Inner Classes

Again our application

```
class SimpleButtonActionListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null,"Thank you!");
    }
}

public class AppWithDialog {

    private void run() {
        JFrame frame=new JFrame("Hello");
        frame.setSize(400,200);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        JButton button=new JButton("Press me!");
        button.addActionListener(new SimpleButtonActionListener());
        frame.add(button);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        AppWithDialog app=new AppWithDialog();
        app.run();
    }
}
```

Not so nice: Message box appears in the center of the screen instead of in the center of the window

To center the message box in the window, showMessageDialog needs the JFrame object as first parameter

Application with improved message box

```
class ButtonActionListenerWithFrame implements ActionListener {
    private AppWithBetterDialog app;

    public ButtonActionListenerWithFrame(AppWithBetterDialog app) {
        this.app=app;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(app.frame, "Thank you!");
    }
}

public class AppWithBetterDialog {
    JFrame frame;

    private void run() {
        frame=new JFrame("Hello");
        [...]

        JButton button=new JButton("Press me!");
        button.addActionListener(new ButtonActionListenerWithFrame(this));
        frame.add(button);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        [...]
    }
}
```

Lot of code to solve a simple problem ☹

The frame field can be only accessed inside the package.

We give the actionlistener a reference to the AppWithBetterDialog object

Application with inner class (fr. *classe interne*)

```
public class AppWithInnerClass {
    private JFrame frame;

    private class MyInnerListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(AppWithInnerClass.this.frame,"Thank you!");
        }
    }

    private void run() {
        frame=new JFrame("Hello");
        [...]

        JButton button=new JButton("Press me!");
        button.addActionListener(new MyInnerListener());
        frame.add(button);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        [...]
    }
}
```

This is an inner class. It can access the fields of the AppWithInnerClass object

You can also simply write:
JOptionPane.showMessageDialog(**frame**,"Thank you!");

Note: Inner classes are syntactic sugar. The Java compiler transforms this program into the same code as on the previous slide.

Inner classes: Summary

- Inner classes help to organize your code better
  - It's clear from the source code that MyInnerListener "belongs" to "AppWithInnerClass"
  - It's allowed that two classes have inner classes with the same name
- An inner class object is always connected to an object of the outer class
  - But it's possible to have *static* inner classes. They can be used without an object of the outer class.

  For example, if MyInnerListener were a *static* inner class, we could write:

  new AppWithInnerClass.MyInnerListener();

  A static inner class can only access static members of the outer class
- Inner classes are useful in many situations. But, in our example, it's not nice that we had to replace the local variable "frame" by a member variable:

```
JFrame frame=new JFrame("Hello");
```

→ **Really?**

```
public class AppWithBetterDialog {
    JFrame frame;
```

What we really wanted...

```
public class AppWithInnerClass2 {
    private class InnerListener implements ActionListener {
        private JFrame frame;

        public InnerListener(JFrame frame) { this.frame=frame; }

        @Override
        public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame,"Thank you!");
        }
    }

    private void run() {
        final JFrame frame=new JFrame("Hello");
        [...]

        JButton button=new JButton("Press me!");
        button.addActionListener(new InnerListener(frame));
        frame.add(button);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        [...]
    }
}
```

I have used “final” here, so I don’t change the variable by accident after creating the InnerListener object

Nice! frame stays a local variable

Anonymous inner class

```
public class AppWithAnonymousClass {
    private void run() {
        JFrame frame=new JFrame("Hello");
        [...]

        JButton button=new JButton("Press me!");
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame,"Thank you!");
            }
        });
        frame.add(button);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        [...]
    }
}
```

This is an anonymous class. An inner class without name.

Note (again): Anonymous classes are syntactic sugar. The Java compiler transforms this program into the same code as on the previous slide.

Anonymous inner class (2)

- Anonymous inner classes are still inner classes. Inside the anonymous inner class, you can access the outer class with

  `AppWithInnerClass.this`

- An anonymous inner class automatically gets a copy of the local variable of the surrounding method.
- But: The local variable must be implicitly final, that means it is not allowed to change its value later! This helps to avoid accidents like this one:

```
JFrame frame=new JFrame("Hello");
JButton button=new JButton("Press me!");
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(frame,"Thank you!");
    }
});
frame=new JFrame("Hello2");
```

This a copy of the local variable “frame”.

Oops. This would mean that the local variable “frame” and its copy in the actionPerformed() method do not have the same value anymore. **Not allowed**!