Tagesprogramm

Instrumentierung für Debugging

Exceptions

Problembereiche

wissen nicht, wo Fehler passiert

Fehler meist nicht dort wo sich Auswirkungen zeigen

→ Fehlermeldung ist **keine Handlungsanweisung**

Instrumentierung für Debugging hilft bei Fehlersuche,
kann aber Programmverhalten ändern

Assert-Anweisung

```
assert Bedingung;
assert Bedingung : String_als_Fehlermeldung;
```

Überprüfungen normalerweise ausgeschaltet,
einschalten durch Interpreter-Flag `-ea`

Eingrenzen von Fehlern

strukturierte Vorgehensweise nötig:

- Orientierung
- Hypothese
- Planung
- Durchführung
- Auswertung

Aufgabe: Debugger oder Instrumentierung?

Such Sie in Gruppen zu 2 bis 3 Personen Antworten auf folgende Frage:

Was ist besser:
Debugger oder Instrumentierung mit Debug-Code?

Zeit: 2 Minuten

Laufzeitfehler

weitere Ausführung unmöglich → **Ausnahme geworfen**

Beispiele:
```
ArrayIndexOutOfBoundsException
NullPointerException
ArithmeticException
AssertionError
OutOfMemoryError
StackOverflowError
```

Ausnahmen sind Objekte von `Throwable`

eigene Ausnahme werfen: `throw new MyException();`

Ausnahme nicht abgefangen → Abbruch mit **Stack-Trace**

Arten von Ausnahmen

vordefinierte Ausnahmen:

Untertypen von `RuntimeException` abfangbar

Untertypen von `Error` sollen nicht abgefangen werden

selbstdefinierte Ausnahmen:

meist Untertypen von `Exception`

müssen abgefangen oder weitergeleitet werden

Weiterleitung nur wenn in Methodenkopf spezifiziert:

```
public void foo() throws ExcA, ExcB {...}
```

Aufgabe: Einsatz von Ausnahmen

Die Notwendigkeit von `throws`-Klauseln (bei selbstdefinierten Ausnahmen) ist umstritten, in anderen Programmiersprachen nicht vorhanden.

Finden Sie Gründe sowohl für als auch gegen die Verwendung selbstdefinierter Ausnahmen (mit `throws`-Klauseln) in Java.