LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

mobile and distributed systems group

Javakurs für Fortgeschrittene

Einheit 02: Streams filtern, Swing

Kyrill Schmid
Lehrstuhl für Mobile und Verteilte Systeme

Heutige Agenda

1. Teil: Datenströme (Streams)

- Filtern
  - FilterWriter, - Reader

2. Teil: Einführung in GUI-Programmierung

- Grundlagen von Swing

Praxis:

- Log-File verschlüsseln
- UI für Log-File

Lernziele

- Kennenlernen und Nutzen von Filter für Datenströme
- Weitere Details und Übungen zu Streams
- Erste Grundlagen zu GUIs in Java mittels Swing

Datenströme filtern

Mit den abstrakten Klassen `FilterReader` bzw. `FilterWriter` können wir Datenströme vor dem eigentlichen Schreibvorgang filtern.

- Um nur bestimmte Informationen durchzulassen
  - Bsp.: Alle Wörter mit „a“
- Um Informationen zu verändern
  - Bsp.: Verschlüsselung

`FilterReader` und `FilterWriter` arbeiten mit dem entsprechenden `Reader` bzw. `Writer` zusammen

- Muss im Konstruktor übergeben werden!
- `protected FilterReader(Reader in)`
- `protected FilterWriter(Writer out)`

Methoden von `FilterWriter`:

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
void	**close**() Closes the stream, flushing it first.
void	**flush**() Flushes the stream.
void	**write**(char[] cbuf, int off, int len) Writes a portion of an array of characters.
void	**write**(int c) Writes a single character.
void	**write**(**String** str, int off, int len) Writes a portion of a string.

- Eigenen `FilterWriter` schreiben (ableiten von `FilterWriter`)
  - Sollte mindestens die `write(int c)` Methode überschreiben!

Unicode

Die Methode `write(int c)` benötigt einen Integer-Wert als Parameter

- `Int c` wird als 16-Bit Unicode Zeichen interpretiert und in den Stream geschrieben
- Dazu werden die beiden niederwertigen Bytes des 4-Byte Integer verwendet
- Für die Umwandlung in ein bestimmtes Zeichen siehe Unicode-Tabelle:
  - Bsp.: A = 65 = 0100 0001
  - Bsp.: Z = 90 = 01011010
  - Bsp.: a = 97 = 0110 0001
  - Bsp.: z = 122 = 0111 1010
- Zum Umwandeln von Groß auf Kleinbuchstaben: +32 rechnen

Quelle: http://unicode-table.com/de/

Eigener `FilterWriter`


```
public class MyFilterWriter extends FilterWriter {

   protected MyFilterWriter(Writer out) {
      super(out);
   }

   @Override
      public void write(int c) throws IOException {

      if (c >= 65 && c <= 90)
         super.write(c+32);
      else
         super.write(c);
   }

   @Override
   public void write(char[] cbuf, int off, int len) throws IOException {
      for(int i=0; i< len; i++)
         write(cbuf[off+i]);
   }

   @Override
   public void write(String str, int off, int len) throws IOException {
      for(int i=0; i< len; i++)
         write(str.charAt(off+i));
   }

   //… Weitere write()-Methoden uebrschreiben

}
```

Konstruktor erwartet einen `Writer`

Entscheidende write-Methode überschreiben

Hier: Wandle alle Großbuchstaben in Kleinbuchstaben um

Überschreibe die anderen write-Methoden unter Verwendung der `write(int c)`

`public abstract class FilterReader extends Reader`

Methoden von `FilterReader`:

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	**close**() Closes the stream and releases any system resources associated with it.
void	**mark**(int readAheadLimit) Marks the present position in the stream.
boolean	**markSupported**() Tells whether this stream supports the mark() operation.
int	**read**() Reads a single character.
int	**read**(char[] cbuf, int off, int len) Reads characters into a portion of an array.
boolean	**ready**() Tells whether this stream is ready to be read.
void	**reset**() Resets the stream.
long	**skip**(long n) Skips characters.

Liest Zeichen aus dem Stream und gibt es als `int` (Unicode-Zeichen) zurück.

-1, falls das Ende erreicht wird

Liest `len` Zeichen aus dem Stream und legt diese im Char-Array `cbuf` ab.
Es wird die Anzahl der gelesenen Zeichen zurückgegeben
-1, falls das Ende erreicht wird

- Eigenen `FilterReader` schreiben (ableiten von `FilterReader`)
  - Sollte mindestens die `read()` Methode überschreiben!

Eigener `FilterReader`


```
public class MyFilterReader extends FilterReader{

   protected MyFilterReader(Reader in) {
      super(in);
   }

   @Override
   public int read() throws IOException {
      return super.read() - 32;
   }

   @Override
   public int read(char[] cbuf, int off, int len) throws IOException {
      int result = super.read(cbuf, off, len);

      for(int i=0; i<result; i++)
         cbuf[off+i] = (char) (cbuf[off+i]-32);

      return result;
   }


}
```

Konstruktor erwartet einen `Reader`

`read()`-Methode überschreiben

Hier: Alle Zeichen um 32 Unicodezeichen verschieben

Ggf. überschreibe die anderen read-Methoden

Programmieraufgabe zu `FilterReader` & `FilterWriter`

Sie wollen nun Einträge in Ihre Log-Datei von letzter Stunde verschlüsseln, so dass kein anderer Nutzer die Datei lesen kann.

- Erzeugen Sie ein neues Eclipse-Package „uebung02“ und kopieren Sie Ihr Programm von letzter Stunde in das Package.
- Schreiben Sie nun einen eigenen `FilterWriter`, der den Ausgabestrom so manipuliert, dass die Daten nicht mehr einfach von Menschen lesbar sind
  - Bspw.: ändern Sie jedes zu übertragene Zeichen in ein anderes.
- Tätigen Sie wieder ein paar Ein- und Auszahlungen und schauen Sie sich die Einträge in Ihrer Log-Datei an.

- Schreiben Sie nun einen eigenen `FitlerReader`, welcher die Verschlüsselung beim Lesen des Eingabestroms rückgängig macht.
- Schreiben Sie eine neue Methode `decodeLogFile()`, welche den dekodierten Inhalt der Datei auf der Konsole ausgibt.
- Rufen Sie die Methode `decodeLogFile()` auf und kontrollieren Sie das Ergebnis.

Einführung in GUI-Programmierung

javax.swing

GUI – Graphical User Interface („Grafische Benutzerschnittstelle“)

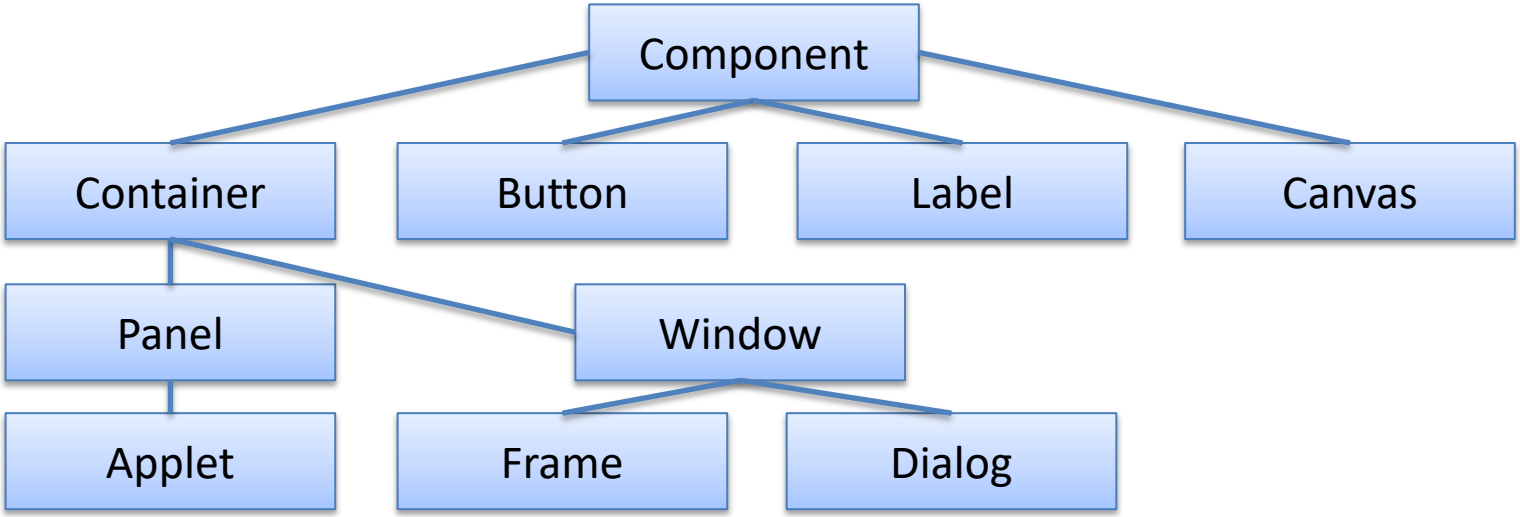
- Anschaulichere und leichtere Möglichkeit zur Dateneingabe und Kontrolle von Programmen
- Stellt Textfelder, Schaltknöpfe, Menüs, usw. zur Verfügung
- Java SE stellt Bibliotheken zur einfachen Implementierung von GUIs zur Verfügung
  - Swing:
    - Früher (Java 1.2 ca. 1998): Swing als fester Bestandteil der JRE
    - Entwickelt von Sun
    - Baut auf dem älteren Abstract Window Toolkit (AWT) auf.
      - Erweiterungen: Drag & Drop, neue Panels und Layouts, weitere Komponenten
  - JavaFX:
    - Heute(Seit Java SE 7 Update 6): JavaFX als fester Bestandteil von Java SE x86
    - Mittlerweile aber wieder aus Java SE ausgegliedert
    - Soll Swing ablösen
    - Schnell erstellbare neue UI-Komponenten (per CSS gestaltbar).

Heute: Kurze Einführung in **Swing**

- Java Swing-API ist umfangreich und sehr flexibel
- Ist im Package `javax.swing` verankert
  - Bietet ca. 18 weitere Unterpakete:
    - `border, event, table, text, tree,` usw.

Grundlage für die grafische Entwicklung ist AWT (Abstract Window Toolkit).

Ein Auszug aus der Klassenhierarchie (`java.awt`):

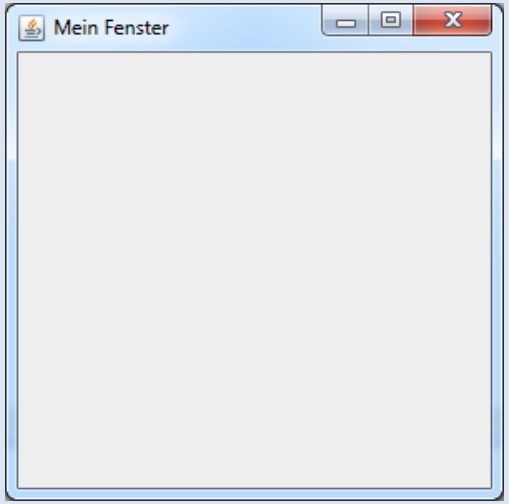


Fenster erstellen mit Swing

Wir wollen nun eine einfache Nutzerschnittstelle entwerfen:

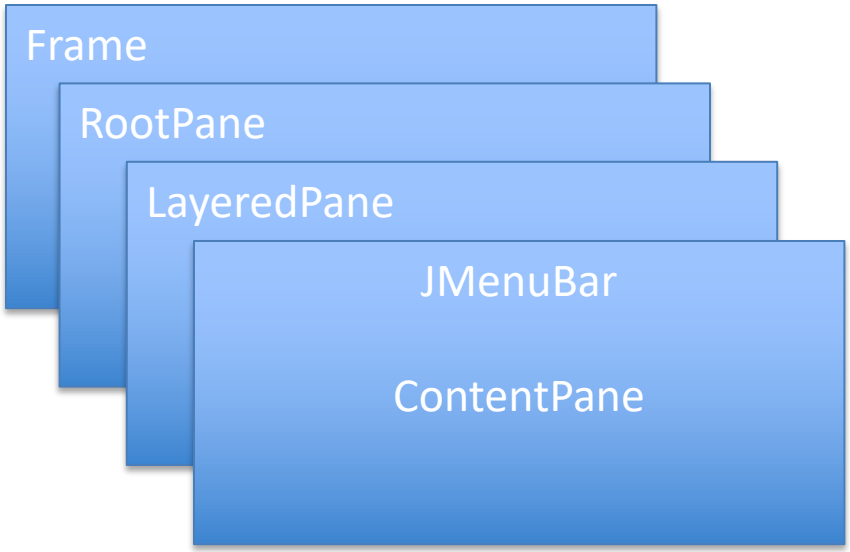
- Mit Hilfe der Klasse `JFrame` aus dem Packet `javax.swing` können wir sehr leicht ein einfaches bewegliches plattformabhängiges Fenster erstellen.
- Die Klasse bringt sehr viele Methoden und Eigenschaften mit, bspw.:
  - `setSize(width, height); // Legt Fenstergröße fest`
  - `setTitle(String title); // Legt Fenstertitel fest`
- Daher am besten von `JFrame` erben:

```
// Beispielcode:
import javax.swing.JFrame;

public class Gui extends JFrame{

   public Gui(){
      this.setTitle(„Mein Fenster");
      this.setSize(300, 300);

      this.setVisible(true);
   }
}
```



Fenster mit Elementen füllen

Allgemeiner Aufbau:



Unser `JFrame` beinhaltet das `JRootPane` als einziges Kind

- Stellt die `ContentPane` zur Verfügung
  - Ist die Basis-Komponente für alle Unterkomponenten

Unserem `JFrame` können nun direkt Elemente (`Components`) hinzugefügt werden:

- `this.add(new JButton(„OK“)) // Fügt einen OK-Button hinzu`

Oder man definiert seine eigene `ContentPane` und gibt diese dem `JFrame`:

- `JPanel contentPane = new JPanel();`
- `setContentPane(contentPane);`

Komponenten in Swing

Als Komponenten stehen sämtliche Bedienelemente zur Verfügung:

- `JButton, JTextField, JLabel`, usw.
- Aber auch neue `JPanel` (Füllwänd)

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextField;

public class Gui extends JFrame{

   private JButton ok_btn;
   private JTextField txt_field;

   public Gui(){

      this.setTitle("Mein Fenster");
      this.setSize(300, 300);

      this.ok_btn = new JButton("OK");
      this.txt_field = new JTextField();

      this.add(this.txt_field);
      this.add(this.ok_btn);

      this.setVisible(true);
   }
}
```

Die Komponenten überlagern sich.
=> Wir brauchen also ein Layout

Layouts

Um die Elemente auf eine bestimmte Art anzuordnen, brauchen wir ein Layout.

- Das Packet `java.awt` hält verschiedene Layouts bereit:
  - `BorderLayout, CardLayout, FlowLayout, GridBagLayout, GridLayout`
- Oder auch `javax.swing`:
  - `BoxLayout, Overlayout, GroupLayout`

- Einen guten Überblick über die verschiedenen Layouts gibt das Oracle Java Tutorial: https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

Hier als Beispiel:

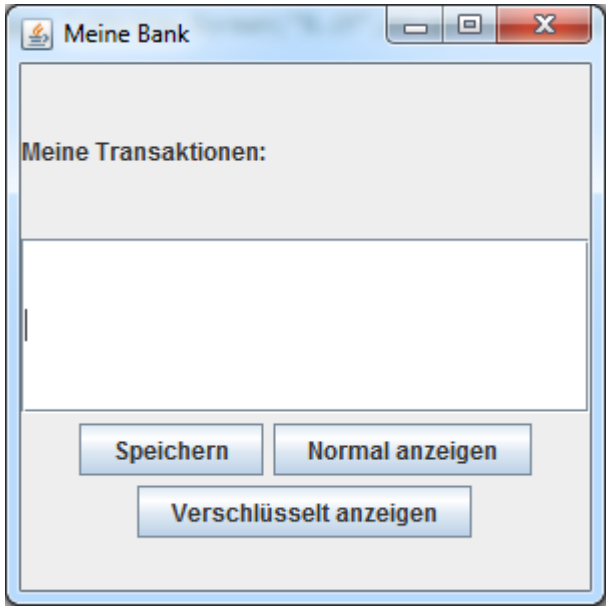
```
…
this.setLayout(new GridLayout(2, 1));
this.add(this.txt_field);
this.add(this.ok_btn);
```

Programmieraufgabe: GUI

Implementieren Sie nun für Ihre Bankkonto-Anwendung eine Einfache GUI,

- die ein Textfeld für die Ein- und Auszahlungen bereithält
- Zudem einen Button „Speichern“
- Und 2 Button „Normal anzeigen“ „Entschlüsselt anzeigen“

Ihre GUI kann beispielsweise wie folgt aussehen:



Button mit Leben füllen

Um nun auf Button-Klicks reagieren zu können, müssen wir das Interface `ActionListener` aus dem Packet java.awt.event implementieren:

- Das geht sehr einfach über eine anonyme innere Klasse
- Oder seit Java 8 über Lambda-Ausdrücke

```
// Einfaches Beispiel:

    this.txt_field = new JTextField();
    this.ok_btn = new JButton("OK");
    this.ok_btn.addActionListener(new ActionListener() {

      @Override
      public void actionPerformed(ActionEvent e) {

        txt_field.setText("OK");
      }
    });

// Per Lambda-Ausdruck:
this.ok_btn.addActionListener((ActionEvent e)-> txt_field.setText("OK"));
```

Hausaufgabe

Versuchen Sie nun Ihre eben erstellte GUI mit Leben zu füllen, indem Sie beim Klick des entsprechenden Buttons die folgenden Aktionen durchführen:

- Speichern: Speichert den Text im Textfeld in eine Datei
- Normal anzeigen: Zeigt den Text von der Datei im Klartext an
- Verschlüsselt anzeigen: Zeigt den Text aus der Datei verschlüsselt an
  - Nehmen Sie hierzu den FilterReader aus der vorherigen Aufgabe