AARHUS UNIVERSITET

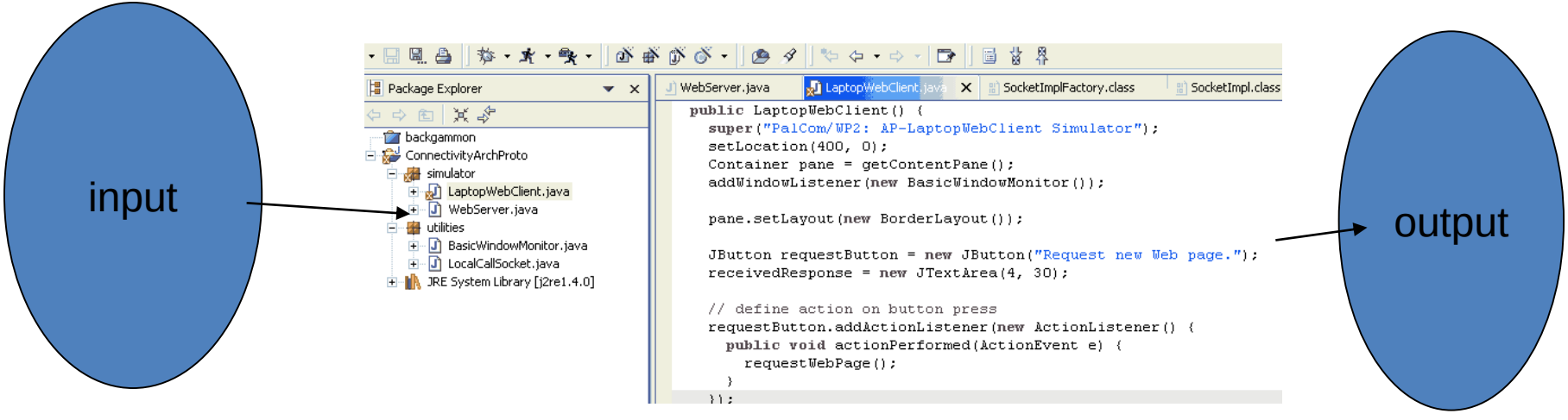
Software Engineering and Architecture

Code Coverage

Quality of your Test Cases (?)

BlackBox and WhiteBox

- Whitebox testing (glassbox / structural)



  - *If we know the structure of code, we may construct testcases that ensures we run through all parts of it – ensure all 'structural' elements are "working".*

AARHUS UNIVERSITET

Program structures

- Basically any program is a combination of only three structures, or *basic primes*
  - sequential: a block of code {...}
  - decision: a switch if, switch, ...
  - iteration: a loop while, repeat, do,

- WB focus on these basic primes and allow us to
  - evaluate test sets with respect to their ability to exercise these structures
  - thus – evaluate quality of test sets (Hm….)
  - and thus judge if a test set should be improved

AARHUS UNIVERSITET

Adequacy

- A necessary result of this focus is on *adequacy* (Da: Tilstrækkelighed(?), dækning)

- Example:
  - A test set T ensures that 100% of all statements in the production code P are executed.
  - T is **statement adequate** for P.
    - More often used term:
      "T ensures **statement coverage** for P"
  - If less than 100% are executed then T is not statement adequate for P.

Criteria

- There are numerous adequacy criteria. WB focus on *program-based criteria* but others are useful also in BB and other types of testing.
- WB criteria
  - statement coverage
  - decision coverage
  - path coverage, and many more
- Other types of criteria
  - use case coverage (system level)
  - interface coverage (integration level)

AARHUS UNIVERSITET

Aspects of WB

- WB look at code
  - Corollary:
    - WB does not start until late in the development process
    - BB can be started earlier than WB
  - Corollary:
    - WB is only feasible for smaller UUTs
      - because the flow graphs explode in large units
    - BB can be used all the way to system level
  - Corollary:
    - WB is expensive for unstable UUTs
      - because implementation changes invalidate the analysis!
    - BB survives if the behavior + interface are stable

AARHUS UNIVERSITET

WB Coverage types

- Overall there are a number of metrics for coverage:
  - statement coverage
  - decision coverage
  - condition coverage
  - decision/condition coverage
  - multiple-condition coverage
  - path coverage

- The all relate to the *flow graph* of code.

AARHUS UNIVERSITET

Flow graph

- The flow graph is simply the route diagrams that has gone out of fashion.

- It defines a graph where nodes are *basic primes* (block, decision, iteration) and edges are control flow between them (the ProgramCounter ☺).

AARHUS UNIVERSITET

Example code

- Danish **mellemskat** (some similarity to current topskat!)
- *Mellemskat is a 6 % tax in 2003*

- *Mellemskatten is calculated based on **personal income** plus **positive net capital income**.*

- *If a **married person** has negative net capital income, this amount is deducted in the spouse's positive net capital income before the mellemskat is calculated. Still, the net capital income cannot be less than 0.*

AARHUS UNIVERSITET

Initial design

- `public static int calculateMellemskat(Taxpayer t)`

- Let us define a **taxation basis** which is the amount that the tax is calulated on.
- Calculations involve:
- `t.netCapitalIncome()` `nci`
- `t.getSpouse()` `s`
- `t.getSpouse().netCapitalIncome()` `s.nci`
- `posNetCapitalIncome` `pos`

AARHUS UNIVERSITET

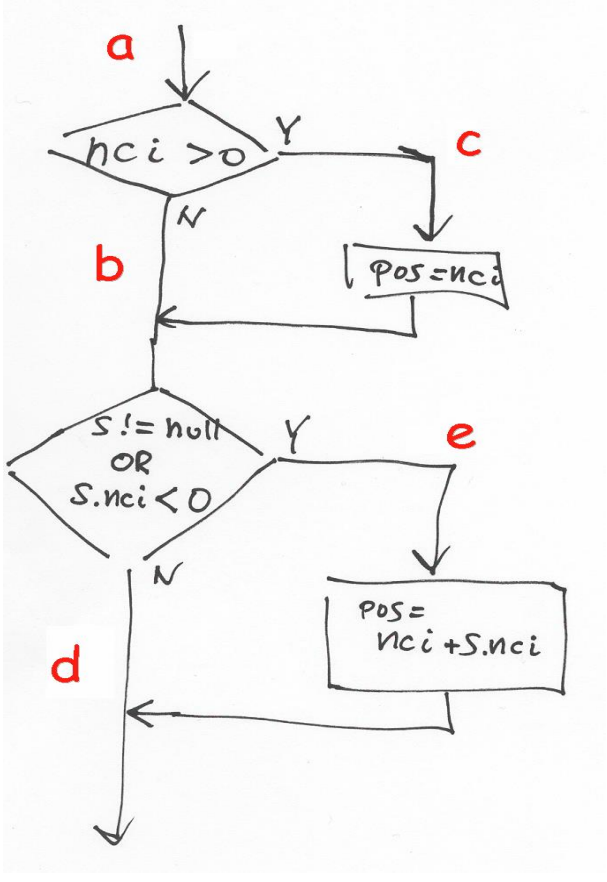
Mr. BrightGuy's first shot

```
public static int calculateMellemskat(Taxpayer t) {
   int posNetCapitalIncome = 0;
   if ( t.netCapitalIncome() > 0 ) {
     posNetCapitalIncome = t.netCapitalIncome();
   }
   if ( t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0 ) {
     posNetCapitalIncome =
       t.netCapitalIncome() +
       t.getSpouse().netCapitalIncome();
   }
   int taxationbasis = t.personalIncome() + posNetCapitalIncome;
```

Note: impressive amount of defects!

AARHUS UNIVERSITET

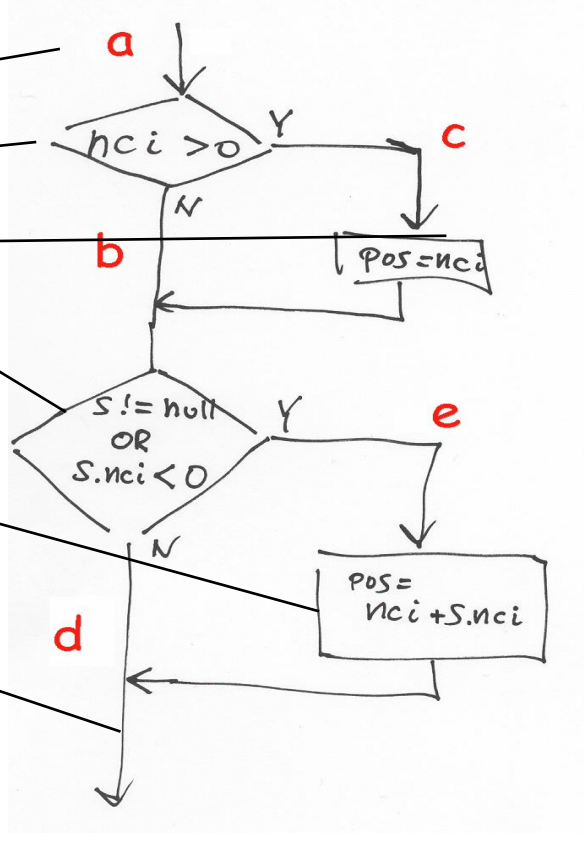
Flow Graph



AARHUS UNIVERSITET

Flow Graph

```
public static int calculateMellemskat(Taxpayer t) {
   int posNetCapitalIncome = 0;
   if ( t.netCapitalIncome() > 0 ) {
     posNetCapitalIncome = t.netCapitalIncome();
   }
   if ( t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0 ) {
     posNetCapitalIncome =
       t.netCapitalIncome() +
       t.getSpouse().netCapitalIncome();
   }
   int taxationbasis = t.personalIncome() + posNetCapitalIncome;
```

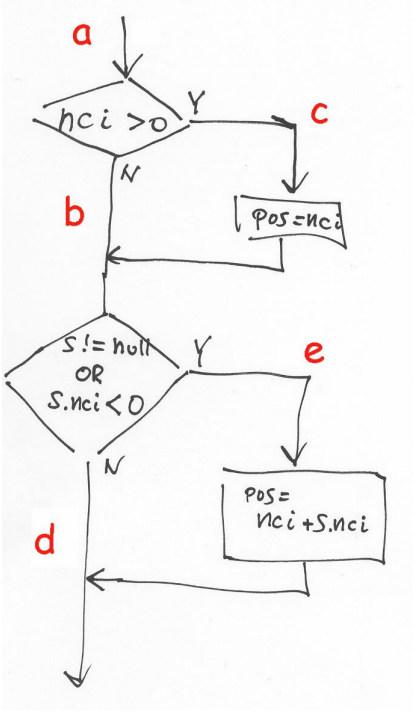


AARHUS UNIVERSITET

Statement coverage

- Statement coverage:
  - ***Require every statement in the program to be executed at least once***

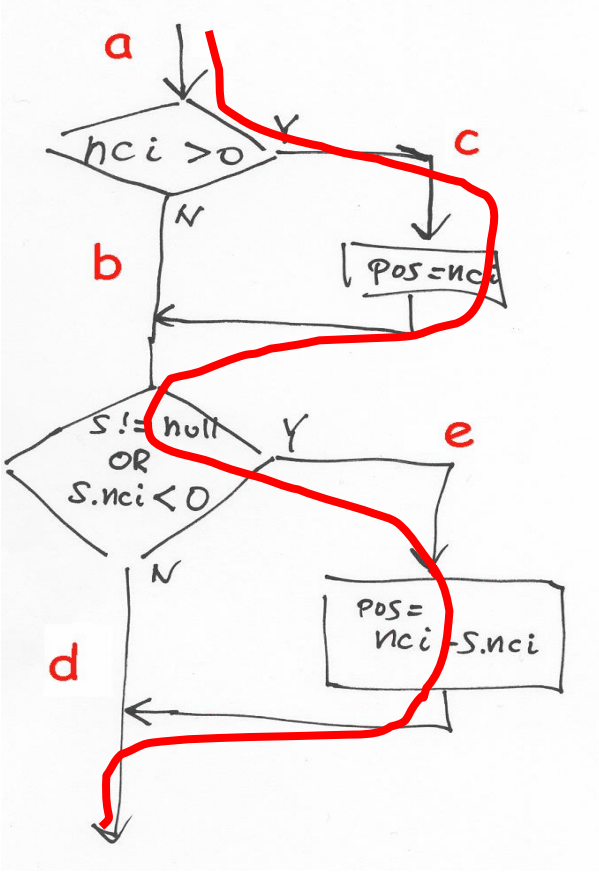
- Exercise:
  - which path satisfy SC criterion?



AARHUS UNIVERSITET

Statement coverage

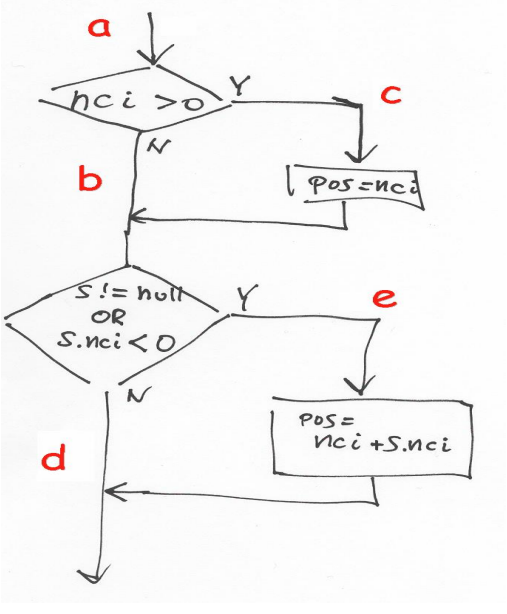
- SC criterion is pretty weak.
- Path **ace** is enough.
- TC: nci = +1000; s.nci = -2000
  - expected tb: 0
  - But tb = -1000 i.e. **defect detected**
  - however, not all defects uncovered !
    - s = null will result in error, as second condition in second decision will throw an exception.



AARHUS UNIVERSITET

Decision coverage

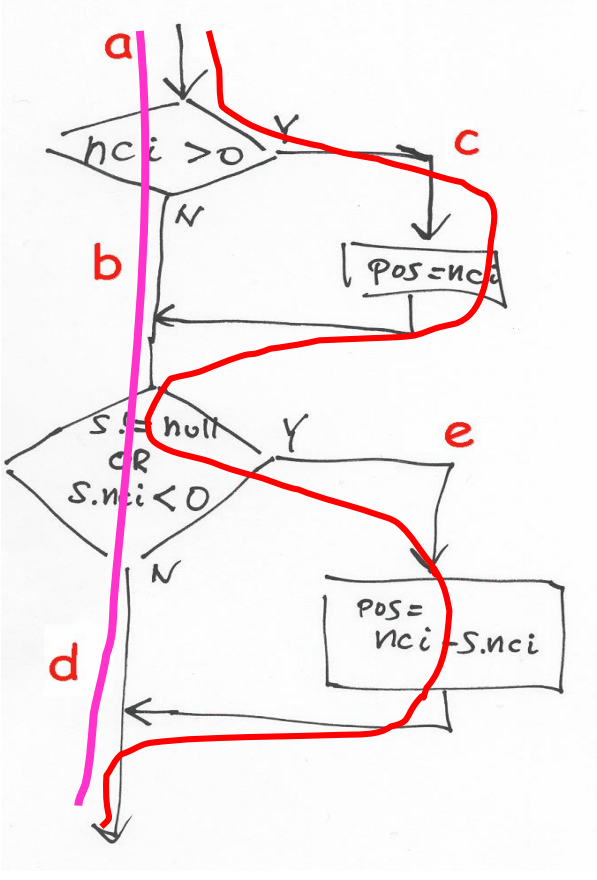
- Decision coverage (branch coverage):
  - ***Require each decision has a true and false outcome at least once***
- Decision 1:
  - nci > 0
- Decision 2
  - s!=null OR s.nci < 0
- Exercise:
  - which paths satisfy DC criterion?



AARHUS UNIVERSITET

Decision coverage

- DC criterion is better
- TC1: D1 true D2 true
- TC2: D1 false D2 false
- Which relates to the paths
  - ace abd
- TC1:
  - nci = +1000; s.nci = -2000 (finds bug!)
- TC2:
  - nci = -2000; s == null
  - expected tb = 0; but result is
- Null exception, ie. defect discovered
  - however, a defect still uncovered:
    - nci < 0 and s.nci < 0; expect tb = 0 is not tested.



AARHUS UNIVERSITET

Decision coverage

- *Usually* decision coverage satisfy statement coverage.

- However, beware of

  - exception handling code
    - because the switch is not apparent in the code!
      - thus exception handling statements are not covered...
  - and some exotic cases from forgotten languages
    - assembler multi entry procedures !
    - Any Cobol sharks around? What about Cobol?

And…

- The other coverage metrics are not considered in SWEA.
- And most CodeCoverage tools out there cannot compute them…
- From a practical point of view
  - Complex to compute and make test cases for
  - I doubt the *return on investment*, but – a personal gut feeling...

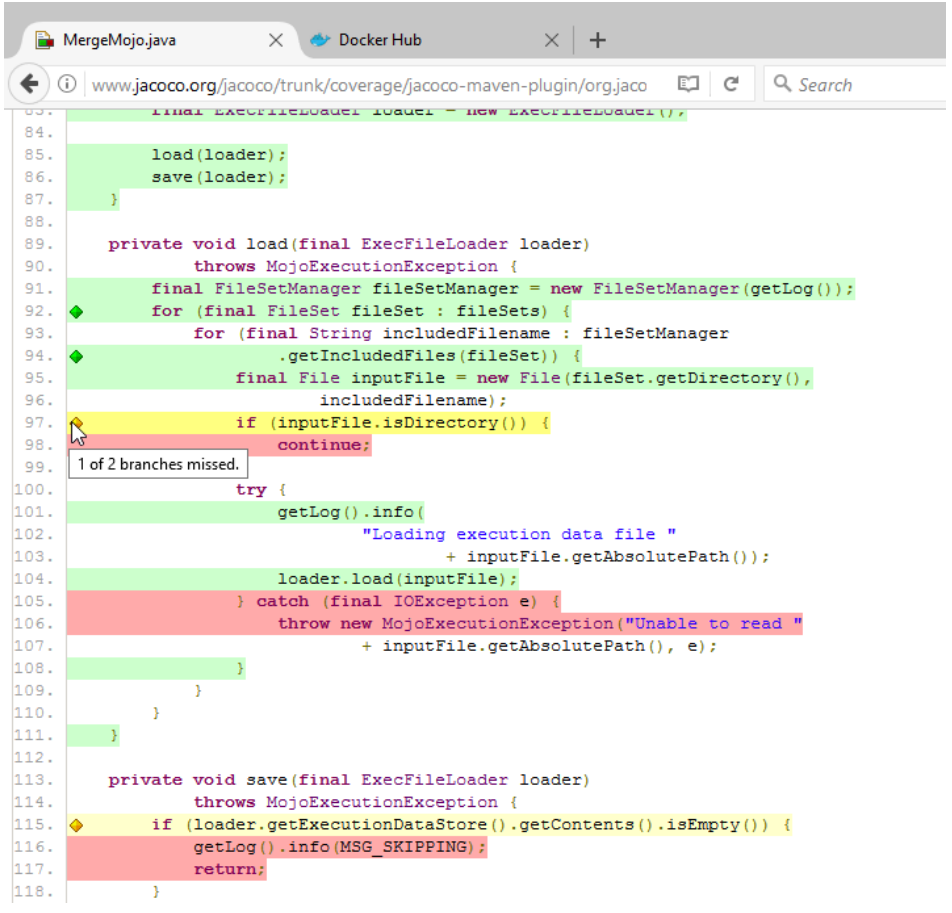
JaCoCo

Java Coverage Tool

AARHUS UNIVERSITET

JaCoCo

- Measures
  - Statement coverage (*)
    - (*) Actually bytecode!
  - Decision coverage
    - Call it ‘branch coverage’
    - Exceptions *not* counted
- Green: Covered
- Yellow: Partially Cov.
  - ‘some branches’ covered
- Red: Not Covered



JaCoCo

Note: Switches are weird!

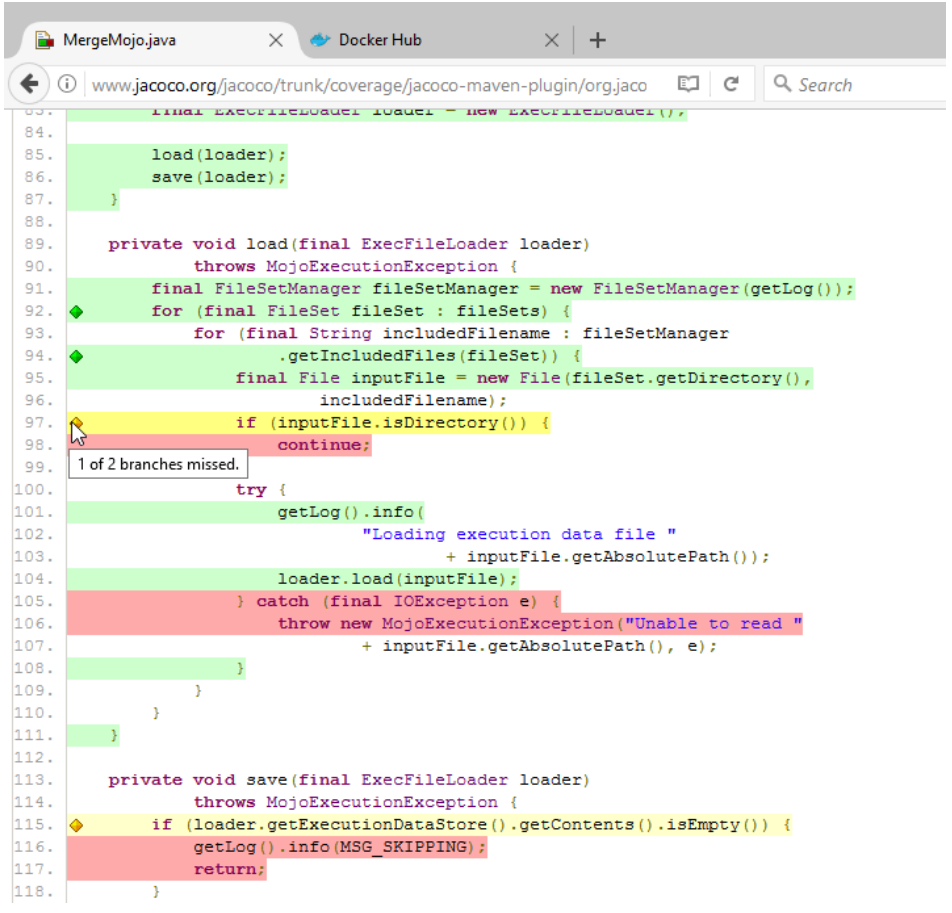
- JaCoCo measures bytecode coverage!
- Switches on strings are *rewritten* to hashcodes
- Thus, often *yelllow* even though the test is actually adequate ☹

```
class Fun  {
  static int fun(String s) {
    switch (s) {
      case "I":
        return 1;
      case "A":
        return 2;
      case "Z":
        return 3;
      case "ABS":
        return 4;
      case "IND":
        return 5;
      default:
        return 6;
    }
  }
}
```

```
int c = -1;
switch (s.hashCode()) {
  case 65: // +1 branch
    if (s.equals("I")) // +2 branches
      c = 0;
    break;
  case 73: // +1 branch
    if (s.equals("A")) // +2 branches
      c = 1;
    break;
  case 90: // +1 branch
    if (s.equals("Z")) // +2 branches
      c = 2;
    break;
  case 64594: // +1 branch
    if (s.equals("ABS")) // +2 branches
      c = 3;
    break;
  case 72639: // +1 branch
    if (s.equals("IND")) // +2 branches
      c = 4;
    break;
  default: // +1 branch
}
switch (c) {
  case 0: // +1 branch
    return 1;
  case 1: // +1 branch
    return 2;
  case 2: // +1 branch
    return 3;
  case 3: // +1 branch
    return 4;
  case 4: // +1 branch
    return 5;
  default: // +1 branch
```

How to Use

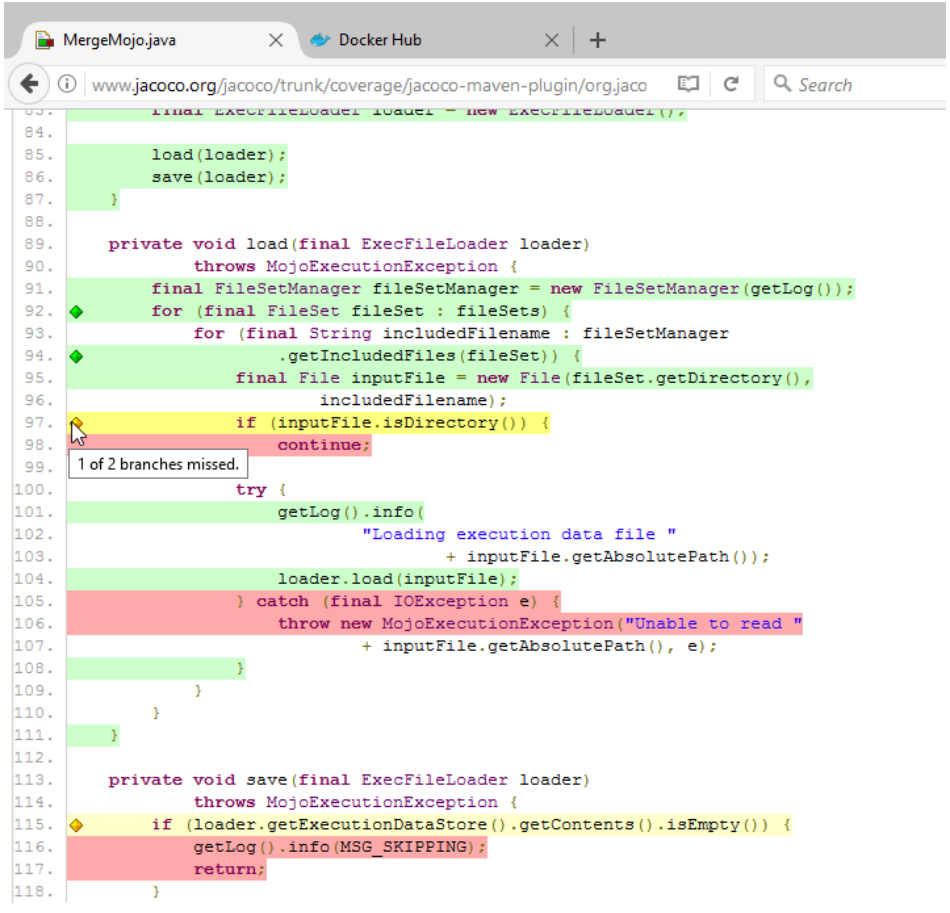
- Again
  - *100% statement coverage tells very little about the functional quality of the code!!!*
- But
  - **15% coverage tells us that we need a lot more testing !!!**
- It is a technique to help us find more/better test cases…



AARHUS UNIVERSITET

How to Use

- And you also get a nice browsable version of your code…



AARHUS UNIVERSITET

Beware of…

- IntelliJ and Eclipse can generate code for ‘equals’ and ‘hashCode’

```
@Override
public int hashCode() {
   final int prime = 31;
   int result = 1;
   result = prime * result + ((playerID == null) ? 0 : playerID.hashCode());
   return result;
}

@Override
public boolean equals(Object obj) {
   if (this == obj)
      return true;
   if (obj == null)
      return false;
   if (getClass() != obj.getClass())
      return false;
   PlayerRecord other = (PlayerRecord) obj;
   if (playerID == null) {
      if (other.playerID != null)
         return false;
   } else if (!playerID.equals(other.playerID))
      return false;
   return true;
}
```

- I assume it is correct code!
  - Thus I do not test
- Annoying low coverage

PlayerRecord

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
equals(Object)		19%		17%	6	7	10	13	0	1
hashCode()		89%		50%	1	2	0	4	0	1
toString()		100%		n/a	0	1	0	1	0	1
PlayerRecord(SubscriptionRecord, String, String)		100%		n/a	0	1	0	8	0	1
isInCave()		100%		100%	0	2	0	1	0	1
setPositionAsString(String)		100%		n/a	0	1	0	2	0	1
setSessionId(String)		100%		n/a	0	1	0	2	0	1
getPlayerID()		100%		n/a	0	1	0	1	0	1
getPlayerName()		100%		n/a	0	1	0	1	0	1
getGroupName()		100%		n/a	0	1	0	1	0	1
getPositionAsString()		100%		n/a	0	1	0	1	0	1
getRegion()		100%		n/a	0	1	0	1	0	1
getSessionId()		100%		n/a	0	1	0	1	0	1
Total	32 of 151	79%	11 of 16	31%	7	21	10	37	0	13

Created with JaCoCo 0.7.1.201405082137

AARHUS UNIVERSITET

Optional…

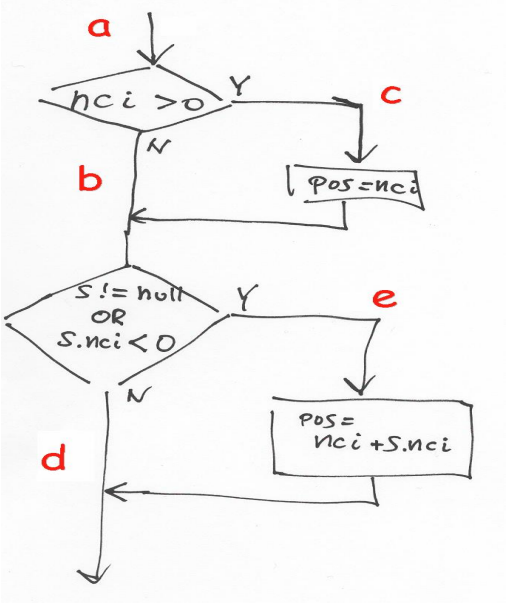
The other coverage metrics…

AARHUS UNIVERSITET

Condition coverage

- Condition coverage:
  - ***Require each condition in a decision takes all possible outcomes at least once***
- C1: nci > 0
- C2: s != null
- C3: s.nci < 0

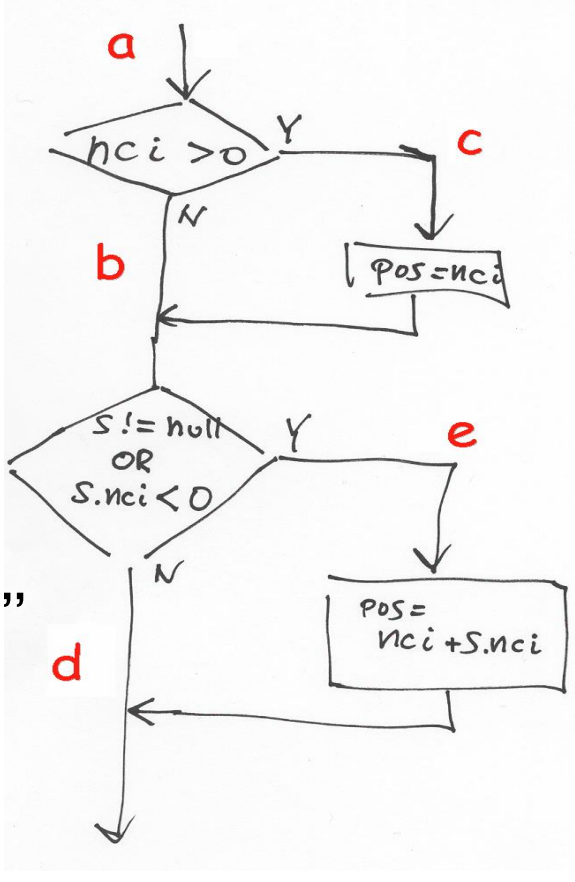
- Exercise:
  - which paths satisfy CC criterion?



AARHUS UNIVERSITET

Condition coverage

- Condition table
  - nci > 0; nci <= 0
  - s != null; s == null
  - s.nci < 0; s.nci >= 0
- Paths:
  - ace: C1, C2, C3
  - abd: !C1, !C2, !C3
- Test Cases that satisfy CC
  - `nci=-1000; s=null; "s.nci = +2000"`
  - `nci =+1000; s!=null; s.nci =-2000`
  - expected tb for both: tb = 0
  - still nci<0 and s.nci<0 defect not tried.



AARHUS UNIVERSITET

Condition coverage

- The example case here is actually not very good at showing CC as the two conditions in the decision is coupled. CC is more relevant when they are independent.
- Artificial example:
  - if (s != null || nci > 10000)
- Then DC is satisfied with (both decision outcomes tried)
  - s != null; nci <= 10000
  - s != null; nci > 10000
- while CC require for instance (both condition outcomes tried)
  - s = null; nci <= 10000
  - s != null; nci > 10000

AARHUS UNIVERSITET

Condition Coverage

- Perhaps somewhat surprising ...
- *Condition coverage does generally **not** satisfy Decision Coverage.*
- Consider a branch
- `if ( C1 && C2 ) { B }`
- Then test cases
  - TC1: !C1, C2 TC2: C1, !C2
- will satisfy CC
  - (both outcomes for both conditions)
- but not DC nor SC!
  - decision is always false; B is never executed...

AARHUS UNIVERSITET

Decision/Condition Coverage

- Combine the two to get a stronger coverage:
  - **Decision/Condition coverage.**
- However, often conditions mask each other
  - A && B && C if A==false then B and C are not evaluated
  - A || B || C if A==true then B and C not evaluated
    - in all languages with short-curcuit boolean evaluation like C, Java

- We have this already in the defective OR
- `if ( t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0 )`
- which means the last condition never has any effect: if t has a spouse, then path e is taken no matter what spouse's net capital income is…

AARHUS UNIVERSITET

DCC Coverage

- In our case DCC is already
- achieved
- Paths:
  - ace: C1, C2, C3
  - abd: !C1, !C2, !C3
- CC
  - all Cx have both Cx and !Cx
- DC
  - both if's in both True and False
- Still: miss the 'abe' path ☹