CSSE2002/7023

Semester 2, 2021

Programming in the Large

Week 8.2: More GUIs

In this Session

- EventHandlers
- Buttons
- TextFields
- Dialogs

Concepts

Things happening in a GUI generate events. Being Java, these are represented by objects.

- In this course we only consider `ActionEvent` objects.
- See `javafx.event` for more.

For something to happen as a result of an Event, there needs to be a *corresponding* `EventHandler`.

- In some languages this would be a function, until “recently”, Java has required EventHandlers to be objects[1].
- We will be using objects which implement `javafx.event.EventHandler`.
- `EventHandler` is a generic interface.

[1] We haven't talked about λs yet

EventHandler<ActionEvent>

```
public class Foo implements
        EventHandler<ActionEvent>{

    public void handle(ActionEvent event} {
        // What you want to happen
    }
}
```

You could use a separate class for this, or it could be part of some other class.

Buttons

Print a message to the console when a button is clicked.

`ButtonDemo.java`

1. Create an instance of the event handler object (if you don't already have one).
2. Link it to the button with `setOnAction`.

Note that our event handler class is *package-private*.

- Allows us to include both classes in the same file.

`getSource()`

`ButtonDemo2.java`

Note that *package-private* classes still make `.class` files so watch for name collisions.

Separate immediate event handling from detail

`ButtonDemo3.java`

Better design?

- Actions being carried out are linked indirectly.

Also, note that the event handler needs a reference back to the other object.

Inner Class

`ButtonDemo4.java`

Better design?

- Actions are still linked indirectly.
- Inner class is private because there is no reason for any other (hypothetical) class in the same package to use it.

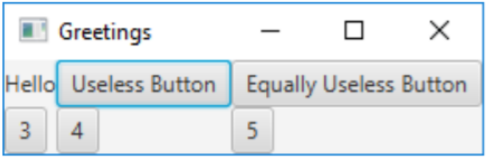
Inner Classes TL;DR

An instance of an inner class has private access to all of the members of its outer instance.

Hence, `ButtonDoer.handle` can call `respondToButton` and can change the stage title.

Multiple Buttons

`ButtonDemo5.java`



Single Event Handler

We can use the source of the event to distinguish between buttons. (Don't use their text — what if your program needs to be localised?)

`ButtonDemo6.java`

Note: Buttons need to be member variables so the inner class can see them.

Nested Classes

Classes can be declared “nested” inside other classes.

Two main possible situations where this applies:

- It makes sense from a namespace/scoping point of view.
  - e.g. `Map.Entry`[2] – there may be other types of Entry that need to be represented but this one specifically relates to Map.
  - Some sort of “Node” would be another example, lots of things could have nodes.
  - These would be declared `static`.
- Second class is “part of” the outer one and should not exist without being bound to a specific instance of the outer class.
  - e.g. A student enrolment record would be associated with one and only one student.
  - Java calls this second type “inner classes”.

See `Outer.java`

[2]Yes, I know this is actually an `interface`.

Anonymous Classes

All we want our inner classes to do is let us call a method ...

- Why do we need to name them?

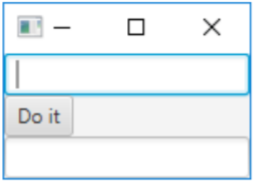
`ButtonDemo7.java`

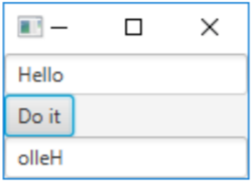
Make sure you understand this syntax

- there are lots of braces flying around

TextFields

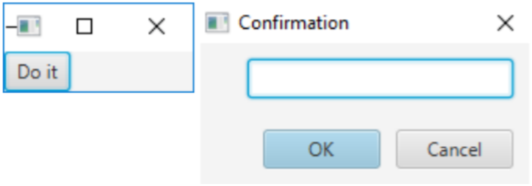
`InputDemo.java`





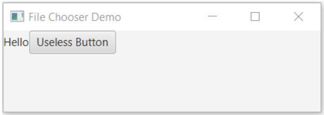
Dialogs

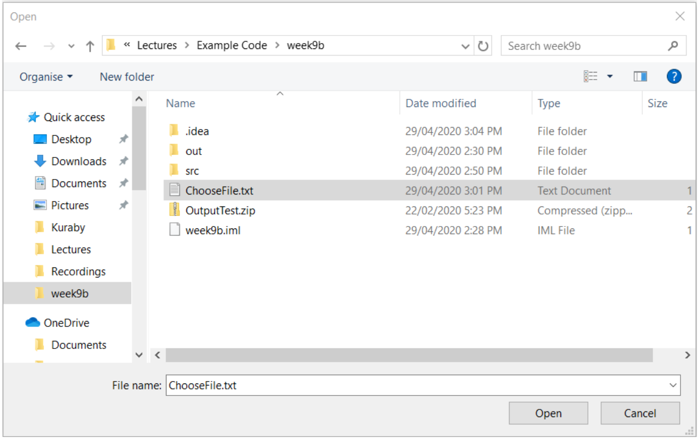
`DialogDemo.java`



FileChooser Dialog

`FileChooserDemo.java`





Name	Date modified	Type	Size
.idea	29/04/2020 3:04 PM	File folder	
out	29/04/2020 2:30 PM	File folder	
src	29/04/2020 2:50 PM	File folder	
ChooseFile.txt	29/04/2020 3:01 PM	Text Document	1
OutputTest.zip	22/02/2020 5:23 PM	Compressed (zipp...	2
week9b.iml	29/04/2020 2:28 PM	IML File	1

File name: ChooseFile.txt