Week 9.2 — GUIs

Concepts

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

- (see `javafx.event`)
- In this course we only need `ActionEvent` objects.

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 don't talk about λs

EventHandler<ActionEvent>

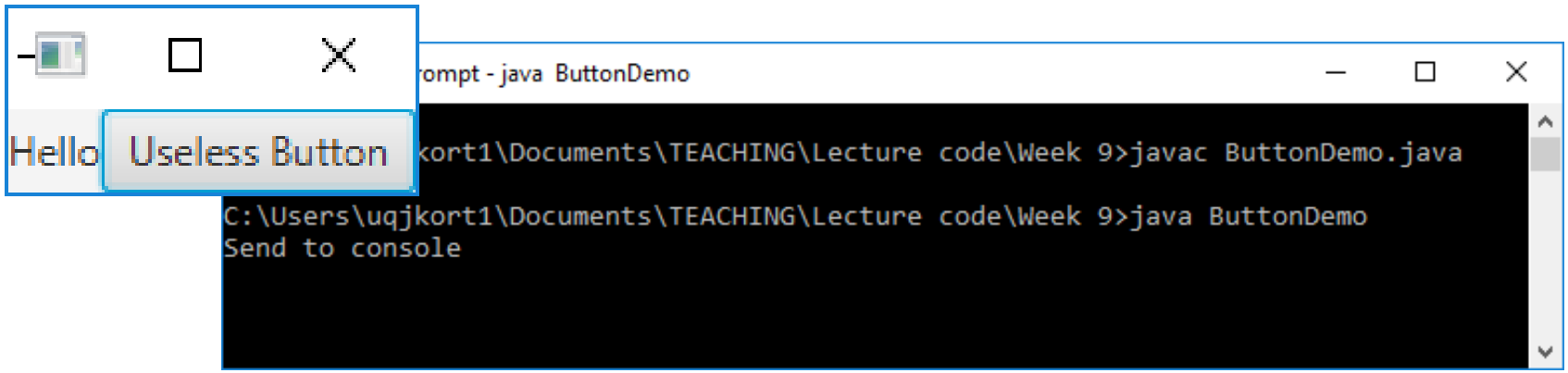
```
public class Foo implements
        EventHandler<ActionEvent>{

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

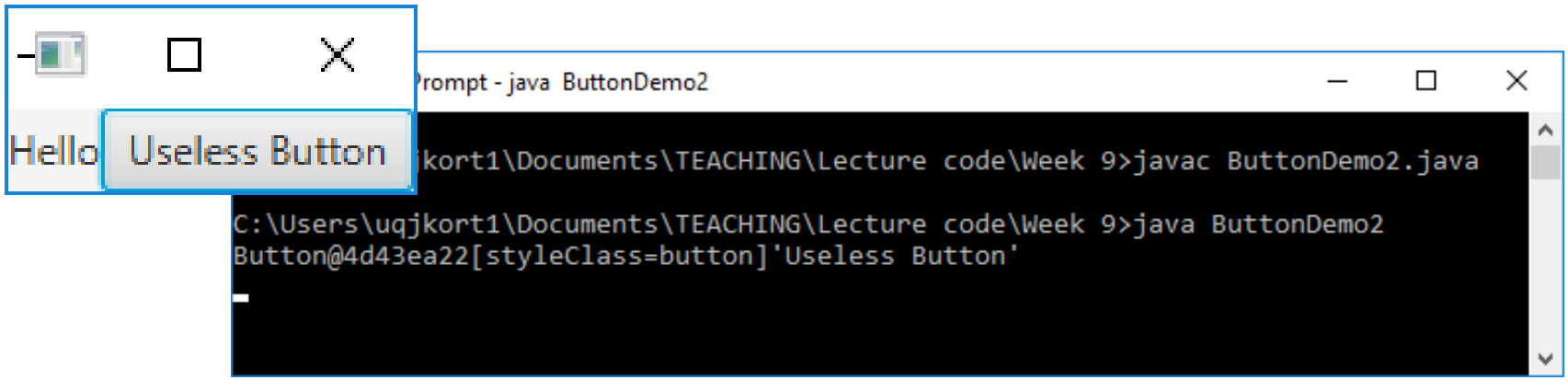
Print a message to the console when a button is clicked.
`ButtonDemo.java`



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*.
(This allows us to include both classes in the same file.)

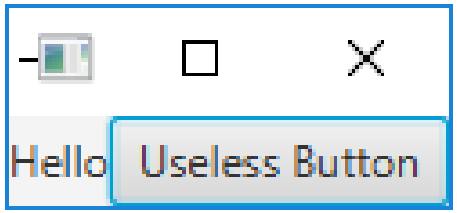
getSource()

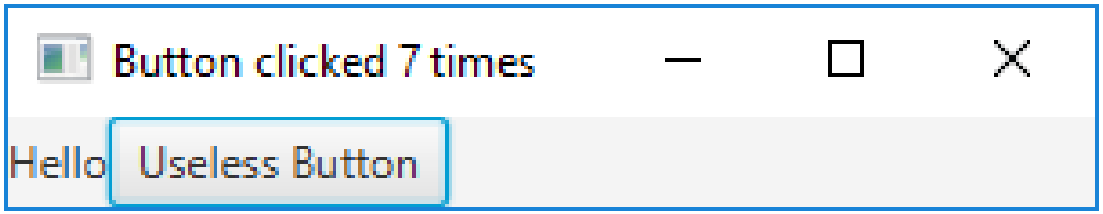


`ButtonDemo2.java`

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

Separate immediate event handling from detail



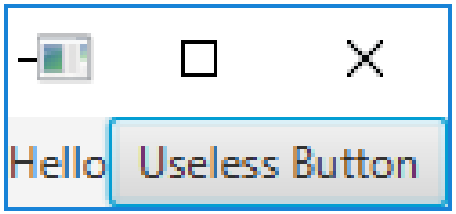


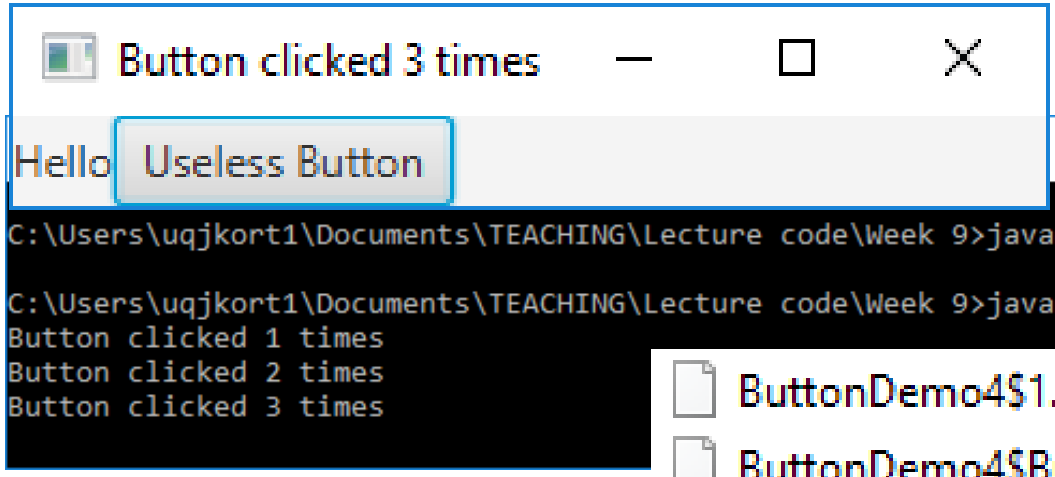
`ButtonDemo3.java`
Better design?

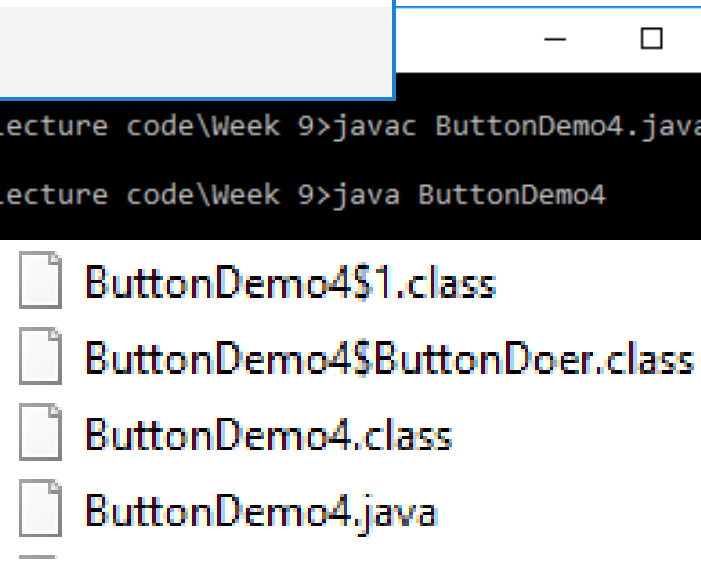
- the actions being carried out are linked indirectly.

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

Inner class







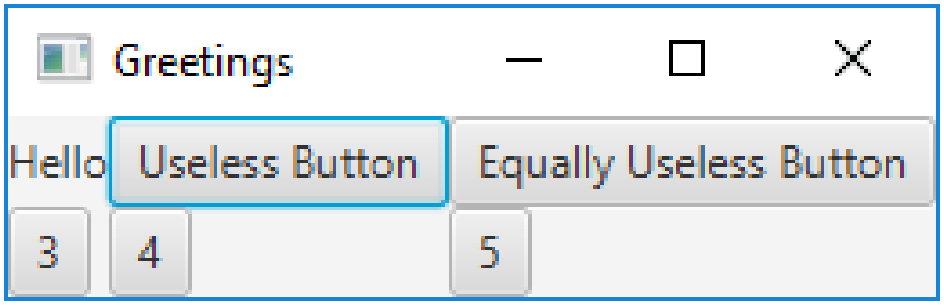
`ButtonDemo4.java`
Better design?

- 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.

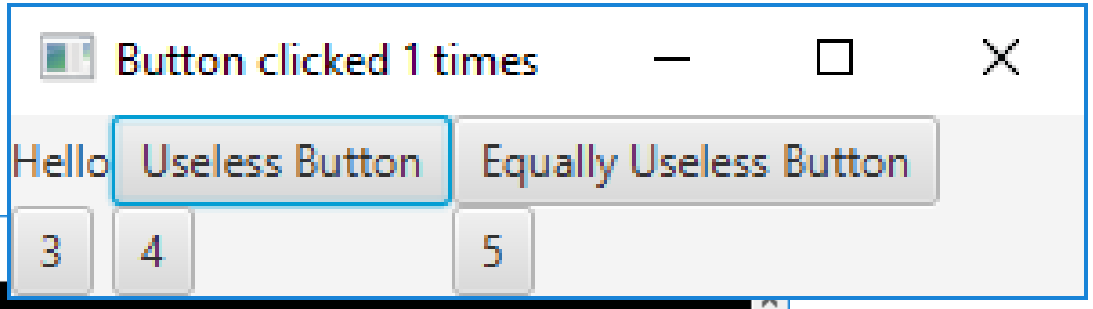
Inner classes TLDR

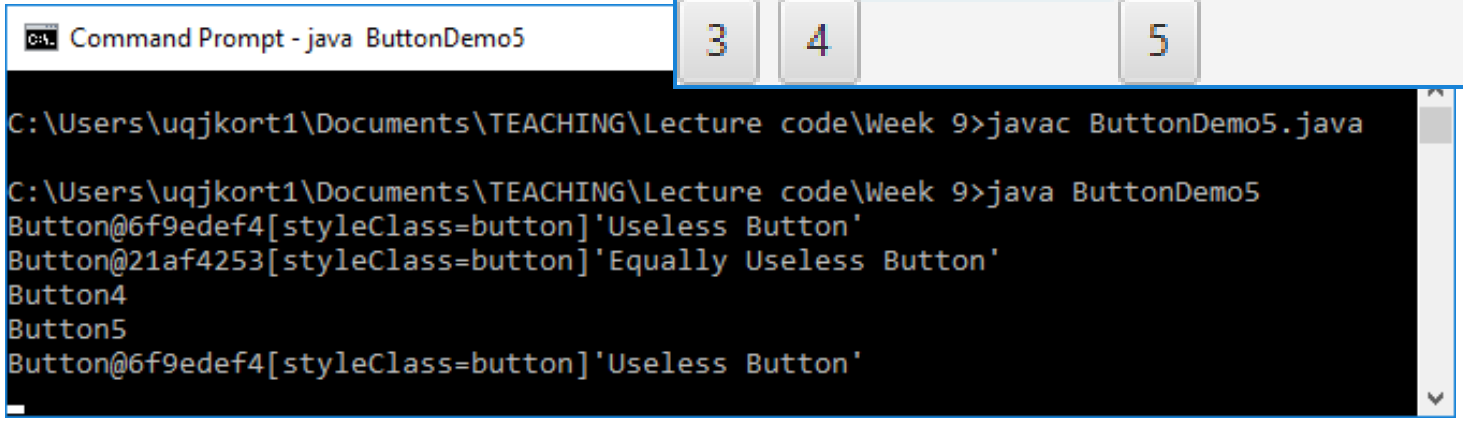
An instance of an inner class has private access to all of the members of its outer instance. Hence, `ButtonDoer.handle` can call `RespondToButton` and can change the stage title.

Multiple buttons

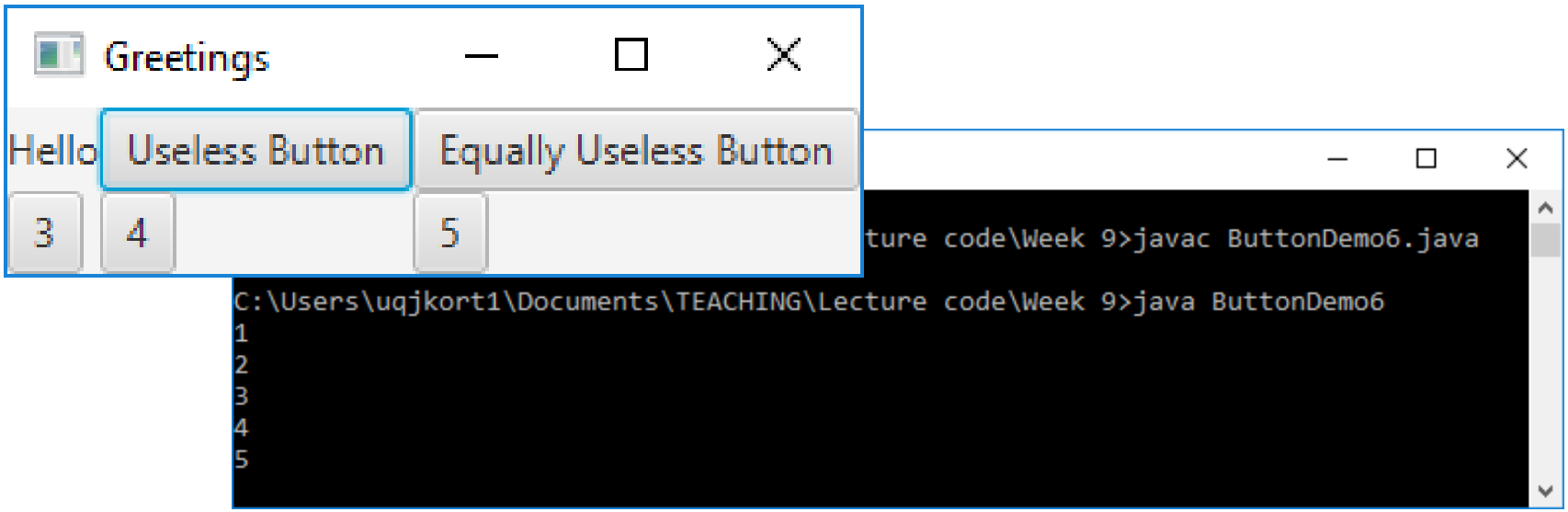


ButtonDemo5.java





Single event handler

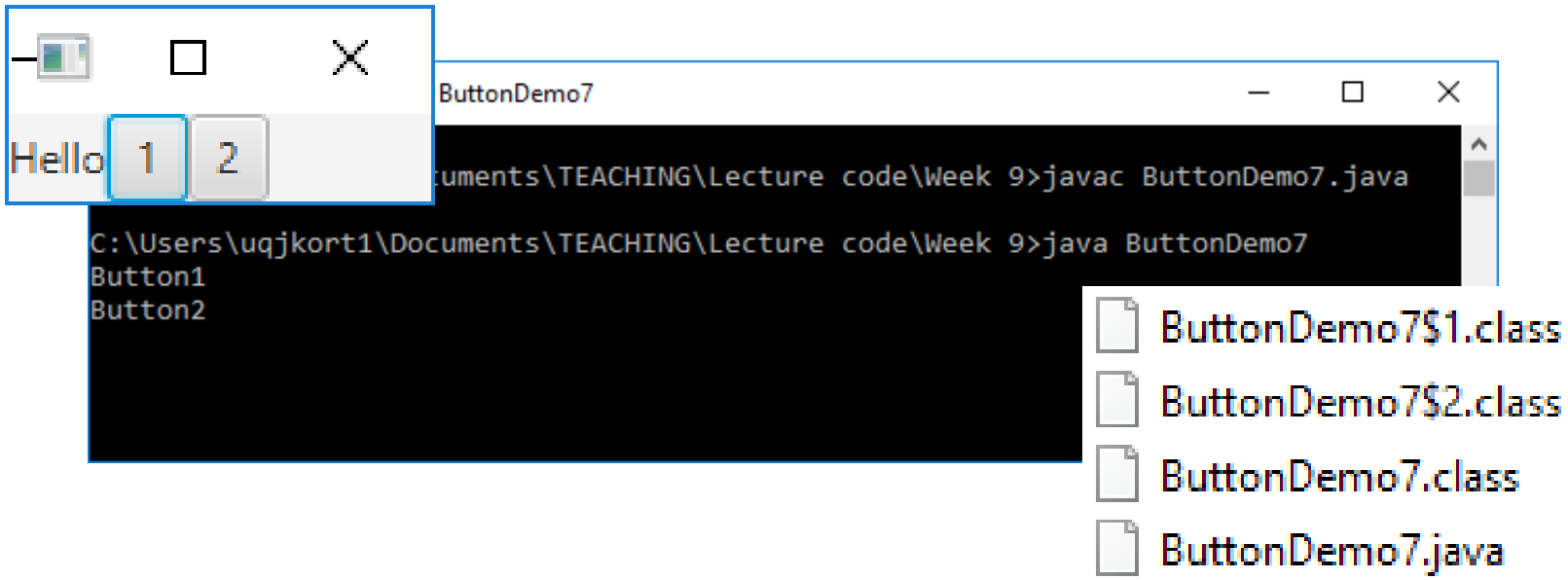


We can use the source of the event to distinguish between buttons. (Don't use their text — what if your program needs to be localised?).

`ButtonDemo6.java`

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

Anonymous classes



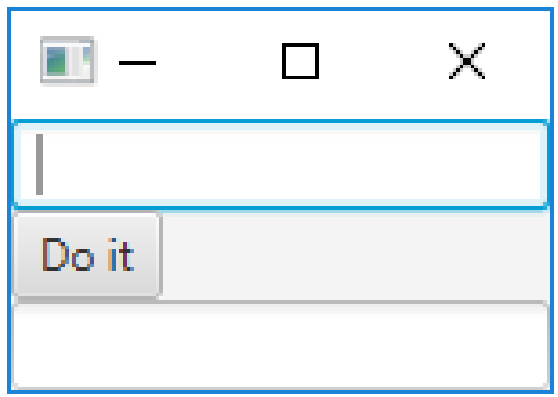
All we want our inner classes to do is let us call a method . . . why do we need to name them?

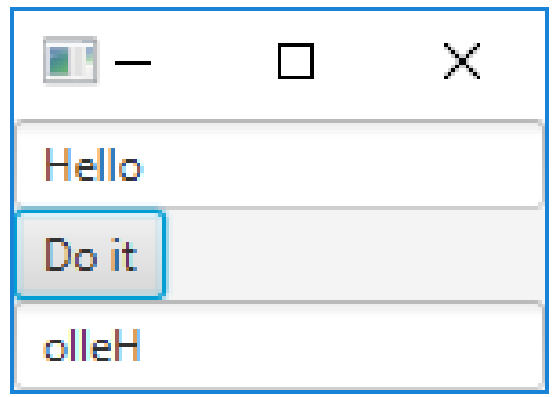
`ButtonDemo7.java`

Make sure you understand this syntax (there are a lot of braces flying around).

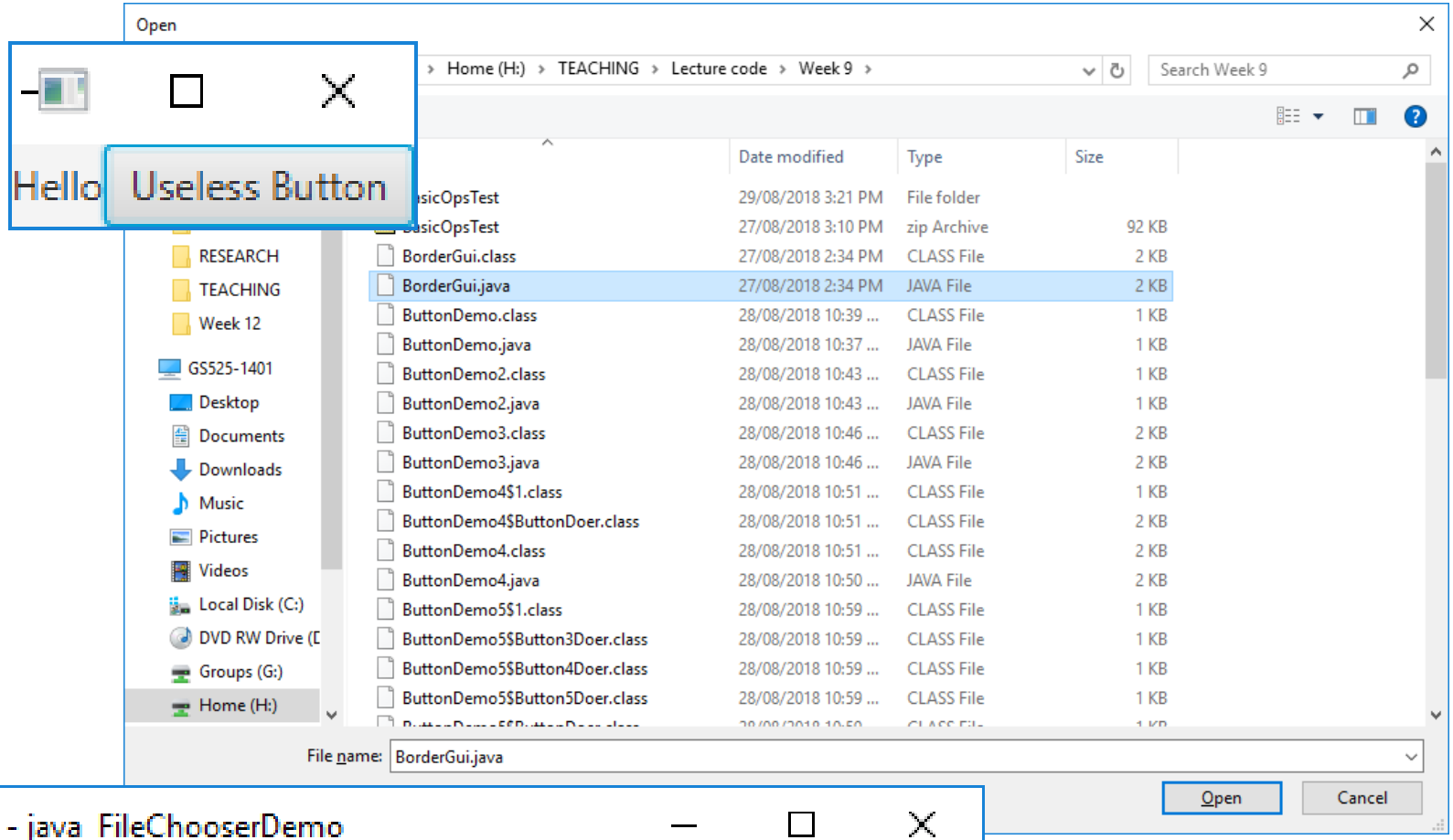
TextFields

`InputDemo.java`





FileChooserDemo.java



Nested classes? — not examinable

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.
  - eg `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`.
- The second class is "part of" the outer one and should not exist without being bound to a specific instance of the outer class.
  - eg 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`.