Nachvollziehen des Programmablaufs

Probleme beim Debuggen

wissen nicht, worauf wir achten müssen

Änderungen von Programmen (Debug-Anweisungen) bzw. Variablenwerten wirken sich oft anders aus als erwartet

Verwendung des Debuggers ändert Programmverhalten

Fehler zeigen sich nicht dort, wo sie passieren
(„Nadel im Heuhaufen“)

betroffene Programmstellen nur statisch zu verstehen

Eingrenzen von Fehlern

Orientierung

Hypothese

Planung

Durchführung

Auswertung

Fehlermeldungen

Compiler erkennt Inkonsistenzen, aber keine Fehler,

daher können Fehlermeldungen irreführend sein

Fehlermeldung nicht als Handlungsanweisung verstehen, sondern
Fehlerursache finden!

Aufgabe: Generische Methoden

Such Sie in Gruppen zu zwei bis drei Personen Antworten auf folgende Frage:

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

Zeit: 2 Minuten

Ausnahmebehandlung

Laufzeitfehler

weitere Ausführung unmöglich → Ausnahme **geworfen**

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

Ausnahmen sind Instanzen 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:

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

Ausnahmen und Untertypen

Untertypen werfen nicht mehr Ausnahmen als Obertypen:

```
class A { void foo() throws ExcA, ExcB {...} }
class B extends A { void foo() throws ExcB {...} }
```

Programmierer muss darauf achten:

Methode in Untertyp darf nur in solchen Fällen Ausnahmen werfen wo man das auch für Methode in Obertyp erwartet

Falsch:

```
class A { int get(){ return 1; } }
class B extends A {
    int get(){ throw new RuntimeException("not OK"); }
}
```

Einsatz von Ausnahmen

Notwendigkeit von `throws`-Klauseln umstritten

oft unklar, wo Ausnahme geworfen wurde

Ausnahmen nicht zum vorzeitigen Ausstieg verwenden!

Ausnahmen nicht für alternative Ergebnistypen verwenden!

Ausnahmen für echte Ausnahmefälle gerechtfertigt, sonst nicht

Ausnahmebehandlung kann nicht alle Auswirkungen von Ausnahmen beseitigen

Aufräumen

`try-catch-finally` als syntaktische Einheit

`finally` immer ausgeführt (auch in Ausnahmefällen)

in `finally`-Blöcken wird aufgeräumt

Aufräumen schwierig, weil unklar wie weit `try` ausgeführt

nur Variablen aus äußeren Blöcken zugreifbar

falsche Programmaufrufe Normalfall, keine Ausnahme