Karlsruher Institut für Technologie

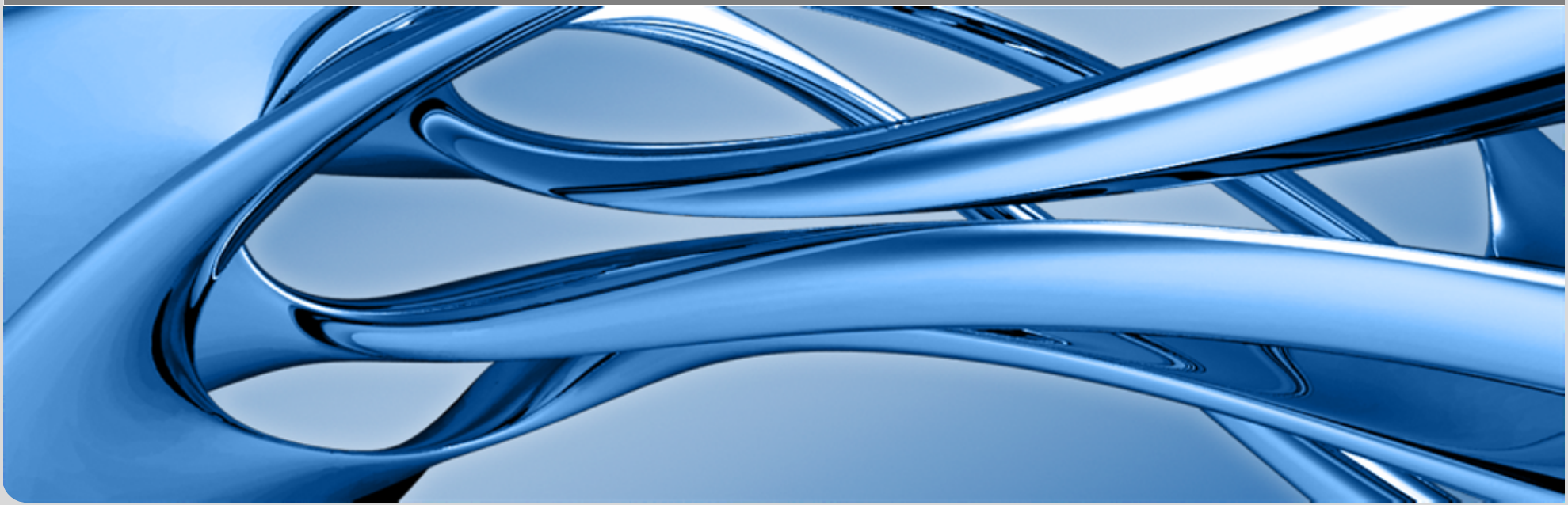
SDQ Software Design and Quality

Tutorium 11

JUnit, Suchen und Sortieren

Christian Zielke | 29. Januar 2019

CHAIR FOR SOFTWARE DESIGN AND QUALITY



Gliederung

Gliederung

Übungsblatt 4

Aufgabe A

- quit mit Argumenten
- häufig wurden nicht alle Eingabefälle beachtet
- ID static
- Stringformat mit führenden Nullen
- JavaDoc

Übungsblatt 4

Aufgabe B

- häufig wurden nicht alle Eingabefälle beachtet
- bei draw zu früh aufgehört
- Spiellogik fehlerhaft
- Array getter bzw. setter (Board Klasse)

Wichtig

Logik und UI trennen!!!

JUnit

- . . . ist ein Komponententest
- Ein Komponententest testet getrennt die einzelnen Bestandteile eines Programms.
- Der einfachste Weg dazu sind Unit-Testklassen (in Java meist JUnit).
- Das übliche Vorgehen dabei:
  - Lege für jede zu testende Klasse eine JUnit-Testklasse an
  - definiere eine Grundmenge von Objekten für die Tests
  - lege für jeden Test einer Methode eine neue Methode an (mehrere Test-Methoden pro “echter” Methode üblich)
- Vorteil: gut zum Testen von Grenz- und Fehlerfällen geeignet
- Nachteil: Schreiben der Unit-Tests relativ zeitaufwendig

JUnit

- Unit-Testklassen sind normale Java-Klassen mit folgenden Besonderheiten:
  - es ist kein Aufruf von einer main-Methode nötig, um Test-Methoden auszuführen
  - spezielle Methoden für Vergleich zwischen Soll- und Ist-Wert vorhanden
  - Methoden lassen sich als Aufbau/Abbau definieren
- Tags für die Ausführungsreihenfolge:
  - @Test : Methode wird als Test-Methode gekennzeichnet
  - @Before, @After :
    Ausführung vor/nach jeder einzelnen Test-Methode (mehrfach)
  - @BeforeClass, @AfterClass:
    Ausführung vor/nach irgendeiner Test-Methode (einmalig vor dem 1. Test)

JUnit

- Nach Import der ”Assert”-Klasse aus JUnit stehen einem einige Methoden zum Vergleich von Soll- und Ist-Wert zur Verfügung:
- import static org.junit.Assert.*;
- Vergleichsmethoden:
  - Am häufigsten verwendet werden diverse assertEquals()-Methoden:
  - assertEquals(Soll-Wert, Ist-Wert)
  - assertEquals(Soll-Double, Ist-Double, Diff)
  - assertTrue/False(boolean)
  - assertNotNull(object)

Suche

- Herausfinden, ob oder wo ein Element in einem Array vorkommt
- Verschiedene Ansätze

Lineare Suche

- Keine Voraussetzungen
- Array wird durchlaufen, bis das Element gefunden (oder Ende des Arrays erreicht) wurde

Suche

Binäre Suche

- Voraussetzung: Array muss sortiert sein
- Vorteil: idR schneller als lineare Suche
- Schaue das mittlere Element an im aktuellen Suchraum an (solange Suchraumgröße größer 1)
  - Gefunden: Suche beenden.
  - Kleiner als das Gesuchte: Suche oberhalb weiter.
  - Größer als das Gesuchte: Suche unterhalb weiter.

Bubblesort

- Recht einfaches Sortierverfahren
- in-place, stabil
- schlechte Laufzeit, nicht empfehlenswert
  - Laufzeiten: B: n, A: n^2, W: n^2
- Liste n Mal von links nach rechts durchgehen und Nachbarn vergleichen
- Wenn links $>$ rechts, dann tauschen

Selection Sort

- Array besteht aus zwei Bereichen: unsortiert und sortiert
- jeweils das kleinste Element des unsortierten Bereichs suchen
- Tauschen des kleinsten Elements an den Rand des sortierten Bereichs
- Laufzeiten: jeweils n^2
- in-place, stabil möglich (wenn statt tauschen eine insertion)

Insertion Sort

- Array besteht aus zwei Bereichen: unsortiert und sortiert
- Füge das erste Element des unsortierten Bereichs an die korrekte Stelle des sortierten Bereichs ein
- Laufzeiten: B: n, A: n^2, W: n^2
- stabil und in-place

Mergesort

- Teile-und-Herrsche Prinzip
- stabil
- Laufzeiten: B: $n \log n$, A: $n \log n$, W: $n \log n$

Mergesort - Pseudocode

```
1   funktion mergesort(liste);
2   falls (Groesse von liste <= 1) return liste
3   sonst
4   halbiere die liste in linkeListe, rechteListe
5   linkeListe = mergesort(linkeListe)
6   rechteListe = mergesort(rechteListe)
7   return merge(linkeListe, rechteListe)
```

merge

- neue Liste erstellen
- solange beide Listen nicht leer:
  - erstes links $<=$ erstes rechts: erstes Links in neue Liste, sonst erstes rechts
- dann die übrig gebliebene Liste noch hinzufügen

Einsicht

- max. 5 Minuten pro Student
- Einzeln vorkommen
- Studentenausweis und Personalausweis
- Keine Änderungen, Fotos, usw.
- finale Notenänderung wird von Prof. Reussner vorgenommen

Übungsaufgabe

- Implementiere eine Binäre Suche auf einem int array.

Übungsaufgabe

- Implementiert Selection und Insertion Sort auf einem int array.

Übungsaufgabe

- Schreibe jeweils eine JUnit Testklasse für die vorherigen Klassen.