

9. Tutorium

Java-API, Prinzipien der ObjektOrientierung

Tutorium 14

Péter Bohner | 18.01.2023

Tutorium 14

2023-01-18



Inhaltsverzeichnis



- 1. Wiederholung
- 2. JavaDoc
- 3. Exceptions
- 4. Java-API
- 5. OO-Design-Prinzipien

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
ooo	ooooo	ooooo	oooooooooooooooooooo	oooooooooooo	ooooooooooooo

Wiederholung



Welche der Aussagen treffen auf Interfaces zu?

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
●○○	○○○○○	○○○○○	○○○○○○○○○○○○○○○○	○○○○○○○○○	○○○○○○○○○○○

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Interfaces zu?

Wiederholung



Welche der Aussagen treffen auf Interfaces zu?

- Definieren eine Schnittstelle
- Definieren eine Sammlung von Methodensignaturen
- Eine Klasse kann mehrere Interfaces implementieren
- Ein Interface kann von beliebig vielen anderen Interfaces erben

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Interfaces zu?

■ Definieren eine Schnittstelle

■ Definieren eine Sammlung von Methodensignaturen

■ Eine Klasse kann mehrere Interfaces implementieren

■ Ein Interface kann von beliebig vielen anderen Interfaces erben

Wiederholung



Welche der Aussagen treffen auf Interfaces zu?

- Definieren eine Schnittstelle
- Definieren eine Sammlung von Methodensignaturen
- Eine Klasse kann mehrere Interfaces implementieren
- Ein Interface kann von beliebig vielen anderen Interfaces erben

Was stellt eine Exception dar?

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Interfaces zu?
■ Definieren eine Schnittstelle
■ Definieren eine Sammlung von Methodensignaturen
■ Eine Klasse kann mehrere Interfaces implementieren
■ Ein Interface kann von beliebig vielen anderen Interfaces erben
Was stellt eine Exception dar?

Wiederholung



Welche der Aussagen treffen auf Interfaces zu?

- Definieren eine Schnittstelle
- Definieren eine Sammlung von Methodensignaturen
- Eine Klasse kann mehrere Interfaces implementieren
- Ein Interface kann von beliebig vielen anderen Interfaces erben

Was stellt eine Exception dar?

Eine Ausnahme

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Interfaces zu?

■ Definieren eine Schnittstelle

■ Definieren eine Sammlung von Methodensignaturen

■ Eine Klasse kann mehrere Interfaces implementieren

■ Ein Interface kann von beliebig vielen anderen Interfaces erben

Was stellt eine Exception dar?

Eine Ausnahme

Wiederholung



Welche der Aussagen treffen auf Interfaces zu?

- Definieren eine Schnittstelle
- Definieren eine Sammlung von Methodensignaturen
- Eine Klasse kann mehrere Interfaces implementieren
- Ein Interface kann von beliebig vielen anderen Interfaces erben

Was stellt eine Exception dar?

Eine Ausnahme

Mit welchen Schlüsselwörtern wirft man in Java eine Exception?

Wiederholung
●○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14
└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Interfaces zu?
■ Definieren eine Schnittstelle
■ Definieren eine Sammlung von Methodensignaturen
■ Eine Klasse kann mehrere Interfaces implementieren
■ Ein Interface kann von beliebig vielen anderen Interfaces erben
Was stellt eine Exception dar?
Eine Ausnahme
Mit welchen Schlüsselwörtern wirft man in Java eine Exception?

Wiederholung



Welche der Aussagen treffen auf Interfaces zu?

- Definieren eine Schnittstelle
- Definieren eine Sammlung von Methodensignaturen
- Eine Klasse kann mehrere Interfaces implementieren
- Ein Interface kann von beliebig vielen anderen Interfaces erben

Was stellt eine Exception dar?

Eine Ausnahme

Mit welchen Schlüsselwörtern wirft man in Java eine Exception?

```
throw new ExceptionName();
```

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Interfaces zu?

- Definieren eine Schnittstelle
- Definieren eine Sammlung von Methodensignaturen
- Eine Klasse kann mehrere Interfaces implementieren
- Ein Interface kann von beliebig vielen anderen Interfaces erben

Was stellt eine Exception dar?

Eine Ausnahme

Mit welchen Schlüsselwörtern wirft man in Java eine Exception?

```
throw new ExceptionName();
```

Wiederholung
●○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

Wiederholung



Wie fange und behandle ich eine Exception?

Wiederholung



Wie fange und behandle ich eine Exception?

```
try {  
    // code  
} catch (ExceptionType e) {  
    // exception-handling  
}
```

Wiederholung



Wie fange und behandle ich eine Exception?

```
try {  
    // code  
} catch (ExceptionType e) {  
    // exception-handling  
}
```

Welche der Aussagen treffen idealerweise auf Exceptions zu?

Wiederholung



Wie fange und behandle ich eine Exception?

```
try {  
    // code  
} catch (ExceptionType e) {  
    // exception-handling  
}
```

Welche der Aussagen treffen idealerweise auf Exceptions zu?

- Steuern **niemals** den Kontrollfluss
- Stellen die Ausnahme dar
- Fangen vorhersehbare und behandelbare Fehler ab
- Trennen Programmlogik und Fehlerbehandlung sauber voneinander

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Wie fange und behandle ich eine Exception?

```
try {  
    // code  
} catch (ExceptionType e) {  
    // exception-handling  
}
```

Welche der Aussagen treffen idealerweise auf Exceptions zu?

- Steuern **niemals** den Kontrollfluss
- Stellen die Ausnahme dar
- Fangen vorhersehbare und behandelbare Fehler ab
- Trennen Programmlogik und Fehlerbehandlung sauber voneinander

Wiederholung

Welche der Aussagen treffen auf Rekursive-Methoden zu?



2023-01-18

Tutorium 14
└─ Wiederholung
 └─ Wiederholung

Welche der Aussagen treffen auf Rekursive-Methoden zu?

Wiederholung



Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Wiederholung



Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

2023-01-18

Tutorium 14

└─ Wiederholung

└─ Wiederholung

Welche der Aussagen treffen auf Rekursive-Methoden zu?
■ Speicher-/Zeitaufwändiger als iteratives Äquivalent
■ Termination kann nicht vorausgesagt werden
■ Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
■ Falls bekannt, besser iterativ
Was passiert in Java, wenn eine Methode aufgerufen wird?

Wiederholung



Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

- Lokale Variablen (inkl. Paramter),
 - Operanden,
 - Rücksprungsadresse
- ... werden auf den Aufrufstapel geschrieben

Wie deklariert man in Java die Verwendung eines Generics T in einer Klasse?

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

- Lokale Variablen (inkl. Paramter),
- Operanden,
- Rücksprungsadresse

... werden auf den Aufrufstapel geschrieben

Wie deklariert man in Java die Verwendung eines Generics T in einer Klasse?

Wiederholung



Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

- Lokale Variablen (inkl. Paramter),
 - Operanden,
 - Rücksprungsadresse
- ... werden auf den Aufrufstapel geschrieben

Wie deklariert man in Java die Verwendung eines Generics T in einer Klasse?

```
public class ClassName<T> { ... }
```

Wie gibt man den konkreten Datentyp bei der Instanzierung einer Klasse die Generics nutzt an?

2023-01-18

Tutorium 14

└─Wiederholung

└─Wiederholung

Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

- Lokale Variablen (inkl. Parameter),
- Operanden,
- Rücksprungsadresse

... werden auf den Aufrufstapel geschrieben

Wie deklariert man in Java die Verwendung eines Generics T in einer Klasse?

```
public class ClassName<T> { ... }
```

Wie gibt man den konkreten Datentyp bei der Instanzierung einer Klasse die Generics nutzt an?

Wiederholung



Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

- Lokale Variablen (inkl. Paramter),
 - Operanden,
 - Rücksprungsadresse
- ... werden auf den Aufrufstapel geschrieben

Wie deklariert man in Java die Verwendung eines Generics T in einer Klasse?

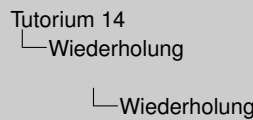
```
public class ClassName<T> { ... }
```

Wie gibt man den konkreten Datentyp bei der Instanziierung einer Klasse die Generics nutzt an?

```
ClassName<String> objectName = new ClassName<>();
```

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
○○●	○○○○○	○○○○○	○○○○○○○○○○○○○○○○	○○○○○○○○○	○○○○○○○○○○○

2023-01-18



Welche der Aussagen treffen auf Rekursive-Methoden zu?

- Speicher-/Zeitaufwändiger als iteratives Äquivalent
- Termination kann nicht vorausgesagt werden
- Vorteilhaft, falls Anzahl der Iterationen noch nicht bekannt
- Falls bekannt, besser iterativ

Was passiert in Java, wenn eine Methode aufgerufen wird?

- Lokale Variablen (inkl. Parameter),
- Operanden,
- Rücksprungsadresse

... werden auf den Aufrufstapel geschrieben

Wie deklariert man in Java die Verwendung eines Generics T in einer Klasse?

```
public class ClassName<T> { ... }
```

Wie gibt man den konkreten Datentyp bei der Instanziierung einer Klasse die Generics nutzt an?

```
ClassName<String> objectName = new ClassName<>();
```

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben

2023-01-18

Tutorium 14
└─ Übungsblatt 4
 └─ Übungsblatt 4

Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!**

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** *Wieso muss ich das beim 4. ÜB noch sagen?*

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
ooo	●oooo	oooo	oooooooooooooooo	oooooooo	oooooooo

Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** Wieso muss ich das beim 4. ÜB noch sagen?

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** *Wieso muss ich das beim 4. ÜB noch sagen?*
Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.

Code Stil

- Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4

Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** Wieso muss ich das beim 4. ÜB noch sagen?
- Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.

Code Stil

- Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** *Wieso muss ich das beim 4. ÜB noch sagen?*
Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.

Code Stil

- Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)
- Bedingung und Block nicht auf eine Zeile.

2023-01-18

Tutorium 14
└─Übungsblatt 4

└─Übungsblatt 4

Allgemein
<ul style="list-style-type: none">■ A 9 und B 15 Abgaben■ Die meisten sollten ihren Übungsschein haben■ Logik Hardcoden bringt nichts! <small>Wieso muss ich das beim 4. ÜB noch sagen?</small> <p>Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.</p>
Code Stil
<ul style="list-style-type: none">■ Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)■ Bedingung und Block nicht auf eine Zeile.

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** *Wieso muss ich das beim 4. ÜB noch sagen?*
Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.

Code Stil

- Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)
- Bedingung und Block nicht auf eine Zeile.
- *Haut einfach den Auto-Formatter vor der Abgabe drauf*

2023-01-18

Tutorium 14
└─ Übungsblatt 4

└─ Übungsblatt 4

Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** Wieso muss ich das beim 4. ÜB noch sagen?
- Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.

Code Stil

- Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)
- Bedingung und Block nicht auf eine Zeile.
- Haut einfach den Auto-Formatter vor der Abgabe drauf

Übungsblatt 4



Allgemein

- A 9 und B 15 Abgaben
- Die meisten sollten ihren Übungsschein haben
- **Logik Hardcoden bringt nichts!** *Wieso muss ich das beim 4. ÜB noch sagen?*
Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.

Code Stil

- Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)
- Bedingung und Block nicht auf eine Zeile.
- *Haut einfach den Auto-Formatter vor der Abgabe drauf*
- *Achtet auf die Rechtschreibung, vermeidet Stilblüten*

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4

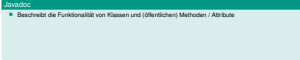
Allgemein
<ul style="list-style-type: none">■ A 9 und B 15 Abgaben■ Die meisten sollten ihren Übungsschein haben■ Logik Hardcoden bringt nichts! <small>Wieso muss ich das beim 4. ÜB noch sagen?</small> <p>Wenn ihr z.B. den BFS nicht hin bekommt, wird das mit den Abschlussaufgaben schwer.</p>
Code Stil
<ul style="list-style-type: none">■ Leere Zeilen zwischen Abschnitten (oder besser in Hilfsmethoden auslagern)■ Bedingung und Block nicht auf eine Zeile.■ Haut einfach den Auto-Formatter vor der Abgabe drauf■ Achtet auf die Rechtschreibung, vermeidet Stilblüten

Übungsblatt 4



Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute





Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll

Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll

Übungsblatt 4



Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll
- Deshalb: (in-) Valide Parameter (Kombinationen) beschreiben

Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll
- Deshalb: (in-) Valide Parameter (Kombinationen) beschreiben



Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll
- Deshalb: (in-) Valide Parameter (Kombinationen) beschreiben
- Beschreiben, was in Randfällen, Fehlerzuständen passiert

Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll
- Deshalb: (in-) Valide Parameter (Kombinationen) beschreiben
- Beschreiben, was in Randfällen, Fehlerzuständen passiert

Übungsblatt 4



Javadoc

- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll
- Deshalb: (in-) Valide Parameter (Kombinationen) beschreiben
- Beschreiben, was in Randfällen, Fehlerzuständen passiert
- Welche Exceptions werden wann geworfen?

Wiederholung
○○○

Übungsblatt 4
○●○○○

Javadoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4

Javadoc

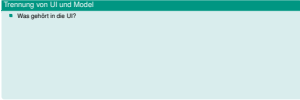
- Beschreibt die Funktionalität von Klassen und (öffentlichen) Methoden / Attribute
- Nur mit Hilfe der Javadoc soll klar sein, was genau das dokumentierte Objekt soll
- Deshalb: (in-) Valide Parameter (Kombinationen) beschreiben
- Beschreiben, was in Randfällen, Fehlerzuständen passiert
- Welche Exceptions werden wann geworfen?

Übungsblatt 4 - Aufgabe A



Trennung von UI und Model

- Was gehört in die UI?



Übungsblatt 4 - Aufgabe A



Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen

Wiederholung
○○○

Übungsblatt 4
○○●○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe A

Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen

Übungsblatt 4 - Aufgabe A



Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen
- Was gehört ins Model?

Wiederholung
○○○

Übungsblatt 4
○○●○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─Übungsblatt 4

└─Übungsblatt 4 - Aufgabe A

Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen
- Was gehört ins Model?

Übungsblatt 4 - Aufgabe A



Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen
- Was gehört ins Model?
- Die Modellierung des wahren Zustandes und Logik
- **Keine Ein und Ausgaben!**

Wiederholung
○○○

Übungsblatt 4
○○●○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe A

- ### Trennung von UI und Model
- Was gehört in die UI?
 - Ein und Ausgabe (System.out, Scanner)
 - Das Parsen von Eingaben
 - Die Serialisation von Daten, die Ausgegeben werden sollen
 - Was gehört ins Model?
 - Die Modellierung des wahren Zustandes und Logik
 - **Keine Ein und Ausgaben!**

Übungsblatt 4 - Aufgabe A



Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen
- Was gehört ins Model?
- Die Modellierung des wahren Zustandes und Logik
- **Keine Ein und Ausgaben!**
- Das Model sollte keine invaliden Daten darstellen (nur richtig übertragenes Terrain sollte auch in einem Terrain-Objekt gespeichert werden)

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe A

Trennung von UI und Model

- Was gehört in die UI?
- Ein und Ausgabe (System.out, Scanner)
- Das Parsen von Eingaben
- Die Serialisation von Daten, die Ausgegeben werden sollen
- Was gehört ins Model?
- Die Modellierung des wahren Zustandes und Logik
- **Keine Ein und Ausgaben!**
- Das Model sollte keine invaliden Daten darstellen (nur richtig übertragenes Terrain sollte auch in einem Terrain-Objekt gespeichert werden)

Übungsblatt 4



Häufige Fehler in Aufgabe A

- Code-Duplication statt Hilfsmethoden parametrisieren

Übungsblatt 4



Häufige Fehler in Aufgabe A

- Code-Duplication statt Hilfsmethoden parametrisieren
- UI und IO vermischen

Häufige Fehler in Aufgabe A

- Code-Duplication statt Hilfsmethoden parametrisieren
- UI und IO vermischen

Übungsblatt 4



Häufige Fehler in Aufgabe A

- Code-Duplication statt Hilfsmethoden parametrisieren
- UI und IO vermischen
- Schlechte OO-Modellierung

Übungsblatt 4



Häufige Fehler in Aufgabe A

- Code-Duplication statt Hilfsmethoden parametrisieren
- UI und IO vermischen
- Schlechte OO-Modellierung
- Kontrollfluss mit Exceptions

Häufige Fehler in Aufgabe B

- Unvollständige JavaDoc (z.B. @throws fehlt)

Häufige Fehler in Aufgabe A
■ Code-Duplication statt Hilfsmethoden parametrisieren
■ UI und IO vermischen
■ Schlechte OO-Modellierung
■ Kontrollfluss mit Exceptions
Häufige Fehler in Aufgabe B
■ Unvollständige JavaDoc (z.B. @throws fehlt)

Übungsblatt 4



Häufige Fehler in Aufgabe A

- Code-Duplication statt Hilfsmethoden parametrisieren
- UI und IO vermischen
- Schlechte OO-Modellierung
- Kontrollfluss mit Exceptions

Häufige Fehler in Aufgabe B

- Unvollständige JavaDoc (z.B. @throws fehlt)
- Falscher Exception-Typ

Häufige Fehler in Aufgabe A
<ul style="list-style-type: none">■ Code-Duplication statt Hilfsmethoden parametrisieren■ UI und IO vermischen■ Schlechte OO-Modellierung■ Kontrollfluss mit Exceptions
Häufige Fehler in Aufgabe B
<ul style="list-style-type: none">■ Unvollständige JavaDoc (z.B. @throws fehlt)■ Falscher Exception-Typ

Übungsblatt 4 - Aufgabe B Exceptions



NullPointerException

Thrown when an application attempts to use null in a case where an object is required.

Übungsblatt 4 - Aufgabe B Exceptions



NullPointerException

Thrown when an application attempts to use null in a case where an object is required.

IllegalStateException

Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.

NullPointerException

Thrown when an application attempts to use null in a case where an object is required.

IllegalStateException

Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.

Übungsblatt 4 - Aufgabe B Exceptions



NullPointerException

Thrown when an application attempts to use null in a case where an object is required.

IllegalStateException

Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.

IllegalArgumentException

Thrown to indicate that a method has been passed an illegal or inappropriate argument.

NullPointerException
Thrown when an application attempts to use null in a case where an object is required.
IllegalStateException
Signals that a method has been invoked at an illegal or inappropriate time. In other words, the Java environment or Java application is not in an appropriate state for the requested operation.
IllegalArgumentException
Thrown to indicate that a method has been passed an illegal or inappropriate argument.

2023-01-18

Tutorium 14

└─Übungsblatt 4

└─Übungsblatt 4 - Aufgabe B Exceptions

Welche Exception passt am besten?

Übungsblatt 4 - Aufgabe B Exceptions



Welche Exception passt am besten?

■

if(sweets == null) throw new Something();

Welche Exception passt am besten?

■

if(sweets == null) throw new Something();

Übungsblatt 4 - Aufgabe B Exceptions



Welche Exception passt am besten?

- `if(sweets == null) throw new Something();`
`if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe B Exceptions

```
Welche Exception passt am besten?  
■ if(sweets == null) throw new Something();  
if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");
```

Übungsblatt 4 - Aufgabe B Exceptions



Welche Exception passt am besten?

- `if(sweets == null) throw new Something();`
`if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`
- `if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();`

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe B Exceptions

Welche Exception passt am besten?

■ `if(sweets == null) throw new Something();`

■ `if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`

■ `if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();`

Übungsblatt 4 - Aufgabe B Exceptions



Welche Exception passt am besten?

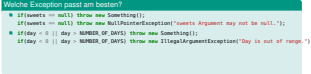
- `if(sweets == null) throw new Something();`
`if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`
- `if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();`
`if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");`

2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe B Exceptions



Übungsblatt 4 - Aufgabe B Exceptions



2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe B Exceptions

```
Welche Exception passt am besten?  
■ if(sweets == null) throw new Something();  
■ if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");  
■ if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();  
■ if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");  
■ if(!canOpenDoor(day)) throw new Something();
```

Welche Exception passt am besten?

- `if(sweets == null) throw new Something();`
`if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`
- `if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();`
`if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");`
- `if(!canOpenDoor(day)) throw new Something();`

Übungsblatt 4 - Aufgabe B Exceptions



2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe B Exceptions

```
Welche Exception passt am besten?  
■ if(sweets == null) throw new Something();  
  if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");  
■ if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();  
  if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");  
■ if(!canOpenDoor(day)) throw new Something();  
  if(!canOpenDoor(day)) throw new IllegalStateException("The door may not be opened yet");
```

Welche Exception passt am besten?

- `if(sweets == null) throw new Something();`
`if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`
- `if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();`
`if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");`
- `if(!canOpenDoor(day)) throw new Something();`
`if(!canOpenDoor(day)) throw new IllegalStateException("The door may not be opened yet");`

Übungsblatt 4 - Aufgabe B Exceptions



2023-01-18

Tutorium 14

Übungsblatt 4

Übungsblatt 4 - Aufgabe B Exceptions

Welche Exception passt am besten?

- `if(sweets == null) throw new Something();`
`if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");`
- `if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();`
`if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");`
- `if(!canOpenDoor(day)) throw new Something();`
`if(!canOpenDoor(day)) throw new IllegalStateException("The door may not be opened yet");`
- Anmerkung: Error messages bitte als Konstanten

Welche Exception passt am besten?

```
■ if(sweets == null) throw new Something();
  if(sweets == null) throw new NullPointerException("sweets Argument may not be null.");
■ if(day < 0 || day > NUMBER_OF_DAYS) throw new Something();
  if(day < 0 || day > NUMBER_OF_DAYS) throw new IllegalArgumentException("Day is out of range.");
■ if(!canOpenDoor(day)) throw new Something();
  if(!canOpenDoor(day)) throw new IllegalStateException("The door may not be opened yet");
■ Anmerkung: Error messages bitte als Konstanten
```

Wiederholung
○○○

Übungsblatt 4
○○○○●

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

JavaDoc - Was ist das?



2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc - Was ist das?

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
ooo	ooooo	●oooo	ooooooooooooooooo	ooooooooo	oooooooooooo

JavaDoc - Was ist das?



Javadoc...

- ... beschreibt Klassen, Methoden, Konstruktoren und Felder
- ... wird direkt in den Code geschrieben
- ... erleichtert das Verständnis

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
●○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc - Was ist das?

Javadoc...

- ... beschreibt Klassen, Methoden, Konstruktoren und Felder
- ... wird direkt in den Code geschrieben
- ... erleichtert das Verständnis

JavaDoc - Was ist das?



Javadoc...

- ... beschreibt Klassen, Methoden, Konstruktoren und Felder
- ... wird direkt in den Code geschrieben
- ... erleichtert das Verständnis

Regeln und Ziele

- **einheitlich** auf Deutsch oder auf Englisch
- vollständige und aussagekräftige Beschreibung

JavaDoc - Was ist das?



Javadoc...

- ... beschreibt Klassen, Methoden, Konstruktoren und Felder
- ... wird direkt in den Code geschrieben
- ... erleichtert das Verständnis

Regeln und Ziele

- **einheitlich** auf Deutsch oder auf Englisch
- vollständige und aussagekräftige Beschreibung

Javadoc zur Java API

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
●○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc - Was ist das?

Javadoc...
<ul style="list-style-type: none">■ ... beschreibt Klassen, Methoden, Konstruktoren und Felder■ ... wird direkt in den Code geschrieben■ ... erleichtert das Verständnis
Regeln und Ziele
<ul style="list-style-type: none">■ einheitlich auf Deutsch oder auf Englisch■ vollständige und aussagekräftige Beschreibung
Javadoc zur Java API

JavaDoc für Klassen



2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc für Klassen

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
○○○	○○○○○	○●○○○	○○○○○○○○○○○○○○○○○	○○○○○○○○○	○○○○○○○○○○○

JavaDoc für Klassen



- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist

2023-01-18

Tutorium 14

└─ JavaDoc

└─ JavaDoc für Klassen

■ beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist

JavaDoc für Klassen

- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
- mögliche Verwendungszwecke können erwähnt werden



2023-01-18

Tutorium 14

└─ JavaDoc

└─ JavaDoc für Klassen

- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
- mögliche Verwendungszwecke können erwähnt werden

JavaDoc für Klassen

- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
- mögliche Verwendungszwecke können erwähnt werden
- der Autor wird angegeben



2023-01-18

■ beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
■ mögliche Verwendungszwecke können erwähnt werden
■ der Autor wird angegeben

JavaDoc für Klassen

- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
- mögliche Verwendungszwecke können erwähnt werden
- der Autor wird angegeben
- die Versionsnummer wird angegeben



2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc für Klassen

- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
- mögliche Verwendungszwecke können erwähnt werden
- der Autor wird angegeben
- die Versionsnummer wird angegeben

JavaDoc für Klassen

- beschreibt, welchen Zweck die Klasse erfüllt und welche Funktionalität sie aufweist
- mögliche Verwendungszwecke können erwähnt werden
- der Autor wird angegeben
- die Versionsnummer wird angegeben

Syntax

```
/**
 * Kurze aber aussagekräftige Beschreibung der Klasse.
 * @author Autor
 * @version versionsNummer
 */
class ClassName {
```

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○●○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

JavaDoc für Klassen - Beispiel



```
/**
 * Modelliert einen zweidimensionalen Vektor.
 * Der Vektor besteht aus einer x- und y-Komponente.
 * @author Peter Bohner
 * @author uqknc
 * @version 1.0
 */
class Vector2D {
    double x;
    double y;
}
```

```
/**
 * Modelliert einen zweidimensionalen Vektor.
 * Der Vektor besteht aus einer x- und y-Komponente.
 * @author Peter Bohner
 * @author uqknc
 * @version 1.0
 */
class Vector2D {
    double x;
    double y;
}
```

JavaDoc für Konstruktoren



2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc für Konstruktoren

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
○○○	○○○○○	○○○●○	○○○○○○○○○○○○○○○○	○○○○○○○○○	○○○○○○○○○○○

JavaDoc für Konstruktoren

- beschreibt Zweck der Konstruktors



2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc für Konstruktoren

■ beschreibt Zweck der Konstruktors

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
ooo	ooooo	ooo●o	ooooooooooooooooo	ooooooooo	oooooooooooo

JavaDoc für Konstruktoren

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann



2023-01-18

Tutorium 14

└─ JavaDoc

└─ JavaDoc für Konstruktoren

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann

JavaDoc für Konstruktoren

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann
 - Spezialfälle, die das Verhalten betreffen



2023-01-18

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann
 - Spezialfälle, die das Verhalten betreffen

JavaDoc für Konstruktoren

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann
 - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und eventuell Wertebereich)



2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc für Konstruktoren

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann
 - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und eventuell Wertebereich)

JavaDoc für Konstruktoren



- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann
 - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und eventuell Wertebereich)

```
Syntax
/**
 * Kurze aber aussagekräftige Beschreibung von Zweck und Kontext des Konstruktors.
 * Sonderfälle werden beschrieben.
 * @param parameter Beschreibung des Parameters
 */
KlassenName(Datentyp parameter, ...) {
```

- beschreibt Zweck der Konstruktors
- beschreibt, in welchem Kontext der Kontruktor verwendet werden kann
 - Spezialfälle, die das Verhalten betreffen
- alle Parameter werden beschrieben (Zweck und eventuell Wertebereich)

```
Syntax
/**
 * Kurze aber aussagekräftige Beschreibung von Zweck und Kontext des Konstruktors.
 * Sonderfälle werden beschrieben.
 * @param parameter Beschreibung des Parameters
 */
KlassenName(Datentyp parameter, ...) {
```

JavaDoc für Konstuktoren - Beispiel



```
/**
 * Konstruiert einen zwei dimensionalen Vektor mit zwei übergebenen Komponenten.
 *
 * @param x die x-Komponente des Vektors
 * @param y die y-Komponente des Vektors
 */
Vector2D(double x, double y) {
    this.x = x;
    this.y = y;
}
```

2023-01-18

Tutorium 14
└─ JavaDoc

└─ JavaDoc für Konstuktoren - Beispiel

```
/**
 * Konstruiert einen zwei dimensionalen Vektor mit zwei übergebenen Komponenten.
 *
 * @param x die x-Komponente des Vektors
 * @param y die y-Komponente des Vektors
 */
Vector2D(double x, double y) {
    this.x = x;
    this.y = y;
}
```

Exceptions - Einführung



Exception

- eine *Ausnahme*
- Zur Laufzeit des Programms
- Zur Unterbrechung des normalen Kontrollflusses

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
●○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Exceptions

└─ Exceptions - Einführung

Exception	
■	eine Ausnahme
■	Zur Laufzeit des Programms
■	Zur Unterbrechung des normalen Kontrollflusses

Beispiele:

1. Eine Datei nicht gefunden wird.
2. Jemand durch 0 teilt.
3. Ungültige Argumente einer Methode übergeben wurden.
4. Versucht wird auf ein nicht-existentes Arrayelement zuzugreifen.

Exceptions - Einführung

Exception

- eine *Ausnahme*
- Zur Laufzeit des Programms
- Zur Unterbrechung des normalen Kontrollflusses

Verwendung einer Exception

- Ein *Problem* tritt auf
- Normales Fortfahren nicht möglich
- Lokale Reaktion darauf nicht sinnvoll/möglich
- Behandlung des Problems an anderer Stelle nötig

Beispiele:

1. Eine Datei nicht gefunden wird.
2. Jemand durch 0 teilt.
3. Ungültige Argumente einer Methode übergeben wurden.
4. Versucht wird auf ein nicht-existentes Arrayelement zuzugreifen.

Exception
<ul style="list-style-type: none">■ eine Ausnahme■ Zur Laufzeit des Programms■ Zur Unterbrechung des normalen Kontrollflusses
Verwendung einer Exception
<ul style="list-style-type: none">■ Ein Problem tritt auf■ Normales Fortfahren nicht möglich■ Lokale Reaktion darauf nicht sinnvoll/möglich■ Behandlung des Problems an anderer Stelle nötig

Exceptions

Exceptions in Java

Ausnahme in Java

- echtes Objekt (Methoden, Attribute, ...)
- Von Klasse `Exception` abgeleitet
- Mindestens zwei Konstruktoren: Default & mit String-Parameter (mit zusätzlichen Informationen)
- Methoden: `getMessage()` & `printStackTrace()`
- Erzeugung mit **new**
- Auslösen mit **throw**

Exceptions - Beispiel



```
public void setMonth(int month) {  
    if ((month < 1) || (month > 12)) {  
        throw new IllegalArgumentException(  
            "Wrong month: " + month);  
    }  
    this.month = month;  
}
```

```
public void setMonth(int month) {  
    if ((month < 1) || (month > 12)) {  
        throw new IllegalArgumentException(  
            "Wrong month: " + month);  
    }  
    this.month = month;  
}
```

Exceptions - Arten von Fehlern



Error

(Katastrophale) Probleme, die eigentlich nicht auftreten dürfen.
Speicher voll, Illegaler Byte-Code, JVM-Fehler, ...

2023-01-18

- Tutorium 14
 - Exceptions
 - Exceptions - Arten von Fehlern

Error
(Katastrophale) Probleme, die eigentlich nicht auftreten dürfen.
Speicher voll, Illegaler Byte-Code, JVM-Fehler, ...

Exceptions - Arten von Fehlern



Error

(Katastrophale) Probleme, die eigentlich nicht auftreten dürfen.
Speicher voll, Illegaler Byte-Code, JVM-Fehler, ...

RuntimeException

Durch *fremde* Fehler erzeugte Probleme.
falsche Benutzung einer Klasse, Programmierfehler

2023-01-18

Error
(Katastrophale) Probleme, die eigentlich nicht auftreten dürfen. Speicher voll, Illegaler Byte-Code, JVM-Fehler, ...
RuntimeException
Durch fremde Fehler erzeugte Probleme. falsche Benutzung einer Klasse, Programmierfehler

Exceptions - Arten von Fehlern



Error

(Katastrophale) Probleme, die eigentlich nicht auftreten dürfen.
Speicher voll, Illegaler Byte-Code, JVM-Fehler, ...

RuntimeException

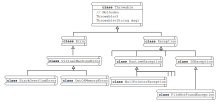
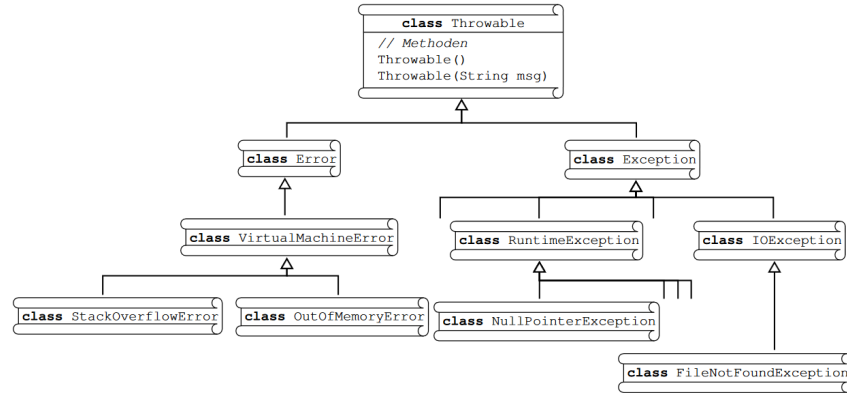
Durch *fremde* Fehler erzeugte Probleme.
falsche Benutzung einer Klasse, Programmierfehler

Geprüfte Exception (*checked Exception*)

Vorhersehbare und behandelbare Fehler.
Datei nicht vorhanden, Festplatte voll, Fehler beim Parsen, ...

Error
(Katastrophale) Probleme, die eigentlich nicht auftreten dürfen. Speicher voll, Illegaler Byte-Code, JVM-Fehler, ...
RuntimeException
Durch fremde Fehler erzeugte Probleme. falsche Benutzung einer Klasse, Programmierfehler
Geprüfte Exception (checked Exception)
Vorhersehbare und behandelbare Fehler. Datei nicht vorhanden, Festplatte voll, Fehler beim Parsen, ...

Exceptions - Hierarchie



Exceptions



Ausnahmebehandlung in Java

```
try {  
    // hier koennte eine Exception auftreten  
} catch (ExceptionType1 e) {  
    // Fehlerbehandlung fuer ExceptionType1  
} catch (ExceptionType2 e) {  
    // Fehlerbehandlung fuer ExceptionType2  
}
```

```
Ausnahmebehandlung in Java  
try {  
    // hier koennte eine Exception auftreten  
} catch (ExceptionType1 e) {  
    // Fehlerbehandlung fuer ExceptionType1  
} catch (ExceptionType2 e) {  
    // Fehlerbehandlung fuer ExceptionType2  
}
```

Exceptions

Ausnahmebehandlung in Java

```
try {  
    // hier koennte eine Exception auftreten  
} catch (ExceptionType1 e) {  
    // Fehlerbehandlung fuer ExceptionType1  
} catch (ExceptionType2 e) {  
    // Fehlerbehandlung fuer ExceptionType2  
}
```

Fall-through, die Zweite

- Java ruft den **ersten** passenden catch Block auf!
- Alle weiteren werden *ignoriert*

Ausnahmebehandlung in Java

```
try {  
    // hier koennte eine Exception auftreten  
} catch (ExceptionType1 e) {  
    // Fehlerbehandlung fuer ExceptionType1  
} catch (ExceptionType2 e) {  
    // Fehlerbehandlung fuer ExceptionType2  
}
```

Fall-through, die Zweite

- Java ruft den **ersten** passenden catch Block auf!
- Alle weiteren werden ignoriert

Exceptions

Beispiel

```
try {  
    FileReader fr = new FileReader(".test");  
    int nextChar = fr.read();  
    while (nextChar != -1) {  
        nextChar = fr.read();  
    }  
} catch (FileNotFoundException e) {  
    System.out.println("Nicht gefunden.");  
} catch (IOException e) {  
    System.out.println("Ooops.");  
}
```

```
Beispiel  
try {  
    FileReader fr = new FileReader(".test");  
    int nextChar = fr.read();  
    while (nextChar != -1) {  
        nextChar = fr.read();  
    }  
} catch (FileNotFoundException e) {  
    System.out.println("Nicht gefunden.");  
} catch (IOException e) {  
    System.out.println("Ooops.");  
}
```

Exceptions



Exception Handler (= catch-Block)

- Behandlung einer Ausnahme
- an einer Stelle
- irgendwo im Aufrufstack
- getrennt von normalen Programmcode

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○●○○○○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Exceptions

└─ Exceptions

Exception Handler (= catch-Block)

- Behandlung einer Ausnahme
- an einer Stelle
- irgendwo im Aufrufstack
- getrennt von normalen Programmcode

Exceptions

Exception Handler (= catch-Block)

- Behandlung einer Ausnahme
- an einer Stelle
- irgendwo im Aufrufstack
- getrennt von normalen Programmcode

Catch or specify

Jede ausgelöste geprüfte (checked) Exception muss

- behandelt (Exception Handler) oder
- deklariert (throws)

werden.

Exceptions



Deklaration von Ausnahmen

- Deklaration im Methodenkopf:
`private String readFile(String filename) throws IOException, FileNotFoundException {}`

Deklaration von Ausnahmen

■ Deklaration im Methodenkopf:
`private String readFile(String filename) throws IOException, FileNotFoundException {}`

Exceptions

Deklaration von Ausnahmen

- Deklaration im Methodenkopf:
`private String readFile(String filename) throws IOException, FileNotFoundException {}`
- Aufrufer muss sich um Exception kümmern
- `throws` ist Teil der Signatur (Vorsicht beim Überschreiben)
Exceptions können in überschriebenen Methoden weggelassen werden, aber nicht hinzukommen.
- Nicht deklarationspflichtig sind `RuntimeException` & `Error` (sowie deren Unterklassen)
- Jede Exception mittels `@throws` im Javadoc beschrieben werden

Exceptions

Deklaration von Ausnahmen

- Deklaration im Methodenkopf:
`private String readFile(String filename) throws IOException, FileNotFoundException {}`
- Aufrufer muss sich um Exception kümmern
- `throws` ist Teil der Signatur (Vorsicht beim Überschreiben)
Exceptions können in überschriebenen Methoden weggelassen werden, aber nicht hinzukommen.
- Nicht deklarationspflichtig sind `RuntimeException` & `Error` (sowie deren Unterklassen)
- Jede Exception mittels `@throws` im Javadoc beschrieben werden

Anmerkung: Da `IOException` Oberklasse von `FileNotFoundException` ist, müsste letzteres nicht extra deklariert werden. Dokumentationszwecke!

Exceptions



Ort der Behandlung

Finden der passenden Ausnahmebehandlung:

- Suche im Aufrufstack nach umgebenden try-catch-Blöcken, gehe zu erstem passenden catch-Block
- Nach der Behandlung: Fortsetzung am Ende des try-catch-Block

Exceptions - Konventionen



Behandlung?

- **Error und Unterklassen:** Nein, nicht sinnvoll behandelbar
- **Exception:** Nein, viel zu allgemein
- **RuntimeException:** Prinzipiell Nein
- **dessen Unterklassen:** Programmierfehler beheben! (Ausnahme: `NumberFormatException`)
- **Andere:** Ja, wenn sinnvoll behandelbar
- try-Block so klein wie möglich halten

Exceptions - Konventionen



Werfen?

- **Error:** Nein.
- **Exception:** Niemals, nur als eigene Unterklasse
- **RuntimeException:** Ja, eigene (semantisch passende) Unterklasse

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○●○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Exceptions

└─ Exceptions - Konventionen

Werfen?
■ **Error:** Nein.
■ **Exception:** Niemals, nur als eigene Unterklasse
■ **RuntimeException:** Ja, eigene (semantisch passende) Unterklasse

Exceptions - Konventionen

Werfen?

- **Error:** Nein.
- **Exception:** Niemals, nur als eigene Unterklasse
- **RuntimeException:** Ja, eigene (semantisch passende) Unterklasse

Beispiel

```
if ((month < 1) || (month > 12)) {  
    throw new IllegalArgumentException(  
        "Wrong month: %s", month);  
}  
switch (month) {  
    case 1: break; // ...  
    default: throw new Error();  
}
```

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○●○○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Exceptions

└─ Exceptions - Konventionen

```
Werfen?  
■ Error: Nein.  
■ Exception: Niemals, nur als eigene Unterklasse  
■ RuntimeException: Ja, eigene (semantisch passende) Unterklasse  
  
Beispiel  
if ((month < 1) || (month > 12)) {  
    throw new IllegalArgumentException(  
        "Wrong month: %s", month);  
}  
switch (month) {  
    case 1: break; // ...  
    default: throw new Error();  
}
```

Exceptions - Konventionen



Verwendung

Exceptions sollen:

- zur Vereinfachung dienen
- die absolute Ausnahme darstellen
- mittels @throws im Javadoc beschrieben werden
- NICHT den normalen Kontrollfluss steuern

Wiederholung
○○○

Übungsblatt 4
○○○○○

Javadoc
○○○○○

Exceptions
○○○○○○○○○○●○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Exceptions

└─ Exceptions - Konventionen

Verwendung
Exceptions sollen:
■ zur Vereinfachung dienen
■ die absolute Ausnahme darstellen
■ mittels @throws im Javadoc beschrieben werden
■ NICHT den normalen Kontrollfluss steuern

Exceptions - Konventionen

Verwendung

Exceptions sollen:

- zur Vereinfachung dienen
- die absolute Ausnahme darstellen
- mittels @throws im Javadoc beschrieben werden
- NICHT den normalen Kontrollfluss steuern

Böse!

```
try {  
    while (character != array[i]) { i++; }  
} catch (Exception e) {  
    System.out.println("Element nicht gefunden.");  
}
```

Wiederholung
○○○

Übungsblatt 4
○○○○○

Javadoc
○○○○○

Exceptions
○○○○○○○○○○●○○○

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

Verwendung

Exceptions sollen:

- zur Vereinfachung dienen
- die absolute Ausnahme darstellen
- mittels @throws im Javadoc beschrieben werden
- NICHT den normalen Kontrollfluss steuern

Böse!

```
try {  
    while (character != array[i]) { i++; }  
} catch (Exception e) {  
    System.out.println("Element nicht gefunden.");  
}
```

Exceptions - Konventionen



Verboten!

- try-Block um das ganze Programm
- Leerer catch-Block
- Explizites Fangen des Typs Exception
- Explizites Fangen des Typs Throwable

Verboten!

- try-Block um das ganze Programm
- Leerer catch-Block
- Explizites Fangen des Typs Exception
- Explizites Fangen des Typs Throwable

Exceptions



Eigene Exceptions

- Ableiten einer eigenen Unterklasse von Exception oder RuntimeException
- Implementierung der zwei Standard-Konstruktoren
- Definition einer eigenen, sinnvollen Exception-Hierarchie (bei Bedarf)
- Verwendung von vorhandenen Exceptions nur für dafür vorgesehene Zwecke (Javadoc anschauen)

Exceptions

Eigene Exceptions

- Ableiten einer eigenen Unterklasse von Exception oder RuntimeException
- Implementierung der zwei Standard-Konstruktoren
- Definition einer eigenen, sinnvollen Exception-Hierarchie (bei Bedarf)
- Verwendung von vorhandenen Exceptions nur für dafür vorgesehene Zwecke (Javadoc anschauen)

Beispiele in der Java-API

- IllegalArgumentException
- IllegalStateException
- UnsupportedOperationException
- NullPointerException

Exceptions - Zusammenfassung



Ausnahmen

- werden ausgelöst (throw) und behandelt (try-catch) oder
- deklariert (throws)
- sollen die Ausnahme bleiben
- trennen sauber Programmlogik und Fehlerbehandlung

Exceptions - Zusammenfassung



Ausnahmen

- werden ausgelöst (throw) und behandelt (try-catch) oder
- deklariert (throws)
- sollen die Ausnahme bleiben
- trennen sauber Programmlogik und Fehlerbehandlung

Fehlererkennung

- so früh wie möglich
- defensiv
- mittels if Exceptions

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○●

Java-API
○○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Exceptions

└─ Exceptions - Zusammenfassung

Ausnahmen
<ul style="list-style-type: none">■ werden ausgelöst (throw) und behandelt (try-catch) oder■ deklariert (throws)■ sollen die Ausnahme bleiben■ trennen sauber Programmlogik und Fehlerbehandlung
Fehlererkennung
<ul style="list-style-type: none">■ so früh wie möglich■ defensiv■ mittels if Exceptions

Java-API - Application Programming Interface

- Sammlung von Klassen und Paketen
- Enthalten häufig verwendete Funktionalität



2023-01-18

Tutorium 14
└─ Java-API

└─ Java-API - Application Programming Interface

■ Sammlung von Klassen und Paketen
■ Enthalten häufig verwendete Funktionalität

1. Klassen in java.lang: Object, String, Math, Enum, ...

Java-API - Application Programming Interface



- Sammlung von Klassen und Paketen
- Enthalten häufig verwendete Funktionalität

Bekannte Klassen

- Object
- String, Math, Enum
- Comparable, Integer, Double, ...

2023-01-18
Tutorium 14
└─ Java-API

└─ Java-API - Application Programming Interface

- Sammlung von Klassen und Paketen
- Enthalten häufig verwendete Funktionalität

Bekannte Klassen

- Object
- String, Math, Enum
- Comparable, Integer, Double, ...

1. Klassen in java.lang: Object, String, Math, Enum, ...

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
●○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○

Java-API - Application Programming Interface

- Sammlung von Klassen und Paketen
- Enthalten häufig verwendete Funktionalität

Bekannte Klassen

- Object
- String, Math, Enum
- Comparable, Integer, Double, ...

Pakete

- `java.lang` Basisfunktionen (oben genannte Klassen)
- `java.util` Java Collections, Zeit-/Datumsfunktionen
- `java.io` Ein-/Ausgabe

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
●○○○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○

1. Klassen in `java.lang`: Object, String, Math, Enum, ...

Java-API - Collection



Das Interface `java.util.Collection<E>`

- `java.util.List`
- `java.util.Set`
- `java.util.SortedSet`
- `java.util.Queue`

2023-01-18
Tutorium 14
└─ Java-API
 └─ Java-API - Collection



1. Die unten stehenden Interfaces erben vom Collection-Interface
2. List → Liste mit implementierenden ArrayList, LinkedList, Stack...
3. Set → Menge mit implementierenden Klassen HashSet, TreeSet, ...
4. SortedSet → sortierte Menge (nach compareTo wird sortiert) mit implementierenden Klassen TreeSet
5. Queues → Warteschlangen, PriorityQueue, sind aber absolut unwichtig
6. HashSet → Elemente werden gehasht (int-Wert bestimmen) und in einen Eimer mit demselben Hash gepackt, in diesem Eimer wird nochmal geprüft, ob Objekt schon enthalten ist (equals-Methode)
7. TreeSet → Elemente werden in einem binären-Suchbaum gespeichert (Rot-Schwarz-Baum, ist egal was das ist), damit man leicht auf einzelne Elemente zugreifen kann (Suchen kann), und nicht durch den gesamten Baum iterieren muss

Java-API - Collection



Das Interface `java.util.Collection<E>`

- `java.util.List`
- `java.util.Set`
- `java.util.SortedSet`
- `java.util.Queue`

Wichtige Methoden

- `boolean contains(Object element)`
- `boolean add(E element)`
- `boolean remove(Object element)`



1. Die unten stehenden Interfaces erben vom Collection-Interface
2. List → Liste mit implementierenden ArrayList, LinkedList, Stack...
3. Set → Menge mit implementierenden Klassen HashSet, TreeSet, ...
4. SortedSet → sortierte Menge (nach compareTo wird sortiert) mit implementierenden Klassen TreeSet
5. Queues → Warteschlangen, PriorityQueue, sind aber absolut unwichtig
6. HashSet → Elemente werden gehasht (int-Wert bestimmen) und in einen Eimer mit demselben Hash gepackt, in diesem Eimer wird nochmal geprüft, ob Objekt schon enthalten ist (equals-Methode)
7. TreeSet → Elemente werden in einem binären-Suchbaum gespeichert (Rot-Schwarz-Baum, ist egal was das ist), damit man leicht auf einzelne Elemente zugreifen kann (Suchen kann), und nicht durch den gesamten Baum iterieren muss

Java-API - Map



Das Interface `java.util.Map<K, V>`

- Paare werden eindeutig nach ihrem Key gespeichert
- Ein Key darf nur einmal enthalten sein

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○●○○○○○○

OO-Design-Prinzipien
○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Java-API

└─ Java-API - Map

Das Interface `java.util.Map<K, V>`

- Paare werden eindeutig nach ihrem Key gespeichert
- Ein Key darf nur einmal enthalten sein

1. Datenstruktur, die einem Schlüssel und Werte-Paare speichert → Schlüssel darf nur einmal in der Menge enthalten sein; Funktion, die von K nach V abbildet/mapt
2. put: the previous value associated with key, or null if there was no mapping for key.
3. Es gibt die Implementierungen HashMap (keys werden gehasht) und TreeMap (nach Keys sortierte Speicherung im Baum nach der compareTo-Methode)

Java-API - Map

Das Interface `java.util.Map<K, V>`

- Paare werden eindeutig nach ihrem Key gespeichert
- Ein Key darf nur einmal enthalten sein

Wichtige Methoden

- `V put(K key, V value)`
- `V get(Object key)`
- `V remove(Object key)`
- `boolean containsKey(Object key)`
- `boolean containsValue(Object value)`

1. Datenstruktur, die einem Schlüssel und Werte-Paare speichert → Schlüssel darf nur einmal in der Menge enthalten sein; Funktion, die von K nach V abbildet/mapt
2. put: the previous value associated with key, or null if there was no mapping for key.
3. Es gibt die Implementierungen HashMap (keys werden gehasht) und TreeMap (nach Keys sortierte Speicherung im Baum nach der compareTo-Methode)

Java-API - Map



Beispiel

```
public static void main(String[] args) {  
    Map<Integer, Integer> intMap = new HashMap<Integer, Integer>();  
    intMap.put(5, 20);  
    intMap.put(10, 30);  
    if (intMap.containsKey(5)) {  
        System.out.println(intMap.get(5)); // 20  
    }  
}
```

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
ooo	ooooo	ooooo	ooooooooooooooooo	ooo●ooooo	ooooooooooooo

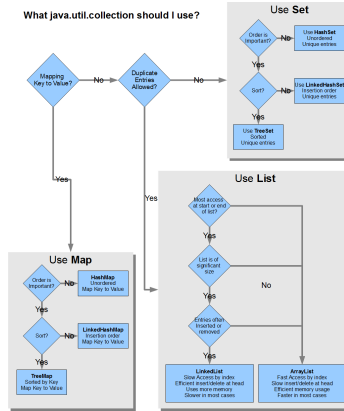
2023-01-18

Tutorium 14
└─Java-API
└─Java-API - Map

Beispiel

```
public static void main(String[] args) {  
    Map<Integer, Integer> intMap = new HashMap<Integer, Integer>();  
    intMap.put(5, 20);  
    intMap.put(10, 30);  
    if (intMap.containsKey(5)) {  
        System.out.println(intMap.get(5)); // 20  
    }  
}
```

Java-API - Welche Collection nehme ich wann?



Quelle: <https://i.stack.imgur.com/aSDsG.png>

Java-API - Welche Collection nehme ich wann?



Quelle: <https://i.stack.imgur.com/aSDsG.png>

Java-API - Reguläre Ausdrücke



java.util.regex und seine Klassen

- Pattern (java.util.regex.Pattern)
- Matcher (java.util.regex.Matcher)

2023-01-18

Tutorium 14
└─ Java-API

└─ Java-API - Reguläre Ausdrücke

java.util.regex und seine Klassen
■ Pattern (java.util.regex.Pattern)
■ Matcher (java.util.regex.Matcher)

1. Backslash zum escapen des Backslashes, + ist selbst Quantor und muss deshalb auch mit Backslash angegeben werden

Java-API - Reguläre Ausdrücke



java.util.regex und seine Klassen

- Pattern (java.util.regex.Pattern)
- Matcher (java.util.regex.Matcher)

```
String positiveIntegerPattern = "\\+?[1-9]\\d*";
Pattern pattern = Pattern.compile(positiveIntegerPattern);
Matcher matcher = pattern.matcher("+529");
boolean isMatch = matcher.matches(); // true
```

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○●○○○

OO-Design-Prinzipien
○○○○○○○○○○○

2023-01-18

Tutorium 14

└─ Java-API

└─ Java-API - Reguläre Ausdrücke

```
java.util.regex und seine Klassen
■ Pattern (java.util.regex.Pattern)
■ Matcher (java.util.regex.Matcher)

String positiveIntegerPattern = "\\+?[1-9]\\d*";
Pattern pattern = Pattern.compile(positiveIntegerPattern);
Matcher matcher = pattern.matcher("+529");
boolean isMatch = matcher.matches(); // true
```

1. Backslash zum escapen des Backslashes, + ist selbst Quantor und muss deshalb auch mit Backslash angegeben werden

Java-API - Reguläre Ausdrücke



Reguläre Ausdrücke werden wie in GBI gebildet

Character

- Klassen
 - \d Ziffer
 - \w Zeichen
 - [a-g] Ein Zeichen von a-g, [1-9] Ziffer von 1-9
- Quantoren
 - * Ausdruck davor 0 mal oder mehr, + Ausdruck davor mindestens 1 mal
 - ? Ausdruck davor 0 oder 1 mal
 - a{5} 5 mal a

Reguläre Ausdrücke testen: <https://regexr.com/>

2023-01-18

Tutorium 14
└─ Java-API

└─ Java-API - Reguläre Ausdrücke

Reguläre Ausdrücke werden wie in GBI gebildet

- ### Character
- Klassen
 - \d Ziffer
 - \w Zeichen
 - [a-g] Ein Zeichen von a-g, [1-9] Ziffer von 1-9
 - Quantoren
 - * Ausdruck davor 0 mal oder mehr, + Ausdruck davor mindestens 1 mal
 - ? Ausdruck davor 0 oder 1 mal
 - a{5} 5 mal a

Reguläre Ausdrücke testen: <https://regexr.com/>

1. Dazu sagen, dass in Java mehrere Backslashes nötig sind, um beispielsweise das d oder das + dort unterzubringen

Aufgabe



Pattern

Schreibe jeweils für die folgenden Muster einen regulären Ausdruck, sodass diese mit Hilfe der Klasse `Pattern` von Java erkannt werden würden.

Wiederholung ○○○	Übungsblatt 4 ○○○○○	JavaDoc ○○○○○	Exceptions ○○○○○○○○○○○○○○○○	Java-API ○○○○○○○●○	OO-Design-Prinzipien ○○○○○○○○○○○
---------------------	------------------------	------------------	--------------------------------	-----------------------	-------------------------------------

2023-01-18

Tutorium 14

└─Java-API

└─Aufgabe

Pattern

Schreibe jeweils für die folgenden Muster einen regulären Ausdruck, sodass diese mit Hilfe der Klasse `Pattern` von Java erkannt werden würden.

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



5-stellige positive Ganzzahl (ohne +)

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
○○○	○○○○○	○○○○○	○○○○○○○○○○○○○○○○	○○○○○○○●○	○○○○○○○○○○○

2023-01-18

- Tutorium 14
 - Java-API
 - Aufgabe

5-stellige positive Ganzzahl (ohne +)

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



5-stellige positive Ganzzahl (ohne +)

`\d{5}`

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
○○○	○○○○○	○○○○○	○○○○○○○○○○○○○○○○	○○○○○○○●○	○○○○○○○○○○○

2023-01-18

- Tutorium 14
 - Java-API
 - Aufgabe

5-stellige positive Ganzzahl (ohne +) \d{5}

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



5-stellige positive Ganzzahl (ohne +)
Ganzzahl (mit + und -)

`\d{5}`

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



5-stellige positive Ganzzahl (ohne +)
Ganzzahl (mit + und -)

`\d{5}`
`(+|-)?\d*`

2023-01-18

5-stellige positive Ganzzahl (ohne +)
Ganzzahl (mit + und -)

`\d{5}`
`(+|-)?\d*`

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



5-stellige positive Ganzzahl (ohne +) `\d{5}`
Ganzzahl (mit + und -) `(+|-)?\d*`
kleingeschriebenes Wort, anschließend Doppelpunkt

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



2023-01-18

Tutorium 14
└─ Java-API

└─ Aufgabe

5-stellige positive Ganzzahl (ohne +) `\d{5}`
Ganzzahl (mit + und -) `(+|-)?\d*`
kleingeschriebenes Wort, anschließend Doppelpunkt `\w+:`

5-stellige positive Ganzzahl (ohne +) `\d{5}`
Ganzzahl (mit + und -) `(+|-)?\d*`
kleingeschriebenes Wort, anschließend Doppelpunkt `\w+:`

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



2023-01-18

Tutorium 14
└─ Java-API

└─ Aufgabe

5-stellige positive Ganzzahl (ohne +) `\d{5}`
Ganzzahl (mit + und -) `(+|-)?\d*`
kleingeschriebenes Wort, anschließend Doppelpunkt `\w+:`
positive Kommazahl (mit + und -)

- 5-stellige positive Ganzzahl (ohne +) `\d{5}`
- Ganzzahl (mit + und -) `(+|-)?\d*`
- kleingeschriebenes Wort, anschließend Doppelpunkt `\w+:`
- positive Kommazahl (mit + und -)

1. Capture Groups eventuell im Kopf haben bei Fragen

Aufgabe



2023-01-18

Tutorium 14
└─ Java-API

└─ Aufgabe

5-stellige positive Ganzzahl (ohne +) \d{5}
Ganzzahl (mit + und -) (+|-)?\d*
kleingeschriebenes Wort, anschließend Doppelpunkt \w+:
positive Kommazahl (mit + und -) (+|-)?\d+(\.|\d+)?

5-stellige positive Ganzzahl (ohne +) \d{5}
Ganzzahl (mit + und -) (+|-)?\d*
kleingeschriebenes Wort, anschließend Doppelpunkt \w+:
positive Kommazahl (mit + und -) (+|-)?\d+(\.|\d+)?

1. Capture Groups eventuell im Kopf haben bei Fragen

Java-API - Wichtige Links



■ Allgemein

<https://docs.oracle.com/en/java/javase/11/docs/api/index.html>

■ java.util.Collection<E>

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collection.html>

■ java.util.Map<K, V>

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Map.html>

■ java.util.regex

[https://docs.oracle.com/en/java/javase/11/docs/api/
java.base/java/util/regex/package-summary.html](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/regex/package-summary.html)

Wiederholung
○○○

Übungsblatt 4
○○○○○

JavaDoc
○○○○○

Exceptions
○○○○○○○○○○○○○○○○

Java-API
○○○○○○○○●

OO-Design-Prinzipien
○○○○○○○○○○○

Objektorientierte Design-Prinzipien

OO-Design-Prinzipien



- 1 Datenkapselung
- 2 Komposition vor Vererbung
- 3 Gegen Schnittstellen programmieren
- 4 SOLID-Prinzipien
- 5 Guter Schnittstellenentwurf

Wiederholung	Übungsblatt 4	JavaDoc	Exceptions	Java-API	OO-Design-Prinzipien
○○○	○○○○○	○○○○○	○○○○○○○○○○○○○○○○	○○○○○○○○○	●○○○○○○○○○

2023-01-18

Tutorium 14

- OO-Design-Prinzipien
 - OO-Design-Prinzipien

- 1 Datenkapselung
- 2 Komposition vor Vererbung
- 3 Gegen Schnittstellen programmieren
- 4 SOLID-Prinzipien
- 5 Guter Schnittstellenentwurf

Datenkapselung



- private Attribute
- Entwurfsentscheidung hinter Schnittstelle verdecken
- Keine Objektattribute in gettern direkt zurückgeben
 - Kopie von Collection erstellen und diese zurückgeben

- private Attribute
- Entwurfsentscheidung hinter Schnittstelle verdecken
- Keine Objektattribute in gettern direkt zurückgeben
 - Kopie von Collection erstellen und diese zurückgeben

Komposition vor Vererbung

- Komposition
 - Verwendung von Klassen als Attributtypen
 - Bestehende Objekte werden zusammengesetzt, um Funktionalität zu erweitern



2023-01-18

- Komposition
 - Verwendung von Klassen als Attributtypen
 - Bestehende Objekte werden zusammengesetzt, um Funktionalität zu erweitern

Komposition vor Vererbung



- Komposition
 - Verwendung von Klassen als Attributtypen
 - Bestehende Objekte werden zusammengesetzt, um Funktionalität zu erweitern
- Vererbung
 - Ableitung von Eigenschaften und Funktionalität
 - Implementierung wird erweitert

2023-01-18

- Komposition
 - Verwendung von Klassen als Attributtypen
 - Bestehende Objekte werden zusammengesetzt, um Funktionalität zu erweitern
- Vererbung
 - Ableitung von Eigenschaften und Funktionalität
 - Implementierung wird erweitert

Komposition vor Vererbung



- Komposition
 - Verwendung von Klassen als Attributtypen
 - Bestehende Objekte werden zusammengesetzt, um Funktionalität zu erweitern
- Vererbung
 - Ableitung von Eigenschaften und Funktionalität
 - Implementierung wird erweitert
- Komposition sollte vor Vererbung genutzt werden, insofern das möglich ist
- Beispiel Klasse `AccessControl`
 - Attribut vom Typ `Map` zur Nutzerverwaltung
 - Man könnte auch `AccessControl` von `HashMap` ableiten und somit die Funktionalität zur Verfügung stellen
 - Ist aber unübersichtlich und verkompliziert den Code
 - Verletzt außerdem die is-a-Beziehung: `AccessControl` ist keine `HashMap`

2023-01-18

- Komposition
 - Verwendung von Klassen als Attributtypen
 - Bestehende Objekte werden zusammengesetzt, um Funktionalität zu erweitern
- Vererbung
 - Ableitung von Eigenschaften und Funktionalität
 - Implementierung wird erweitert
- Komposition sollte vor Vererbung genutzt werden, insofern das möglich ist
- Beispiel Klasse `AccessControl`
 - Attribut vom Typ `Map` zur Nutzerverwaltung
 - Man könnte auch `AccessControl` von `HashMap` ableiten und somit die Funktionalität zur Verfügung stellen
 - Ist aber unübersichtlich und verkompliziert den Code
 - Verletzt außerdem die is-a-Beziehung: `AccessControl` ist keine `HashMap`

Gegen Schnittstellen programmieren



Nicht gegen die Implementierung programmieren!

- Leichte Austauschbarkeit
- Bessere Wiederverwendbarkeit

```
public int getStretchedCollectionSize(Collection<?> collection) {  
    return collection.size() * this.factor;  
}
```

2023-01-18

Nicht gegen die Implementierung programmieren!

- Leichte Austauschbarkeit
- Bessere Wiederverwendbarkeit

```
public int getStretchedCollectionSize(Collection<?> collection) {  
    return collection.size() * this.factor;  
}
```

SOLID-Prinzipien



- Jede Klasse sollte eine Verantwortung haben
- Open-Close Prinzip (nächste Folie)
- Eine Instanz einer Unterklasse sollte sich so verhalten, dass sie jederzeit auch als Instanz der Oberklasse verwendet werden könnte
- Klassen sollten durch ein Interface nicht unnötige Methoden implementieren müssen ⇒ Interface aufteilen
- Abstraktes darf nicht von Details abhängen

2023-01-18

- Jede Klasse sollte eine Verantwortung haben
- Open-Close Prinzip (nächste Folie)
- Eine Instanz einer Unterklasse sollte sich so verhalten, dass sie jederzeit auch als Instanz der Oberklasse verwendet werden könnte
- Klassen sollten durch ein Interface nicht unnötige Methoden implementieren müssen ⇒ Interface aufteilen
- Abstraktes darf nicht von Details abhängen

Abstraktes darf nicht von Details abhängen



Schlecht

```
public class UserDatabase {  
    LinkedList<User> users;  
  
    public UserDatabase() {  
        this.users = new LinkedList<>();  
    }  
}
```

2023-01-18

Tutorium 14
└─ OO-Design-Prinzipien

└─ Abstraktes darf nicht von Details abhängen

Schlecht
public class UserDatabase {
 LinkedList<User> users;

 public UserDatabase() {
 this.users = new LinkedList<>();
 }
}

- 1. Code muss nur an einer Stelle geändert werden, wenn man die Datenstruktur ändern möchte (LinkedList zu ArrayList bspw.)

Abstraktes darf nicht von Details abhängen



Schlecht

```
public class UserDatabase {  
    LinkedList<User> users;  
  
    public UserDatabase() {  
        this.users = new LinkedList<>();  
    }  
}
```

Gut

```
public class UserDatabase {  
    List<User> users;  
  
    public UserDatabase() {  
        this.users = new LinkedList<>();  
    }  
}
```

Schlecht public class UserDatabase { LinkedList<User> users; public UserDatabase() { this.users = new LinkedList<>(); } }	Gut public class UserDatabase { List<User> users; public UserDatabase() { this.users = new LinkedList<>(); } }
--	---

1. Code muss nur an einer Stelle geändert werden, wenn man die Datenstruktur ändern möchte (LinkedList zu ArrayList bspw.)

Open-Close Prinzip



- Offen für Erweiterung ⇔ Geschlossen für Veränderung
- Programmmodule sollen nicht verändert werden müssen
- Erweiterbaren Code schreiben
- Code soll nicht verändert werden

Schnittstellenentwurf



- Implementierungsdetails verbergen
- Konsistentes und angemessenen Abstraktionsniveau
- Generelle Schnittstellen verwenden (`void addUsers(Collection<User> users)`)

■ Implementierungsdetails verbergen
■ Konsistentes und angemessenen Abstraktionsniveau
■ Generelle Schnittstellen verwenden (`void addUsers(Collection<User> users)`)

Weitere Prinzipien



- Niemals gleichzeitig in einer Methode Objekt-Zustand zurückgeben und ändern
- **Idempotente** Methoden schreiben (Hintereinanderausführung liefert konsistente Ergebnisse)
- Richtige Methodennamen wählen (Beschreiben das Verhalten der Methode und sind konsistent)

- Niemals gleichzeitig in einer Methode Objekt-Zustand zurückgeben und ändern
- **Idempotente** Methoden schreiben (Hintereinanderausführung liefert konsistente Ergebnisse)
- Richtige Methodennamen wählen (Beschreiben das Verhalten der Methode und sind konsistent)

1. konsistente Methodennamen: add, remove, first, last usw. immer weiter verwenden

Bis zum nächsten Tutorium am 25.01.2022.