Universität Bremen

Praktische Informatik 1
Fehlerbehandlung 2

Thomas Röfer

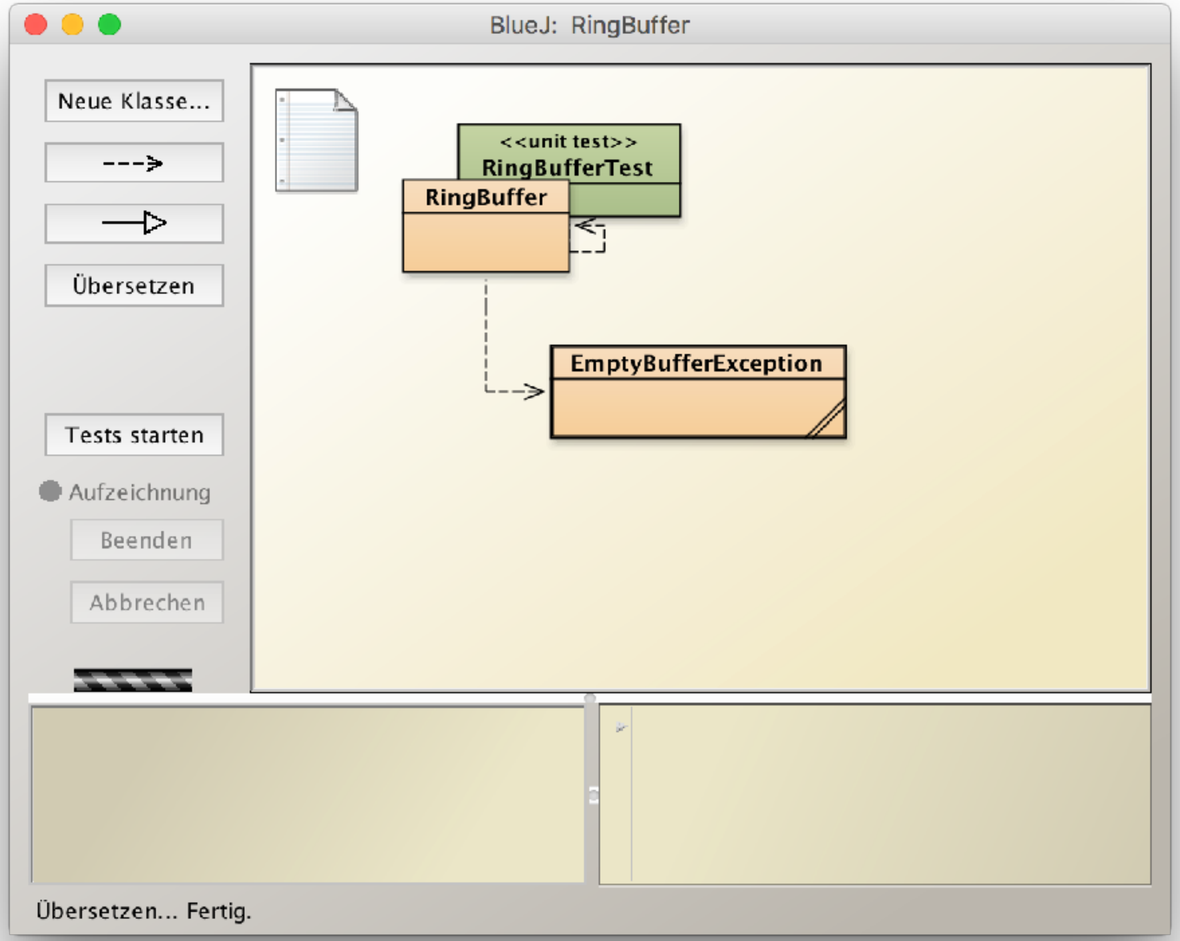
Cyber-Physical Systems
Deutsches Forschungszentrum für Künstliche Intelligenz

Multisensorische Interaktive Systeme
Fachbereich 3, Universität Bremen

Bild: ©Michael Ihle

Universität Bremen

RingBuffer mit Exceptions: Demo



Universität Bremen

Neue Exception-Klassen definieren

- Eigene Exception-Klassen definieren, um
  - neue Fehlertypen zu definieren
  - weitere Informationen zurückzuliefern
- Können geprüft oder ungeprüft sein
- **assertThrows** prüft, ob eine Exception erzeugt wird (mit Lambda-Ausdruck, → später)

```
class EmptyBufferException
        extends RuntimeException
{
    EmptyBufferException()
    {
        super("Zugriff auf leeren Ringpuffer");
    }
}
```

```
assertThrows(EmptyBufferException.class,
        () -> new RingBuffer(3).pop())
```

Universität Bremen

Wiederaufsetzen

- Erfordert üblicherweise komplexere Kontrollstrukturen als für Fälle, in denen kein Fehler auftreten kann
- Anweisungen in der **catch**-Klausel stellen die Voraussetzungen für ein Wiederaufsetzen her

```
final JFileChooser fc = new JFileChooser();
while (fc.showOpenDialog(null)
        == JFileChooser.APPROVE_OPTION) {
    try {
        return new FileInputStream(fc.getSelectedFile());
    }
    catch (final FileNotFoundException e) {
        JOptionPane.showMessageDialog(null,
                "Datei nicht gefunden!", "Meine App",
                JOptionPane.ERROR_MESSAGE);
    }
}
return null;
```

- Wiederaufsetzen bedeutet häufig Wiederholung
- Erfolgreiches Wiederaufsetzen kann nicht garantiert werden
- Endlos wiederholtes, erfolgloses Wiederaufsetzen verhindern

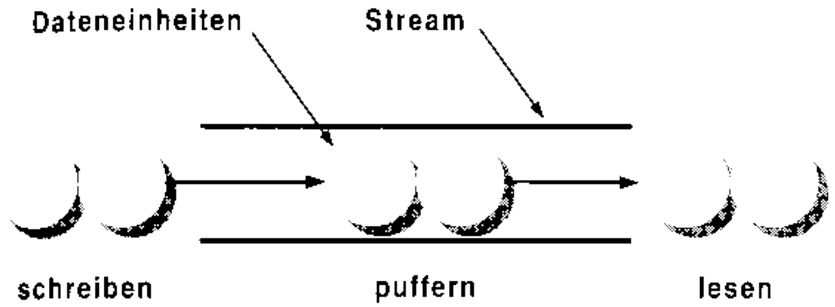
Universität Bremen

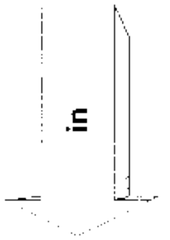
Fehlervermeidung

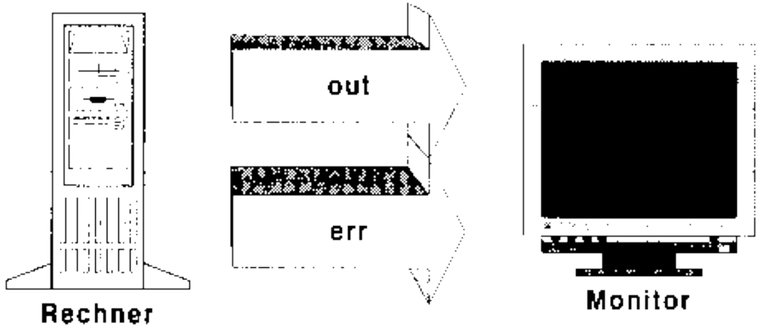
- Klient kann mit Hilfe von Methoden des Dienstleisters Fehler vermeiden
  - Exception tritt doch auf → Programmierfehler im Klienten
  - Ungeprüfte Exceptions → Kein **try**-Block im Klienten notwendig
- Nachteile
  - Klient bekommt Einblick in innere Abläufe des Dienstleisters
  - Wenn Klient und Dienstleister prüfen, wird Code doppelt ausgeführt

Datenströme

- Leiten Daten seriell in ein Programm hinein bzw. wieder heraus
- Können umgeleitet werden, z.B. in eine Datei
- Standardeingabe: **System.in** (Instanz von **InputStream**)
- Standardausgabe: **System.out** (Instanz von **PrintStream**)
- Standardfehlerausgabe: **System.err** (Instanz von **PrintStream**)
- Paket **java.io**

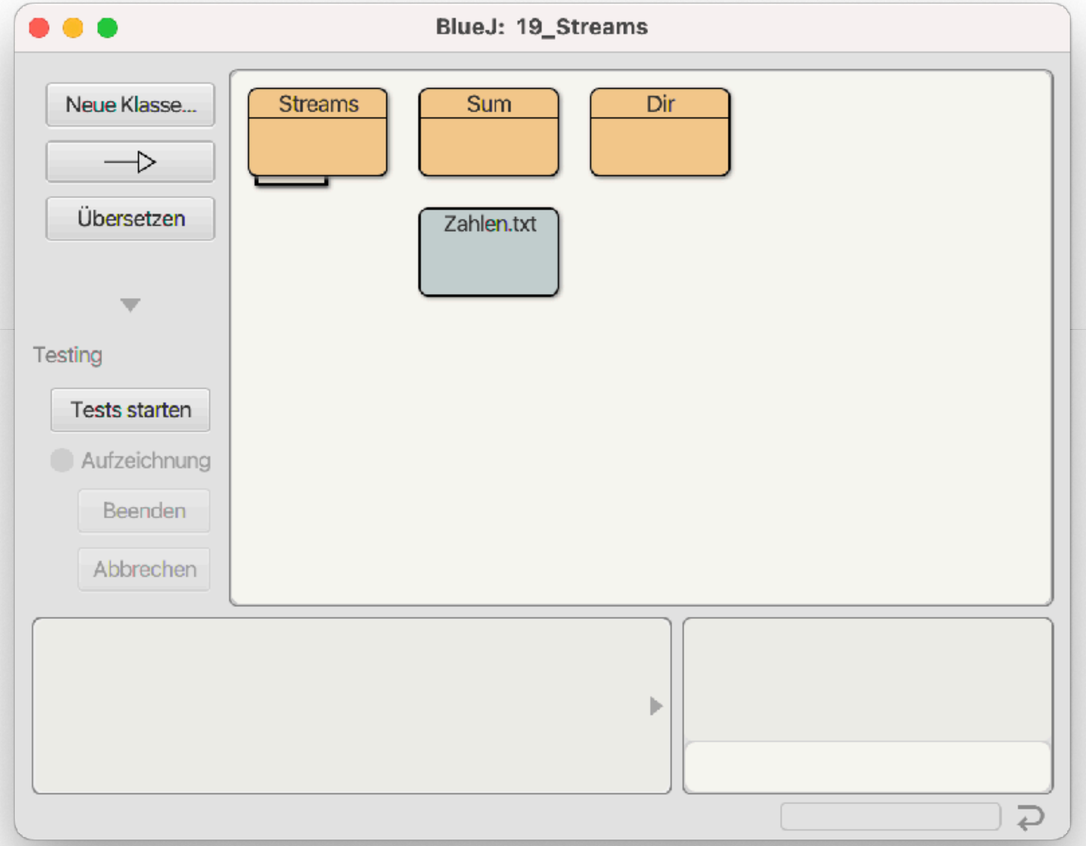






Universität Bremen

Datenströme: Demo

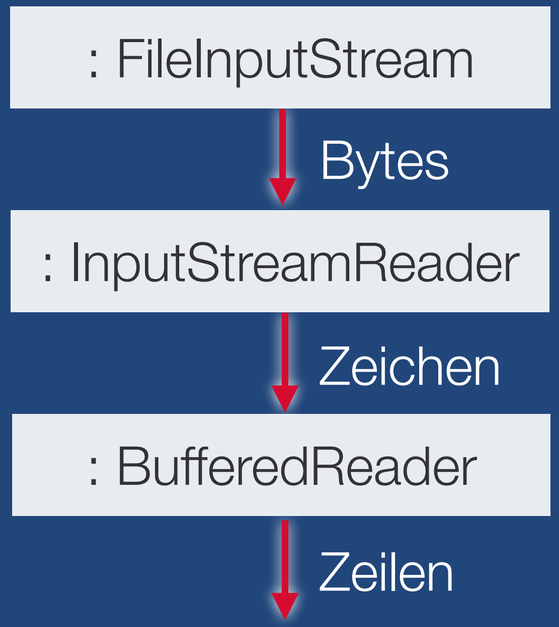


Universität Bremen

Alternativ:
new FileReader(filename, StandardCharsets.UTF_8)

```
static void more(final String filename) throws FileNotFoundException, IOException
{
    final InputStream stream = new FileInputStream(filename);
    final Reader reader = new InputStreamReader(stream, "UTF-8");
    final BufferedReader buffer = new BufferedReader(reader);
    System.out.print("\f");
    String line;
    int count = 1;
    while ((line = buffer.readLine()) != null) {
        System.out.println(count + ":\t" + line);
        if (++count % 20 == 1) {
            System.in.read();
            System.out.print("\f");
        }
    }
}
```

Zuweisung
in einem Ausdruck



Universität Bremen

Byte-Datenströme in Java

- **InputStream**: Basisklasse von z.B. **FileInputStream**, **ByteArrayInputStream** ...
  - Lesen: **int read()**, **int read(byte[ ] b)** ...
  - Überspringen von Daten: **void skip(long n)**
- **OutputStream**: Basisklasse von z.B. **FileOutputStream**, **ByteArrayOutputStream**, **PrintStream** ...
  - Schreiben: **void write(int b)**, **void write(byte[ ] b)** ...
  - Wegschreiben der Daten: **flush()**
- Schließen des Datenstroms: **close()**

Universität Bremen

Zeichen-Datenströme in Java

- **Reader**: Basisklasse von z.B. **InputStreamReader**, **StringReader**, **BufferedReader**, **FileReader** ...
  - Lesen: **int read()**, **int read(char[ ] c)** ...
  - Überspringen von Zeichen: **void skip(long n)**
- **Writer**: Basisklasse von z.B. **OutputStreamWriter**, **StringWriter**, **PrintWriter**, **BufferedWriter**, **FileWriter** ...
  - Schreiben: **void write(int c)**, **void write(char[ ] c)** ...
  - Wegschreiben der Zeichen: **flush()**
- Schließen des Datenstroms: **close()**

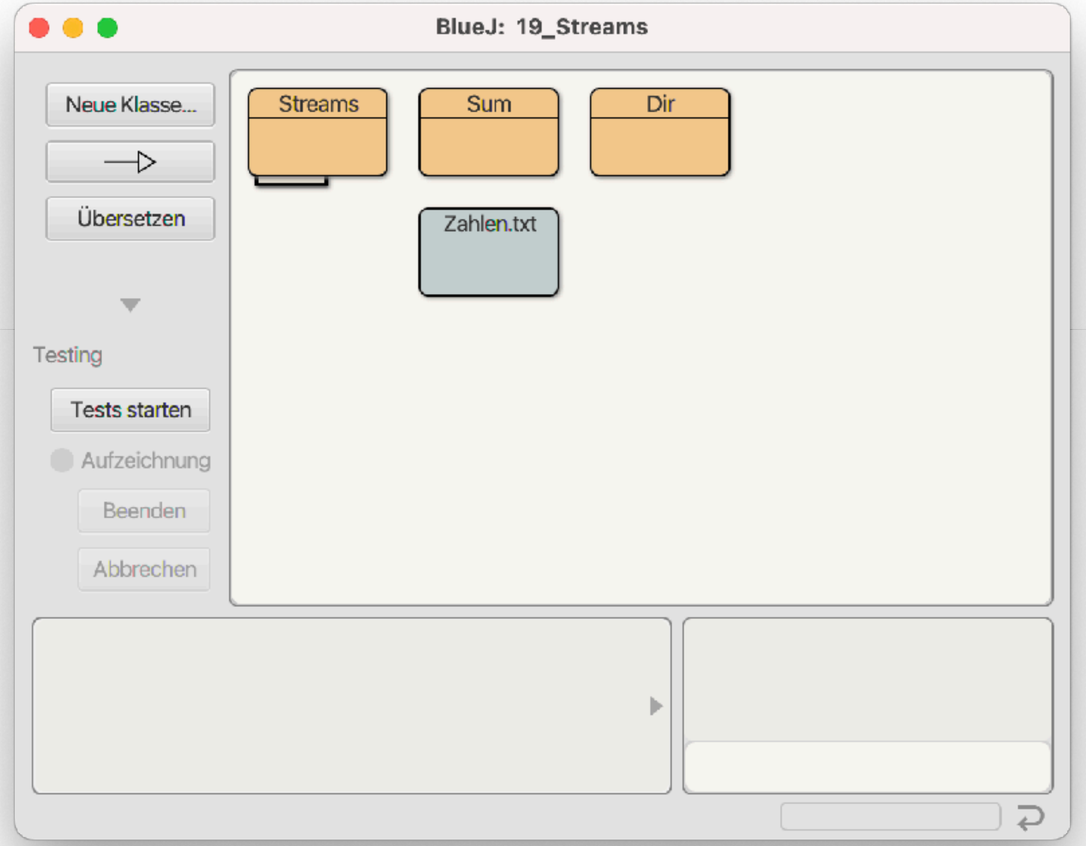
Universität Bremen

Datenströme in Java

- **int read()** liefert das nächste gelesene Byte/Zeichen zurück
  - **-1**: Das Ende des Datenstroms wurde erreicht (End Of File/EOF)
- **String readLine()** (**BufferedReader**) liefert eine gesamte Textzeile zurück
  - **null**: Das Ende des Datenstroms wurde erreicht
- **IOException**: Fehler beim Lesen aufgetreten

Universität Bremen

Datenströme schließen: Demo



Universität Bremen

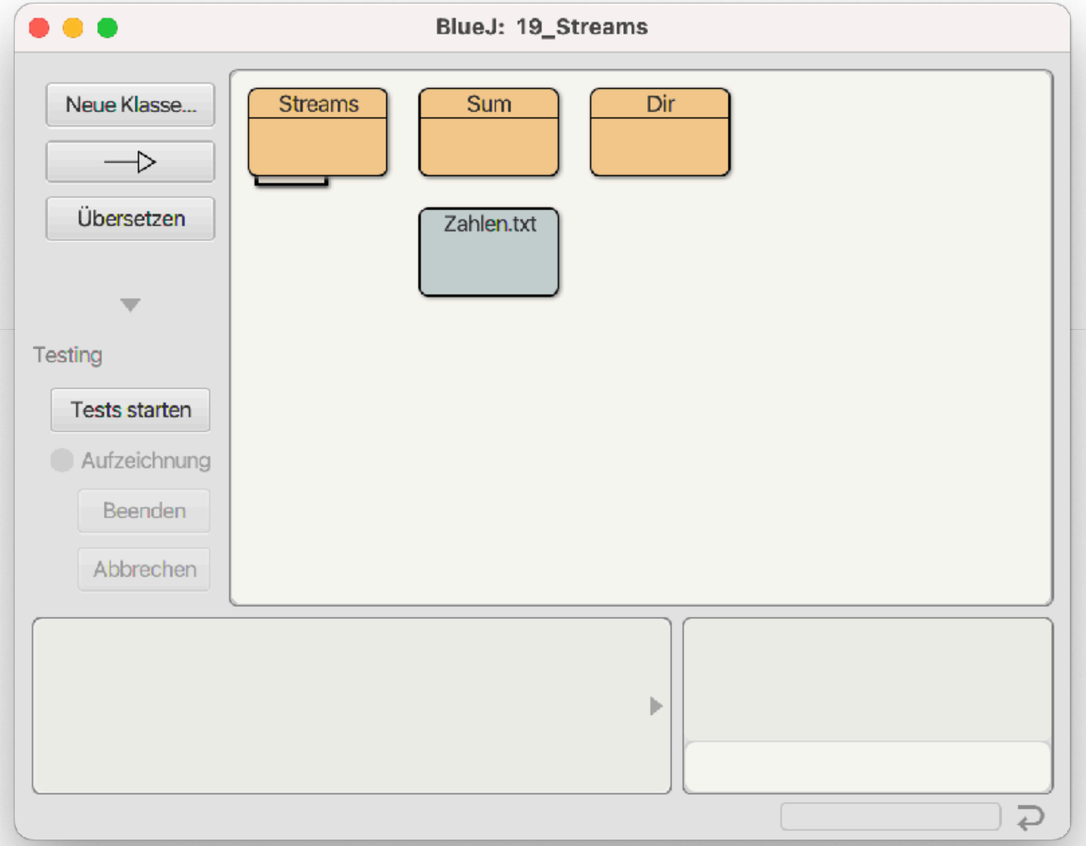
try mit Ressourcen

- **try**-**catch** wird häufig mit externen Ressourcen verwendet
- Diese sollten wieder freigegeben werden, egal, ob Exceptions auftraten oder nicht
- Behandlung in **finally** aufwändig, da Freigabe auch wieder Exceptions erzeugen kann
- **try (***Anforderung von* **AutoClosable***s***) {…}**
  - In den runden Klammern angeforderte Ressourcen werden auf jeden Fall wieder freigegeben (Mehrere durch **;** trennen)

```
static int length(final String name)
{
    try (final FileInputStream stream
            = new FileInputStream(name)) {
        int chars = 0;
        while (stream.read() != -1) {
            ++chars;
        }
        return chars;
    }
    catch (final IOException e) {
        return -1;
    }
}
```

Universität Bremen

Scanner: Demo



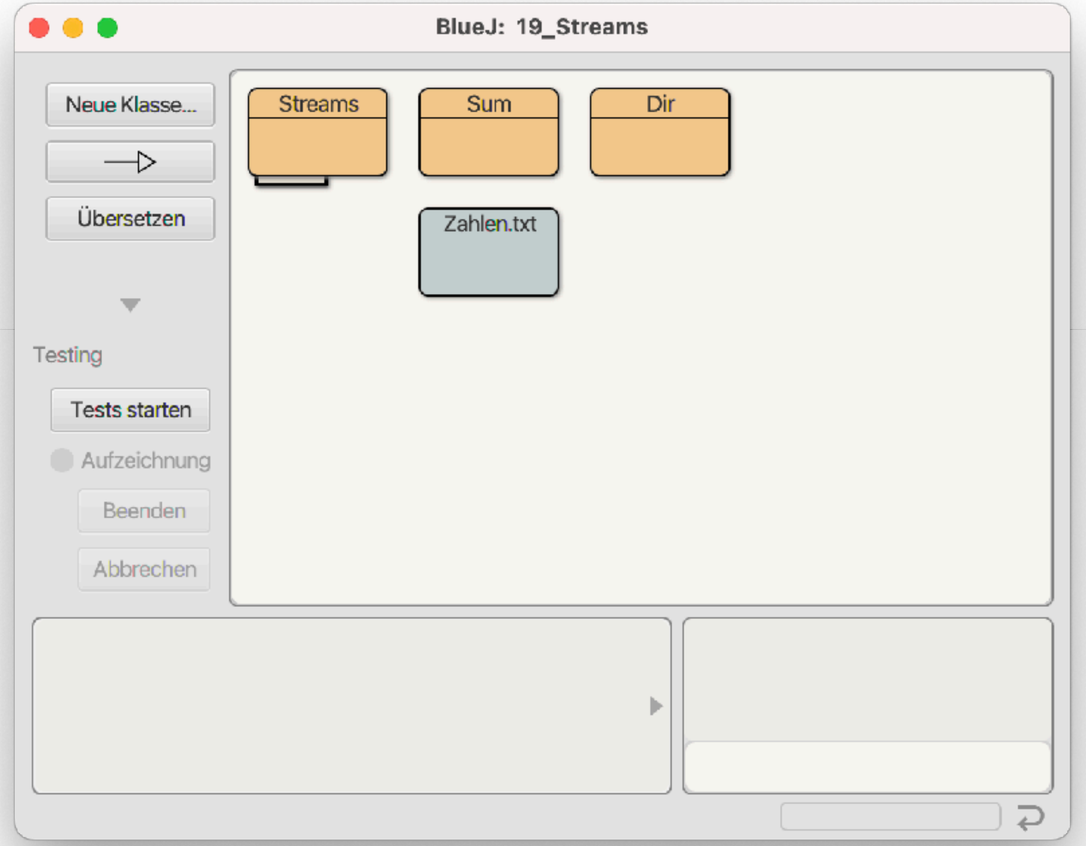
Universität Bremen

Scanner

- Bietet Methoden, um Werte von primitiven Datentypen zu lesen
- Lesen eines Werts eines bestimmten Typs: **next*Type*()**, z.B **nextInt()**
  - **InputMismatchException**, wenn Datentyp falsch
  - **NoSuchElementException**, wenn Eingabestrom zu Ende
- Kommt als nächstes ein Wert eines bestimmten Datentyps? **hasNext*Type*()**
- Kommt überhaupt noch etwas? **hasNext()**
- Lesen bis zum Zeilenende: **nextLine()**

Universität Bremen

Inhaltsverzeichnis anzeigen: Demo



Universität Bremen

Die Klasse **File**

- Liefert Informationen über ein(e) Datei/Verzeichnis (muss nicht existieren)
- **exists()**: Existiert die Datei/das Verzeichnis überhaupt?
- **getName()**: Pfadloser Name der Datei/des Verzeichnisses
  getAbsolutePath(): Absoluter Pfad
  getCanonicalPath(): Eindeutiger Pfad (z.B. "." und ".." beseitigt)
- **length()**: Länge der Datei
- **delete()**: Löscht die Datei/das Verzeichnis (wenn leer)
- **renameTo(File dest)**: Benennt die Datei um und/oder verschiebt sie

Universität Bremen

Die Klasse **File**: Verzeichnisinformationen

- **isFile()**, **isDirectory()**: Datei oder Verzeichnis?
- **listFiles()**: Array aller enthaltenen Dateien und Unterverzeichnisse
- **mkdir()**: Erzeugt dieses Verzeichnis
  mkdirs(): Erzeugt kompletten Pfad zu diesem Verzeichnis
- **getTotalSpace()**: Platz auf dem Datenträger mit Datei/Verzeichnis?
  getFreeSpace(): Wie viel ist davon noch frei?
- Klassenmethode **listRoots()**: Alle Dateisystemwurzeln des Systems
- Klassenkonstante **separatorChar**: Verzeichnistrenner

Zusammenfassung der Konzepte

- **assertThrows**
- **Datenstrom**
- **try** mit Ressourcen
- **Scanner** und **File**