Hochschule Harz	FB Automatisierung und Informatik
Programmierung2	Dipl.-Inf., Dipl.-Ing. (FH) M. Wilhelm
Tutorial / Hausaufgabe 05:	„Programmierung 2“ für MI / WI Thema: **Binäres Lesen und Schreiben, Thread**

Versuchsziele

Kenntnisse in der Anwendung von:

- Benutzen von
  - DataOutputStream
  - DataInputStream
  - Middleware
  - Thread

Tutorial- Hausaufgabe05: Data Mining: Auswahl von Bestellungen

In dieser Aufgabe soll für das unten abgebildete Fenster die Funktionalität implementiert werden.

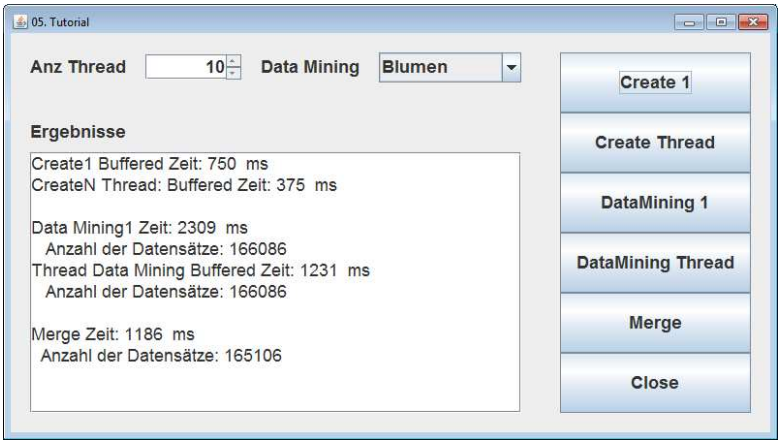


Abbildung 1 Musterlösung

Einleitung

Ihre Firma mit Namen „odnalaz“ hat mehrere Tausend Bestellungen am Tag erhalten. Nun soll eine Analyseprogramm, Data Mining, erstellt werden. Dazu sollen folgende Schritte durchgeführt werden:

1. Die Datensätze der Bestellungen enthalten Namen und Adressen der Kunden. Die müssen in einem ersten Schritt entfernt werden und in eine anonymisierte Datei geschrieben (create1 bzw. CreateN_Files, Maske *.bst).
2. Nun wird ein Begriff aus der ComboBox ausgewählt, und nach diesem werden die Datensätze durchsucht. Die gefundene Datensätze werden in eine neue Datei geschrieben (*.dm).
3. Am Schluss werden alle Dateien mit den gefundenen Datensätzen in eine gemeinsame Datei „Aufgabe05.merge“ geschrieben.
4. Da die Firma viele Bestellungen bekommen hat, werden Threads eingesetzt. Das heißt, die Aufgaben werden auf einzelne Thread verteilt.
5. Zu Testzwecken soll auch eine einfache Lösung, also ohne Threads, programmiert werden.

Hinweise:

- Da die Bestellungen „**real**“ nicht existieren, werden diese im ersten Schritt, „create1“ und „createThread“, in einer Schleife erzeugt. Dazu werden 1000000 Instanzen der Klasse Bestellung erzeugt und in eine Datei gespeichert.

- Da das ganze Programm Thread-orientiert ist, werden die Daten auf „MAX-Thread“, normal 10, aufgeteilt.
- Klasse Bestellung
    - Der Konstruktor der Klasse Bestellung ist schon fertig implementiert.
    - Die Suchbegriffe werden im Attribut „bez“, Bezeichnung, gespeichert.
- Um eine bessere Wartung des Programms zu erlangen, wird eine Klasse „Middleware“ eingeführt. Diese empfängt die Aufrufe aus dem JFrame und verarbeitet diese. Damit wird die grafische Oberfläche von der eigentlichen Verarbeitung entkoppelt.
- Ein weiterer Vorteil ist es, das man nun **im JFrame „Aufgabe05“ nichts ändern oder eintragen muss**.

1. Teilaufaufgaben: Projekt erstellen und aufbauen

- Erstellen Sie ein neues Eclipse–Projekt:
    - Projektname: Aufgabe05
- Erstellen Sie eine neue Klasse:
    - Menü File, Eintrag New, Eintrag class
    - Name: Aufgabe05
- Erstellen Sie folgenden Klassen und kopieren Sie die Quellcodes von meiner Homepage:
    - Aufgabe5
    - Bestellung
    - Middleware
    - CreateThread
    - DataMiningThread
    - Basic

Oder

- Sie laden das Musterprojekt für die Aufgabe05 runter.
- Danach starten Sie Eclipse.
- Menü Datei
- Eintrag „Import“

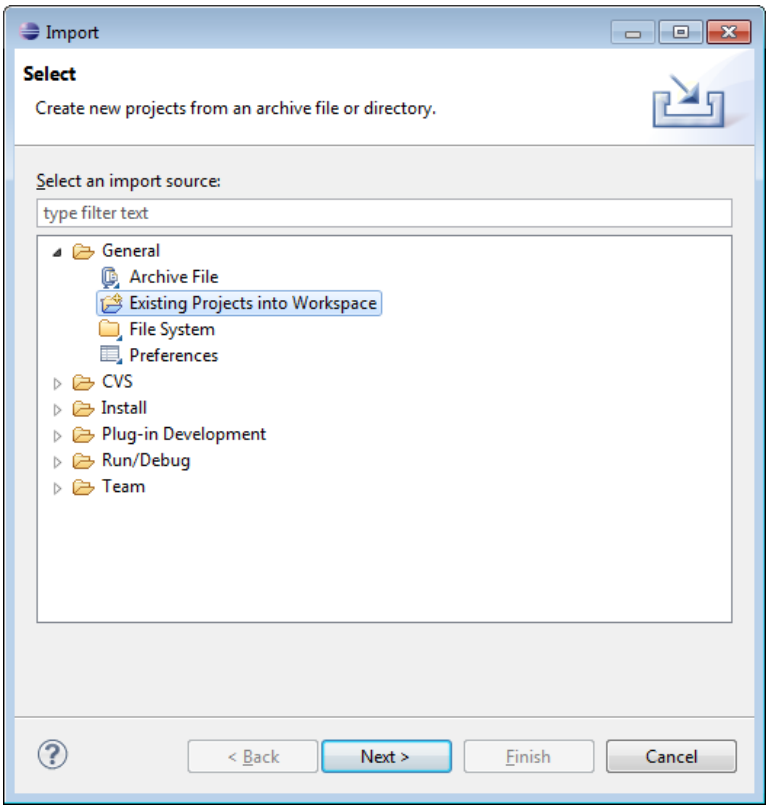


Abbildung 2 Import-Dialog

Auswahl: Existing Projects into Workspace

Nun das Projekt auswählen mit dem Schalter „Browse“

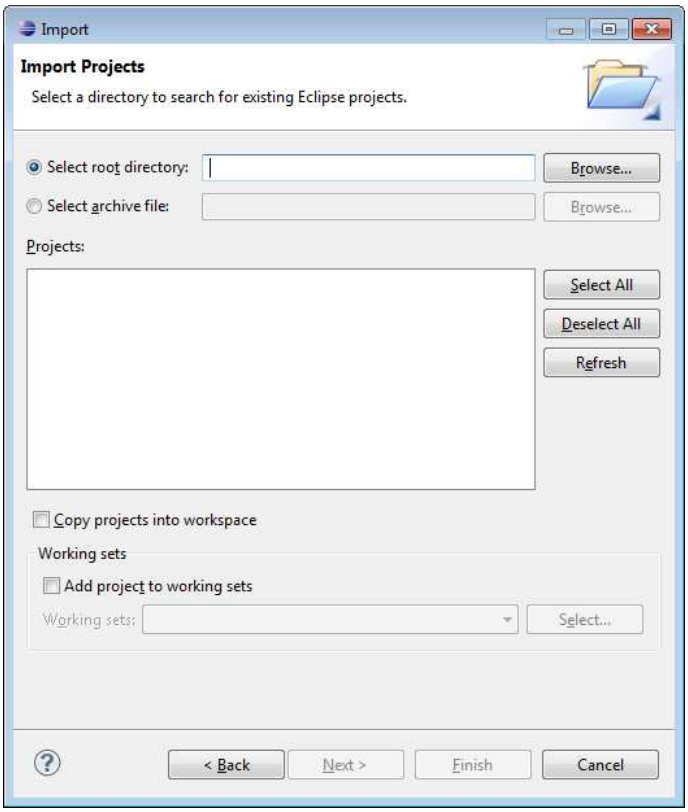


Abbildung 3 Zweiter Import-Dialog

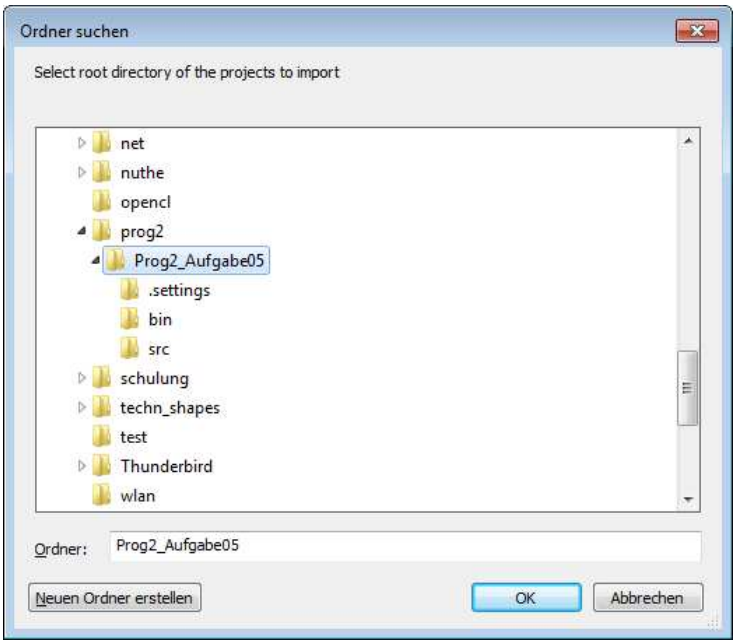


Abbildung 4 Auswahl des Projekt-Pfades

Ergebnis:

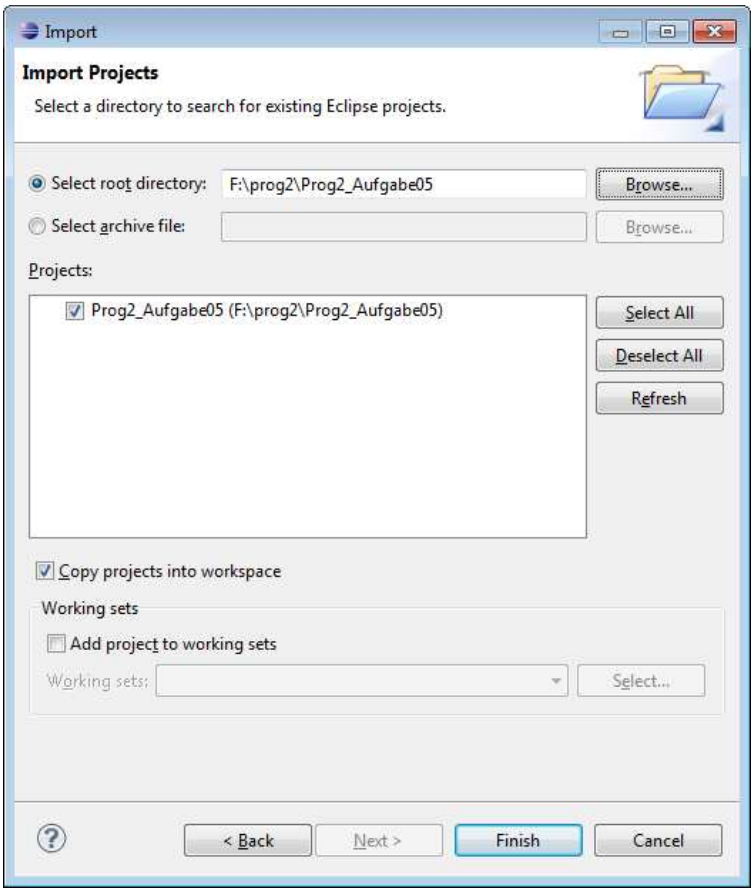


Abbildung 5 Vorhandenen Projekt wurde importiert

Klassenübersicht:

- Aufgabe05
  - Hauptframe, keine Änderung notwendig

- Basic
  - liefert Basis-Routinen, keine Änderung notwendig

- Bestellung
  - Speichert eine Bestellung
  - hier sind Änderungen notwendig:
    - Methode write2File
    - Methode readfromFile

- Middleware hier sind Änderungen notwendig
  - Verwaltet die Logik:
    - Create1_Files
      - Erzeugt alle „MaxThread“ Dateien in einer Schleife. Hier werden keine Threads benötigt.
    - CreateN_Files
      - Erzeugt „MaxThread“ Thread
      - Jeder Thread erzeugt eine Datei
    - DataMining1
      - Liest, prüft und schreibt alle „MaxThread“ Dateien in einer Schleife. Hier werden keine Threads benötigt.
    - DataMiningN
      - Erzeugt „MaxThread“ Thread
      - Jeder Thread liest, prüft und schreibt eine Datei.
    - Merge_Files
      - Liest alle „MaxThread“ Dateien in einer Schleife ein.
      - Sammelt diese in einer Liste
      - Speichert die Liste in eine „Mergedatei“.
      - Hier werden keine Threads benötigt.

- CreateThread
  - Erstellt als Thread **eine** Datei.
  - Wird von Middleware aufgerufen.
  - hier sind Änderungen notwendig.

- DataMiningThread
  - Liest als Thread **eine** Bestellung-Datei ein.
  - Prüft alle Bestellungen, ob sie den Suchstring enthalten.
  - Speichert die gefundenen Bestellungen in eine dm-Datei
  - Wird von Middleware aufgerufen.
  - hier sind Änderungen notwendig.

Überblick

Mit Create werden 10 Dateien mit jeweils 100.000 Bestellungen erzeugt.
In Data-Mining werden diese Dateien gelesen und nach dem Suchbegriff gefiltert.
Die gefilterten Daten werden danach in eine „*.dm“ gespeichert.
Mit dem Schalter „Merge“ werden alle 10 Dateien in eine Datei zusammengefasst.

Create1

Mit zwei Schleifen werden anzThread-Dateien erzeugt.
Pro Datei werden „anzItems“ Bestellungen in die jeweilige Datei geschrieben.
Prinzip:

```
for (i=0; i<anzThread…) {
    Öffnen der i-ten Datei
    Schreiben der Anzahl in die Datei
        for (j=0; i<anzItems…) {
          Bestellung erzeugen (Random)
          Bestellung schreiben
    }
}
```

CreateNThread

Mit einer Schleife werden anzThread-Dateien erzeugt.
Die Dateien werden aber mittels Threads erzeugt

Datamining1

Mit zwei Schleifen werden anzThread-Dateien gelesen und gefiltert erzeugt.
Pro Datei werden „anzItems“ Bestellungen aus der jeweiligen Datei gelesen.
Die gelesene Bestellung wird nach erfolgreicher Filterung in die Liste eingetragen.
Nach dem Lesen wird die Liste in die i.te Data Mining-Datei geschrieben.
Prinzip:

```
for (i=0; i<anzThread…) {
    Öffnen der i-ten Datei
    Lesen der Anzahl in die Datei
    Liste löschen
        for (j=0; i<anzItems…) {
          Bestellung erzeugen ()
          Bestellung lesen
          Bestellung prüfen und ggfs. in die Liste eintragen
    }
    Liste speichern
}
```

DataminingNThread

Mit einer Schleife werden anzThread-Dateien erzeugt.
Die Dateien werden aber mittels Threads gelesen und geschrieben.

Merge

Mit einer Schleife werden anzThread-Dateien gelesen und in einer Liste gespeichert.
Pro Datei werden „anzItems“ Bestellungen aus der jeweiligen Datei gelesen.
Die gelesene Bestellung wird in die Liste eingetragen.
Nach dem Lesen wird die Liste in die Merge-Datei geschrieben.

Prinzip:

```
Liste löschen
for (i=0; i<anzThread…) {
    Öffnen der i-ten Data Mining Datei
    Lesen der Anzahl in die Datei
        for (j=0; i<anzItems…) {
          Bestellung erzeugen ()
          Bestellung lesen
          in die Liste eintragen
    }
}
Liste in die Merge-Datei speichern
```

2. Teilaufaufgaben: In der Klasse Bestellung die I/O einbauen

- Methode write2File
  - Schreiben der Attribute
    - Als erstes die Version, Konstante in der Klasse
    - Artikel
    - Preis
    - Bezeichnung
  - Verwendet wird der Parameter „dout“
- Methode readfrom2File
  - Lesen der Attribute
    - Als erstes die Version in eine lokale Variable eingelesen
      - Vergleich mit der Konstante in der Klasse
    - Lesen von Artikel
    - Lesen von Preis
    - Lesen von Bezeichnung
  - Verwendet wird der Parameter „din“

3. Teilaufaufgaben: Klasse Middleware

- Methode create1_File
  - **Diese Methode arbeitet ohne Threads**
  - Erstellen von „anzThread“-Dateien in einer Schleife
    - Der Name der Datei wird mit der Methode „getBstFilename“ bestimmt.
    - Verwendet wird ein FileOutputStream
    - Verwendet wird ein BufferedOutputStream
    - Verwendet wird ein DataOutputStream
    - Schreiben der Anzahl der Bestellungen in den Stream (maxItems)
      - Erzeugen einer Instanz der Klasse „bestellung“
      - Aufrufen der Methode „write2File“
      - Insgesamt werden „maxItems“ Bestellungen in die Datei geschrieben
      - Bitte die jeweilige Datei mit Close schließen
    - Eintragen der Zeit in den Editor:
      - Basic.MessageBox("Habe fertig", "Die Dateien wurde erzeugt");
      - editor.append("\nCreate1 Buffered Zeit: "+(t2-t1)+" ms")

- Methode createN_File
  - **Diese Methode arbeitet mit Threads**
  - Erstellen eines Arrays von „anzThread“-Threads, Klasse „CreateThread“
  - Erstellen von „anzThread“-Instanzen in Array
    - Parameter:
      - i
      - maxItems
      - getBstFilename
  - Starten der Threads in einer Schleife
  - Warten, dass alle Threads fertig beendet werden
  - Eintragen der Zeit in den Editor:
    - Basic.MessageBox("Habe fertig", "Die Dateien wurde erzeugt");
    - editor.append("\nCreate1 Buffered Zeit: "+(t2-t1)+" ms")

4. Teilaufaufgaben: Klasse CreateThread

- Methode run
  - Erstellen einer Datei mit „maxItems“-Bestellungen
  - Verwendet wird ein FileOutputStream
  - Verwendet wird ein BufferedOutputStream
  - Verwendet wird ein DataOutputStream
  - Schreiben der Anzahl der Bestellungen in den Stream (maxItems)
    - Erzeugen einer Instanz der Klasse „bestellung“
    - Aufrufen der Methode „write2File“

- Insgesamt werden „maxItems“ Bestellungen in die Datei geschrieben
- Bitte die Datei mit Close schließen

Hausaufgabe 05

5. Teilaufaufgaben: Klasse Middleware

- Methode dataMining1
  - **Diese Methode arbeitet ohne Threads**
  - Liest, prüft und schreibt „anzThread“-Dateien in einer Schleife:
    - Der i.te Name der Quelldatei wird mit der Methode „getBstFilename“ bestimmt.
    - Der i.te Name der Zieldatei wird mit der Methode „getDataMiningFilename“ bestimmt.
    - Einlesen der Bestellungen der i.te Datei:
    - Lesen der Anzahl der Bestellungen aus dem Stream
      - Verwendet wird ein FileInputStream
      - Verwendet wird ein BufferedInputStream
      - Verwendet wird ein DataInputStream
      - Pro Datei werden die Bestellungen in eine Liste eingelesen
        - Lesen der Anzahl
        - for-Schleife
        - Prüfen, ob der Suchstring in „bez“ enthalten ist (Methode ind…)
        - Wenn der Suchstring gefunden wurde, wird die aktuelle Bestellung in die Liste eingefügt
    - Schreiben der gefundenen Bestellungen der i.te Datei:
      - Verwendet wird ein FileOutputStream
      - Verwendet wird ein BufferedOutputStream
      - Verwendet wird ein DataOutputStream
      - Schreiben der Anzahl
      - For-Schleife über die Liste der Bestellungen
        - Aufrufen der Methode „write2File“
      - Bitte die jeweilige Datei mit Close schließen
      - Summieren der Anzahl der gefundenen Datensätze in der Variablen „anz“
  - Eintragen der Zeit in den Editor:
    - Basic.MessageBox("Habe fertig", "Data Mining: Die Dateien wurde verarbeitet");
    - editor.append("\nData Mining1 Zeit: "+(t2-t1)+" ms");
    - editor.append("\nAnzahl der Datensätze: "+anz);

- Methode createN_File
  - **Diese Methode arbeitet mit Threads**
  - Erstellen eines Arrays von „anzThread“-Threads, Klasse „DataMiningThread“
  - Erstellen von „anzThread“-Instanzen in Array
    - Parameter:
      - i
      - searchBez
      - getBstFilename
      - getDataMiningFilename
  - Starten der Threads in einer Schleife
  - Warten, dass alle Threads fertig beendet werden
  - Summieren der Anzahl der gefundenen Bestellungen pro Threads in einer Schleife
  - Eintragen der Zeit in den Editor:
    - Basic.MessageBox("Habe fertig", "Data Mining: Die Dateien wurde verarbeitet");
    - editor.append("\nThread Data Mining Buffered Zeit: "+(t2-t1)+" ms");
    - editor.append("\nAnzahl der Datensätze: "+anz);

6. Teilaufaufgaben: Klasse DataMiningThread:

- Methode run
  - Liest eine Datei von Bestellungen
  - Verwendet wird ein FileInputStream
  - Verwendet wird ein BufferedInputStream
  - Verwendet wird ein DataInputStream
  - Einlesen der Anzahl der Bestellungen in den Stream
  - Einlese-Schleife:
    - For-Schleife
    - Prüfen, ob der Suchstring in „bez“ enthalten ist (Methode ind…)
    - Wenn ja, wird er in eine Liste eingetragen (oder gleich gespeichert)
  - Nach der Schleife werden alle gefundenen Bestellungen in die „dataMiningFilename“ Datei geschrieben.
    - Verwendet wird ein FileOutputStream
    - Verwendet wird ein BufferedOutputStream
    - Verwendet wird ein DataOutputStream
    - Schreiben der Anzahl der Bestellungen in den Stream (Listen-Anzahl)
    - Schleife über die Liste der Bestellungen
      - Aufrufen der Methode „write2File“
    - Bitte die Datei mit Close schließen
  - In der globalen Variable wird die Anzahl der gefundenen Datensätze gespeichert

7. Teilaufaufgaben: Klasse Middleware

- Methode Merge_Files
  - **Diese Methode arbeitet ohne Threads**
  - Erstellen einer ArrayList für die Bestellungen (Dateien Data-Mining)
  - Lesen der „anzThread“-Dateien in einer Schleife
    - Der i.te Name der Quelldatei wird mit der Methode „getDataMiningFilename“ bestimmt.
    - Verwendet wird ein FileInputStream
    - Verwendet wird ein BufferedInputStream
    - Verwendet wird ein DataInputStream
    - Lesen der Anzahl der Bestellungen aus dem Stream
    - Einlesen der Bestellungen der i.te Datei mittels For-Schleife
      - Pro Datei werden die Bestellungen in eine Liste eingelesen
    - i.te-Datei wird geschlossen
  - **Nach** der Schleife werde alle gefundenen Bestellungen der die Merge-Datei gespeichert (getMergeFilename).
    - Verwendet wird ein FileOutputStream
    - Verwendet wird ein BufferedOutputStream
    - Verwendet wird ein DataOutputStream
    - Schreiben der Anzahl der Bestellungen in den Stream (Listenanzahl)
    - Schleife über die Liste der Bestellungen des ArrayList.
      - Aufrufen der Methode „write2File“
    - Bitte die jeweilige Datei mit Close schließen.
  - Eintragen der Zeit in den Editor:
    - Basic.MessageBox("Habe fertig", "Merge: Die Dateien wurden zusammengefasst");
    - editor.append("\nMerge Zeit: "+(t2-t1)+" ms");
    - editor.append("\nAnzahl der Datensätze: "+liste.size());

Anhang:

IO mit Klassen

```
class Student {
     String name="Meier";
     int mnr=12345;

  public void write2File(DataOutputStream dout) throws IOException {
     dout.writeUTF(name);
     dout.writeInt(mnrs);        // oder auch writeDouble
  }

  public void readfromFile(DataInputStream din) throws IOException  {
     name = din.readUTF();
     mnr = din.readInt();        // oder auch readDouble
  }
```

Aufruf:

```
     FileOutputStream fout= new FileOutputStream(  "test.bin" );  // byteweise
     BufferedOutputStream bout=new BufferedOutputStream(fout);
     DataOutputStream dout=new DataOutputStream(bout);
     Student std = new Student();
     std.write2File(dout);


     FileInputStream fin= new FileInputStream(  "test.bin" );  // byteweise
     BufferedInputStream bin=new BufferedOutputStream(fin);
     DataOutputStream din=new DataOutputStream(bin);
     Student std = new Student();
     std.readfrom2File(din);
```

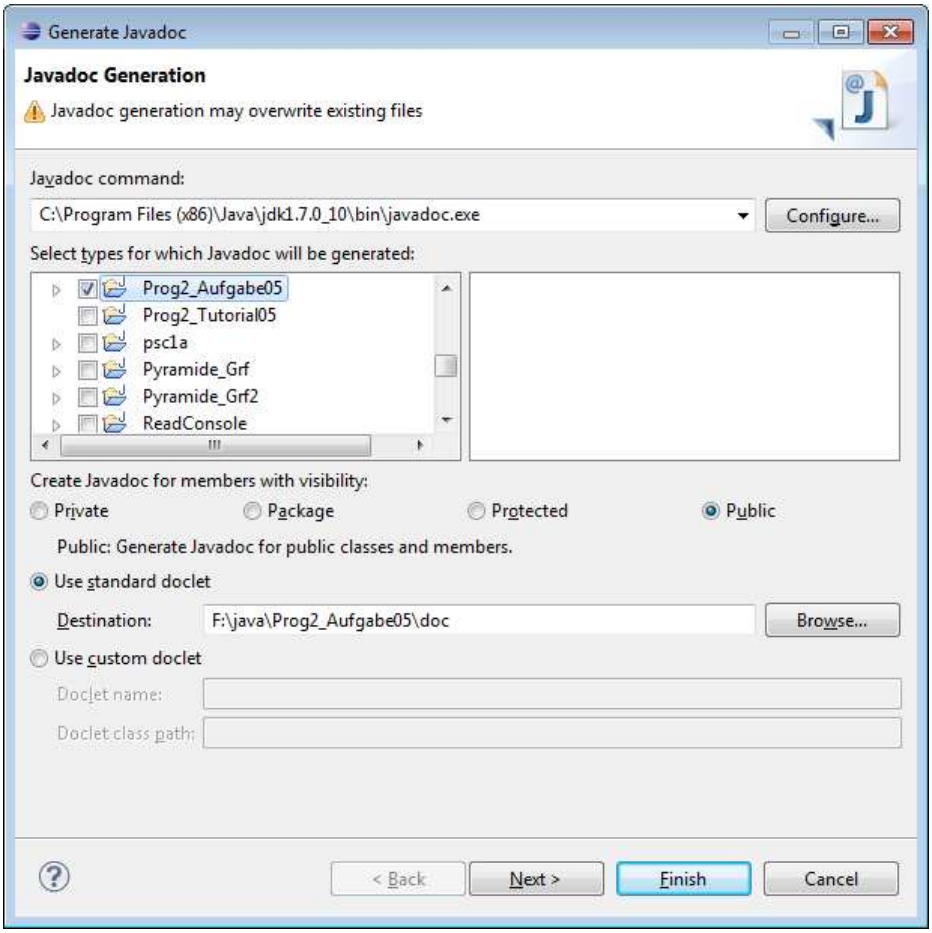
Threads

// Das Beispiel erzeugt fünf Thread:

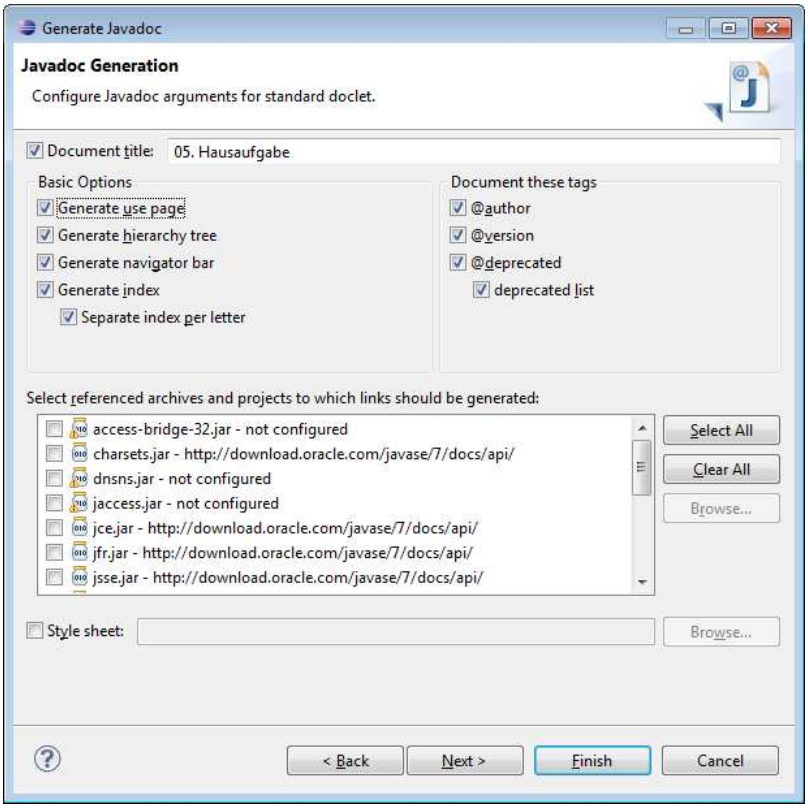
```
public void test() {
    myThread[]  th = myThread[5]
    for (int i=0; i<th.length; i++) {
        th[i] = myThread(i+1);
    }
    for (int i=0; i<th.length; i++) {
        th[i].start();
    }

    try {
        for (int i=0; i<th.length; i++) {
            th[i].join();     // wartet darauf, dass Thread(i) fertig ist
        }
    } catch (InterruptedException ex) {
          // syso(error)
    }
}  // test


// Threadklasse
class myThread  extends  Thread{
    int nr;

public myThread(int nr) {
    this.nr = nr;
}

public void run(){
    for (int i=0; i<100; i++) {
        syso("nr: “+nr+"    i: "+i);
    }
}

}
```

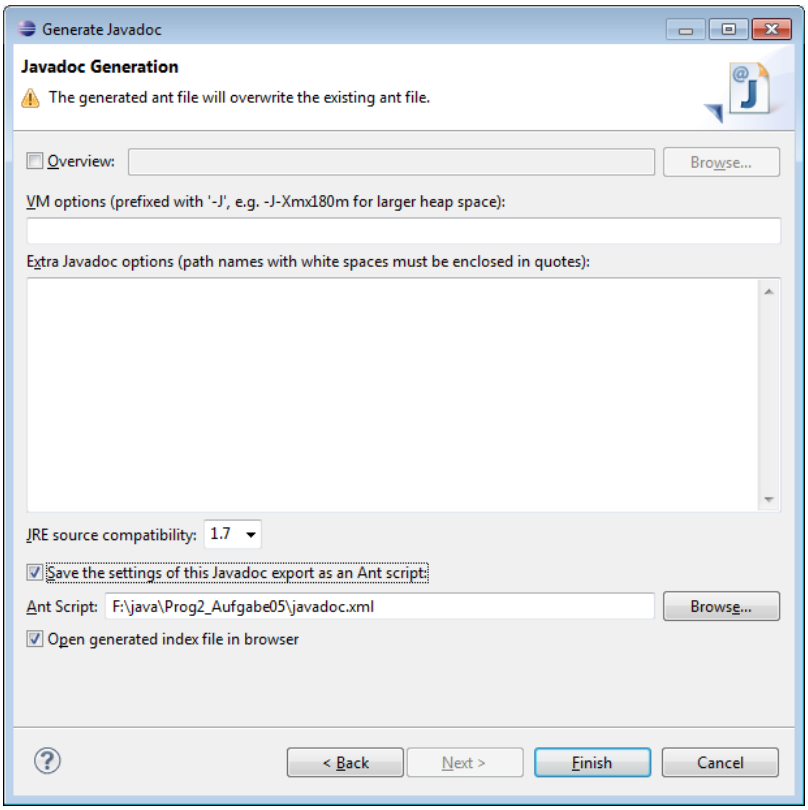
Javadoc

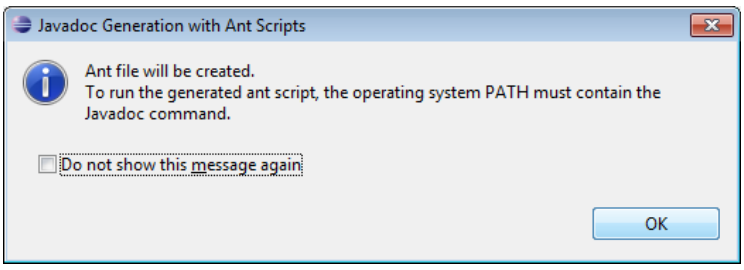
Menü Projekt:
Eintrag: JavaDoc

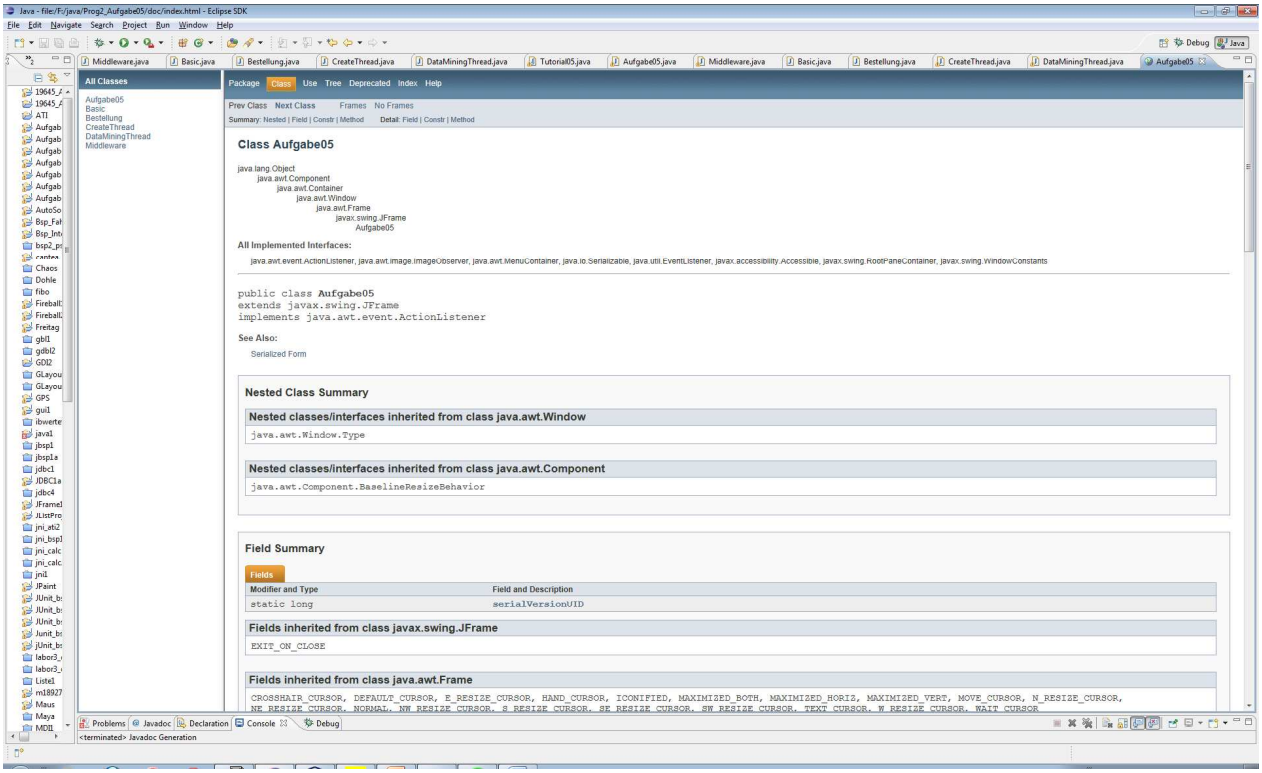


Hinweis:
JavaDoc Command ist der Pfad zum Programm „javadoc.exe“


Generate Javadoc
Javadoc Generation
Configure Javadoc arguments for standard doclet.
Document title: 05. Hausaufgabe
Basic Options
Generate use page
Generate hierarchy tree
Generate navigator bar
Generate index
Separate index per letter
Document these tags
@author
@version
@deprecated
deprecated list
Select referenced archives and projects to which links should be generated:
access-bridge-32.jar - not configured
charsets.jar - http://download.oracle.com/javase/7/docs/api/
dnsns.jar - not configured
jaccess.jar - not configured
jce.jar - http://download.oracle.com/javase/7/docs/api/
jfr.jar - http://download.oracle.com/javase/7/docs/api/
jsse.jar - http://download.oracle.com/javase/7/docs/api/
Select All
Clear All
Browse...
Style sheet:
Browse...
< Back
Next >
Finish
Cancel


Generate Javadoc
Javadoc Generation
The generated ant file will overwrite the existing ant file.
Overview:
Browse...
VM options (prefixed with '-J', e.g. -J-Xmx180m for larger heap space):
Extra Javadoc options (path names with white spaces must be enclosed in quotes):
JRE source compatibility: 1.7
Save the settings of this Javadoc export as an Ant script:
Ant Script: F:\java\Prog2_Aufgabe05\javadoc.xml
Browse...
Open generated index file in browser
< Back
Next >
Finish
Cancel


Javadoc Generation with Ant Scripts
Ant file will be created.
To run the generated ant script, the operating system PATH must contain the Javadoc command.
Do not show this message again
OK


Java - file:/F:/java/Prog2_Aufgabe05/doc/index.html - Eclipse SDK
File Edit Navigate Search Project Run Window Help
All Classes
Aufgabe05
Basic
Bestellung
CreateThread
DataMiningThread
Middleware
Package Class Use Tree Deprecated Index Help
Prev Class Next Class Frames No Frames
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
Class Aufgabe05
java.lang.Object
java.awt.Component
java.awt.Container
java.awt.Window
java.awt.Frame
javax.swing.JFrame
Aufgabe05
All Implemented Interfaces:
java.awt.event.ActionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.util.EventListener, javax.accessibility.Accessible, javax.swing.RootPaneContainer, javax.swing.WindowConstants
public class Aufgabe05
extends javax.swing.JFrame
implements java.awt.event.ActionListener
See Also:
Serialized Form
Nested Class Summary
Nested classes/interfaces inherited from class java.awt.Window
java.awt.Window.Type
Nested classes/interfaces inherited from class java.awt.Component
java.awt.Component.BaselineResizeBehavior
Field Summary
Fields
Modifier and Type
Field and Description
static long
serialVersionUID
Fields inherited from class javax.swing.JFrame
EXIT_ON_CLOSE
Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR,
NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR
Problems Javadoc Declaration Console Debug
<terminated> Javadoc Generation