Typen und Klassen

Variable bzw. Ausdruck:

hat einen Typ.

Typ ist zur Compilezeit überprüfbar.

Objekt:

ist Exemplar einer Klasse.

Klasse ist erst zur Laufzeit bekannt.

Wenn Variable x ein Objekt obj enthält, dann gilt: Typ(x) $\geq$ Klasse(obj)

$\Rightarrow$ Typ von x erlaubt zur Compilezeit Abschätzung, welche Methoden für obj definiert sind.

Überschreiben von Methoden

```
class A {
  public D test(C someC) {
    ...
  }
}
```

```
class B extends A {
  public D test(C someC) {
    ...
  }
}
```

Unterklasse muß Vertrag der Oberklasse einhalten:

Gleicher Resultattyp.

Keine restriktiveren Zugriffsrechte.

Keine „neuen“ Exceptions.

final (I)

```
final BitSet used = new BitSet();
// Die Referenz used darf nicht
// verändert werden (aber das
// Objekt selbst durchaus!)

final int DIMENSION;
...
DIMENSION = 3;

class Point {
  ...
  public double dist(final Point pt) {
    // Innerhalb dieser Methode darf
    // pt nicht verändert werden.
    ...
  }
}
```


final (II)

```
class Point {
  ...
  final public double dist(Point pt) {
    // Diese Methode darf in einer
    // Unterklasse von Point nicht
    // überschrieben werden.
    ...
  }
}

final class NoExtending {
  // Diese Klasse kann keine
  // Unterklassen haben.
  ...
}
```

final (III)

Vor- und Nachteile von `final`:

- Erhöhte Sicherheit (Variable können nicht versehentlich überschrieben werden; Unterklassen können Verhalten der Oberklasse nicht ändern).
- Erhöhte Effizienz (Codeoptimierung, z. B.: Inlining)
- Verringerte Flexibilität (`final`-Methoden verhindern möglicherweise Wiederverwendung).

Exceptions (I)

Klassenhierarchie:

- `Throwable`
  - `Error`
    - z. B.: `OutOfMemoryError`, `ClassFormatError`
  - `Exception`
    - `RuntimeException`
      - z. B.: `ArithmeticException`, `NullPointerException`, `IndexOutOfBoundsException`
    - sonstige
      - z. B.: `FileNotFoundException`, `EOFException`

Exceptions (II)

Alle `Throwable`s, die eine Methode potentiell werfen kann, müssen deklariert werden ...

```
public void f(int n)
    throws TooBigException,
           TooSmallException,
           DataCorruptException {
  ...
}
```

... außer: (Unterklassen von) Error und RuntimeException.

Ein-/Ausgabe (I)

Stream:

- Objekt, von dem sequentiell gelesen oder auf das sequentiell geschrieben werden kann.
- Kann z. B. mit einer Datei verbunden sein (oder auch: Netzverbindung, String, anderer Stream).

Ein-/Ausgabe (II)

Java 1.0: byteweise Ein-/Ausgabe.

- `InputStream` (z.B.: `System.in`)
  - `FileInputStream`
  - `BufferedInputStream`
  - ...
- `OutputStream`
  - `FileOutputStream`
  - `BufferedOutputStream`
  - `PrintStream` (z.B.: `System.out`, `System.err`)
  - ...

Ein-/Ausgabe (III)

Java 1.1: zeichenweise Ein-/Ausgabe.

- `Reader`
  - `InputStreamReader`
    - `FileReader`
  - `BufferedReader`
  - `StringReader`
  - ...
- `Writer`
  - `OutputStreamWriter`
    - `FileWriter`
  - `BufferedWriter`
  - `PrintWriter`
  - `StringWriter`
  - ...
- `StreamTokenizer`

Ein-/Ausgabe (IV)

Textdatei lesen: zeichenweise, ungepuffert.

```
int ch;
try {
  FileReader in =
            new FileReader("file.txt");
  try {
    while( (ch = in.read()) != -1 ) {
      ...
    }
    in.close();
  } catch (Exception e) {
    System.out.println
                   ("Read error " + e);
  }
} catch (Exception e) {
  System.out.println
                   ("Open error " + e);
}
```


Ein-/Ausgabe (V)

Textdatei lesen: zeilenweise, gepuffert.

```
String line;
try {
  BufferedReader in =
      new BufferedReader(
          new FileReader("file.txt") );
  try {
    while( (line = in.readLine())
                              != null ) {
      ...
    }
    in.close();
  } catch (Exception e) {
    System.out.println("Read error");
  }
} catch (Exception e) {
  System.out.println("Open error");
}
```

Dokumentationskommentare (I)

```
/**
 * Testet, ob ein Student an der
 * Vorlesung teilgenommen hat.
 * Ein Student hat teilgenommen,
 * falls er mindestens ein Übungsblatt
 * bearbeitet hat.
 */
public boolean participate
      (String name, int number)
            throws UnknownStudent
```


Dokumentationskommentare (II)

```
/**
 * Testet, ob ein Student an der
 * Vorlesung teilgenommen hat.
 *
 * @param name      Name des Studenten.
 * @param number    Matrikelnummer.
 *
 * @return          true, falls teilgen.
 *
 * @exception UnknownStudent
 *                  Student mit Name
 *                  und Matrikelnummer
 *                  existiert nicht.
 */
public boolean participate
      (String name, int number)
            throws UnknownStudent
```


Dokumentationskommentare (III)

```
/**
 * Testet, ob ein Student an der
 * Vorlesung teilgenommen hat.
 *
 * @author         Anna Bremser
 * @author         Theo Bremser
 *
 * @version        1.2
 *
 * @since          1.1
 *
 * @deprecated     Benutze stattdessen
 *                 participate(Student).
 * @see            participate(Student)
 */
public boolean participate
      (String name, int number)
            throws UnknownStudent
```