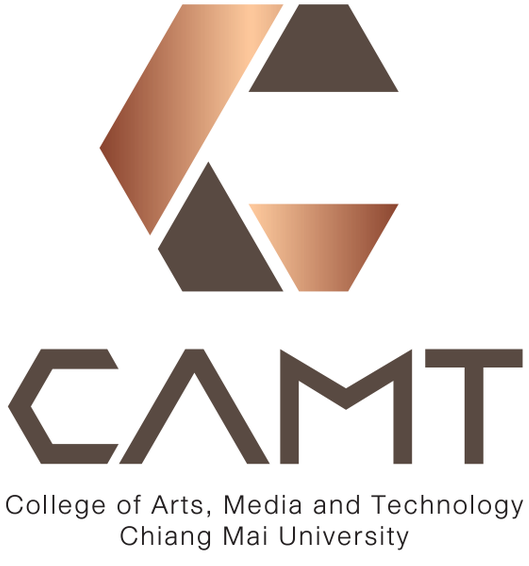
SE 233 Advanced Programming

Chapter I Event driven development



Lect Passakorn Phannachitta, D.Eng.

passakorn.p@cmu.ac.th

College of Arts, Media and Technology

Chiang Mai University, Chiangmai, Thailand

Agenda

- GUI
- Events
- Events driven programming

Overview

- Software has been more adapted to Graphic user interfaces (GUI)
- A GUI application has to be able to:
  - accept inputs from the user,
  - process the input, and
  - deliver the output as predefined by software requirements.

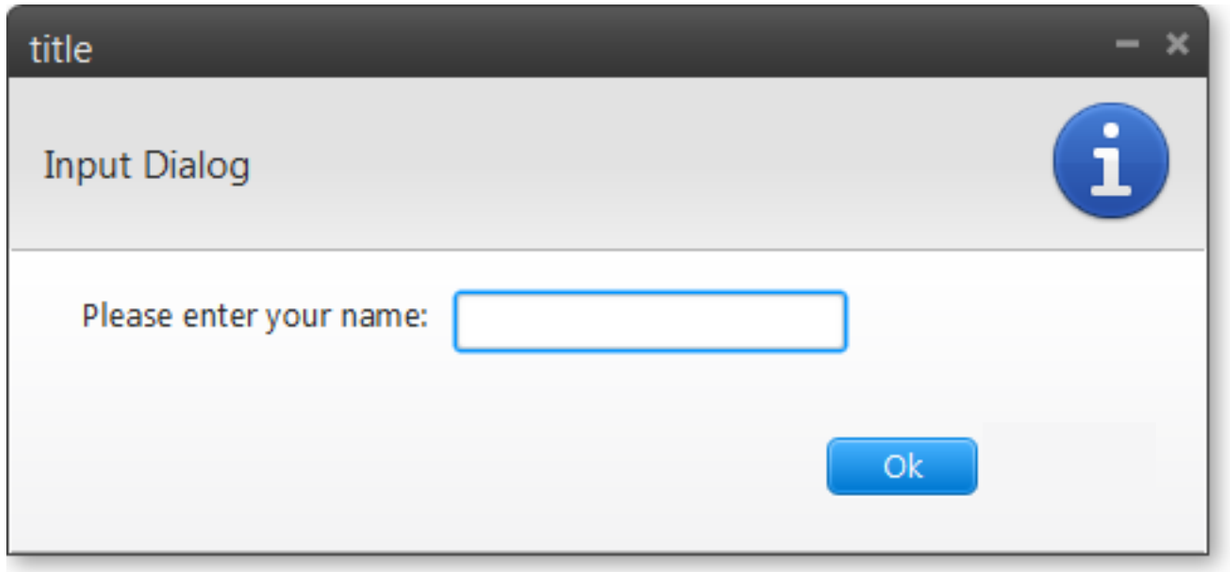
AS PROMPTLY AS POSSIBLE

Why is OOP necessary for GUI dev?

- **Class:** high-level abstract data types we created to mimic real-life thing
  - **Single responsibility**, **Self contained** and Reusable

Why is OOP Necessary for GUI

- **Object:** Data + Operations
- **Draw this UI:**

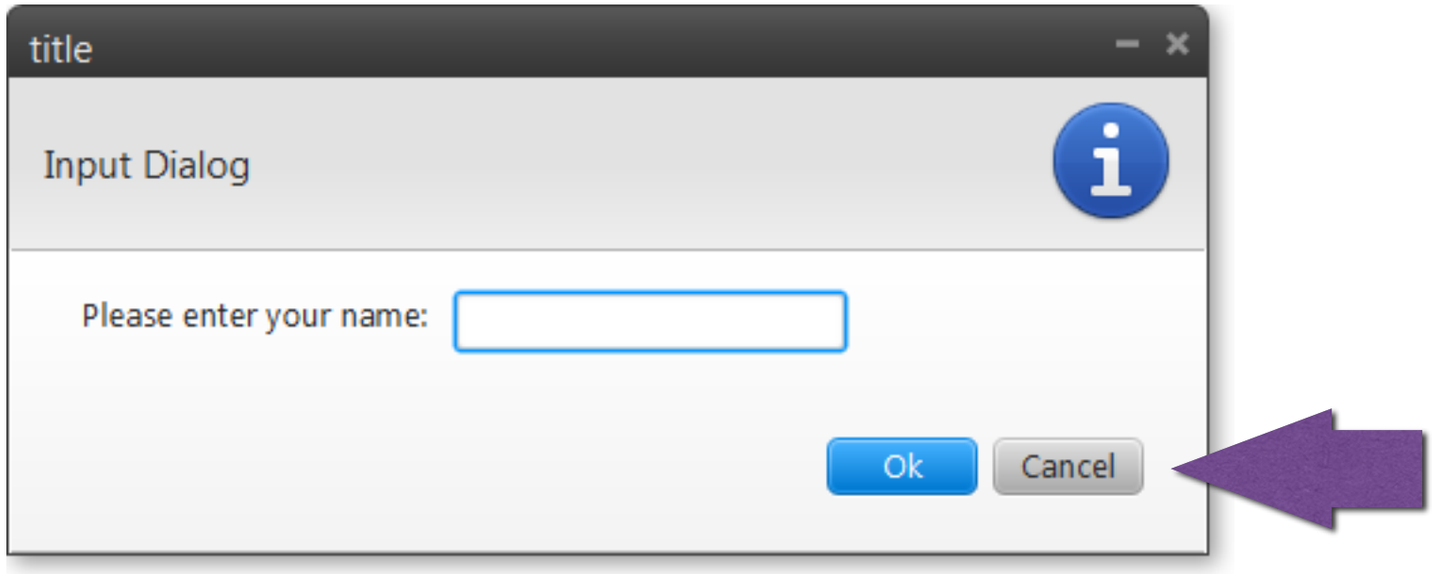


Why is OOP Necessary for GUI

- Without the OOP principle, e.g., to create the ok button
  - We have to code all from scratch !!!
    - Draw a rounded rectangle
    - Place in an (x, y) on the application window
    - Write the word ***Ok*** on that rectangle
    - Code about what shall be responded if we click on it
    - Perhaps, many more

Why is OOP Necessary for GUI

- **When the requirement is changed**
  - E.g., suppose that we have to add a cancel button



Why is OOP Necessary for GUI

- **Without the OOP principle, again**
  - **We have to code all from scratch !!!**
    - Draw a rounded rectangle
    - Place in an (x, y) on the application window
    - Write the word ***Cancel*** on that rectangle
    - Code about what shall be responded if we click on it
    - Perhaps, many more

Why is OOP Necessary for GUI

- One of the biggest concerns is

Even though we have done a lot for creating the ***OK*** button with much effort, we are unable to simply reuse anything for creating the ***Cancel*** button

How does OOP help?

- **With OOP**
  - We created a Button object for the **OK** button
    - The Round rectangle maybe inherited from a Shape object
      - Many actions can be inherited from parent classes
  - In the case of this **Cancel** button,
    - We may simply use the Button object, or
    - We may utilize the polymorphism principle

Why is OOP Necessary for GUI

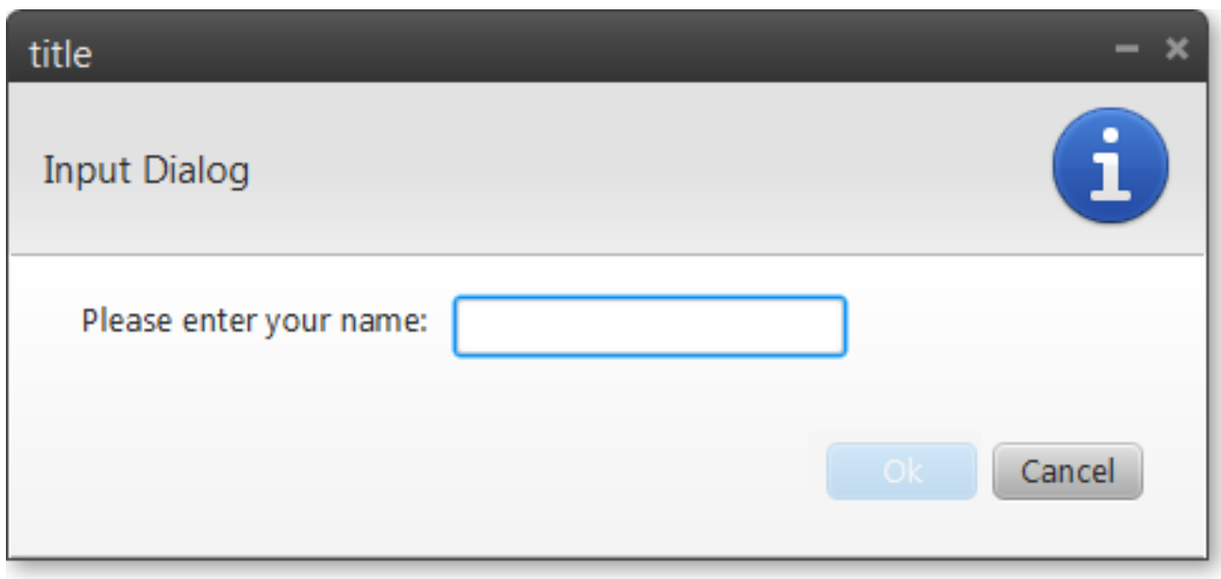
- One of the biggest gains is

This button object can be reused for any project in the future if we want

- E.g., BS4 button

Why is OOP Necessary for GUI

- **It is noteworthy that, recently, stuff related to Button has become much more complicated, e.g.,**
  - Conditional hide / show
  - Being shown but conditionally disabled / enabled
  - Message box
  - etc.



Which tool are we going to use ?

- JavaFX

JAVA + Adobe FLASH + Apache FLEX

Some (too brief :) history

- GUI in Java is commonly implemented by using two packages, namely Java.awt and Javax.swing.
- However, more recently, trends have been steadily reshaped to JavaFX for client application

Stated in the official website of JavaFX

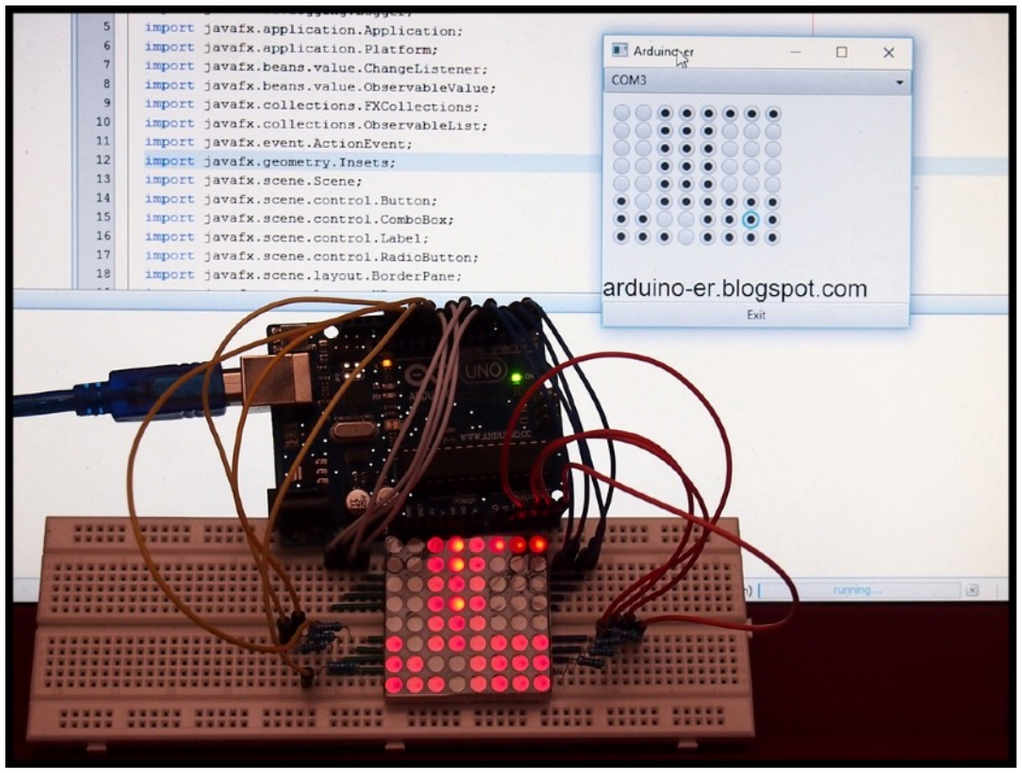
"It was emerged by a collaborative effort by many individuals and enterprises with the common goal of obtaining a modern, efficient, and fully featured toolkit for developing rich client applications. In this course, JavaFX will be used as a medium to study the GUI development and the Event driven development programming methodology"

The reasons that may make JavaFX popular

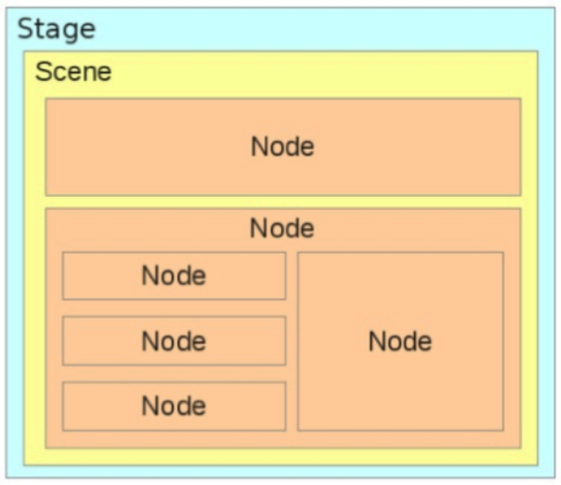
- 100% Java APIs compatible with millions Java library
- GUI development with cross platform
- Highly suitable for beginner — drag and drop component builders are available
- Can be used for desktop application, mobile application, web application, and embedded system.

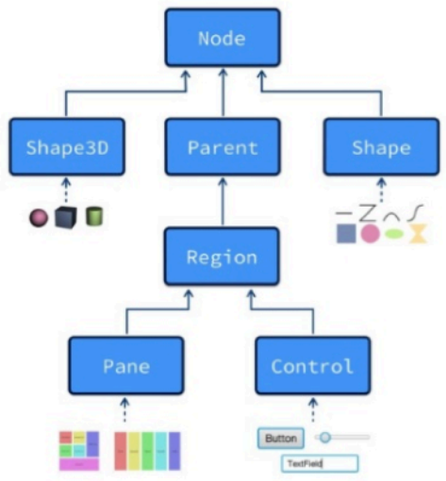
Where can we use JavaFX ?

- Basically everywhere
  - Desktop apps
  - Mobile apps
  - Web apps
  - Embedded system !



Anatomy





- GUI is constructed as a **scene graph** plus a collection of visual elements called **nodes**.
- All **nodes** are arranged hierarchically, analogous to how HTML elements are arranged in a DOM tree.

An example Hello World in JavaFX

```
+ 1  import javafx.application.Application;
+ 2  import javafx.scene.Scene;
+ 3  import javafx.stage.Stage;
+ 4  import javafx.scene.layout.StackPane;
+ 5  public class Launcher extends Application {
+ 6     @Override
+ 7     public void start(Stage primaryStage) throws Exception{
+ 8        StackPane root = new StackPane();
+ 9        primaryStage.setTitle("Hello World");
+ 10       primaryStage.setScene(new Scene(root, 300, 275));
+ 11       primaryStage.show();
+ 12    }
+ 13    public static void main(String[] args) {
+ 14       launch(args);
+ 15    }
+ 16 }
```

The launcher class must be Inherited from javafx.application

An example Hello World in JavaFX

```
+ 1  import javafx.application.Application;
+ 2  import javafx.scene.Scene;
+ 3  import javafx.stage.Stage;
+ 4  import javafx.scene.layout.StackPane;
+ 5  public class LaunchApp extends Application {
+ 6     @Override
+ 7     public void start(Stage primaryStage) throws Exception{
+ 8        StackPane root = new StackPane();
+ 9        primaryStage.setTitle("Hello World");
+ 10       primaryStage.setScene(new Scene(root, 300, 275));
+ 11       primaryStage.show();
+ 12    }
+ 13    public static void main(String[] args) {
+ 14       launch(args);
+ 15    }
+ 16 }
```

The overridden start method indicates the application's entry point

An example Hello World in JavaFX

```
+ 1  import javafx.application.Application;
+ 2  import javafx.scene.Scene;
+ 3  import javafx.stage.Stage;
+ 4  import javafx.scene.layout.StackPane;
+ 5  public class Launcher extends Application {
+ 6     @Override
+ 7     public void start(Stage primaryStage) throws Exception{
+ 8        StackPane root = new StackPane();
+ 9        primaryStage.setTitle("Hello World");
+ 10       primaryStage.setScene(new Scene(root, 300, 275));
+ 11       primaryStage.show();
+ 12    }
+ 13    public static void main(String[] args) {
+ 14       launch(args);
+ 15    }
+ 16 }
```

- JavaFX application's user interface container: Stage and Scene
- Scene is in Stage.
- Stage can be considered as the top-level container
- Scene is the container for any other visual elements

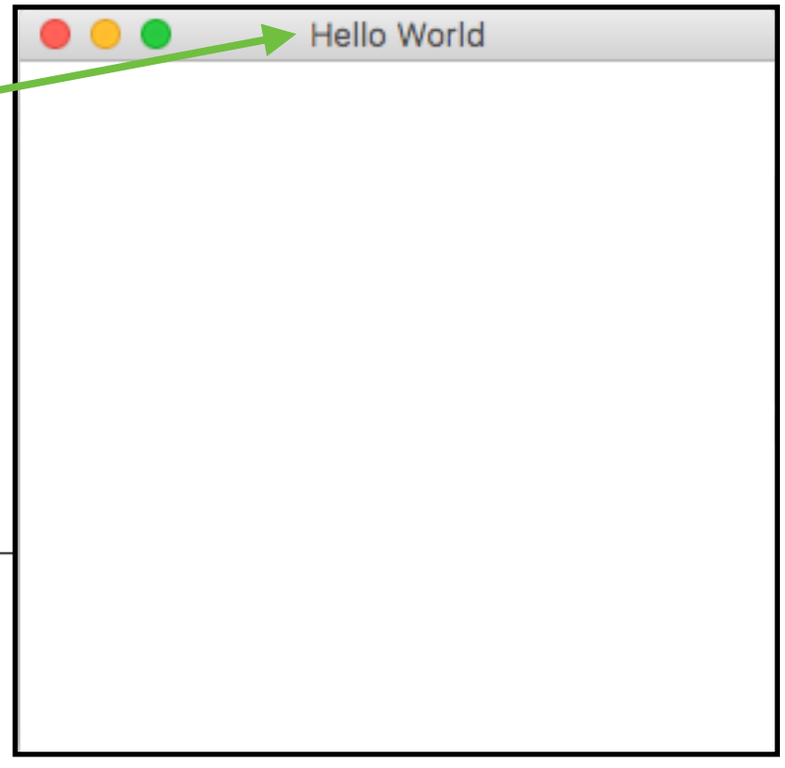
An example Hello World in JavaFX

```
+ 1  import javafx.application.Application;
+ 2  import javafx.scene.Scene;
+ 3  import javafx.stage.Stage;
+ 4  import javafx.scene.layout.StackPane;
+ 5  public class Launcher extends Applic[illegible]n {
+ 6     @Override
+ 7     public void start(Stage primar[illegible]tage) throws Exception{
+ 8        StackPane root = new StackPane();
+ 9        primaryStage.setTitle("Hello World");
+10        primaryStage.setScene(new Scene(root, 300, 275));
+11        primaryStage.show();
+12     }
+13     public static void main(String[] args) {
+14        launch(args);
+15     }
+16  }
```

In this example, a StackPane is the root node of the graph.

An example Hello World in JavaFX

```
+ 1  import javafx.application.Application;
+ 2  import javafx.scene.Scene;
+ 3  import javafx.stage.Stage;
+ 4  import javafx.scene.layout.StackPane;
+ 5  public class Launcher extends Application {
+ 6     @Override
+ 7     public void start(Stage primaryStage) throws Exception{
+ 8        StackPane root = new StackPane();
+ 9        primaryStage.setTitle("Hello World");
+10        primaryStage.setScene(new Scene(root, 300, 275));
+11        primaryStage.show();
+12     }
+13     public static void main(String[] args) {
+14        launch(args);
+15     }
+16  }
```



Adding a button

```
1  //Imports are omitted
2  public class Launcher extends Application {
3     @Override
4     public void start(Stage primaryStage) throws Exception{
5        Button btn = new Button();
6        btn.setText("Say Hello World");
7        StackPane root = new StackPane();
8        root.getChildren().add(btn);
9        primaryStage.setTitle("Hello World");
10       primaryStage.setScene(new Scene(root, 300, 275));
11       primaryStage.show();
12    }
13    public static void main(String[] args) {
14       launch(args);
15    }
16 }
```

- Create the visual element as a JavaFX node, i.e., a button;
- Then, append the created node to an existing node in the Application Scene graph.
  - xxxx.getChildren().add(**btn**)

What can be added?

Method Detail

getChildren

```
public ObservableList<Node> getChildren()
```

Description copied from class: `Parent`
Gets the list of children of this `Parent`.

See the class documentation for `Node` for scene graph structure restrictions on setting a `Parent`'s children list. If these restrictions are violated by a change to the list of children, the change is ignored and the previous value of the children list is restored. An `IllegalArgumentException` is thrown in this case.

If this `Parent` node is attached to a `Scene` attached to a `Window` that is showning (`Window.isShowing()`), then its list of children must only be modified on the JavaFX Application Thread. An `IllegalStateException` is thrown if this restriction is violated.

Note to subclasses: if you override this method, you must return from your implementation the result of calling this super method. The actual list instance returned from any getChildren() implementation must be the list owned and managed by this Parent. The only typical purpose for overriding this method is to promote the method to be public.

Overrides:

`getChildren in class Parent`

Returns:

`modifiable list of children.`

ObservableList<E>

Interface ObservableList<E>

Type Parameters:

`E - the list element type`

All Superinterfaces:

`Collection<E>, Iterable<E>, List<E>, Observable`

All Known Subinterfaces:

`ObservableListValue<E>, WritableListValue<E>`

Methods inherited from interface java.util.List

`add, add, addAll, addAll, clear, contains, containsAll, equals, get, hashCode, indexOf, isEmpty, iterator, lastIndexOf, listIterator, listIterator, remove, remove, removeAll, replaceAll, retainAll, set, size, sort, spliterator, subList, toArray, toArray`

Methods inherited from interface java.util.Collection

`parallelStream, removeIf, stream`

Methods inherited from interface java.lang.Iterable

`forEach`

Methods inherited from interface javafx.beans.Observable

`addListener, removeListener`

Events

- Foreground
  - E.g., clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page
- Background
  - E.g., The operating system interruptions, hardware or software failure, timer expiry, operation completion are the example of background events.

The main difference is <u>user interaction</u>

Events in JavaFX

- **Fundamental components**
  - **Event source** — a GUI component that user can interact with
  - **Event listener** — an object that listens to events from a specific GUI component
  - **Event handler** — a method that is created to process the input signal and generate the corresponding output.

- **Programmers must undertake two tasks**
  - Create an event source
  - Register event listener for the event source
  - Implement the event handler

Event handling example

```
 1  //Imports are omitted
 2  public class Launcher extends Application {
 3     @Override
 4     public void start(Stage primaryStage) throws Exception{
 5        Button btn = new Button();
 6        btn.setText("Say Hello World");
+7        btn.setOnAction(new EventHandler<ActionEvent>() {
+8           @Override
+9           public void handle(ActionEvent event) {
+10             System.out.println("Hello World!");
+11          }
+12       });
13        StackPane root = new StackPane();
14        root.getChildren().add(btn);
15        primaryStage.setTitle("Hello World");
16        primaryStage.setScene(new Scene(root, 300, 275));
17        primaryStage.show();
18    }
19    public static void main(String[] args) {
20       launch(args);
21    }
22 }
```

Event handler

Event handling example

```
 1  //Imports are omitted
 2  public class Launcher extends Application {
 3     @Override
 4     public void start(Stage primaryStage) throws Exception{
 5        Button btn = new Button();
 6        btn.setText("Say Hello World");
+7        btn.setOnAction(new EventHandler<ActionEvent>() {
+8           @Override
+9           public void handle(ActionEvent event) {
+10             System.out.println("Hello World!");
+11          }
+12       });
 13       StackPane root = new StackPane();
 14       root.getChildren().add(btn);
 15       primaryStage.setTitle("Hello World");
 16       primaryStage.setScene(new Scene(root, 300, 275));
 17       primaryStage.show();
 18    }
 19    public static void main(String[] args) {
 20       launch(args);
 21    }
 22 }
```

Polymorphism

(Alternative handle function)

Anonymous class

The code

```
new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        System.out.println("Hello World!");
    }
}
```

creates an **anonymous inner class** that implements EventHandler, and defines the handle method.

EventHandler is an interface, and this code does not create an instance of it, but an instance of the newly declared **anonymous class**.

Event driven programming

- Can be considered as a methodology to develop a pro- gram with numerous subprograms inside it.
- A subprogram will be launched after the arrival of a specific user signal meant for it.
- The flow of the program is determined by events such as actions (e.g., key press), sensor outputs, or messages from other program.
- It now become dominant paradigm used in almost every cutting-edge technology — Why?

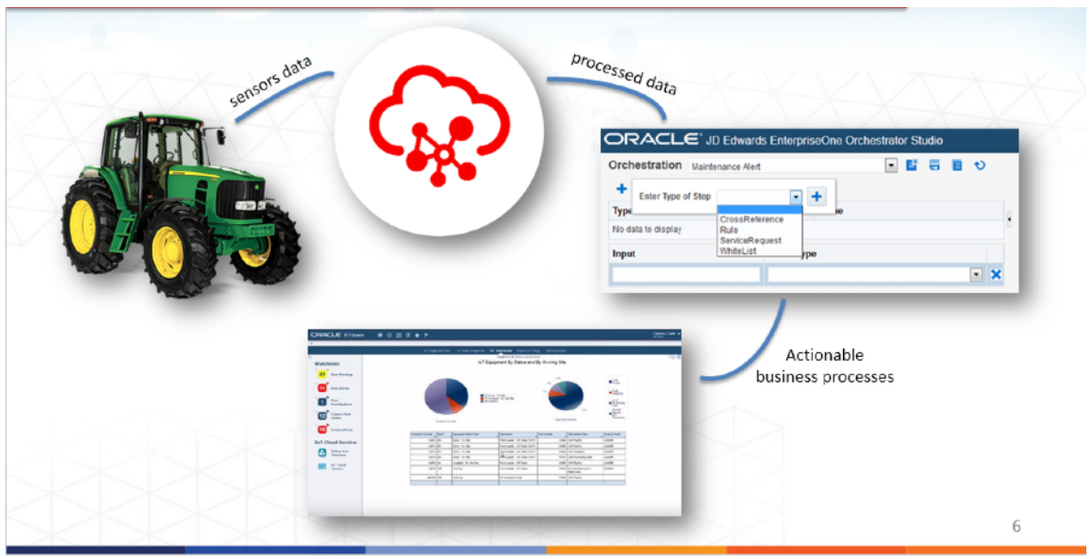
For example — in an action game,

- Moving a game character forward can be considered a subprogram of the entire game,
- It will only be executed after the user emits the particular signal, e.g., pressing of the key *w* on the keyboard
- This subprogram will not do anything the key *w* is pressed.
- Pressing other keys, e.g., spacebar, will trigger another subprogram that generates a different outcome.

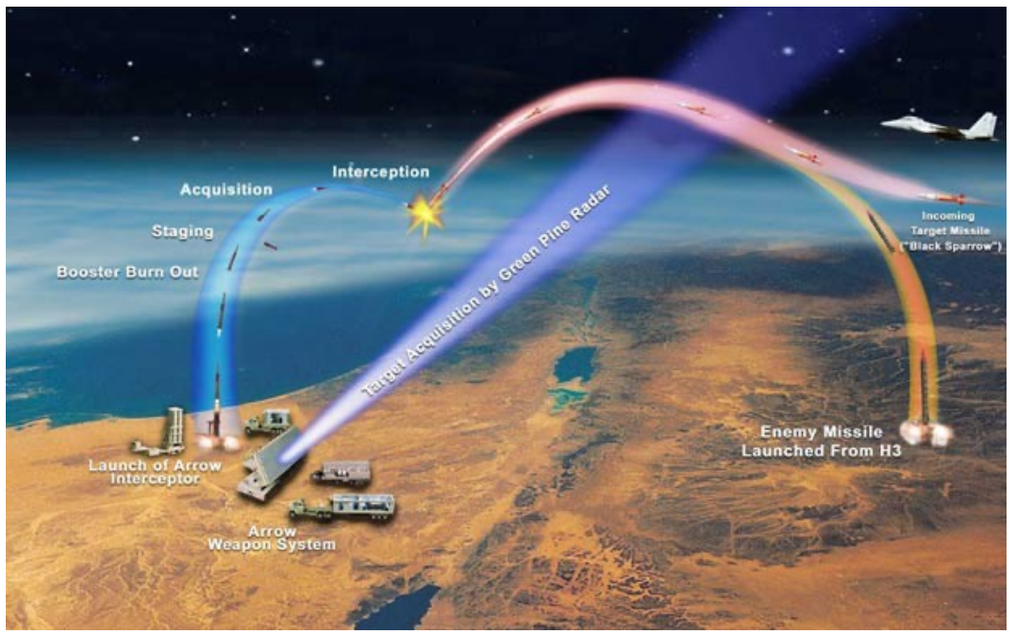
Event driven programming

- Automated technology harnessing AI, e.g., algorithmic trading

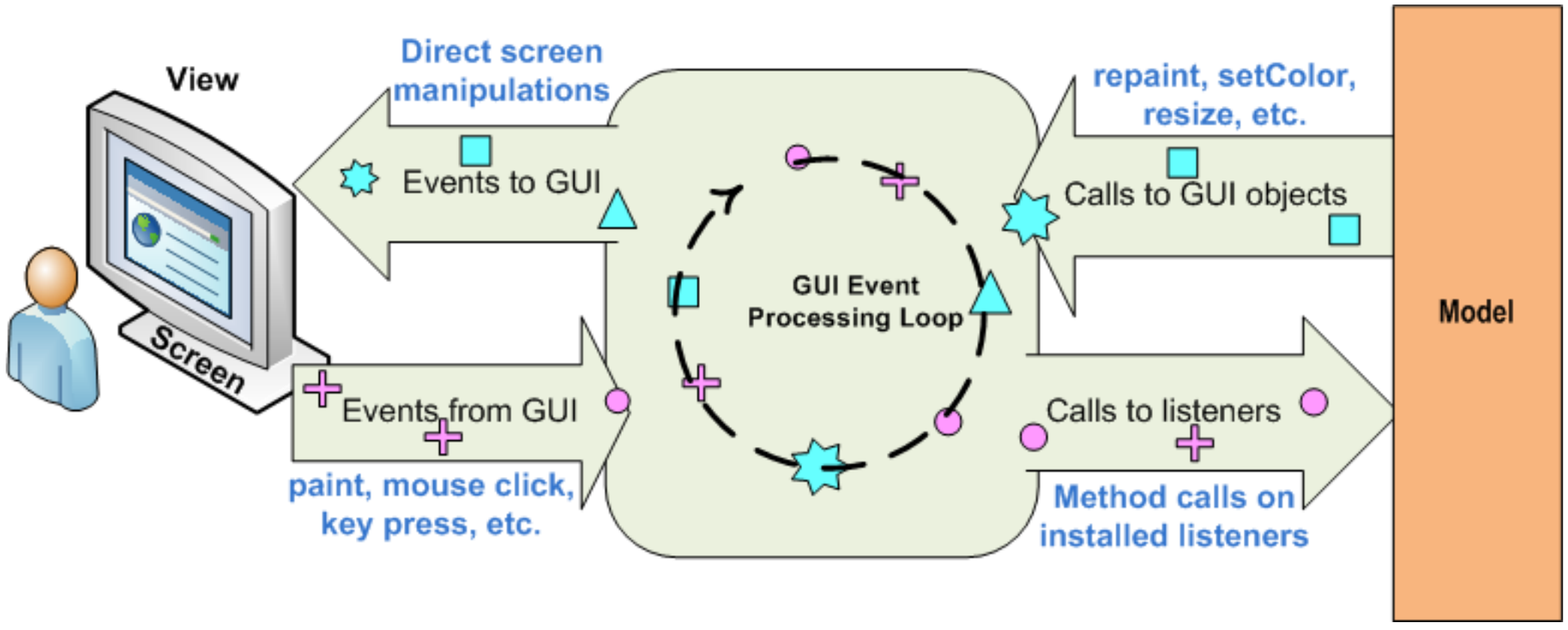
- IoT



- Defense System



Events and GUI



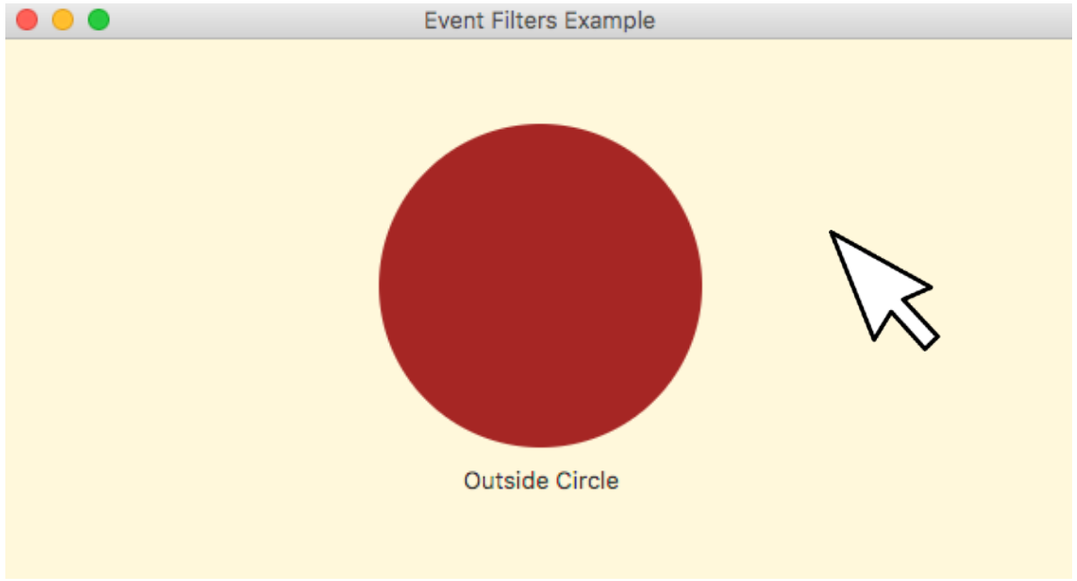
Ref https://www.clear.rice.edu/comp310/JavaResources/GUI/ (Retrieved May 2020)

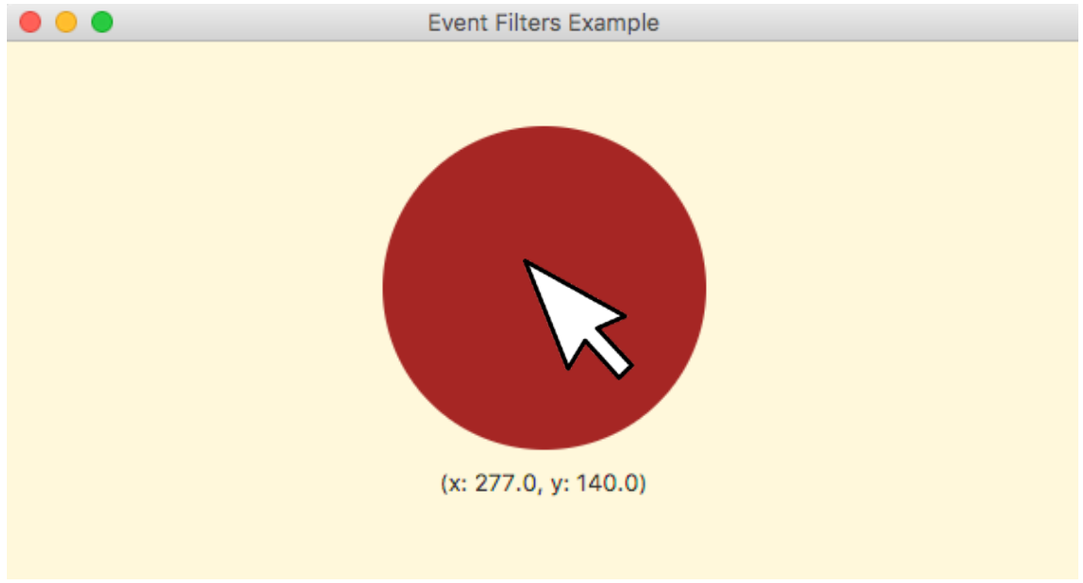
More example

```
13      StackPane root = new StackPane();
14      root.getChildren().add(btn);
15      primaryStage.setTitle("Hello Wor  );
-16     primaryStage.setScene(new Scen  ot, 300, 275
+17     Scene scene = new Scene(root  00, 275);
+18     scene.widthProperty().addListener(new ChangeListener<Number>() {
+19        @Override
+20        public void changed(ObservableValue<? extends Number> observableValue,
              Number oldSceneWidth, Number newSceneWidth) {
+21           System.out.println("Width: " + newSceneWidth);
+22        }
+23     });
+24     scene.heightProperty().addListener(new ChangeListener<Number>() {
+25        @Override
+26        public void changed(ObservableValue<? extends Number> observableValue,
             Number oldSceneHeight, Number newSceneHeight) {
+27           System.out.println("Height: " + newSceneHeight);
+28        }
+29     });
+30     primaryStage.setScene(scene);
31      primaryStage.show();
32   }
33   public static void main(String[] args) {
```

What will happen?

More example





State

- State is data or information that will be changed or manipulated throughout a program's runtime.
- For this example, we are interested in the mouse state at the time the pointer is inside the circle, and that of when it just moves out of the circle. These are states we are interested in.
- On the other hand, when the mouse pointer is anywhere outside the circle, we do not care.

More example

```
+17      Circle circle = new Circle(30);
+18      circle.setOnMouseMoved(new EventHandler<MouseEvent>() {
+19         @Override
+20         public void handle(MouseEvent event) {
+21            reporter.setText("(x: "+event.getX()+", y: " + event.getY()+")");
+22         }
+23      });
+24      circle.setOnMouseExited(new EventHandler<MouseEvent>() {
+25         @Override
+26         public void handle(MouseEvent event) {
+27            reporter.setText("Outside Circle");
+28         }
+29      });
+30      layout.getChildren().setAll(circle, reporter);
```

States of the circle we are interested in:

Only mouse move and mouse exit

More example

```
+17        Circle circle = new Circle(30);
+18        circle.setOnMouseMoved(new EventHandler<MouseEvent>() {
+19           @Override
+20           public void handle(MouseEvent event) {
+21              reporter.setText("(x: "+event.getX()+", y: " + event.getY()+")");
+22           }
+23        });
```

setOnMouseMoved

```
public final void setOnMouseMoved(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseMoved.

Property description:

Defines a function to be called when mouse cursor moves within this Node but no buttons have been pushed.

An anonymous class is created as to execute the sequence of commands when the mouse cursor moves within this node

Available mouse events

setOnMouseClicked

```
public final void setOnMouseClicked(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseClicked.

Property description:

Defines a function to be called when a mouse button has been clicked (pressed and released) on this Scene.

setOnMouseDragged

```
public final void setOnMouseDragged(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseDragged.

Property description:

Defines a function to be called when a mouse button is pressed on this Scene and then dragged.

setOnMouseEntered

```
public final void setOnMouseEntered(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseEntered.

Property description:

Defines a function to be called when the mouse enters this Scene.

Available mouse events

setOnMouseExited

```
public final void setOnMouseExited(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseExited.

Property description:

Defines a function to be called when the mouse exits this Scene.

setOnMouseMoved

```
public final void setOnMouseMoved(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseMoved.

Property description:

Defines a function to be called when mouse cursor moves within this Scene but no buttons have been pushed.

Available mouse events

setOnMousePressed

```
public final void setOnMousePressed(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMousePressed.

Property description:

Defines a function to be called when a mouse button has been pressed on this Scene.

setOnMouseReleased

```
public final void setOnMouseReleased(EventHandler<? super MouseEvent> value)
```

Sets the value of the property onMouseReleased.

Property description:

Defines a function to be called when a mouse button has been released on this Scene.

Available mouse events

setOnMouseDragEntered

```
public final void setOnMouseDragEntered(EventHandler<? super MouseDragEvent> value)
```

Sets the value of the property onMouseDragEntered.

Property description:

Defines a function to be called when a full press-drag-release gesture enters this Scene.

Since:

JavaFX 2.1

setOnMouseDragExited

```
public final void setOnMouseDragExited(EventHandler<? super MouseDragEvent> value)
```

Sets the value of the property onMouseDragExited.

Property description:

Defines a function to be called when a full press-drag-release gesture exits this Scene.

Since:

JavaFX 2.1

Available mouse events

setOnMouseDragOver

```
public final void setOnMouseDragOver(EventHandler<? super MouseDragEvent> value)
```

Sets the value of the property onMouseDragOver.

Property description:

Defines a function to be called when a full press-drag-release gesture progresses within this Scene.

Since:

JavaFX 2.1

setOnMouseDragReleased

```
public final void setOnMouseDragReleased(EventHandler<? super MouseDragEvent> value)
```

Sets the value of the property onMouseDragReleased.

Property description:

Defines a function to be called when a full press-drag-release gesture ends within this Scene.

Since:

JavaFX 2.1

Quick Question — How to drag & drop

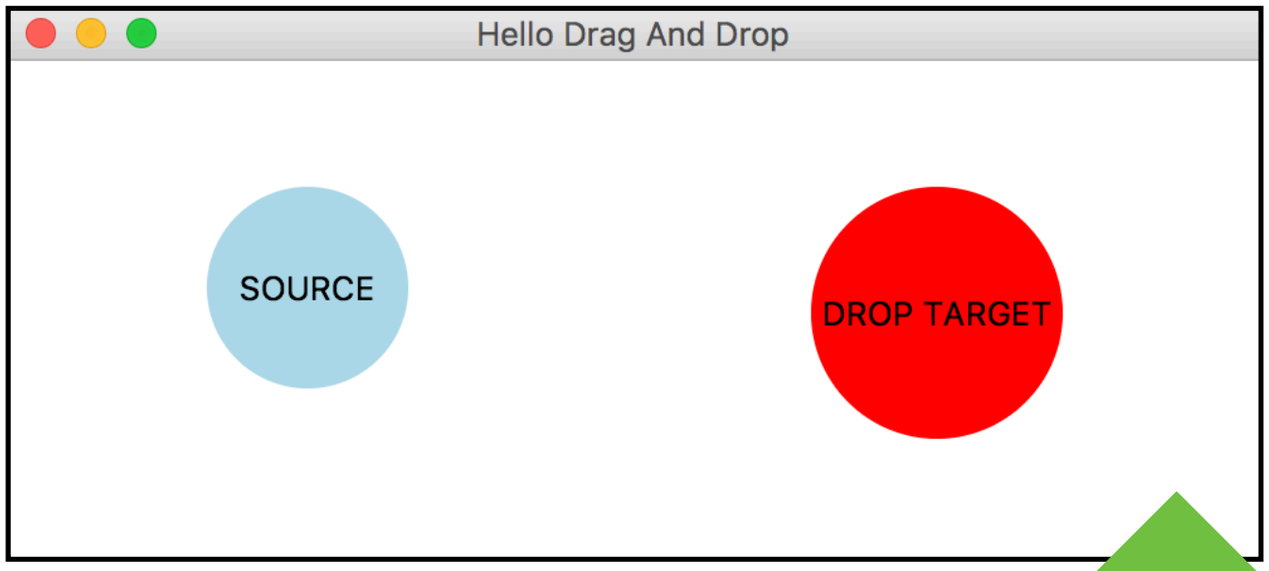
- One of the very best approaches to learn about various mouse events at once
- What/which kinds of event do we have to consider?

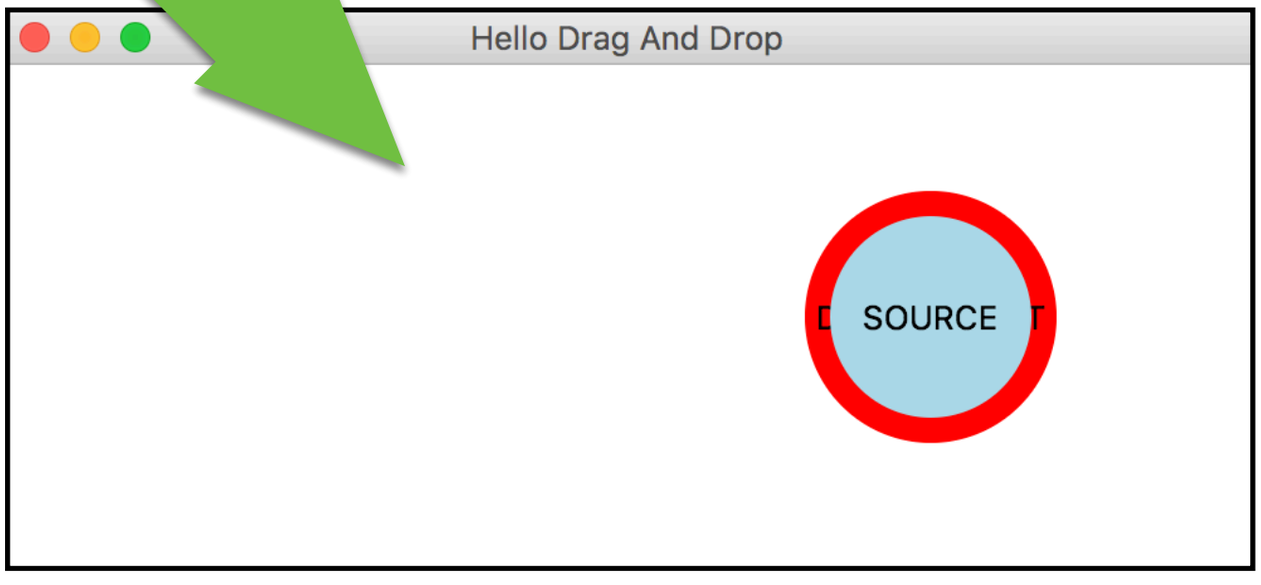
Drag & drop

- Key things to consider
  - What allows us to start dragging?
  - What can we drag?
  - How do we know that the mouse is on the object we want to drag?
  - What happen during the drag?
  - Will the object move along the mouse pointer?
  - When and how to stop dragging?
  - Does the distances of moving matter?
  - Does object coordinate matter?

Drag & drop

- A toy problem





Example requirements

1. There are two circles, the source is in blue, and the target is in red;

2. Once we click on the source, it will be brought to the front and moved with the mouse pointer;

3. If we drag the source and release the mouse button over the target, the source object will be snapped in front of the target object.

4. If we drag the source and release the mouse button outside the target, the source object moves back to the origin coordinate.

Requirement #1

- It will be more efficient to declare objects and make utilized it, if we may need more of these figures in the future

```
+ 1  //Imports are omitted
+ 2  public class FigureGroup {
+ 3     private StackPane figure;
+ 4     public FigureGroup(int x, int y, int radius, Color color, String str) {
+ 5        Circle circle = new Circle(radius, color);
+ 6        Text text = new Text(str);
+ 7        this.figure = new StackPane( circle, text );
+ 8        this.figure.setLayoutX(x);
+ 9        this.figure.setLayoutY(y);
+10     }
+11     public StackPane getFigure() { return figure; }
+12  }
```

```
FigureGroup source = new FigureGroup(80,50,40,Color.LIGHTBLUE,"SOURCE");
FigureGroup target = new FigureGroup(320,50,50,Color.RED,"DROP TARGET");
root.getChildren().addAll(source.getFigure(),target.getFigure());
```

Requirement #4

```
1  //Imports are omitted
2  public class FigureGroup {
3     private StackPane figure;
+ 4   private double origin_x, origin_y;
5     public FigureGroup(int x, int y, int radius, Color color, String str) {
6        Circle circle = new Circle(radius, color);
7        Text text = new Text(str);
8        this.figure = new StackPane( circle, text );
9        this.figure.setLayoutX(x);
10       this.figure.setLayoutY(y);
11    }
+ 12   public void setBackOrigin() {
+ 13      this.figure.setLayoutX(origin_x);
+ 14      this.figure.setLayoutY(origin_y);
+ 15   }
+ 16   public void setPosition(double x, double y) {
+ 17      this.figure.setLayoutX(this.figure.getLayoutX()+x);
+ 18      this.figure.setLayoutY(this.figure.getLayoutY()+y);
+ 19   }
20    public StackPane getFigure() { return figure; }
21 }
```

4. If we drag the source and release the mouse button outside the target, the source object moves back to the origin coordinate.

Req #2 & #4 have to considered the source

- Source object
  - setOnMousePressed(EventHandler<MouseEvent>)
  - setOnDragDetected(EventHandler<DragEvent>)
  - setOnMouseDragged(EventHandler<MouseEvent>)
  - setOnMouseReleased(EventHandler<MouseEvent>)

Requirement #2 and #4

```
+13        source.getFigure().setOnMousePressed(new EventHandler<MouseEvent>() {
+14            @Override
+15            public void handle(MouseEvent event) {
+16                mouse_x = event.getScreenX();
+17                mouse_y = event.getScreenY();
+18            }
+19        });
+20        source.getFigure().setOnDragDetected(new EventHandler<MouseEvent>() {
+21            @Override
+22            public void handle(MouseEvent event) {
+23                source.getFigure().setMouseTransparent(true);
+24                source.getFigure().toFront();
+25                source.getFigure().startFullDrag();
+26            }
+27        });
```

Requirement #2 and #4

```
+28      source.getFigure().setOnMouseDragged(new EventHandler<MouseEvent>() {
+29         @Override
+30         public void handle(MouseEvent event) {
+31            double deltaX = event.getScreenX() - mouse_x;
+32            double deltaY = event.getScreenY() - mouse_y;
+33            source.setPosition(deltaX, deltaY);
+34            mouse_x = event.getScreenX();
+35            mouse_y = event.getScreenY();
+36         }
+37      });
+38      source.getFigure().setOnMouseReleased(new EventHandler<MouseEvent>() {
+39         @Override
+40         public void handle(MouseEvent event) {
+41            source.setBackOrigin();
+42            source.getFigure().setMouseTransparent(false);
+43         }
+44      });
```

What to condor more on Req #3 and #4

1. If we drag the source and release the mouse button over the target, the source object will be snapped in front of the target object.

2. If we drag the source and release the mouse button outside the target, the source object moves back to the origin coordinate.

- **Target object**
  - setOnMouseDragEntered(EventHandler<MouseEvent>)
  - setOnMouseDragExited(EventHandler<MouseEvent>)

What to condor more on Req #3 and #4

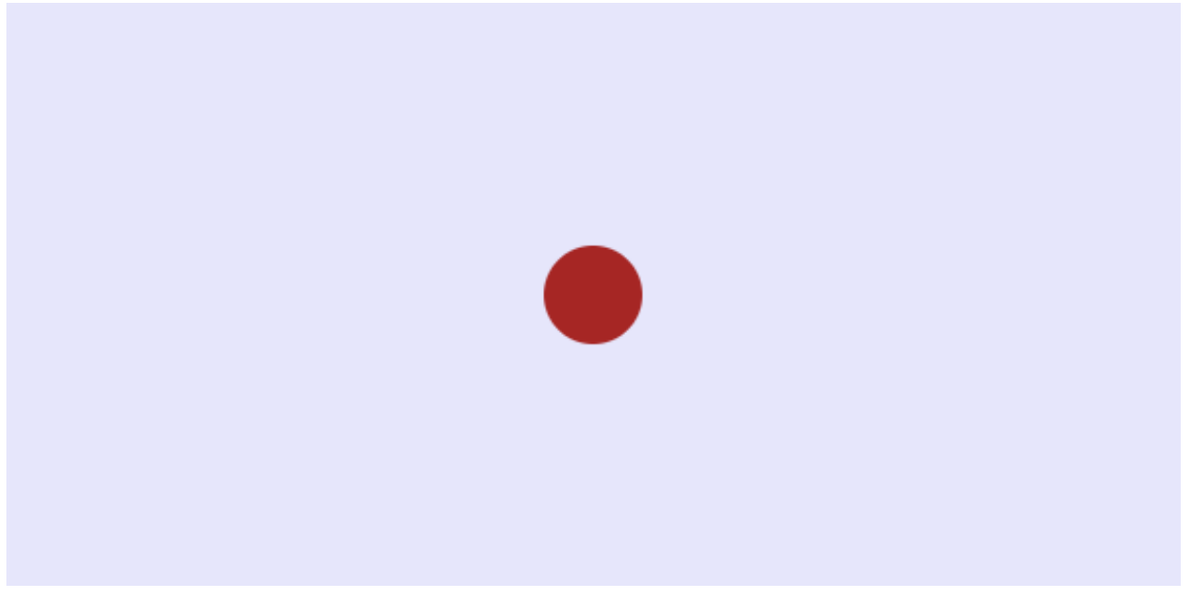
1. If we drag the source and release the mouse button over the target, the source object will be snapped in front of the target object.

2. If we drag the source and release the mouse button outside the target, the source object moves back to the origin coordinate.

State !

Toggle state — Flag variable

- A logical concept that is commonly implemented by using a boolean variable
- Similar to using an electronic switch to control an electric device's on/off state, such as a light bulb.



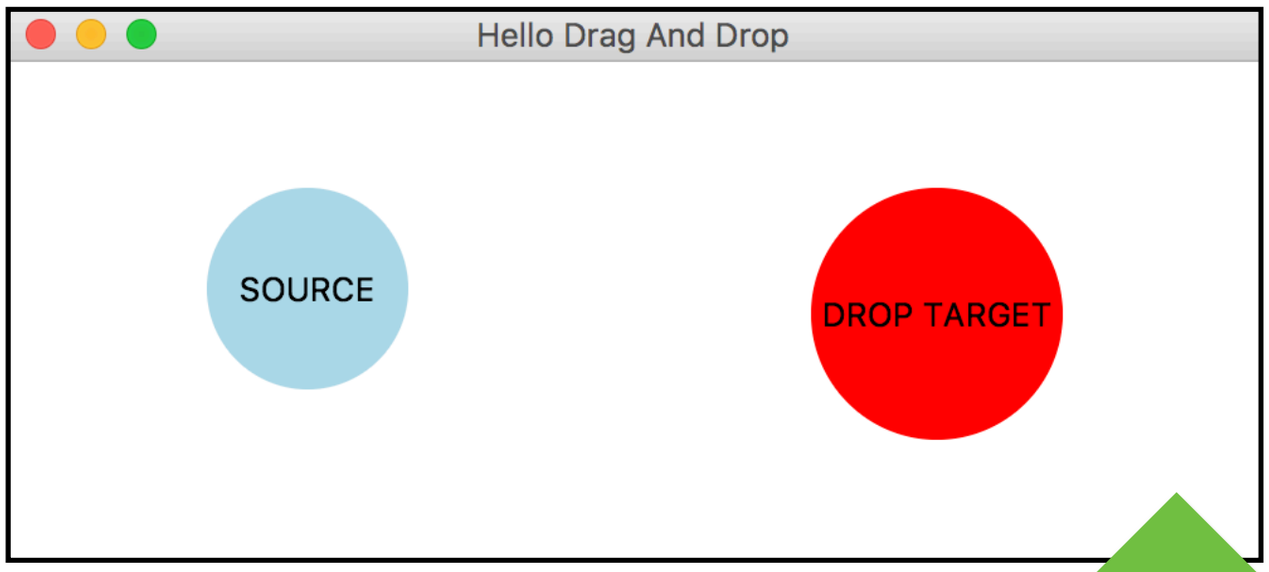
```
if (flag) {
  circle.setFill(Color.DARKSLATEBLUE);
} else {
  circle.setFill(Color.BROWN);
}
flag = !flag;
```

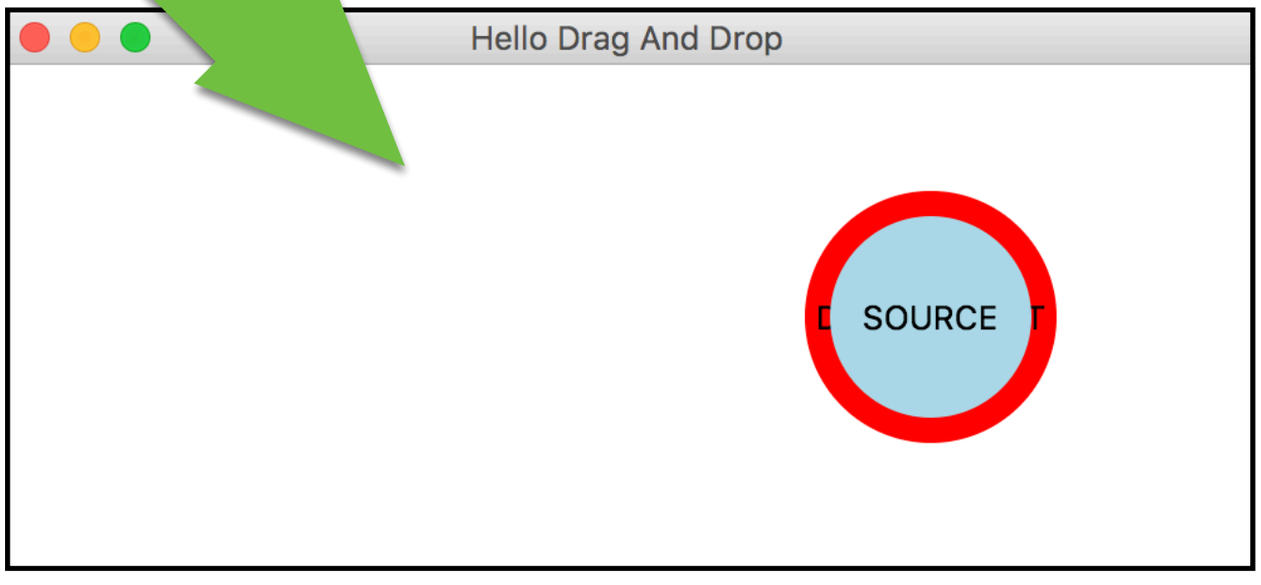
Requirement #3 and #4

```
+ 54        target.getFigure().setOnMouseDragEntered(new EventHandler<MouseEvent>() {
+ 55            @Override
+ 56            public void handle(MouseEvent event) {
+ 57                isEntered = true;
+ 58            }
+ 59        });
+ 60        target.getFigure().setOnMouseDragExited(new EventHandler<MouseEvent>() {
+ 61            @Override
+ 62            public void handle(MouseEvent event) {
+ 63                isEntered = false;
+ 64            }
+ 65        });
```

Drag & drop

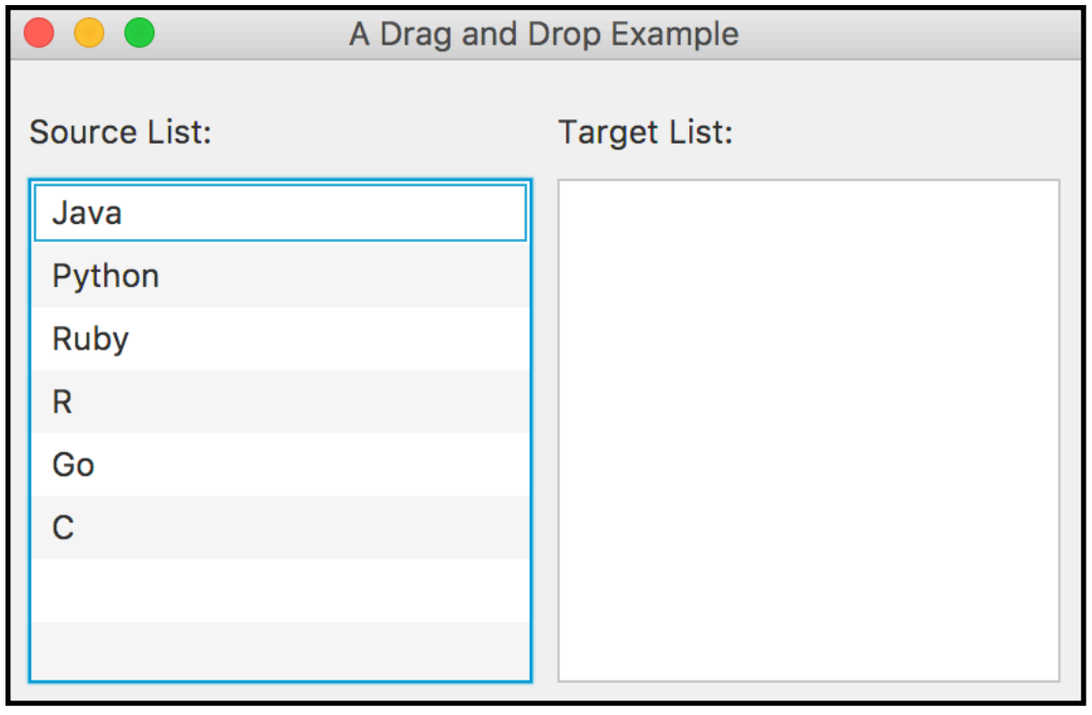
- A toy problem





How to apply to a list of texts

- Another toy problem



More things to consider

1. How to transfer the data associated with each word between lists.

2. When and how to let the target list accept the data being dragged from the source list.

3. After dropped, what are needed to be done on the source list and the target list.

More events

- Mainly because both lists can be either source or target.
- Reusable methods for drag and drop between source and target are required
  - dragDetected -> Store **data** to clipboard
  - dragOver -> Prepare the paste target
  - dragDropped -> Paste **data** on target
  - dragDone -> Remove **data** from source

More events

```
+ 7      lV.setOnDragDetected(new EventHandler<MouseEvent>() {
+ 8         public void handle(MouseEvent event) {
+ 9            Dragboard dragboard = lV.startDragAndDrop(TransferMode.MOVE);
+ 10           String selectedItems = lV.getSelectionModel().getSelectedItem();
+ 11           ClipboardContent content = new ClipboardContent();
+ 12           content.putString(selectedItems);
+ 13           dragboard.setContent(content);
+ 14        }
+ 15     });
+ 16     lV.setOnDragOver(new EventHandler <DragEvent>() {
+ 17        public void handle(DragEvent event) {
+ 18           Dragboard dragboard = event.getDragboard();
+ 19           if (event.getGestureSource() != lV && dragboard.hasString()) {
+ 20              event.acceptTransferModes(TransferMode.MOVE);
+ 21           }
+ 22        }
+ 23     });
```

More events

```
+24      lV.setOnDragDropped(new EventHandler <DragEvent>() {
+25         public void handle(DragEvent event) {
+26            boolean dragCompleted = false;
+27            Dragboard dragboard = event.getDragboard();
+28            if(dragboard.hasString()) {
+29               String list = dragboard.getString();
+30               lV.getItems().addAll(list);
+31               dragCompleted = true;
+32            }
+33            event.setDropCompleted(dragCompleted);
+34         }
+35      });
+36      lV.setOnDragDone(new EventHandler <DragEvent>() {
+37         public void handle(DragEvent event) {
+38            lV.getItems().remove(lV.getSelectionModel().getSelectedItem());
+39         }
+40      });
```

Keyboard events

- The event will be received after a particular keyboard key is pressed

The main difference from mouse events

- For a mouse event, we can imply that the user obviously knows the visual element that she or he may intend to control by using the mouse pointer over the specific visual element.
- Thus, we cannot guess a specific event sources, so we have to bind a keyboard event on the entire application window.

For example

```
+ 9       FigureGroup obj = new FigureGroup(200,50,50,Color.CORAL,"FIGURE");
+ 10      scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
+ 11         @Override
+ 12         public void handle(KeyEvent event) {
+ 13            if (event.getCode() == KeyCode.UP) {
+ 14               obj.setPosition(0,-5);
+ 15            } else if (event.getCode() == KeyCode.DOWN) {
+ 16               obj.setPosition(0,5);
+ 17            } else if (event.getCode() == KeyCode.LEFT) {
+ 18               obj.setPosition(-5,0);
+ 19            } else if (event.getCode() == KeyCode.RIGHT) {
+ 20               obj.setPosition(5,0);
+ 21            }
+ 22         }
+ 23      });
+ 24      root.getChildren().add(obj.getFigure());
```

For example

```
+  9        FigureGroup obj = new FigureGroup(200,50,50,Color.CORAL,"FIGURE");
+ 10        scene.setOnKeyPressed(new EventHandler<KeyEvent>() {
+ 11           @Override
+ 12           public void handle(KeyEvent event) {
+ 13              if (event.getCode() == KeyCode.UP) {
+ 14                 obj.setPosition(0,-5);
+ 15              } else if (event.getCode() == KeyCode.DOWN) {
+ 16                 obj.setPosition(0,5);
+ 17              } else if (event.getCode() == KeyCode.LEFT) {
+ 18                 obj.setPosition(-5,0);
+ 19              } else if (event.getCode() == KeyCode.RIGHT) {
+ 20                 obj.setPosition(5,0);
+ 21              }
+ 22           }
+ 23        });
+ 24        root.getChildren().add(obj.getFigure());
```

Bound with the application's scene

Summary

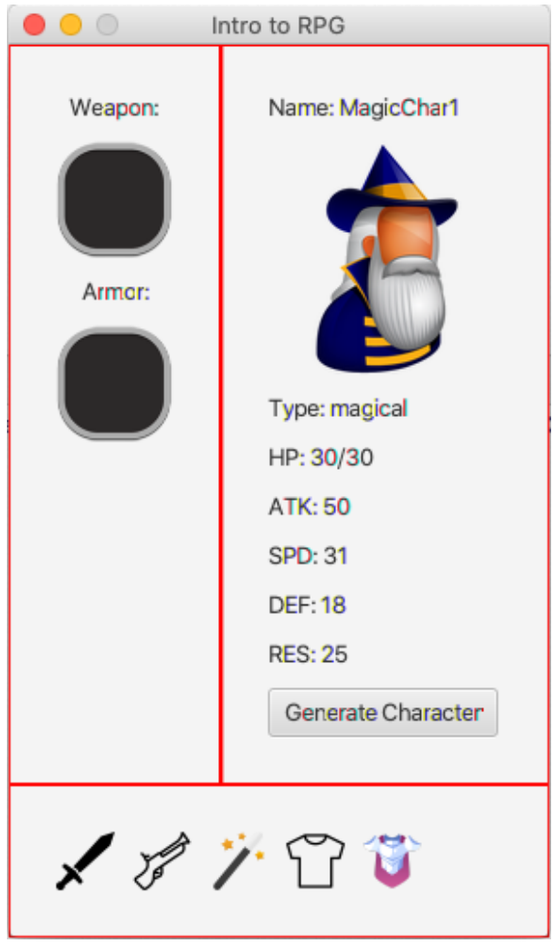
- OOP is necessary for GUI development.
- We will use JavaFX for the entire course.
- Fundamental components of events are event sources, event listeners, and event handler.
- State is data or information that will be changed or manipulated throughout a program's runtime.
- Flag is a logical concept commonly used in implementing what can be toggled.
- Drag & drop is one of the very best approaches to learn about various mouse events at once — Many things to consider
- Key events are bound with the entire application's scene.

Question

Assignment I

- 1 mark of the total grade, credit to the lecture participation.
- Follow the appendix of the lecture handout and configure JavaFX in your computer before starting the first lab.
- Submit the screenshot showing that you can run the HelloWorld example of the Chapter 1.
- Due Wednesday September 16, 2020 @23.59

Lab brief


Intro to RPG
Weapon:
Armor:
Name: MagicChar1
Type: magical
HP: 30/30
ATK: 50
SPD: 31
DEF: 18
RES: 25
Generate Character

Remarks — Lab tutorial

1. The first video shall be captured at 1.4.3, you will get the score for 1.4.1 and 1.4.2. with this submission.
2. The second video shall be captured at 1.4.4. It is to show that the application is still able to run.
3. The third video shall be captured at 1.4.5. It is to show that the drag & drop is implemented
4. The third video shall be captured at 1.4.6. It is to show that the attributes value can be propagated with the drag & drop functionality.