Karlsruher Institut für Technologie

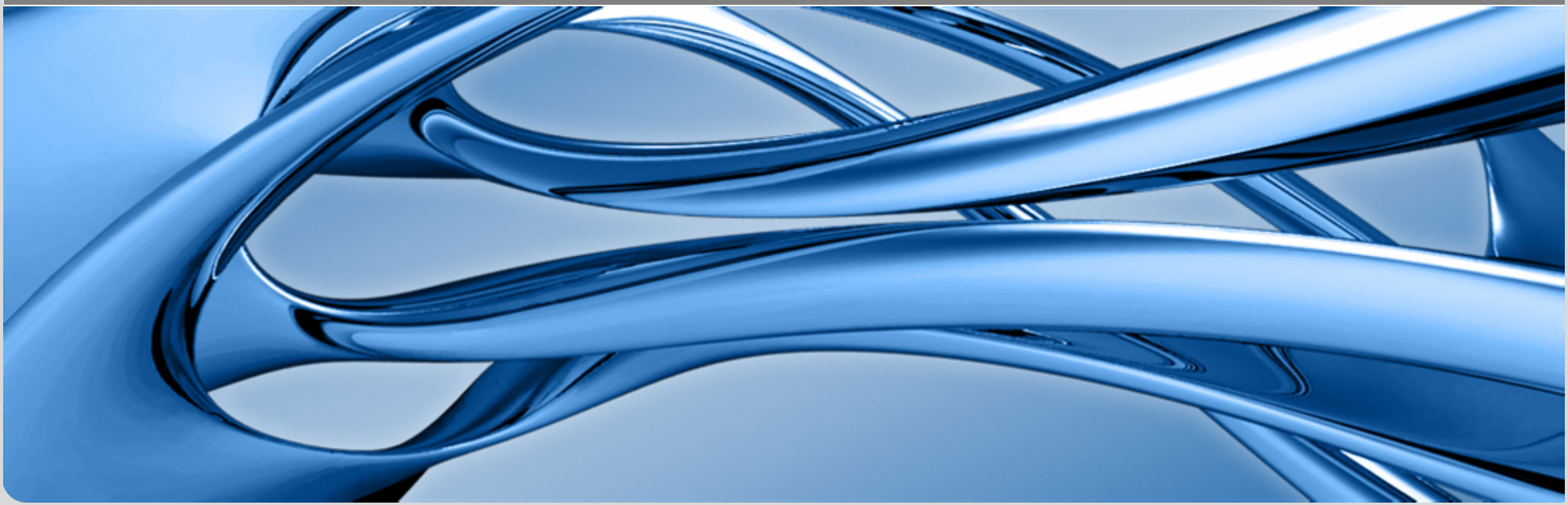
SDQ Software Design and Quality

Tutorium 10

Rekursion, Suchen und Beheben von Fehlern

Christian Zielke | 22. Januar 2018

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

1. Allgemeines
   - Präsenzübung
2. Rekursion
   - divide and conquer
   - Grundlegendes
   - Arten
   - Übungsaufgabe

Gliederung

Präsenzübung

- Ergebnisse werden im ILIAS veröffentlicht
- Einsicht nächste Woche im Tutorium

Wichtig

- Studentenausweis für die Einsicht mitbringen

divide and conquer

- Wichtiges Grundprinzip der Algorithmik
- Teile Problem in kleinere Teilprobleme auf
- Teilprobleme lösen -> zu Gesamtlösung zusammensetzen

Grundlegendes

- Methoden rufen sich selbst direkt oder indirekt auf.
- Bei jedem rekursiven Aufruf wird eine neue Instanz der Methode gestartet.
- Jede Instanz hat eigene lokale Variablen und Parameter.

Arten

rekursiv

Im Rumpf von f wird f mindestens einmal aufgerufen.

indirekt rekursiv

Im Rumpf von f wird eine Methode g aufgerufen, die f aufruft.

endständig rekursiv

Es kommt kein anderer Code im Rumpf von f, nachdem f aufgerufen wurde.

Übungsaufgabe

Palindrom

- Implementiere eine Klasse mit einer main Methode, die für einen gegebenen String s prüft, ob es sich um ein Palindrom handelt.
- Verwende dazu eine rekursive Methode
  boolean isPalindrom(String s)

Murphy's law

Everything that can go wrong, will go wrong!

Allgemein

- Perfekte Software entsteht selten im ersten Versuch
- Programm muss getestet werden, damit Fehler/Abweichungen entdeckt werden können
- Am besten jede Benutzereingabe und jeden möglichen Fehlerfall überprüfen
- Versuchen, das Programm abstürzen zu lassen

Wie sollte eine Methode getestet werden?

- Standardeingaben
- Grenzfälle
- auch möglich: Fehler produzierende/ungültige Fälle

Was ist ein guter Testfall?

- Prüft genau eine (spezifische) Eigenschaft
- Dokumentation
- Aussagekräftige Namen (zur Erkennung und Dokumentation)
- Am besten ein Testing-Framework (z.B. JUnit) benutzen
- kurz und prägnant
- unabhängig
- nach Spezifikation
- Schema:
  - Initialisieren
  - Durchführen
  - Prüfen
  - (bei Bedarf) Aufräumen

failure vs fault vs error

failure

- Fehlfunktion
- Programm verhält sich nicht nach Anforderung
- Quasi: Wirkung

fault

- Fehler, inkorrekter Schritt im Programm
- Ursache des Versagens
- Quasi Ursache

error

- Unterschied tatsächlicher und erwünschter Programmzustand
- kann, muss aber nicht zu Failure führen
- kann mit Assertions überprüft werden

Testen ist nicht

- Fehlerlokalisierung
- Fehler beheben
- Alternativen: statische Codeanalyse, Beweise, ...

Teststrategien

- Datenbasiert
  - Alle Gruppen (=gleiches Verhalten) von Daten testen
  - Grenzwerte
- Programmbasiert
  - jede Zeile
  - jede Bedingung zu wahr und falsch
  - jeder Pfad
  - etc.

Zusicherungen (assertions)

- Assertion erlauben die Überprüfung von Bedingungen zur Laufzeit
- Syntax:
  assert Bedingung;
  assert Bedingung: “Fehlermeldung”;
- Es gibt drei Arten von Zusicherungen
  - Vorbedingung
  - Nachbedingung
  - Schleifeninvariante

Java und assertions

Vorteile und Nachteile

- assert im Programm
  - Vorteil: Kann zur Laufzeit überprüft werden
  - Nachteil: Nur Java-Syntax möglich
- Kommentar in natürlicher Sprache
  - Vorteil: Freiheit im Ausdruck
  - Nachteil: Kann nicht (so einfach) automatisch geprüft werden

- Assertion müssen zum Programmstart erst aktiviert werden!
  java -ea MyClass
- In Eclipse unter Run Configurations → Arguments → VM arguments

assert vs if

assert

- Annahme des Programmierers zur Korrektheit
- Dokumentation
- Nicht zur Laufzeit behandelbare Fehler
- abschaltbar

if und Exception

- Prüfung von Eingabedaten des Benutzers
- Sonderfälle
- für von aufrufenden Funktionen behandelbare Fehler
- nicht abschaltbar

Beispiel

```
1   public static void setNumber(double d) {
2     assert d >= 60;
3     this.number = d;
4   }
```

```
1   public static void setNumber(double d) {
2     if (d < 60) {
3       throw new IllegalArgumentException();
4     }
5     this.number = d;
6   }
```

Debugging

Ablauf

- Man kann die Ausführung des Programms pausieren.
- An sogenannten Breakpoints wird das Programm angehalten.
- Aktueller Zustand kann analysiert werden.
- Anschließend Programm fortsetzen.
- Eclipse: Doppelklick neben Zeile.

Übungsaufgaben

Vorgehen

- Interface Binomial erstellen. (optional Javadocs)
- Interface in Klasse BinomialImplementation implementieren.
- Main Klasse, die n und k vom Benutzer einliest und falsche Eingaben behandelt. (Terminalklasse verwenden)
- Testklasse BinomialImplementationTest schreiben.

Übungsaufgaben

Binomialkoeffizient

Schreibe Methoden *int binRek(int n, int k)* und *int bin(int n, int k)*, welche den Binominialkoeffizienten einmal rekursiv und einmal durch eine geschlossene Formel berechnen. Es gilt:
$\binom{n}{k} = \frac{n!}{(n-k)!*k!}$ und $\binom{n}{0} = \binom{n}{n} = 1, \binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$

Erkenne ungültige Eingaben und werfe eine *IllegalArgumentException*!