AUSNAHMEBEHANDLUNG (EXCEPTIONS)

Übersicht

- Begriffsklärung
- Übersicht zum Ablauf
- Arten von Exceptions
- Checked Exceptions
- try-catch
- throws
- try-catch-finally
- mehrere catch-Blöcke

Begriffsklärung

Exception

- engl. für „Ausnahme“
- steht für **Exceptional Event**
- Tritt während des Programmablaufs auf
- unterbricht „normalen“ Ablaufplan

Exception-Objekt

- wird erzeugt, weitergegeben und behandelt
- Objekt mit Information zu Fehlerursache
  - Fehlertyp
  - Stelle im Programmcode

Exception Handling (Fehlerbeseitigung)

- Exception auffangen
- Maßnahmen zur Behebung / Ablaufplanänderung
- Programmablauf (ggf. anders als geplant) fortsetzen

Beispiele für Ursachen

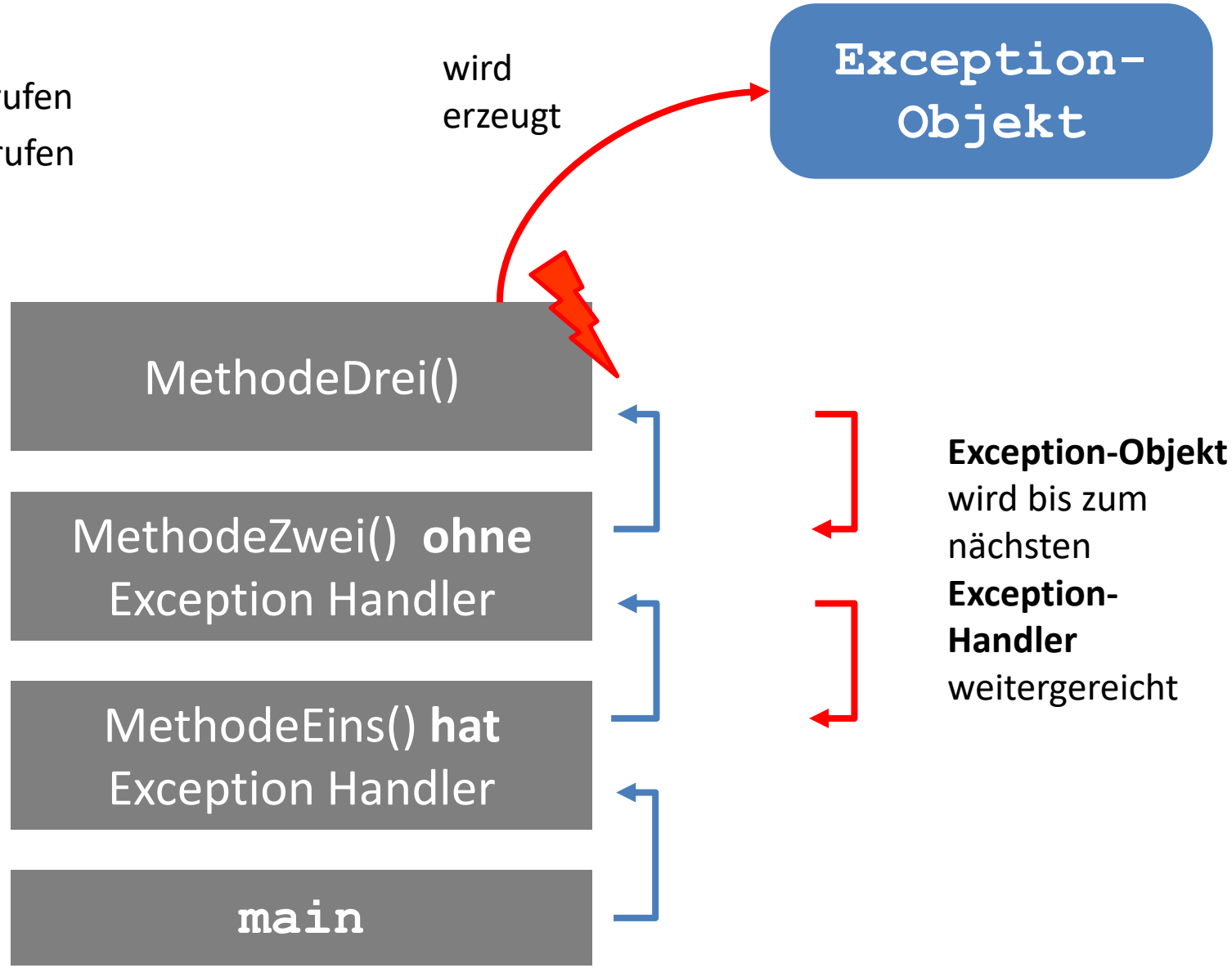
- Nutzereingabe ist fehlerhaft
- Datei kann nicht geöffnet werden
- Netzwerkfehler tritt auf
- Speicher steht nicht zur Verfügung

Exception Handling Übersicht

Ablauf ohne Fehler

- In main wird MethodeEins aufgerufen
- In MethodeEins wird MethodeZwei aufgerufen
- In MethodeZwei wird MethodeDrei aufgerufen

➔ **Fehler tritt auf in MethodeDrei**



Arten von Exceptions

Checked Exception

- im Programmcode muss das Auftreten erwartet werden
- Compiler prüft, ob auf throws mit einem try-catch reagiert wird
- Fehlerbehandlung und Rückkehr in normalen Programmablauf muss gewährleistet sein

Unchecked Exceptions

- im Programmcode muss das Auftreten nicht erwartet werden
- bspw. bei Benutzungsfehler (Fehler durch Benutzer hervorgerufen)
  - NullPointerException (Objekt fehlt an einer Stelle, wo darauf zugegriffen wird)
  - IllegalArgumentException (Eingabe ist fehlerhaft)

Errors

- Fehler, die den Ablauf des gesamten Programms in Frage stellen
- bspw. bei Entwicklungsfehlern
  - OutOfMemoryException (es ist nicht genügend Speicher vorhanden)

Checked Exceptions

Checked Exception

- im Programmcode muss das Auftreten erwartet werden (Compiler prüft das!)
- Fehlerbehandlung und Rückkehr in normalen Programmablauf muss gewährleistet sein

Beispiel: Datei öffnen

- FileReader ist Klasse für das Lesen aus Dateien
- Konstruktor nimmt Dateiname als Parameter
  - prüft ob Datei vorhanden ist
  - Fehlermöglichkeit: Datei ist nicht vorhanden
  - Konstruktor wirft (throws) FileNotFoundException im Fehlerfall

Fehlerbehandlungsstrategie: Catch or Specify

- **Catch** („versuche mal auszuführen, ansonsten Fehler behandeln“)
  - Umrahmen des kritischen Programmcodes mit **try**
  - Auffangen der Exception in **catch**
  - nach Fehler im catch-Block wieder in normalen Ablauf zurückkehren
- **Specify** („soll sich doch ein anderer um die Fehlerbehandlung kümmern“)
  - Exception an den Aufrufer zurückgeben

try / catch

try:
Versuch mal, diesen Programmcode auszuführen. Ich bin mir bewusst, dass dabei ein Fehler auftreten könnte, den ich jetzt aber noch nicht prüfen kann.

catch:
Fang den Fehler auf! Es kann ein bestimmter Fehler auftreten und dieser wird hier behandelt. Es wird zu einem fehlerfreien Programmablauf zurückgeführt. Ggf. passiert noch eine Fehleranalyse und –logging.

```
try {
    // Programmcode, bei dem ein Fehler
    // auftreten könnte

    // mgl. ohne Fehler, dann wird nur
    // dieser Teil ausgeführt

    // bei Fehler unmittelbar diesen Teil
    // verlassen
} catch( Exception e ) {
    // Programmcode für den Fall, dass
    // der Fehler aufgetreten ist
}
```

Checked Exception – Check

```
public void testFileReading(String[] args) {

     File file = new File(" C:\\daten\\files\\test.txt ");

     BufferedReader br;
     try {
          br = new BufferedReader(new FileReader(file));
     } catch (FileNotFoundException e) {
          e.printStackTrace();
          return;
     }

     String st;

     try {
          while ((st = br.readLine()) != null)
          System.out.println(st);
     } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
     }
}
```

Neues Objekt für eine Datei anlegen, aber noch nichts damit machen (z. B. Öffnen)

Neue Datei wird im FileReader-Constructor geöffnet. Falls Datei nicht vorhanden, dann wird FileNotException geworfen.

Checked Exception – Check

```
public void testFileReading(String[] args) {

    File file = new File(" C:\\daten\\files\\test.txt ");

    BufferedReader br;
    try {
        br = new BufferedReader(new FileReader(file));
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return;
    }
```

Neues Objekt für eine Datei anlegen, aber noch nichts damit machen (z. B. Öffnen)

Neue Datei wird im FileReader-Constructor geöffnet. Falls Datei nicht vorhanden, dann wird FileNotException geworfen.

FileReader https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html

```
public FileReader(String fileName)
          throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

Parameters:

`fileName` - the name of the file to read from

Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

Spezifikation des Constructors enthält
`throws FileNotFoundException`

➔ Da kann ein Fehler passieren, weil die Datei nicht gefunden wird. In diesem Fall wird abgebrochen und eine Exception geworfen.

Checked Exception – Check

```
public void testFileReading(String[] args) {

     File file = new File(" C:\\daten\\files\\test.txt ");

     BufferedReader br;
     try {
          br = new BufferedReader(new FileReader(file));
     } catch (FileNotFoundException e) {
          e.printStackTrace();
          return;
     }

     String st;

     try {
          while ((st = br.readLine()) != null)
          System.out.println(st);
     } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
     }
}
```

Neues Objekt für eine Datei anlegen, aber noch nichts damit machen (z. B. Öffnen)

Neue Datei wird im FileReader-Constructor geöffnet. Falls Datei nicht vorhanden, dann wird **FileNotException** geworfen.

Falls **FileNotFoundException** geworfen wurde (Fehler ist passiert), wird hiermit zum normalen Programmablauf zurückgeführt.

Hier: Abbruch mit **return**, ohne dass weiter in der Datei gelesen wird.

Checked Exception – Check

```
public void testFileReading(String[] args) {

     File file = new File(" C:\\daten\\files\\test.txt ");

     BufferedReader br;
     try {
          br = new BufferedReader(new FileReader(file));
     } catch (FileNotFoundException e) {
          e.printStackTrace();
          return;
     }

     String st;

     try {
          while ((st = br.readLine()) != null)
          System.out.println(st);
     } catch (IOException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
     }
}
```

Neues Objekt für eine Datei anlegen, aber noch nichts damit machen (z. B. Öffnen)

Neue Datei wird im FileReader-Constructor geöffnet. Falls Datei nicht vorhanden, dann wird **FileNotException** geworfen.

Falls **FileNotFoundException** geworfen wurde (Fehler ist passiert), wird hiermit zum normalen Programmablauf zurückgeführt.

Hier: Abbruch mit **return**, ohne dass weiter in der Datei gelesen wird.

Ähnliches Verfahren für das Lesen der einzelnen Zeilen

Checked Exception – Specify

```
public void testFileReading(String[] args) throws Exception
{
     File file = new File("C:\\daten\\files\\test.txt");

     BufferedReader br = new BufferedReader(new FileReader(file));

     String st;

     while ((st = br.readLine()) != null)
          System.out.println(st);
}
```

Zusatzinformation zur Methode

- hier könnte ein Fehler auftreten
- **Aufrufer von Methode muss sich um Exception kümmern**
- Exception kann auch genauer mit genauerer Unterklasse angegeben werden

FileReader

```
public FileReader(String fileName)
          throws FileNotFoundException
```

Creates a new `FileReader`, given the name of the file to read from.

Parameters:

`fileName` - the name of the file to read from

Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.

Constructor FileReader kann die eigentliche Fehlerursache sein

https://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html

finally

Optionale abschließende Behandlung

finally:
Abschließend machen! Egal ob try erfolgreich ohne Fehler gelaufen ist oder im catch ein Fehler abgefangen wurde, finally wird immer ausgeführt.

```
try {
    // try to do something
} catch( Exception e ) {
    // Programmcode für den Fall, dass
    // der Fehler aufgetreten ist
} finally {
    // hier wird jetzt etwas gemacht, dass
    // auf jeden Fall immer gemacht werden
    // muss
}
```

finally

```
public static void main(String[] args) {

     Scanner myScanner = new Scanner(System.in);

     try {
          System.out.println("Bitte einen Integer eingeben:");
          int einInteger = myScanner.nextInt();
          System.out.println("Eingegebener Integer: "+einInteger);

     } catch(InputMismatchException inputMismatchException) {

          System.out.println("Fehler bei der Eingabe! kein Integerwert eingegeben.");

     } finally {

          System.out.println("Finally, der Scanner wird noch geschlossen.");
          myScanner.close();

     }

}
```

Dieser Block wird im Fehlerfall ausgeführt.

Ob ohne oder mit Fehler, dieser Block wird auf jeden Fall am Ende ausgeführt.

Mehrere catch-Blöcke

Falls im try-Block mehrere Exception-Typen auftreten können, dann können diese auch differenziert nach Typ aufgefangen werden

Unterschiedliche Typen von Exceptions werden in unterschiedlichen catch-Blöcken aufgefangen und individuell behandelt

```
try {
    // try to do something
    // mehrere Typen von Exceptions könnten
    // hier auftreten
} catch( ExceptionTypeOne e ) {
    // Programmcode für den Fall, dass
    // erster Typ von Exceptions aufgetreten
} catch( ExceptionTypeTwo e ) {
    // Programmcode für den Fall, dass
    // zweiter Typ von Exceptions aufgetreten
} catch( ExceptionTypeThree e ) {
    // Programmcode für den Fall, dass
    // dritter Typ von Exceptions aufgetreten
}
```

Sonstiges zu Exceptions

Weiterführende Möglichkeiten

- eigene Exceptions können erstellt werden
- Vererbungshierarchie vorhanden
- Stack Trace-Informationen einsehen für Details zu Fehler
- Logger für Ausgabe des Fehlers nutzen

Zusammenfassung

- Exceptions betreffen Fehler und deren Behandlung
- Checked Exceptions MÜSSEN behandelt werden
- Möglichkeiten zur Ausnahmebehandlung im Programmcode
  - try-catch
  - throws
  - try-catch-finally
  - mehrere catch-Blöcke

Eigene Exception- Klasse ableiten

```
public class SomeVerySpecialException extends Exception {

    public SomeVerySpecialException(String info) {
        super(info);
    }
}
```

Quellen zum Nachlesen:

https://rollbar.com/blog/java-exceptions-hierarchy-explained/

https://www.geeksforgeeks.org/exceptions-in-java/

```java
package business.order.basic;

public class OrderDemo {

    public static void main(String[] args) {

        OrderBook orders = new OrderBook();

        try {
            Order o1 = new Order("IBett5", -99.12, 200);
            orders.addOrder(o1);
        } catch (IllegalArgumentException ex)
        {
            System.out.println("The was an Exception: "+ex.getMessage());
            ex.printStackTrace();
        }

        Order ord = orders.getOrder("IBett5");
        System.out.println(ord);
    }

}
```