

Praktikum „SEP: Java-Programmierung“

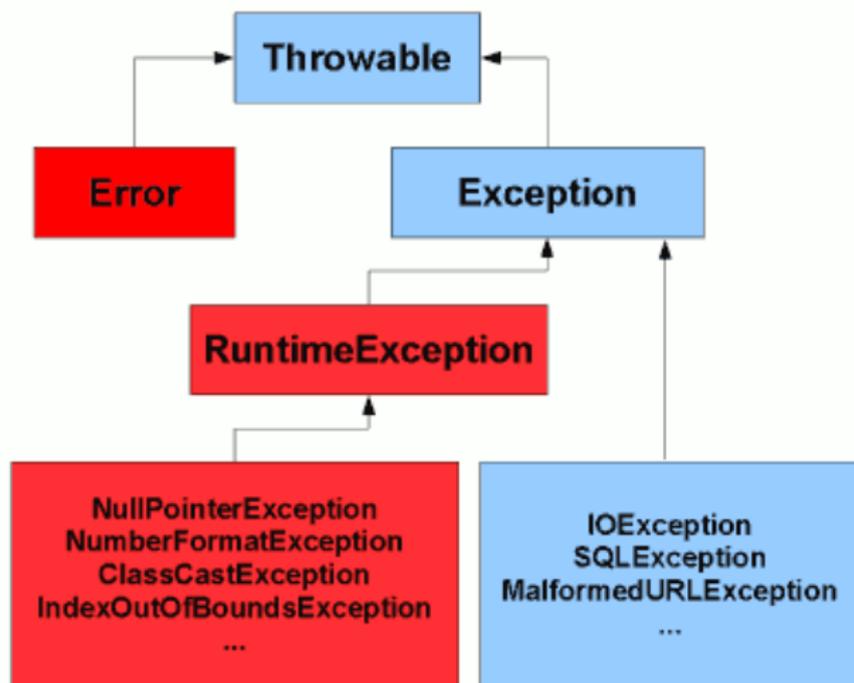
WS 2018/19

Exceptions

Thomas Lemberger und Matthias Spießl

Exception-Hierarchie in Java

- ▶ Exceptions sind spezielle Java-Objekte
- ▶ Rot: *unchecked* Exceptions
- ▶ Alles andere: *checked* Exceptions
- ▶ Alle Exceptions beschreiben **unerwartetes** Verhalten



Exceptions: Unchecked und Checked

unchecked

Meist keine sinnvolle Reaktion möglich

Kann „überall“ und „jederzeit“ auftreten

Nicht deklariert, kann immer passieren

Muss nicht behandelt werden (aber darf)

`NumberFormatException`,
`OutOfMemoryException`,
`IllegalArgumentException`,
`NullPointerException`

checked

Sinnvolle Reaktion auf Ausnahme kann erwartet werden

Tritt unter festgelegten, wohl definierten Umständen auf

Deklariert & erklärt

Muss behandelt werden

`FileNotFoundException`,
`TimeoutException`,
`IOException`

- ▶ Exceptions **nur** zum Signalisieren und Behandeln von Ausnahmen verwenden, und **nie** zum Lenken des normalen Kontrollfluss!
 1. Bestehende Exceptions behandeln
 2. Selbst Exceptions werfen/verwenden
 3. Exceptions dokumentieren

Exceptions behandeln

► Drei Möglichkeiten:

1. Propagieren
2. catch und behandeln
3. catch und abstraktere Exception propagieren

```
public void handleAdd(String user, String pointsArgument) throws IOException {  
    int points;  
    try {  
        points = Integer.parseInt(pointsArgument);  
    } catch (NumberFormatException e) {  
        log(m: "Invalid points argument: " + pointsArgument + " - using points = 0");  
        points = 0;  
    }  
  
    addEntry(user, points); // may throw IOException (only in this example!)  
}
```

Propagieren

Catch u. behandeln

```
try {  
    points = Integer.parseInt(pointsArgument);  
} catch (NumberFormatException e) {  
    throw new InvalidInputException("Invalid input for points: " + pointsArgument, e);  
}
```

Abstraktere Exception

Exceptions propagieren

- ▶ Wenn unchecked Exception aus main propagiert wird:
 - ▶ Java-VM terminiert
 - ▶ Stack-Trace wird auf stderr ausgegeben
 - ▶ Exception message & causes werden gezeigt, falls existieren

Exception Message

```
Exception in thread "main" com.hig.trie.Shell$InvalidInputException: Invalid input for points: 19.5
    at com.hig.trie.Shell.main(Shell.java:32)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
Cause \ at java.base/java.lang.reflect.Method.invoke(Method.java:566)
Caused by: java.lang.NumberFormatException: For input string: "19.5"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at com.hig.trie.Shell.main(Shell.java:30)
    ... 5 more
```

Exception catch

- ▶ Verbotene Pattern beim Behandeln von Exceptions:

```
try {  
    int points = Integer.parseInt(pointsArgument);  
} catch (NumberFormatException e) { // swallow  
}  
  
try {  
    int points = Integer.parseInt(pointsArgument);  
} catch (NumberFormatException e) { // introduce noise  
    log(m: "Error! " + e.getMessage());  
    throw e;  
}
```

1. Erste Entscheidung: Checked oder unchecked?

- ▶ Wird erwartet, dass der Grund für die Exception sinnvoll behandelt werden kann?
 - ▶ Ungültiger User-Input
 - ⇒ Fehlermeldung und weiter mit neuer Eingabe ✓
 - ▶ User-Input kann nicht gelesen werden
 - ⇒ ✗ Fehler im System oder der Programmierung

2. Welche konkrete Exception werfen?

- ▶ **Niemals** nichtssagende Exception oder RuntimeException „Exception“ << „FileNotFoundException“
- ▶ **Niemals** einen Error selber werfen, außer AssertionError
- ▶ Existierende Exception verwenden, falls Semantik passt
- ▶ Eigene Exception-Klasse erstellen, falls nicht

Eigene Exception-Klasse

- ▶ Sinnvolle Oberklasse wählen.
checked Exception: Exception
unchecked Exception: RuntimeException
- ▶ Standard: Konstruktoren der Oberklasse implementieren

```
public class InvalidInputException extends Exception {  
    public InvalidInputException() {  
    }  
  
    public InvalidInputException(String message) {  
        super(message);  
    }  
  
    public InvalidInputException(String message, Throwable cause) {  
        super(message, cause);  
    }  
  
    public InvalidInputException(Throwable cause) {  
        super(cause);  
    }  
}
```

Eigene Exception-Klasse

- ▶ Erweitert: Informationen für Exception-Handling in Exception speichern

```
public class InvalidInputException extends Exception {  
    private final String failingInput;  
  
    public InvalidInputException(String input) {  
        super("Invalid user input: " + input);  
        failingInput = input;  
    }  
  
    public InvalidInputException(String input, Throwable cause) {  
        super("Invalid user input: " + input, cause);  
        failingInput = input;  
    }  
  
    public String getFailingInput() {  
        return failingInput;  
    }  
}
```

Häufige Exceptions

IllegalArgumentException

Gegebenes (Methoden-)argument ungültiger Wert

NullPointerException

Gegebenes (Methoden-)argument ungültig, weil null

IllegalStateException

Methodenaufruf im aktuellen Programmzustand nicht gültig

IndexOutOfBoundsException

Ungültiger Index gegeben

UnsupportedOperationException

Objekt unterstützt Methode nicht

- ▶ Beim Werfen: Alle für Behandlung/Debugging notwendigen Informationen mitgeben
 - ▶ Sinnvolle Message (gedacht für Entwickler, **nicht** User)
 - ▶ Alle Variablenwerte, die Mitgrund für Ausnahme sind
 - ▶ Auf jeden Fall als Teil der Message
 - ▶ Im Idealfall als Objekt-Felder mit Getter-Methoden
 - ▶ Bsp.: Sinnvolle Infos für `IndexOutOfBoundsException`:
lower bound, upper bound, used index
 - ▶ Falls vorhanden: causing Exception

Exceptions dokumentieren

- ▶ In Methodensignatur: Nur checked Exceptions
- ▶ Im JavaDoc: **Alle** Exceptions, die Methode wirft
- ▶ Immer Umstände angeben, unter denen Exception geworfen wird

```
/**
 * ...
 *
 * @throws InvalidInputException if givenPointsArgument is not a positive integer
 * @throws NullPointerException if one of the arguments is null
 */
public void handleAdd(String user, String pointsArgument) throws InvalidInputException {
    if (user == null) {
        throw new NullPointerException("Given user is null");
    }
    if (pointsArgument == null) {
        throw new NullPointerException("Given points argument is null");
    }

    try {
        int points = Integer.parseInt(pointsArgument);
        addEntry(user, points);
    } catch (NumberFormatException e) {
        throw new InvalidInputException("Points param not a valid number: " + pointsArgument, e);
    }
}
```

- ▶ Java ist auch eine Insel: „Exceptions in try und catch“
- ▶ Java in a Nutshell: „Checked and Unchecked Exceptions“ und „Exceptions and Exception Handling“
- ▶ Effective Java, 3rd Ed.: „Exceptions“

Appendix

Exception catch

► try-catch-finally Blöcke

```
try {  
    int points = Integer.parseInt(pointsArgument);  
    doSomethingThatMayThrowIOException(user, points);  
  
} catch (NumberFormatException e) {  
    // some handling  
    handle(e);  
  
} catch (IOException e) {  
    // some other handling  
    handleIO(e);  
  
} finally {  
    // stuff that is always executed  
    cleanUpAndGo();  
}
```