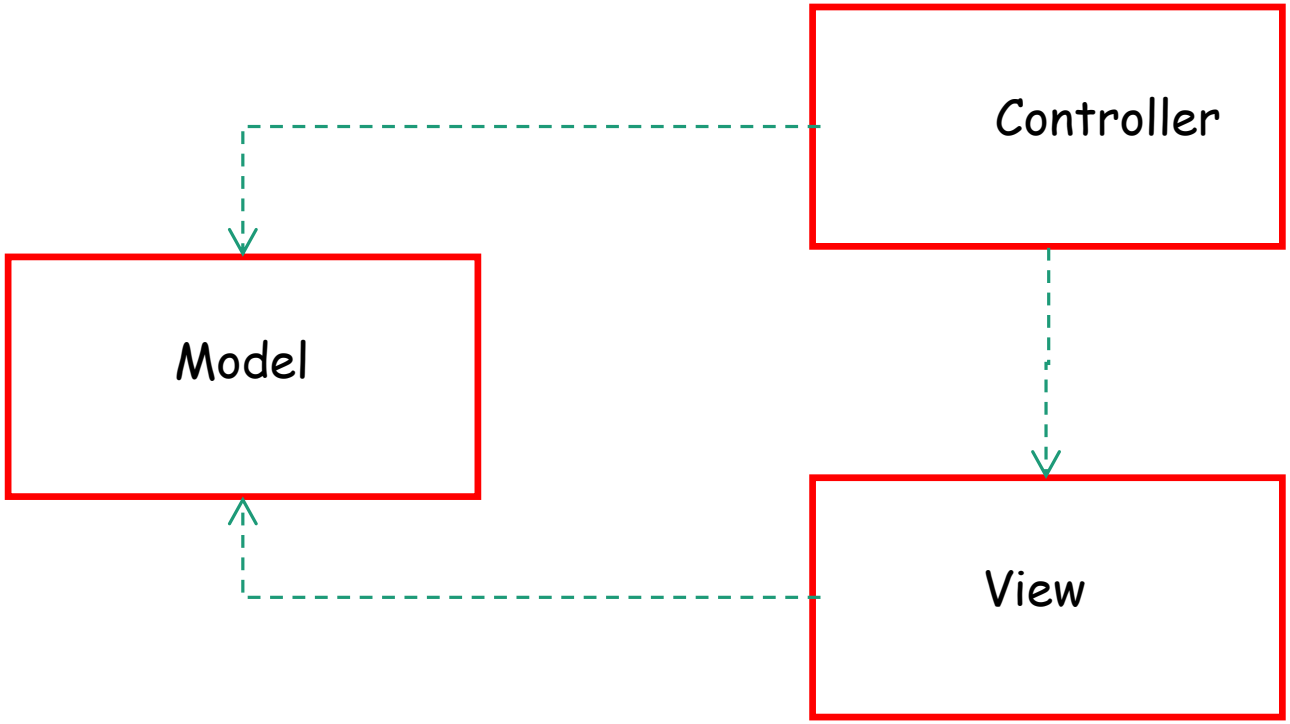
- Many computer systems are to display data and update data in a data store
- Two sets of terms
  - Business logic: the modeling of the application domain (model, data, business logic)
  - User interfaces: the visual feedback to the users (view, presentation)



Need to Separate Concerns

- User interfaces often changes more frequently than business logic
- Applications may display the same set of data differently
- User interface design and application logic design require different skill sets
  - User interface design and user interface development are two different concepts
- User interface code tends to be more device-dependent than business logic
- Create automatic tests for user interfaces is generally more difficult and time-consuming than business logic

Model-View-Controller



Model-View-Controller

- It separates an application three separated components/classes,
  - (Model) the modeling of the application domain
  - (View) the presentation,
  - (Controller) and the actions based on user input
- A fundamental design pattern for the separation of user interface from business logic.

Model

- Model is independent of view and controller
- Manages the behavior and data of the application domain
- Responds to requests for information about its state (usually from the view)
- Responds to instructions to change state (usually from the controller).

View

- Depends on model, but is independent of the controller
- Manages the display of information.

Controller

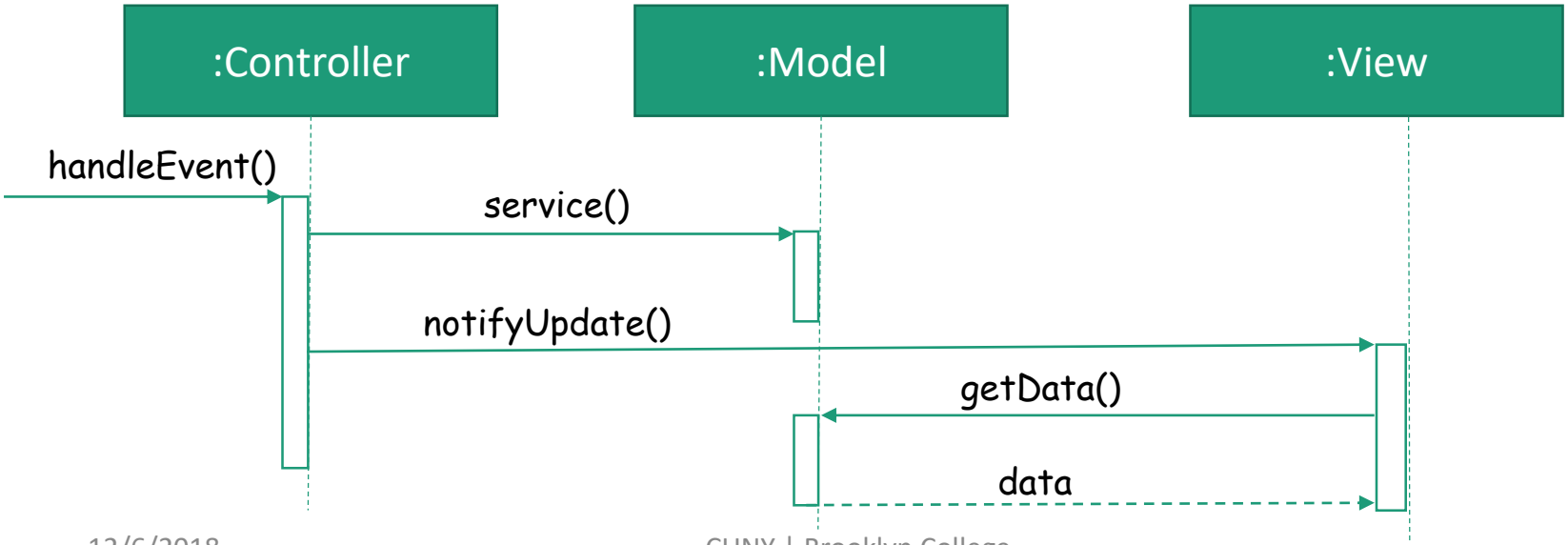
- Depends on both model and view
- Interprets the mouse and keyboard inputs from the user
- Inform the model and/or the view to change as appropriate.

MVC Models

- Passive model
- Active model

Passive MVC Model

- One controller manipulates the model exclusively
  - updates the model (data)
  - inform the view that the model has changed and request the view to refresh

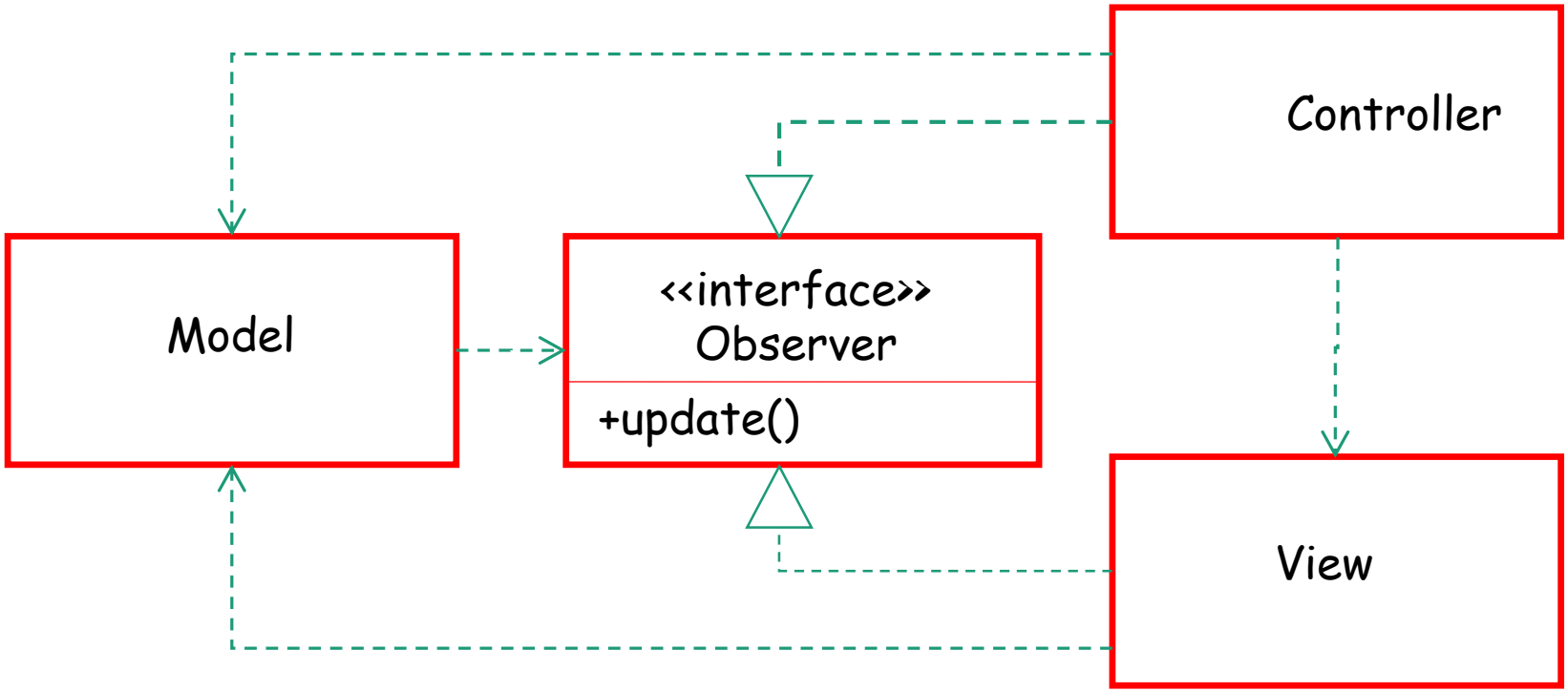


Passive MVC Model: Discussion

- Model is passive
  - Model is completely independent of View and Controller
  - Model does not notify View or Controller any changes on it
  - Controller is responsible for updating model, and for requesting view to refresh
- Often realized via dependency injection
- Limitation
  - If model can be updated from multiple controllers, the view may be out-of-date

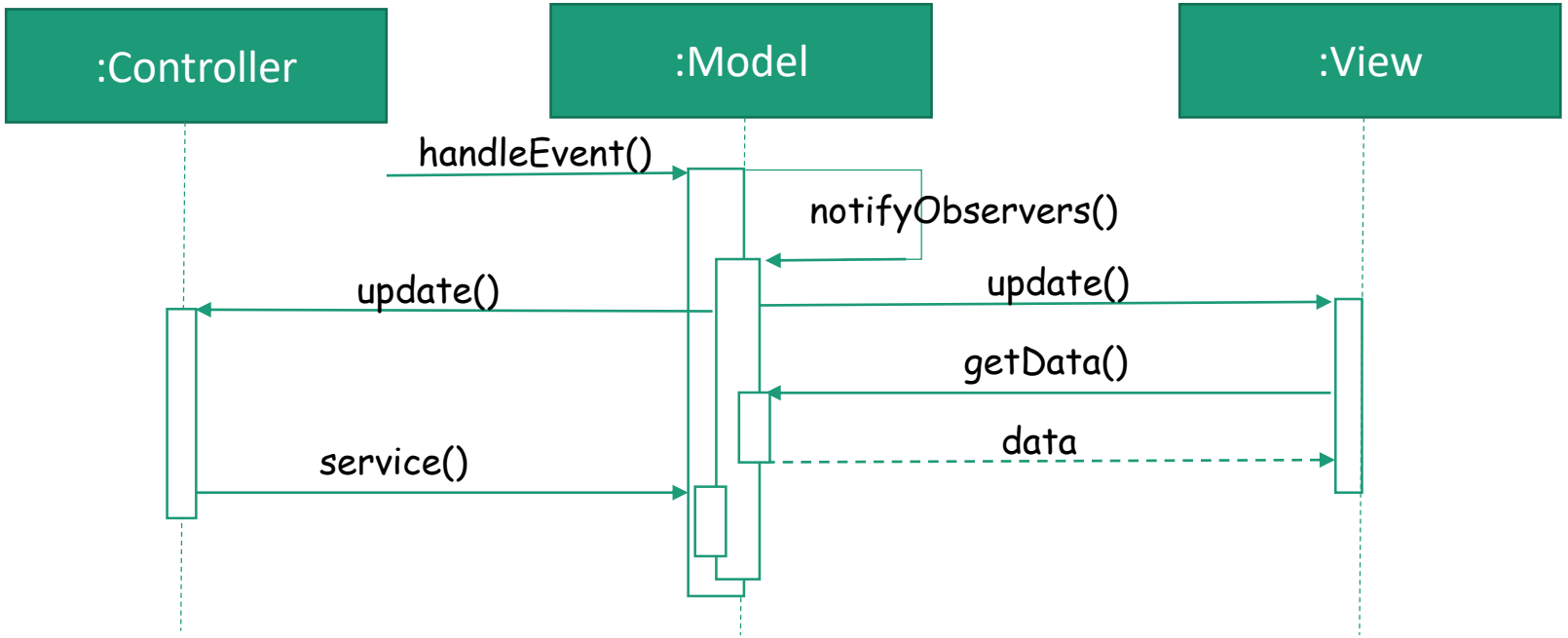
Active MVC Model

- Introduce an observer



Model and View Interaction

- Separation of concerns: code/logic are separated, but the objects interact



Active MVC Model Discussion

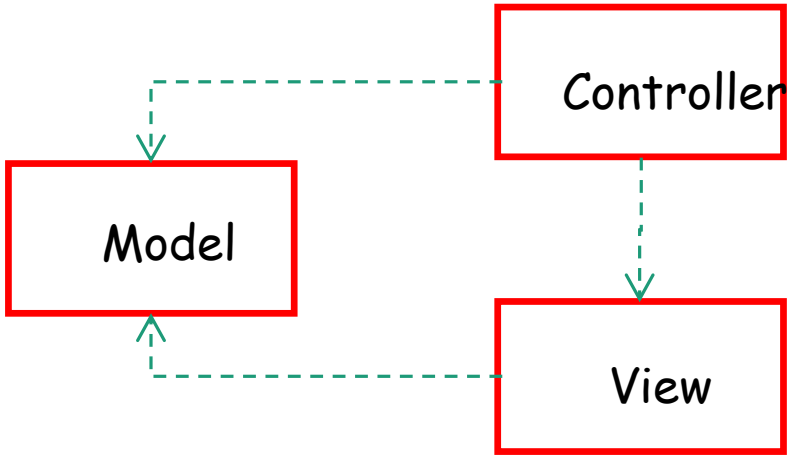
- Model is active
  - Model may change state without controller's involvement
    - e.g., in particular, when two or more sources may result in model update
- How do we separate model from view when model is active?
  - Model updates view
  - Realized via the Observer pattern
    - Model is an observable that notifies view or controller that is an observer
    - The model never requires specific information about any views
    - Controller or model implements the Observer whenever necessary
- Also called: the publish-subscribe pattern

Publish-Subscribe Pattern

- Subject: a subset of Observables in the model
- Subscriber: a subject's observer
- In an application that has multiple views, we often have multiple subjects
  - Each describe a specify type of model change
  - Each view can then subscribe only types of changes that are relevant to the view

Realizing MVC: Passive Model

- Three (categories) of classes
  - Controller class
  - Model class
  - View class



Passive MVC: Dependency Injection

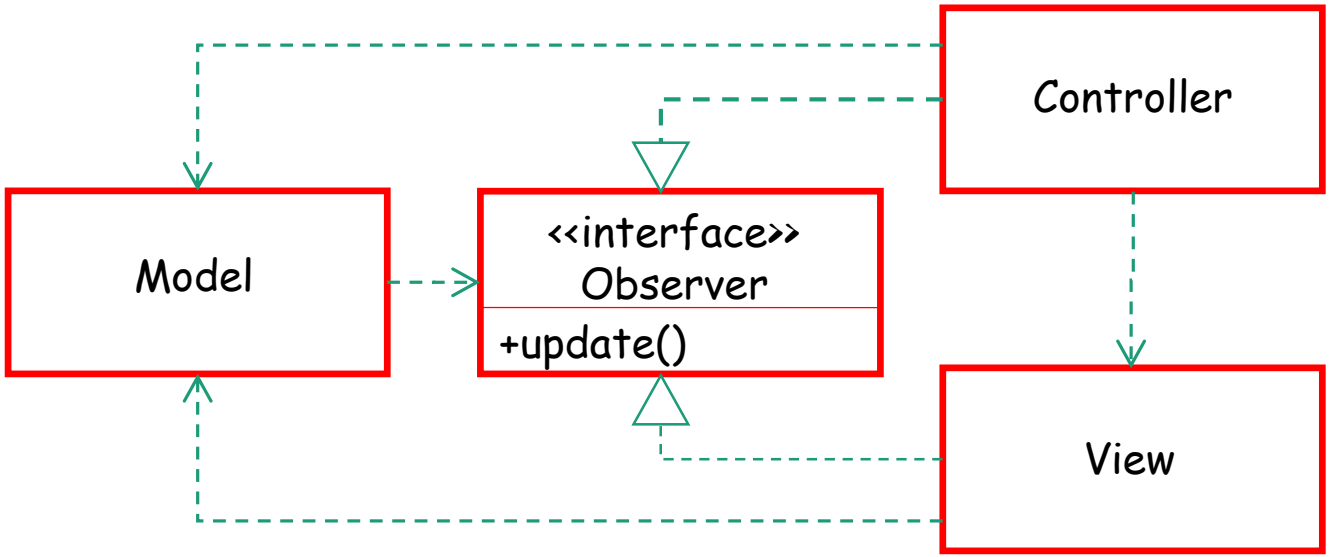
- Interpret dependency as association
- Model class is independent completely
  - No reference to either the Controller or the View class at all
- View depends on Model
  - The View class has instance variable that references to Model
- Controller depends on both View and Model
  - The Controller class has instance variables that reference to the Model and the View, respectively

Passive MVC: Dependency Injection

- Interpret dependency as a weaker dependency relationship than the association
- Model class is independently completely
  - No reference to either the Controller or the View class at all
- View depends on Model
  - The View class may not have instance variable that references to Model
  - A method of View has a parameter of the Model type (e.g., getData(Model m))
- Controller depends on both View and Model
  - The Controller class may not have instance variables that reference to either the Model or the View.
  - It has methods that requires parameters of either the Model or the View type, e.g., service(Model m), notifyUpdate(View v).

Active MVC: Publish-Subscribe

- Three (types) of classes: Model, View, and Controller
  - Model has instance variables to observables (so the Model is related to the Observer, via the platform)
  - View and controller implements the Observer interface



Active MVC: Publish-Subscribe

- Observer pattern via JavaFX Properties
- Model
  - Has instance variables references to subjects (JavaFX properties, or classes that wrap JavaFX properties)
- View and Controller
  - Has event listener either as instance method parameter or instance variables to listen to changes in Model
- Controller
  - Has references to model either as instance method parameter or instance variables (for update model)

Questions?

- How do we structure a GUI application? Is there a pattern to follow?

Write Larger JavaFX Applications

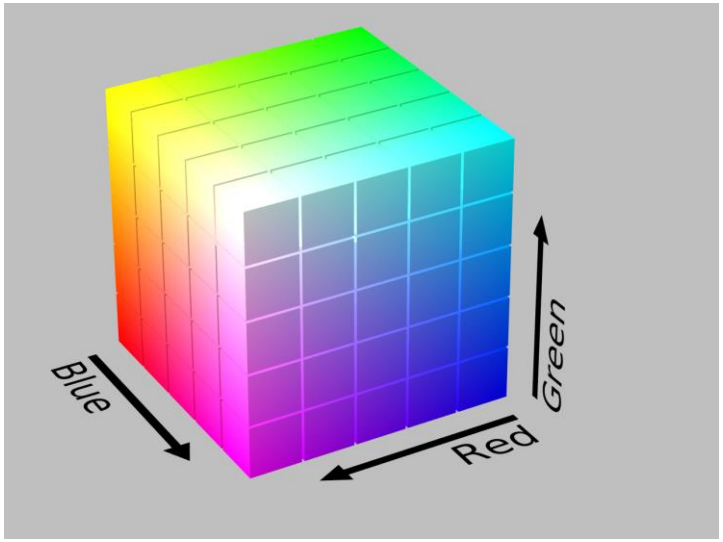
- Now, ready to engage in writing slightly larger applications in JavaFX
- A few essential concepts
  - Window & node coordinates, colors
- Use JavaFX build-in user interface components
- Design user interface and example application

Color Space

- Color is a human perception
- (Mathematical) models for color are developed
  - Including a model for human perceptual color space
  - Examples
    - Machine first
      - Additive: Red-Green-Blue (RGB)
      - Subtractive: Cyan-Magenta-Yellow-Black/Key (CMYK)
    - Human first
      - Hue-Saturation-Brightness (HSB)
    - Processing first
      - LAB (Luminance and a & b color channels)

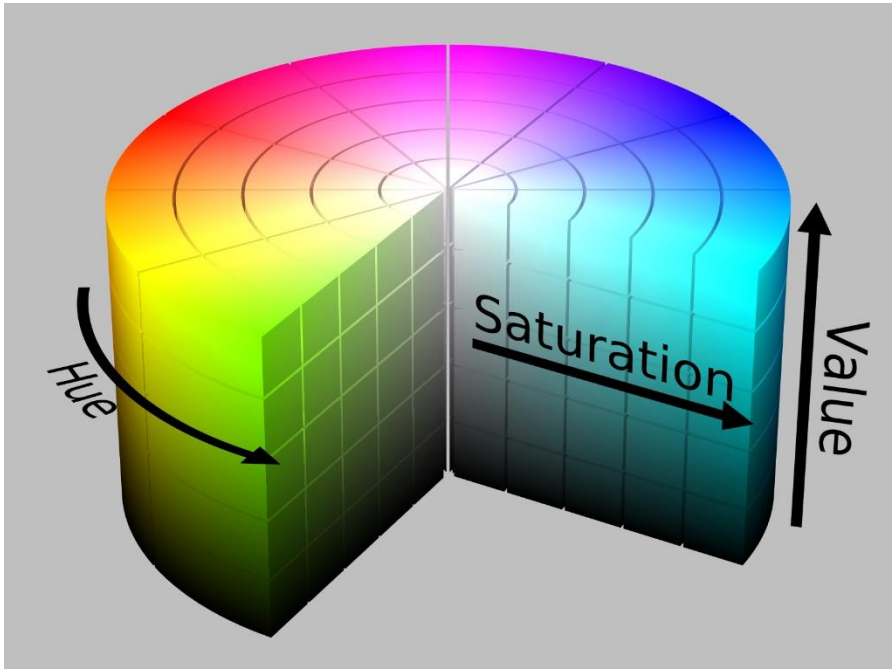
Standard Red-Green-Blue (sRGB)

- Red, Green, Blue
  - 0. – 1.
- Alpha (transparency or opacity)
  - 0.0 – 1.0 or 0 – 255; 1. or 255
  - 0. or 0: completely opaque
  - 1. or 1: completely transparent



Hue-Saturation-Brightness (HSB)

- Hue:
  - 0. – 360.
- Saturation:
  - 0. – 1.
- Brightness (or Value):
  - 0. – 1.
- Alpha (transparency or opacity)
  - 0.0 – 1.0 or 0 – 255; 1. or 255
  - 0. or 0: completely opaque
  - 1. or 1: completely transparent



Color and Static Factory Method

- A static method that returns an instance of the class
  - Examples:
    - static Color hsb(double hue, double saturation, double brightness, double opacity)
    - static Color rgb(int red, int green, int blue, double opacity)
- In your application design: advantage and disadvantage?

Blocking and Non-Blocking

- The show() method of a Stage object does not block the caller and returns "immediately".
- The showAndWait() method of a Stage object shows the stage and waits for it to be hidden (closed) before returning to the caller.
  - Cannot be called on the primary stage

Questions?

- Window coordinate system
- Blocking and non-blocking behaviors
- Color and color spaces

User Interface Components

- Layouts
- UI controls
- Text
- Canvas and Shapes
- Images
- Charts
- HTML content & embedded web browser
- Groups

Use Build-in UI Controls and Layouts

- Layout containers: prebuilt layouts for UI controls and more
- UI controls: prebuilt user interface controls
- Use texts
- Use 2D graphics
- Handle user interactions with simple event handlers

Layout Containers (Panes)

- Packaged in javafx.scene.layout
- Arrangements of the UI controls within a scene graph
- Provide the following common layout models

  - BorderPane
  - HBox
  - VBox
  - StackPane
  - GridPane
  - FlowPane
  - TilePane
  - AnchorPane

UI Controls

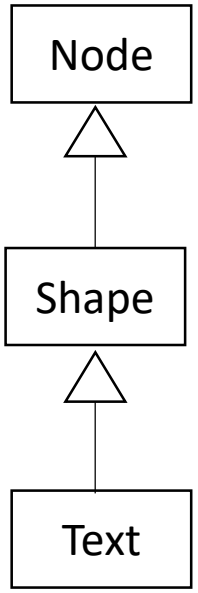
- Packaged in javafx.scene.control

- Label
- Button
- Radio Button
- Toggle Button
- Checkbox
- Choice Box
- Text Field
- Password Field
- Scroll Bar
- Scroll Pane
- List View
- Table View
- Tree View
- Tree Table View
- ComboBox
- Separator
- Slider
- Progress Bar
- Progress Indicator
- Hyperlink
- Tooltip
- HTML Editor
- Titled Pane
- Accordion
- Menu
- Color Picker
- Date Picker
- Pagination Control
- File Chooser

A Gallery of Selected UI Controls

Text

- Packaged in javafx.scene.text.Text



- Text class inherits from the Shape class, and the Shape class inherits from the Node class
  - You can apply effects, animation, and transformations to text nodes in the same way as to any other Nodes.
  - you can set a stroke or apply a fill setting to text nodes in the same way as to any other Shapes.

2-D Graphics

- Draw images on Canvas
  - Canvas
    - javafx.scene.canvas.Canvas
- Using a set of graphics commands provided by a GraphicsContext.
  - GraphicsContext
    - javafx.scene.canvas.GraphicsContext

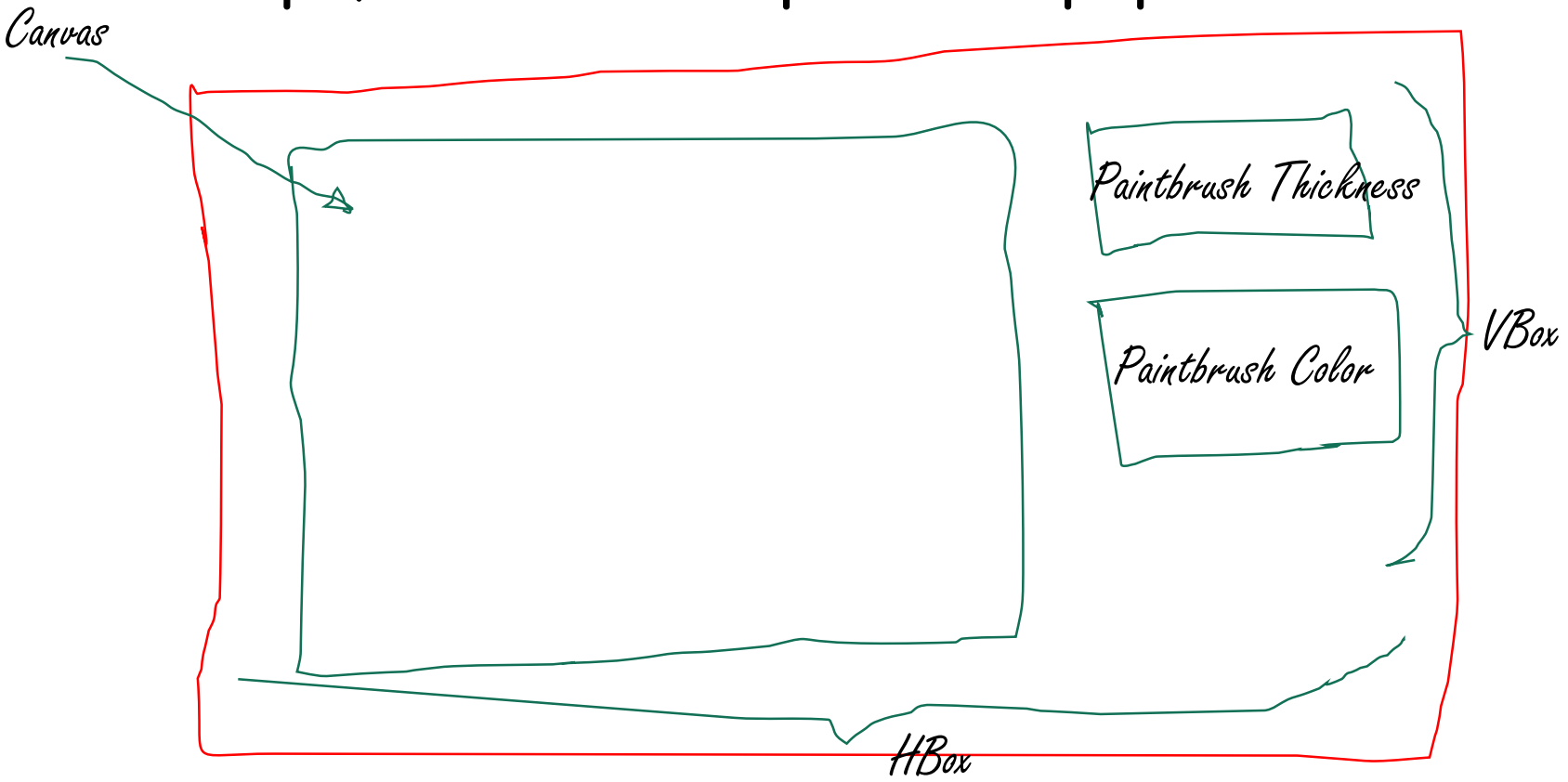
```
Canvas canvas = new Canvas(WIDTH, HEIGHT);
GraphicsContext gc = canvas.getGraphicsContext2D();
```

Use Build-in UI Controls and Layouts: Example

- Write a JavaFX application with build-in UI controls and layouts

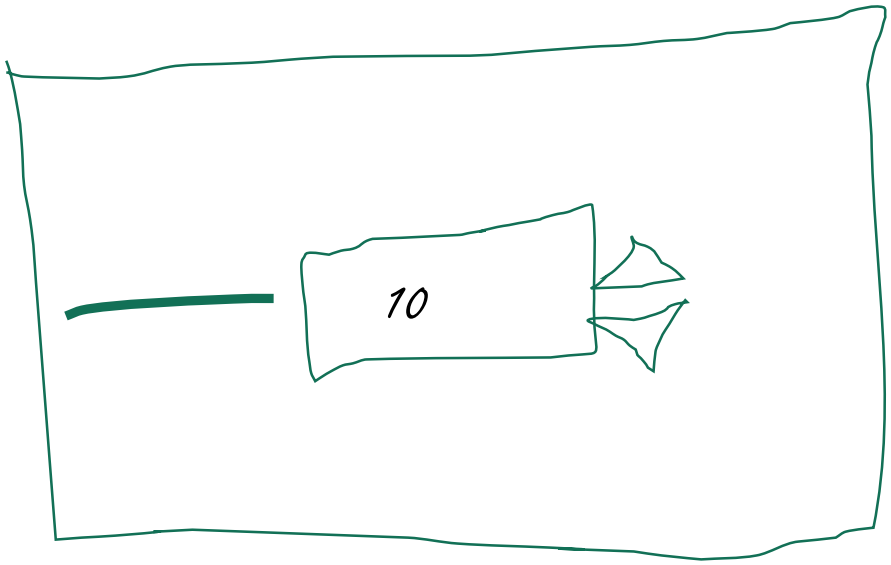
UI Design: Main Scene

- Perhaps, sketch on a piece of paper



UI Design: Brush Thickness

- Perhaps, sketch on a piece of paper



Questions?

- JavaFX build-in components
  - UI controls
  - Text
  - Layouts
  - UI design
- What available in JavaFX?
- Sample applications for exploring JavaFX features