Introduction to Java for C++ Programmers

JavaFx

Segment 2- Event Driven Programming

Professor: Mahboob Ali

Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order/ statement order.
- In event-driven programming, code is executed upon activation of events.
- Operating systems constantly monitor events
  - Like keystrokes, mouse click etc.
- The OS
  - Sorts out these events
  - Reports them to appropriate programs

Control Flow

- For each control (button, mouse etc.):
  - Define an event handler
  - Construct an instance of event handler
  - Tell the controller who its event handler is

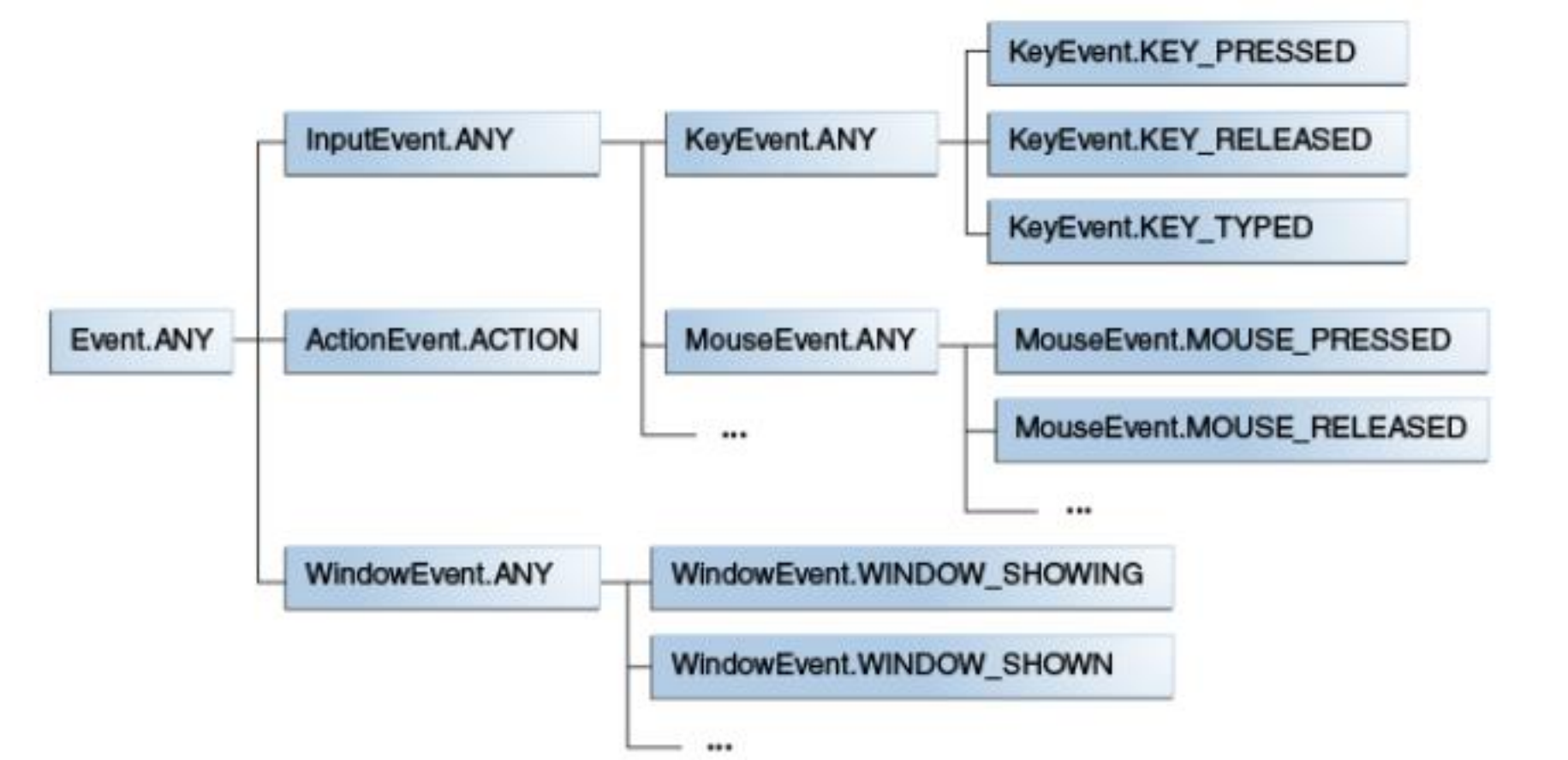
Event Handler

- What is an event handler?
  - Code with response to events
  - a.k.a event listeners

What is an Event?

- An event represents an occurrence of something of interest to the application, such as
  - Mouse being moved
  - Mouse being clicked
  - Key being pressed etc.
- In JavaFX all events are instances of the javafx.event.Event class or any subclass of Event.
- Every JavaFX event has:
  - eventType.
  - source.
  - target.

Types of Events



https://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

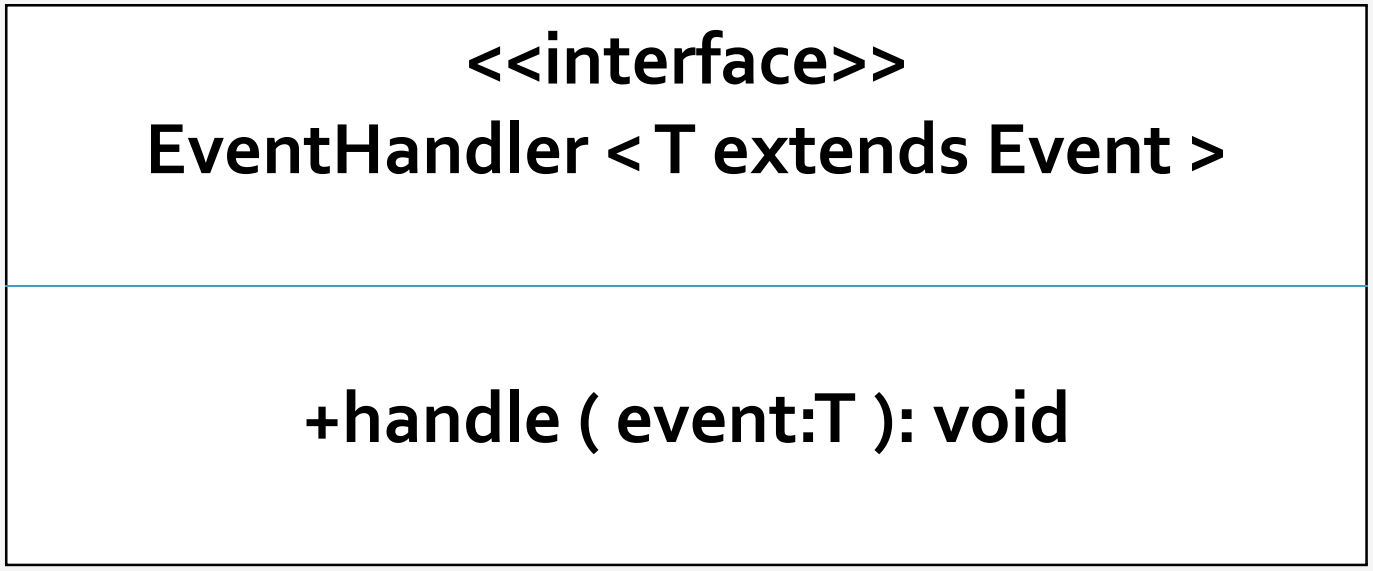
What is Source of an Event?

- A component or *node* can be a "source" of many kinds of events.
- Some event types are different for each node or component.

Button	ActionEvent (button pressed)
TextField	ActionEvent KeyEvent(KeyPress, KeyRelease, KeyTyped)
Any kind of Node	MouseEvent: MousePress, MouseReleased, MouseClicked, MouseDragged etc. Rotation events, Touch Events, etc.

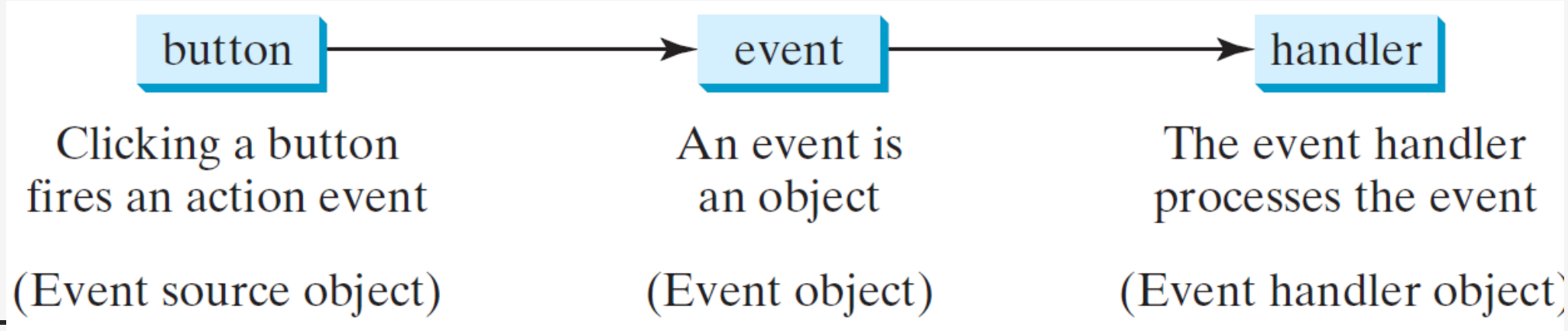
What is an EventHandler?

- JavaFX uses just ***one interface*** for all kinds of events.

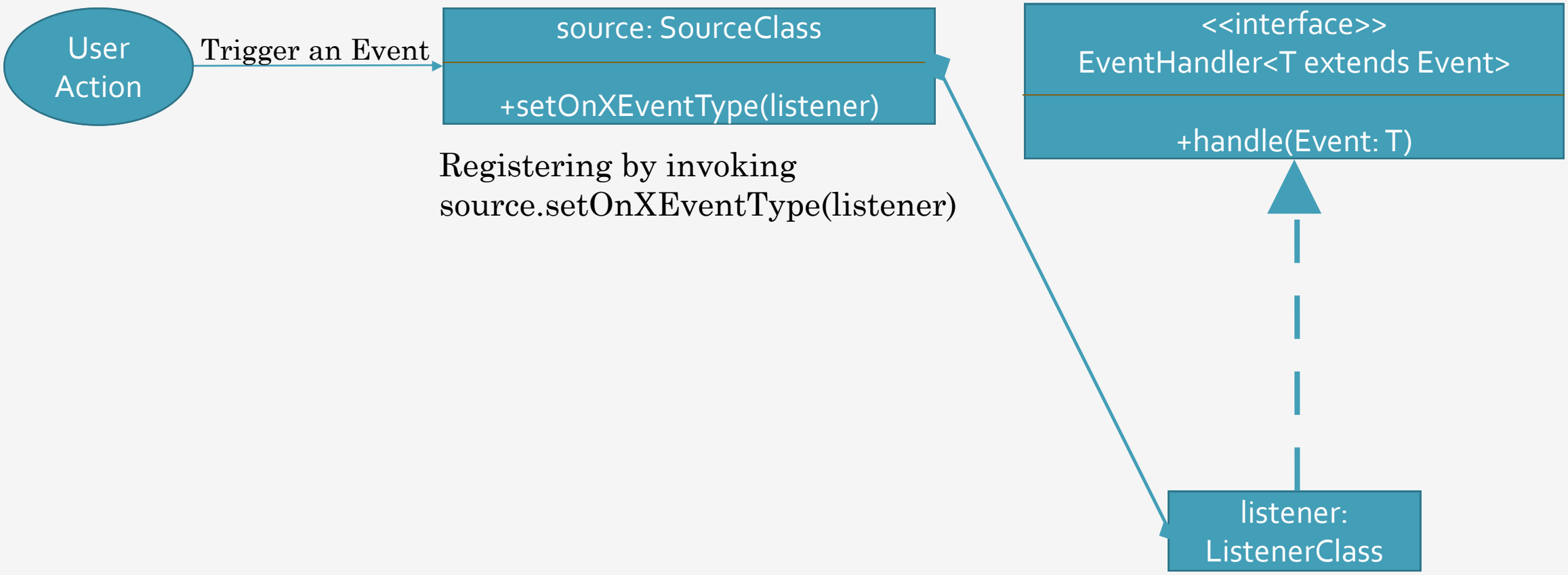


Java Event Handling

- When the user interacts with a controller (source):
  - –An *event object* is constructed.
  - –The event object is sent to all the registered *listener objects.*
  - –The listener objects (handlers) responds as you defined it to



The Delegation Model



How to add an EventHandler?

- There are two ways
  - **addEventHandler** – the general way
  - **setOnXxxx –** convenience methods for specific evet type, such as

    - **setOnAction** (EventHandler<ActionEvent> e)
    - **setOnKeyTyped** (EventHandler<KeyEvent> e)
    - **setOnMouseClicked** (EventHandler<MouseEvent> e)
    - **setOnMouseMoved** (EventHandler<MouseEvent> e)

    ….

Example: Enter or Button clicked

- User types name and clicks a button (or ENTER)
- **Event type**: ActionEvent

```
class ButtonHandler implements EventHandler<ActionEvent>{
	@Override
	public void handle(ActionEvent evt) {
		String text = nameField.getText();
		//TODO greet user
		nameField.setText("");  //clear input
	}
}
```

2 ways to Add Event Handler

- // 1. Use addEventHandler:

  button.addEventHandler (ActionEvent.ANY, new ButtonHandler())

- // 2. Use setOnAction:

  button.setOnAction (new ButtonHandler())

- They both are same and produce the same result

re-use of an Event Handlers

- For clarity, or to *reuse* the event handler on multiple components, create the handler first.

```
ButtonHandler greetHandler = new ButtonHandler( );
button.setOnAction( greetHandler );
nameField.setOnAction( greetHandler );
```

- It is bad programming to create two objects to do the same thing (greet the user).

4 ways to Define an EventHandler

1. Define an (inner) class that implements EventHandler.

2. Write it as *anonymous class*.

3. Write it as a *method* and use a *method reference*.

   Method reference is new in Java 8.

   Works because Event Handler has only 1 method.

4. Write it as a *lambda expression* and use a reference variable to add it.

Will be discussed later in the course

Example of Handling an Event

- Creating a simple event handling event. (Individual Classes)

Simple Event Handler

Ok Button Handler

Cancel Button Handler

Inner Classes

- An inner class, is a class defined within the scope of another class.
- Inner classes are useful for defining handler classes.
- Inner class may be used just like a regular class.
- You define a class as inner if it is used only by its outer class.

Inner Classes Features

- An inner class is compiled into a class named **OuterClassName$*InnerClassName*.class**.

- An inner class can reference the data and the methods defined in the outer class in which it nests, so you need not pass the reference of an object of the outer class to the constructor of the inner class.

- An inner class can be declared public, protected, or private subject to the same visibility rules applied to a member of the class.

Inner Class Example

```
public class OuterClass{
        private intdata = 0;
    OuterClass(){
        InnerClass y = new InnerClass();
        y.m2();
    }
    public void m1() {
        data++;
    }
    public static void main(String[] args)   {
    OuterClass x = new OuterClass();
    System.out.println(x.data);
    }

    class InnerClass{
        public void m2() {
            /* Directly reference data and method defined in outer class */
        data++;
        m1();
        }
    }
}
```

Inner Classes Features

- An inner class can be declared static:
  - The static inner class can be accessed using the outer class name.
  - However, a static inner class cannot access non-static members of the outer class.

- Enclosing object will not be Garbage collected until the inner is alive.

Common Uses of Inner Classes

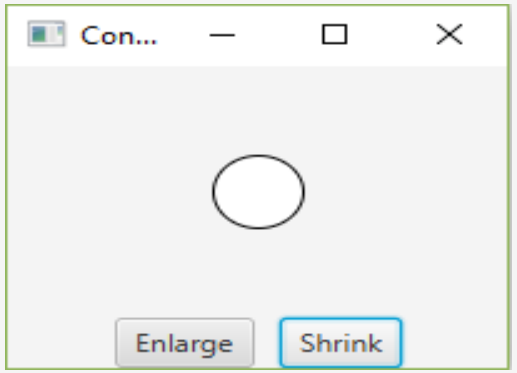
- A simple use of inner classes is to combine dependent classes into a primary class.
  - This reduces the number of source files.
  - This will also makes the organization of the classes easy, as all the inner classes are named with the primary classes as prefix.

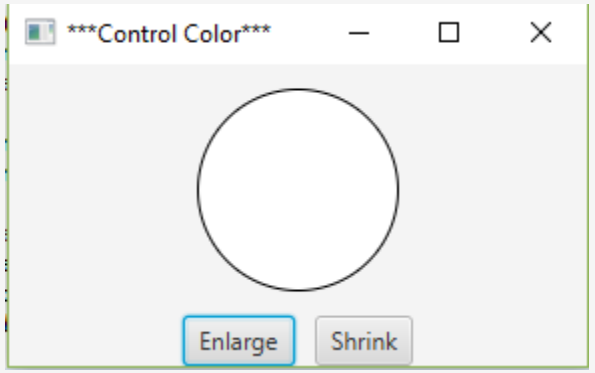
- Another practical use of inner classes is to avoid class-naming conflicts.

Control Circle Example (With Listener)

- Lets look at an example where we want to write a program which uses two button to control the size of a Circle.

Control Circle Example (With Listener)





Anonymous Inner Classes

- An anonymous inner class is an inner class without a name.
  - It combines defining an inner class and creating an instance of the class into one step.
  - An anonymous inner class is declared as follows:

```
new SuperClassName/InterfaceName(){
    //Implement or Override methods
    //Other code
}
```

Anonymous Inner Classes Features

- An anonymous inner class is a special kind of inner class, it is treated like an inner class with the following features:
  - An anonymous inner class must always extend a superclass or implement an interface, but it cannot have an explicit **extends** or **implements** clause.
  - An anonymous inner class must implement all the abstract methods in the superclass or in the interface.
  - An anonymous inner class always uses the no-arg constructor from its superclass to create an instance.
  - An anonymous inner class is compiled into a class named **OuterClassName$*n*.class**. For example, if the outer class **Test** has two anonymous inner classes, they are compiled into **Test$1.class** and **Test$2.class**.