Ausnahmen

Gilbert Beyer und Annabelle Klarl

Zentralübung zur Vorlesung Einführung in die Informatik

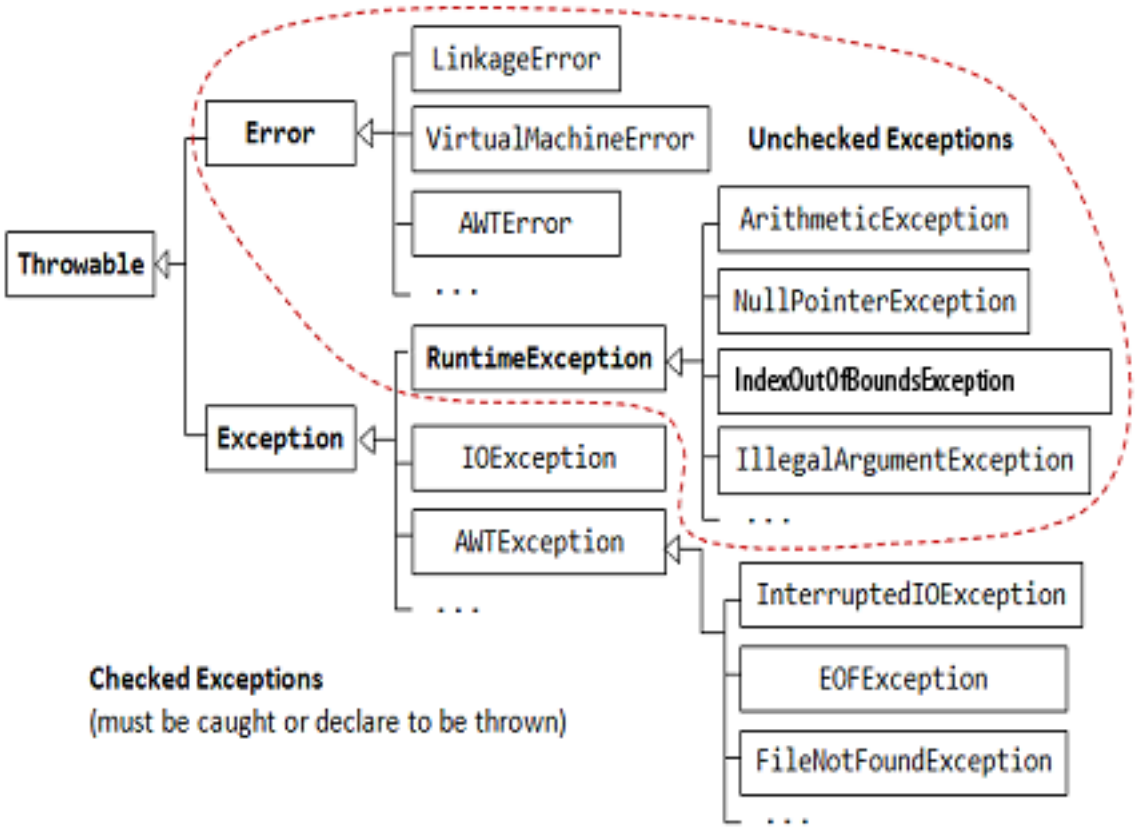
`http://www.pst.ifi.lmu.de/Lehre/wise-11-12/infoeinf`

Inhalte der heutigen Vorlesung:

- Arten von Fehlern
- Auslösen von Ausnahmen
- Vermeiden von Ausnahmen
- Ausnahmeklassen in Java
- Checked Exceptions
- Unchecked Exceptions
- Ausnahmebehandlung
- Benutzerdefinierte Ausnahmen

Fehler- und Ausnahmen in Java

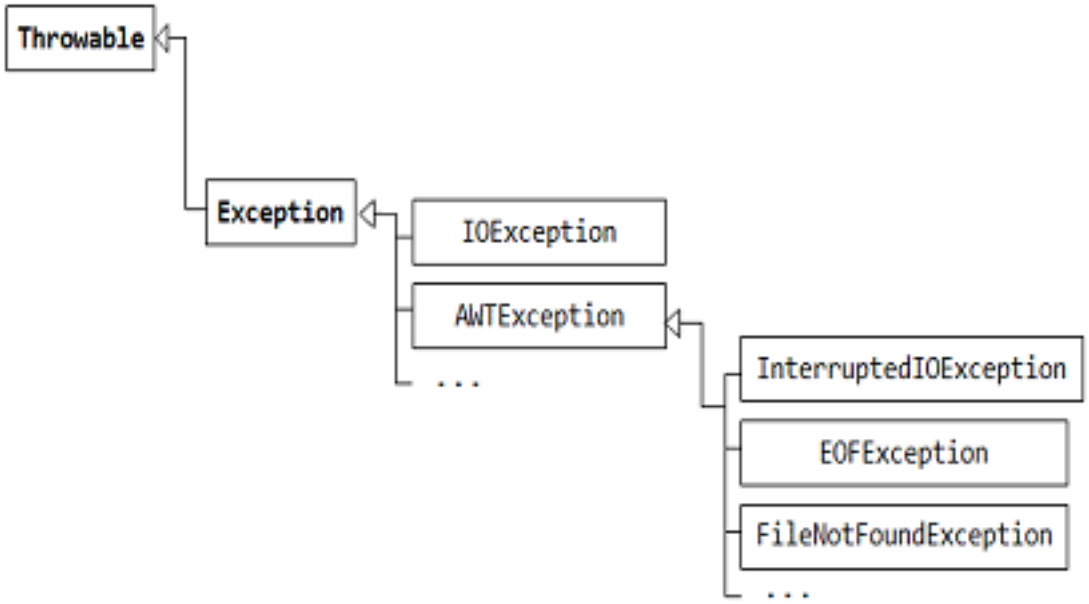
Alle Unterklassen der Klasse **`java.lang.Throwable`**.



Source: http://www.ustudy.in/sites/default/files/Exception-Classes.gif

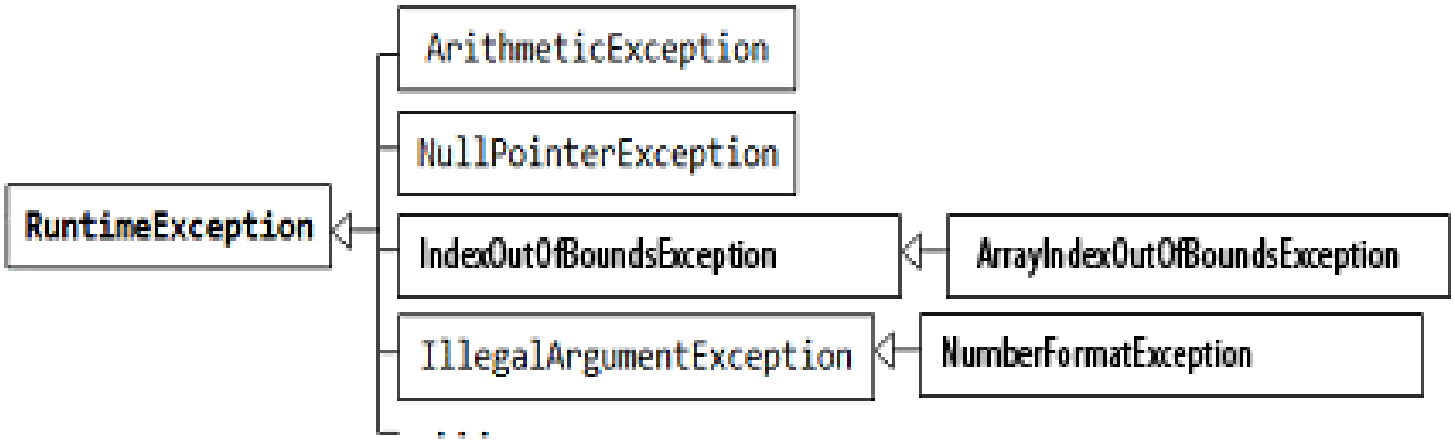
Arten von Ausnahmen: Checked Exceptions

Alle Unterklassen der Klasse **`Exception`** mit Ausnahme aller Unterklassen der Klasse **`RuntimeException`** müssen abgefangen werden.



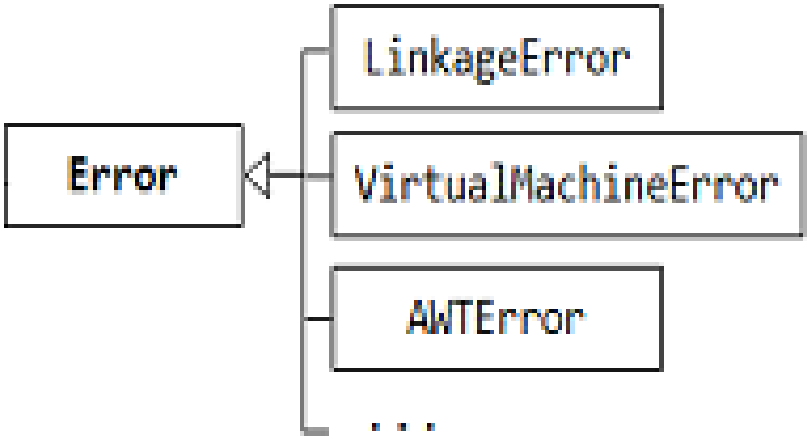
Arten von Ausnahmen: Unchecked Exceptions

Alle Unterklassen der Klasse **`RuntimeException`** müssen nicht abgefangen werden.



Arten von Ausnahmen: Errors

Alle Unterklassen der Klasse **`Error`**.



Aufgabe 1)

Jede der folgenden Situationen hat zur Folge, dass vom Java-Laufzeitsystem ein Objekt der Klasse **`Throwable`** geworfen wird.

1. Geben Sie jeweils an, zu welcher der Kategorien *Error*, *checked Exception*, *uncheckedException* dieses Objekt gehört.

2. Beantworten Sie außerdem jeweils die Fragen: Darf das Java-Programm explizit angeben, was damit geschehen soll? Muss es das explizit angeben? Sollte es das normalerweise explizit angeben, wenn es darf aber nicht muss?

Aufgabe 1a)

Ein Array a wurde ordnungsgemäß erzeugt, danach ist der Ausdruck **`a[n]`** erlaubt, aber der Wert von n ist zum Ausführungszeitpunkt negativ (**`ArrayIndexOutOfBounds`**).

Loesung 1a)

1. Klasse: **`ArrayIndexOutOfBoundsException`**
   Kategorie: **`RuntimeException`** bzw. *unchecked Exception*.
2. Verwendung in **`catch`**- oder **`throws`**-Konstrukten:
   Programm darf, muss aber nicht*

*Empfehlung: Das Programm sollte das normalerweise auch nicht tun, weil es diese Kategorie von Fehler selbst verhindern kann und soll. Wenn es diese Kategorie von Fehler selbst verhindert, kommen solche Exceptions auch nicht vor, deshalb wird es nicht dazu gezwungen, diese Kategorie von Fehler zu behandeln.

Aufgabe 1b)

Ein neues Objekt soll erzeugt werden, aber dafür ist kein Speicherplatz mehr verfügbar in dem es gespeichert werden kann (**`OutOfMemory`**).

Loesung 1b)

1. Klasse: **`OutOfMemoryError`**

   Kategorie: *Error*.

2. Verwendung in **`catch`**- oder **`throws`**-Konstrukten:

   Programm darf, muss aber nicht *

*Empfehlung: Das Programm sollte das normalerweise auch nicht tun.
Das Programm kann diese Kategorie von Fehler zwar nicht selbst verhindern. Wenn diese Kategorie von Fehler aber einmal auftritt, kann das Programm kaum noch sinnvoll reagieren – das Programmstück für die Fehlerbehandlung könnte zum Beispiel selbst Speicher benötigen und damit nicht mehr korrekt ausführbar sein – deshalb wird es nicht dazu gezwungen, diese Kategorie von Fehler zu behandeln.

Aufgabe 1c)

Eine Datei wurde zum Lesen geöffnet und daraufhin überprüft, dass das Dateiende noch nicht erreicht ist. Unmittelbar vor der nächsten Leseoperation wird die Stromversorgung des Geräts unterbrochen, auf dem die Datei gespeichert ist. Die nächste Leseoperation kann damit nichts mehr von der Datei lesen, versucht also rein physikalisch (wenn auch nicht logisch) über das Ende der Datei hinaus zu lesen (**`EOF`**).

Loesung 1c)

1. Klasse: **`EOFException`**

   Kategorie: *checked Exception*.

2. Verwendung in **`catch`**- oder **`throws`**-Konstrukten:

   Das Programm muss dies tun!*

*Begründung: Das Programm kann diese Kategorie von Fehler nicht selbst verhindern. Wenn diese Kategorie von Fehler aber einmal auftritt, kann es durchaus noch sinnvoll darauf reagieren – zum Beispiel alle anderen offenen Dateien in einen konsistenten Zustand bringen und dann beenden –, deshalb wird es vom Compiler dazu gezwungen, zu sagen, ob es diese Kategorie von Fehler selbst behandelt oder weitergibt.

Aufgabe 1d)

Die Klasse Main benutzt eine Methode **`m()`** aus einer Klasse **`K`**. Nachdem beide Dateien fehlerlos übersetzt wurden, wird die Datei **`K.java`** editiert, die Methode **`m()`** umbenannt zu **`n()`**, dann die Datei **`K.java`** neu übersetzt. Danach wird das unveränderte Hauptprogramm wieder gestartet, aber die von ihm verwendete Methode **`m()`** existiert in der Klasse **`K`** gar nicht mehr (**`NoSuchMethod`**, **`IncompatibleClassChange`**).

Aufgabe 1d)

1. Klasse: **`NoSuchMethodError`**

   Kategorie: *Error*.

2. Verwendung in **`catch`**- oder **`throws`**-Konstrukten:

   Programm darf, muss aber nicht *

*Empfehlung: Das Programm sollte das normalerweise auch nicht tun.
Das Programm kann diese Kategorie von Fehler zwar nicht selbst verhindern.
Wenn diese Kategorie von Fehler aber einmal auftritt, kann das Programm kaum noch sinnvoll reagieren – deshalb wird es nicht dazu gezwungen, diese Kategorie von Fehler zu behandeln.

Aufgabe 1e)

Eine Variable **`v`** ist vom Typ **`K`**, der eine Unterklasse von **`Object`** ist, also ist der Ausdruck **`v.toString()`** erlaubt, aber zum Ausführungszeitpunkt hat die Variable **`v`** den Wert **`null`** (**`NullPointer`**).

Aufgabe 1e)

1. Klasse: **`NullPointerException`**

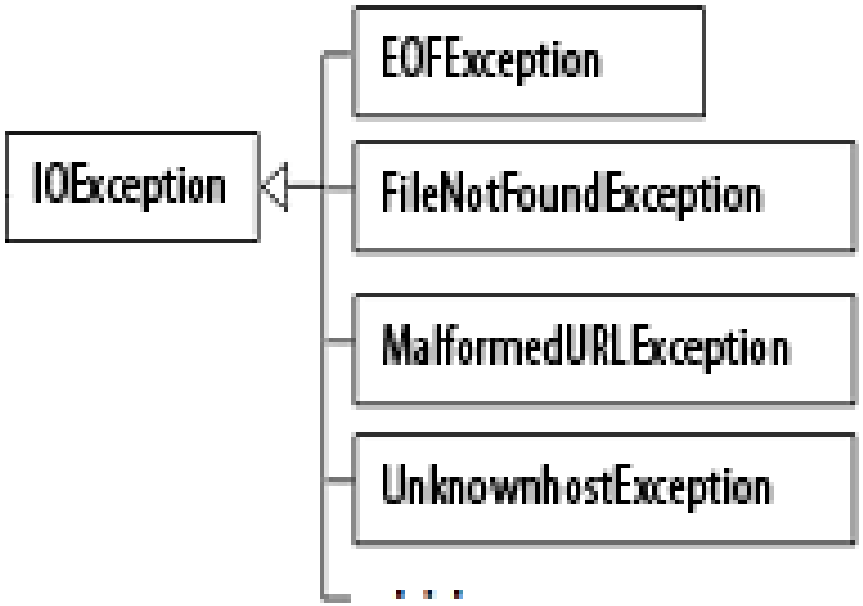
   Kategorie: **`RuntimeException`** bzw. *unchecked Exception*.
2. Verwendung in **`catch`**- oder **`throws`**-Konstrukten:

   Programm darf, muss aber nicht*

*Empfehlung: Das Programm sollte das normalerweise auch nicht tun, weil es diese Kategorie von Fehler selbst verhindern kann und soll. Wenn es diese Kategorie von Fehler selbst verhindert, kommen solche Exceptions auch nicht vor, deshalb wird es nicht dazu gezwungen, diese Kategorie von Fehler zu behandeln.

IOExceptions

Beispiele:

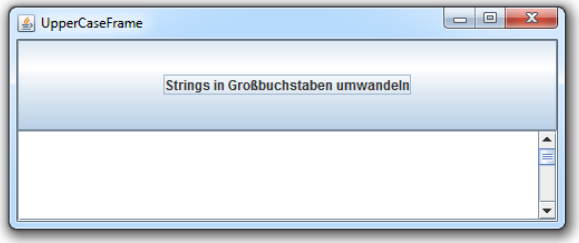


Eingabe und Ausgabe bisher

(1) GUIs mit AWT und Swing

Eingabe über JOptionPane-Dialog

Ausgabe über eine JTextArea



(2) Standardausgabe:

Ausgabe über einen Printstream

```
System.out.println("Ergebnis: " + result);
```

Standardein- und Ausgabe in Java

In jedem Java-Programm erzeugt der Compiler automatisch eine Instanz der Klasse System mit Attributen *in* und *out*.

(1) Standardausgabe:

System.*out* (Ausgabestrom vom Typ: PrintStream)

(2) Standardeingabe

System.*in* (Eingabestrom vom Typ: InputStream)

Ein- und Ausgabe über Streams

Zwei Arten von Stream-Klassen :

(1) Byteorientierte Ströme/ByteStreams

- Lesen/Schreiben jeweils nur **ein Byte**
- Basisklassen: java.io.InputStream, java.io.OutputStream
- Beispiele: InputStream, FileInputStream, FileOutputStream, …

(2) Zeichenorientierete Ströme/CharacterStreams

- Lesen/Schreiben **zwei Byte** oder ein **char**
- Basisklassen: java.io.Reader, java.io.Writer
- Beispiele: BufferedReader, FileReader, FileWriter, …

Komfortable Ein- und Ausgabe

Beispiel Einlesen:

(1) *InputsStreams* haben eine Methode **`int read()`**
zum Einlesen des nächsten 1 Byte

(2) *Reader* haben eine Methode **`int read()`**
zum Einlesen der nächsten 2 Byte (**`char`** muss man casten)

(3) *BufferedReader* hat eine Methode **`String readline()`**
zum zeilenweisen Einlesen von Strings

Komfortable Ein- und Ausgabe

```
import java.io.*;

public class Eingabe {
  public static void main(String[] args) throws IOException {
    InputStreamReader reader = InputStreamReader(System.in));
    BufferedReader console = new BufferedReader(reader);
    System.out.println("Hallo, mein Name ist HAL!");
    System.out.println("Wie lautet Dein Name?");
    String name = console.readLine();
    System.out.println("Hallo, " + name + ". "
        + "Freut mich, Dich kennenzulernen.");
  }
}
```

Abfangen von IO-Exceptions

```
import java.io.*;

public class Eingabe {
  public static void main(String[] args) throws IOException {
    BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in));
    System.out.println("Hallo, mein Name ist HAL!");
    System.out.println("Wie lautet Dein Name?");
    String name = in.readLine();
    System.out.println("Hallo, " + name + ". "
        + "Freut mich, Dich kennenzulernen.");
  }
}
```

Behandeln mit try- und catch- Block

```
public static void main(String[] args) {
    BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in));
     System.out.println("Wie lautet Dein Name?");
     String name = "";
     try {
       name = in.readLine();
     } catch (IOException e) {
      System.out.println("Fehler beim Einlesen: "
          + e.getMessage());
      System.out.println("Programm wird beendet!");
      System.exit(1); // abnormal termination
     }  System.out.println("Hallo, " + name + ".");
  }
}
```

Behandeln mit try- und catch- Block

```
public static String readString() {
    BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in));
    while(true) {
       try {
         return in.readLine();
       } catch (IOException e) {
           System.out.println("Fehler beim Einlesen: "
             + e.getMessage());
           System.out.println("Versuchen Sie es nochmal!");
       }
     }
```

...

Einlesen von Zahlen

```
public class ZahlEinlesen {
  public static void main(String[] args) {
    BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in));
    System.out.println("Zahl eingeben: ");
    String s = "";
    try {
      s = in.readLine();
      int zahl = Integer.parseInt(s);
    } catch (NumberFormatException e) {
      System.out.println("Keine Zahl!");
    } catch (IOException e) {
      System.out.println("Fehler!");
    }
```

...

Programm das Dateien liest und schreibt

```
public static void aneinanderhaengen(String[] dateinamen) {
 BufferedOutputStream out = new BufferedOutputStream(System.out)
    int data;
    for (int n = 0; n < dateinamen.length; n++) {
      try {BufferedInputStream in = new BufferedInputStream(
            new FileInputStream(dateinamen[n]));
        while ((data = in.read()) != -1) {//EOF Test
          out.write(data);
        } out.flush(); //Leeren des Puffers
      } catch (FileNotFoundException exception) {
        System.err.println("Die Datei " + dateinamen[n]
            + " wurde nicht gefunden.");
      } catch (IOException exception) {
        System.err.println("Fehler beim Lesen/Schreiben.");
      }
```