Design Patterns

- Automated Testing
- JUnit
- Test-Driven Development
- Test Coverage
- Integration Tests

Test Coverage

- Automated testing raises an interesting problem:
  - How do we know we've tested everything

- How many tests do we actually need?
  - Tests take time to execute
  - There is no benefit to redundant tests
  - We might be missing an important test case

Test Coverage

Function coverage: has every function/method been called at least once?

Statement coverage: has every statement been executed once?

Branch coverage: has every branch been executed?

Condition coverage: has every Boolean expression been evaluated as both true and false.

Which one of these is the least/most difficult to achieve?

Function Coverage

Every function called at least once.

This is the absolute bare minimum level of coverage.

Even still, it is surprisingly difficult, especially for imperative code. [why?]

- It’s hard to test methods without meaningful returns.

Statement Coverage

The most common “bare-minimum” level of testing observed in real software.

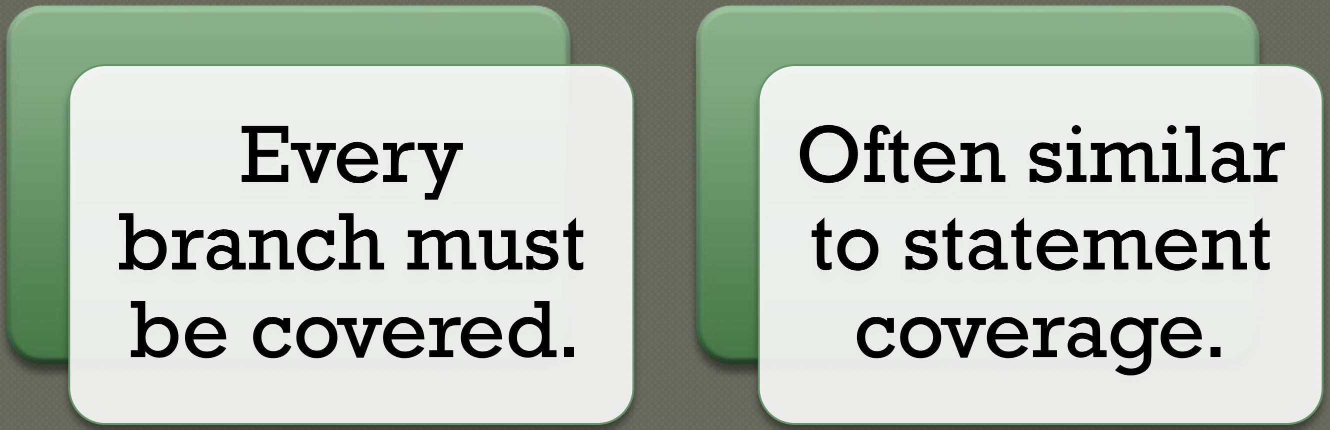
Ensure that each statement is executed at least once.

This seems like “perfect” coverage, but isn’t. [why?]

Statement Coverage: How to Calculate

- (#tested statements) / (#statements)
- That’s it.
- Shoot for 100%.
- Note: we care about the innermost statements
  - the ones inside the body of `if`s and loops

Branch Coverage



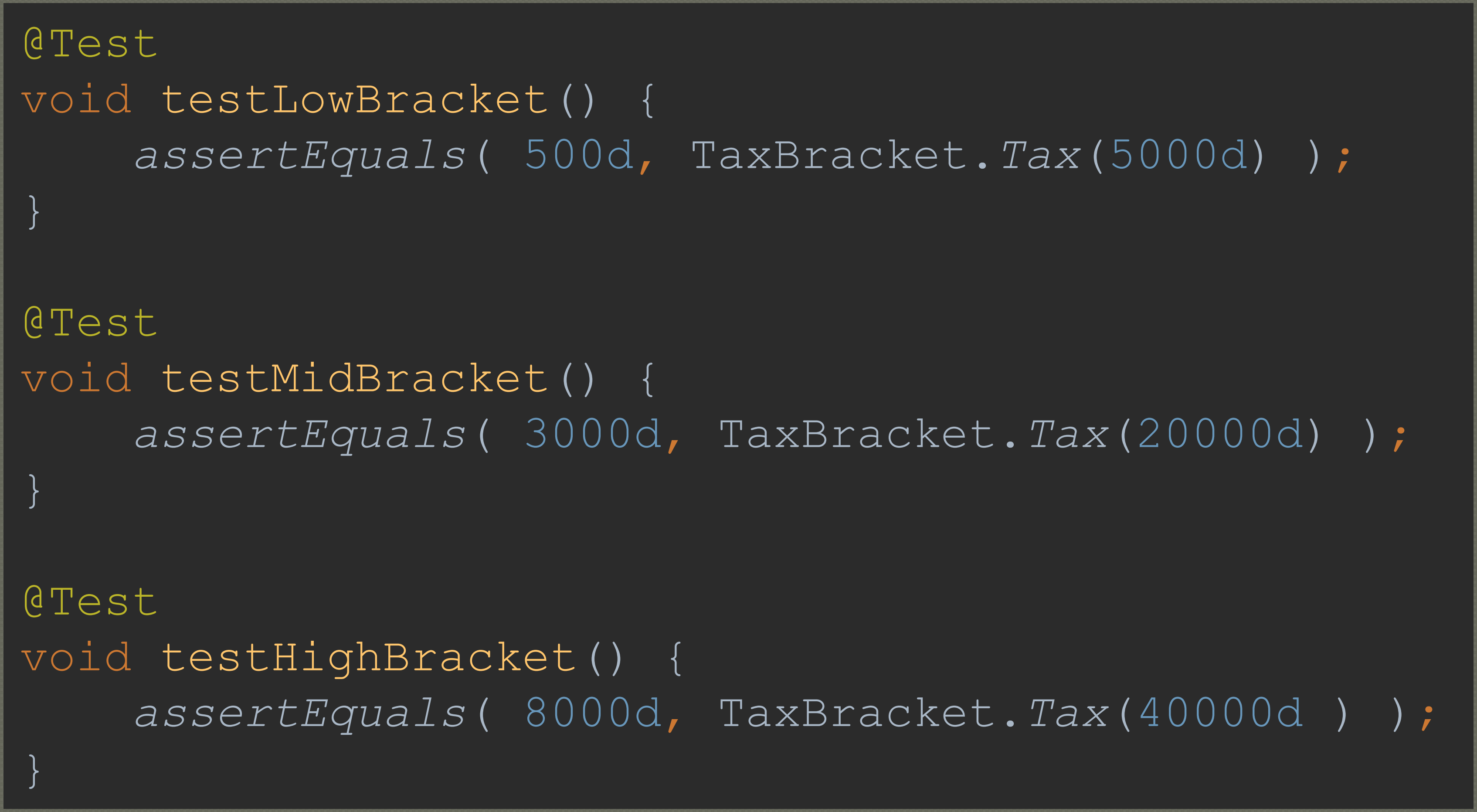
Every branch must be covered.

Often similar to statement coverage.

Example: Calculating Income Tax

```
static double Tax( double income ) {
    if ( income <= 10000d )
        return .1 * income;
    else if ( income <= 30000d )
        return .1 * 10000d + .2 * ( income - 10000d );
    else
        return .1 * 10000d +
                .2 * 20000d +
                .3 * (income - 30000d );
}
```

Total Statement/ Branch Coverage

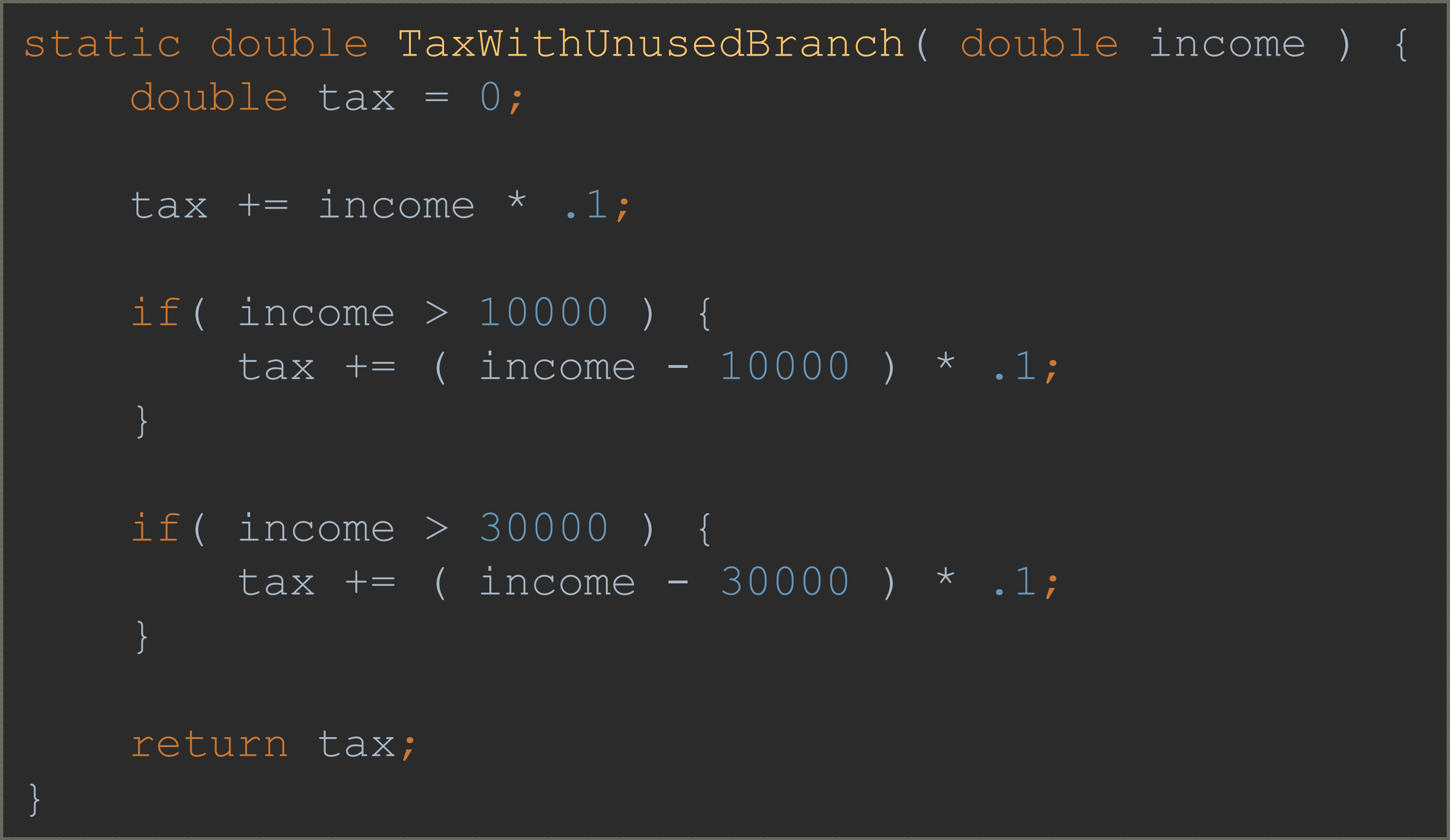


```
@Test
void testLowBracket() {
    assertEquals( 500d, TaxBracket.Tax(5000d) );
}

@Test
void testMidBracket() {
    assertEquals( 3000d, TaxBracket.Tax(20000d) );
}

@Test
void testHighBracket() {
    assertEquals( 8000d, TaxBracket.Tax(40000d ) );
}
```

How is Statement Coverage Different?

- At first glance statement coverage seems to be the same
- But there’s a subtle difference…
- What happens when we have an `if`-statement that doesn’t execute?

Example



```
static double TaxWithUnusedBranch( double income ) {
    double tax = 0;

    tax += income * .1;

    if( income > 10000 ) {
        tax += ( income - 10000 ) * .1;
    }

    if( income > 30000 ) {
        tax += ( income - 30000 ) * .1;
    }

    return tax;
}
```

Important Notes

Technically, we only need one test (40,000 dollars) for complete statement coverage.

But, we wouldn’t have tested all the empty branches (“invisible `else`s”).

We need to take these into account to fully test our method.

Branch Coverage Calculation

- (#tested branches) / (#branches)
- Easy.

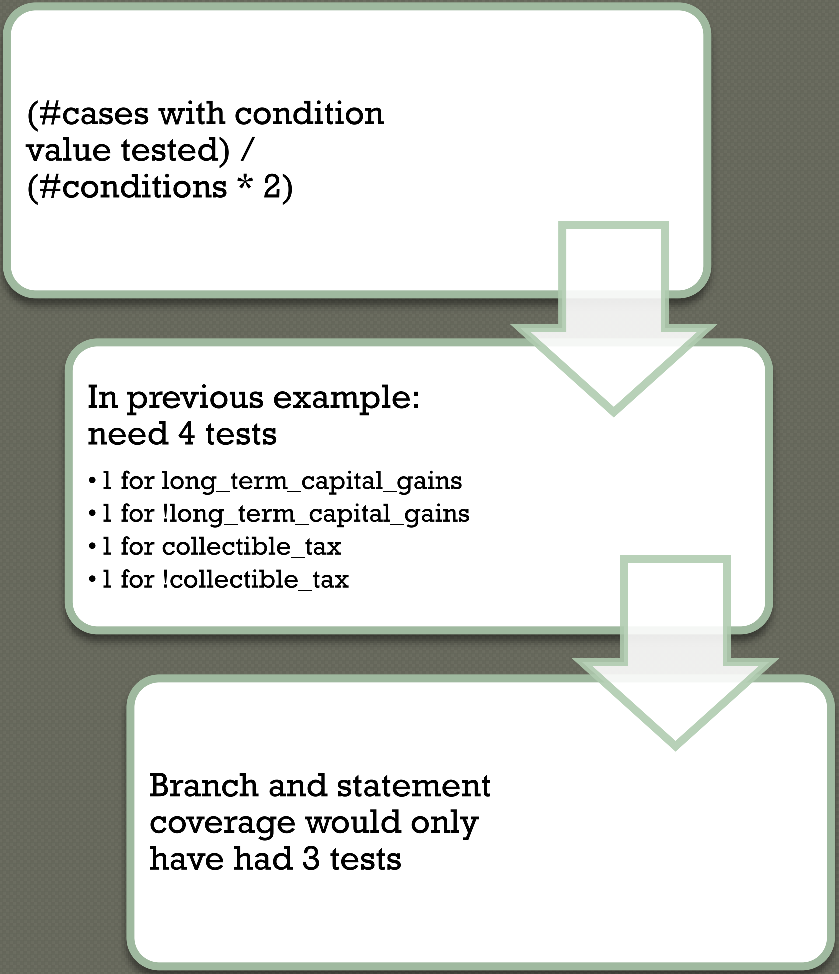
Condition Coverage

- What if there is more than one way that an `if`-statement can be executed?
- Especially important if one of these ways has a consequence in a different method (side effects)

Condition Coverage Example

```
static double MaxRate(
        boolean saleOfAssets, boolean heldForLong, boolean collectible ) {

    boolean long_term_capital_gains =
            saleOfAssets && heldForLong && !collectible;
    boolean collectible_tax =
            heldForLong && collectible;

    if( long_term_capital_gains )
        return 20.0;
    else if( collectible_tax )
        return 35.0;

    return 39.6;
}
```

Condition Coverage Metrics



(#cases with condition value tested) / (#conditions * 2)

In previous example: need 4 tests

- 1 for long_term_capital_gains
- 1 for !long_term_capital_gains
- 1 for collectible_tax
- 1 for !collectible_tax

Branch and statement coverage would only have had 3 tests

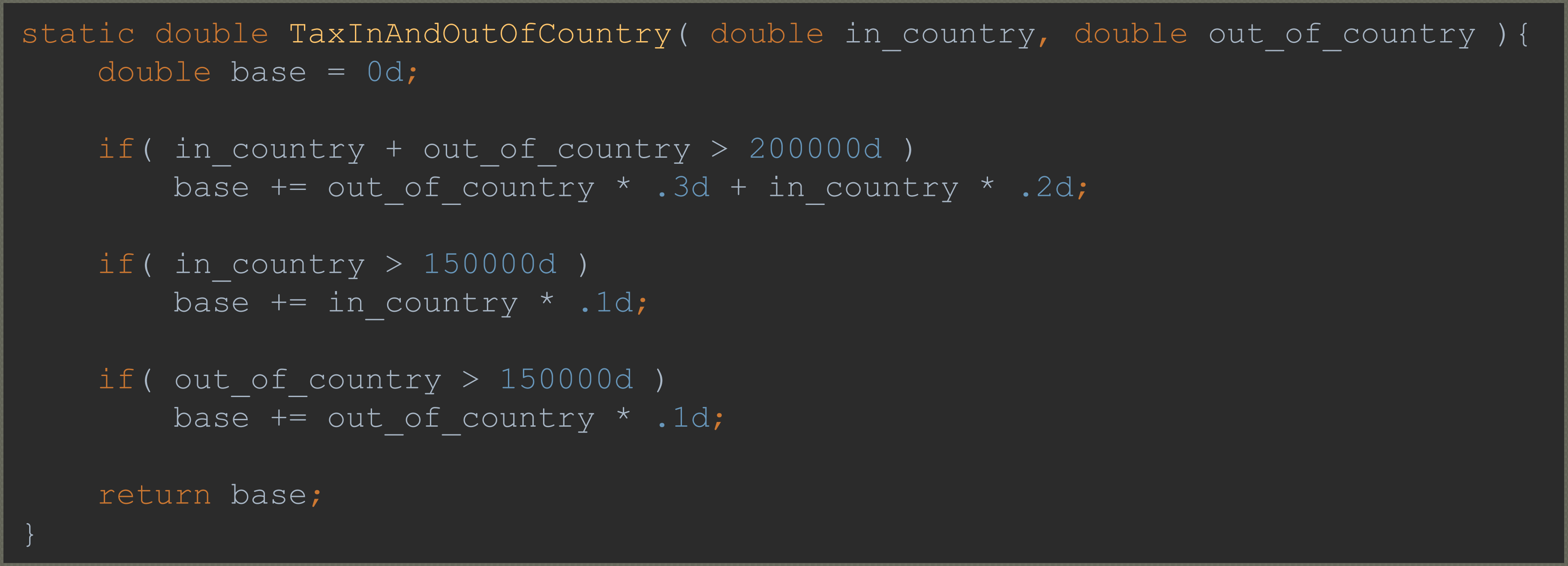
Decision Coverage

Each combination of branches is tested.

So if we have 3 consecutive if-statements (with single conditions), we need 2^3 tests at most

- Compound conditions require additional tests

Example



```
static double TaxInAndOutOfCountry( double in_country, double out_of_country ){
    double base = 0d;

    if( in_country + out_of_country > 200000d )
        base += out_of_country * .3d + in_country * .2d;

    if( in_country > 150000d )
        base += in_country * .1d;

    if( out_of_country > 150000d )
        base += out_of_country * .1d;

    return base;
}
```

Need to Test the Following

1. In country + out of country > 200k but both < 150k
2. In country + out of country > 200k but both > 150k
3. In country + out of country > 200k but in country > 150k, out of country < 150k
4. In country + out of country > 200k but out of country > 150k, in country < 150k
5. In country + out of country < 200k but in country > 150k
6. In country + out of country < 200k but out of country > 150k
7. In country + out of country < 200k and both < 150k
8. One combination is mathematically impossible