

PK

Antwortzeiten

Gründe für lange Antwortzeiten

Wartezeiten auf Daten

versteckte Aktivitäten (können Schaden anrichten)

aufwendige Präsentation

Schwerpunkt auf Entwicklungszeiten und Wartung

Schwerpunkt auf Speicherverbrauch

übermäßiger Technologieeinsatz

falsche Optimierung (z.B. Busy Waiting)

Grundsätze der Optimierung

kein Experte → keine Optimierungen

Experte → noch keine Optimierungen

Vermeidung versteckter Aktivitäten

Virens Scanner und andere Schutzsoftware

zertifizierte Software

Open-Source-Software

PK

Grenzen

Ganze Zahlen

primitiv	Referenz	Bits	größte darstellbare Zahl
byte	Byte	8	127
short	Short	16	32.767
int	Integer	32	2.147.483.647
long	Long	64	9.223.372.036.854.775.807
—	BigInteger	—	unbeschränkt

Überlauf = Wertebereich überschritten

Unterlauf = Wertebereich unterschritten

Rechnen mit ganzen Zahlen

Wertebereich abschätzen, nötigenfalls BigInteger

Wertebereich z.B. auch für Präsentation wichtig

nie Fließkommazahlen statt ganzer Zahlen verwenden (Rundungsfehler)

Programmänderungen ändern Wertebereiche → neuerlich abschätzen

Division durch 0 vermeiden

Modulo = Divisionsrest (%) nur für positive Operanden

asymmetrischer Wertebereich (z.B. -2.147.483.648)

equals (Referenz) versus == (primitiv)

Aufgabe: Division durch 0

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

Wie kann man eine Division durch 0 vermeiden?

Zeit: 2 Minuten

Arten reeller Zahlen (am Computer)

Fließkommazahlen: Komma variabel

großer Wertebereich, aber beschränkte Genauigkeit
(z.B. 1.23456, 0.000123456, 123456000.0, ...)

je näher an 0.0 desto genauer

Festkommazahlen: fixe Zahl an Nachkommastellen

Wertebereich und Genauigkeit beschränkt

manche Arten für Geldbeträge geeignet

in Java nur `BigDecimal`, kein primitiver Typ

Spezielle Fließkommazahlen

`-0.0` verschieden von `0.0` (aber `-0.0 == 0.0` ergibt `true`)

`x / 0.0` ergibt `POSITIVE_INFINITY` oder `NEGATIVE_INFINITY`

sinnlose Berechnungen ergeben `NaN` (Not a Number)

keine Ausnahmen sondern kontrollierte Über- bzw. Unterläufe

Probleme mit Fließkommazahlen

Auslöschung: $1.2345678 - 1.2345677 = 0.0000001$

Absorption: $1e10 + 1e-10$ ergibt $1e10$

gut oder schlecht **konditionierte** Algorithmen und Programme

`Math.abs(x - y) < eps` statt `x == y`

viele Spezialfälle im Umgang mit NaN, 0, etc.

Verwendung von null

```
public interface Motor { double gCO2proKm(); }

public class Benzinmotor implements Motor {
    private double liter;
    public double gCO2proKm() { return liter * 23.7; }
    ...
}

public class Fahrzeug1 {
    private Motor motor;           // null für Fahrrad, etc.
    public double gCO2proKm() {
        if (motor != null) return motor.gCO2proKm();
        else return 0.0;
    }
    ...
}
```

Vermeidung von null

```
public class KeinMotor implements Motor {  
    public double gCO2proKm() { return 0.0; }  
    ...  
}
```

```
public class Fahrzeug2 {  
    private Motor motor; // darf nicht null sein  
    public double gCO2proKm() {  
        return motor.gCO2proKm();  
    }  
    ...  
}
```

Anfang, Ende und Sonderfälle

Fehler eher an Randzonen bzw. Grenzen als im Kern
(Initialisierung, Abbruchbedingung, Ausnahmebehandlung, etc.)

häufig als **Off-by-one-Fehler**

zwischen Kern und Randzone umdenken notwendig
= abstraktes versus konkretes Denken
analog zu Induktionsschritt und Induktionsanfang

Fehlervermeidung in Randzonen

Zusicherungen beschreiben vor allem Grenzfälle

Testfälle konzentrieren sich auf Grenzfälle

Code-Review (anstrengend)

Fehler vor Korrektur genau analysieren

statisches Verstehen, Bruchlinien vermeiden

Termination sicherstellen (erfordert statisches Verstehen)

Aufräumen nach Ausnahmen

Plausibilitätsprüfungen

Grenzen als Gefahrenquellen

nicht jeder Fehler an Grenzen fällt auf

Angreifer nutzen Fehler gezielt aus

Beispiel: Arrayzugriff außerhalb des Indexbereichs

auch Java-Interpreter können falsche Zugriffe nicht zu 100% ausschließen
(z.B. auf Smartphones)

Angreifer können dadurch eigenen Code einschleusen

Angriffe über **Pufferüberläufe** am weitesten verbreitet

Sensibilität beim Programmieren notwendig