

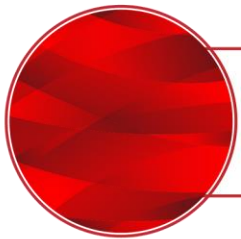


# Exception Handling

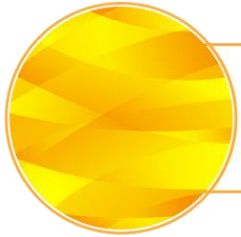
04. Dezember 2024

# Konventionen zu den Aufgaben

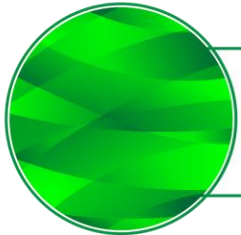
---



Alle Aufgaben mit einer roten oder rot-gelben Bewertung **müssen** bis **Samstag 23:59 Uhr** nachgebessert werden.



Alle Aufgaben mit einer gelben oder gelb-grünen Bewertung können bis **Samstag 23:59 Uhr** nachgebessert werden.



Die  
Bewertung  
**kann** sich  
damit  
verbessern!



# Dungeon Crawl

---

Jeder bekommt ein kurzes Feedback bis Montag.

Der Stand, welcher betrachtet wird, ist der, der auf Git am 04.12.2024 um 23:59 Uhr liegt.

Es wird keine Note geben!

Wer möchte, kann heute eine Stunde im Tutorium noch nutzen oder freiwillig nach dem Kurs weiter arbeiten.



# Wiederholung Git

---

WIE IST UNSER WORKFLOW?

# Plan für die Woche

---

Montag

Dienstag

Mittwoch

Donnerstag

Freitag

- Einführung Exceptions und Errors
- Vorträge halten

- Try-catch
- Throw
- Debugging
- null

- Interfaces
- Static, final, inner classes

- Vorträge

# Plan für heute

---

- Try-catch
- Throw
- Debugging
- null



# Quiz

---

Welche Aussagen sind zu Exceptions korrekt?

- a) Exceptions sind Ausnahmen, die außerhalb des normalen Programmablaufs auftreten können, aber den Programmablauf nicht stören.
- b) Exceptions sind schwerwiegende Systemprobleme.
- c) Exceptions treten auf, wenn während der Programmausführung ein Fehler auftritt.
- d) Exceptions sind Anwendungsfehler, die vom Programm behandelt werden können.

Es gibt **zwei** mögliche Antworten.



# Quiz

---

Welche der folgenden Aussagen über Checked Exceptions ist korrekt?

- a) Checked Exceptions müssen nicht explizit in einem try-catch-Block behandelt werden.
- b) Checked Exceptions sind Unterklassen von RuntimeException.
- c) Checked Exceptions müssen entweder behandelt oder in der Methodendeklaration angegeben werden.
- d) Checked Exceptions treten nur während der Kompilierungszeit auf.



# try-catch

---

# Try, catch und finally

---

- **try:** Code-Block, in dem eine Exception fallen könnte
- **catch:**
  - Exceptionshandler: Fängt eine Exception und behandelt die Ausnahme
  - Es können mehrere Exception gefangen werden entweder in einem oder mehreren catch-Blöcken
- **finally:** Der hier enthaltene Code wird immer ausgeführt

```
try {  
    // Code, der eine Exception verursachen könnte  
} catch (ExceptionType e) {  
    // Code zur Behandlung der Exception  
} finally {  
    // Code, der immer ausgeführt wird, egal ob eine Exception aufgetret  
}
```



# Intellij-Aufgabe


---

# try mit Ressourcen

---

- try-Statement, welches Ressourcen, wie Dateien oder Datenbankenverbindungen etc., automatisch schließt, wenn sie nicht mehr benötigt werden

```
try (BufferedReader reader = new BufferedReader(new FileReader("example.txt"));  
    BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"));  
    // Code zur Nutzung der Ressourcen  
} catch (IOException e) {  
    // Exception Handling  
}
```



# Thrown Exceptions

---

# Thrown by a Method (throws)

---

- Nicht direkte Behandlung der Exception
- Weitergabe der Exception

```
public void methodName() throws ExceptionType1, ExceptionType2 {  
    // Methodenkörper  
}
```

# Throw

---

- Manuelles/explizites Auslösen von Exceptions
- Innerhalb des Codes

```
throw new ExceptionType("Fehlermeldung");
```



# Intellij-Aufgabe

---



# Weiteres zu Exceptions

---

# Wrap und Rethrowing

---

– Rethrow:

```
public List<Player> loadAllPlayers(String playersFile)
    throws IOException {
    try {
        // ...
    } catch (IOException io) {
        throw io;
    }
}
```

–Wrap und Rethrow:

```
public List<Player> loadAllPlayers(String playersFile)
    throws PlayerLoadException {
    try {
        // ...
    } catch (IOException io) {
        throw new PlayerLoadException(io);
    }
}
```

Aus: [Exception Handling in Java | Baeldung](#)

# Wichtige Methoden von Exceptions

---

- **getMessage()**: Gibt eine beschreibende Nachricht der Ausnahme zurück.
- **printStackTrace()**: Druckt die Stack Trace der Ausnahme. Dies ist nützlich, um den Ort zu finden, an dem die Ausnahme ausgelöst wurde.
- **toString()**: Gibt eine kurze Beschreibung der Ausnahme zurück, die den Klassennamen und die detaillierte Nachricht enthält.
- **getStackTrace()**: Gibt ein Array von StackTraceElementen zurück, die den Stack Trace dieser Throwable darstellen.



Info: Ein **Stacktrace** ist eine detaillierte Liste, die den Ablauf der Methodenaufrufe zeigt, die zu einem bestimmten Punkt im Programm führten, normalerweise wenn eine Exception auftritt.

# Eigene Exceptions

---

- Erstellen von benutzerdefinierten Exceptions
- Müssen von passender Super-Klasse bspw. *Exception* oder einer der Sub-Klassen erben
- Es sollte ein Konstruktor mit einer Fehlermeldung bereitgestellt werden
- Durch throw kann sie im Code ausgelöst werden und an anderer Stelle normal behandelt werden



# Intellij-Beispiel

---

# null

---

# Null-Werte

---

- Wert, der anzeigt, dass eine Variable keinen Verweis auf ein gültiges Objekt hat
- „kein Objekt“, „nicht initialisiert“
- anwendbar auf alle Referenztypen
- Alle nicht initialisierten Objekt-Referenzvariablen sind standardmäßig null
- **Problemvermeidung:** Immer prüfen, ob Objekte null sind!

– Beispiel:

```
if (meinObjekt != null) {  
    meinObjekt.meineMethode();  
} else {  
    System.out.println("Objekt ist null");  
}
```

- Passendes Exceptionhandling im Falle von null



Info: Andere Libraries verfügen noch über andere Möglichkeiten des Prüfens auf null. Bspw. Google Guava hat einen Preconditions-Check.

# Debugging

---





# Intellij-Aufgabe

---

# Quellen

---

[Lesson: Exceptions \(The Java™ Tutorials > Essential Java Classes\)](#)

[10.1 Java-Ausnahmen \(Exceptions\) behandeln | Scalingbits](#)

[Exception Handling in Java | Baeldung](#)

<https://pixabay.com/de/illustrations/ampel-element-design-rot-gelb-2001073/>