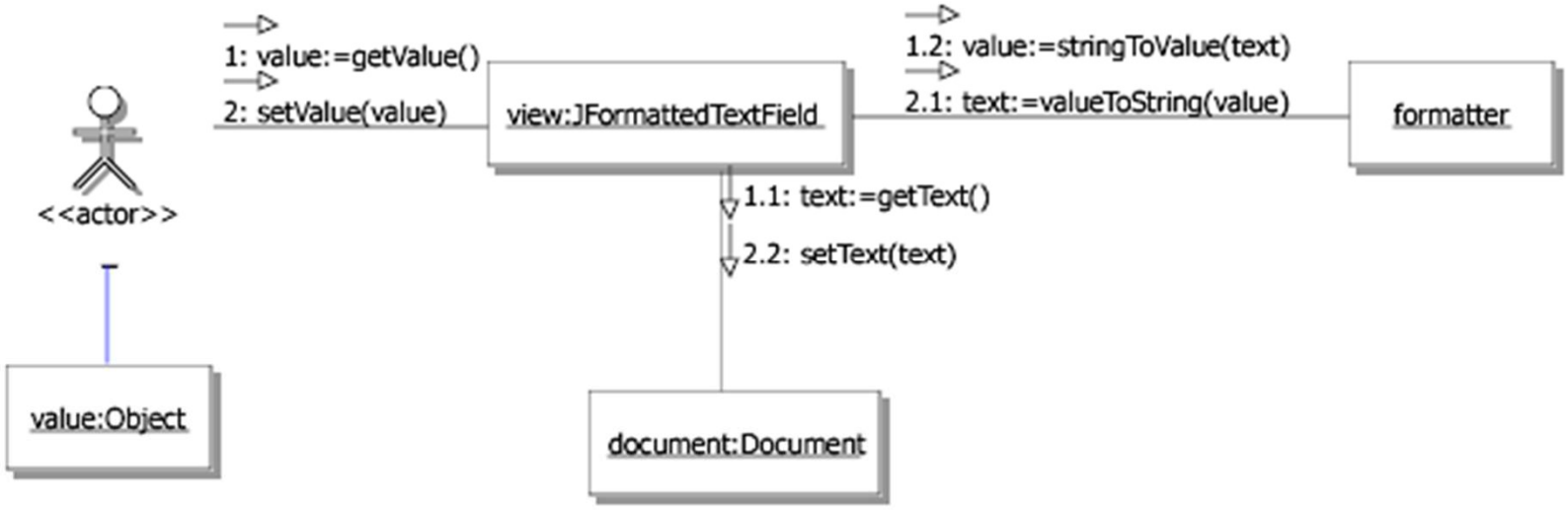
Swing

- JFormattedTextField

JFormattedTextField

- JFormattedTextField erlaubt die formatierte Eingabe von beliebigen Werten
  - Formatter wird verwendet, um zwischen String-Repräsentation und Objektwert zu konvertieren
  - Verifikation, dass korrekte Zeichenkette eingegeben wurde
  - Unterschiedliche Reaktionen bei ungültigen Eingaben
  - Filtern von ungültigen Eingaben



Erzeugen von JFormattedTextField

Beim Anlegen eines `JFormattedTextField` muss Formatter definiert werden

- Erzeugen mit Format-Objekten: Formatter wird auf Basis des Format-Objekts erstellt

```
JFormattedTextField intField = new JFormattedTextField(
        NumberFormat.getIntegerInstance());

JFormattedTextField currencyField = new JFormattedTextField(
        NumberFormat.getCurrencyInstance());

JFormattedTextField dateField = new JFormattedTextField(
        DateFormat.getDateInstance());

JFormattedTextField dateField = new JFormattedTextField(
        DateFormat.getDateInstance(DateFormat.SHORT);
```

(siehe dazu `NumberFormat` und `DateFormat`)

Erzeugen von JFormattedTextField (Forts.)

- Mit DefaultFormatter
  - stringToValue arbeitet mit Objekt-Konstruktor mit String-Parameter
  - valueToString ruft toString auf

```
DefaultFormatter formatter = new DefaultFormatter();
JFormattedTextField urlField = new JFormattedTextField(formatter);
urlField.setValue(new URL("http://java.sun.com"));
```

- Mit MaskFormatter und Definition einer Eingabemaske

```
MaskFormatter formatter = new MaskFormatter("###-##-####");
formatter.setPlaceholderCharacter('0');
JFormattedTextField ssnField = new JFormattedTextField(formatter);
ssnField.setValue("078-05-1120");
```

wobei in der Maske die folgenden Zeichen verwendet werden

#	eine Ziffer
?	ein Buchstabe
U	ein Buchstabe, konvertiert in einen Großbuchstaben
L	ein Buchstabe, konvertiert in einen Kleinbuchstaben
A	ein Buchstabe oder eine Ziffer
H	eine Hexadezimalziffer [0-9A-Fa-f]
*	beliebiges Zeichen

DefaultFormatter durch initialen Wert

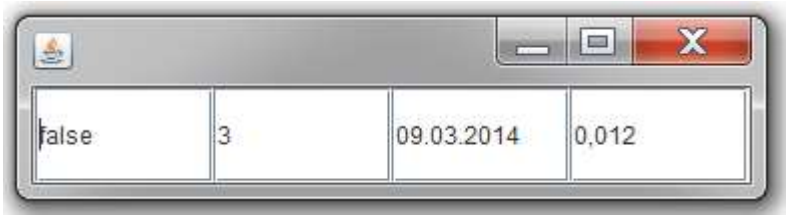
- Durch intialen Wert bei Konstruktion wird aufgrund des Datentyps des Werts ein Formatter erzeugt

```
frame.getContentPane().add(new JFormattedTextField(new Boolean(false)));

frame.getContentPane().add(new JFormattedTextField(new Integer(3)));

frame.getContentPane().add(new JFormattedTextField(new Date()));

frame.getContentPane().add(new JFormattedTextField(new Double(1.23e-2)));
```



Schreiben eines eigenen Formatters

- Ableiten von DefaultFormatter
- Überschreiben der Methoden valueToString und stringToValue

Beispiel: Formatter für IP-Adressen

```
class IPAddressFormatter extends DefaultFormatter {

   public String valueToString(Object value) throws ParseException {
      if (!(value instanceof byte[])) {
         throw new ParseException("Kein byte[]", 0);
      }
      byte[] a = (byte[])value;
      if (a.length != 4) {
         throw new ParseException("Länge != 4", 0);
      }
      StringBuffer buffer = new StringBuffer();
      for (int i = 0; i < 4; i++) {
         int b = a[i];
         if (b < 0) b += 256;
         buffer.append(String.valueOf(b));
         if (i < 3) buffer.append('.');
      }
      return buffer.toString();
   }
   ...
```

Schreiben eines eigenen Formatters (Forts.)

```
   ...
   public Object stringToValue(String text) throws ParseException {
      StringTokenizer tokenizer = new StringTokenizer(text, ".");
      byte[] a = new byte[4];
      for (int i = 0; i < 4; i++) {
         int b = 0;
         try {
            b = Integer.parseInt(tokenizer.nextToken());
         } catch (NumberFormatException e) {
            throw new ParseException("Keine Ganzzahl", 0);
         }
         if (b < 0 || b >= 256) {
            throw new ParseException("Byte außerhalb des Bereichs", 0);
         }
         a[i] = (byte)b;
      }
      return a;
   }
}
```

Verifikation

- Bei Abgabe des Fokus wird geprüft, ob die aktuelle Zeichenkette gültig ist
  - kann der Formatter die Zeichenkette in ein Objekt umwandeln
    dann gültig, sonst ungültig
- Mit Methode
  - boolean isEditValid()

  kann man das auch explizit prüfen
- Bei ungültiger Eingabe gibt sich ein entsprechendes Verhalten, das man mit
  - void setFocusLostBehavior(int behavior)

  einstellen kann

Verhalten bei Abgabe des Focus

- COMMIT_OR_REVERT (Standardverhalten)
  - Wenn gültig, den Wert übernehmen (Methode commitEdit),
  - Wenn ungültig, den letzten Wert ins Ausgabefeld zurückschreiben
- COMMIT
  - Wenn gültig wie oben
  - Wenn ungültig, den letzten Wert und den Inhalt des Ausgabefeldes behalten
- PERSIST
  - Obwohl gültig, den Wert nicht übernehmen, sondern erst bei expliziten Aufruf von commitEdit, setValue oder setText übernehmen
- REVERT
  - Den Wert nicht übernehmen und auch das Ausgabefeld mit dem alten Wert überschreiben

JFormattedTextField Weiteres

- Neben Formatter kann für JFormattedTextField folgendes definiert werden
  - DocumentFilter:
    - Erlauben die Filterung von Eingabezeichen

```
JFormattedTextField intField3 = new JFormattedTextField(
    new InternationalFormatter(NumberFormat.getIntegerInstance()) {
            protected DocumentFilter getDocumentFilter() {
                return filter;
            }
            private DocumentFilter filter = new IntFilter();
    });
```

  - InputVerifier:
    - Explizite Kodierung der Gültigkeitsprüfung in eigener Klasse

```
intField4.setInputVerifier(new FormattedTextFieldVerifier());
```

```
class FormattedTextFieldVerifier extends InputVerifier {
   public boolean verify(JComponent component){
      JFormattedTextField field = (JFormattedTextField)component;
      return field.isEditValid();
   }
}
```