AARHUS UNIVERSITET

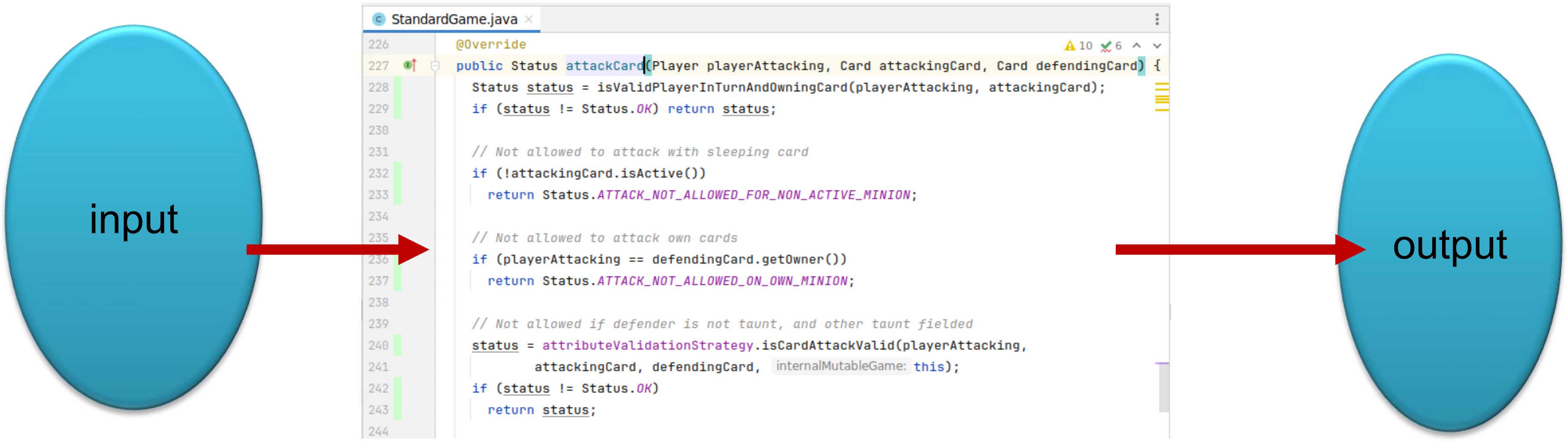
Software Engineering and Architecture

Code Coverage

Quality of your Test Cases (?)

BlackBox and WhiteBox

- Whitebox testing (glassbox / structural)



```
@Override
public Status attackCard(Player playerAttacking, Card attackingCard, Card defendingCard) {
  Status status = isValidPlayerInTurnAndOwningCard(playerAttacking, attackingCard);
  if (status != Status.OK) return status;

  // Not allowed to attack with sleeping card
  if (!attackingCard.isActive())
    return Status.ATTACK_NOT_ALLOWED_FOR_NON_ACTIVE_MINION;

  // Not allowed to attack own cards
  if (playerAttacking == defendingCard.getOwner())
    return Status.ATTACK_NOT_ALLOWED_ON_OWN_MINION;

  // Not allowed if defender is not taunt, and other taunt fielded
  status = attributeValidationStrategy.isCardAttackValid(playerAttacking,
          attackingCard, defendingCard,  internalMutableGame: this);
  if (status != Status.OK)
    return status;

```

  - *If we know the structure of code, we may construct test cases that ensures we run through all parts of it – ensure all ‘structural’ elements are ”working”.*

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"

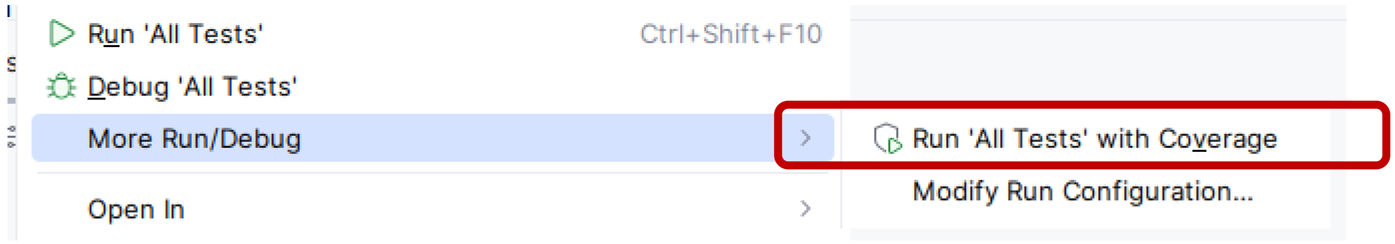
"kode-linjerne er 100% dækket af test"

  - If less than 100% are executed then T is not statement adequate for P.
  - *Note: IntelliJ calls it 'line coverage' which is actually confusing*
    - *"int x = 8; int y = 9"* is one line and two statements...

AARHUS UNIVERSITET

Example

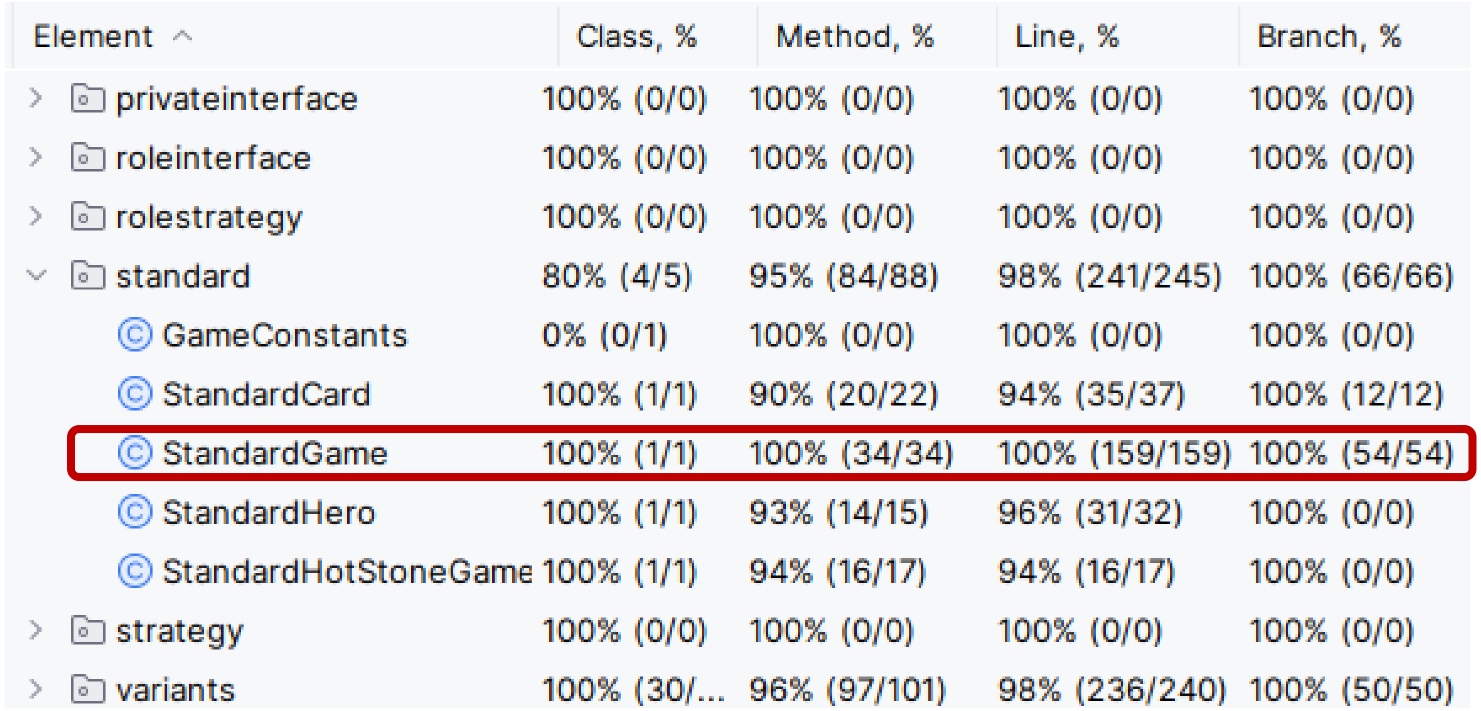
- (With some fear,) I ran IntelliJ’s code coverage tool on my own HotStone solution code…



Example

- Result

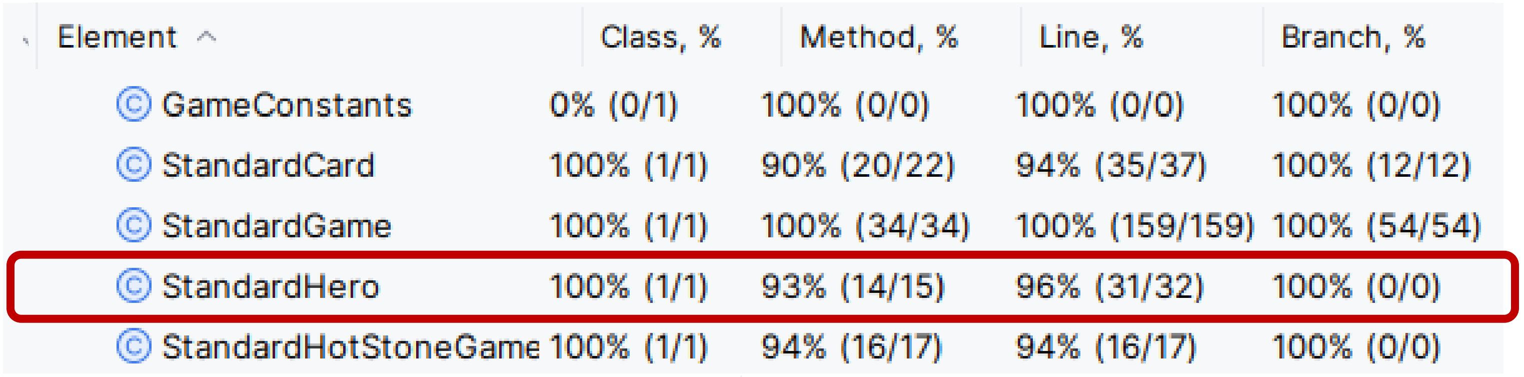
Conclusion: My StandardGame is 100% statement adequately covered by test.



Element	Class, %	Method, %	Line, %	Branch, %
privateinterface	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
roleinterface	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
rolestrategy	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
standard	80% (4/5)	95% (84/88)	98% (241/245)	100% (66/66)
GameConstants	0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
StandardCard	100% (1/1)	90% (20/22)	94% (35/37)	100% (12/12)
StandardGame	100% (1/1)	100% (34/34)	100% (159/159)	100% (54/54)
StandardHero	100% (1/1)	93% (14/15)	96% (31/32)	100% (0/0)
StandardHotStoneGame	100% (1/1)	94% (16/17)	94% (16/17)	100% (0/0)
strategy	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
variants	100% (30/...	96% (97/101)	98% (236/240)	100% (50/50)

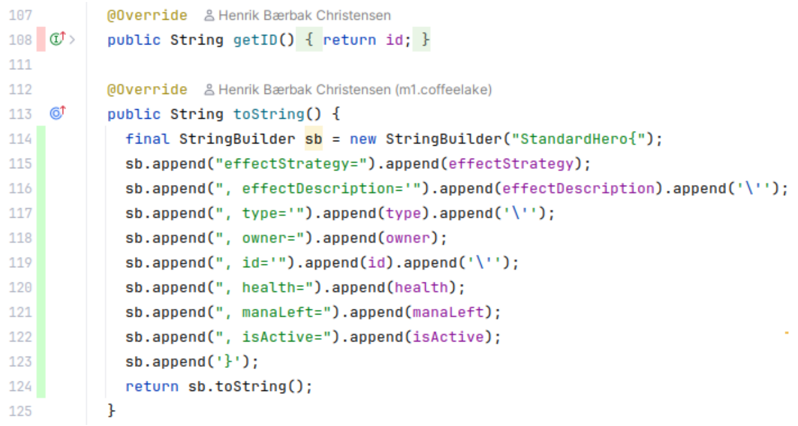
But?

- Why this low number for my Hero???



Element	Class, %	Method, %	Line, %	Branch, %
GameConstants	0% (0/1)	100% (0/0)	100% (0/0)	100% (0/0)
StandardCard	100% (1/1)	90% (20/22)	94% (35/37)	100% (12/12)
StandardGame	100% (1/1)	100% (34/34)	100% (159/159)	100% (54/54)
StandardHero	100% (1/1)	93% (14/15)	96% (31/32)	100% (0/0)
StandardHotStoneGame	100% (1/1)	94% (16/17)	94% (16/17)	100% (0/0)

- Am I to be worried?
  - No…
    - getID() is *obvious implementation...*
    - *toString() is auto-generated by IntelliJ*



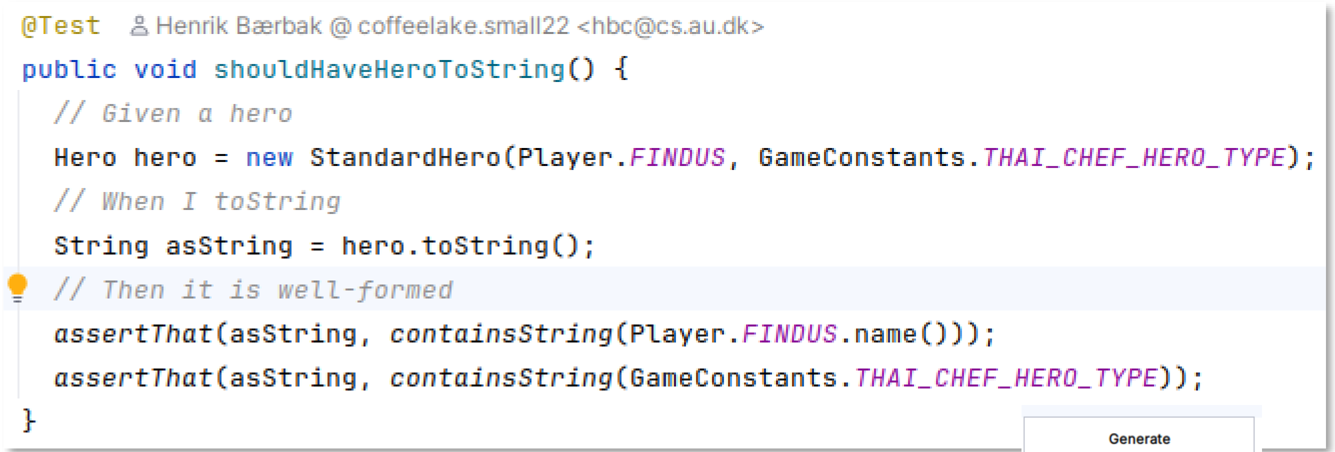
```
@Override  Henrik Bærbak Christensen
public String getID() { return id; }

@Override  Henrik Bærbak Christensen (m1.coffeelake)
public String toString() {
  final StringBuilder sb = new StringBuilder("StandardHero{");
  sb.append("effectStrategy=").append(effectStrategy);
  sb.append(", effectDescription='").append(effectDescription).append('\'');
  sb.append(", type='").append(type).append('\'');
  sb.append(", owner=").append(owner);
  sb.append(", id='").append(id).append('\'');
  sb.append(", health=").append(health);
  sb.append(", manaLeft=").append(manaLeft);
  sb.append(", isActive=").append(isActive);
  sb.append('}');
  return sb.toString();
}
```

Key Points

- In my toolbox I perceive Code Coverage tools to help me
  - Spot *weaknesses* in my testing
    - Huh??? Why do I not get 100% statement coverage of my Hero???
    - I will have to investigate that!
- *But getting 100% often introduce extra work with no value*
  - Spring 2025 I actually invested time to increase my coverage

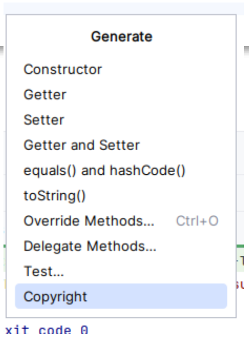
A Coverage Induced Test

- So I actually looked for missing line coverage, and then introduced test cases to cover them
  - Ala



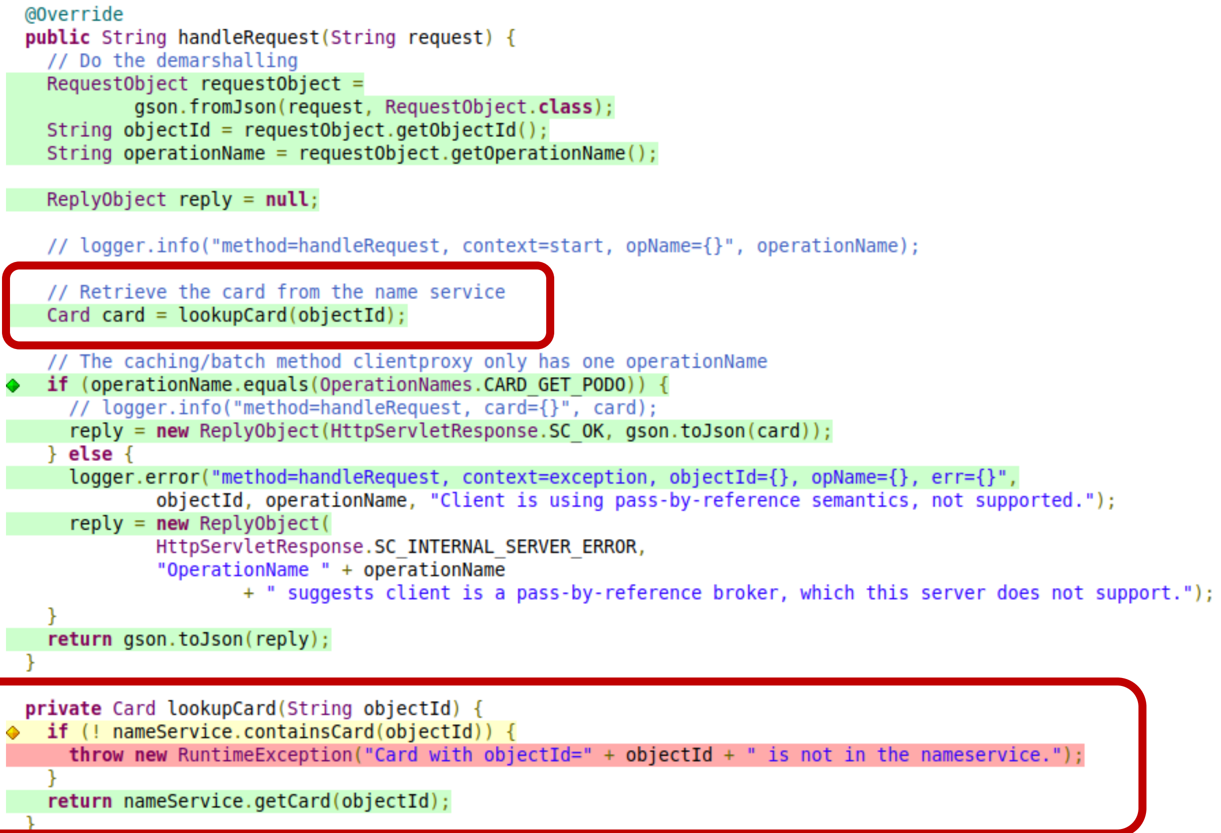
```
@Test   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk>
public void shouldHaveHeroToString() {
  // Given a hero
  Hero hero = new StandardHero(Player.FINDUS, GameConstants.THAI_CHEF_HERO_TYPE);
  // When I toString
  String asString = hero.toString();
  // Then it is well-formed
  assertThat(asString, containsString(Player.FINDUS.name()));
  assertThat(asString, containsString(GameConstants.THAI_CHEF_HERO_TYPE));
}
```

- But...
  - It is not TDD – toString() was generated by Intellij
  - Thus – bit waste of time...



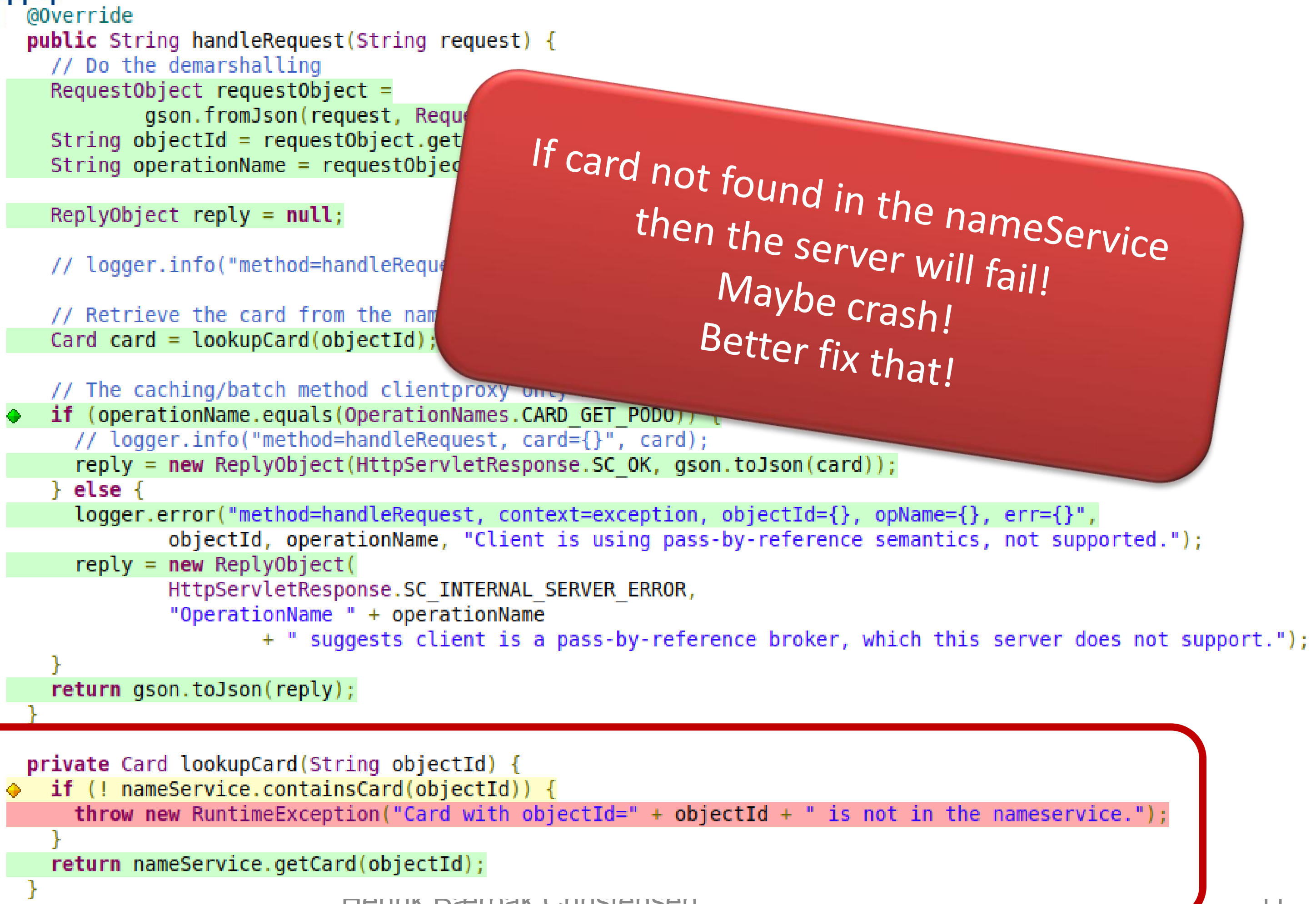
But good to find weak spots

- Coverage helps me to spot aspects I have overlooked
  - Often the *red lines are in error handling*
- Example
  - Server side code to handle Card



```
@Override
public String handleRequest(String request) {
  // Do the demarshalling
  RequestObject requestObject =
          gson.fromJson(request, RequestObject.class);
  String objectId = requestObject.getObjectId();
  String operationName = requestObject.getOperationName();

  ReplyObject reply = null;

  // logger.info("method=handleRequest, context=start, opName={}", operationName);

  // Retrieve the card from the name service
  Card card = lookupCard(objectId);

  // The caching/batch method clientproxy only has one operationName
  if (operationName.equals(OperationNames.CARD_GET_PODO)) {
    // logger.info("method=handleRequest, card={}", card);
    reply = new ReplyObject(HttpServletResponse.SC_OK, gson.toJson(card));
  } else {
    logger.error("method=handleRequest, context=exception, objectId={}, opName={}, err={}",
            objectId, operationName, "Client is using pass-by-reference semantics, not supported.");
    reply = new ReplyObject(
            HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
            "OperationName " + operationName
                    + " suggests client is a pass-by-reference broker, which this server does not support.");
  }
  return gson.toJson(reply);
}

private Card lookupCard(String objectId) {
  if (! nameService.containsCard(objectId)) {
    throw new RuntimeException("Card with objectId=" + objectId + " is not in the nameservice.");
  }
  return nameService.getCard(objectId);
}
```

Example



```
@Override
public String handleRequest(String request) {
  // Do the demarshalling
  RequestObject requestObject =
          gson.fromJson(request, Reque
  String objectId = requestObject.get
  String operationName = requestObjec

  ReplyObject reply = null;

  // logger.info("method=handleReque

  // Retrieve the card from the nam
  Card card = lookupCard(objectId);

  // The caching/batch method clientproxy
  if (operationName.equals(OperationNames.CARD_GET_PODO)) {
    // logger.info("method=handleRequest, card={}", card);
    reply = new ReplyObject(HttpServletResponse.SC_OK, gson.toJson(card));
  } else {
    logger.error("method=handleRequest, context=exception, objectId={}, opName={}, err={}",
            objectId, operationName, "Client is using pass-by-reference semantics, not supported.");
    reply = new ReplyObject(
            HttpServletResponse.SC_INTERNAL_SERVER_ERROR,
            "OperationName " + operationName
                    + " suggests client is a pass-by-reference broker, which this server does not support.");
  }
  return gson.toJson(reply);
}

private Card lookupCard(String objectId) {
  if (! nameService.containsCard(objectId)) {
    throw new RuntimeException("Card with objectId=" + objectId + " is not in the nameservice.");
  }
  return nameService.getCard(objectId);
}
```

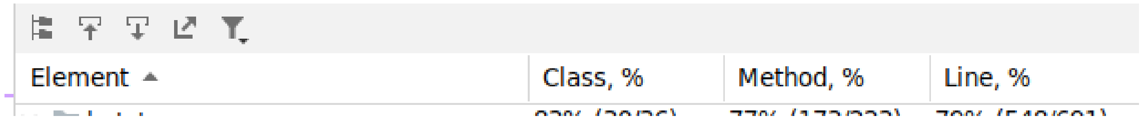
If card not found in the nameService
then the server will fail!
Maybe crash!
Better fix that!

AARHUS UNIVERSITET

Adequacy Criteria

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
  - method coverage
  - decision coverage
  - path coverage, and many more
- Other types of criteria
  - use case coverage (system level)
  - interface coverage (integration level)



Element	Class, %	Method, %	Line, %

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

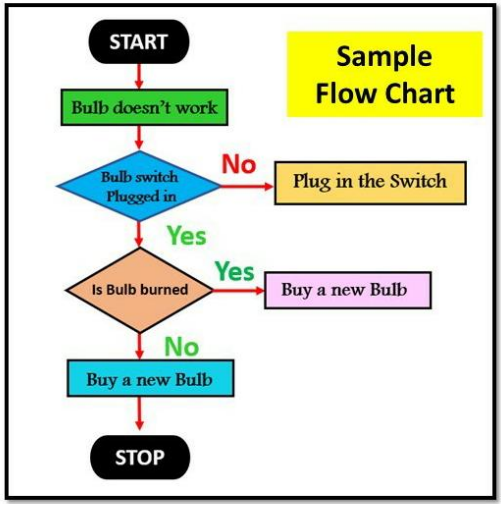
- Overall, there are a number of metrics for coverage:
  - statement coverage
  - decision coverage
  - condition coverage
  - decision/condition coverage
  - multiple-condition coverage
  - path coverage

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

AARHUS UNIVERSITET

Flow graph/Flow chart

- The flow graph (Da: Rute diagram) is simply a diagram that shows the flow of execution. Really old school ☺
- It defines a graph where nodes are *basic primes* (block, decision, iteration) and edges are control flow between them (the ProgramCounter ☺).



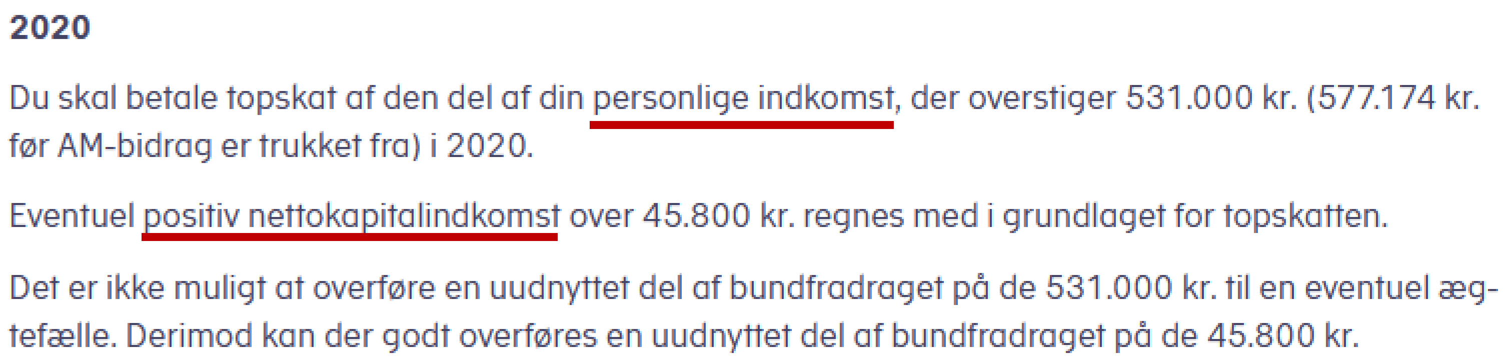
AARHUS UNIVERSITET

Example

Danish Tax

Source of Complex Algorithms

- Danish Topskat



2020

Du skal betale topskat af den del af din personlige indkomst, der overstiger 531.000 kr. (577.174 kr. før AM-bidrag er trukket fra) i 2020.

Eventuel positiv nettokapitalindkomst over 45.800 kr. regnes med i grundlaget for topskatten.

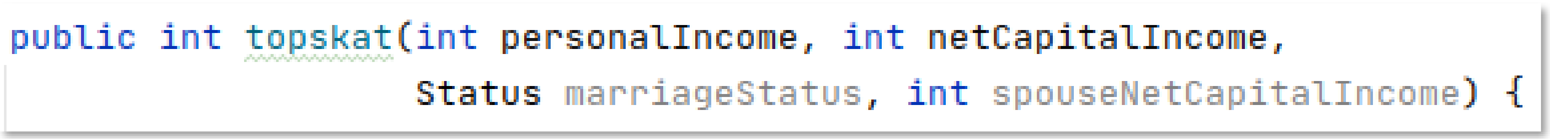
Det er ikke muligt at overføre en uudnyttet del af bundfradraget på de 531.000 kr. til en eventuel æg-tefælle. Derimod kan der godt overføres en uudnyttet del af bundfradraget på de 45.800 kr.

- Limitation
  - Only look at *unmarried* person (which simplifies it greatly)
  - Exercise: complete the analysis ☺

AARHUS UNIVERSITET

Design

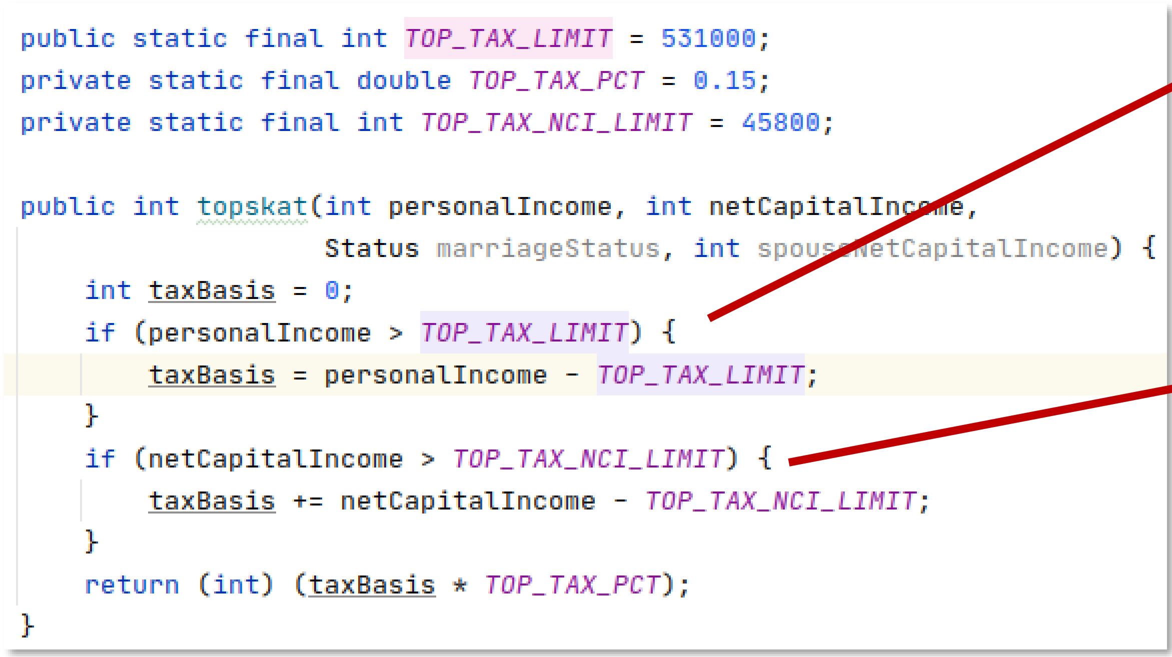
- Direct – functional – approach



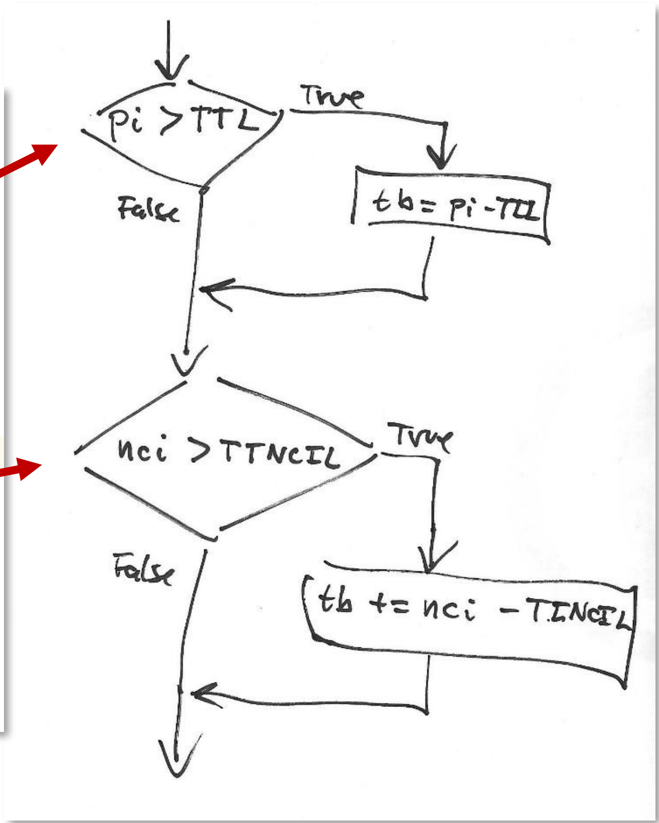
```
public int topskat(int personalIncome, int netCapitalIncome,
                   Status marriageStatus, int spouseNetCapitalIncome) {
```

- EC analysis?
  - Booleans
    - Below and above ‘bundfradrag’
      - 531.000 for personal income   true/false
      - 45.800 for capital income   true/false
- However, we will look at the code for a WB analysis
- Exercise: Do the EC analysis and compare…

Code and Flow Graph



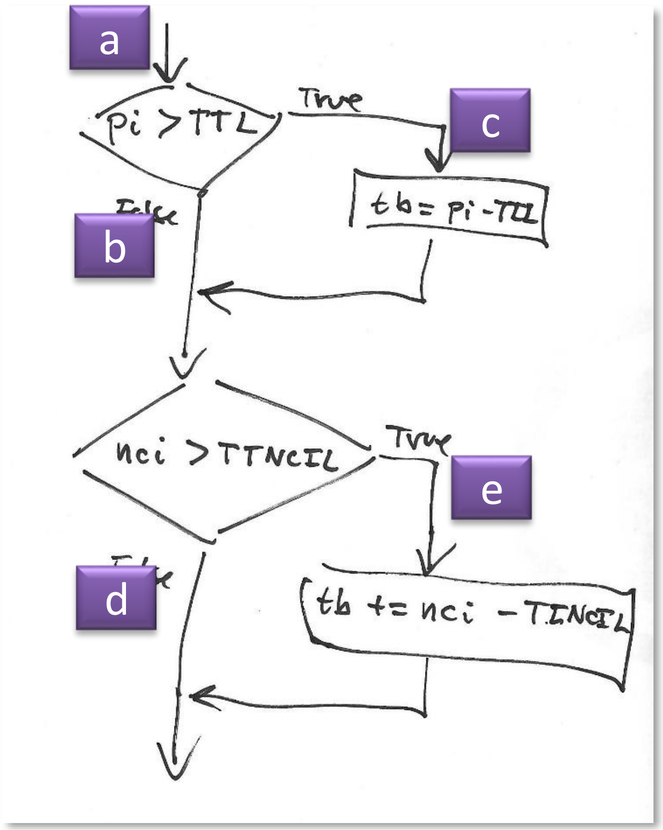
```
public static final int TOP_TAX_LIMIT = 531000;
private static final double TOP_TAX_PCT = 0.15;
private static final int TOP_TAX_NCI_LIMIT = 45800;

public int topskat(int personalIncome, int netCapitalIncome,
                   Status marriageStatus, int spouseNetCapitalIncome) {
    int taxBasis = 0;
    if (personalIncome > TOP_TAX_LIMIT) {
        taxBasis = personalIncome - TOP_TAX_LIMIT;
    }
    if (netCapitalIncome > TOP_TAX_NCI_LIMIT) {
        taxBasis += netCapitalIncome - TOP_TAX_NCI_LIMIT;
    }
    return (int) (taxBasis * TOP_TAX_PCT);
}
```



Note: On purpose this code is full of defects!

Paths in Flow Graph

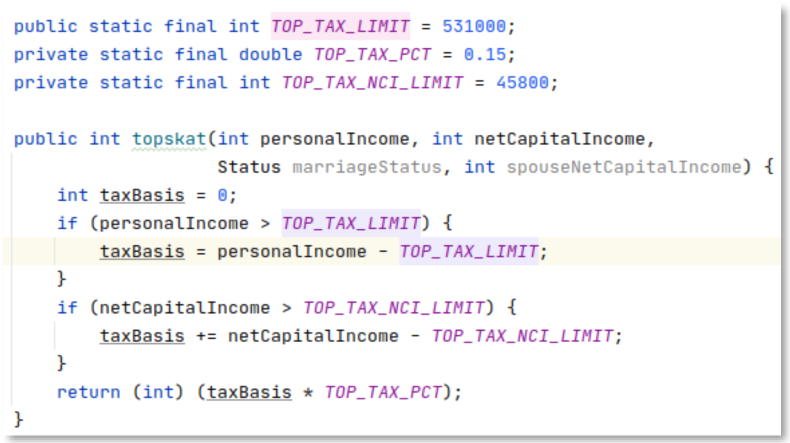
- We can name *paths* in the flowgraph by the subpaths
- E.g
  - acd
  - abe
  - abd
  - ace



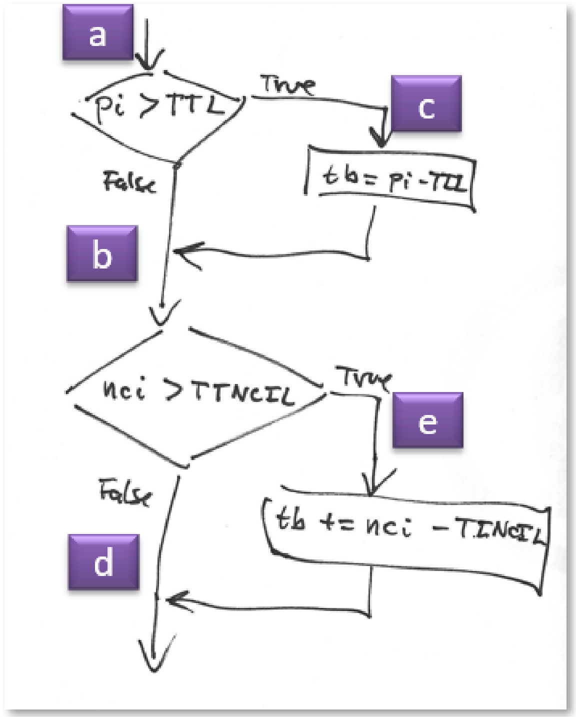
Statement coverage

- Statement coverage (SC):
  - ***Requires every statement in the program to be executed at least once***

- Exercise:
  – which path satisfy SC criterion?



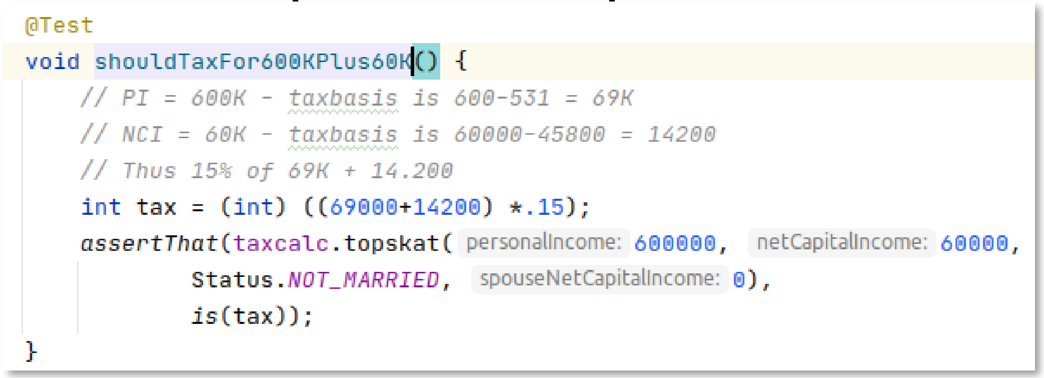
```
public static final int TOP_TAX_LIMIT = 531000;
private static final double TOP_TAX_PCT = 0.15;
private static final int TOP_TAX_NCI_LIMIT = 45800;

public int topskat(int personalIncome, int netCapitalIncome,
                   Status marriageStatus, int spouseNetCapitalIncome) {
    int taxBasis = 0;
    if (personalIncome > TOP_TAX_LIMIT) {
        taxBasis = personalIncome - TOP_TAX_LIMIT;
    }
    if (netCapitalIncome > TOP_TAX_NCI_LIMIT) {
        taxBasis += netCapitalIncome - TOP_TAX_NCI_LIMIT;
    }
    return (int) (taxBasis * TOP_TAX_PCT);
}
```



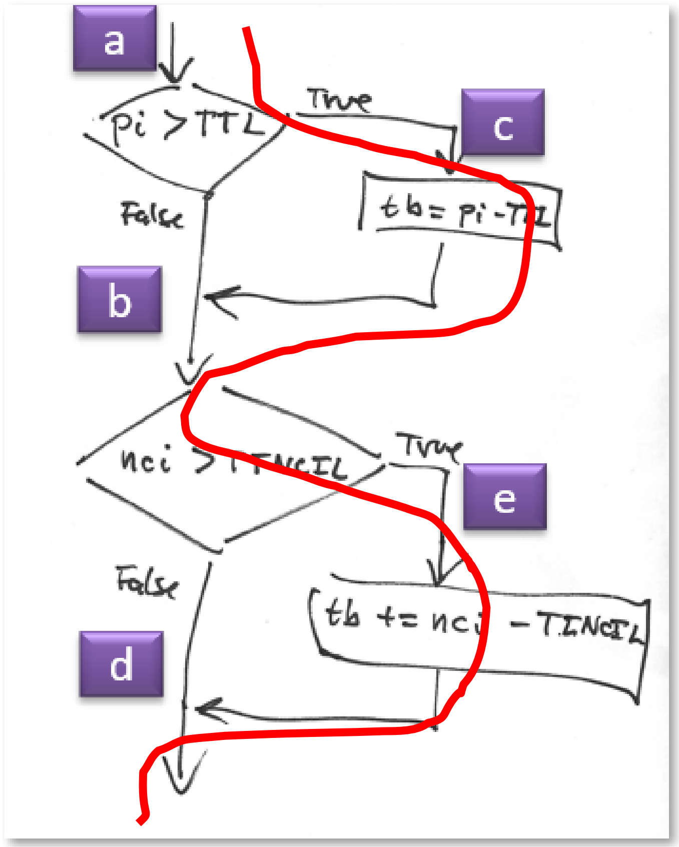
Statement coverage

- SC criterion is pretty weak.
- Path **ace** is enough.

- A single test case suffice, which ensures path c and path e...



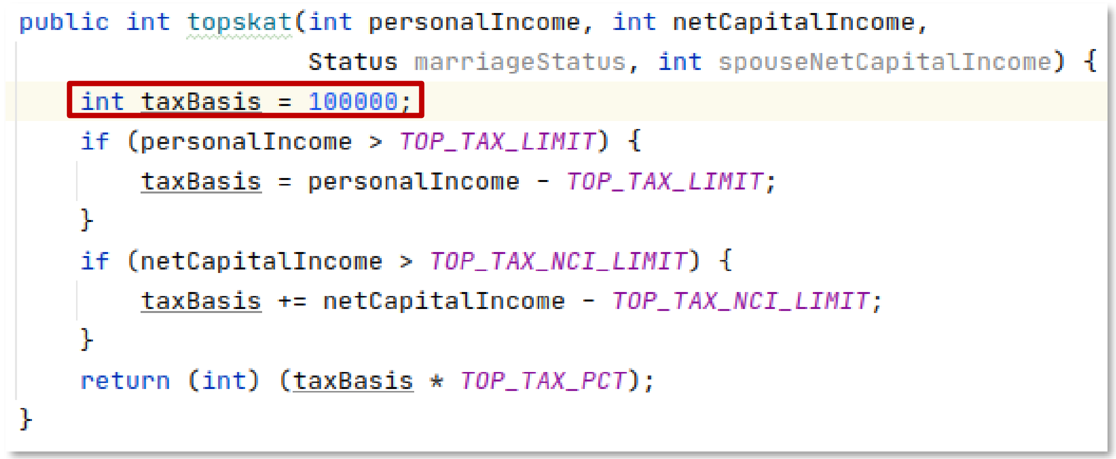
```
@Test
void shouldTaxFor600KPlus60K() {
    // PI = 600K - taxbasis is 600-531 = 69K
    // NCI = 60K - taxbasis is 60000-45800 = 14200
    // Thus 15% of 69K + 14.200
    int tax = (int) ((69000+14200) *.15);
    assertThat(taxcalc.topskat( personalIncome: 600000,  netCapitalIncome: 60000,
            Status.NOT_MARRIED,  spouseNetCapitalIncome: 0),
            is(tax));
}
```



AARHUS UNIVERSITET

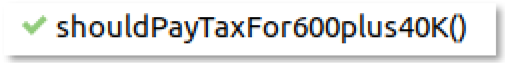
But SC is a Weak Coverage

- Easy to introduce an error that our test case will not find



```
public int topskat(int personalIncome, int netCapitalIncome,
                   Status marriageStatus, int spouseNetCapitalIncome) {
    int taxBasis = 100000;
    if (personalIncome > TOP_TAX_LIMIT) {
        taxBasis = personalIncome - TOP_TAX_LIMIT;
    }
    if (netCapitalIncome > TOP_TAX_NCI_LIMIT) {
        taxBasis += netCapitalIncome - TOP_TAX_NCI_LIMIT;
    }
    return (int) (taxBasis * TOP_TAX_PCT);
}
```

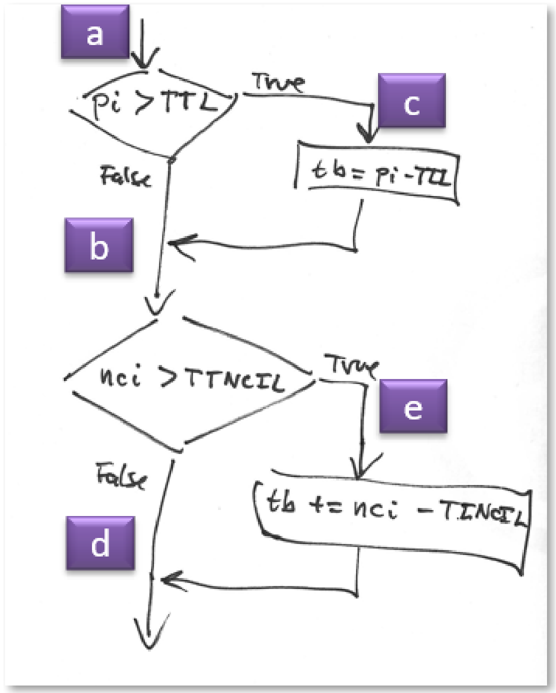
Code is correct *only* when personal income is above 531.000 kr. (As taxBasis is reset correctly, then)



✓ shouldPayTaxFor600plus40K()

Decision coverage

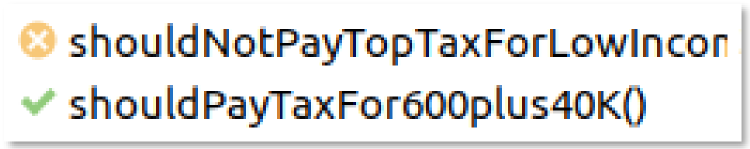
- Decision coverage (branch coverage):
- ***Requires each decision has a true and false outcome at least once***
- Decision 1:
  - pi > TTL
- Decision 2
  - nci > TTNCIL
- Exercise:
  - which paths satisfy DC criterion?



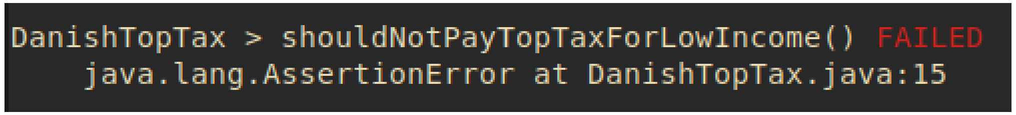
AARHUS UNIVERSITET

Decision coverage

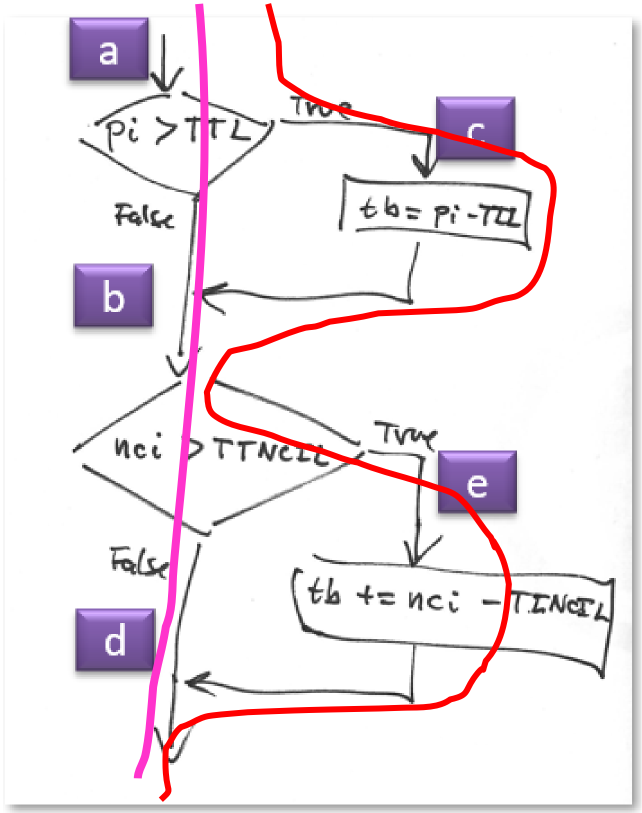
- DC criterion is better
- TC1: D1 true D2 true
- TC2: D1 false D2 false
  - Will find our ‘taxbasis = 100.000’ bug



✗ shouldNotPayTopTaxForLowIncom
✓ shouldPayTaxFor600plus40K()

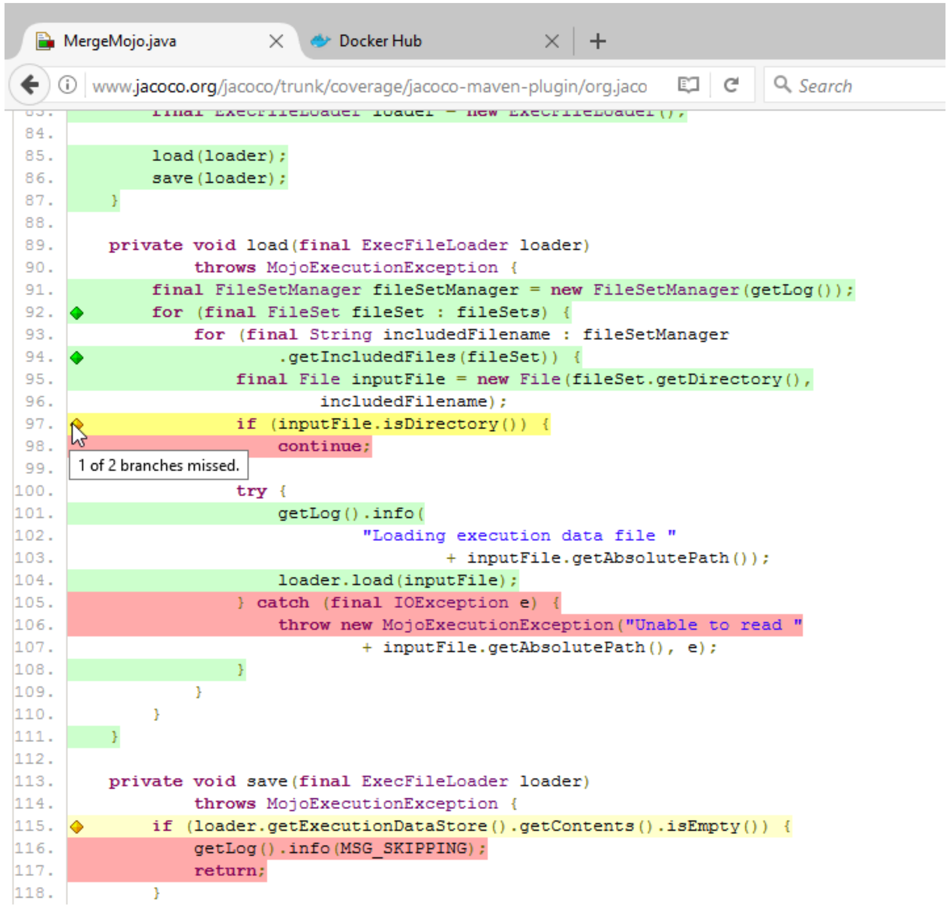


```
DanishTopTax > shouldNotPayTopTaxForLowIncome() FAILED
    java.lang.AssertionError at DanishTopTax.java:15
```



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

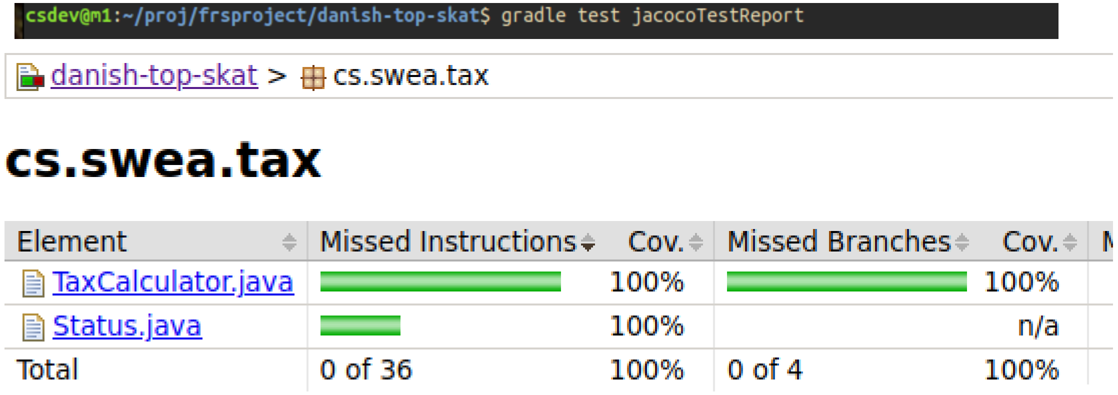


MergeMojo.java
Docker Hub
www.jacoco.org/jacoco/trunk/coverage/jacoco-maven-plugin/org.jaco
Search
1 of 2 branches missed.

```
        final ExecFileLoader loader = new ExecFileLoader();

        load(loader);
        save(loader);
    }

    private void load(final ExecFileLoader loader)
            throws MojoExecutionException {
        final FileSetManager fileSetManager = new FileSetManager(getLog());
        for (final FileSet fileSet : fileSets) {
            for (final String includedFilename : fileSetManager
                    .getIncludedFiles(fileSet)) {
                final File inputFile = new File(fileSet.getDirectory(),
                        includedFilename);
                if (inputFile.isDirectory()) {
                    continue;
                }
                try {
                    getLog().info(
                            "Loading execution data file "
                                    + inputFile.getAbsolutePath());
                    loader.load(inputFile);
                } catch (final IOException e) {
                    throw new MojoExecutionException("Unable to read "
                            + inputFile.getAbsolutePath(), e);
                }
            }
        }
    }

    private void save(final ExecFileLoader loader)
            throws MojoExecutionException {
        if (loader.getExecutionDataStore().getContents().isEmpty()) {
            getLog().info(MSG_SKIPPING);
            return;
        }
```

AARHUS UNIVERSITET

Our Tax Code

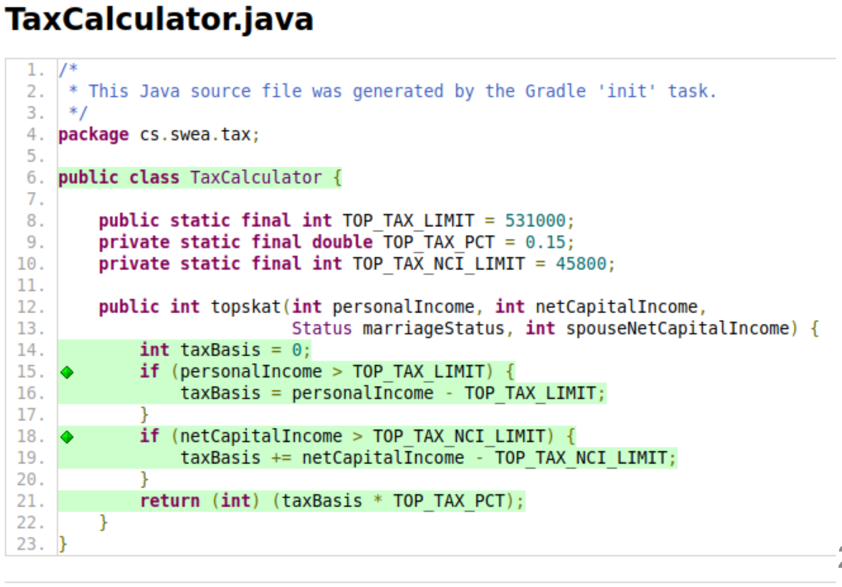


```
csdev@m1:~/proj/frsproject/danish-top-skat$ gradle test jacocoTestReport
```

danish-top-skat > cs.swea.tax

cs.swea.tax

Element	Missed Instructions	Cov.	Missed Branches	Cov.
TaxCalculator.java	🟩	100%	🟩	100%
Status.java	🟩	100%		n/a
Total	0 of 36	100%	0 of 4	100%



TaxCalculator.java

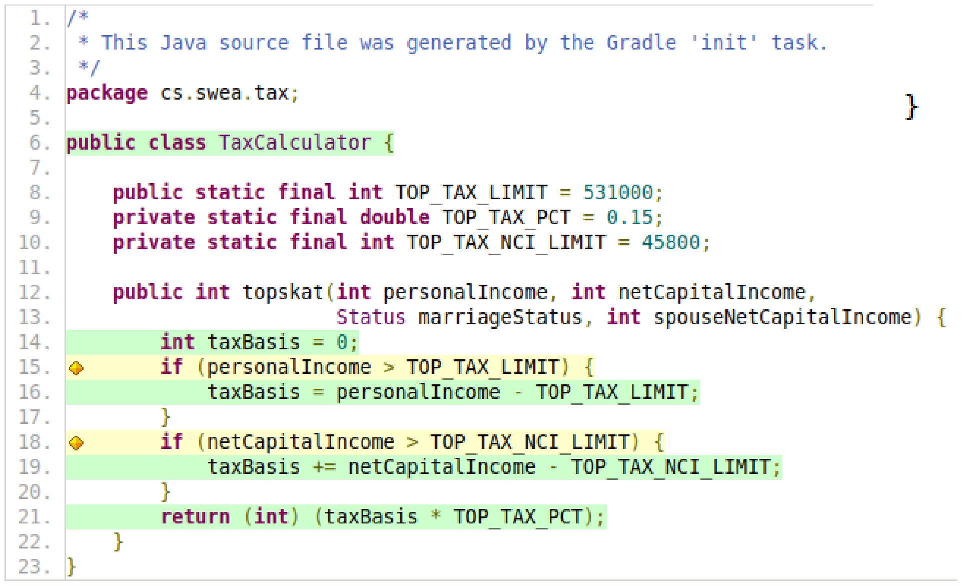
```
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
package cs.swea.tax;

public class TaxCalculator {

    public static final int TOP_TAX_LIMIT = 531000;
    private static final double TOP_TAX_PCT = 0.15;
    private static final int TOP_TAX_NCI_LIMIT = 45800;

    public int topskat(int personalIncome, int netCapitalIncome,
                       Status marriageStatus, int spouseNetCapitalIncome) {
        int taxBasis = 0;
        if (personalIncome > TOP_TAX_LIMIT) {
            taxBasis = personalIncome - TOP_TAX_LIMIT;
        }
        if (netCapitalIncome > TOP_TAX_NCI_LIMIT) {
            taxBasis += netCapitalIncome - TOP_TAX_NCI_LIMIT;
        }
        return (int) (taxBasis * TOP_TAX_PCT);
    }
}
```

Flagging Missed Branches:

- Disabled the low income test case and ran again...

```
@Disabled
@Test
void shouldNotPayTopTaxForLowIncome() {
    assertThat(taxcalc.topskat(520000, 10000,
            Status.NOT_MARRIED, 0),
            is(0));
}
```

TaxCalculator.java

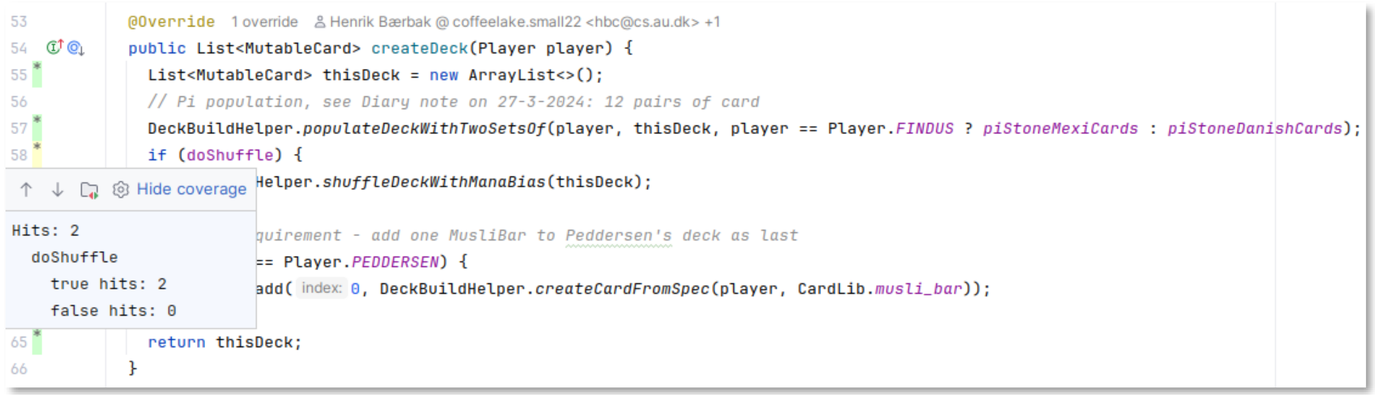


```
/*
 * This Java source file was generated by the Gradle 'init' task.
 */
package cs.swea.tax;

public class TaxCalculator {

    public static final int TOP_TAX_LIMIT = 531000;
    private static final double TOP_TAX_PCT = 0.15;
    private static final int TOP_TAX_NCI_LIMIT = 45800;

    public int topskat(int personalIncome, int netCapitalIncome,
                       Status marriageStatus, int spouseNetCapitalIncome) {
        int taxBasis = 0;
        if (personalIncome > TOP_TAX_LIMIT) {
            taxBasis = personalIncome - TOP_TAX_LIMIT;
        }
        if (netCapitalIncome > TOP_TAX_NCI_LIMIT) {
            taxBasis += netCapitalIncome - TOP_TAX_NCI_LIMIT;
        }
        return (int) (taxBasis * TOP_TAX_PCT);
    }
}
```

You can actually see if it is the true or false branch that is missed here. How?

IntelliJ

- Up until E24, IntelliJ did not do branch coverage, but it has changed ☺
  - Click on the ‘yellow’ marking and you will see which branch was and was not taken!



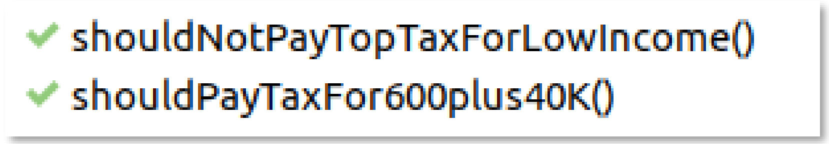
```
@Override   1 override   Henrik Bærbak @ coffeelake.small22 <hbc@cs.au.dk> +1
public List<MutableCard> createDeck(Player player) {
  List<MutableCard> thisDeck = new ArrayList<>();
  // Pi population, see Diary note on 27-3-2024: 12 pairs of card
  DeckBuildHelper.populateDeckWithTwoSetsOf(player, thisDeck, player == Player.FINDUS ? piStoneMexiCards : piStoneDanishCards);
  if (doShuffle) {
                 Helper.shuffleDeckWithManaBias(thisDeck);

                 quirement - add one MusliBar to Peddersen's deck as last
                 == Player.PEDDERSEN) {
                 add( index: 0, DeckBuildHelper.createCardFromSpec(player, CardLib.musli_bar));

  return thisDeck;
}
```

```
Hits: 2
  doShuffle
    true hits: 2
    false hits: 0
```

  - The branch was not covered for ‘doShuffle == false’
    - [Puzzling as this is the test flag; but was used in another gradle project, so...]

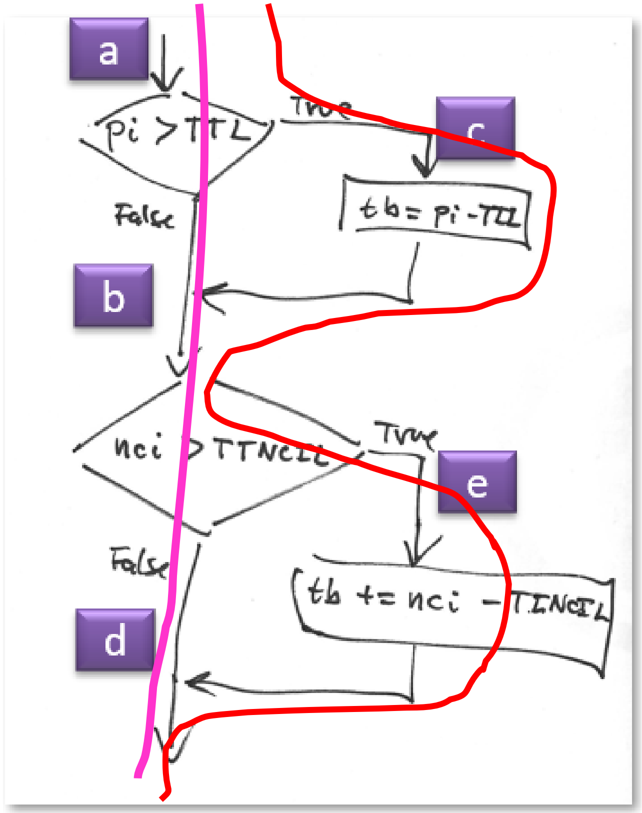
Decision coverage

- But an import aspect remains in the code that DC's does not find?
- Which is?



✓ shouldNotPayTopTaxForLowIncome()
✓ shouldPayTaxFor600plus40K()

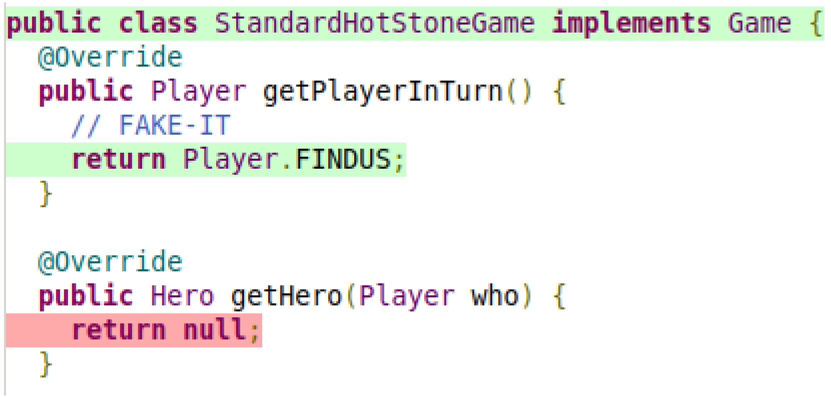
- … that it only handles unmarried persons
  - The married stuff is not implemented yet ☺



AARHUS UNIVERSITET

WhiteBox Test…

- ... looks at *the code that is*
- Not the specification ☺…
  - Thus, the whole un-implemented part of handling married people is not treated at all, yet the test cases pass and the DC coverage is adequate…
- Another example:
  - Is 100% statement coverage of ‘getPlayerInTurn()’ a good thing?



```
public class StandardHotStoneGame implements Game {
  @Override
  public Player getPlayerInTurn() {
    // FAKE-IT
    return Player.FINDUS;
  }

  @Override
  public Hero getHero(Player who) {
    return null;
  }
```

And…

- The other coverage metrics are not considered in SWEA.
  - Like *multiple-condition coverage*
    - *If (a && b && c && !d)*
      - All combinations of true/false of a, b, c, and d must be covered

- And most Code Coverage tools out there cannot even 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...

AARHUS UNIVERSITET

Summary

AARHUS UNIVERSITET

Code Coverage Metrics

- Statement Coverage
  - Tests execute every **statement** at least once
- Decision Coverage
  - Tests execute every **decision** to *true* AND *false* at least once

- Code Coverage Tool
  - Is nice to review in order to inspect potential code blocks that you have forgotten to write test code for…