Wie gelangen Sie an ActionEvents für zwei verschiedene Buttons, wenn jeder Button etwas anderes tun muss?

③ *Möglichkeit 3:* Erstellen Sie zwei separate ActionListener-Klassen.

```
class MeineGui {
  JFrame frame;
  JLabel label;
  void gui() {
    // Code zur Instantiierung der beiden Listener und zur Registrierung
    // des einen beim Farb-Button und des anderen beim Label-Button
  }
} // Klasse schließen
```

```
class ColorButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    frame.repaint();
  }
}
```

Funktioniert nicht! Diese Klasse hat keine Referenz auf die Variable >>frame<< der Klasse MeineGui.

```
class LabelButtonListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    label.setText("Autsch!");
  }
}
```

Ein Problem! Diese Klasse hat keine Referenz auf die Variable >>label<<.

Das Problem: Diese Klassen haben keinen Zugriff auf die Variablen »frame« und »label«, mit denen sie etwas machen sollen! Das könnten Sie zwar hinbiegen, aber dafür müssten Sie jeder der Listener-Klassen eine Referenz auf die Haupt-GUI-Klasse geben, damit der Listener innerhalb seiner actionPerformed()-Methode mit Hilfe dieser Referenz auf die Variablen der GUI-Klasse zugreifen könnte. Aber das bricht wiederum das Prinzip Kapselung, also müssten wir wahrscheinlich Getter-Methoden für die GUI-Widgets schreiben (getFrame(), getLabel() usw.). Und wahrscheinlich müssten Sie für die Listener-Klasse zusätzlich einen Konstruktor schreiben, damit Sie bei der Instantiierung des Listeners die GUI-Referenz an den Listener übergeben können. Und dann ... wird es immer verwirrender und komplizierter.

Es muss einfach einen besseren Weg geben!

Wäre es nicht wundervoll, wenn Sie zwei verschiedene Listener-Klassen hätten, aber diese Listener-Klassen könnten auf die Instanzvariablen der Haupt-GUI-Klasse zugreifen, fast so, als gehörten die Listener-Klassen *zu der anderen Klasse?* So könnten Sie beides unter einen Hut bringen.

Ja, das wäre fantastisch. Aber es ist ja nur ein Traum ...

Innere Klassen kommen uns zu Hilfe

Sie können tatsächlich eine Klasse in eine andere einbetten. Es ist ganz einfach. Sie müssen nur die innere Klasse *innerhalb* der geschweiften Klammern der äußeren Klasse definieren.

Einfache innere Klasse:

```
class MeineÄußereKlasse {

  class MeineInnereKlasse {
    void los() {
    }
  }

}
```

Die innere Klasse ist vollständig von der äußeren Klasse umschlossen.

Eine innere Klasse hat Sonderzutritt zu den Sachen ihrer äußeren Klasse. *Sogar zu den privaten Sachen.* Und die innere Klasse kann diese privaten Variablen und Methoden der äußeren Klasse so benutzen, als wären diese Member in der *inneren* Klasse definiert. Das ist das Praktische bei inneren Klassen – sie haben fast alle Vorteile einer normalen Klasse, aber zusätzlich spezielle Zugriffsrechte.

Eine innere Klasse kann alle Methoden und Variablen der äußeren Klasse benutzen, sogar die privaten.

Die innere Klasse kann diese Variablen und Methoden genauso verwenden, als wären die Methoden und Variablen innerhalb der inneren Klasse deklariert.

Innere Klasse, die eine Variable der äußeren Klasse benutzt

```
class MeineÄußereKlasse {

  private int x;

  class MeineInnereKlasse {
    void los() {
      x = 42;
    }
  } // innere Klasse schließen

} // äußere Klasse schließen
```

Verwenden Sie >>x<<, als wäre es eine Variable der inneren Klasse.

Eine Instanz einer inneren Klasse muss an eine Instanz der äußeren Klasse gebunden sein.*

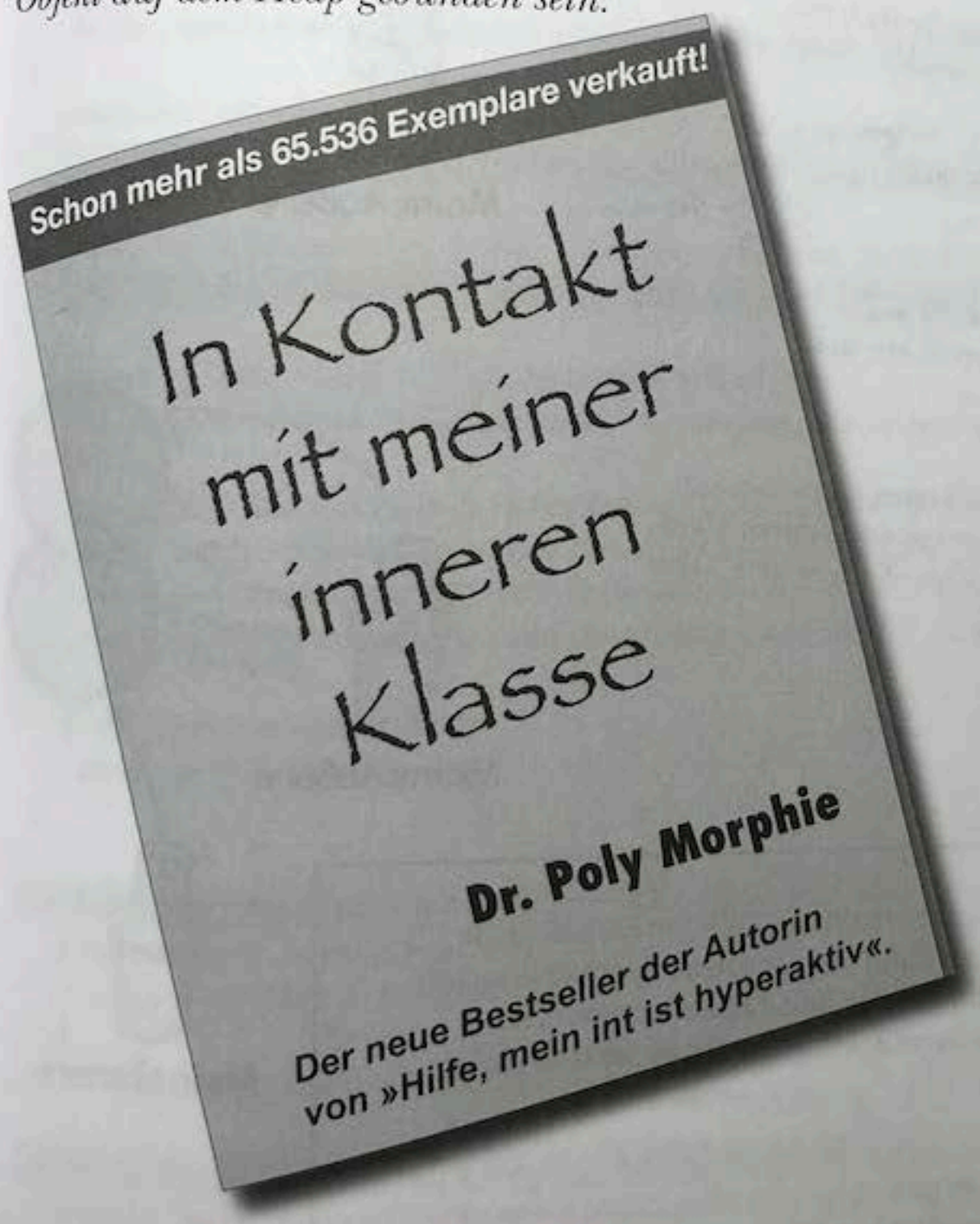
Denken Sie daran – wenn wir sagen, dass eine innere *Klasse* auf etwas in der äußeren Klasse zugreift, meinen wir eigentlich eine *Instanz* der inneren Klasse, die auf etwas in einer *Instanz* der äußeren Klasse zugreift. Aber in *welcher* Instanz?

Kann jede *beliebige* Instanz der inneren Klasse auf die Methoden und Variablen *jeder* Instanz der äußeren Klasse zugreifen?

Nein!

Ein ***inneres*** *Objekt muss an ein ganz bestimmtes* ***äußeres*** *Objekt auf dem Heap gebunden sein.*

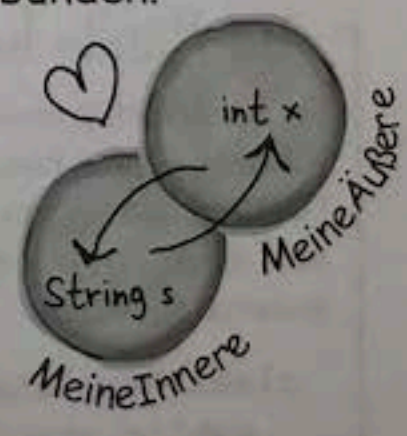
Ein inneres Objekt hat eine ganz besondere Bindung an ein äußeres Objekt.



1. Erzeugen Sie eine Instanz der äußeren Klasse.

2. Erzeugen Sie mit Hilfe der Instanz der äußeren Klasse eine Instanz der inneren Klasse.

3. Das äußere und das innere Objekt sind jetzt innigst miteinander verbunden.



Diese beiden Objekte auf dem Heap haben eine ganz besondere Bindung aneinander. Das innere kann die Variablen des äußeren benutzen (und umgekehrt).

* Es gibt eine Ausnahme bei einem Sonderfall – einer inneren Klasse, die in einer statischen Methode definiert ist. Aber darauf gehen wir hier nicht ein, und es kann auch gut sein, dass Ihnen das in Ihrem gesamten Java-Leben nie begegnen wird.

Wie man eine Instanz einer inneren Klasse erzeugt

Wenn Sie eine innere Klasse über Code instantiieren, der *innerhalb* einer äußeren Klasse steht, ist es die Instanz dieser äußeren Klasse, an die sich das innere Objekt »bindet«. Wenn beispielsweise Code innerhalb einer Methode die innere Klasse instantiiert, bindet sich das innere Objekt an die Instanz, deren Methode ausgeführt wird.

Der Code in einer äußeren Klasse kann eine der eigenen inneren Klassen auf genau die gleiche Weise instantiieren, wie er irgendeine andere Klasse instantiiert ... mit `new meineInnere()`:

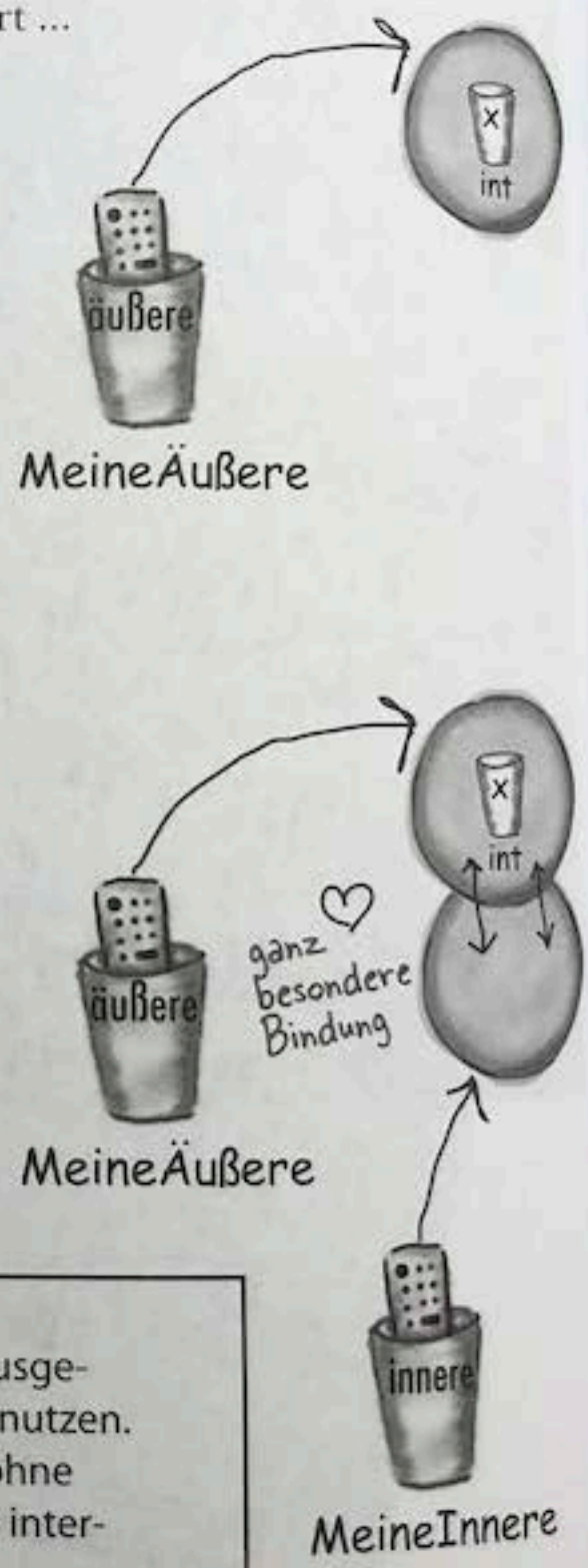
```
class MeineÄußere {

  private int x;

  MeineInnere meineInnere = new MeineInnere();

  public void tuWas() {
    meineInnere.los();
  }

  class MeineInnere {
    void los() {
      x = 42;
    }
  } // innere Klasse schließen

} // äußere Klasse schließen
```

Die äußere Klasse besitzt eine private Instanzvariable »x«.

Erzeugen Sie eine Instanz der inneren Klasse.

Rufen Sie eine Methode auf der inneren Klasse auf.

Die Methode in der inneren Klasse benutzt die Instanzvariable »x« der äußeren Klasse, als wäre »x« ein Element der inneren Klasse.



Am Rande bemerkt

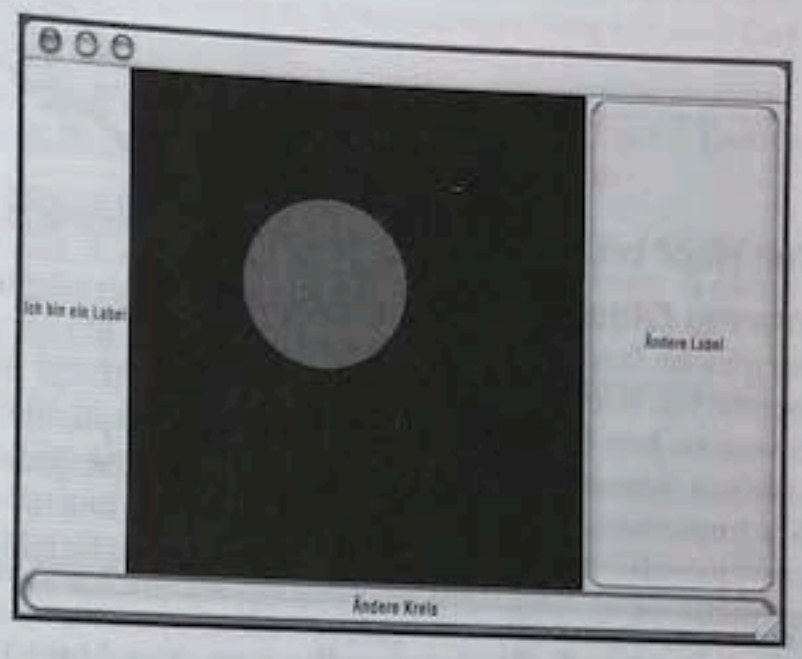
Sie *können* eine innere Instanz auch aus fremdem – außerhalb der äußeren Klasse ausgeführtem – Code heraus instantiieren, aber Sie müssen dafür eine spezielle Syntax benutzen. Die Chancen stehen allerdings gut, dass Sie durch Ihr ganzes Java-Leben kommen, ohne eine einzige innere Klasse von außerhalb erzeugen zu müssen, aber falls es Sie doch interessieren sollte ...

```
class RaritätenKabinett {
  public static void main (String[] args) {
    MeineÄußere äußeresObj = new MeineÄußere();
    MeineÄußere.MeineInnere inneresObj = äußeresObj.new MeineInnere();
  }
}
```

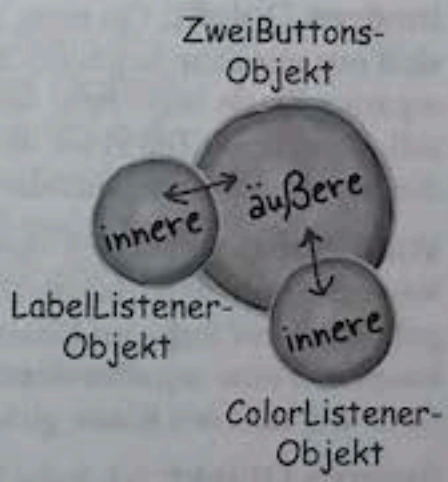
Jetzt können wir den Code mit den zwei Buttons zum Laufen bringen

```
public class ZweiButtons {

  JFrame frame;
  JLabel label;

  public static void main (String[] args) {
    ZweiButtons gui = new ZweiButtons ();
    gui.los();
  }

  public void los() {
    frame = new JFrame();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    JButton labelButton = new JButton("Ändere Label");
    labelButton.addActionListener(new LabelListener());

    JButton colorButton = new JButton("Ändere Kreis");
    colorButton.addActionListener(new ColorListener());

    label = new JLabel("Ich bin ein Label");
    MeinZeichenPanel zeichenPanel = new MeinZeichenPanel();

    frame.getContentPane().add(BorderLayout.SOUTH, colorButton);
    frame.getContentPane().add(BorderLayout.CENTER, zeichenPanel);
    frame.getContentPane().add(BorderLayout.EAST, labelButton);
    frame.getContentPane().add(BorderLayout.WEST, label);

    frame.setSize(420,300);
    frame.setVisible(true);
  }

  class LabelListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      label.setText("Autsch!");
    }
  } // innere Klasse schließen

  class ColorListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
      frame.repaint();
    }
  } // innere Klasse schließen

}
```

Die Haupt-GUI-Klasse implementiert ActionListener jetzt nicht mehr.



Anstatt der Listener-Registrierungsmethode des Buttons this zu übergeben, übergeben Sie jetzt eine neue Instanz der passenden Listener-Klasse.



Jetzt haben wir endlich ZWEI ActionListener in einer einzigen Klasse!

Die innere Klasse kennt das >>label<<.

Die innere Klasse kann jetzt die Instanzvariable >>frame<< benutzen, ohne eine explizite Referenz auf das Objekt der äußeren Klasse zu haben.