Programming in JAVA

lecture 8

Object oriented programming
(interfaces)

Interfaces

1. If **all** methods inside a class are abstract (undefined), this class is called an interface.
2. Implementation of interfaces methods are required when a class is implementing an interface.
3. Interfaces are widely used in event handling.
4. The **interface** keyword is used to an interface.
5. It is possible to implement more than one interfaces.

Interfaces - example

```
interface MyInterface
{
   public void method1();
   public void method2();
}
```

Interfaces – example (implementation)

```
public class MyClass implements MyInterface
{
   public void method1(){
      //method1 definition
   }

   public void method2(){
      //method1 definition
   }
}
```

Role of interfaces in GUI programming

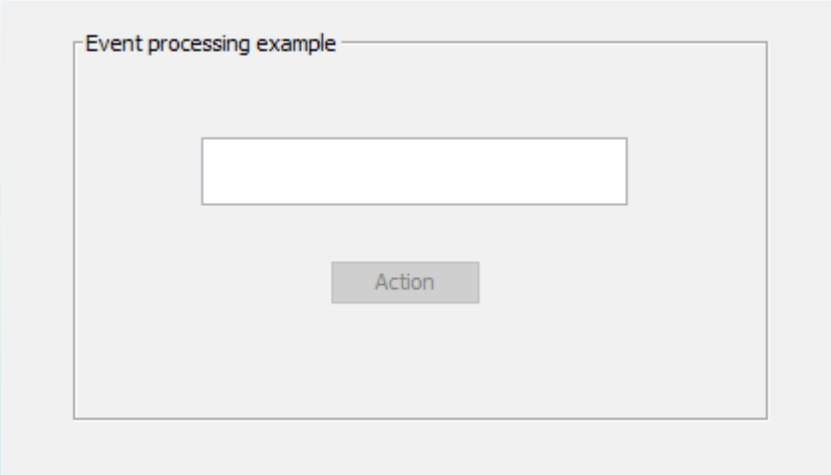
1. Implementing an interface makes a class ready for recieving events. This class becomes an event handling class.
2. The object of event handling class should be registered inside the component that can be a source of events.
3. When the user triggers an event on a component, the component calls the method of the interface implemented inside the event handling class.

Interfaces – example (implementation)

```
public class MyClass implements MyInterface
{
   public void method1(){
      //method1 definition
   }

   public void method2(){
      //method1 definition
   }
}
```

TextField events processing example



In this example, we want to achieve that the button is disabled until there is at least 10 characters in the textfield.

After the button was activated the program should immediately deactivate the button when the user removes characters and the number falls again below 10.

TextField events processing example – part 2

```
8   import javax.swing.event.DocumentListener;
9
10  /**
11   *
12   * @author luke__widsmnl
13   */
    public class NewJFrame extends javax.swing.JFrame implements DocumentListener {
15
16      /**
17       * Creates new form NewJFrame
18       */
19      public NewJFrame() {
20          initComponents();
21      }
```

1. First, we implement an interface `DocumentListener` by adding `implements` keyword after class declaration.

2. After that, right click on the code and choose FixImports.

TextField events processing example – part 3

```
18      */
19     public NewJFrame() {
20         initComponents();
21         jButton1.setEnabled(false);
22     }
23
```

3. Add button deactivation statement inside the constructor. You can also do this in the design mode in Properties window.

TextField events processing example – part 4

```
15  public class NewJFrame extends javax.swing.JFrame implements DocumentListener {
16
17      @Override
19      public void insertUpdate(DocumentEvent e) {
19          throw new UnsupportedOperationException("Not supported yet."); //To cha
20      }
21
22      @Override
24      public void removeUpdate(DocumentEvent e) {
24          throw new UnsupportedOperationException("Not supported yet."); //To cha
25      }
26
27      @Override
29      public void changedUpdate(DocumentEvent e) {
29          throw new UnsupportedOperationException("Not supported yet."); //To cha
30      }
31
32
33
```

4. Move cursor to a line inside the class. Right click and choose Insert Code->Implement Method. Make sure every method is checked and press Generate button.

TextField events processing example – part 5

```
15  public class NewJFrame extends javax.swing.JFrame implements DocumentListener
16
17      @Override
        public void insertUpdate(DocumentEvent e) {
19          System.out.println("insertUpdate event");
20          if (jTextField1.getText().trim().length()>=10)
21              jButton1.setEnabled(true);
22          else
23              jButton1.setEnabled(false);
24      }
25
26      @Override
        public void removeUpdate(DocumentEvent e) {
28          System.out.println("removeUpdate event");
29          if (jTextField1.getText().trim().length()>=10)
30              jButton1.setEnabled(true);
31          else
32              jButton1.setEnabled(false);
33      }
34
35      @Override
        public void changedUpdate(DocumentEvent e) {
37
38      }
```

5. Fill the event handling methods with code (like on the screen)

TextField events processing example – part 6

```
47
48      public NewJFrame() {
49          initComponents();
50          jButton1.setEnabled(false);
            jTextField1.getDocument().addDocumentListener(this);
52      }
53
54
55
```

6. The last thing to do is to register object of event handling class inside the TextField component. We do this by calling the `addDocumentListener` method.

Anonymous classes

1. When programming event handling it is often required to create a new class for a single event implementation.
2. To avoid excessive name creation, anonymous class can be used.
3. In anonymous class, we define a class that implements an interface and in the same place we define the event handling method.

Anonymous classes - example

```
64      @SuppressWarnings("unchecked")
65      // <editor-fold defaultstate="collapsed" desc="Generated Code">
66      private void initComponents() {
67
68          jPanel1 = new javax.swing.JPanel();
69          jTextField1 = new javax.swing.JTextField();
70          jButton1 = new javax.swing.JButton();
71
72          setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
73
74          jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Event proc
75
76          jButton1.setText("Action");
77          jButton1.setEnabled(false);
78          jButton1.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent evt) {
80                  jButton1ActionPerformed(evt);
81              }
82          });
```

Here an example of anonymous class created by NetBeans

for button press event.