



# Ausnahmen

---

Dr. Annabelle Klarl

Zentralübung zur Vorlesung

„Einführung in die Informatik: Programmierung und Softwareentwicklung“

<http://www.pst.ifi.lmu.de/Lehre/wise-16-17/infoeinf>



Action required now



1. Smartphone: installiere die App "socrative student" **oder**  
Laptop: öffne im Browser [b.socrative.com/login/student](https://b.socrative.com/login/student)
2. Betrete den Raum **InfoEinf.**
3. Beantworte die erste Frage sofort!



## Fehlerarten

Bei der Programmierung können viele Arten von Fehlern auftreten:

- Syntaktische Fehler

z.B. `String name = "annabelle;`  
-> vom **Compiler** erkannt

- Logische Fehler

z.B. `System.out.println("Hllo Wlt!");`  
-> **nicht** automatisch erkennbar

- Fehler bei der Umsetzung

- Ungenügender Umgang mit außergewöhnlichen Situationen

meist angezeigt durch  
**Exceptions oder Errors**  
**(Throwables)**

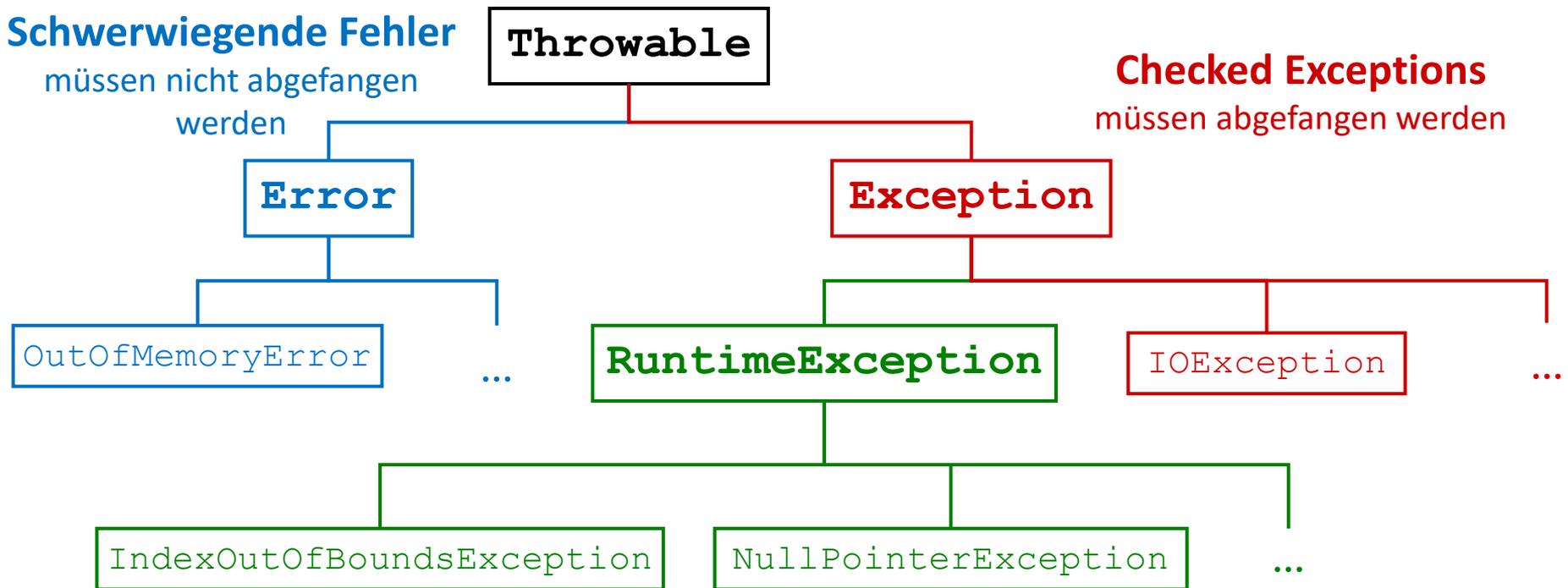


# Fehler und Ausnahmen in Java

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

## Schwerwiegende Fehler

müssen nicht abgefangen  
werden



## Checked Exceptions

müssen abgefangen werden

## Unchecked Exceptions

müssen nicht abgefangen werden



## Aufgabe

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*, *Unchecked Exception* dieses Objekt gehört.
  
2. Beantworten Sie außerdem jeweils die Fragen:
  - a. **Darf** im Java-Programm explizit angegeben werden, wie auf diese Fehlersituation reagiert werden soll?  
(Verwendung eines `catch`-Blocks oder einer `throws`-Deklaration)
  - b. **Muss** es explizit angegeben werden?

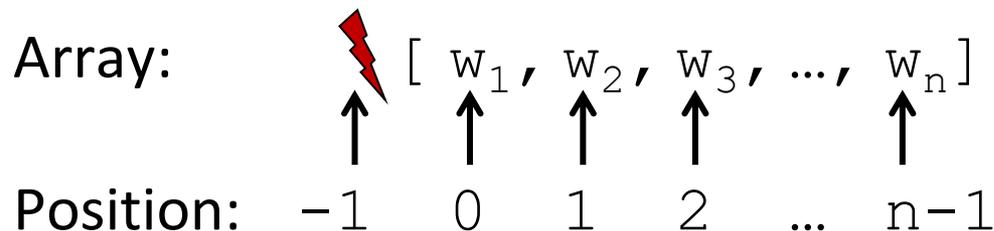


## Aufgabe a.

Ein Array `array` wurde ordnungsgemäß erzeugt. Nach der Erzeugung wird die folgende Deklaration ausgeführt:

```
int a = array[-1];
```

Das Programm kompiliert korrekt, bei der Ausführung tritt allerdings die Exception `ArrayIndexOutOfBoundsException` auf.





## Lösung a.

1. Klasse: `ArrayIndexOutOfBoundsException`  
Kategorie: `RuntimeException` bzw. *unchecked Exception*



2. Wie darf/muss das Java-Programm reagieren?

- a) Das Java-Programm **darf** diese Ausnahme **nicht** abfangen.
- b) Das Java-Programm **darf** diese Ausnahme abfangen, **muss** sie aber **nicht** abfangen.
- c) Das Java-Programm **darf** diese Ausnahme abfangen und **muss** sie auch abfangen.

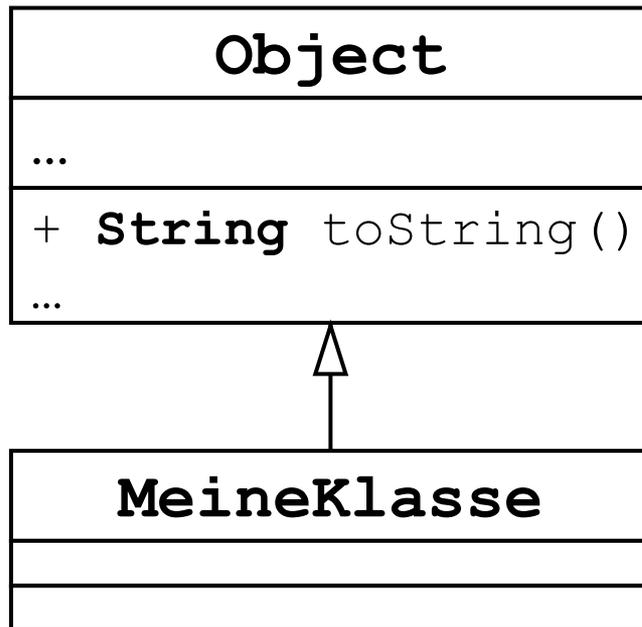
## Bemerkung:

Diese Kategorie von Fehlern kann bei korrekter Implementierung vermieden werden. Deshalb wird diese Kategorie normalerweise nicht behandelt.



## Aufgabe b.

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



### Beispielcode:

```
MeineKlasse v = null;
v.toString();
```



## Lösung b.



### 1. Wie heißt die geworfene ungeprüfte Ausnahme?

- a) `UnsupportedOperationException`
- b) `IllegalArgumentException`
- c) `NullPointerException`
- d) `IllegalStateException`

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

- a. Das Java-Programm **darf** diese Ausnahme abfangen.
- b. Das Java-Programm **muss** diese Ausnahme **nicht** abfangen.

### Bemerkung:

Diese Kategorie von Fehlern kann bei korrekter Implementierung vermieden werden. Deshalb wird diese Kategorie normalerweise nicht behandelt.



## Aufgabe c.

Ein neues Objekt soll erzeugt werden, aber es ist kein Speicherplatz mehr verfügbar, in dem es gespeichert werden kann.



## Lösung c.

1. Klasse: `OutOfMemoryError`

Kategorie: *Error*



2. Wie darf/muss das Java-Programm reagieren?

- a) Das Java-Programm **darf** diese Ausnahme nicht abfangen.
- b) Das Java-Programm **darf** diese Ausnahme abfangen, **muss** sie aber **nicht** abfangen.
- c) Das Java-Programm **darf** diese Ausnahme abfangen und **muss** sie auch abfangen.

### Bemerkung:

Wenn diese Kategorie von Fehler einmal auftritt, kann das Programm kaum noch sinnvoll reagieren. Deshalb wird diese Kategorie normalerweise nicht behandelt.



## Aufgabe d.

Eine Datei wurde zum Lesen geöffnet und daraufhin überprüft, dass das Dateiende noch nicht erreicht ist.

- Unmittelbar vor der ersten 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.



## Lösung d.

1. Klasse: `EOFException`  
Kategorie: *checked Exception*



### 2. Wie darf/muss das Java-Programm reagieren?

- a) Das Java-Programm **darf** diese Ausnahme nicht abfangen.
- b) Das Java-Programm **darf** diese Ausnahme abfangen, **muss** sie aber **nicht** abfangen.
- c) Das Java-Programm **darf** diese Ausnahme abfangen und **muss** sie auch abfangen.



## Lösung d. (Fortsetzung)

1. Klasse: `EOFException`  
Kategorie: *checked Exception*
2. Verwendung in **catch**- oder **throws**-Konstrukten:
  - a. Das Java-Programm **darf** diese Ausnahme abfangen.
  - b. Das Java-Programm **muss** diese Ausnahme abfangen!!!

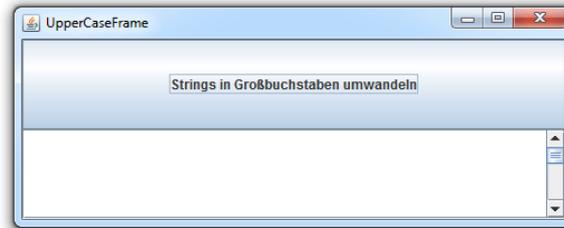
### Bemerkung:

Wenn diese Kategorie von Fehlern auftritt, kann das Programm 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, diese Kategorie von Fehlern selbst zu behandeln (**catch**-Block) oder weiterzugeben (**throws**-Deklaration).



## Eingabe- und Ausgabemöglichkeiten

1. GUIs mit AWT und Swing:  
Eingabe über `JOptionPane`-Dialog  
Ausgabe über `JTextArea`



2. Standardeingabe und -ausgabe über die Konsole:  
Eingabe über einen `InputStream`  
Ausgabe über einen `PrintStream`

```
System.out.println("Hallo Welt");
```



## Standardeingabe und -ausgabe in Java

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

### 1. Standardeingabe von der Konsole:

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



???

### 2. Standardausgabe auf der Konsole:

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

```
System.out.println("Hallo Welt");
```



# Streams lesen (und schreiben)

Zwei Arten von Stream-Klassen:

## 1. Byteorientierte Ströme (= Byte-Streams)

- Lesen/Schreiben von jeweils **einem Byte**
- Basisklassen: `java.io.InputStream/java.io.OutputStream`
- Beispiele: `FileInputStream/FileOutputStream, ...`

## 2. Zeichenorientierte Ströme (= Character-Streams)

- Lesen/Schreiben von **zwei Bytes** oder einem **char**
- Basisklassen: `java.io.Reader/java.io.Writer`
- Beispiele: `InputStreamReader/OutputStreamWriter,`  
`BufferedReader/BufferedWriter,`  
`FileReader/FileWriter, ...`

Effizientes Lesen  
mit `readLine()`



## Standardeingabe und -ausgabe in Java: Beispiel

```
import java.io.*;

public class Eingabe {
    public static void main(String[] args) throws IOException{
        System.out.println("Wie lautet dein Name?");

        BufferedReader konsole =
            new BufferedReader(new InputStreamReader(System.in));
        String name = konsole.readLine();

        System.out.println("Hallo " + name + ". ");
    }
}
```

**Fehler weitergeben** ⚡

Eingabe über die Konsole

Ausgabe über die Konsole



## Standardeingabe und -ausgabe in Java: Beispiel verbessert

```
import java.io.*;
public class Eingabe_verbessert {
    public static void main(String[] args) {
        System.out.println("Wie lautet dein Name?");
        try {
            BufferedReader konsole =
                new BufferedReader(new InputStreamReader(System.in));
            String name = konsole.readLine();
            System.out.println("Hallo " + name + ".");
        } catch (IOException e) {
            System.out.println("Fehler:" + e.getMessage());
            System.exit(1); // abnormal termination
        }
    }
}
```

Fehler abfangen



# Standardeingabe und -ausgabe in Java: Beispiel verbessert

```
import java.io.*;

public class Eingabe_verbessert2 {
    public static void main(String[] args) {
        System.out.println("Wie lautet dein Name?");
        String name = stringEinlesen();
        System.out.println("Hallo " + name + ".");
    }

    public static String stringEinlesen() {
        BufferedReader konsole =
            new BufferedReader(new InputStreamReader(System.in));
        while (true) {
            try { return konsole.readLine(); }
            catch (IOException e) { System.out.println("Nochmal!"); }
        }
    }
}
```