

Übung 1: *Installation + Test von Eclipse*

Führen Sie die Installation der Entwicklungsumgebung *Eclipse* durch gemäss Dokument "*InstallationTest.Eclipse.7.0.pdf*" auf der CD im Directory "Eclipse".

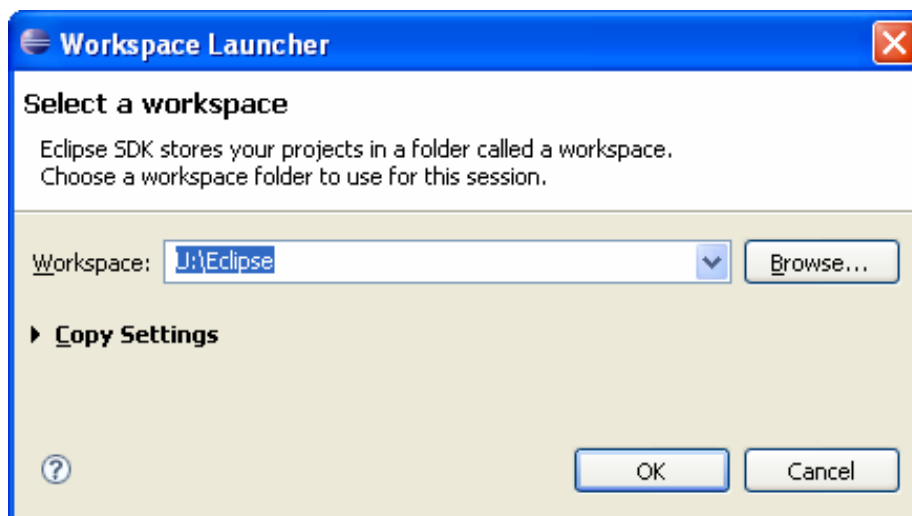
Hinweise:

Die Beschreibung "*InstallationTest.Eclipse.7.0.pdf*" ist ausgelegt für die Installation auf einem persönlichen Rechner.

Ihre persönliche Daten werden an der HSR als Drive \mathfrak{U} : *ge'mapped*.

Installieren Sie Eclipse im Directory \mathfrak{U} : \test (anstelle von C:\Program Files; das Directory **test** muss dabei explizit angelegt werden).

Beim Workspace-Launcher soll als Workspace (für die Meta-Daten von Eclipse) das Directory \mathfrak{U} : \Eclipse festgelegt werden.



Übung 2: *Hello World*

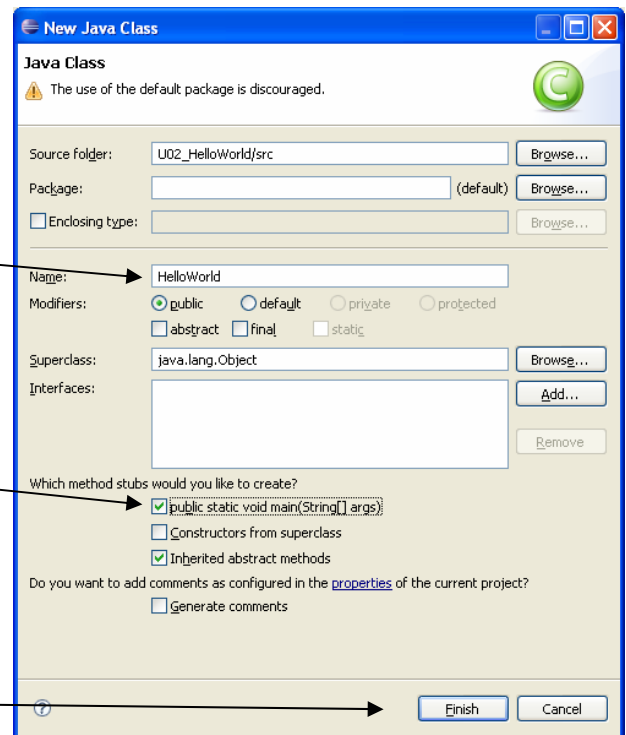
Erzeugen Sie ein neues Projekt mit dem Namen *U02_HelloWorld*.
Selektieren Sie dann den neuen 'Ordner' *U02_HelloWorld* im *Package Explorer* und wählen Sie *Menü:File>New>Class*.

Damit wird eine neue sog. *Klasse* erzeugt.

1. Wählen Sie als Namen *HelloWorld*:

2. Es soll ein '*main()*' erzeugt werden:

3. *Finish*.



Im *Package Explorer* erscheint ein neues File: *HelloWorld.java*
Das ist das Source-File indem das Programm erstellt wird.

Mit `system.out.println()` kann auf die Konsole geschrieben werden.
Für ein "Hello World" erweitern Sie das Programm wie folgt:

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World.");  
    }  
}
```

Bringen Sie das Programm zur Ausführung.

Verifizieren Sie die Ausgabe von `println()` auf der *Console*.

Übung 3: Standard-Eingabe

Um Eingaben von der Tastatur einlesen zu können muss eine Applikation die sog. *Standard-Eingabe (stdin)* lesen.

Das folgende Beispiel-Programm illustriert das Einlesen von Strings und Zahlen (ganze Zahlen (`int`) und Gleitkommazahlen (`double`)):

```
import java.util.Scanner;

public class StdIn {

    public static void main(String[] args) {

        // Vergindung zur Standard-Eingabe resp.zum Keyboard:
        Scanner eingabe = new Scanner(System.in);

        System.out.print("Ein String : ");
        // Einlesen eines Strings von der Standard-Eingabe:
        String str = eingabe.next();
        System.out.println("String = " + str);

        System.out.print("Ganze Zahl : ");
        // Einlesen einer Zahl von der Standard-Eingabe als Ganzzahl
        // vom Typ 'int':
        int i = eingabe.nextInt();
        System.out.println("Zahl = " + i);

        System.out.print("Gleikomma-Zahl : ");
        // Einlesen einer Zahl von der Standard-Eingabe als Gleitkommazahl
        // vom Typ 'double':
        double d = eingabe.nextDouble();
        System.out.println("Zahl = " + d);

        System.out.println("ENDE.");

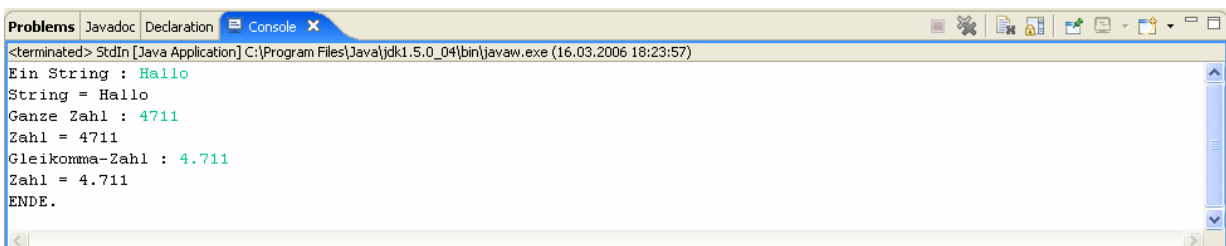
    }
}
```

Erzeugen Sie in *Eclipse* ein neues Projekt mit dem Namen *U03_StdIn* und programmieren Sie obiges Programm.

Hinweise:

Machen Sie bewusst Syntax-Fehler und schauen Sie, wie sich Eclipse verhält.

Bei der Ausführung sind die Tastatur-Eingabe im Tab *Console* vorzunehmen:



```
<terminated> StdIn [Java Application] C:\Program Files\Java\jdk1.5.0_04\bin\javaw.exe (16.03.2006 18:23:57)
Ein String : Hallo
String = Hallo
Ganze Zahl : 4711
Zahl = 4711
Gleikomma-Zahl : 4.711
Zahl = 4.711
ENDE.
```

Beobachten Sie was passiert, wenn Sie anstelle einer Zahl eine Zeichenkette eingeben, z.B. "Hallo".

Übung 4: Rechnen

Erstellen Sie ein Programm, das zuerst zwei ganze Zahlen von der Konsole einliest und von diesen dann die *Summe*, die *Differenz*, das *Produkt*, den *Quotienten* und den *Rest der Ganzzahldivision* berechnet und ausgibt, sowie daraufhin das selbe für zwei Gleitkommazahlen wiederholt.

(Hinweis: die entsprechenden Operatoren: **+**, **-**, *****, **/**)

Nachfolgend die ist ein Session-Log dargestellt, dass zeigt wie sich das Programm verhalten soll:

```
1. ganze Zahl = 8
2. ganze Zahl = 3
Summe      = 11
Differenz  = 5
Produkt    = 24
Quotient   = 2
Rest       = 2

1. Gleitkommazahl = 8
2. Gleitkommazahl = 3.1
Summe        = 11.1
Differenz    = 4.9
Produkt      = 24.8
Quotient     = 2.5806451612903225
Rest         = 1.7999999999999998
```

Übung 5: *Zeit-Umrechnung*

Erstellen Sie ein Programm, das eine Anzahl von *Sekunden* einliest und diese dann in die entsprechende Anzahl von *Tagen*, *Stunden*, *Minuten* und *Sekunden* umrechnet.

Das Programm soll sich wie im nachfolgenden Beispiel verhalten.
Programmabbruch bei *Sekunden* < 0.

Beispiel:

```
Sekunden   : 145
Tage       = 0
Stunden    = 0
Minuten    = 2
Sekunden   = 25
```

```
Sekunden   : 100000
Tage       = 1
Stunden    = 3
Minuten    = 46
Sekunden   = 40
```

```
Sekunden   : -1
Good Bye .
```

Übung 6: *Potenz-Rechnung*

Erstellen Sie ein Programm, das die Potenz zu einer beliebigen *Basis* und *Exponent* berechnet.

Das Programm soll sich wie im nachfolgenden Beispiel verhalten.
Programmabbruch bei einer *Basis* < 0.

Beispiel:

```
Basis:    2
Exponent: 3
2^3 = 8
```

```
Basis:    2
Exponent: 20
2^20 = 1048576
```

```
Basis:    10
Exponent: 3
10^3 = 1000
```

```
Basis:    -1
Good Bye.
```

Übung 7: *Eclipse* - 'New Project' und nachfolgend hineinkopieren von bestehenden Files

Zweck:

Wir wollen ein neues Projekt erstellen und bestehende Java-Files darin *hineinkopieren*.
Die Original-Files sollen dabei unberührt bleiben.

Durchführung:

Erstellen Sie in Eclipse ein Projekt mit dem Namen "*U03_StdIn_ML1*" (*ML*: Musterlösung).

Kopieren Sie die Musterlösung der "*Übung 3: Standard-Eingabe*" von www.letsch-informatik.ch: *Download>MAS>2_TrainingskursJava>u03_StdIn* in das neu erzeugte Projekt-Directory von *Eclipse* (befindet sich im *Workspace* unter *U:\Eclipse*).

Achtung: je nach Web-Browser wird dem Filename eine Extension ".htm" oder ".html" hinzugefügt. In diesem Fall den Filename von Hand wieder auf `StdIn.java` anpassen.

In Eclipse wird das neue File im *Package Explorer* noch nicht sichtbar!

Selektieren Sie nun im *Package Explorer* den Ordner *U03_StdIn_ML1*.
Dann im *Context-Menü* (rechte Maustaste) des obigen Ordners den **Refresh** wählen (oder einfach Funktionstaste **F5**).
Jetzt sollte das File sichtbar werden (im *default package*).

Bringen Sie das Programm zur Ausführung.

Hinweise:

- Sie können so auch beliebig viele Files (auch mit Sub-Directories) auf einmal in ein Projekt integrieren.
- man kann nach dem 'Copy' im Explorer den 'Paste' auch direkt im *Package Explorer* von Eclipse durchführen (somit kann man auf den *Refresh* verzichten ;-)

Übung 8: *Eclipse* - 'New Project' und nachfolgender Import von bestehenden Files

Zweck:

Wir wollen ein neues Projekt erstellen und bestehende Java-Files darin 'importieren' (d.h. ins Projekt hineinkopieren).

Die Original-Files sollen dabei unberührt bleiben.

Durchführung:

Kopieren Sie die komplette Musterlösung (*java*- und *pdf*-File) von www.letsch-informatik.ch>Download>MAS>2_TrainingskursJava>u03_StdIn nach z.B. `C:\TEMP\u03_StdIn`.

Erstellen Sie in Eclipse ein Projekt mit dem Namen "`U03_StdIn_ML2`" (*ML*: Musterlösung).

Importieren Sie in dieses Projekt obige Musterlösung mit:

- im *Package Explorer* das Projekt öffnen und den Folder 'src' selektieren

- Menü: *File* > *Import...* > *General* > *File System*

- *From directory:* `C:\TEMP\u03_StdIn`

- *Selektion:* `Stdin.java` (⇒ *CheckBox*)

`Stdin.java.pdf` (⇒ *CheckBox*)

- *Finish*

Verifizieren Sie auf dem Disk, ob die Files jetzt auch wirklich dort vorhanden sind wo Sie sie erwartet haben (im Verzeichnis *src*)!

Bringen Sie das Programm zur Ausführung.

Vergegenwärtigen Sie sich den Unterschied zur letzten Übung.

Übung 9: *Eclipse* - 'New Project' auf bestehenden Files

Zweck:

Wir wollen ein neues Projekt erstellen auf bereits bestehenden Java-Files.
D.h. wir arbeiten danach mit *diesen* Files.

Durchführung:

Kopieren Sie die komplette Musterlösung (*java*- und *pdf*-File) von www.letsch-informatik.ch/Download/MAS/2_TrainingskursJava/u03_StdIn zu sich irgendwo auf den Home-Drive (z.B. U:\TrainingsKursJava\u03_StdIn).

Hinweis: Projekte *nicht* 'von Hand' im *Workspace* (U:\Eclipse) anlegen!
☞ die Verwaltung des *Workspace* ist für *Eclipse* reserviert ;-)

Erstellen Sie in *Eclipse* ein Projekt mit dem Namen "U03_StdIn_ML3".

- *Contents*:

 Create project from existing source: `enable`
 (Create new project in workspace: `disable` (automatisch))
 Directory: ... Browse ... U:\TrainingsKursJava\u03_StdIn

- *Finish*

Beachten Sie die beiden neuen Files **.project** und **.classpath** im Projekt-Directory *u03_StdIn* !

Diese enthalten für *Eclipse* entsprechende Meta-Informationen über das Projekt.
Es sind XML-Files und können mit einem Text-Editor angesehen werden.

Bringen Sie das Programm zur Ausführung.

Vergegenwärtigen Sie sich den Unterschied zu den letzten Übungen.

Übung 10: *Eclipse* - Import eines bestehenden *Eclipse*-Projektes

Zweck:

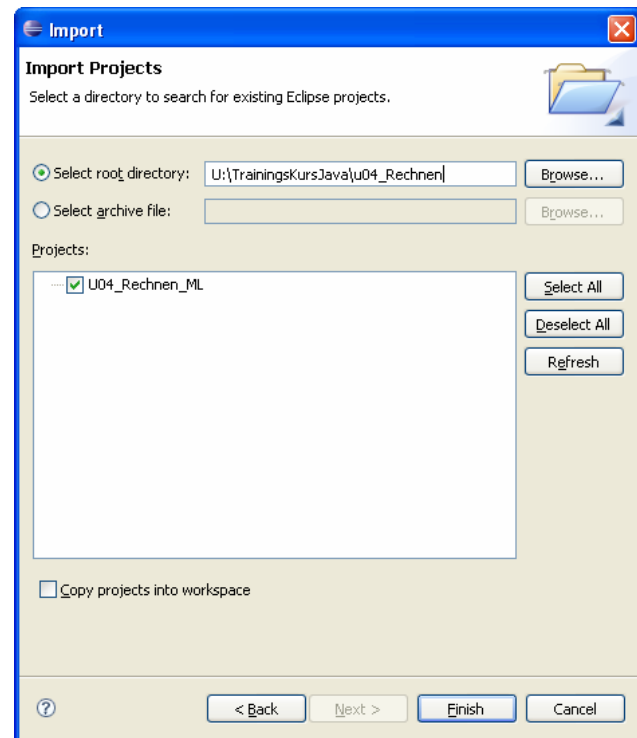
Wir wollen ein bestehendes *Eclipse*-Projekt (z.B. vom Büro oder Zuhause) in den *Workspace* importieren.

Durchführung:

1. Bestehendes *Eclipse*-Projekt komplett, d.h. insbesondere incl. der beiden Files *.classpath* und *.project*, an den Zielort kopieren (z.B. mit *MS-Explorer* oder mit *WinZip* dorthin auspacken).

2. Dann in *Eclipse*: *File* > *Import...* > *General* > *Existing Projects into Workspace*

Beachten Sie, dass nach dem Setzen von 'Select root directory:' das Projekt unter 'Projects' angezeigt wird. Dabei wird der *Projekt-Name* vom importierten Projekt übernommen (dieser muss nicht identisch sein wie das Directory, in welchem sich das Projekt befindet!).



Konkret:

Laden Sie die Musterlösung der Übung 4 "Rechnen" runter:

www.letschinformatik.ch/Download/MAS/2_TrainingskursJava/u04_Rechnen/u04_Rechnen.zip

Entpacken Sie das ZIP-File, z.B. nach `U:\TrainingsKursJava\u04_Rechnen` (Hinweis: **nicht** in den Workspace `U:\Eclipse!`)

Importieren Sie dieses Projekt gemäss obigen Angaben in *Eclipse*.

ACHTUNG:

- mit *File* > *Import...* > *Existing Projects into Workspace*
- und **NICHT** mit *File* > *Import...* > *Zip file* !!!

Verifizieren Sie, dass das File geändert, kompiliert und ausgeführt werden kann.

Übung 11: *Fibonacci-Folge*

Die Fibonacci-Folge ist definiert als:

Jedes Element der Folge ist die Summe der beiden vorangegangenen Elemente. Wobei die ersten beiden Elemente den Wert 1 haben.

Somit: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Erstellen Sie ein Programm, dass vom Benutzer die Anzahl der zu berechnenden Elemente der Fibonacci-Folge erfragt und diese dann entsprechend berechnet und auf die Konsole ausgibt.

```
Anzahl der zu berechnenden Fibonacci-Zahlen: 6
Fibonacci-Zahlen: 1, 1, 2, 3, 5, 8
```

Übung 12: *Cast*

Da Java eine *typensichere* Sprache ist, sind Zuweisungen von ungleichen Datentyp nicht erlaubt.

Dennoch kann z.B. eine Zuweisung eines *double* in einen *int* erzwungen werden unter Zuhilfenahme des sog. *Cast-Operators*: (*type*)

Dabei wird mit *type* der Ziel-Datentyp spezifiziert.

Beispiel:

```
double franken = 1.25;
double rappenDouble = franken * 100;
int rappenInt = (int) rappenDouble;
```

Erstellen Sie ein Programm, dass vom Benutzer einen Franken-Betrag erfragt und dann diesen in Rappen umrechnet und auf die Konsole ausgibt.

Beispiel:

```
Geben Sie einen Franken-Betrag an : 1.25
Umgerechnet in Rappen                = 125
```

Übung 13: Überlauf

Definieren Sie eine Variable vom Type `byte` und initialisieren Sie diese mit 0. Inkrementieren Sie daraufhin 300 mal diese `byte`-Variable um 1 und geben Sie das Resultat jeweils auf die Konsole aus.

Was stellen Sie fest ?

Hinweis:

Eine Zahl kann mit dem sog. *Inkrement-Operator* `++` um den Wert 1 inkrementiert werden.

Beispiel: `meineZahl++;`

Übung 14: Fakultät

Erstellen Sie ein Programm, dass die Fakultät berechnet.

Die Fakultät für eine Zahl n ist definiert als: $1 * 2 * 3 * .. n$

Die Fakultät von 0 ist 1 und für negative Zahlen nicht bestimmt.

Beispiel: Fakultät von 4 ist $1 * 2 * 3 * 4 \Rightarrow 24$

Schreiben Sie ein entsprechendes Programm, welches die Zahl n von der Konsole einliest, die Fakultät berechnet und das Resultat auf die Konsole ausgibt.

Übung 15: Quersumme

Erstellen Sie ein Programm, dass die Quersumme berechnet.

Die Quersumme ist definiert als die Summe der einzelnen Ziffern einer Dezimalzahl.

Beispiel: $1234 = 1 + 2 + 3 + 4 \Rightarrow 10$

Schreiben Sie ein entsprechendes Programm.

Hinweise:

Verwenden Sie dazu die *Ganzzahl-Division* (`/`) und den *Modulo-Operator* (`%`).

Übung 16: *Zahlen raten*

Ein Spiel-Programm: das Programm ermittelt eine Zufallszahl zwischen 0 und 100 und der Spieler muss diese Zahl herausfinden.

Ablauf:

Das Programm bestimmt eine Zufallszahl mit:

```
int zufallsZahl = (int) (101 * Math.random());
```

Daraufhin wird der Benutzer nach einer Zahl abgefragt.

Diese Zahl wird mit der Zufallszahl verglichen und wenn sie falsch ist, wird dem Benutzer mitgeteilt, ob die eingegebene Zahl zu klein oder zu gross ist.

Dies wird wiederholt bis die richtige Zahl gefunden wurde.

Am Ende gibt das Programm bekannt, wieviele Versuche nötig waren.

Überlegen Sie sich im weiteren, wieviele Versuche maximal nötig sind um die richtige Zahl zu finden.

Übung 17: *Primzahlen*

Erstellen Sie ein Programm, dass alle Primzahlen in einem definierbaren Bereich sucht.

Beispiel:

```
Geben Sie den Bereich ein.
```

```
Untere Grenze: 2
```

```
Obere Grenze: 30
```

```
Primzahlen: 2 3 5 7 11 13 17 19 23 29
```

Hinweis:

Primzahlen sind natürliche Zahlen (ganze Zahlen > 0), die nur durch sich selbst und 1 ohne Rest teilbar sind (wobei 1 keine Primzahl ist).

Übung 18: Arrays

Ein *int*-Array soll mit den Zahlen 1, 2, 3, 4, 5 und 6 initialisiert werden.
Geben Sie zur Verifikation den Inhalt des Array auf die Konsole aus.

Dann sollen von der Konsole sechs *int*-Werte eingelesen werden und der Reihe nach im Array gespeichert werden.
Geben Sie zur Verifikation wieder den Inhalt des Array auf die Konsole aus.

Vertauschen Sie sodann alle Werte innerhalb des Arrays (das erste wird zum letzten, das zweite zum zweitletzten, etc.).
Geben Sie zur Verifikation wieder den Inhalt des Array auf die Konsole aus.

Beispiel:

```
Ausgabe des Arrays:  
1 2 3 4 5 6  
Einlesen des Arrays:  
Zahl fuer Index [0] : 11  
Zahl fuer Index [1] : 22  
Zahl fuer Index [2] : 33  
Zahl fuer Index [3] : 44  
Zahl fuer Index [4] : 55  
Zahl fuer Index [5] : 66  
Ausgabe des Arrays:  
11 22 33 44 55 66  
Ausgabe des gedrehten Arrays:  
66 55 44 33 22 11
```

Übung 19: *String-Längen-Sortierung*

Schreiben Sie ein Programm, welches von der Konsole sequentiell maximal fünf Strings einliest oder bis der String "ENDE" eingegeben wird.

Die Strings sollen in einem String-Array gespeichert werden.

Dann sollen die Strings innerhalb des Arrays aufgrund ihrer **Länge** sortiert werden. Kürzester String zuerst.

Am Schluss soll der Inhalt des sortierten Arrays auf die Konsole ausgegeben werden.

Beispiele:

```
Neuer String: Hans
Neuer String: Katharina
Neuer String: Joe
Neuer String: Claudia
Neuer String: James
```

```
Ausgabe:
Joe
Hans
James
Claudia
Katharina
```

Oder:

```
Neuer String: Claudia
Neuer String: Hans
Neuer String: Joe
Neuer String: ENDE
```

```
Ausgabe:
Joe
Hans
Claudia
```

Übung 20: Library-API-Dokumentation

In Java ist ein Mechanismus für die Programm-Dokumentation eingebaut ⇒ **Javadoc**
Dabei wird aus speziellen Programm-Kommentaren eine HTML-Dokumentation generiert.

Die komplette *Laufzeit-Bibliothek (Runtime-Library)*, die mit der *Java 2 Platform Standard Edition (J2SE)* mitkommt, ist so dokumentiert.

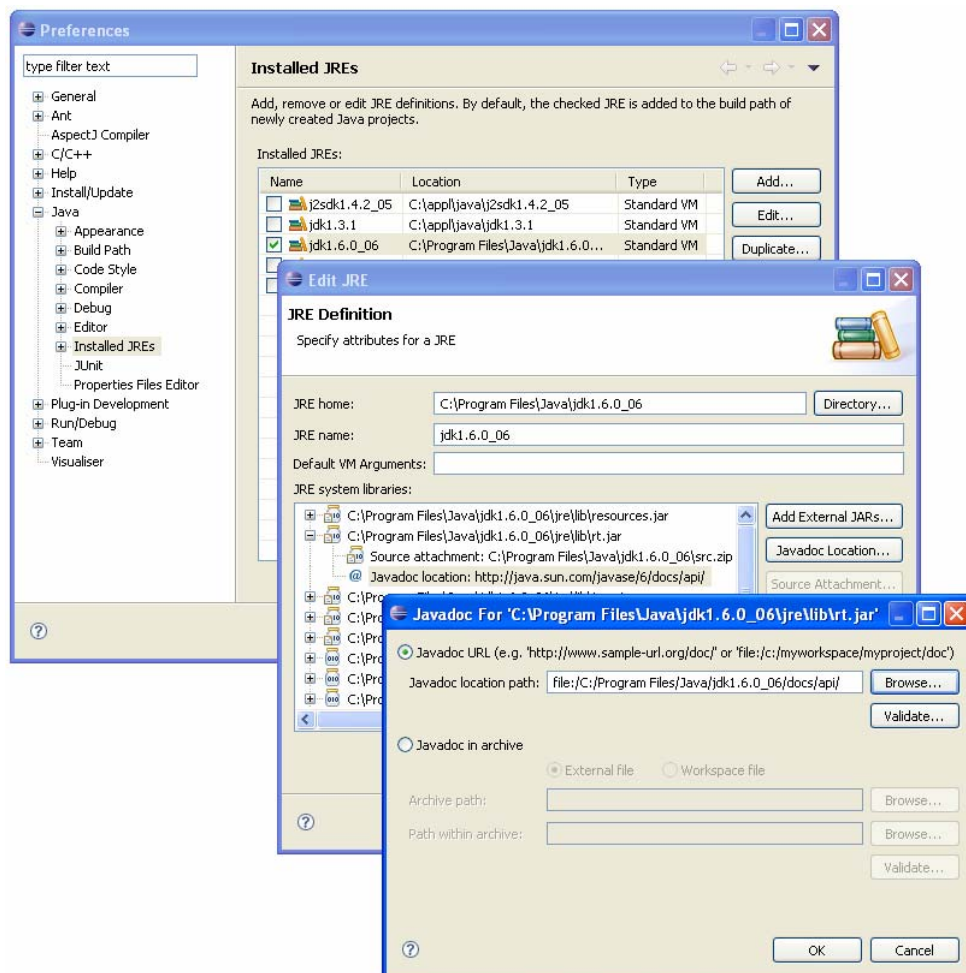
Dazu muss diese jeweils speziell heruntergeladen werden, z.B. für `jdk-6-doc.zip` für `jdk1.6.0_06` und im Installations-Directory des J2SE ausgepackt werden ⇒ Sub-Directory: `jdk1.6.0_06/docs`.

Wenn diese Dokumentation vorhanden ist, kann diese in Eclipse eingebunden werden, sodass aus dem Editor ein direkter Zugriff darauf möglich wird.

Hinweis: für JDK 1.6 ist das entsprechende ZIP-File (`jdk-6-doc.zip`) auf der CD.

Einbinden der *Library-API-Dokumentation* in *Eclipse (API: Application Programming Interface)*:

- Menü : *Window > Preferences > Java > Installed JREs*
- Die aktuelle *JRE* anwählen, dann *EDIT...*
- Die Bibliothek *rt.jar* expandieren.
 - Den Eintrag "*Javadoc location:*" editieren (Doppel-Klick):
 - "*Javadoc location path:*" auf das entsprechende Directory einstellen, z.B.: `file:/C:/Program Files/Java/jdk1.6.0_06/docs/api/`



Daraufhin hat man von allen Elementen, welche in der Laufzeit-Bibliothek mit einem *Javadoc-Kommentar* versehen sind, direkten Zugriff darauf.

Versuchen Sie z.B. bei der Übung 3 “Standard-Eingabe” direkt nach dem `main()` den Cursor auf den String “Scanner” zu positionieren. Es werden als Tool-Tip die ersten Zeilen des *Javadoc-Kommentars* von `Scanner` dargestellt.

```
public class StdIn {
    public static void main(String[] args) {
        // Verbindung zur Standard-Eingabe resp.zum Keyboard:
        Scanner eingabe = new Scanner (System.in);

        System.out.print ("Ein
// Einlesen eines Stri
String str = eingabe.n
System.out.println("St

System.out.print ("Ganz
// Einlesen einer Zahl von der Standard-Eingabe als Ganzzahl vom Typ int :
int i = eingabe.nextInt();
System.out.println("Zahl = " + i);
```

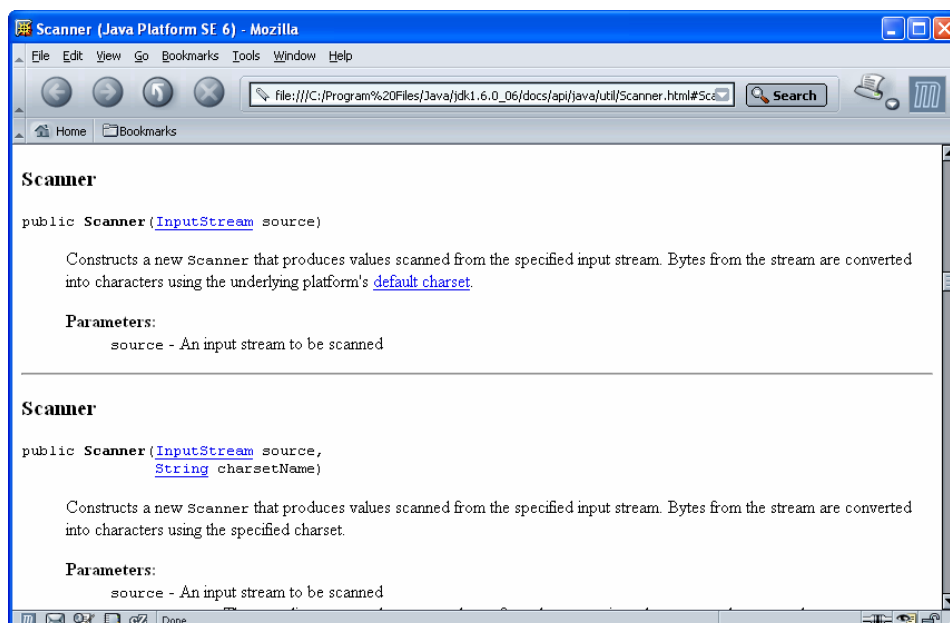
java.util.Scanner.Scanner(InputStream source)
Constructs a new Scanner that produces values scanned from the specified input stream. Bytes from the stream are converted into characters using the underlying platform's [default charset](#).

Parameters:
source An input stream to be scanned

Press 'F2' for focus

Wenn man die Funktionstaste *F2* drückt, wird der ganze Kommentar als Tool-Tip mit *Scrollbars* dargestellt.

Setzt man den Cursor auf das Wort “Scanner” und drückt man *SHIFT-F2* wird ein *HTML-Browser* mit obigem Kommentar gestartet (kann u.U. einen Moment dauern).



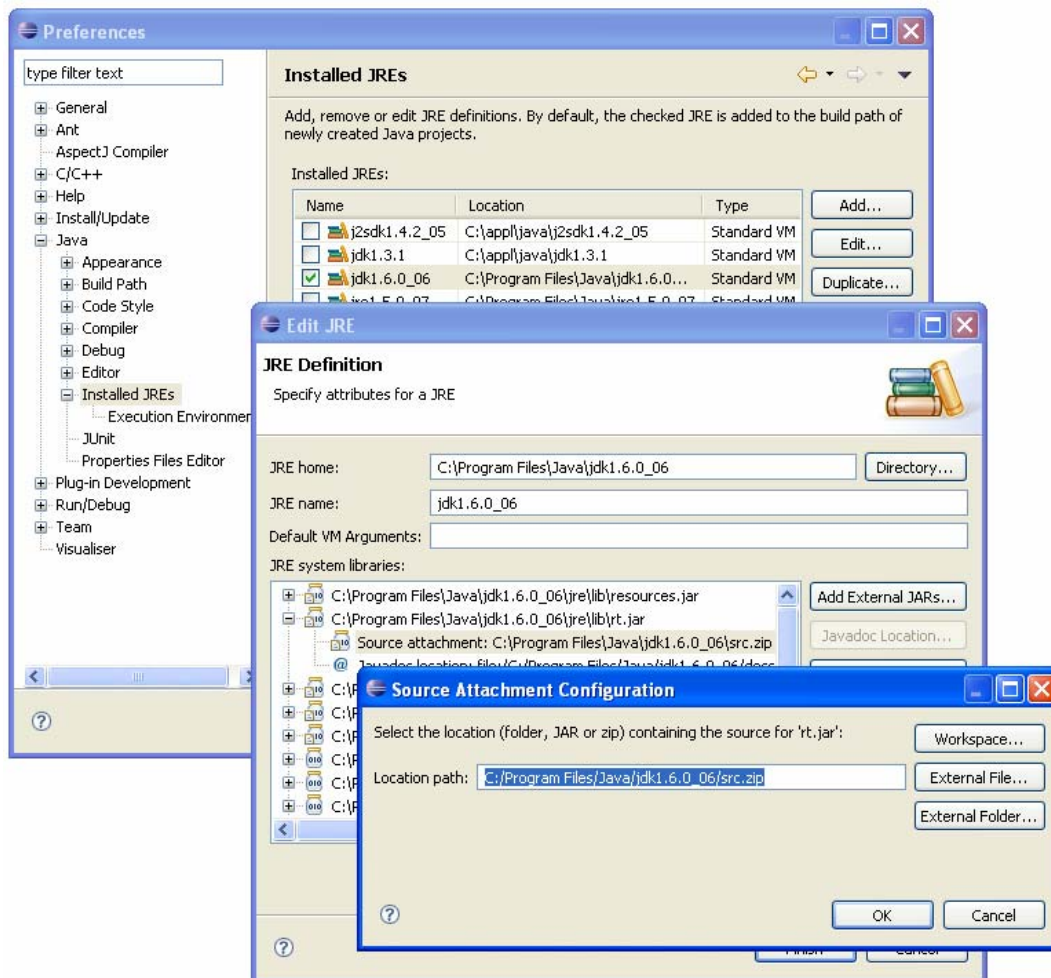
Der konkret benutzte *HTML-Browser* wird unter Menü: *Window > Preferences > Help* festgelegt.

Übung 21: Anbindung der *Library*-Sources

Die Quellen der gesamten *Laufzeit-Bibliothek* sind ebenfalls verfügbar. Diese sind je nach J2SE-Version separat zu beschaffen und einzubinden. Das File heisst jeweils *src.zip* und ist normalerweise im Root-Directory der JDK-Installation.

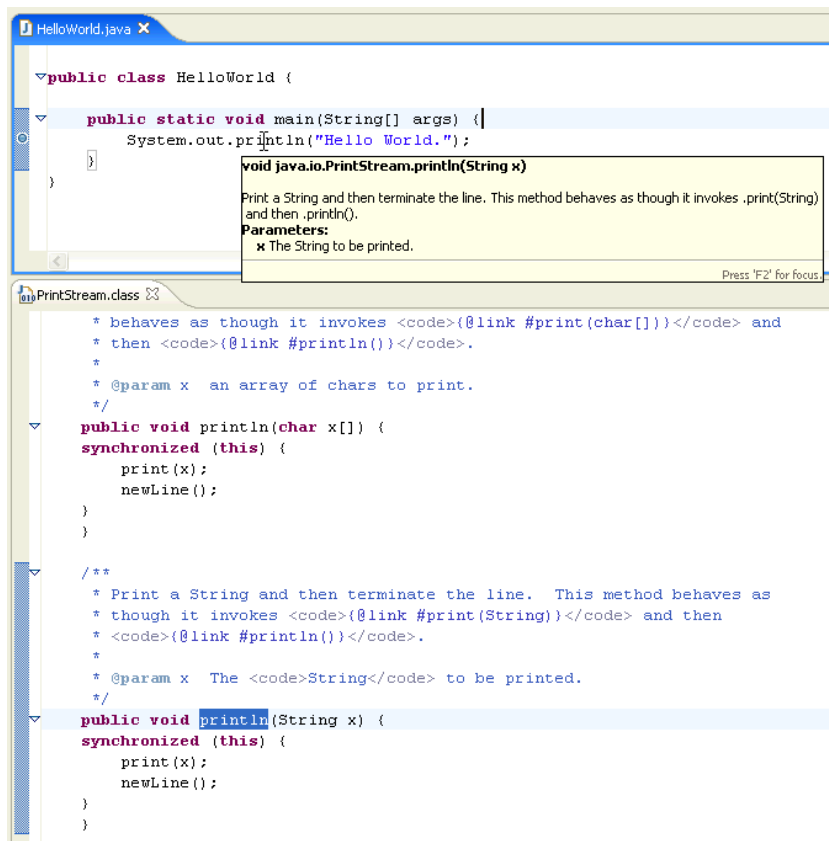
Die eigentliche *Laufzeit-Bibliothek (Runtime-Library)* ist im File *rt.jar* enthalten. Diesem File können nun die Quellen *attach'ed* werden.

Die Einbindung erfolgt analog zur vorherigen Einbindung der Javadoc:



Die *Library-Sourcen* können geöffnet werden, indem ein API-Element selektiert wird, und dann:

- *Context-Menü (rechte Maustaste) > Open Declaration*
oder
- **F3**
oder
- **CTRL+linke Maustaste**



Die *Library-Sourcen* sind auch mit dem *Debugger* mittels **Step Into** (F5) sichtbar !!
Verifizieren Sie das bei der Funktion `println()` bei der Übung 2 *Hello World*.

Übung 22: *String*

Schreiben Sie ein Programm, das ein Satz auf einer Zeile von der Konsole einliest und dann den Satz wieder auf die Konsole ausgibt, aber dabei die Reihenfolge der Wörter vertauscht.

Beispiel:

- Eingabe: "heute regnet es"
- Ausgabe: "es regnet heute"

Die Klasse `Scanner` verfügt über `nextLine()` um eine ganze Zeile incl. sog. *Whitespaces* (*Leerschläge, Tabulatoren, etc*) einzulesen (analog zu `nextInt()` für `Integer`).

Verwenden Sie die *Online-API-Dokumentation* der Klasse '*String*' um geeignete *Methoden (Funktionen)* für die Lösung dieses Problems zu finden.

Übung 23: *Vokabel-Trainer*

Schreiben Sie ein Programm, das Sie beim Lernen von Vokabeln unterstützt.

Hinweise:

- Benutzen Sie für die Speicherung der Informationen einen zweidimensionalen Array von Strings.
- die Klasse `Scanner` besitzt auch einen Konstruktor `Scanner(String source)`. Bei diesem kann anstelle von `System.in` auch ein normaler `String` übergeben werden, um daraus z.B. mit `nextInt()` einen entsprechenden Integer zu erhalten.
- Um aus einem `String` `meineNummer` einen entsprechenden `int` `i` zu erhalten:
`int i = Integer.parseInt(meineNummer);`
- Um aus einem `int` `i` wieder einen entsprechenden `String` zu erhalten:
`String meineNummer = Integer.toString(i);`

Beispiel auf der Konsole (fett gedrucktes sind User-Eingaben):

Geben Sie max. 10 zu übende Vokabeln ein.

Format: Frage-String:Antwort-String:nötige Anzahl richtiger Antworten
Beispiel: Baum:Tree:3
Ende: leerer String

Beginn der Eingabe:

1. Vokabel: **Baum:Tree:2**
2. Vokabel: **See:Lake:1**
3. Vokabel:
O.K.: Ende der Eingabe.

Beginn des Trainings:

Baum: **Tree**
o.k :-) ; noch 1

See: **Lago**
NOK :-(; noch 1

Baum: **Tree**
o.k :-) ; Fertig

See: **Lake**
o.k :-) ; Fertig

Alle richtig, Gratulation!

Übung 24: Methoden I

In der Übung 18 "Arrays" musste der Inhalt des Arrays mehrmals auf die Konsole ausgegeben werden.

Erstellen Sie in jenem Programm eine *Methode* `printArray(int[] pArray)`, welche diese Aufgabe übernimmt und benutzen Sie diese dann für die Ausgaben der Arrays auf die Konsole.

Übung 25: Methoden II

Erstellen Sie eine *Methode*, welche als *Parameter* drei Integer aufweist und die grösste der drei Zahlen bestimmt und sodann als return-Wert zurückgibt. Testen Sie die erstellte Methode.

Übung 26: Zweidimensionales Array

Erstellen Sie eine *Methode*, welche als *Parameter* die Grösse für eine Matrix erhält. Die Methode erzeugt sodann aufgrund des Parameters ein zweidimensionales `int`-Array und initialisiert die Elemente aufgrund der Position in der Matrix (siehe nachfolgendes Beispiel).

Dann wird die Matrix auf die Konsole ausgegeben.

Daraufhin wird die Matrix mit einer zweiten Methode transponiert (d.h. die Elemente werden um die Diagonale a_{nn} gespiegelt).

Daraufhin wird die transponierte Matrix wieder ausgegeben.

Hinweis: setzen Sie auch wieder die `printArray()`-Methode von Übung 24 ein (einfach nur mit copy&past im Editor).

Session-Log:

Matrix:

```
11 12 13 14
21 22 23 24
31 32 33 34
41 42 43 44
```

Transponierte Matrix:

```
11 21 31 41
12 22 32 42
13 23 33 43
14 24 34 44
```

Übung 27: *Container*

Erstellen Sie ein Programm, das den Benutzer nach einer Zahl fragt.

Dann werden gemäss dieser Zahl **int-Arrays** wie folgt erzeugt.

Inhalt der Arrays: Nummer des Arrays nach dem Schema:

x xx xxx xxxx ... wobei x = Nummer des Arrays [1..9]

Diese werden der Reihe nach in einen **vector** eingefügt.

Am Ende werden die Arrays aus dem Vector ausgelesen und auf die Konsole ausgegeben. Verwenden Sie dazu die Methode `printArray()` aus der vorherigen Übung.

Wenn eine Zahl eingegeben wird, die nicht zwischen 1 und 9 liegt, soll das Programm terminieren.

Beispiel:

```
Zahl: 7
1
2 22
3 33 333
4 44 444 4444
5 55 555 5555 55555
6 66 666 6666 66666 666666
7 77 777 7777 77777 777777 7777777
Zahl: -1
ENDE.
```


Weitere Anforderungen:

Die Dimensionen sollen als Konstanten definiert werden können.

Beispiel:

```
public class NavigationSystem {  
  
    final static int COLS = 30; // Number of Columns  
    final static int ROWS = 10; // Number of Rows  
  
    // ev. weitere Konstanten ...  
}
```

Konstanten welche auf diese Art deklariert werden sind in allen Methoden sichtbar.

Ein paar Tips:

Es macht Sinn, den ganzen Inhalt des Bildschirms in einem char-Array aufzubereiten und zu bewirtschaften (`char[][] carr;`) und dann mit einer Schleife auf den Monitor auszugeben.

Um Arrays zu kopieren: `System.arraycopy()`

Um von einem *String* zum entsprechenden *char*-Array zu kommen:

```
String.toCharArray()
```

Um von zwei Zahlen die grössere oder kleinere zu erhalten: `Math.max()` resp. `Math.min()`

Phytagoras:

```
Math.sqrt(Math.pow(y-newYPos, 2) + Math.pow(x-newXPos, 2));
```

Um von einem *int* zum entsprechenden *String* zu kommen:

```
Integer.toString(i)
```

Um eine bestimmte Anzahl von Millisekunden (z.B. 300 Millisekunden) zu warten:

```
try { Thread.sleep(300); } catch (InterruptedException e) {}
```

Globale Konstanten werden direkt nach der Klassen-Deklaration und vor der ersten Funktion definiert (z.B. `final static int COLS = 30;`)

“Hupen”;-): `java.awt.Toolkit.getDefaultToolkit().beep();`

Der Routing-Algorithmus muss nicht genau gleich funktionieren wie bei der alten Applikation (der Weg kann unterschiedlich sein. Das Ziel muss einfach erreicht werden).

Viel Spass ;-)