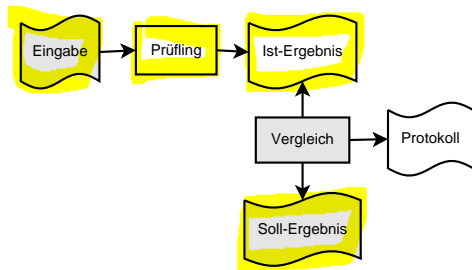


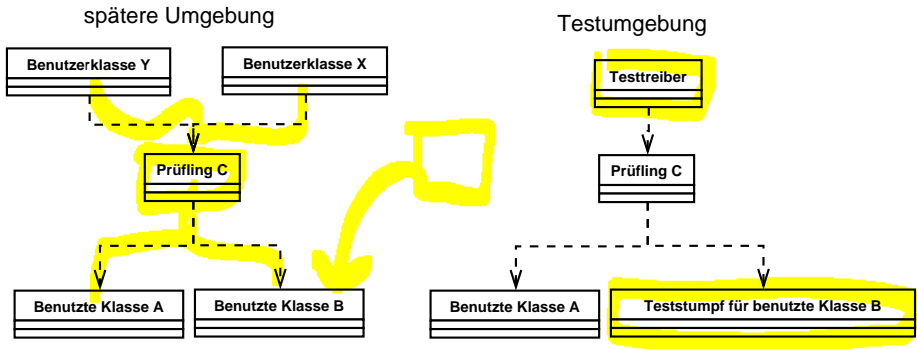
Testfall

Ein Testfall besteht aus:

- Name
- Ort: vollständiger Pfadname des lauffähigen Testprogramms
- Eingabe: Eingabedaten oder Befehle
- Orakel: erwartete Testergebnisse, die mit den aktuellen Testergebnissen des Testlaufs verglichen werden
- Protokoll: gesamte Ausgabe, die durch Test erzeugt wird



Komponententest: Teststümpfe und -treiber



Typen von Komponententests

Definition

Black-Box-Tests: Komponente wird nur mit Kenntnis der Schnittstellenbeschreibung entworfen (Implementierung unbekannt).

Techniken:

- Äquivalenztest
- Grenztest
- (zustandsbasierter Test)
- ...

Typen von Komponententests

Definition

White- / Glass-Box-Tests: Testfall wird mit Kenntnis der Implementierung entworfen.

Techniken:

- Pfadtest
- (zustandsbasierter Test)
- ...

Äquivalenztest I

Ziel: Testfälle minimieren.

Idee:

- Äquivalente Testfälle werden zusammengefasst.
- Ein Testfall wird als Repräsentant der Äquivalenzklasse aufgestellt.

Kriterien:

- Abdeckung: Jede mögliche Eingabe gehört zu einer der Äquivalenzklassen
- Disjunktion: Keine Eingabe gehört zu mehr als einer einzigen Äquivalenzklasse
- Repräsentation (die Hoffnung): Falls Ausführung einen fehlerhaften Zustand anzeigt, sobald ein bestimmtes Mitglied einer Äquivalenzklasse als Eingabe benutzt wird, dann kann derselbe fehlerhafte Zustand entdeckt werden, wenn irgendein anderes Mitglied dieser Äquivalenzklasse als Eingabe verwendet wird

Ein korrektes Programm?

```
class Kalender {
    public static class MonatUngueltig extends Exception {};
    public static class JahrUngueltig extends Exception {};
    public static boolean istSchaltJahr(int jahr)
        {return (jahr % 4) == 0;}
    public static int TageProMonat (int monat, int jahr)
        throws MonatUngueltig, JahrUngueltig {
        int anzTage;
        if (jahr < 1) { throw new JahrUngueltig(); }
        if (monat in {1, 3, 5, 7, 10, 12}) { anzTage = 32;}
        else if (monat in {4, 6, 9, 11}) { anzTage = 30;}
        else if (monat == 2) {
            if (istSchaltJahr (jahr)) anzTage = 29;
            else anzTage = 28;
        } else throw new MonatUngueltig();
        return anzTage;
    }
}
```

`Kalender.TageProMonat(monat, jahr)` liefere die Tage eines Monats wie folgt:

- $\text{monat} \in \{1, 3, 5, 7, 8, 10, 12\} \Rightarrow 31$
- $\text{monat} \in \{4, 6, 9, 11\} \Rightarrow 30$
- jahr ist ein Schaltjahr $\wedge \text{monat} = 2 \Rightarrow 29$
- jahr ist kein Schaltjahr $\wedge \text{monat} = 2 \Rightarrow 28$

`Kalender.TageProMonat(monat, jahr)` liefere die Tage eines Monats wie folgt:

- $\text{monat} \in \{1, 3, 5, 7, 8, 10, 12\} \Rightarrow 31$
- $\text{monat} \in \{4, 6, 9, 11\} \Rightarrow 30$
- $\text{jahr ist ein Schaltjahr} \wedge \text{monat} = 2 \Rightarrow 29$
- $\text{jahr ist kein Schaltjahr} \wedge \text{monat} = 2 \Rightarrow 28$

Fehlerfall

- $\text{monat} < 1 \vee \text{monat} > 12 \Rightarrow \text{throw MonatUnguelting}$
- $\text{jahr} < 0 \Rightarrow \text{throw JahrUnguelting}$

Äquivalenzklassen:

- für Monate:
 - Monate mit 31 Tagen
 - Monate mit 30 Tagen
 - Monate mit 28 oder 29 Tagen
 - `monat` außerhalb gültigen Bereichs
- für Jahre:
 - Schaltjahre
 - Jahre, die keine Schaltjahre sind
 - `jahr` außerhalb gültigen Bereichs

→ Kombination der Äquivalenzklassen für alle Eingabeparameter ergibt Äquivalenzklassen für `TageProMonat`

Beispieltestfälle

Test der Interaktion von `monat` und `jahr` durch Kombination:

Äquivalenzklasse	monat	jahr	Soll
31-Tage-Monat, Nicht-Schaltjahre	7	1901	31
31-Tage-Monat, Schaltjahre	7	1904	31
30 Tage-Monat, Nicht-Schaltjahre	6	1901	30
30 Tage-Monat, Schaltjahre	6	1904	30
Februar, Nicht-Schaltjahre	2	1901	28
Februar, Schaltjahre	2	1904	29
monat inkorrekt, jahr korrekt	17	1901	MonatUnguelutig
monat korrekt, jahr inkorrekt	7	-1901	JahrUnguelutig
monat inkorrekt, jahr inkorrekt	17	-1901	MonatUnguelutig VJahrUnguelutig

Komponententest mit JUnit

```
import junit.framework.*;

public class KalenderTest extends TestCase {

    public KalenderTest(String name) {
        super(name);
    }
    ...
    // Test-Methoden
    ...
    public static void main(String[] args) {
        junit.swingui.TestRunner.run(KalenderTest.class);
    }
}
```

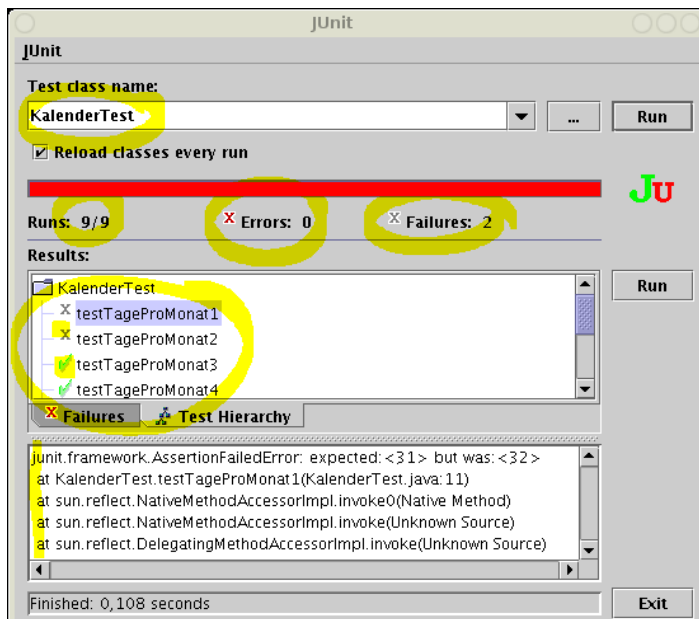
Komponententest mit JUnit

```
public void testTageProMonat1() {  
    // 31-Tage-Monat, Nicht-Schaltjahre  
    assertEquals(31, Kalender.TageProMonat (7, 1901));  
}
```

Komponententest mit JUnit

```
public void testTageProMonat1() {  
    // 31-Tage-Monat, Nicht-Schaltjahre  
    assertEquals(31, Kalender.TageProMonat (7, 1901));  
}  
  
public void testTageProMonat9() {  
    // monat inkorrekt, jahr inkorrekt  
    try {int t = Kalender.TageProMonat (13, -1901);  
        assertTrue (false);}  
    catch (Kalender.JahrUngueltig je) {assertTrue (true);}  
    catch (Kalender.MonatUngueltig me) {assertTrue (true);}  
    catch (Exception e) { assertTrue (false);}  
}
```

Komponententest mit JUnit



Beobachtung: “Off-by-one”-Fehler kommen häufig vor.

Idee: Grenzen (Randbereiche) von Äquivalenzklassen testen.

Beobachtung: “Off-by-one”-Fehler kommen häufig vor.

Idee: Grenzen (Randbereiche) von Äquivalenzklassen testen.

Schaltjahrregel: Ein Jahr ist ein Schaltjahr, wenn

- es durch 4 teilbar ist,
- es sei denn, es ist durch 100 teilbar,
- es sei denn, es ist durch 400 teilbar

2000 ist ein Schaltjahr, 1900 ist kein Schaltjahr.

Grenzttest

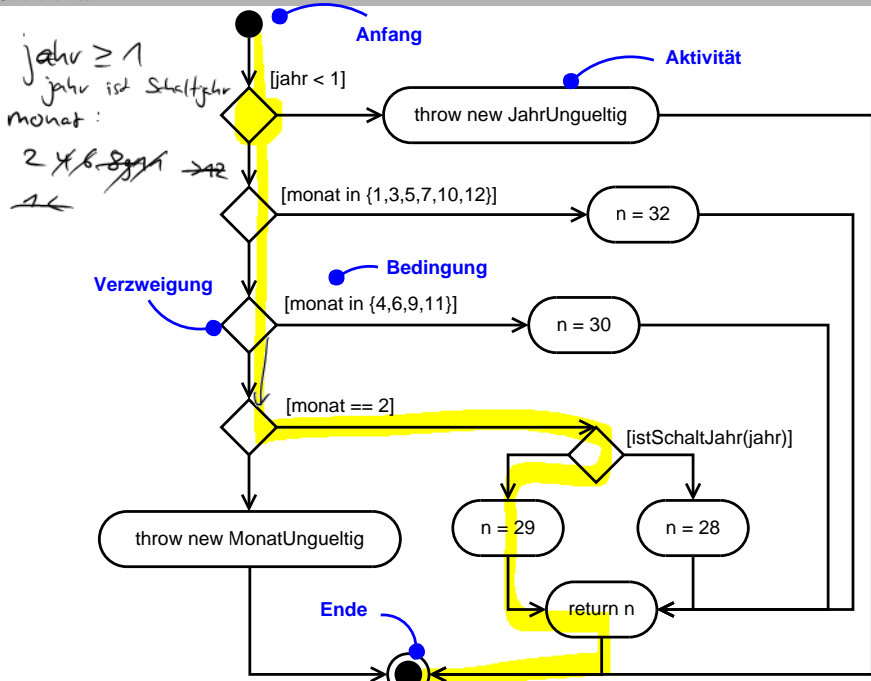
Äquivalenzklasse	monat	jahr	Soll
Schaltjahre teilbar durch 400	2	2000	29
Nicht-Schaltjahre teilbar durch 100	2	1900	28
gültige Monate	1	2000	31
gültige Monate	12	2000	31
Nichtpositive ungültige Monate	0	1234	MonatUnguechtig
Positive ungültige Monate	13	1234	MonatUnguechtig
gültiges Jahr	2	0	29
gültiges Jahr	3	Max-Int	30
ungültiges Jahr	2	-1	JahrUnguechtig

Ziel: Testfälle sollten alle Code-Teile testen.

Idee: Konstruiere Testfälle, die jeden möglichen Pfad mindestens einmal ausführen.

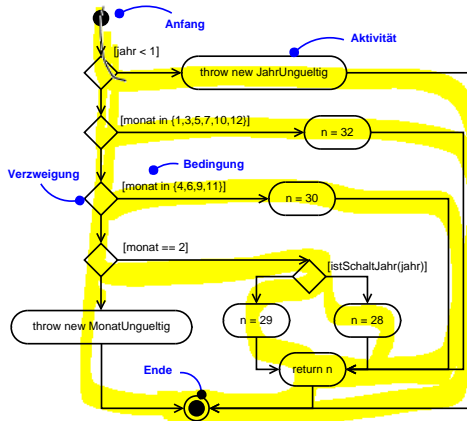
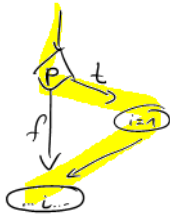
Ausgangspunkt: Kontrollflussgraph (KFG) pro Methode.

- Knoten: Anweisungen und Prädikate
- zwei weitere Knoten: eindeutiger Anfang und Ende einer Methode
- Kanten: Kontrollfluss zwischen Anweisungen (unbedingt) und Prädikaten (bedingt)
- Pfad: ~~Menge~~ Folge von Kanten, die vom Anfang zum Ende des KFGs führen



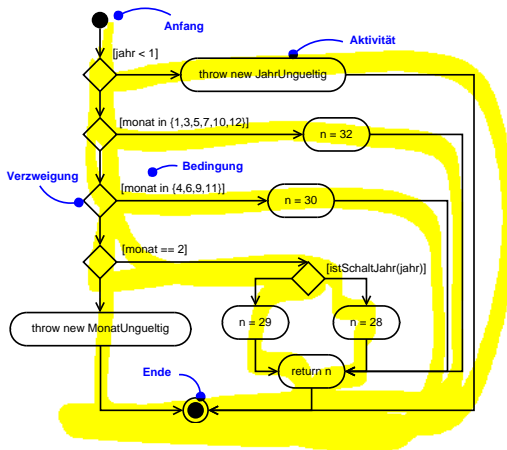
Maße der Testabdeckung

C0, **Anweisungsüberdeckung** (Befehlsabdeckung/Statement-Coverage):
Verhältnis von Anzahl der mit Testdaten durchlaufenen
Anweisungen zur Gesamtanzahl der Anweisungen.



Maße der Testabdeckung

C1, **Zweig-/Entscheidungsüberdeckung** Verhältnis von Anzahl der mit Testdaten durchlaufenen Zweige zur Gesamtanzahl der Zweige.



C2, Bedingungsabdeckung Verhältnis von Anzahl der mit Testdaten durchlaufenen Terme innerhalb von Entscheidungen zur Gesamtanzahl der Terme.

```
i := 1;  
loop  
    found := a(i) = key;  
    exit when found or i = Max_Entries;  
    i := i + 1;  
end loop;
```