# TND002 Object-oriented programming

## Lecture 12: Graphical User Interfaces 2

Mark Eric Dieckmann,
MIT division,
ITN, Linköping University.

February 24, 2023

# Outline of the lecture

- I discuss active components of graphical user interfaces.
- I show how to add a button to the GUI.
- I introduce radio buttons.
- I conclude with a text field.

# Code skeleton

We start off with this code and add AWT components.

```
import javax.swing.*; import java.awt.*;
public class GUI extends JFrame {
// Variable declarations come here (global scope).
public GUI() { Font theFont = new Font("SansSerif", Font.PLAIN, 25);
// Variable initializations come here.
Container c = getContentPane(); c.setLayout(new GridLayout(2,2));
// Here we add AWT components to c.
setVisible(true); setDefaultCloseOperation(EXIT_ON_CLOSE);}
public static void main(String[] args) {
GUI myGUI = new GUI();}}
```

Until now, the code creates a miniscule GUI with no content.

We can place 4 AWT components on the GUI.

# Adding a clickable button

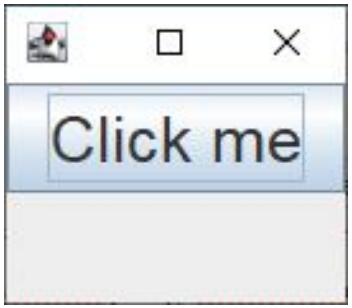
We initialize a button (in the Swing library):

```
JButton theButton = new JButton("Click me");
theButton.setFont(theFont);
theButton.setEnabled(boolean arg);
```

By default, the button is enabled (*arg = true*). We add it to *c*:

```
c.add(theButton); pack(); // adapts the GUI size to the needs.
```

Running the code gives a GUI with a clickable button.




The GUI has one column.

A second column is added when we add more than two AWT components.

# Let the GUI react to a click

The button is clickable but so far the GUI does not react to it.

We need to send the click signal from the button to the GUI.

Java has an interface, which provides the method for it.

The interface is located in subdirectory of the awt library, which we must also import: *java.awt.event.**

We extend the declaration of our class **GUI**

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;
public class GUI extends JFrame implements ActionListener {
```

# Informing the GUI about the click

```
theButton.addActionListener(this);
```

The GUI is our "action listener" because we call the method with *this* in the constructor of **GUI**.

A click lets the button create an instance of the class **ActionEvent** (in java.awt.event).

The button sends it, together with its own address, to the object attached to *this*: In our case the instance of **GUI**.

We analyze it in the method *actionPerformed(ActionEvent arg)*, which we inherit from the interface.

# The method actionPerformed(ActionEvent arg)

```
public void actionPerformed(ActionEvent ae) {
if (ae.getSource() == theButton) {System.out.println("Works");}}
```

The GUI sends the instance of **ActionEvent** (we call it *ea*), which we use like a local variable inside the method.

The class **ActionEvent** has the instance method *getSource()*, which returns the address of the object that sent *ea*.

*if (ea.getSource() == theButton)* compares the address of our button to the address of the source of the action event.

# Adding a radio button

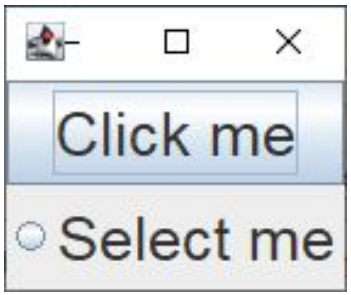
Another useful active component is a radio button.

We declare, initialize and add a radio button to the GUI.

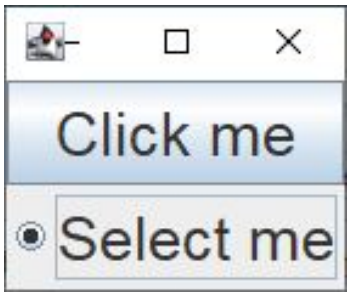
```
private JRadioButton theRadioButton;

theRadioButton = new JRadioButton("Select me");
theRadioButton.addActionListener(this);
theRadioButton.setFont(theFont);

c.add(theRadioButton);
```




Left: Unclicked radio button.
Right: Clicked radio button.



# Expanding actionPerformed(ActionEvent ae)

The radio button sends an action event every time it is clicked.

Its instance method *isSelected()* tells us its state.

We expand our method to include the radio button.

```
public void actionPerformed(ActionEvent ae) {
if (ae.getSource() == theRadioButton) {
System.out.println(theRadioButton.isSelected());}
if (ae.getSource() == theButton) {
System.out.println("Works");}}
```

It will write to the console "Works" if we pressed the button.

If we press the radio button, it will write *true* if the radio button is selected after we pressed it and *false* if not.

# Writing text to a text label

We can connect AWT components.

We want to add or remove "Hello" from a text label by clicking the button.

The radio button should change the background color.

We start with the label (JLabels cannot fire an action event):

```
private JLabel theLabel;

theLabel = new JLabel(); theLabel.setFont(theFont);
theLabel.setOpaque(true);
theLabel.setBackground(Color.BLUE);

c.add(theLabel);
```

# Expanding actionPerformed(ActionEvent ae)

```
public void actionPerformed(ActionEvent ae) {
if (ae.getSource() == theRadioButton) {
if (theRadioButton.isSelected())
theLabel.setBackground(Color.RED);
else theLabel.setBackground(Color.BLUE);}
if (ae.getSource() == theButton) {
if (theLabel.getText().equals("Hello")) theLabel.setText("");
else theLabel.setText("Hello");}}
```


We can switch between 4 states.

# Adding a text field

The class **JTextField** allows us to input directly in the GUI.

We want to introduce a text field into the final slot of our GUI.

```
private JTextField theTextField;

theTextField = new JTextField(); theTextField.setFont(theFont);
theTextField.addActionListener(this);

c.add(theTextField);
```

The text field fires an action event when we press return.

The constructor of **JTextField** is overloaded. We can set the number of visible columns (int) and the initial text (String):

*theTextField = new JTextField("Hello", 20);*

# Important settings and methods of a text field

We can get and set the text on our text field with

```
theTextField.setText("Hello");
String result = theTextField.getText();
```

We can also highlight text with our mouse and get it with

```
String result = theTextField.getSelectedText();
```

If we want to disable out text field (no input or output) we set

```
theTextField.setEnabled(false);
```

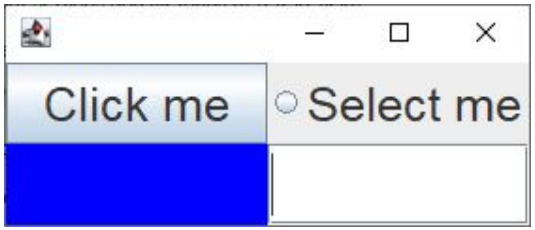
If we want to make it write-only (no input) we set

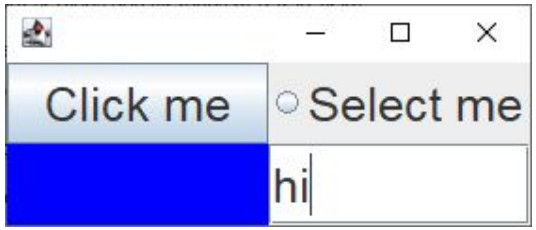
```
theTextField.setEditable(false);
```

# Expanding actionPerformed(ActionEvent ae)

Pressing the button or return should copy the content of the text field into the text label and clear the text field.

```
public void actionPerformed(ActionEvent ae) {
if (ae.getSource() == theRadioButton) {
if (theRadioButton.isSelected())
theLabel.setBackground(Color.RED);
else theLabel.setBackground(Color.BLUE);}
if (ae.getSource() == theButton || ae.getSource() ==
theTextField) { theLabel.setText(theTextField.getText());
theTextField.setText("");}}
```




# Summary

I have discussed

- active components of graphical user interfaces.
- how to add a button to the GUI.
- how radio buttons work.
- text fields.