

Übungen

Java Einführung

Copyright © 2011–2020, thomas.iten@iten-engineering.ch

Alle Rechte vorbehalten.

Reproduktion (auch auszugsweise) ist nur mit schriftlicher Bewilligung des Verfassers gestattet.

Inhaltsverzeichnis

1. About.....	3
2. Write Once, Run Anywhere	3
2.1. Infrastruktur.....	3
3. Hello World.....	4
3.1. HelloJava mit Eclipse	4
3.2. HelloJava selber kompilieren und ausführen	4
3.3. ReverseArgs	4
4. Grundlegende Sprachelemente.....	5
4.1. Syntax Check.....	5
4.2. Rechteck.....	5
4.3. Reduzierte Notation	6
5. Kontrollstrukturen	7
5.1. Note	7
5.2. Modulo	8
5.3. Quadrat.....	8
6. Klassen, Attribute und Methoden.....	9
6.1. Kreis	9
6.2. MathUtil.....	10
7. Kapselung und Konstruktoren	11
7.1. Kreis 2	11
7.2. Person	12
8. Vererbung	13
8.1. Zylinder.....	13
8.2. Fahrzeug.....	14
9. Packages	15
9.1. JDK Dokumentation	15
10. Interfaces und Adapterklassen	16
10.1. Media.....	16
11. Strings und Wrapper Klassen.....	17
11.1. Compare	17
11.2. Builder	17
11.3. SplitDate	18
11.4. WordCount.....	18
12. Arrays, Varargs und Enum	19
12.1. ArrayValues	19
12.2. ArraySearch.....	19
12.3. ArrayCalculation.....	20
12.4. Message	20
13. Collections.....	21
13.1. Bauernhof	21
13.2. Book	22
14. Exceptions.....	23
14.1. NumberException	23
15. Assertions	23
15.1. AssertionDemo	23
16. Dateien.....	24
16.1. SquareNumbers.....	24
17. Streams.....	25
17.1. PersonDemo	25
17.2. More Streams	26
18. Misc.....	27
18.1. Lotto.....	27
18.2. Date & Time API (java.time).....	27
19. Funktionale Programmierung	28

19.1. Demo's.....	28
19.2. Warenhaus	28

1. About

Der Kurs vermittelt Ihnen eine umfangreiche Einführung in die Java Programmiersprache und eine Übersicht über die vielen Einsatzmöglichkeiten. Sie verstehen Javas Systemarchitektur und kennen die fundamentalen Klassen und Sprachelemente. Nach dem Kurs sind Sie in der Lage, selbständig einfache Java-Programme zu schreiben

2. Write Once, Run Anywhere

2.1. Infrastruktur

Kurze Einführung in den Aufbau der Kursunterlagen und verwendeten Programme.
Installation JDK und Eclipse.

3. Hello World

3.1. HelloJava mit Eclipse

- a) Erstellen Sie im Sub-Package **exercise** eine Klasse mit dem Namen **HelloJava** und einer main Methode.
- b) Programmieren Sie eine einfache Ausgabe auf die Konsole.
- c) Führen Sie das Programm mit Eclipse aus und prüfen Sie Ihre Ausgabe.

3.2. HelloJava selber kompilieren und ausführen

- a) Starten Sie im Sub-Package **exercise** mit der Batch Datei **start_cmd.bat** eine Windows Comand Shell.
- b) Wechseln Sie ins Verzeichnis: ...\\src\\chapter03\\helloworld\\exercise
- c) Kompilieren Sie die Datei mit dem Befehl: `javac HelloJava.java`
→ Es wird die Datei `HelloJava.class` erstellt
- d) Wechseln Sie ins Verzeichnis: ...\\src
- e) Starten Sie die Anwendung mit dem Befehl:
`java chapter03.helloworld.exercise>HelloJava`

3.3. ReverseArgs

- a) Erstellen Sie im Sub-Package `exercise` eine Klasse **ReverseArgs** mit einer main Methode.
- b) Geben Sie alle Programm Argumente auf der Konsole aus.
- c) Kompilieren sie Ihr Programm auf der Kommandozeile mit `javac`
- d) Führen Sie Ihr Programm aus und testen Sie es mit einer unterschiedlichen Anzahl Argumente.
- e) Zusatz: Stellen Sie Ihr Programm um, so dass die Argumente in umgekehrter Reihenfolge ausgegeben werden.

4. Grundlegende Sprachelemente

4.1. Syntax Check

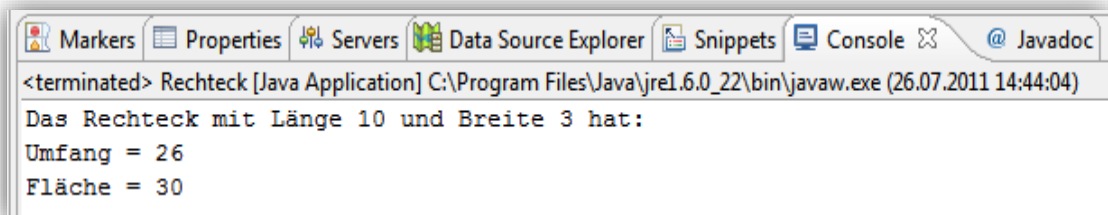
- a) Prüfen Sie den folgenden Programmausschnitt und korrigieren Sie die Fehler:

```
class SyntaxCheck {  
    public static void main(String[] args) {  
        int a b;  
  
        a = b = 10;  
  
        System.out.println("Beide Zahlen haben jetzt den Wert 10");  
    }  
}
```

4.2. Rechteck

- a) Erstellen Sie die Klasse **Rechteck** mit einer main Methode.
- b) Definieren Sie zwei Variablen für die Länge und Breite und initialisieren Sie diese mit den Werten 10 und 3.
- c) Berechnen Sie den Umfang und die Fläche und geben Sie anschliessend die Resultate auf der Konsole aus.

Beispiel Ausgabe:



- d) Zusatz: Übergeben Sie die Länge und Breite dem Programm als Argumente.

4.3. Reduzierte Notation

- a) Notieren Sie sich für jede Anweisungszeile die Werte von x und y.

Code	x	y
<code>int x = 1, y = 2;</code>		
<code>x++;</code>		
<code>x += y;</code>		
<code>x *= y;</code>		
<code>x -= y++;</code>		
<code>x /= 8 - 2 * y--;</code>		
<code>x = y = x + y</code>		

- b) Schreiben Sie ein Programm, dass die folgenden Anweisungen enthält und geben Sie nach jeder Zeile die Werte von x und y auf die Konsole aus.
- c) Überprüfen Sie Ihre Angaben

5. Kontrollstrukturen

5.1. Note

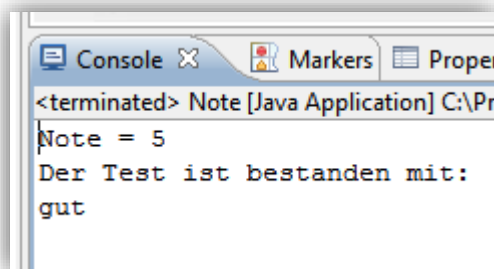
- a) Erstellen Sie eine Klasse Note mit einer main Methode, die als Argument einen Ganzzahl Wert zwischen 1..6 einliest und den eingelesenen Wert ausgibt.
- b) Prüfen Sie die eingelesene Note und falls diese grösser oder gleich 4 ist, geben Sie folgenden Text aus: „ Der Test ist bestanden“

Falls die Note kleiner 4 ist geben Sie den Text „Der Test ist nicht bestanden“ aus.

- c) Starten Sie das Programm mit verschiedenen Werten und prüfen Sie die Ausgabe.
- d) Ergänzen Sie die bisherige Ausgabe nun noch mit einer detaillierten Angabe der Bewertung folgendermassen:

Note 6: sehr gut
Note 5: gut
Note 4: genügend
allen anderen Fälle: ungenügend

Beispiel Ausgabe:



- e) Zusatz: Damit Ihr Programm auch bei falschen Eingaben nicht abstürzt, soll am Anfang geprüft werden, ob ein Parameter eingegeben wurde und ob dieser einen korrekten Wertebereich aufweist.

Falls kein Parameter übergeben wurde, machen Sie folgende Ausgabe und beenden Sie das Programm mit exit: "Falscher Aufruf: Bitte übergeben Sie ein Argument!"

Falls der eingegebene Werte nicht zwischen 1..6 liegt, machen Sie folgende Ausgabe und beenden Sie das Programm: "Falscher Wert: Die Note muss einen Wert zwischen 1..6 haben!"

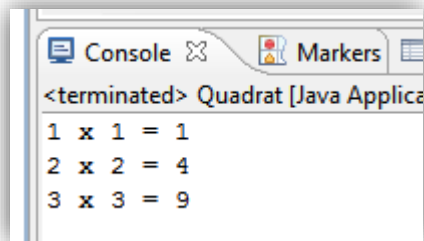
5.2. Modulo

- a) Erstellen Sie ein Programm Modulo dass nacheinander alle ungeraden Zahlen zwischen 1 und 30 ausgibt. Verwenden Sie dazu den Modulo Operator (%).

5.3. Quadrat

- a) Erstellen Sie ein Programm Quadrat, das in einer Schleife alle Quadratzahlen ausgibt.

Beispiel:



- b) Die Schleife soll solange durchlaufen werden, wie die berechnete Quadratzahl kleiner oder gleich 15 ist.

6. Klassen, Attribute und Methoden

6.1. Kreis

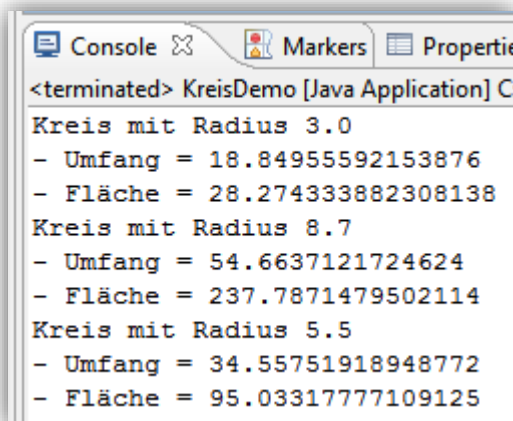
- a) Erstellen Sie eine Klasse Kreis mit folgenden Attributen:
 - radius (von Typ double)
- b) Ergänzen Sie die Klasse mit folgenden Methoden:
 - getUmfang()
Berechnung und Rückgabe des Kreisumfangs ($2 * \text{PI} * \text{Radius}$)
 - getFlaeche()
Berechnung und Rückgabe der Kreisfläche ($\text{PI} * \text{Radius}^2$)
 - print()
Ausgabe von Kreisradius, Umfang und Fläche auf die Konsole wie folgt:

```
Kreis mit Radius 3.0
- Umfang = 18.84955592153876
- Fläche = 28.274333882308138
```

Hinweis:

Der Wert von PI ist in der statischen Konstante der Klasse Math definiert und kann im Code mit Math.PI verwendet werden.

- c) Erstellen Sie eine Klasse KreisDemo mit einer main Methode.
- d) Erzeugen Sie dort drei Kreis Instanzen mit den Radien 3.0, 8.7 und 5.5 und geben Sie jeweils den Radius, Umfang und die Fläche auf der Konsole aus.

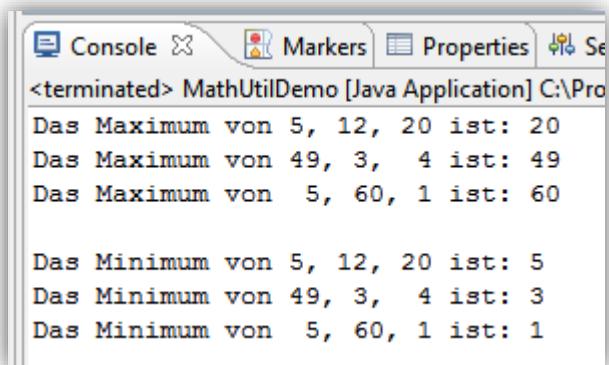


```
<terminated> KreisDemo [Java Application] C
Kreis mit Radius 3.0
- Umfang = 18.84955592153876
- Fläche = 28.274333882308138
Kreis mit Radius 8.7
- Umfang = 54.6637121724624
- Fläche = 237.7871479502114
Kreis mit Radius 5.5
- Umfang = 34.55751918948772
- Fläche = 95.03317777109125
```

6.2. MathUtil

- a) Erstellen Sie eine Klasse MathUtil mit folgenden statischen Methoden:
- min (int a, int b, int c)
Berechnung und Rückgabe des Minimums der drei Zahlen.
 - max (int a, int b, int c)
Berechnung und Rückgabe des Maximum der drei Zahlen.
- b) Erstellen Sie eine Klasse MathUtilDemo mit einer main Methode und testen Sie die beiden Hilfsmethoden mit verschiedenen Zahlenwerten.

Geben Sie die Resultate wie folgt auf der Konsole aus:



```
<terminated> MathUtilDemo [Java Application] C:\Pro  
Das Maximum von 5, 12, 20 ist: 20  
Das Maximum von 49, 3, 4 ist: 49  
Das Maximum von 5, 60, 1 ist: 60  
  
Das Minimum von 5, 12, 20 ist: 5  
Das Minimum von 49, 3, 4 ist: 3  
Das Minimum von 5, 60, 1 ist: 1
```

7. Kapselung und Konstruktoren

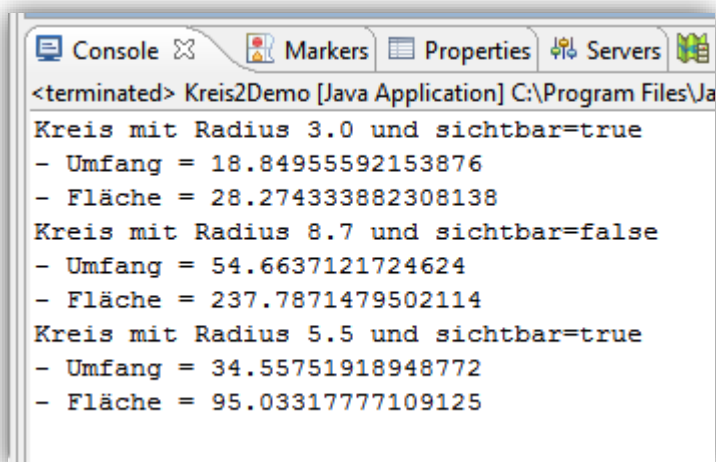
7.1. Kreis 2

- a) Kopieren Sie die Klasse Kreis aus dem vorherigen Kapitel und ändern Sie den Namen auf Kreis2.
- b) Ergänzen Sie die Klasse, so dass Sie folgende Attribute aufweist
 - radius (von Typ double)
 - sichtbar (von Typ boolean)
- c) Deklarieren sie diese als private und fügen Sie entsprechenden Getter und Setter hinzu.
- d) Erstellen Sie für die Klasse die folgenden Konstruktoren
 - Konstruktor ohne Parameter
 - Konstruktor bei dem der Radius angegeben werden kann
 - Konstruktor bei dem der Radius und die Sichtbarkeit angegeben werden kann
- e) Ergänzen Sie die Methode print() mit der Angabe der Sichtbarkeit, wie folgt:

```
Kreis mit Radius 3.0 und sichtbar=true  
- Umfang = 18.84955592153876  
- Fläche = 28.274333882308138
```

- f) Kopieren Sie die Klasse KreisDemo aus dem vorherigen Kapitel und ändern Sie den Namen auf Kreis2Demo.
- g) Passen Sie den Code an und testen Sie Ihre Anwendung.

Beispiel:

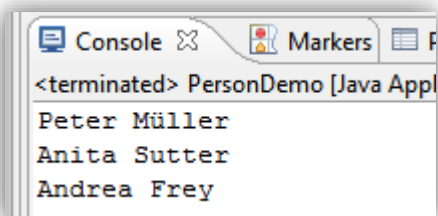


```
<terminated> Kreis2Demo [Java Application] C:\Program Files\Ja  
Kreis mit Radius 3.0 und sichtbar=true  
- Umfang = 18.84955592153876  
- Fläche = 28.274333882308138  
Kreis mit Radius 8.7 und sichtbar=false  
- Umfang = 54.6637121724624  
- Fläche = 237.7871479502114  
Kreis mit Radius 5.5 und sichtbar=true  
- Umfang = 34.55751918948772  
- Fläche = 95.03317777109125
```

7.2. Person

- a) Erstellen Sie eine Klasse Person mit zwei Attributen für Vor- und Nachname.
- b) Deklarieren Sie diese als private und erstellen Sie die entsprechenden Getter und Setter Methoden.
- c) Definieren Sie einen leeren Konstruktor sowie einen Konstruktor bei dem beide Attribute gesetzt werden können.
- d) Erstellen Sie eine Methode toString() die einen String mit dem zusammengesetzten Vor- und Nachnamen zurückgibt.
- e) Erstellen Sie ein Testprogramm mit drei Personen und geben Sie deren Namen auf der Konsole aus.

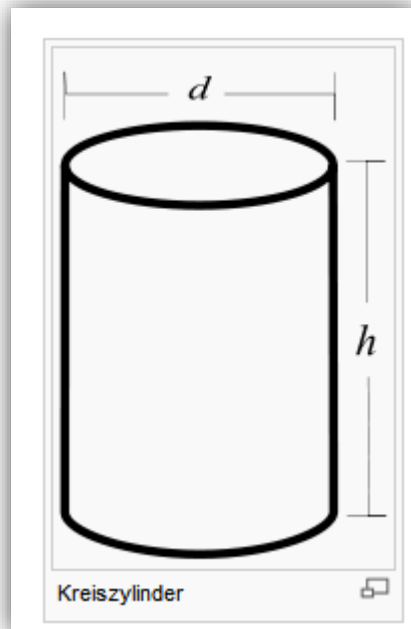
Beispiel:



8. Vererbung

8.1. Zylinder

- a) Erstellen sie eine Klasse Kreis mit folgenden Eigenschaften:
- Privates Attribut für den Radius mit Getter und Setter
 - Leere Konstruktor und Konstruktor mit Radius
 - Methode getUmfang für die Berechnung und Rückgabe des Umfang
 - Methode getFlaeche für die Berechnung und Rückgabe der Fläche
 - Methode toString() welche eine String mit den Angaben des Kreises zurückgibt
- b) Erstellen Sie eine Klasse Zylinder, die die Klasse Kreis erweitert und folgende zusätzlichen Eigenschaften aufweist:
- Privates Attribut für die Höhe mit Getter und Setter
 - Leerer Konstruktor und Konstruktor für den Radius und die Höhe
 - Methode getVolumen für die Berechnung des Volumen (Fläche * Höhe)
 - Methode toString() welche eine String mit den Angaben des Zylinders zurückgibt



- c) Erstellen Sie ein Testprogramm, das je eine Instanz eines Kreises und eines Zylinders erstellt und deren Angaben auf die Konsole ausgibt.

Beispiel:

```
Console X
<terminated> ZylinderDemo [Java Application] C:\Program Files\Java\jre1.6.0_22\bin\javaw.exe (27.07.2011 08:56:07)
Kreis mit Radius=12.5, Umfang=78.53981633974483, Fläche=490.8738521234052
Zylinder mit Radius=3.0, Höhe=5.5, Volumen=155.50883635269477
```

8.2. Fahrzeug

- a) Die folgenden Fahrzeuge sollen in einer Vererbungshierarchie abgebildet werden:

Fahrrad

- Attribute:
Farbe, Baujahr und Marke
- Operationen:
fahren(), toString() mit Angabe aller Attributwerte, sowie Getter/Setter für alle Attribute

PKW

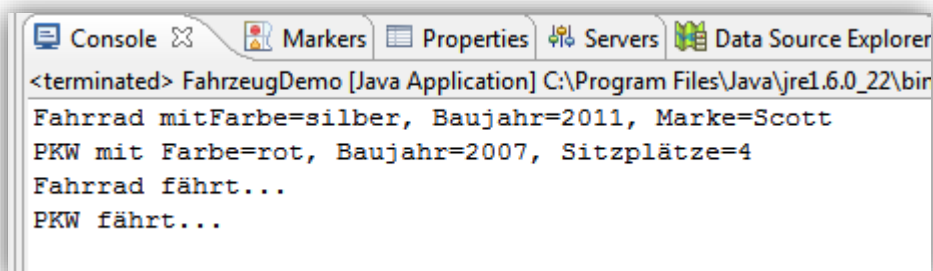
- Attribute:
Farbe, Baujahr und Sitzplätze
- Operationen:
fahren(), toString() mit Angabe aller Attributwerte, sowie Getter/Setter für alle Attribute

- b) Erstellen Sie ein Klassendiagramm mit einer Vererbungshierarchie, so dass die gemeinsamen Attribute und Operationen soweit möglich in gemeinsame abstrakte Oberklassen gruppiert werden.
- c) Erstellen Sie die Klassenhierarchie in Java. Die fahren Methode vom PKW und Fahrrad sollen dabei jeweils eine Meldung auf die Konsole ausgeben wie im folgenden Beispiel:

```
Fahrrad fährt...  
PKW fährt...
```

- d) Erstellen Sie ein Testprogramm und prüfen Sie Ihre Anwendung.

Beispiel:

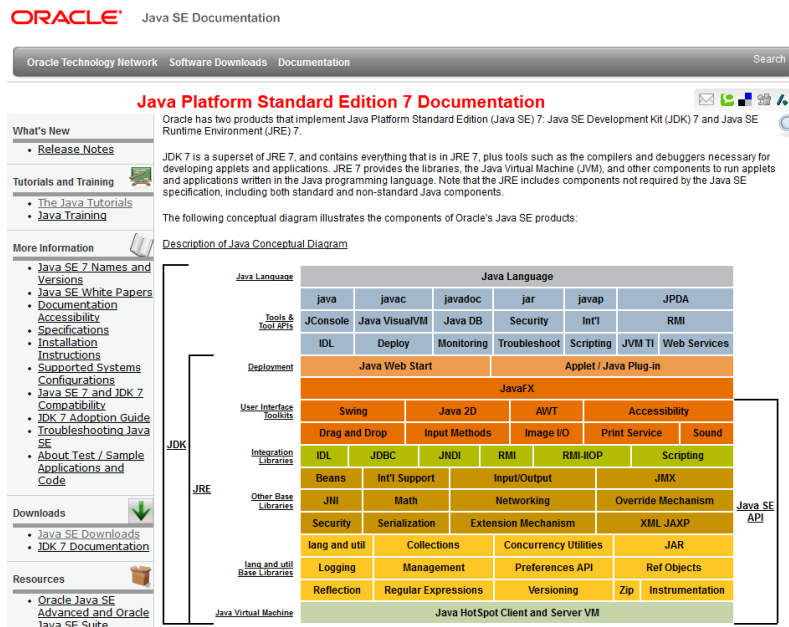


```
<terminated> FahrzeugDemo [Java Application] C:\Program Files\Java\jre1.6.0_22\bin  
Fahrrad mit Farbe=silber, Baujahr=2011, Marke=Scott  
PKW mit Farbe=rot, Baujahr=2007, Sitzplätze=4  
Fahrrad fährt...  
PKW fährt...
```

9. Packages

9.1. JDK Dokumentation

- a) Schauen Sie sich die Online Dokumentation der Java Standard Edition unter dem Link <http://docs.oracle.com/javase/7/docs/> an:



- b) Schauen Sie sich die JavaDoc API Spezifikation unter <http://docs.oracle.com/javase/7/docs/api/index.html> an:

Java™ Platform Standard Ed. 7

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.

- c) Durchstöbern Sie die einzelnen Packages oder schauen Sie sich zum Beispiel die mitgelieferten Methoden der Klasse Math aus dem Package java.lang an.

10. Interfaces und Adapterklassen

10.1. Media

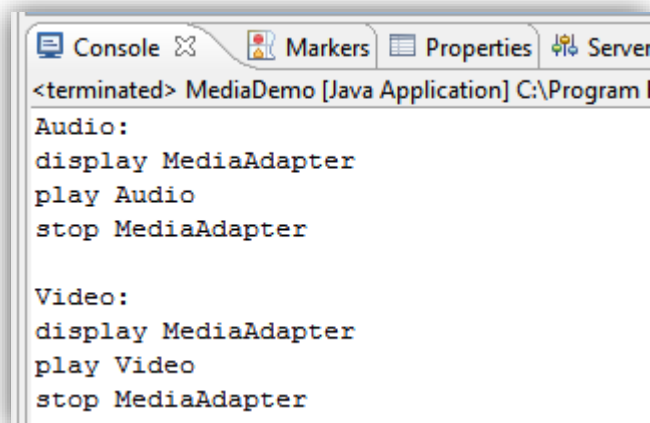
- a) Erstellen Sie ein Interface **Media** mit folgenden drei Methoden:
 - play()
 - stop()
 - display()
- b) Nun sollen zwei Klassen Audio und Video erstellt werden, welche das Interface implementieren. Dabei sollen die beiden Methoden stop() und display() durch eine gemeinsame Adapterklasse bereitgestellt werden.

Erstellen Sie also zunächst eine abstrakte Adapterklasse **MediaAdapter** die das Interface Media implementiert und für die beiden Methoden stop() und display() eine Implementation bereitstellt.

Die stop() Methode soll dabei den Text „stop MediaAdapter“ auf die Konsole ausgeben. Die Methode display den Text „display MediaAdapter“.

- c) Erstellen Sie die Klasse **Video**, welche den MediaAdapter erweitert und die Methode play() implementiert. Die play() Methode soll dabei den Text „play Video“ ausgeben.
- d) Erstellen Sie die Klasse **Audio**, welche den MediaAdapter erweitert und die Methode play() implementiert. Die play() Methode soll dabei den Text „play Audio“ ausgeben.
- e) Erstellen Sie eine Testapplikation, welche je eine Instanz von Video und Audio erstellt und die Methoden ausführt.

Beispiel Ausgabe:



```
<terminated> MediaDemo [Java Application] C:\Program I
Audio:
display MediaAdapter
play Audio
stop MediaAdapter

Video:
display MediaAdapter
play Video
stop MediaAdapter
```


11. Strings und Wrapper Klassen

11.1. Compare

- a) Erstellen Sie die Klasse Compare mit einer main Methode.
- b) Definieren Sie zwei Strings wie zum Beispiel „Abderhalden“ und „Müller“.
- c) Vergleichen Sie die beiden und geben Sie diese alphabetisch sortiert aus.

Verwenden Sie dazu die Methode `compareTo` der Klasse `String`. Schauen Sie die Funktionsweise von `compareTo` mit Hilfe der Javadoc nach.

- d) Zusatz: Ändern Sie die Ausgabe so ab, dass die Namen in Kleinbuchstaben ausgegeben werden. Suchen Sie dazu eine passende Methode der Klasse `String`.

11.2. Builder

- a) Erstellen Sie die Klasse `Builder` mit einer main Methode.
- b) Lesen Sie einen Namen als Parameter ein und stellen Sie mit Hilfe der Klasse `StringBuilder` den folgenden Text zusammen:

Sehr geehrte(r) Frau/Herr **<Name>**

Wir gratulieren Ihnen zur erfolgreichen Lösung der Aufgabe.

Anstelle von `<Name>` soll dabei der eingelesene Name stehen.

- c) Geben Sie den Text auf der Konsole aus.

11.3. SplitDate

- Definieren Sie einen String der ein Datum mit Tagen, Monat und Jahr beinhaltet wie zum Beispiel "17.04.1966" oder "01.03.2007".
- Zerlegen Sie nun die Eingabe in drei einzelne Strings für die Tage, den Monat und das Jahr. Geben Sie die einzelnen Strings auf der Konsole aus.

Tipp:

Schauen sie sich dazu die Methode `substring(...)` der String Klasse an.

- Erweitern Sie die Anwendung so, dass die einzelnen Werte auch korrekt extrahiert werden, wenn die Tage und / oder der Monat einstellig angegeben wird, wie zum Beispiel "17.4.1966" oder "1.3.2007".

Tipp:

Schauen sie sich zusätzlich die Methoden `indexOf(...)` der String Klasse an.

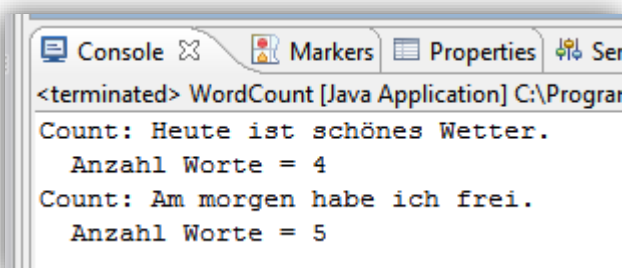
Beispiel:

```
Split: 17.04.1966
  day   = 17
 month  = 04
 year   = 1966
Split: 1.3.2007
  day   = 1
 month  = 3
 year   = 2007
```

11.4. WordCount

- Erstellen Sie eine Anwendung, die die Anzahl Worte eines Satzes zählt. Verwenden Sie dazu die Klasse `StringTokenizer` aus dem Package `java.util`.

Beispiel:



```
<terminated> WordCount [Java Application] C:\Program
Count: Heute ist schönes Wetter.
  Anzahl Worte = 4
Count: Am morgen habe ich frei.
  Anzahl Worte = 5
```

12. Arrays, Varargs und Enum

12.1. ArrayValues

- a) Erstellen Sie einen Array von Typ `int` mit der Länge 3 und initialisieren Sie diesen mit den Werten 1, 2 und 3 und geben Sie die einzelnen Werten in einer Schleife aus.

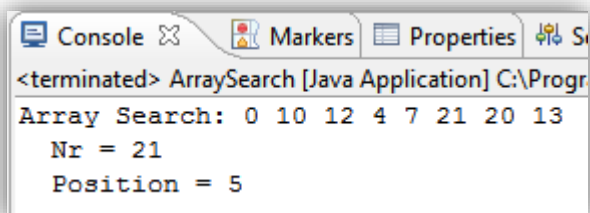
```
Array Values:  
Element 0 = 1  
Element 1 = 2  
Element 2 = 3
```

- b) Verdoppeln Sie in einer weiteren Schleife die einzelnen Werte und geben Sie diese wieder auf der Konsole aus:

```
Array Values mit verdoppelten Werten:  
Element 0 = 2  
Element 1 = 4  
Element 2 = 6
```

12.2. ArraySearch

- a) Definieren Sie einen Array mit folgenden Werten: 0, 10, 12, 4, 7, 21, 20, 13
- b) Definieren Sie ein Variable `nr` mit dem Wert 21.
- c) Suchen sie in einer Schleife die Position der `nr` und geben Sie diese auf der Konsole aus.

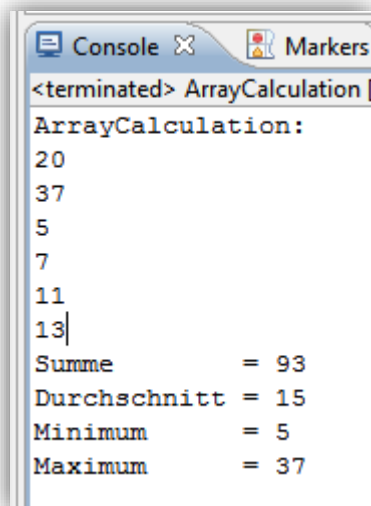


```
<terminated> ArraySearch [Java Application] C:\Progr  
Array Search: 0 10 12 4 7 21 20 13  
Nr = 21  
Position = 5
```

- d) Variante: Übergeben Sie die gesuchte Zahl als Programm Argument.

12.3. ArrayCalculation

- Erstellen sie einen Array mit den Werten: 20, 37, 5, 7, 11, 13
- Berechnen Sie die Summe und den Durchschnitt aller Werte und geben Sie diese aus.
- Suchen Sie zusätzlich den kleinsten und grössten Wert des Array und gegen Sie diese aus.



```
<terminated> ArrayCalculation [  
ArrayCalculation:  
20  
37  
5  
7  
11  
13  
Summe      = 93  
Durchschnitt = 15  
Minimum     = 5  
Maximum     = 37
```

12.4. Message

- Erstellen Sie einen Enum Message mit folgenden Werten: INFO, WARN, ERROR.
- Erstellen sie eine Klasse MessageDemo mit folgender Main Methode:

```
public static void main(String[] args) {  
    show(Message.INFO, "Die Bestellung wurde erfolgreich abgeschlossen.");  
    show(Message.WARN, "Die Bestellung ist noch nicht vollständig.");  
    show(Message.ERROR, "Die Bestellung wurde wegen einem technischen Defekt abgebrochen.");  
}
```

- Implementieren Sie in der Klasse MessageDemo eine statische Methode show, damit auf der Konsole folgenden Ausgabe erscheint:

```
Information:  
Die Bestellung wurde erfolgreich abgeschlossen.  
  
Warnung:  
Die Bestellung ist noch nicht vollständig.  
  
Fehler:  
Die Bestellung wurde wegen einem technischen Defekt abgebrochen.
```

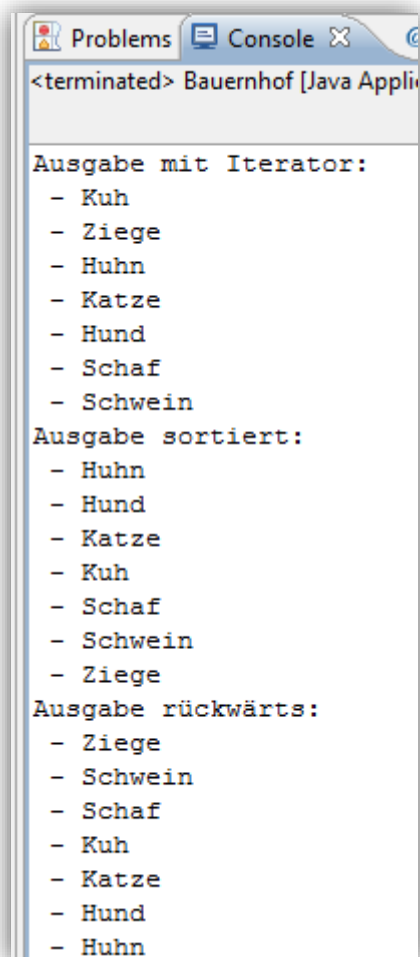
13. Collections

13.1. Bauernhof

- Erstellen sie eine Klasse Bauernhof mit einer main() Methode und definieren Sie eine Liste mit folgenden Tieren: Kuh, Ziege, Huhn, Katze, Hund, Schaf, Schwein
- Gegen Sie die Liste mit Hilfe eines Iterators aus.
- Sortieren sie die Liste und geben Sie diese mit Hilfe einer foreach Schleife aus.

Für das sortieren der Liste finden Sie in der Klasse java.util.Collections eine entsprechende Hilfsmethode.

- Geben Sie die Liste in umgekehrter Reihenfolge mit Hilfe einer for Schleife aus.



```
<terminated> Bauernhof [Java Appli...

Ausgabe mit Iterator:
- Kuh
- Ziege
- Huhn
- Katze
- Hund
- Schaf
- Schwein
Ausgabe sortiert:
- Huhn
- Hund
- Katze
- Kuh
- Schaf
- Schwein
- Ziege
Ausgabe rückwärts:
- Ziege
- Schwein
- Schaf
- Kuh
- Katze
- Hund
- Huhn
```

13.2. Book

- a) Erstellen Sie eine Klasse Book mit den Attributen title (String), author (String) und edition (Integer) für die Auflage inkl. Getter und Setter.
- b) Erstellen Sie einen leeren Konstruktor und einen Konstruktor für die Übergabe der genannten Attribute.
- c) Erstellen Sie eine toString() Methode, die die Attribute gemäss dem folgenden Beispiel als String zurückgibt: 'Faust I' von Goethe, Auflage 20000 Stück
- d) Generieren Sie mit Hilfe von Eclipse die beiden Methoden equals() und hashCode().
- e) Erstellen Sie die Klasse BookDemo mit einer main() Methode und verwalten Sie die einige Bücher mit Hilfe eines HashSet. Geben Sie die Bücher auf der Konsole aus:

```
Bücher:  
- 'Effi Briest' von Fontane, Auflage 10000 Stück  
- 'Der Untertan' von Mann, Auflage 15000 Stück  
- 'Die Aula' von Kant, Auflage 50000 Stück  
- 'Faust I' von Goethe, Auflage 20000 Stück  
- 'Faust II' von Goethe, Auflage 20000 Stück  
- 'Wilhelm Tell' von Schiller, Auflage 10000 Stück
```

Zusatz:

- f) Ergänzen Sie die Klasse Book mit dem Comparable Interface und implementieren Sie die compareTo Methode. Die Bücher sollen dabei nach titel, author und edition sortiert werden.

Tipp:

Schauen Sie sich dazu die Implementation der Demo Klasse ComparablePerson an.

- g) Ergänzen Sie die BookDemo Klasse, mit einem TreeSet und geben Sie die Bücher sortiert aus.

Tipp:

Für die Erstellung einer TreeSet können Sie direkt die bestehende Bücherliste verwenden.

```
Bücher sortiert:  
- 'Der Untertan' von Mann, Auflage 15000 Stück  
- 'Die Aula' von Kant, Auflage 50000 Stück  
- 'Effi Briest' von Fontane, Auflage 10000 Stück  
- 'Faust I' von Goethe, Auflage 20000 Stück  
- 'Faust II' von Goethe, Auflage 20000 Stück  
- 'Wilhelm Tell' von Schiller, Auflage 10000 Stück
```

14. Exceptions

14.1. NumberException

- a) Erstellen Sie eine eigene Exception „NumberException“ die von der Klasse Exception abgeleitet ist und die beiden Standard Konstruktoren bereitstellt:
 - public NumberException() {...}
 - public NumberException(String message) {...}
- b) Erstellen Sie die Klasse NumberExceptionDemo mit einer main() Methode und der folgenden Methode:

public static int **parseLottoNumber**(String lottoNumber) throws NumberException

Die Methode liest eine Lotto Nummer als String ein und konvertiert diese in einen int Wert. Dabei prüft Sie ob die Nummer im gültigen Bereich von 1..45 ist.

Parameters:

lottoNumber Die Lotto Nummer als String.

Returns:

Die konvertierte Nummer.

Throws:

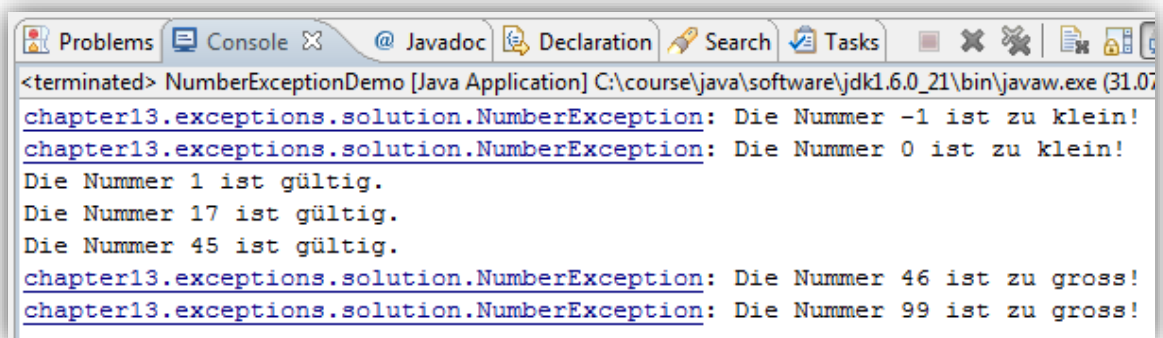
[NumberException](#) - Falls die Lotto Nummer ein ungültiges Format hat oder nicht im gültigen Bereich liegt.

- c) Rufen Sie die parseLottoNumber Methode mit folgenden Testdaten auf und geben Sie die Resultate auf der Konsole aus.

Testdaten:

"-1", "0", "1", "17", "45", "46", "99"

Beispiel Ausgabe:



```
<terminated> NumberExceptionDemo [Java Application] C:\course\java\software\jdk1.6.0_21\bin\javaw.exe (31.07)
chapter13.exceptions.solution.NumberException: Die Nummer -1 ist zu klein!
chapter13.exceptions.solution.NumberException: Die Nummer 0 ist zu klein!
Die Nummer 1 ist gültig.
Die Nummer 17 ist gültig.
Die Nummer 45 ist gültig.
chapter13.exceptions.solution.NumberException: Die Nummer 46 ist zu gross!
chapter13.exceptions.solution.NumberException: Die Nummer 99 ist zu gross!
```

15. Assertions

15.1. AssertionDemo

Testen Sie die Demo Anwendung in Eclipse je einmal mit ein- und ausgeschaltetem Assertions.

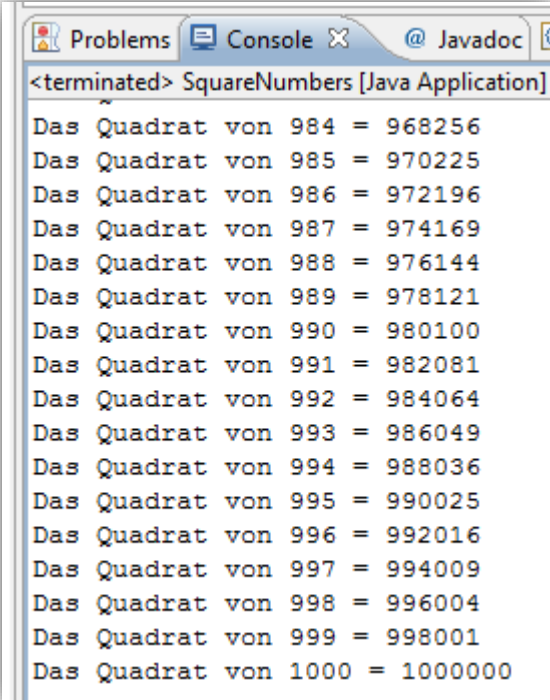
16. Dateien

16.1. SquareNumbers

- a) Erstellen Sie eine Textdatei mit dem Namen SquareNumbers.txt und speichern Sie darin die Zahlen von 1 bis 1000 und die dazugehörige Quadratzahl in der Form:

Das Quadrat von 1 = 1
Das Quadrat von 2 = 4
Das Quadrat von 3 = 9
Das Quadrat von 4 = 16
Das Quadrat von 5 = 25
u.s.w.

- b) Positionieren Sie den Dateizeiger zurück auf den Dateianfang, lesen Sie die Werte wieder ein und geben Sie diese auf die Konsole aus:



```
<terminated> SquareNumbers [Java Application]
Das Quadrat von 984 = 968256
Das Quadrat von 985 = 970225
Das Quadrat von 986 = 972196
Das Quadrat von 987 = 974169
Das Quadrat von 988 = 976144
Das Quadrat von 989 = 978121
Das Quadrat von 990 = 980100
Das Quadrat von 991 = 982081
Das Quadrat von 992 = 984064
Das Quadrat von 993 = 986049
Das Quadrat von 994 = 988036
Das Quadrat von 995 = 990025
Das Quadrat von 996 = 992016
Das Quadrat von 997 = 994009
Das Quadrat von 998 = 996004
Das Quadrat von 999 = 998001
Das Quadrat von 1000 = 1000000
```


17. Streams

17.1. PersonDemo

Es soll eine Anwendung erstellt werden, die Personendaten via Konsole erfasst und anschliessend in eine Datei speichert.

- a) Erstellen Sie eine Klasse **Person** mit Vorname, Nachname und Jahrgang. Definieren Sie die notwendigen Attribute, Konstruktoren sowie Getter und Setter Methoden.
- b) Erstellen Sie ein Klasse **PersonReader** die Personen via Konsole einliest. Die Klasse soll die folgenden beiden Methoden aufweisen:

- public Person **readPerson()** throws IOException

Die Methode liest eine Person von der Konsole ein und gibt ein Objekt von Typ Person zurück.

Die Eingabe sollte in etwa folgendermassen aussehen:

```
Geben Sie bitte eine Person ein:
Vorname  = Michelle
Nachname = Amport
Jahrgang  = 1980
```

- public List<Person> **readPersons()**

Die Methode liest mit Hilfe von readPerson() mehrere Personen ein. Nach jeder Eingabe soll der Benutzer gefragt werden ob weitere Personen eingelesen werden sollen oder nicht. Am Schluss werden die eingelesenen Personen als Liste zurückgegeben.

Die Eingabe sollte in etwa folgedermassen aussehen:

```
Geben Sie bitte eine Person ein:
Vorname  = Michelle
Nachname = Amport
Jahrgang  = 1980
Möchten Sie eine weitere Person erfassen (j/n)? j
Geben Sie bitte eine Person ein:
Vorname  = Zino
Nachname  = Müller
Jahrgang  = 1966
Möchten Sie eine weitere Person erfassen (j/n)? n
```

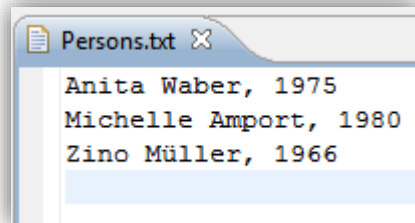
- c) Erstellen Sie die Klasse **PersonDemo** mit einer main() Methode und testen Sie die Eingabe mit Hilfe der erstellten Klassen.

- d) Nun sollen die Daten noch in eine Datei Person.txt geschrieben werden. Erstellen Sie dazu die Klasse **PersonWriter** mit folgender Methode:

– `public void writePersons(List<Person> persons)`

Die Methode speichert die Liste von Personen in eine Textdatei. Der Dateiname kann dabei via Konstruktor übergeben werden. Wird keine Angabe gemacht, so soll ein vordefinierter (Default) Dateiname verwendet werden.

Die Textdatei sollte in etwa folgendermassen aussehen



- e) Ergänzen Sie die Klasse PersonDemo so, dass nach der Eingabe, die Daten in die Datei gespeichert werden. Nach dem Speichern soll noch der Dateiname mit den gespeicherten Personen ausgegeben werden:

```
Datei <../src\chapter16\streams\solution\Persons.txt> mit folgenden Personen erstellt:
Anita Waber, 1975
Michelle Ampert, 1980
Zino Müller, 1966
```

Beispiel der ganzen Anwendung:

```
Geben Sie bitte eine Person ein:
  Vorname = Anita
  Nachname = Waber
  Jahrgang = 1975
Möchten Sie eine weitere Person erfassen (j/n)? j
Geben Sie bitte eine Person ein:
  Vorname = Michelle
  Nachname = Ampert
  Jahrgang = 1980
Möchten Sie eine weitere Person erfassen (j/n)? j
Geben Sie bitte eine Person ein:
  Vorname = Zino
  Nachname = Müller
  Jahrgang = 1966
Möchten Sie eine weitere Person erfassen (j/n)? n

Datei <../src\chapter16\streams\solution\Persons.txt> mit folgenden Personen erstellt:
Anita Waber, 1975
Michelle Ampert, 1980
Zino Müller, 1966
```

17.2. More Streams

Schauen Sie sich im Script die restlichen Abschnitte zu den Streams an und testen Sie die Demo Anwendungen dazu.

18. Misc

18.1. Lotto

a) Erstellen Sie ein Klasse Lotto mit der folgenden Methode:

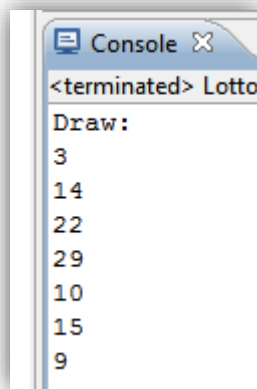
– `public int[] draw()`

Die Methode gibt einen Array mit Lotto Zahlen zurück. Total 7 Zahlen im Bereich von 1..45.

Die Zahlen werden mit der Klasse `java.util.Random` zufällig generiert, so dass jeder Aufruf von `draw()` unterschiedliche Werte liefert.

b) Erstellen Sie eine Klasse `LottoDemo` und testen Sie die `draw()` Methode. Geben Sie die Daten auf die Konsole aus.

Beispiel:



18.2. Date & Time API (java.time)

Schauen sie sich im die verschiedenen Klassen der mit Java 8 neu eingeführten Date & Time API des `java.time` Package an.

Beispiele:

Klassen	Einsatz
Instant, Duration	Die Berechnung von Zeitpunkten und deren Differenz
LocalDate, Period	Das Arbeiten mit Datumsangaben und Perioden
LocalDateTime, ZonedDateTime	Das Arbeiten mit Datumsangaben mit und ohne Zeitzonen
TemporalAdjuster	Kalendermanipulationen (wie zum Beispiel das bestimmen des ersten Montag im Jahr)
LocalTime	Berechnung von Zeiten
DateTimeFormatter	Formattierung und Darstellung von Datum und Zeit
Clock	Aktuelles Datum mit Zeit und Zeitzone

Probieren Sie die einzelnen Demo's aus und testen Sie weitere Methoden.

19. Funktionale Programmierung

19.1. Demo's

Schauen Sie sich die gezeigten Beispiel im Package demo an.

19.2. Warenhaus

Schauen Sie sich das Warenhaus Beispiel mit Package demo2 an und versuchen Sie dies nachzuvollziehen. Ergänzen Sie das Beispiel mit eigenen Erweiterungen.