

Objektorientierte Programmierung in Java

Vorlesung 8 - Ausnahmebehandlung

Emily Lucia Antosch

HAW Hamburg

30.06.2025

Inhaltsverzeichnis

1. Einleitung	2
2. Exception Handling	6
3. Ausnahme werfen	13
4. Ausnahmen fangen	18
5. Eigene Ausnahmen definieren	34
6. License Notice	42

1. Einleitung

- In der letzten Vorlesung haben wir uns mit dem Erstellen von graphischen Oberflächen beschäftigt
- Sie können nun
 - Fenster erzeugen, in dem andere Elemente leben können,
 - Elemente mittels Layouts und Panels arrangieren
 - und Grafiken direkt in Java erzeugen.
- Heute geht es weiter mit den **Ausnahmebehandlungen**.

1.1 Wo sind wir gerade?

1. Imperative Konzepte
2. Klassen und Objekte
3. Klassenbibliothek
4. Vererbung
5. Schnittstellen
6. Graphische Oberflächen
7. **Ausnahmebehandlung**
8. Eingaben und Ausgaben
9. Multithreading (Parallel Computing)

- Sie behandeln bei Programmausführung auftretende Ausnahmen und Fehler, um in aufgetretenen Ausnahmesituationen einen geordneten Programmfluss herzustellen.
- Sie definieren eigene, an die Bedürfnisse Ihrer konkreten Anwendung angepasste, Ausnahmeklassen.

2. Exception Handling

2.1 Introductory Example

2. Exception Handling

? Frage

- Was wird von folgendem Programm ausgegeben?

```
1 public class ProvokeException {
2     public static void main(String[] args) {
3         int a = 3;
4         int b = 2;
5         printRatio(a, b);
6         System.out.println("Exiting main()");
7     }
8
9     public static void printRatio(int a, int b) {
10        int ratio = a / b;
11        System.out.println("Ratio = " + ratio);
12    }
13 }
```

 Java

? Frage

- Und was wird für a = 7 und b = 0 ausgegeben?

? Frage

- Was kann in einem Programm alles „schief gehen“?
- Wann muss der normale Programmfluss unterbrochen werden?
- Wann muss ein Programm beendet werden, wann kann es fortgeführt werden?



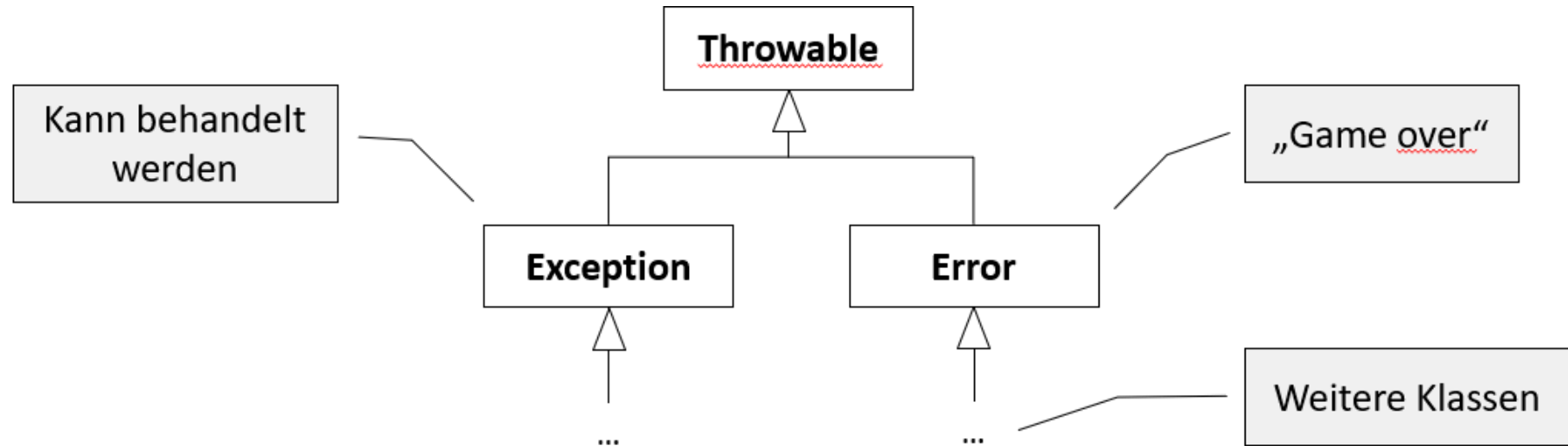
Beispiel

- Division durch Null
- Aufruf `a.method()`, obwohl Variable `a` den Wert `null` hat
- Negativer oder zu hoher Index für Arrays
- Wandeln der Zeichenkette „Dies ist Text“ in eine Ganzzahl vom Typ `int`
- Datei nicht gefunden
- Kein Speicher mehr verfügbar

2.1 Introductory Example

2. Exception Handling

- Ausnahmen und Fehler durch Objekte spezieller Klassen dargestellt
- Basisklasse aller Ausnahmeklassen ist Throwable

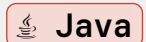


- Man unterscheide:
 - Exception (Ausnahme): Behandelbar, Programm kann fortgeführt werden
 - Error bzw. fatal error (Fehler): Nicht behandelbar, Programm beenden

! Merke

- Ausnahme wird auch als Oberbegriff für Ausnahmen und Fehler verwendet.
 - Ausnahmebehandlung wird auch als Exception handling bezeichnet.
-
- Einige Klassen für Ausnahmen:
 - Division durch Null (ArithmeticException)
 - Zugriff auf Methode oder Attribut über null-Referenz (NullPointerException)
 - Unzulässiger Feldindex (ArrayIndexOutOfBoundsException)
 - Unzulässige Zeichen beim Lesen einer Zahl (NumberFormatException)
 - Datei nicht gefunden (FileNotFoundException)

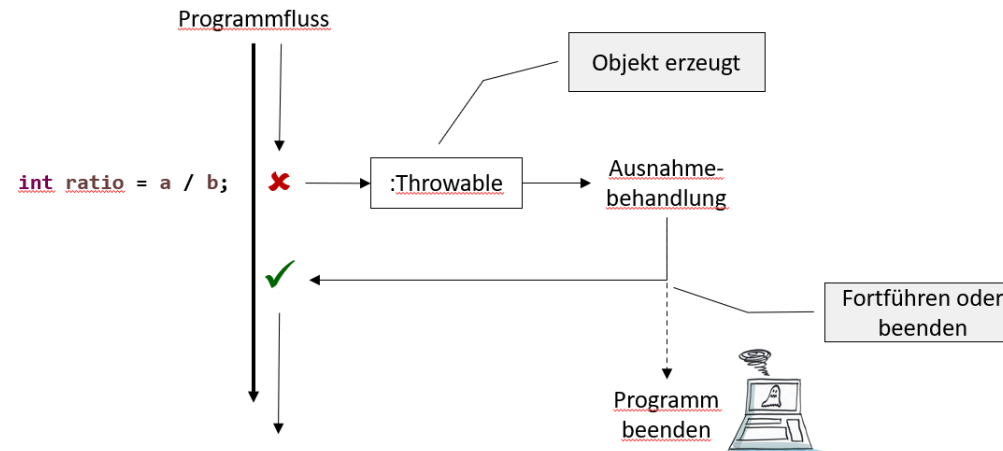
```
1 int[] array = {1, 2, 3, 4};
2 System.out.println(array[4]);
3
4 String message;
5 System.out.println(message.length());
6
7 int code = Integer.parseInt("12a4");
```



2.2 Ablauf der Ausnahmebehandlung

2. Exception Handling

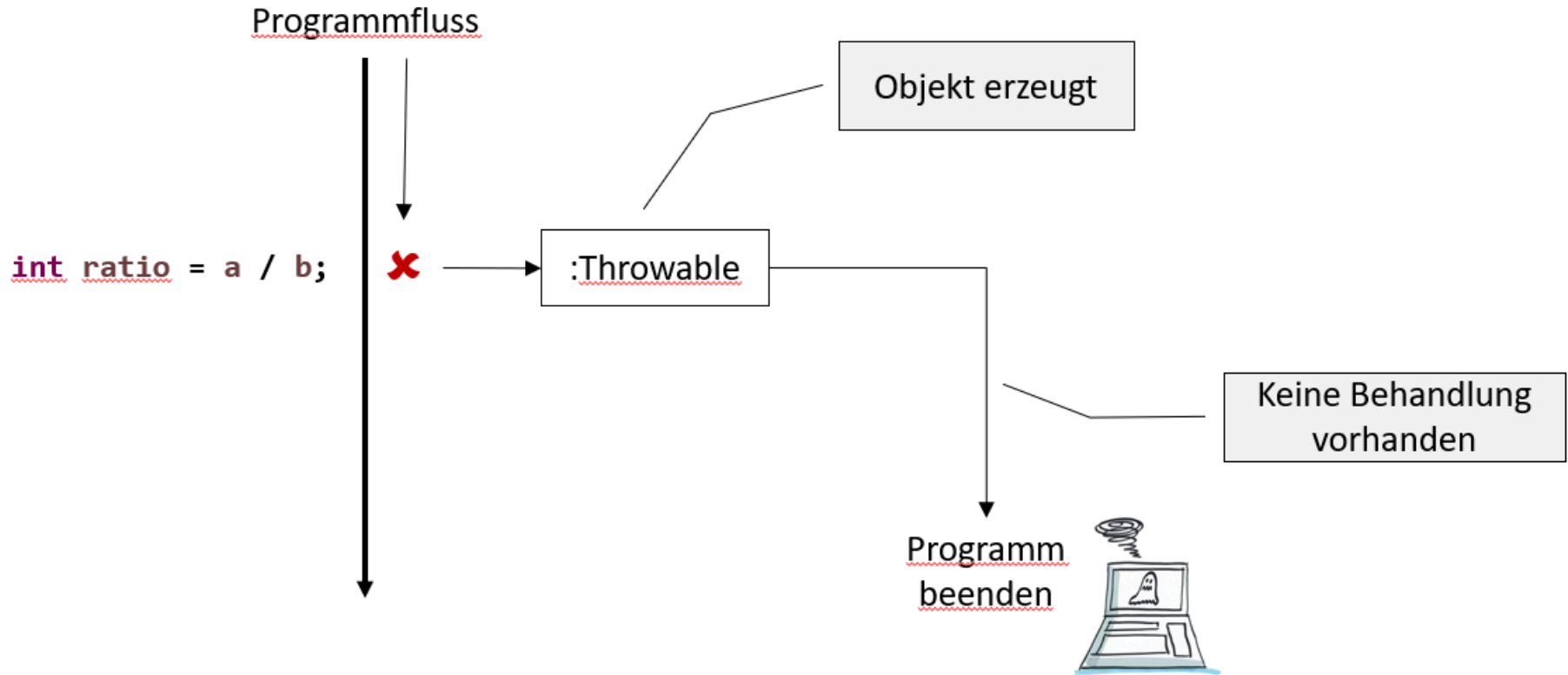
1. Ausnahme werfen:
 - Programmfluss wird unmittelbar unterbrochen
 - Objekt erzeugt, das Ausnahme repräsentiert
2. Ausnahme fangen:
 - Programmierer kann Ausnahme abfangen und behandeln



2.2 Ablauf der Ausnahmebehandlung

2. Exception Handling

- Falls keine Ausnahmebehandlung programmiert: Programm wird beendet



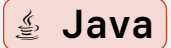
3. Ausnahme werfen

3.1 Ausnahme werfen

3. Ausnahme werfen

- Im Fehlerfall werden Ausnahmen automatisch erzeugt (z.B. Division durch Null).
- Ausnahmen lassen sich aber auch explizit werfen.

```
1 throw ExceptionObject;
```

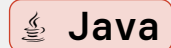


- Konstruktor kann String (z.B. als Fehlerbeschreibung) übergeben werden



Beispiel

```
1 throw new Exception();  
2 throw new Exception("Division by zero");  
3 Exception exception = new Exception(); throw exception;
```

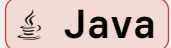


3.1 Ausnahme werfen

3. Ausnahme werfen

- Zur Veranschaulichung:
 - Werfen Sie eine Ausnahme, bevor versucht wird, durch Null zu teilen.

```
1  public class ThrowException {
2      public static void main(String[] args) {
3          int a = 3;
4          int b = 0;
5          printRatio(a, b);
6          System.out.println("Exiting main()");
7      }
8
9      public static void printRatio(int a, int b) {
10         int ratio = a / b;
11         System.out.println("Ratio = " + ratio);
12     }
13 }
```



3.1 Ausnahme werfen

3. Ausnahme werfen

- Beispiellösung:

```
1  public class ThrowException {
2      public static void main(String[] args) {
3          int a = 3;
4          int b = 0;
5          printRatio(a, b);
6          System.out.println("Exiting main()");
7      }
8
9      public static void printRatio(int a, int b) {
10         if (b == 0) {
11             throw new ArithmeticException("Division by zero");
12         }
13         System.out.println("Ratio = " + (a / b));
14     }
15 }
```



- Ausgabe im Fehlerfall:
 - Ausnahmetyp (z.B. `ArithmeticException`)
 - Fehlermeldung (z.B. „Division by zero“)
 - Stacktrace (d.h. Kette der aufgerufenen Methoden)



Beispiel

```
1 Exception in thread "main" java.lang.ArithmeticException: Division by zero at
2     kapitel8_exceptions.ThrowException.printRatio(E02_ThrowException.java:20)
   at
3     kapitel8_exceptions.ThrowException.main(E02_ThrowException.java:14)
```

- Methode `main()` hat in Zeile 14 `printRatio()` aufgerufen
- Methode `printRatio()` hat in Zeile 20 die Ausnahme geworfen

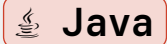
4. Ausnahmen fangen

4.1 Exception Handling

4. Ausnahmen fangen

- Ausnahmen lassen sich fangen und behandeln:

```
1  try {  
2      // Aweisungen ...  
3  } catch (ExceptionType e) {  
4      // Aweisungen ...  
5  }
```



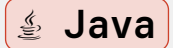
- Try-Block enthält Code, der Ausnahme werfen kann
- Falls Ausnahme im try-Block geworfen wird:
 1. Try-Block unmittelbar beendet
 2. Catch-Block ausgeführt, sofern Ausnahmetyp (ExceptionType) passt
 3. Programm läuft nach catch-Block weiter
- Ausnahmetyp des catch-Blocks passt nicht: Ausnahme wird nicht gefangen!
- Keine Ausnahme geworfen: Catch-Block wird übersprungen

4.1 Exception Handling

4. Ausnahmen fangen

- Vermeiden Sie den „Absturz“:
 - Fangen Sie die geworfene Ausnahme!

```
1  public class TryCatch {
2      public static void main(String[] args) {
3          int a = 3;
4          int b = 0;
5          printRatio(a, b);
6          System.out.println("Exiting main()");
7      }
8
9      public static void printRatio(int a, int b) {
10         int ratio = a / b;
11         System.out.println("Ratio = " + ratio);
12     }
13 }
```

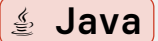


4.1 Exception Handling

4. Ausnahmen fangen

- Bespiellösung:

```
1  public static void printRatio(int a, int b) {  
2      try {  
3          int ratio = a / b;  
4          System.out.println("Ratio = " + ratio);  
5      } catch (ArithmeticException e) {  
6          System.out.println("Exception caught in printRatio()");  
7          System.out.println("e.getMessage(): " + e.getMessage());  
8          System.out.println("e.toString(): " + e + "\n");  
9      }  
10     System.out.println("Exiting printRatio()");  
11 }
```




- Ausgewählte Methoden für Ausnahmeobjekte:
 - getMessage()
 - printStackTrace()
 - toString()

? Frage

Und nun?


```
1 public class TryCatchChain1 {
2     public static void main(String[] args) {
3         int ratio = getRatio(3, 0);
4         System.out.println("Ratio = " + ratio);
5         System.out.println("Exiting main()");
6     }
7     public static int getRatio(int a, int b) {
8         int ratio = 0;
9         try {
10             ratio = a / b;
11         } catch (ArithmeticException e) {
12             System.out.println("Exception caught in getRatio()");
13         }
14         System.out.println("Exiting getRatio()");
15         return ratio;
16     }
17 }
```

 Java

? Frage

Und nun?

```
1 public class TryCatchChain2 {
2     public static void main(String[] args) {
3         try {
4             int ratio = getRatio(3, 0);
5             System.out.println("Ratio = " + ratio);
6         } catch (ArithmeticException e) {
7             System.out.println("Exception caught in main()");
8         }
9         System.out.println("Exiting main()");
10    }
11
12    public static int getRatio(int a, int b) {
13        int ratio = a / b;
14        System.out.println("Exiting getRatio()");
15        return ratio;
16    }
17 }
```

 Java

4.1 Exception Handling

4. Ausnahmen fangen

- Können mehrere Ausnahmearten auftreten, werden mehrere catch-Blöcke benötigt.
- Ausnahmetypen der catch-Blöcke müssen sich unterscheiden
- Es wird der erste passende catch-Block ausgeführt.

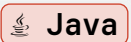
```
1  try {  
2      // ...  
3  } catch (ExceptionTyp1 e) {  
4      // ...  
5  } catch (ExceptionTyp2 e) {  
6      // ...  
7  } catch (ExceptionTyp3 e) {  
8      // ...  
9  }
```



? Frage

- Folgender Quelltext enthält zwei Fehlerquellen. Welche?
- Welche Ausgabe erzeugt das Programm?

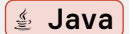
```
1  public class ExceptionTypes1 {
2      static int recursiveIncrease(int i) {
3          return recursiveIncrease(i + 1);
4      }
5
6      public static void main(String[] args) {
7          int[] a = new int[4];
8          try {
9              a[4] = recursiveIncrease(7);
10         } catch (ArrayIndexOutOfBoundsException e) {
11             System.out.println("Caught ArrayIndexOutOfBoundsException");
12         }
13         System.out.println("Exiting main()");
14     }
15 }
```



☰ Aufgabe 1

- Ändern Sie den vorherigen Quelltext derart, dass beide Fehlerquellen gefangen werden.

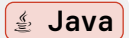
```
1 public class ExceptionTypes2 {
2     static int recursiveIncrease(int i) {
3         return recursiveIncrease(i + 1);
4     }
5
6     public static void main(String[] args) {
7         int[] a = new int[4];
8         try {
9             a[4] = recursiveIncrease(7);
10        } catch (ArrayIndexOutOfBoundsException e) {
11            System.out.println("Caught ArrayIndexOutOfBoundsException");
12        } catch (StackOverflowError e) {
13            System.out.println("Caught StackOverflowError");
14        }
15        System.out.println("Exiting main()");
16    }
17 }
```



? Frage

- Hoppla, unten läuft etwas im catch-Block schief!
- Wird die erneute Ausnahme behandelt? Was wird ausgegeben?

```
1 public class ExceptionTypes3 {
2     static int recursiveIncrease(int i) {
3         return recursiveIncrease(i + 1);
4     }
5     public static void main(String[] args) {
6         int[] a = new int[4];
7         try {
8             a[4] = 0;
9         } catch (ArrayIndexOutOfBoundsException e) {
10             recursiveIncrease(7);
11         } catch (StackOverflowError e) {
12             System.out.println("Caught StackOverflowError");
13         }
14         System.out.println("Exiting main()");
15     }
16 }
```



- Ein catch-Block bezieht sich nur auf den zugehörigen try-Block.
- Wirft catch-Block Ausnahme, wird diese nicht durch nachfolgende Blöcke gefangen

? Frage

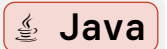
- Wie können wir die im catch-Block erzeugte Ausnahme fangen?

4.1 Exception Handling

4. Ausnahmen fangen

- Quelltext, der Ausnahme erzeugt, in geschachteltem try-Block

```
1  public static void main(String[] args) {
2      int[] a = new int[4];
3      try {
4          a[4] = 0;
5      } catch (ArrayIndexOutOfBoundsException e1) {
6          try {
7              recursiveIncrease(7);
8          } catch (StackOverflowError e2) {
9              System.out.println("Caught inner StackOverflowError");
10         }
11     } catch (StackOverflowError e) {
12         System.out.println("Caught outer StackOverflowError");
13     }
14     System.out.println("Exiting main()");
15 }
```

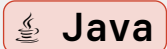


4.1 Exception Handling

4. Ausnahmen fangen

- Mitunter muss bestimmter Code auf jeden Fall ausgeführt werden.
- Beispiel: Schließen geöffneter Dateien oder Datenströme
- Optionaler finally-Block:
 - Steht immer als letztes (d.h. nach try- und catch-Blöcken)
 - Code wird am Ende des Konstruktes ausgeführt ... wirklich immer ... ganz ehrlich!

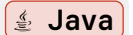
```
1  try {  
2      // ...  
3  } catch (ExceptionTyp1 e) {  
4      // ...  
5  } catch (ExceptionTyp2 e) {  
6      // ...  
7  } finally {  
8      // Wird garantiert ausgeführt  
9  }
```



? Frage

- Was wird ausgegeben?

```
1 public class TryCatchFinally1 {
2     static int recursiveIncrease(int i) {
3         return recursiveIncrease(i + 1);
4     }
5     public static void main(String[] args) {
6         int[] a = new int[4];
7         try {
8             a[4] = 0;
9         } catch (ArrayIndexOutOfBoundsException e1) {
10             recursiveIncrease(7);
11             System.out.println("Caught ArrayIndexOutOfBoundsException");
12         } finally {
13             System.out.println("Finally");
14         }
15         System.out.println("Exiting main()");
16     }
17 }
```



? Frage

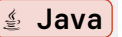
- Was wird ausgegeben?

```
1 public class TryCatchFinally2 {
2     public static void main(String[] args) {
3         System.out.println("Ratio = " + getRatio(3, 0));
4     }
5     public static int getRatio(int a, int b) {
6         int ratio = 0;
7         try {
8             ratio = a / b;
9         } catch (ArithmeticException e) {
10             System.out.println("Exception caught in getRatio()");
11             return 0;
12         } finally {
13             System.out.println("Finally");
14         }
15         System.out.println("Exiting getRatio()");
16         return ratio;
17     } }
```



- Regeln für Blöcke:
 - Genau einen try-Block als ersten Block
 - Keinen oder beliebig viele catch-Blöcke nach dem try-Block
 - Keinen oder einen finally-Block als letzten Block
 - Ein try-Block muss mindestens einen catch- oder finally-Block haben.
- Folgender Aufbau ist zulässig:

```
1 try {  
2     // ...  
3 } finally {  
4     // ...  
5 }
```



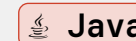
5. Eigene Ausnahmen definieren

5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

- Betrachten wir folgendes Programm:

```
1  public class OwnException1 {
2      public static void main(String[] args) {
3          double x = 25.0;
4          System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
5      }
6
7      public static double squareRoot(double x) {
8          return Math.sqrt(x);
9      }
10 }
```



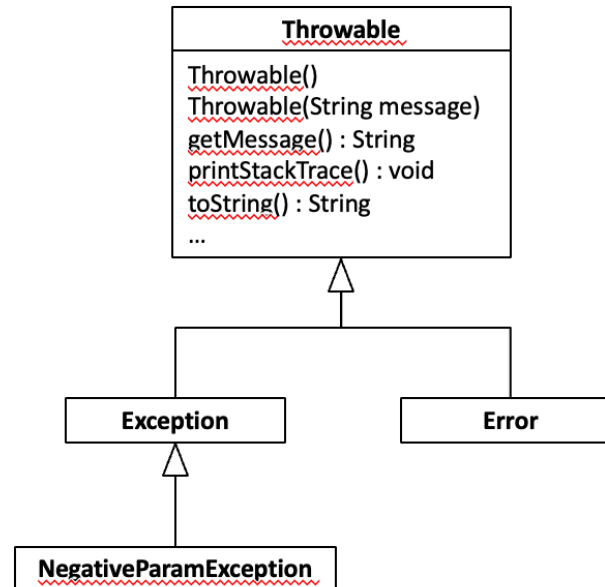
? Frage

- Methode squareRoot() soll für negative Parameter eine Ausnahme werfen
- Wie könnten wir einen eigenen Typ (z.B. NegativeParameterException) definieren?

5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

- Eigenen Ausnahmetyp durch Ableiten einer bestehenden Klasse
- Erster Ansatz: Ableiten der Klasse Exception



5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

- Ansatz erzeugt Fehlermeldung („Unbehandelte Ausnahme“)
- Wieso denn das jetzt?!

```
1  class NegativeParamException extends Exception {
2  }
3
4  public class OwnException2 {
5      public static void main(String[] args) {
6          double x = 25.0;
7          System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
8      }
9
10     public static double squareRoot(double x) {
11         if (x < 0.0) {
12             throw new NegativeParamException();
13         }
14         return Math.sqrt(x);
15     }
16 }
```

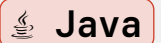


5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

- Hintergrund:
 - Ausnahmen müssen gefangen werden ODER
 - Methode muss über throws deklarieren, dass sie einen Ausnahmetyp werfen kann.

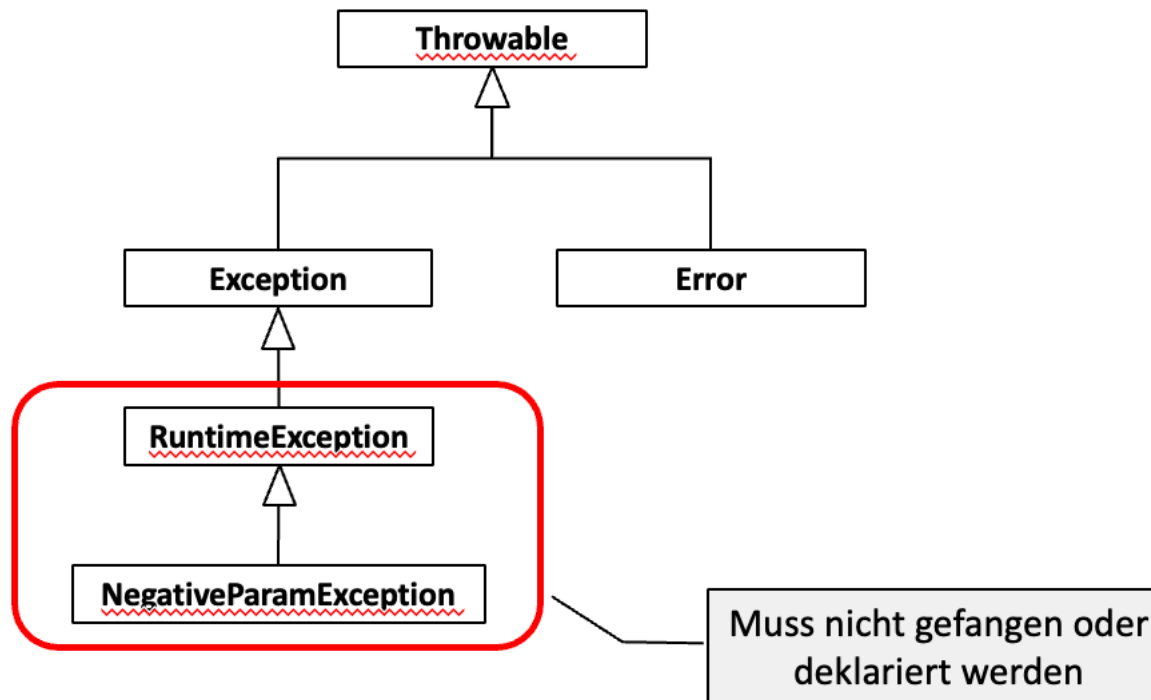
```
1  public class OwnException2 {
2      public static void main(String[] args) throws NegativeParamException {
3          double x = 25.0;
4          System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
5      }
6
7      public static double squareRoot(double x) throws NegativeParamException {
8          if (x < 0.0) {
9              throw new NegativeParamException();
10         }
11         return Math.sqrt(x);
12     }
13 }
```



5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

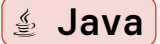
- Dies gilt für alle Ausnahmetypen (d.h. Throwable und davon abgeleitet) außer für:
 - Klasse RuntimeException
 - Von RuntimeException (direkt oder indirekt) abgeleitete Klassen



5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

```
1  class NegativeParamException extends RuntimeException {
2  }
3
4  public class OwnRuntimeException {
5      public static void main(String[] args) {
6          double x = 25.0;
7          System.out.printf("sqrt(%f) = %f", x, squareRoot(x));
8      }
9
10     public static double squareRoot(double x) {
11         if (x < 0.0) {
12             throw new NegativeParamException();
13         }
14         return Math.sqrt(x);
15     }
16 }
```

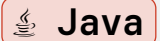


5.1 Eigene Ausnahmen

5. Eigene Ausnahmen definieren

- Beschreibung („message“) an Konstruktor der Basisklasse übergeben

```
1  class MyException extends Exception {
2      public MyException(String message) {
3          super(message);
4      }
5  }
6
7  public class OwnExceptionWithMessage {
8      public static void main(String[] args) {
9          try {
10             throw new MyException("An exception just for fun :-) ...");
11         } catch (MyException e) {
12             System.out.println("Message: " + e.getMessage());
13         }
14     }
15 }
```



6. License Notice

6.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- `link(„https://creativecommons.org/licenses/by-nc-sa/4.0/“)`
- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.