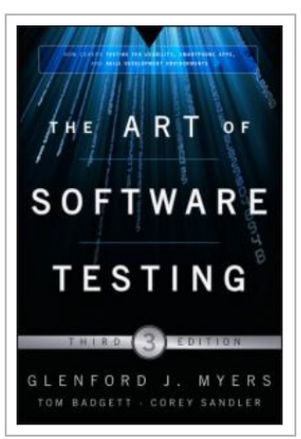
Lecture 18

ECE 1145: Software Construction and Evolution

Code Coverage

Announcements

- Iteration 6: Compositional Design due Nov. 7
  - Bonus: EtaCiv due Dec. 12
- References
  - The Art of Software Testing, Myers 1979 (library.pitt.edu)
  - A Practical Tutorial on Modified Condition/Decision Coverage (NASA)
- Midterm grades posted (see Gradescope)
- Midterm survey on Canvas



Read Online

Download Book

The Art of Software Testing

Glenford J. Myers, Corey Sandler, and Tom Badgett

Availability

Your institution has unlimited access to this book.

Read Online

Download Book

Get all pages, require free third-party software, Check out this book for up to 365 days.

Download PDF Chapter

Get up to 51 pages, use any PDF software, does not expire.

13 pa

51 pa pr

Questions for Today

How can we measure how much of our code is tested?

How can we find parts of our code that are untested?

Recall: Types of Testing

Black-box testing: The UUT is treated as an opaque "black box"
→ Use the **specification** of the UUT and a **general knowledge** of programming techniques, constructs, and common programming mistakes to guide testing

White-box testing: The full implementation of the UUT is known (transparent or "white box")
→ Actual code can be inspected to generate test cases

Code Structure

Generally a program is a combination of three types of structures:

- **Sequences**: blocks of code, sequential operations
- **Selections**: decisions, conditions, handled with if, switch, etc.
- **Iterations:** repetitions, implemented with loops (for, while, etc.)

Code Structure

Generally a program is a combination of three types of structures:

- **Sequences**: blocks of code, sequential operations
- **Selections**: decisions, conditions, handled with if, switch, etc.
- **Iterations:** repetitions, implemented with loops (for, while, etc.)

If we know the structures contained in a program (white-box), we can evaluate the ability of our test suites to **exercise** these structures.

Code Coverage

How much of our production code is executed by our test suite?

→ Adequacy or **coverage**

Code Coverage

How much of our production code is executed by our test suite?

→ Adequacy or **coverage**

→ Goal is 100%, but not always as simple as lines executed…

Code Coverage

How much of our production code is executed by our test suite?

→ Adequacy or **coverage**

→ Goal is 100%, but not always as simple as lines executed…

Criteria:

- Statement coverage
- Decision coverage
- etc.

Code Coverage

How much of our production code is executed by our test suite?

→ Adequacy or **coverage**

→ Goal is 100%, but not always as simple as lines executed…

Criteria:

- Statement coverage
- Decision coverage
- etc.

- Interface coverage (integration)
- Use case coverage (system)

Aspects of Testing

White-box Testing	**Black-box Testing**
Late in development process	Can start earlier
Only feasible for smaller UUTs	Can use at the system level
Expensive for unstable UUTs (tied to implementation)	Tests continue to apply as long as behavior and interface are stable

Aspects of Testing

White-box Testing	Black-box Testing
Late in development process	Can start earlier
Only feasible for smaller UUTs	Can use at the system level
Expensive for unstable UUTs (tied to implementation)	Tests continue to apply as long as behavior and interface are stable
“Structural coverage”	“Requirements coverage”

Aspects of Testing

White-box Testing	Black-box Testing
Late in development process	Can start earlier
Only feasible for smaller UUTs	Can use at the system level
Expensive for unstable UUTs (tied to implementation)	Tests continue to apply as long as behavior and interface are stable
“Structural coverage”	“Requirements coverage”

White-box testing is still based on requirements!

Aspects of Testing

Requirements Coverage: required/intended functionality is properly implemented

Structural Coverage: all code is reachable and adequately tested (behavior matches requirements)

Test Coverage

Structural coverage:

- Statement coverage
- Decision coverage
- Condition coverage
- Condition/Decision coverage
- Multiple-condition coverage
- Path coverage

Test Coverage

Structural coverage:

- **Statement coverage**
- **Decision coverage**
- Condition coverage
- Condition/Decision coverage
- Multiple-condition coverage
- Path coverage

Test Coverage

Structural coverage:

- Statement coverage
- Decision coverage

Condition: true/false
Decision: use conditions to decide path of the code

Test Coverage

Structural coverage:

- Statement coverage
- Decision coverage

Condition: true/false
Decision: use conditions to decide path of the code

All relate to the **flow graph** of the code

Test Coverage

Flow graph:

- Nodes are code structures (blocks, decisions, iterations)
- Edges represent control flow

Test Coverage

Example: Calculating a tax bracket based on personal income, net capital income, spouse net capital income

```
calculateBracket(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;
```

Test Coverage

Example: Calculating a tax bracket based on personal income, net capital income, spouse net capital income

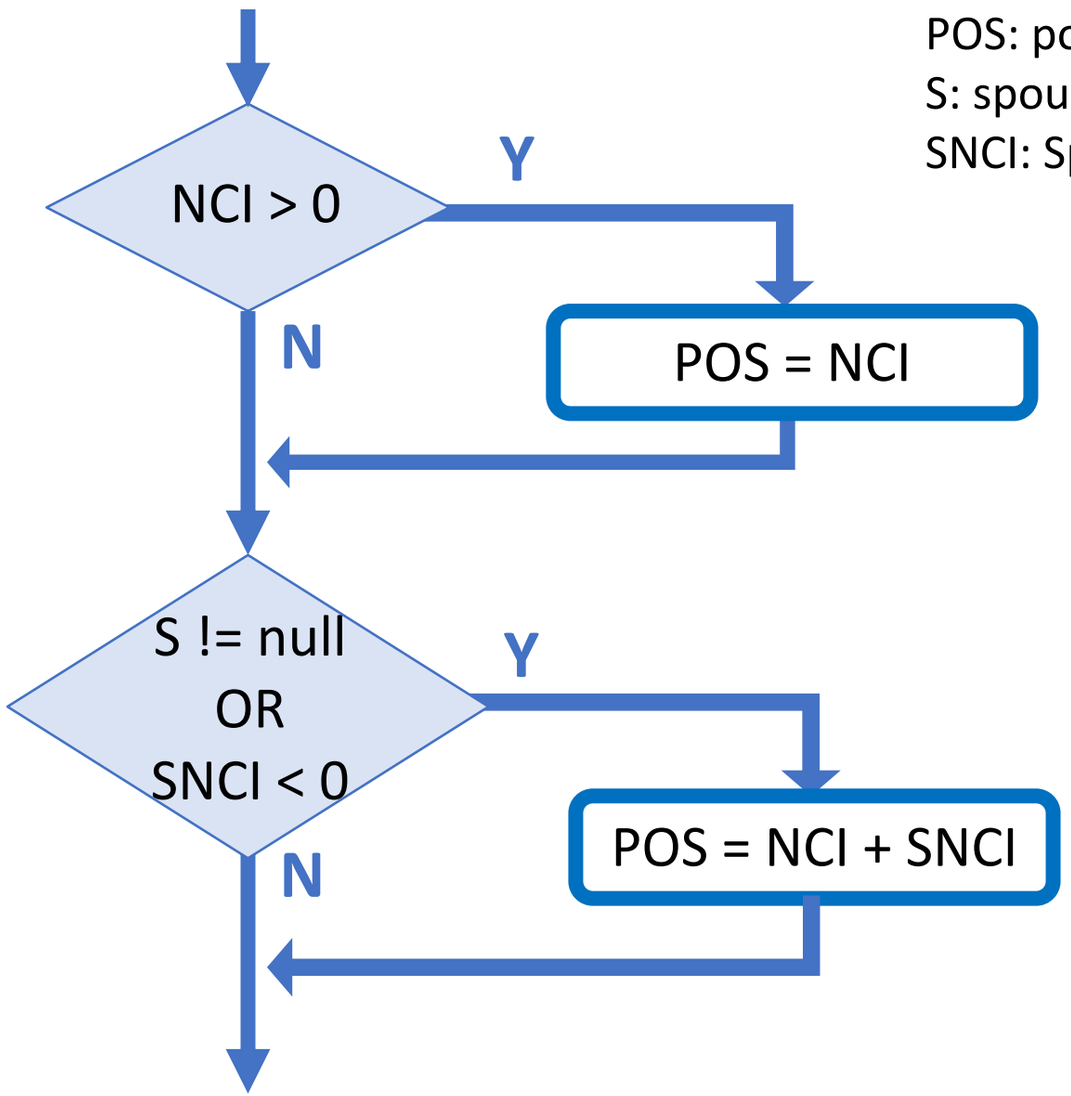
There are defects here!
Can we find them with testing?

```
calculateBracket(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;
```

Test Coverage

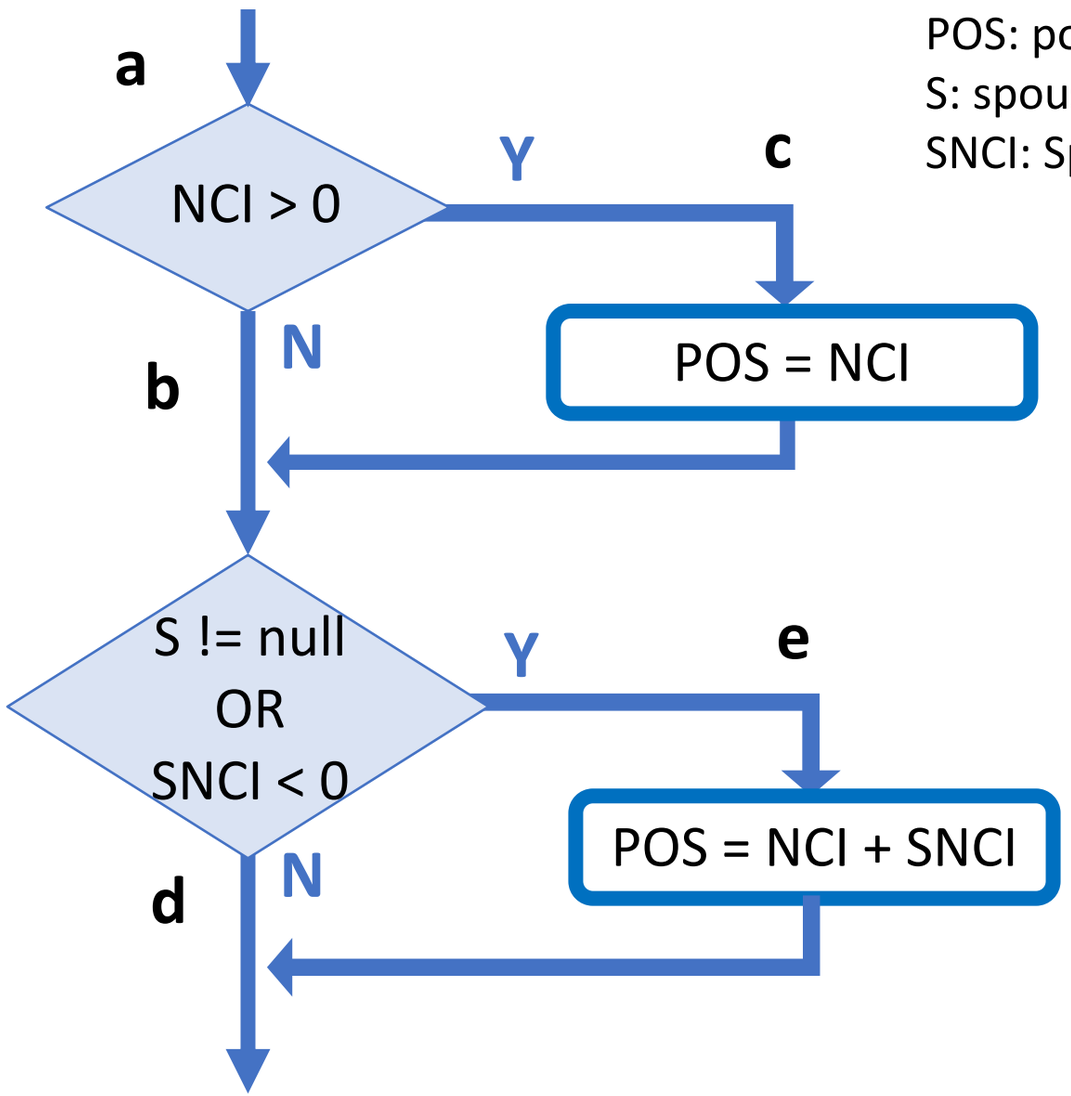
NCI: Net capital income
POS: positive net capital income
S: spouse
SNCI: Spouse net capital income

Flow graph:



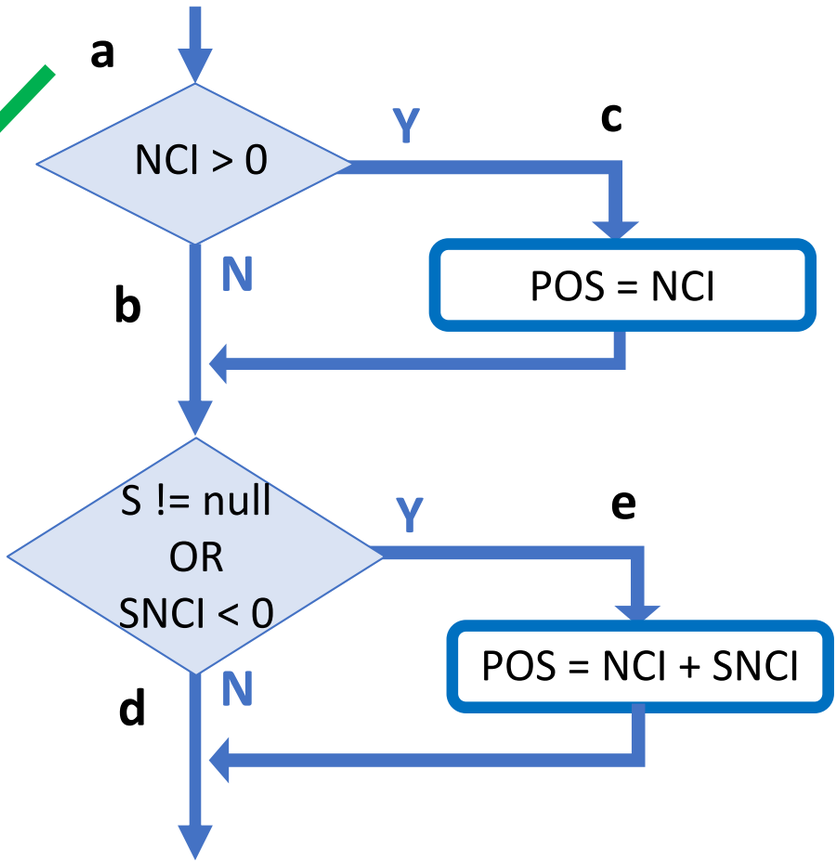
Test Coverage

Flow graph:



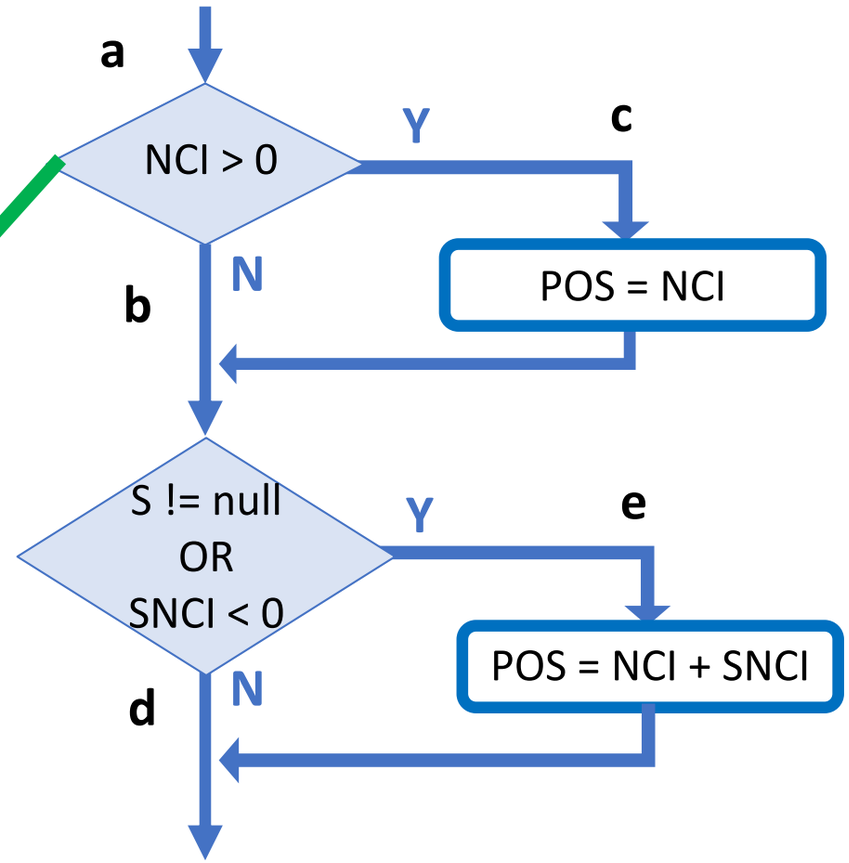
NCI: Net capital income
POS: positive net capital income
S: spouse
SNCI: Spouse net capital income

Test Coverage



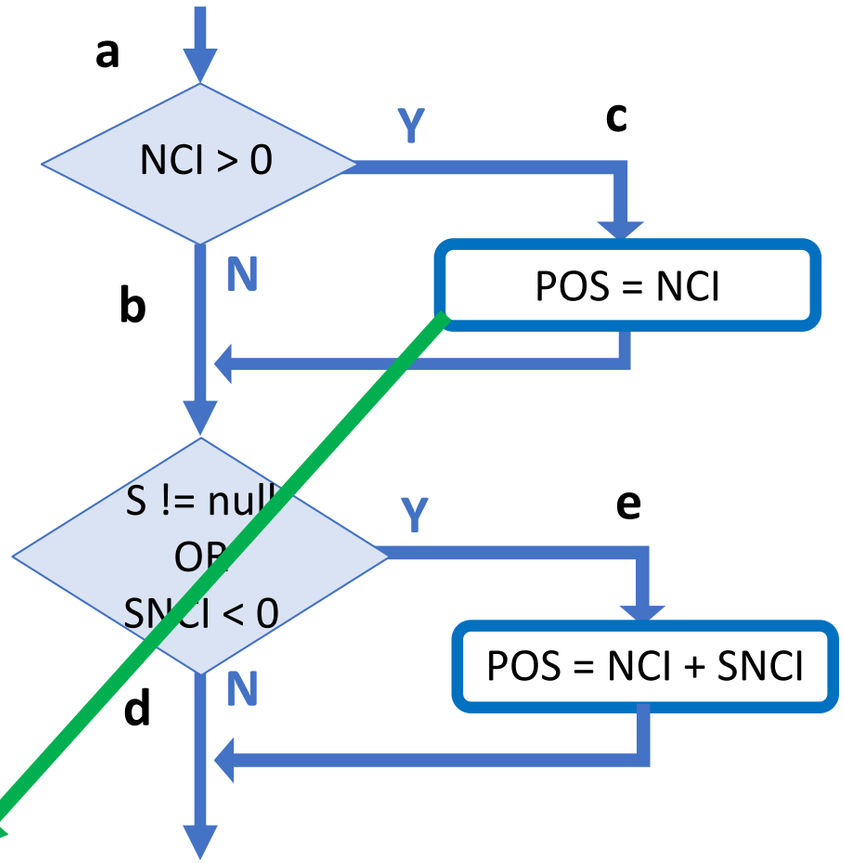

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage



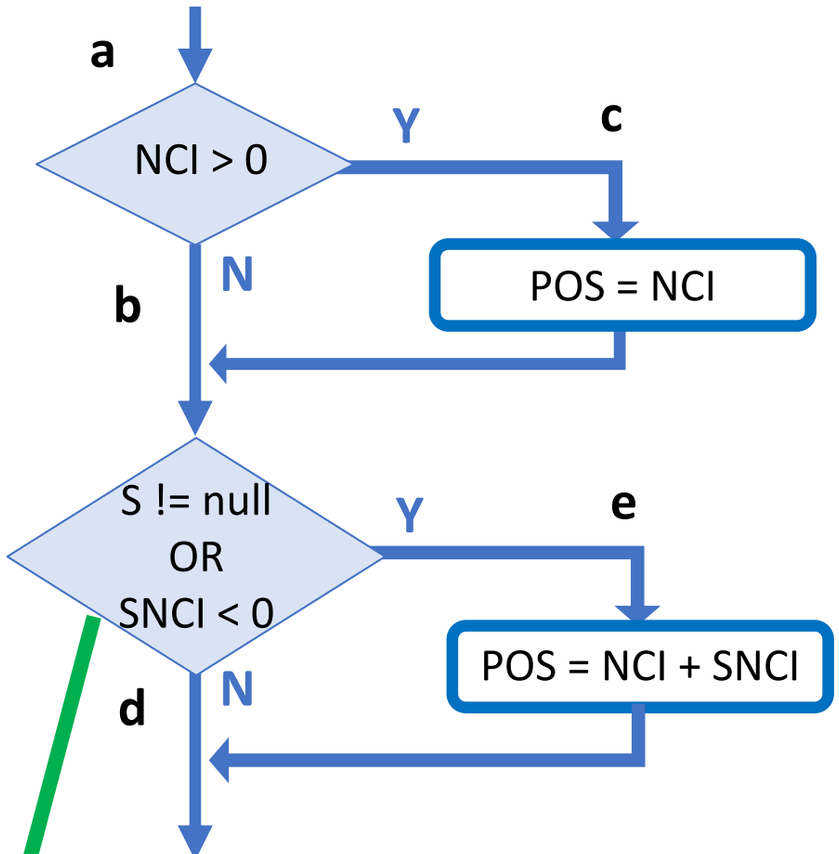

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage



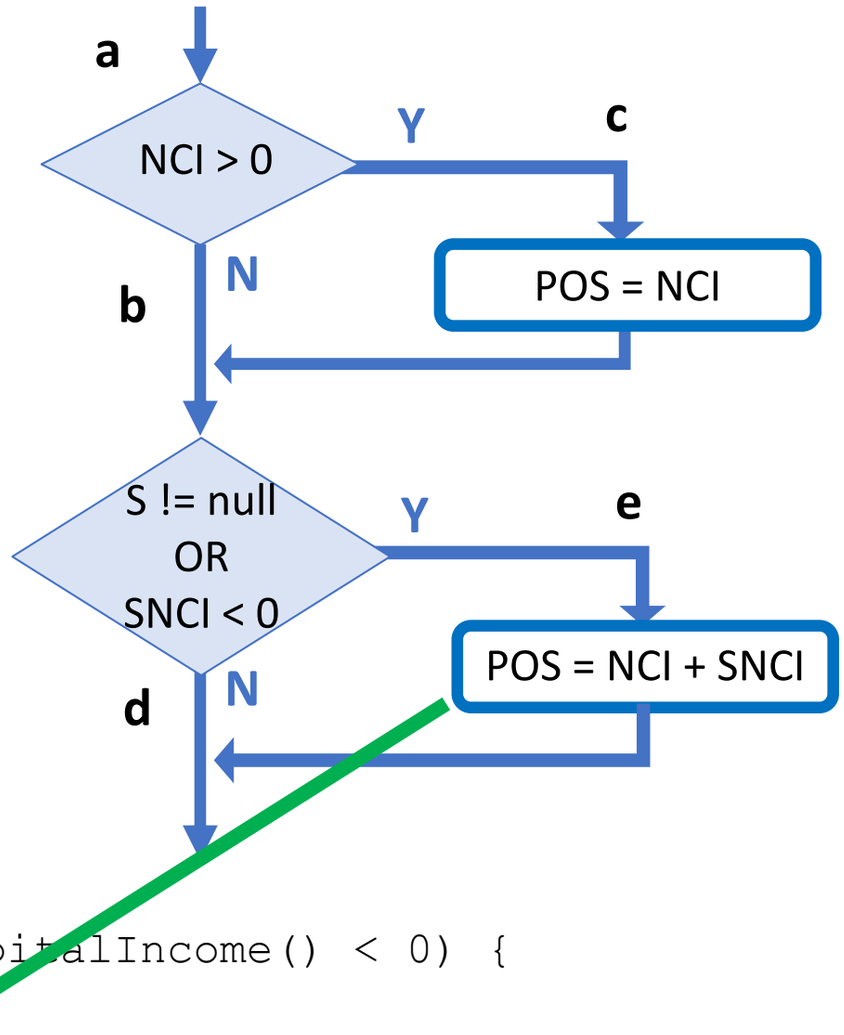

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage



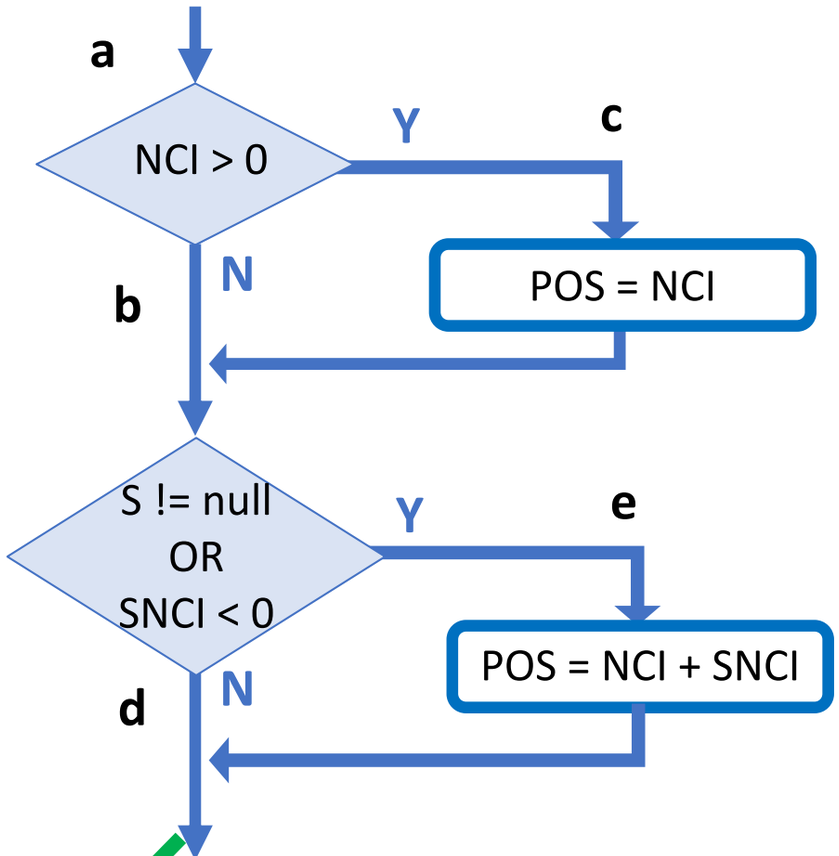

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage




```
calculateBracket(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;
```

Test Coverage



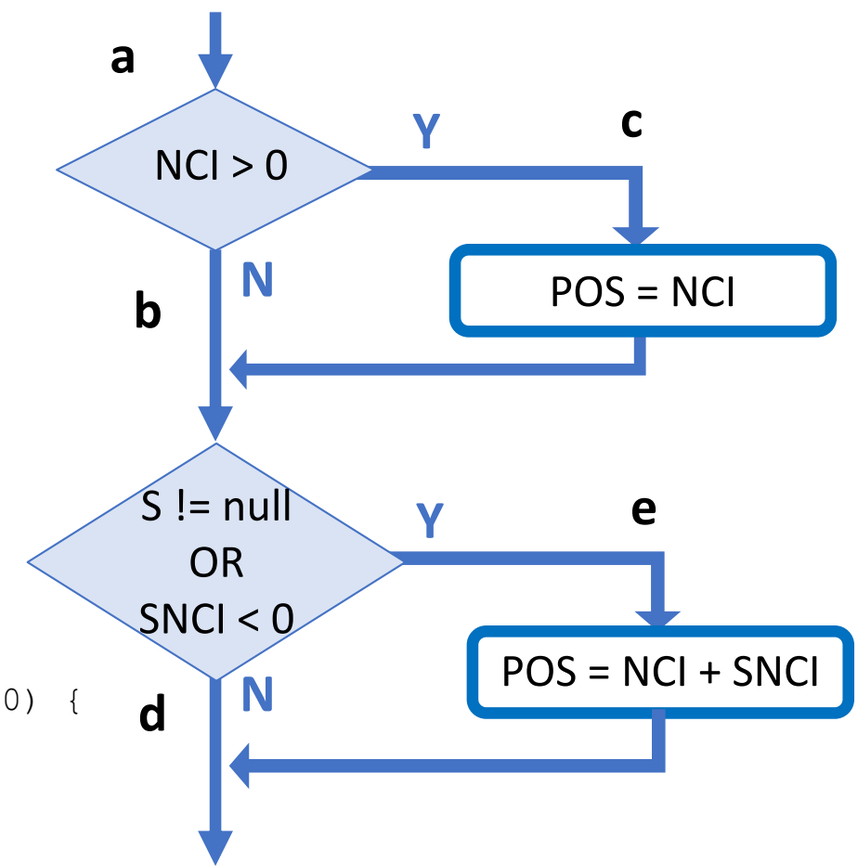

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage

Statement coverage: require every statement of the program to be executed at least once

Test Coverage

Statement coverage: require every statement of the program to be executed at least once

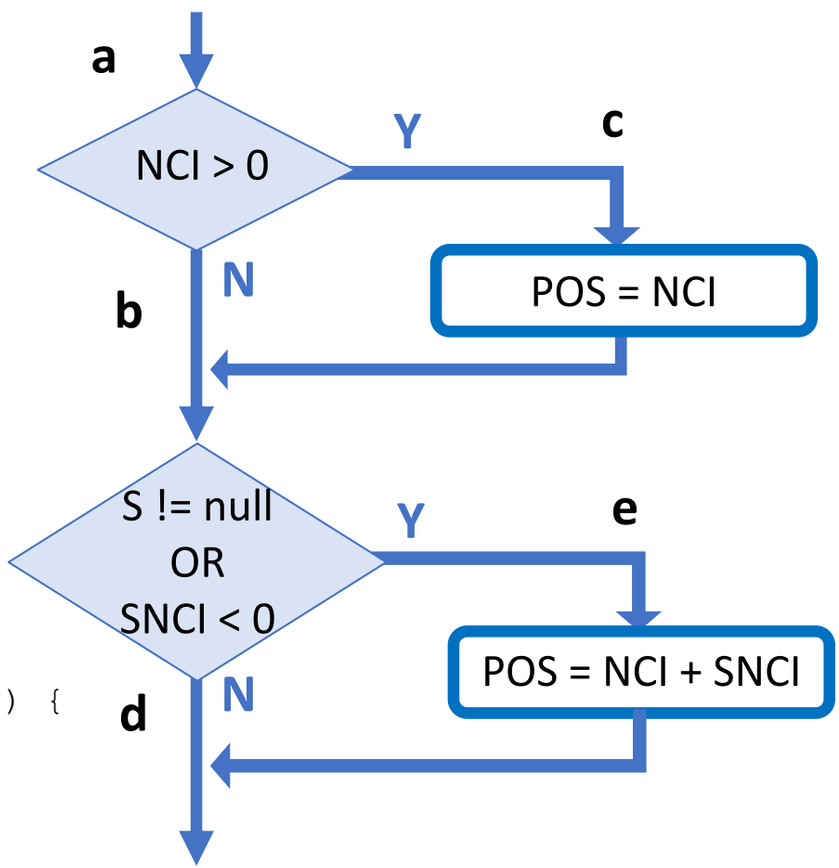



```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage

Statement coverage: require every statement of the program to be executed at least once

Test Case: NCI = 1000, SNCI = -2000



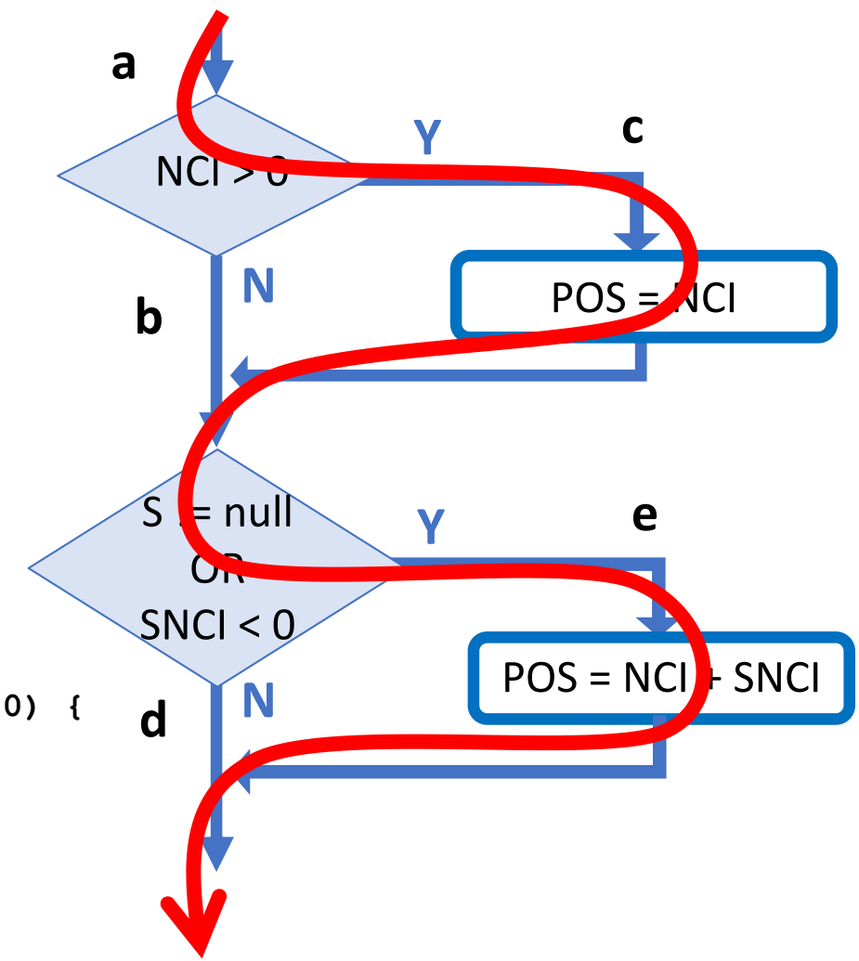

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = t.netCapitalIncome() +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage

Statement coverage: require every statement of the program to be executed at least once

Test Case: NCI = 1000, SNCI = -2000

```
calculateBracket(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;
```



Test Coverage

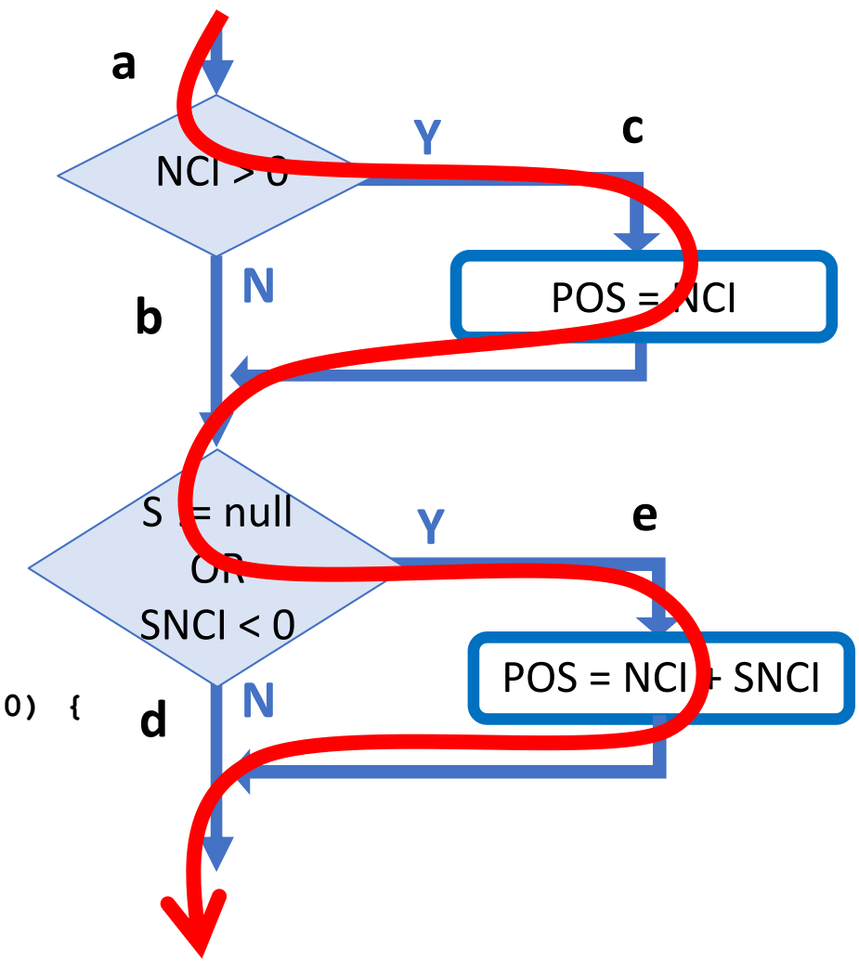
Statement coverage: require every statement of the program to be executed at least once

Test Case: NCI = 1000, SNCI = -2000

Expected taxationBasis 0, result is -1000

→ Defect detected!

```
calculateBracket(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;
```



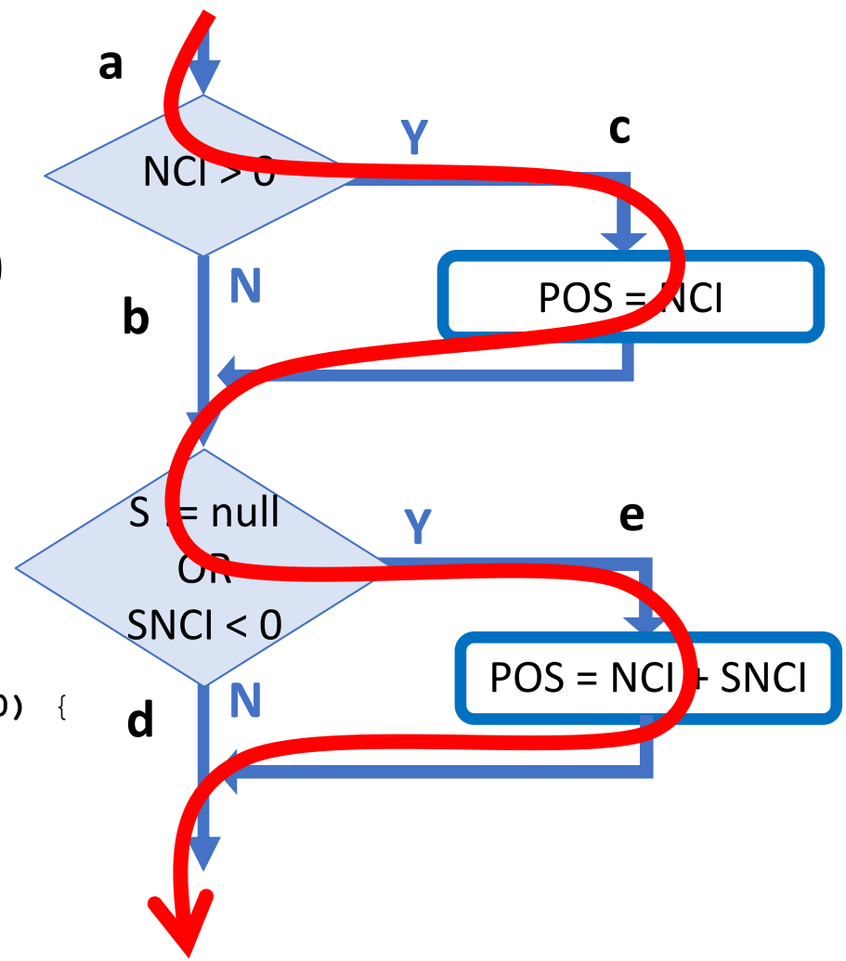
Test Coverage

Statement coverage: require every statement of the program to be executed at least once

A defect remains!

S == null will cause an error in the check SNCI < 0

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

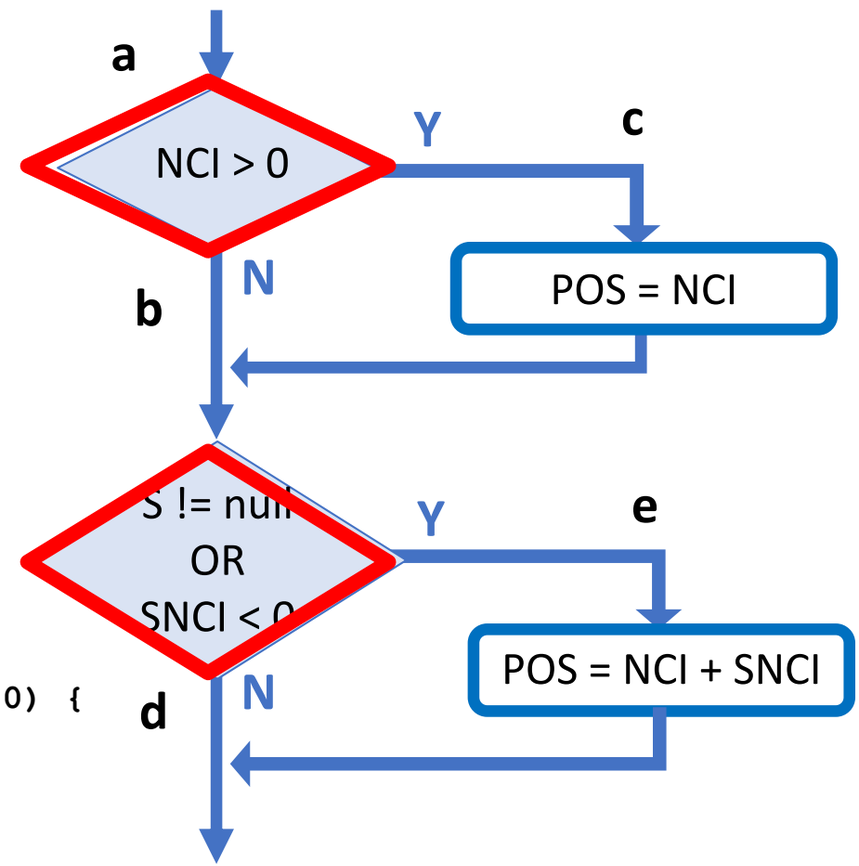


Test Coverage

Decision (branch) coverage: require each condition has a true and false outcome at least once

Test Coverage

Decision (branch) coverage: require each condition has a true and false outcome at least once

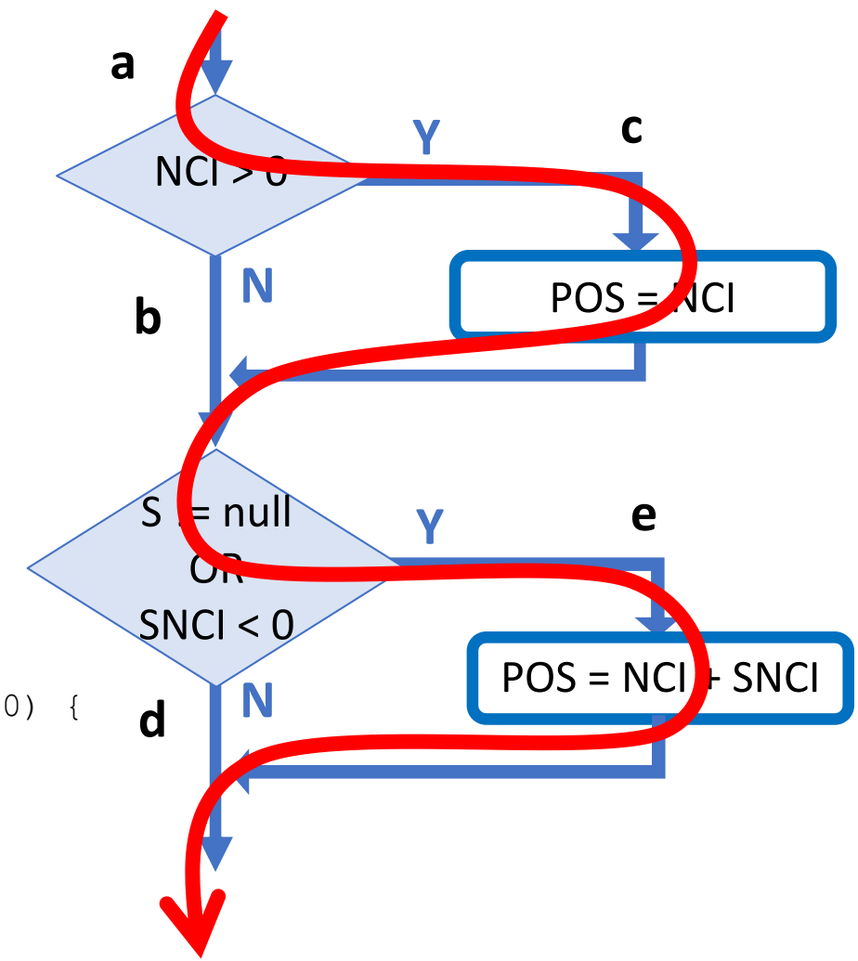



```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

Test Coverage

Decision (branch) coverage: require each condition has a true and false outcome at least once

Test Case 1: NCI = 1000, SNCI = -2000




```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```

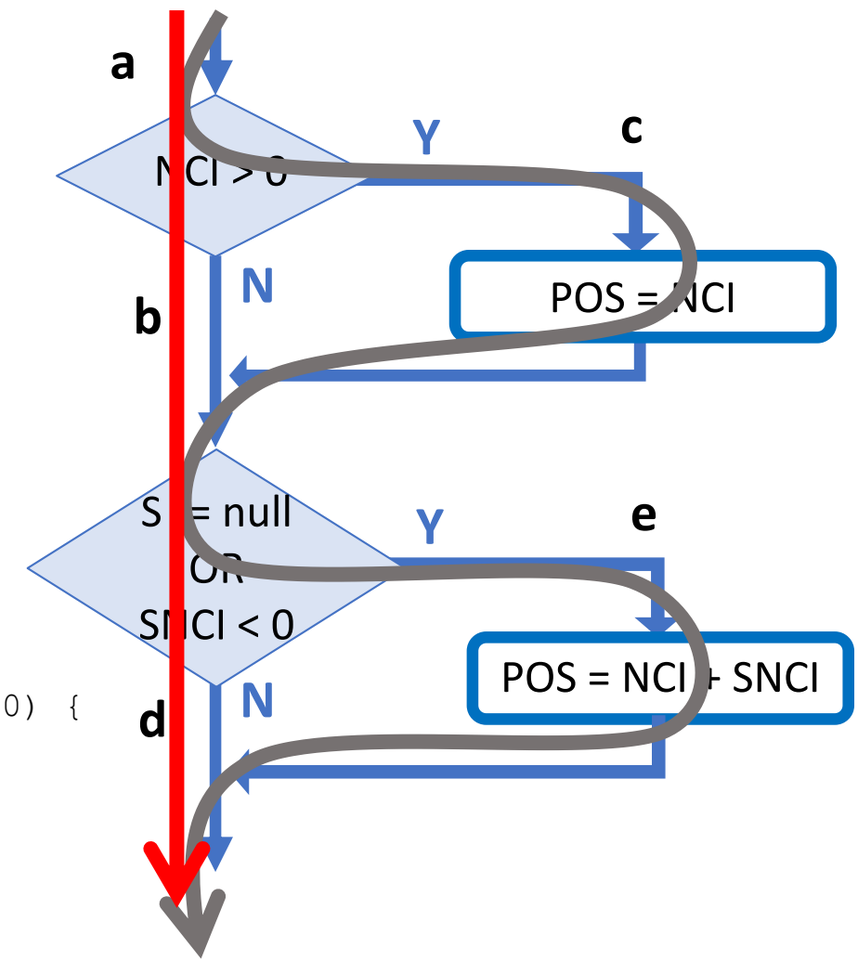
Test Coverage

Decision (branch) coverage: require each condition has a true and false outcome at least once

Test Case 1: NCI = 1000, SNCI = -2000

Test Case 2: NCI = -2000, S == null

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```



Test Coverage

Decision (branch) coverage: require each condition has a true and false outcome at least once

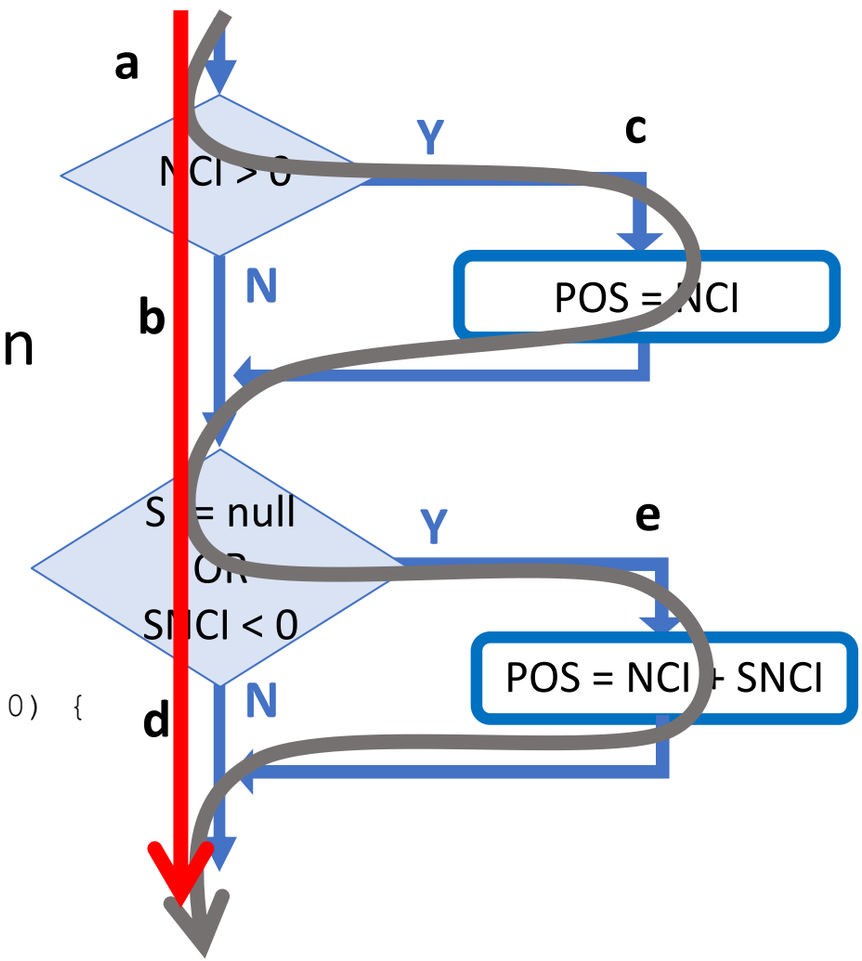
Test Case 1: NCI = 1000, SNCI = -2000

Test Case 2: NCI = -2000, S == null

Expected taxationBasis = 0, result is null exception

→ Defect detected!

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null || t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```



Test Coverage

Decision (branch) coverage: require each condition has a true and false outcome at least once

Test Case 1: NCI = 1000, SNCI = -2000

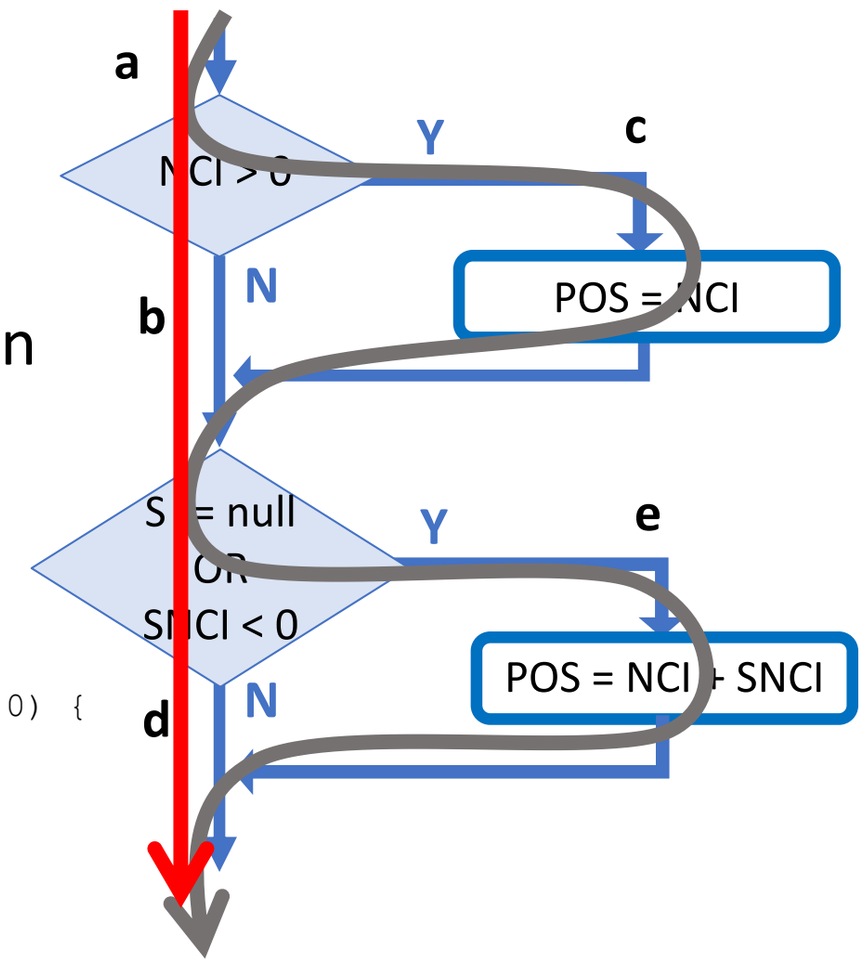
Test Case 2: NCI = -2000, S == null

Expected taxationBasis = 0, result is null exception

→ Defect detected!

Fix!

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null && t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```



Test Coverage

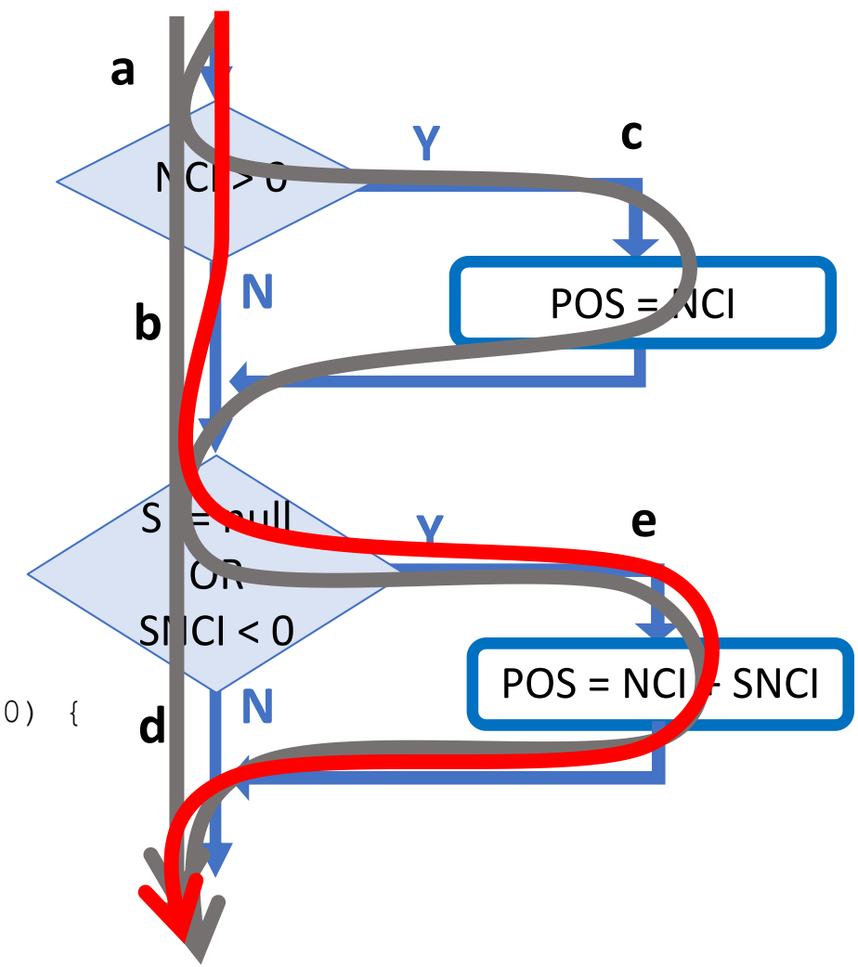
Decision (branch) coverage: require each condition has a true and false outcome at least once

A defect remains!

NCI < 0; SNCI < 0 path is untested

expect taxationBasis 0, result is negative

```
calculateBracket(Taxpayer t) {
    int posNetCapitalIncome = 0;
    if (t.netCapitalIncome() > 0) {
        posNetCapitalIncome = t.netCapitalIncome();
    }
    if (t.getSpouse() != null && t.getSpouse().netCapitalIncome() < 0) {
        posNetCapitalIncome = posNetCapitalIncome +
            t.getSpouse().netCapitalIncome();
    }
    int taxationBasis = t.personalIncome() + posNetCapitalIncome;
```



Test Coverage

- Condition/Decision coverage: test every condition and decision outcome
- Multiple-condition coverage: test all condition combinations in a decision
- **Path coverage: test every possible combination of decisions**
- Others… (exceptions?)

Complex (and sometimes impractical) to compute and make test cases for, not handled by most code coverage tools.

→Become more important for safety-critical systems
A Practical Tutorial on Modified Condition/Decision Coverage (NASA)

Test Coverage

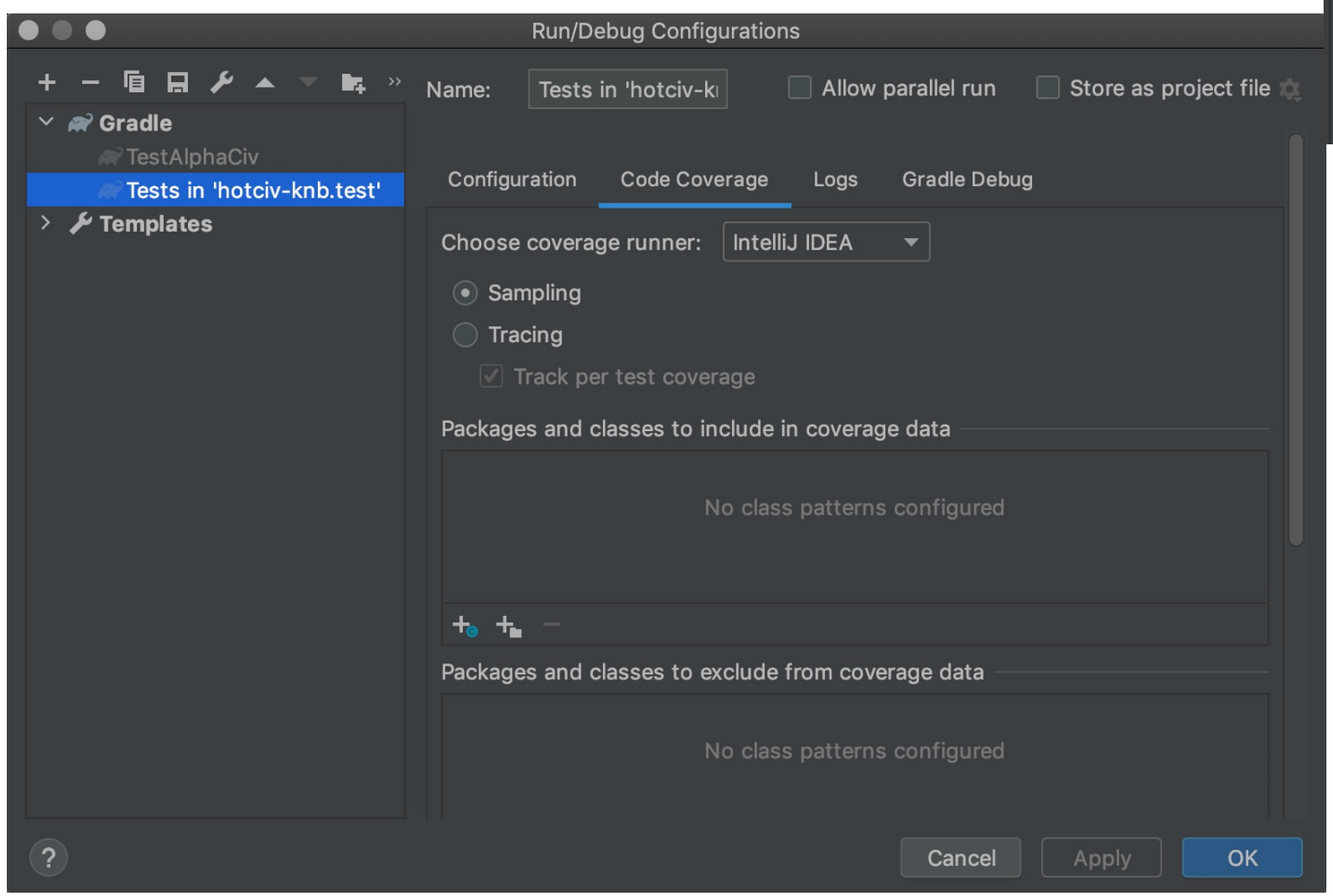
Table 1. Types of Structural Coverage

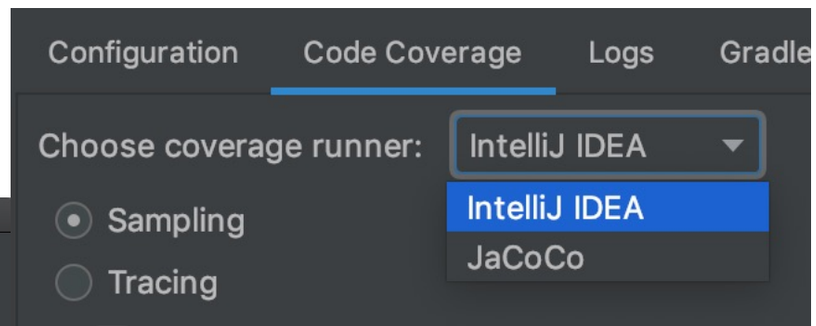
Coverage Criteria	Statement Coverage	Decision Coverage	Condition Coverage	Condition/ Decision Coverage	MC/DC	Multiple Condition Coverage
Every point of entry and exit in the program has been invoked at least once		•	•	•	•	•
Every statement in the program has been invoked at least once	•					
Every decision in the program has taken all possible outcomes at least once		•		•	•	•
Every condition in a decision in the program has taken all possible outcomes at least once			•	•	•	•
Every condition in a decision has been shown to independently affect that decision's outcome					•	•[8]
Every combination of condition outcomes within a decision has been invoked at least once						•

A Practical Tutorial on Modified Condition/Decision Coverage (NASA)

IntelliJ Code Coverage

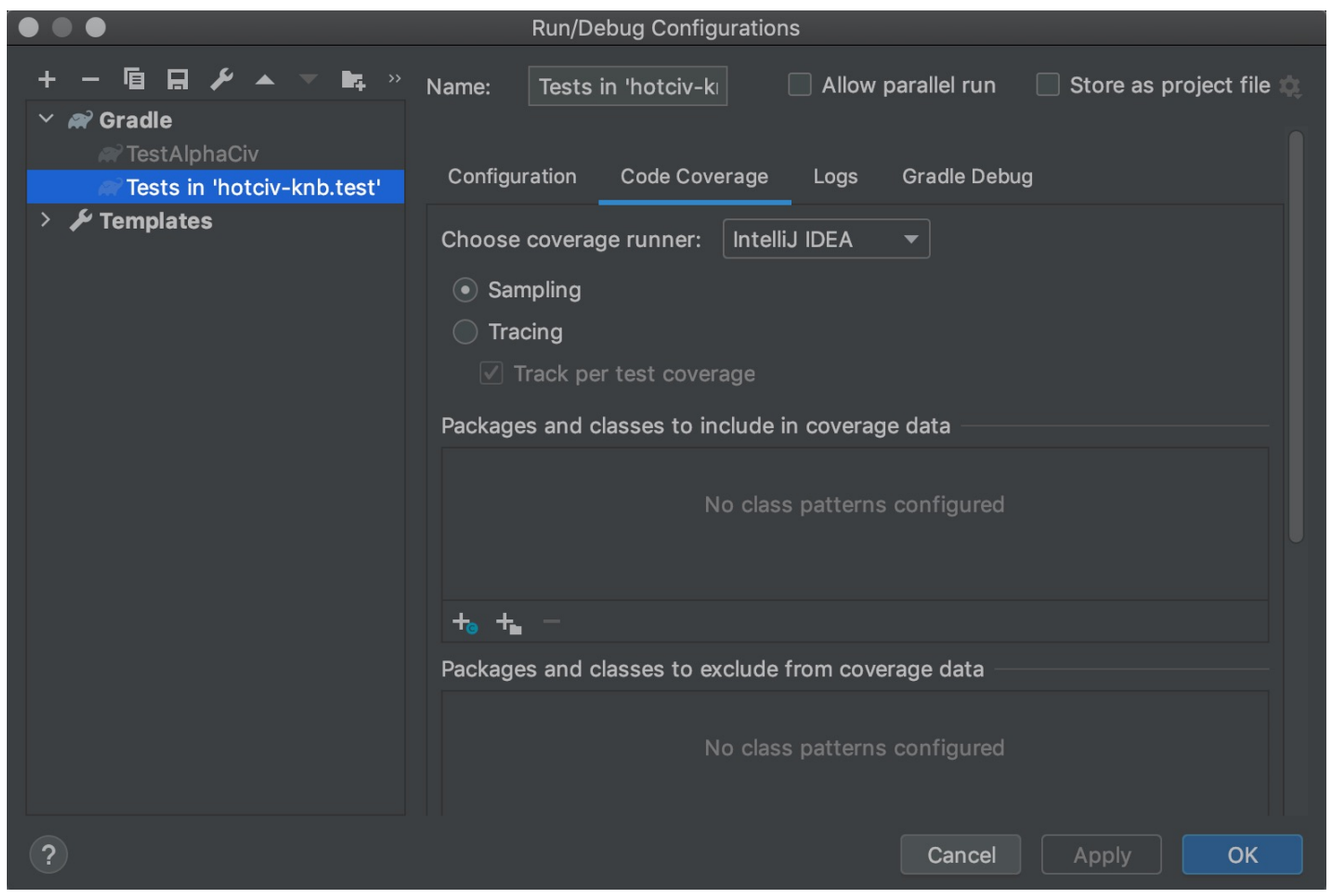
Run > Edit Configurations





IntelliJ Code Coverage

Run > Edit Configurations

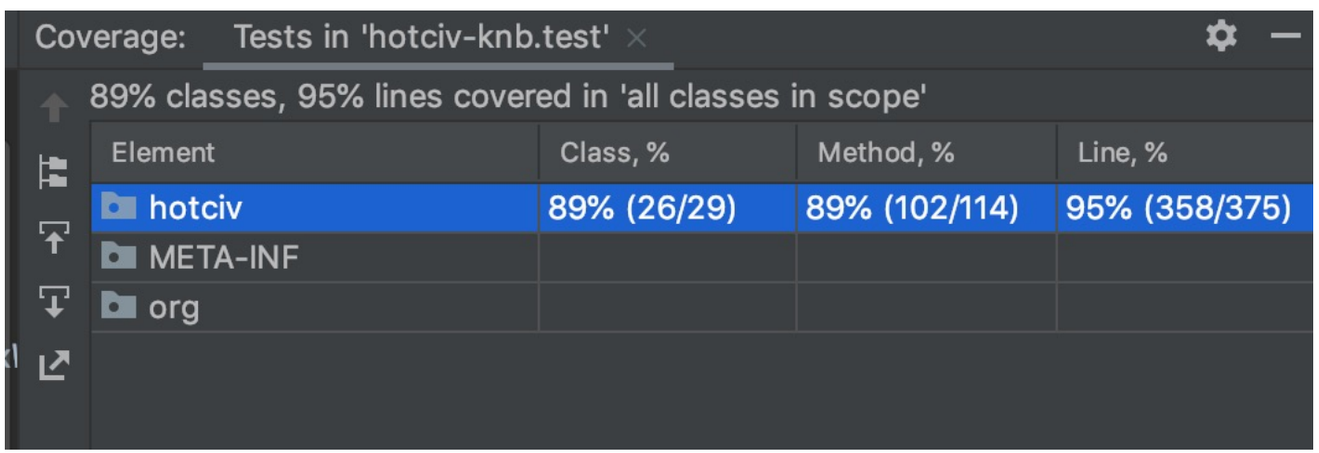


Sampling: Line coverage

Tracing: Branch coverage

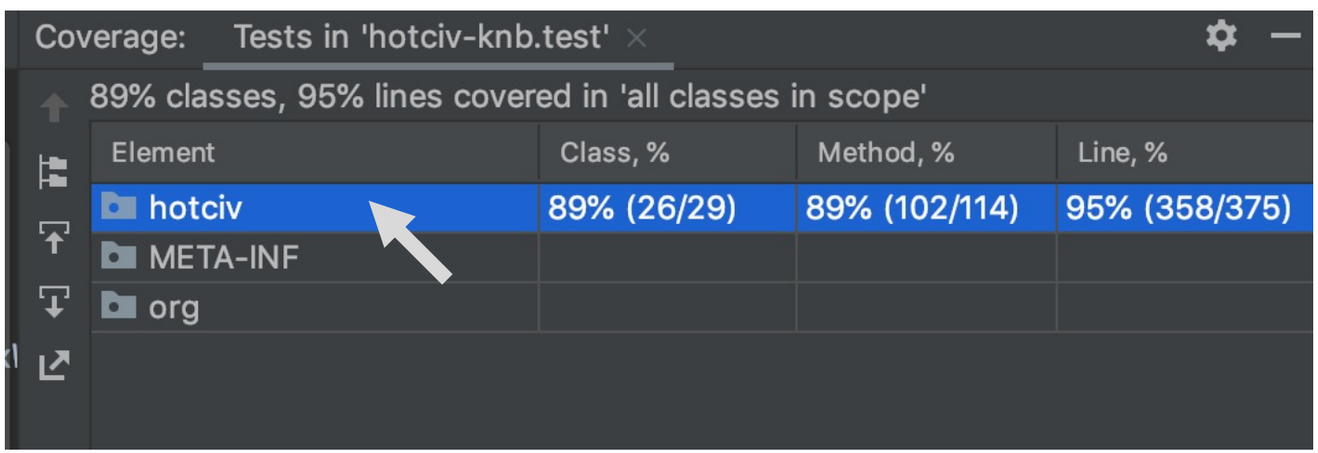
IntelliJ Code Coverage

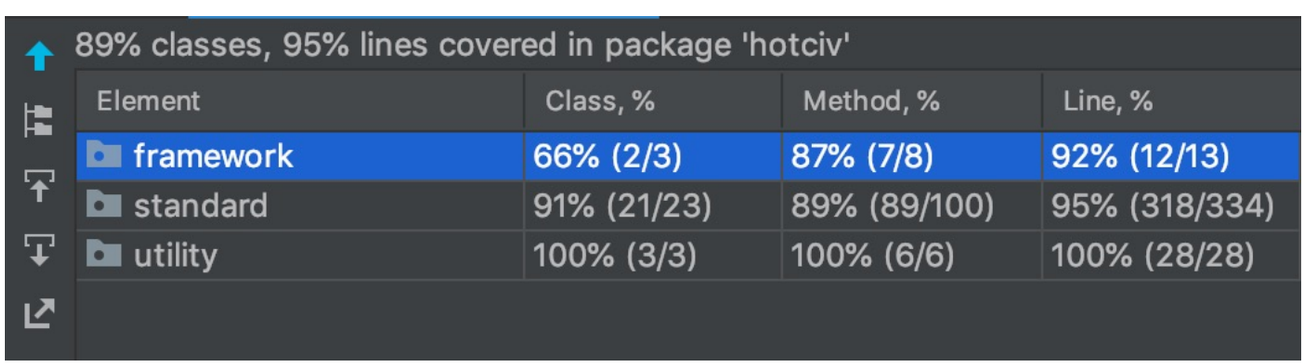
Run > Run … With Coverage



IntelliJ Code Coverage

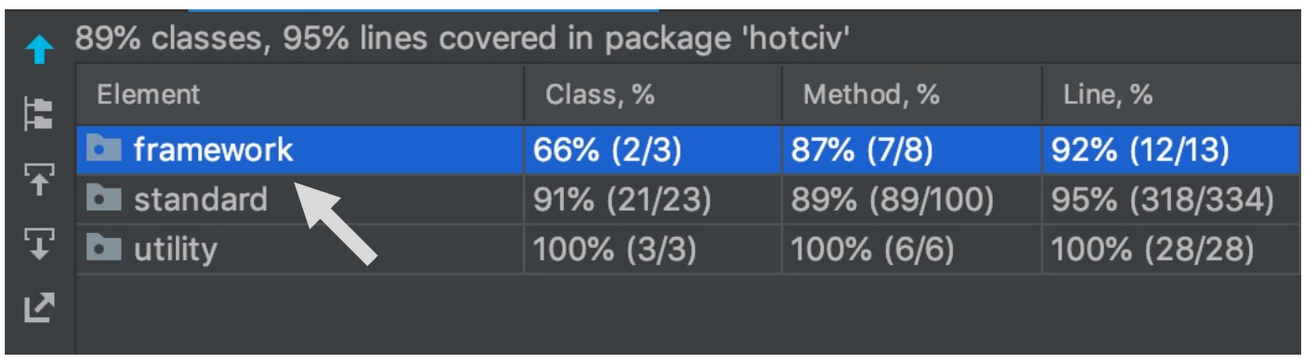
Run > Run … With Coverage

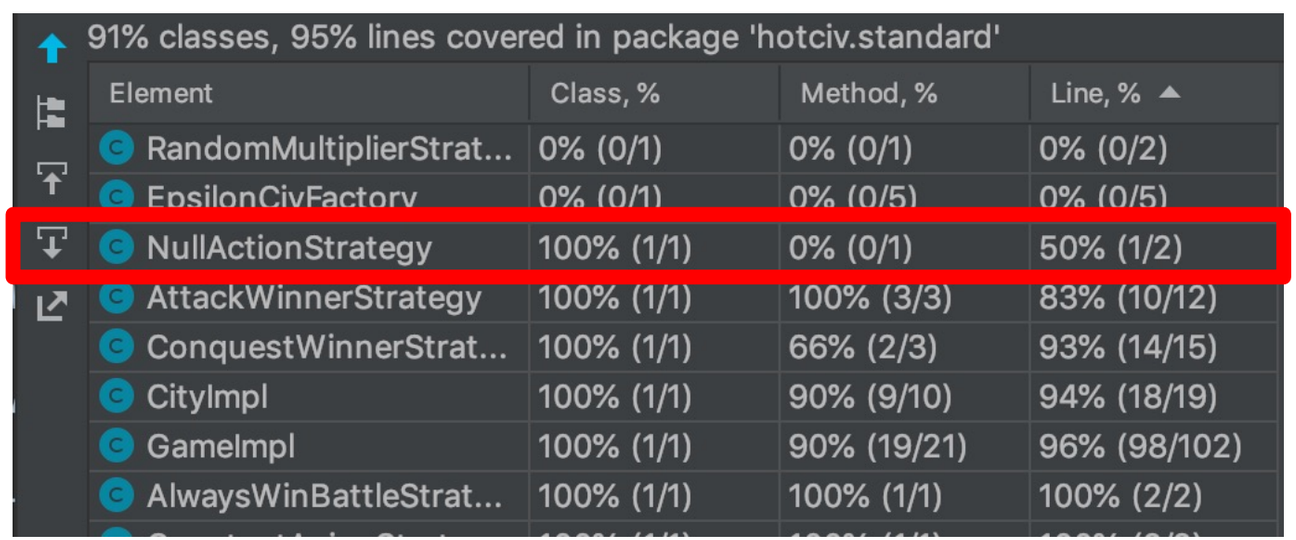




IntelliJ Code Coverage

Run > Run … With Coverage





IntelliJ Code Coverage

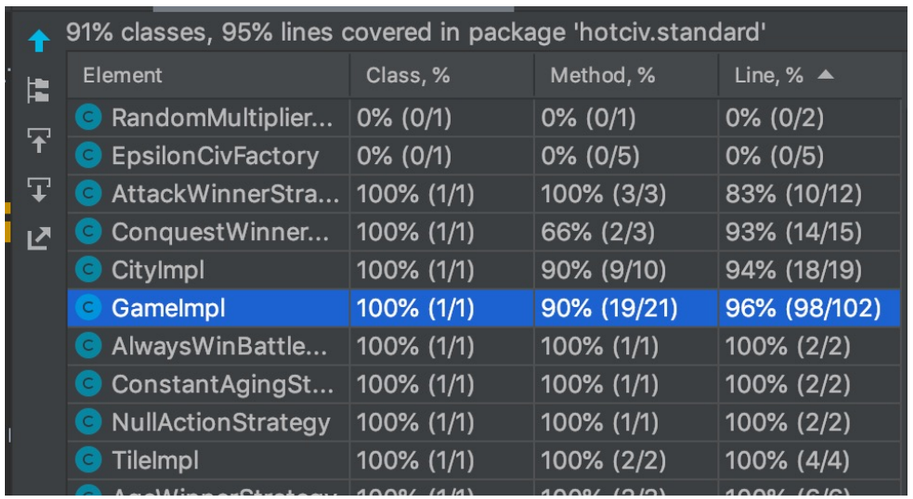
TestAlphaCiv

```
@Test
public void shouldBeNullArcherAction() {
  Position to = new Position( r: 3, c: 0);
  game.performUnitActionAt(redArcherStart);
  game.moveUnit(redArcherStart, to);
  assertThat(game.getUnitAt(to).getTypeString(), is(GameConstants.ARCHER));
}
```

91% classes, 95% lines covered in package 'hotciv.standard'

Element	Class, %	Method, %	Line, %
RandomMultiplie...	0% (0/1)	0% (0/1)	0% (0/2)
EpsilonCivFactory	0% (0/1)	0% (0/5)	0% (0/5)
AttackWinnerStr...	100% (1/1)	100% (3/3)	83% (10/12)
ConquestWinne...	100% (1/1)	66% (2/3)	93% (14/15)
CityImpl	100% (1/1)	90% (9/10)	94% (18/19)
GameImpl	100% (1/1)	90% (19/21)	96% (98/10...
AlwaysWinBattle...	100% (1/1)	100% (1/1)	100% (2/2)
ConstantAgingS...	100% (1/1)	100% (1/1)	100% (2/2)
NullActionStrate...	100% (1/1)	100% (1/1)	100% (2/2)
TileImpl	100% (1/1)	100% (2/2)	100% (4/4)
AgeWinnerStrat...	100% (1/1)	100% (3/3)	100% (6/6)

IntelliJ Code Coverage

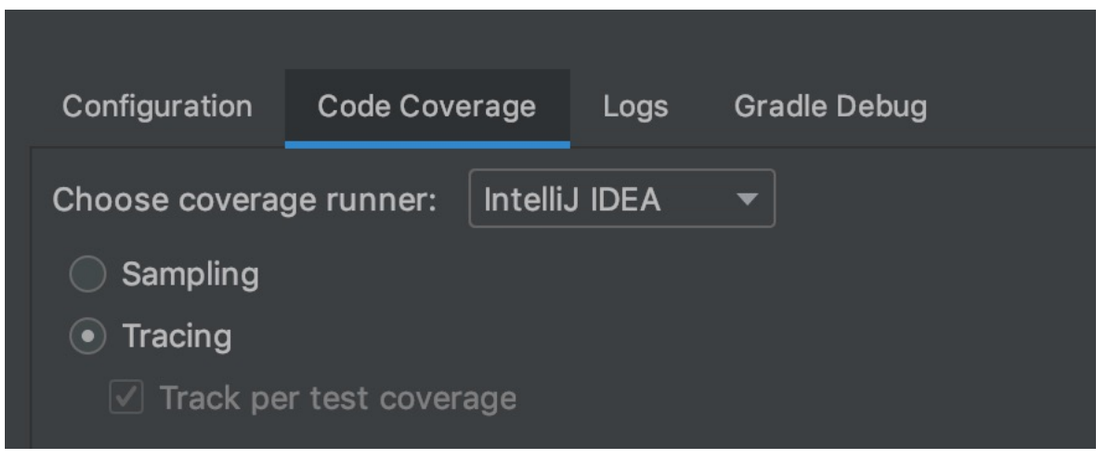


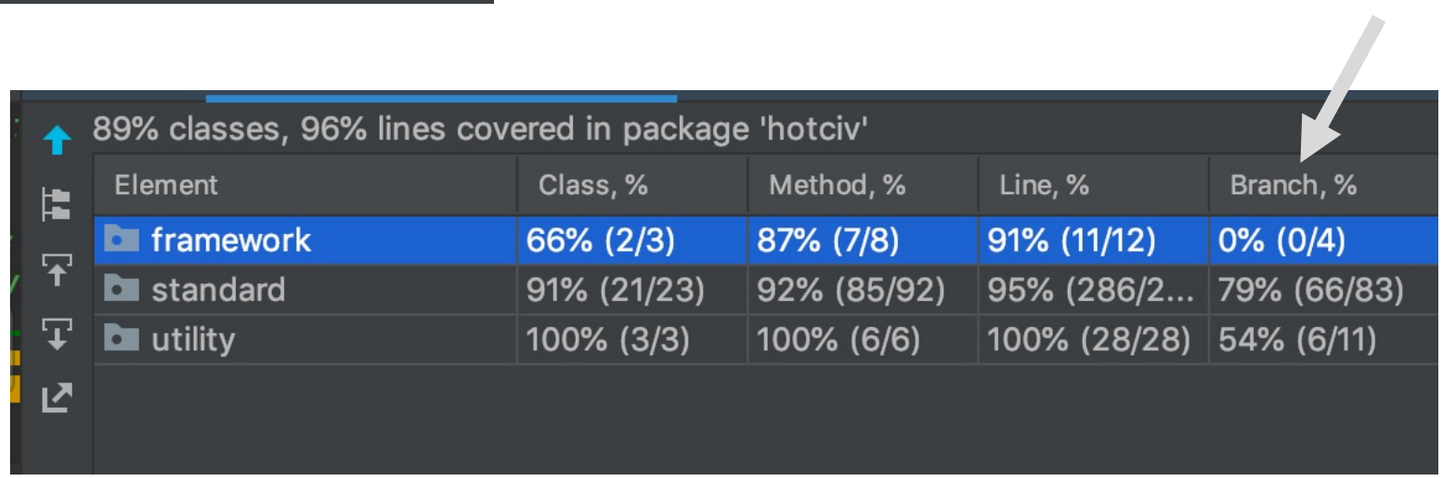
GameImpl

```
202     @   private int getUnitCost(String producedUnit) {
203             if (producedUnit.equals(GameConstants.ARCHER)) {
204                 return GameConstants.ARCHER_COST;
205             } else if (producedUnit.equals(GameConstants.LEGION)) {
206                 return GameConstants.LEGION_COST;
207             } else if (producedUnit.equals(GameConstants.SETTLER)) {
208                 return GameConstants.SETTLER_COST;
209             } else {
210                 return GameConstants.ARCHER_COST;
```

```
226
227     public void changeWorkForceFocusInCityAt(Position p, String balance) {
228     }
229
230     public void changeProductionInCityAt(Position p, String unitType) {
231     }
```

IntelliJ Code Coverage





IntelliJ Code Coverage

VaryingAgingStrategy

```
5   public class VaryingAgingStrategy implements AgingStrategy {
6       public int advanceAge(int age) {
7           if (age >= -4000 && age < -100)  {
8               return age + 100;
9           } else if (age == -100) {
10              return -1;
11          } else if (age == -1) {
12              return 1;
13          } else if (age == 1) {
14              return 50;
15          } else if (age >= 50 && age < 1750) {
16              return age + 50;
17          } else if (age >= 1750 && age < 1900) {
18              return age + 25;
19          } else if (age >= 1900 && age < 1970) {
20              return age + 5;
21          } else {
22              return age + 1;
```

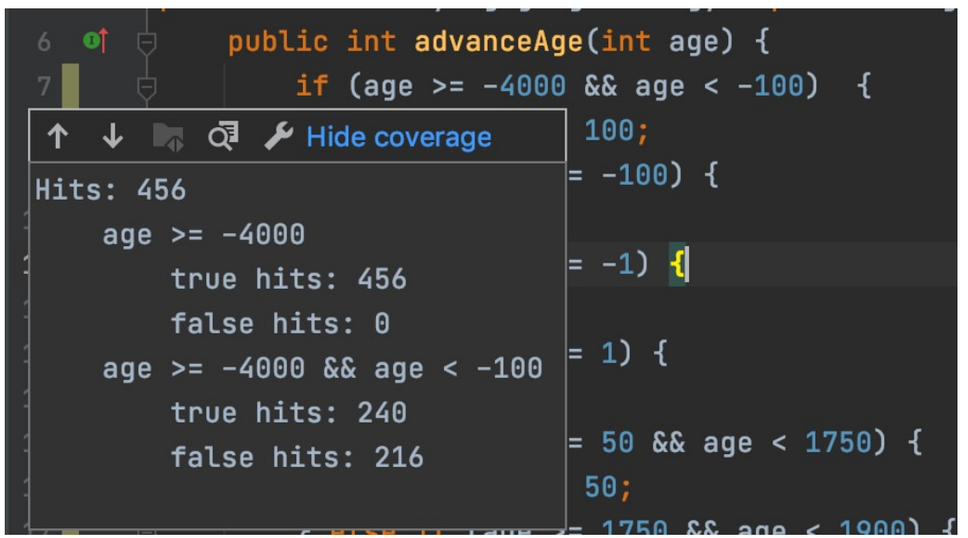
91% classes, 95% lines covered in package 'hotciv.standard'

Element	Class, %	Method, %	Line, %	Branc...
AttackWinnerSt...	100% (1...	100% (2...	80% (8/...	0% (0/2)
VaryingAgingStr...	100% (1...	100% (1...	100% (1...	63% (7/...
ActiveActionStr...	100% (1...	100% (1...	100% (1...	66% (2/3)
ConquestWinne...	100% (1...	100% (1...	100% (1...	66% (2/3)
UnitImpl	100% (1...	100% (1...	100% (2...	75% (3/4)
GameImpl	100% (1...	100% (1...	97% (89...	77% (17...
MinimalWorldLa...	100% (1...	100% (1...	100% (2...	88% (16...

IntelliJ Code Coverage

VaryingAgingStrategy

```
5   public class VaryingAgingStrategy implements AgingStrategy {
6       public int advanceAge(int age) {
7           if (age >= -4000 && age < -100)  {
8               return age + 100;
9           } else if (age == -100) {
10              return -1;
11          } else if (age == -1) {
12              return 1;
13          } else if (age == 1) {
14              return 50;
15          } else if (age >= 50 && age < 1750) {
16              return age + 50;
17          } else if (age >= 1750 && age < 1900) {
18              return age + 25;
19          } else if (age >= 1900 && age < 1970) {
20              return age + 5;
21          } else {
22              return age + 1;
```



Code Coverage: Summary

Decision coverage usually satisfies statement coverage.

100% line/branch coverage is not necessary, but low coverage is worrisome.

100% line/branch coverage does not guarantee no defects!

→ Consider use cases, common configurations to prioritize paths

Next time: More Patterns
(Composite, Observer, MVC, Template Method)