



JAVA

Wiederholung

WIEDERHOLUNG ALLGEMEIN

- `public static void main(String args[]) {...}`
- Namenskonventionen:
 - KlassenNamen beginnen mit Großbuchstaben
 - Variablen- und Methoden-Namen mit kleinem Buchstaben
 - Beide haben CamelCase

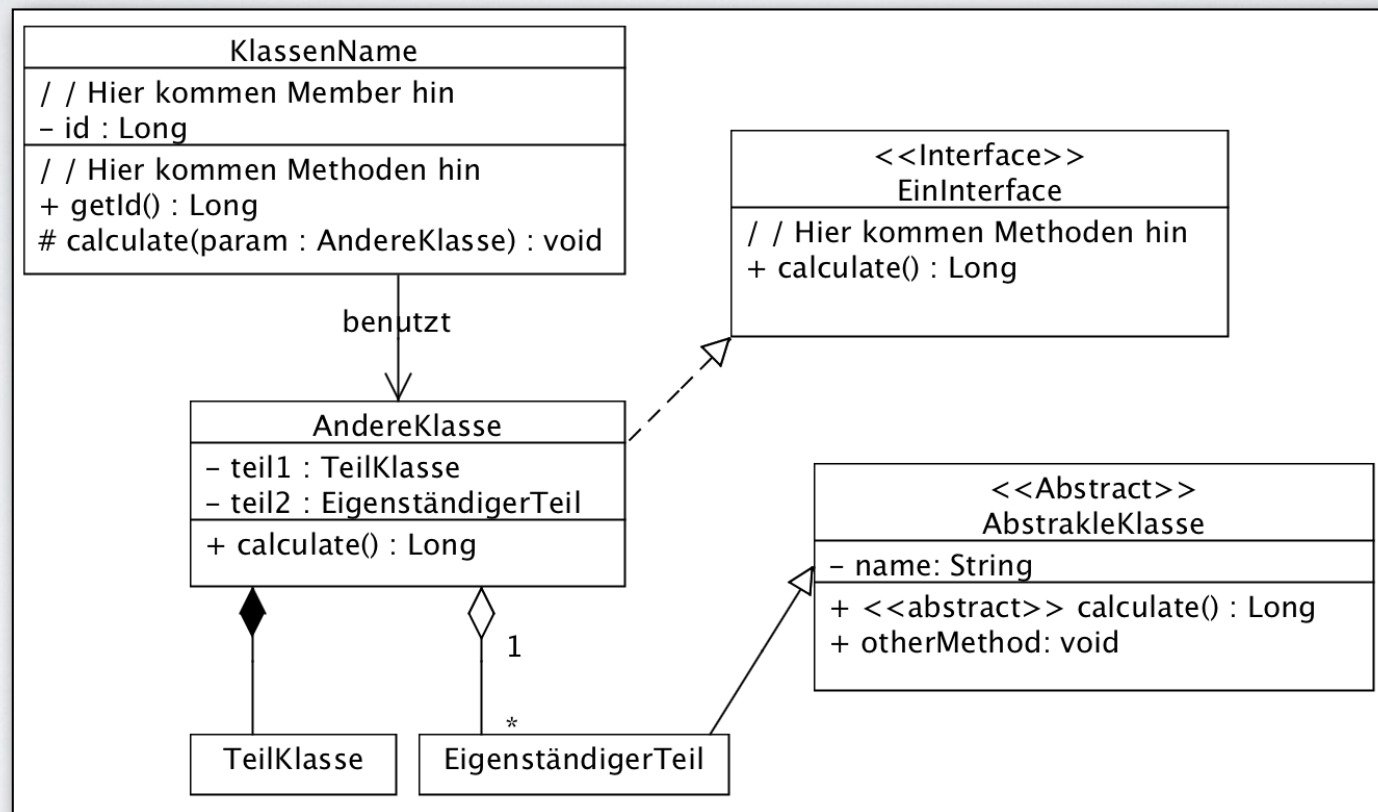
WDH REFERENZEN

```

3 public class Rechteck {
4     private int laenge;
5     private int breite;
6
7     public Rechteck(int laenge) { this(laenge, 12); }
8
9     public Rechteck(int laenge, int breite) {
10         this.laenge = laenge;
11         this.breite = breite;
12     }
13
14     private static Rechteck doSomething(Rchteck rechteck) {
15         return rechteck;
16     }
17
18     public static void main(String[] args) {
19         Rechteck rechteck1 = new Rechteck(1, 2);
20         Rechteck rechteck2 = new Rechteck(1, 2);
21         System.out.println(rechteck1 == rechteck2); // false, obwohl Inhalt gleich
22         System.out.println(rechteck1 == doSomething(rechteck1)); // true
23         String string1 = "rechteck";
24         String string2 = "rechteck";
25         String string3 = new String(string2);
26         System.out.println(string1 == "rechteck"); // true
27         System.out.println(string1 == string2); // true
28         System.out.println(string1 == string3); // false
29         System.out.println(compareStringsReference(string1, string2)); // true
30         System.out.println(compareStringsReference(string1, string3)); // false
31         System.out.println(compareStrings(string1, string2)); // true
32         System.out.println(compareStrings(string1, string3)); // true
33     }
34
35     public static boolean compareStringsReference(String string1, String string2) {
36         return string1 == string2;
37     }
38
39     public static boolean compareStrings(String string1, String string2) {
40         if (string1 != null) {
41             return string1.equals(string2);
42         }
43         return string1 == string2;
44     }
45 }

```

UML



ÜBUNG (30 MIN)

- OO-Modellierung einer Bibliothek
 - Die Bibliothek besitzt Bücher und Zeitschriften, welche an Studenten ausgeliehen werden
 - Um die Ausleihfrist zu überprüfen wird notiert, wann etwas ausgeliehen wird
- Ziel: Klassendiagramm mit Klasse, Attributen, Methoden und Beziehungen



JAVA

Java Klassenbibliotheken

AGENDA



- Allgemeines
- Collections
- Utility-Klassen
- Dateihandling
- Reflection
- Weiterführende API

UMFANG DER KLASSENbibliotheken

- Die Klassenbibliothek von Java ist groß und mächtig
- Rahmen dieser Vorstellung:
 - Was gibt es wichtiges und wo finde ich es?
- Die Klassenbibliothek ist gut in JavaDoc dokumentiert

COLLECTIONS

- Eine Collection ist eine Datenstruktur, um Mengen von Daten aufzunehmen und zu verarbeiten
 - Die Verwaltung wird gekapselt
- Ein Array ist einfachste Art der Collection
 - Collections sind aber mächtiger und einfacher zu benutzen
 - Daher werden Arrays nur relativ selten in Java benutzt

WICHTIGSTE COLLECTIONS

- `java.util.*`
- Seit Java 5 sind die Collections generisch
- Namens Konvention: `<Stil><Interface>`
- Wichtigste Interfaces: **List, Set, Map, Queue**
- **Tree** vs. **Hash** Implementierungen

ARBEITSWEISE HASH-TABELLE

```
Map<Integer, String> mitarbeiter = new HashMap<>();  
mitarbeiter.put(4711, "Klaus");  
mitarbeiter.put(4712, "Peter");  
mitarbeiter.put(4747, "Hans");  
System.out.println(mitarbeiter.get(4711));
```

- Schlüssel-Wert-Paare
- Berechnung Schlüssel durch Hash-Funktion
- Schlüssel dient als Index eines internen Arrays
- Füllgrad bzw. load factor

IMPLEMENTIERUNGEN LIST

- ArrayList
- LinkedList
- Stack
- Vector

IMPLEMENTIERUNGEN SET

- EnumSet
- HashSet
- TreeSet

IMPLEMENTIERUNGEN MAP

- EnumMap
- HashMap
- Hashtable (auch klein table!)
- TreeMap

DIE KLASSE COLLECTIONS

- Statische Methoden zur Manipulation und Verarbeitung von Collections
- Besonders die Methoden zum Synchronisieren sind im Zusammenhang mit Multithreading wichtig

```
public CollectionsBeispiel() {  
    ... List<Integer> meineListe = new LinkedList<>();  
    ... meineListe.add(5);  
    ... meineListe.add(2);  
    ... meineListe.add(10);  
    ... // sortiert Liste aufsteigend  
    ... // Integer implementiert das Interface Comparable<Integer>  
    ... Collections.sort(meineListe);  
    ... // Ein spezieller Vergleich,   
    ... // der zur Sortierung heran gezogen wird  
    ... Comparator<Integer> vergleich = new Comparator<Integer>() {  
        ... @Override  
        ... public int compare(Integer int1, Integer int2) {  
            ... return int2.compareTo(int1);  
        }  
    };  
    ... // Liste speziell sortieren  
    ... Collections.sort(meineListe, vergleich);  
    ... // dreht die Elemente der Liste (wieder) um  
    ... Collections.reverse(meineListe);  
    ... List<Integer> synchronisierteListe =  
        ... Collections.synchronizedList(meineListe);  
    ... List<Integer> unmodifizierbareListe =  
        ... Collections.unmodifiableList(meineListe);  
    ... Integer max = Collections.max(meineListe);  
    ... // ...  
}
```

DIE KLASSE ARRAYS

- Statische Methoden zur Manipulation und Verarbeitung von Arrays
- parallelSort

```
private void arraysBeispiele() {  
    ... Integer[] meinArray = {5, 2, 10, 25, 1, 20};  
    ... List<Integer> alsListe = Arrays.asList(meinArray);  
    ... // sortiert Liste aufsteigend  
    ... Arrays.sort(meinArray);  
    ... // Vergleich von Arrays.equals geht auf den Inhalt!  
    ... Integer[] zweiterArray = {1, 2, 3};  
    ... boolean vergleich = Arrays.equals(meinArray, zweiterArray);  
    ... // Es können Elemente gesucht werden  
    ... int indexOfSearch = Arrays.binarySearch(meinArray, 20);  
    ... System.out.println(Arrays.toString(meinArray));  
    ... // Gibt "[1, 2, 5, 10, 20, 25]" aus  
}
```

WICHTIGE UTILITY KLASSEN

- Utility Klassen sind Klassen, welche sich nicht konkret einordnen lassen, die man aber immer wieder benötigt
- Wichtige Utility Klassen
 - java.util.Random
 - java.security.SecureRandom
 - java.util.GregorianCalendar und java.util.Date

```
private void utilities() {  
    Random random =  
        new Random(System.currentTimeMillis() % 14930352);  
    for(int i = 0; i<10;i++) {  
        System.out.println(random.nextInt());  
    }  
}
```

Vergleiche: <http://joda-time.sourceforge.net>

WICHTIGE UTILITY KLASSEN

- `java.lang.System`
 - `getProperty()` (HomeVerzeichnis, Tempverzeichnis, ...)
 - `getEnv()`
 - `lineSeparator()`
 - Standard Streams (in, out, err)
 - `exit()`
 - `gc()`
 - `currentTimeMillis()`

WICHTIGE UTILITY KLASSEN

- `java.lang.Runtime`
- `java.lang.Math`
- `java.math.BigDecimal` & `BigInteger`

NEUE DATE API

8

```
public class NewDates {  
    public static void main(String[] args) {  
        Clock clock = Clock.systemUTC(); // UTC der Systemzeit  
  
        Clock clock = Clock.systemDefaultZone(); // Uhrzeit in der Zeitzone des Systems  
  
        long time = clock.millis(); // Millisekunden seit 01.01.1970  
  
        ZoneId zone = ZoneId.of("Europe/Berlin"); // Zeitzone für Namen  
  
        Clock clock = Clock.system(zone); // Setzt eine Zeitzone  
  
        LocalDate date = LocalDate.now(); // LocalDate = menschenlesbare Zeitangaben  
        String year = date.getYear();  
        String month = date.getMonthValue();  
        String day = date.getDayOfMonth();  
  
        Period p = Period.of(2, HOURS); // Zeitperioden  
        LocalDateTime time = LocalDateTime.now();  
        LocalDateTime newTime = time.plus(p);  
    }  
}
```

DATEIEN UND VERZEICHNISSE

- Java bietet sehr umfangreiche API zum Datei- und Verzeichnishandling an
 - Handling von Dateien und Verzeichnisse selbst
 - Streams zum sequentiellen I/O
 - Random I/O
- `java.io.*`

HANDLING VON DATEIEN

- Dateien und Verzeichnisse sind Objekte, **nicht aber der Inhalt** von Dateien!

```
public FileBeispiel() {  
    ... // Abstrahiert von Datei und Verzeichnis  
    ... File homeVerzeichnis = new File("/Users/junterstein");  
    ... File prsentation = new File(homeVerzeichnis.getAbsolutePath()  
    ...     + "/documents/java12.key");  
    ... // Absolute und relative Namen untersttzt  
    ... File testFile = new File("./test.jar");  
    ... // Abfrage der Datei-/Verzeichnisattribute  
    ... System.out.println(testFile.exists() + ": " + testFile.canRead()  
    ...     + ": " + testFile.canWrite());  
    ... // Iterieren ber Verzeichnisse  
    ... for (File kind : homeVerzeichnis.listFiles()) {  
    ...     if (kind.isFile()) {  
    ...         ... // tue etwas  
    ...     }  
    ...     if (kind.isDirectory()) {  
    ...         ... // tue etwas anderes  
    ...     }  
    ... }  
    ... // Anlegen und Lschen von Dateien und Verzeichnissen  
    ... testFile.delete();  
    ... // mkdir() legt nur den angegebenen Ordner an, wenn der Vater existiert  
    ... // mkdirs() legt alle Ordner an, bis der bergebene Pfad erreicht ist  
    ... new File(homeVerzeichnis.getAbsolutePath() + "/mein/ausgedachter/Pfad").mkdirs();  
    ... // Verwalten Temporrer Dateien  
    ... try {  
    ...     File temporreDatei = File.createTempFile("meinProjektName", "meineDateiEndung");  
    ... } catch (IOException e) {  
    ...     e.printStackTrace();  
    ... }  
}
```

JAVA.NIO.FILE.PATHS

- Pfadverkettung bitte mit Path und Paths

```
10 public class PathBeispiel {  
11  
12     public static void main(String[] args) {  
13         File file = new File("mein/ausgedachter/Pfad");  
14         File newFile = new File(file.getAbsolutePath() + "../..../was/lustiges/dran"); // <-- unüblich  
15         Path path = Paths.get(file.getAbsolutePath(), "../..../was/lustiges/dran");  
16         Path parent = path.getParent();  
17         File parentFile = parent.toFile();  
18         Path parent2 = parentFile.toPath();  
19     }
```

STREAMS

- Sehr abstraktes Konstrukt, zum Zeichnen auf imaginäres Ausgabegerät zu schreiben oder von diesem zu lesen
 - Erste konkrete Unterklassen binden Zugriffsroutinen an echte Ein- oder Ausgabe
- Streams können verkettet und geschachtelt werden
 - Verkettung: Zusammenfassung mehrerer Streams zu einem
 - Schachtelung: Konstruktion von Streams mit Zusatzfunktion
- Bitte apache-commons verwenden: FileUtils, IOUtils, StreamUtils, ...

CHARACTER- UND BYTE-STREAMS

- Byte-Streams: Jede Transporteinheit genau 1 Byte lang
 - Problem bei Unicode (> 1 Byte) & Umständlich
- Character-Streams: Unicode-fähige textuelle Streams
- Wandlung von Character- und Byte-Streams und umgekehrt möglich

CHARACTER-STREAMS FÜR DIE AUSGABE

- Abstrakte Klasse Writer
 - OutputStreamWriter
 - FileWriter
 - PrintWriter
 - BufferedWriter
 - StringWriter
 - CharArrayWriter
 - PipedWriter

```
private void charWriter() {  
    String s;  
    FileWriter fw = null;  
    BufferedWriter bw = null;  
    try {  
        fw = new FileWriter("buffer.txt");  
        bw = new BufferedWriter(fw);  
        for (int i = 1; i <= 10000; ++i) {  
            s = "Dies ist die " + i + ". Zeile";  
            bw.write(s);  
            bw.newLine();  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        // Java Version < 7 Bedarf etwas  
        // mehr aufwand zum stream closen  
        try {  
            if (bw != null) {  
                bw.close();  
            }  
            if (fw != null) {  
                fw.close();  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

CHARACTER-STREAMS FÜR DIE EINGABE

- Abstrakte Klasse Reader
 - InputStreamReader
 - FileReader
 - BufferedReader
 - LineNumberReader
 - StringReader
 - CharArrayReader
 - PipedReader

```
private void charReader() {  
    ... String line = null;  
    ... try (BufferedReader br =  
    ...         new BufferedReader(new FileReader("config.sys"))) {  
    ...         while ((line = br.readLine()) != null) {  
    ...             System.out.println(line);  
    ...         }  
    ...         br.close();  
    ...     } catch (Exception e) {  
    ...         e.printStackTrace();  
    ...     }  
}
```

BYTE-STREAMS

- Bei Character-Streams wird von Readern und Writern (analog zur jeweiligen Basisklasse) gesprochen, hier spricht man von Byte-Streams, also InputStreams und OutputStreams
- Abstrakte Klasse InputStream und OutputStream
- Funktionieren genauso wie Character-Streams, arbeiten jedoch auf Bytes
- Spezialfall: ZipOutputStream und ZipInputStream im Paket `java.util.zip` zum Schreiben und lesen von Archivdateien

RANDOM I/O

- Streams vereinfachen den sequentiellen Zugriff auf Dateien
- Manchmal wahlfreier Zugriff auf Dateien notwendig
- `RandomAccessFile` stellt entsprechende Methoden zur Verfügung um in Dateien zu navigieren/lesen/schreiben

INTROSPECTION & REFLECTION

- Möglichkeit, zur Laufzeit Klassen zu instanziiieren und zu verwenden ohne diese zur Compilezeit zu kennen
- Abfragemöglichkeiten, welche Member eine Klasse besitzt
- Reflection ist ein sehr mächtiges Werkzeug, sollte aber mit Bedacht eingesetzt werden
 - Reflection Code schlägt oft erst zur Laufzeit fehl
 - Macht Code schwer lesbar

INTROSPECTION & REFLECTION

- Die Klasse `java.lang.Class`
 - Erreichbar über `getClass()` der Klasse `Object`
 - Methoden zur Abfrage der Member der Klasse, sowie weiterer Eigenschaften

```
.try {  
    ....String meinKlassenName = "ba.java.auto.AudiQFuenf";  
    ....Class<?> meineKlasse = Class.forName(meinKlassenName);  
    ....Object meinObjekt = meineKlasse.newInstance();  
    ....AudiQFuenf meinAudi = (AudiQFuenf) meinObjekt;  
} catch (Exception e) {  
    ....e.printStackTrace();  
}
```

```
.try {  
    ....String str = "Test";  
    ....Class<?> clazz = str.getClass();  
    ....Method m = clazz.getDeclaredMethod("length");  
    ....Object ret = m.invoke(str);  
    ....System.out.println(ret);  
} catch (Exception e) {  
    ....e.printStackTrace();  
}
```

ÜBUNGEN

- <https://github.com/unterstein/dhbw-java-lecture/tree/master/src/ba/java/uebungen>
- <https://git.io/vrF8U>

SERIALISIERUNG

- Fähigkeit, ein Objekt im Hauptspeicher in ein Format zu konvertieren, um es in eine Datei zu schreiben oder über das Netzwerk zu transportieren
- Deserialisierung: Umkehrung der Serialisierung in ein Objekt
- Manchmal wird Serialisierung zur Persistenz eingesetzt
 - Meist nicht die klassische Serialisierung, sondern XML, JSON
- Standard Serialisierung ist binärbasiert!

XML SUPPORT

- eXtensible Markup Language
- XML ist ein semi-strukturiertes Datenformat
- Es können in XML eigene Strukturen abgebildet werden, die jedoch nur mit eigenem allgemeinen Parser verarbeitet werden können
- Java liefert DOM und SAX Parser mit und bietet Möglichkeiten zur Transformation von XML Dokumenten

ANNOTATIONS

- Möglichkeit Java Code mit Meta Informationen zu versehen
- Nur ganz kurze Einführung in Zusammenhang mit JAXB!
- @Override, ...

```

9 public class AnnotationBeispiel {
10
11     ... public AnnotationBeispiel() {
12         ... UserBean bean = new UserBean();
13         ... bean.name = "James";
14         ... bean.surename = "Hetfield";
15         ... bean.birthDate = new Date(63, 7, 3); // 3. August 1963
16         ... // XMLSerializeHelper ist kein Bestandteil von Java
17         ... System.out.println(XMLSerializeHelper.instance().serialize(bean));
18     }
19
20     ... @XmlElement
21     ... public static class UserBean implements Serializable {
22         ... private static final long serialVersionUID = -909105371662696039L;
23
24         ... @XmlElement(name = "vorname")
25         ... public String name;
26         ... public String surename;
27         ... @XmlElement(name = "geburtstag")
28         ... public Date birthDate;
29     }
30 }

```

Console

```

<terminated> AnnotationBeispiel [Java Application] /Library/Java/JavaVirtualMachines/1.7.0.jdk/Contents/H
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<userBean>
  <vorname>James</vorname>
  <surename>Hetfield</surename>
  <geburtstag>1963-08-03T00:00:00+01:00</geburtstag>
</userBean>

```

JDBC

- Java Database Connectivity
- Schnittstelle zwischen einer Applikation und einer SQL-DB
- Es sind weitere Treiber der Datenbank-Hersteller notwendig

NETZWERK

- Java stellt API zur Socket-Programmierung via TCP/IP bereit
- Über Streams ist das Lesen und Schreiben möglich
- Dies ist im Allgemeinen umständlich, weshalb meist andere Mechanismen verwendet werden

RMI

- Remote Method Invocation
- Socket Programmierung erlaubt Austausch von Daten
 - RMI erlaubt transparente Methodenaufrufe auf Server
- RMI abstrahiert
- RMI ist nur eine Möglichkeit für verteilte Objekte

SICHERHEIT UND KRYPTOGRAPHIE

- Daten müssen oft verschlüsselt werden
 - Verschlüsselung von Dateien
 - Verschlüsselung von Daten bei Netzwerktransport
- Symmetrische, Asymmetrische Verfahren und Signaturen unterstützt

MULTITHREADING

- Konzepte der Nebenläufigkeit von Programmteilen
 - Ähneln einem Prozess, arbeitet aber auf einer feineren Ebene
- Threads sind in Java direkt als Sprachkonstrukt umgesetzt
- Threads können synchronisiert werden

MULTITHREADING BEISPIEL

```

3 public class ThreadBeispiel {
4
5     ... public ThreadBeispiel() {
6         ... Runnable codeImThread = new Runnable() {
7
8             @Override
9             ... public void run() {
10                 ... while (true) {
11                     ... System.out.println("als runnable");
12                     ... try {
13                         ... Thread.sleep(1000);
14                     } catch (InterruptedException e) {
15                         ... e.printStackTrace();
16                     }
17                 }
18             }
19         };
20         ... Thread thread1 = new Thread(codeImThread);
21         ... Thread thread2 = new Thread() {
22             @Override
23             ... public void run() {
24                 ... while (true) {
25                     ... System.out.println("direkt überschrieben");
26                     ... try {
27                         ... Thread.sleep(300);
28                     } catch (InterruptedException e) {
29                         ... e.printStackTrace();
30                     }
31                 }
32             }
33         };
34         ... thread1.start();
35         ... thread2.start();
36     }

```

Console Output:

```

<terminated> ThreadBeispiel
direkt überschrieben
als runnable
direkt überschrieben
direkt überschrieben
direkt überschrieben
als runnable
direkt überschrieben
direkt überschrieben
direkt überschrieben
als runnable
direkt überschrieben
direkt überschrieben
direkt überschrieben
als runnable
direkt überschrieben
direkt überschrieben
als runnable
direkt überschrieben
direkt überschrieben
als runnable
direkt überschrieben

```



JAVA

Tests

TESTING

- Tests sind wichtig
- Tests erleichtern euer Leben
- Tests bringen Vertrauen und Sicherheit
- Tests bringen nichts, wenn sie nichts testen

JUNIT

- Externe Bibliothek
- Defacto Standard
- `@Test`
- Assertions

ÜBUNGEN

- <https://github.com/unterstein/dhbw-java-lecture/tree/master/src/ba/java/uebungen>
- <https://git.io/vrF8U>