CSE 403

Software Engineering

Coverage-based testing and mutation testing

Code coverage: example

Average of the absolute values of an array of doubles

```java
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

https://github.com/rjust/testing-ci-gradle

Line coverage

Classes in this File	Line Coverage	Branch Coverage	Complexity
Avg	100% 10/10	100% 8/8	6

```
 1     package avg;
 2
 3  4  public class Avg {
 4
 5         /*
 6          * Compute the average of the absolute values of an array of doubles
 7          */
 8         public double avgAbs(double ... numbers) {
 9             // We expect the array to be non-null and non-empty
10  4          if (numbers == null || numbers.length == 0) {
11  2              throw new IllegalArgumentException("Array numbers must not be null or empty!");
12             }
13
14  2          double sum = 0;
15  8          for (int i=0; i<numbers.length; ++i) {
16  6              double d = numbers[i];
17  6              if (d < 0) {
18  2                  sum -= d;
19                 } else {
20  4                  sum += d;
21                 }
22             }
23  2          return sum/numbers.length;
24         }
25     }
```

(Cobertura's Code coverage report.)

https://github.com/rjust/testing-ci-gradle

What is the point of code coverage metrics?

Examples of metrics:

- Statement coverage
- Branch coverage
- Path coverage
- Mutant coverage
- …

Example code coverage metrics

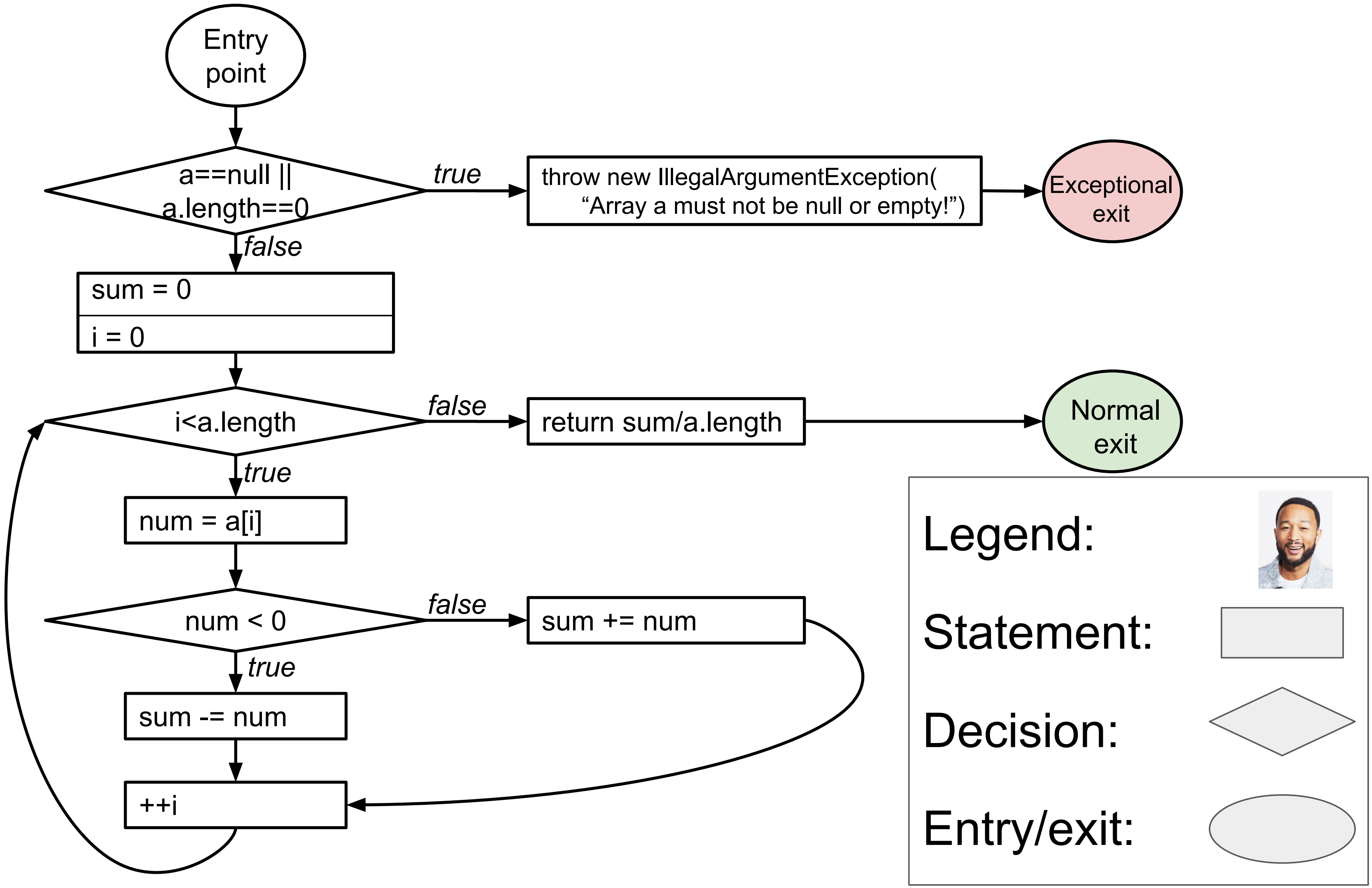
Beyond line coverage

Average of the absolute values of an array of doubles

```
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

Another representation: the control flow graph (CFG)

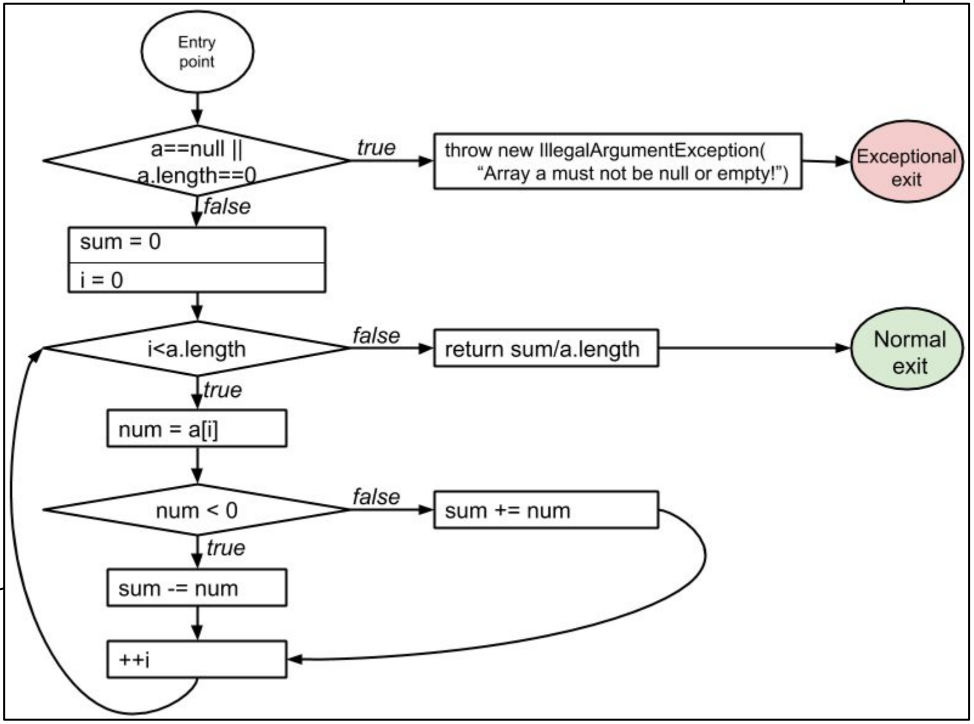
Control flow graph (CFG)



Structural code coverage: representations

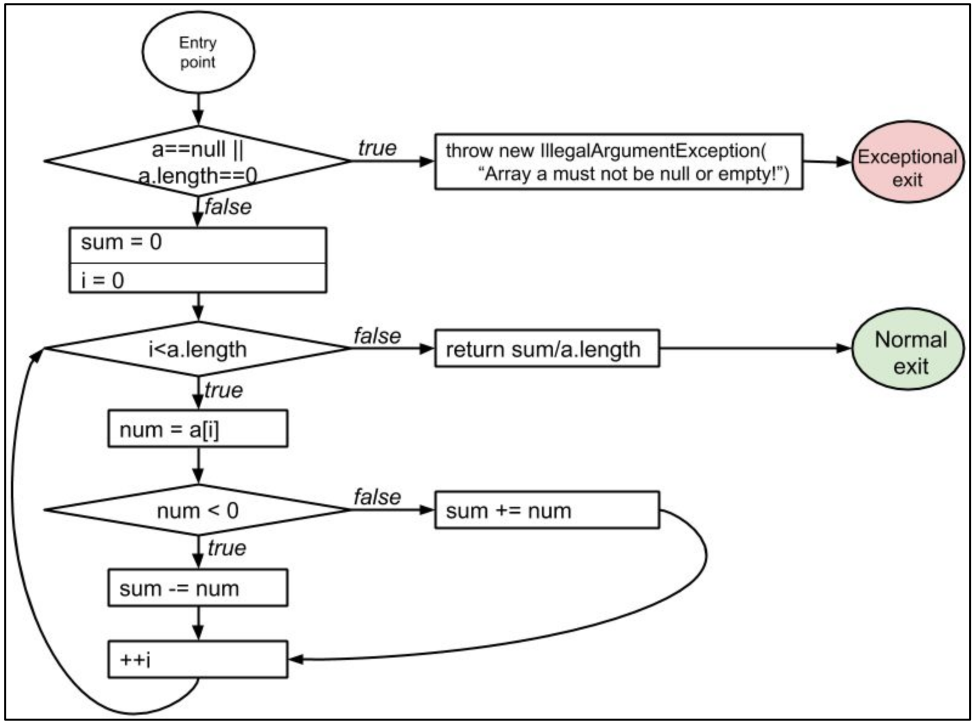
Average of the absolute values of an array of doubles

```
public double avgAbs(double ... numbers) {

  // We expect the array to be non-null and non-empty
  if (numbers == null || numbers.length == 0) {
    throw new IllegalArgumentException("Array numbers must not be null or empty!");
  }

  double sum = 0;
  for (int i=0; i<numbers.length; ++i) {
    double d = numbers[i];
    if (d < 0) {
      sum -= d;
    } else {
      sum += d;
    }
  }

  return sum/numbers.length;
}
```

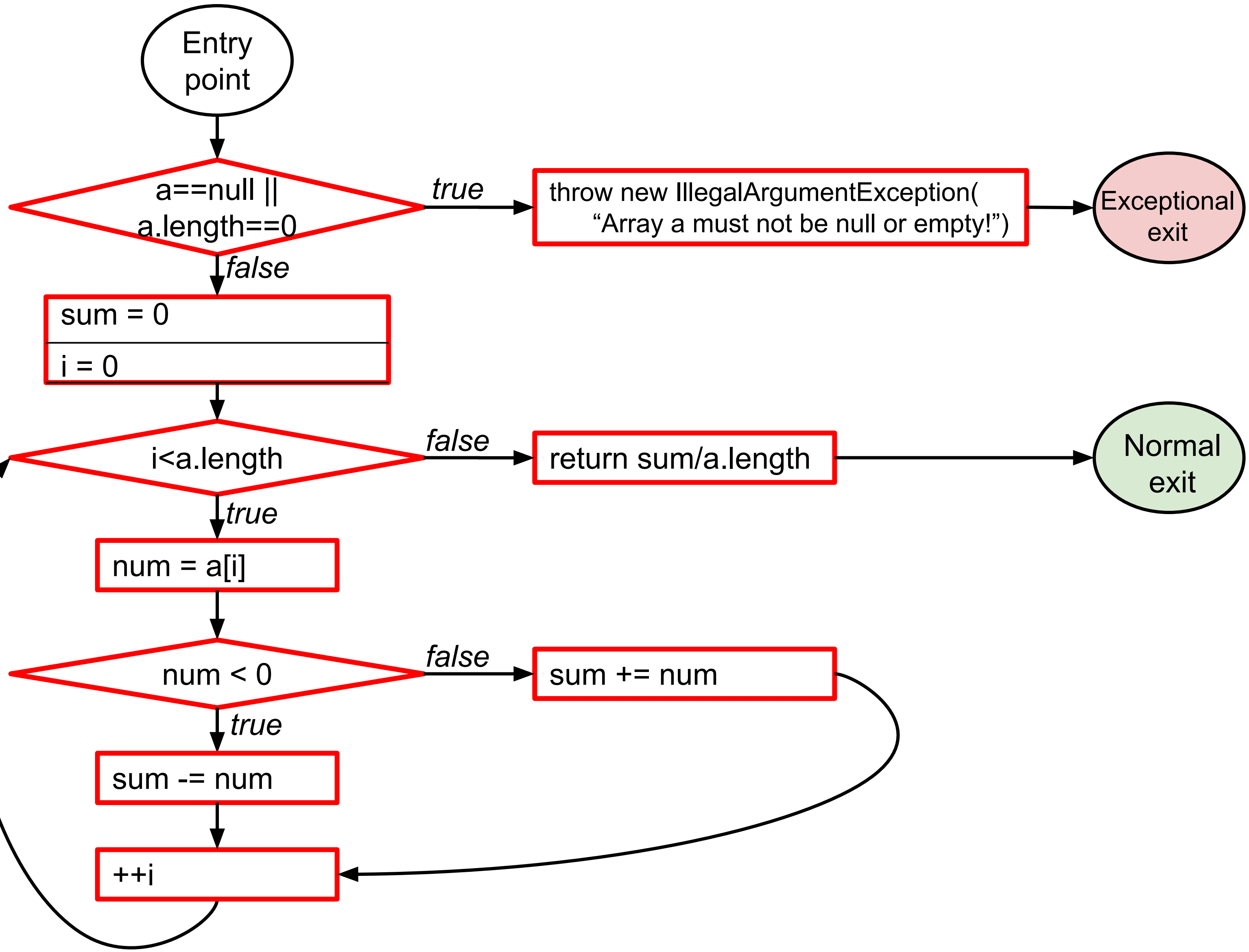


Statement coverage

- **Every** **statement** must be **executed at least once.**

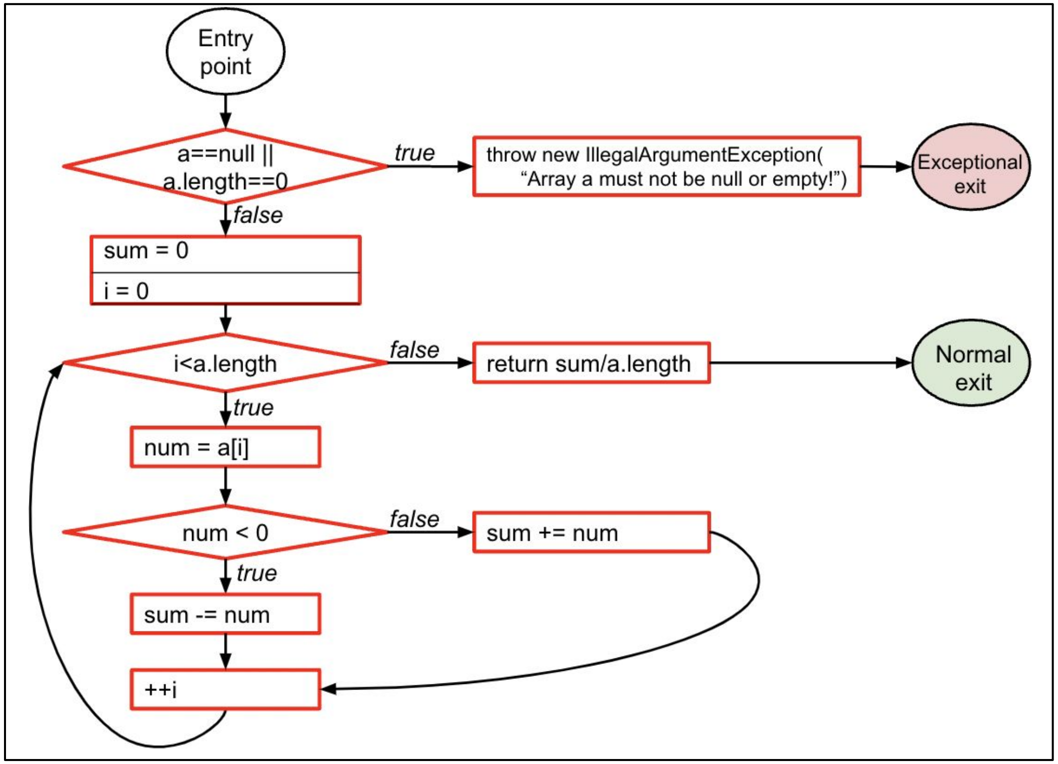


Statement coverage



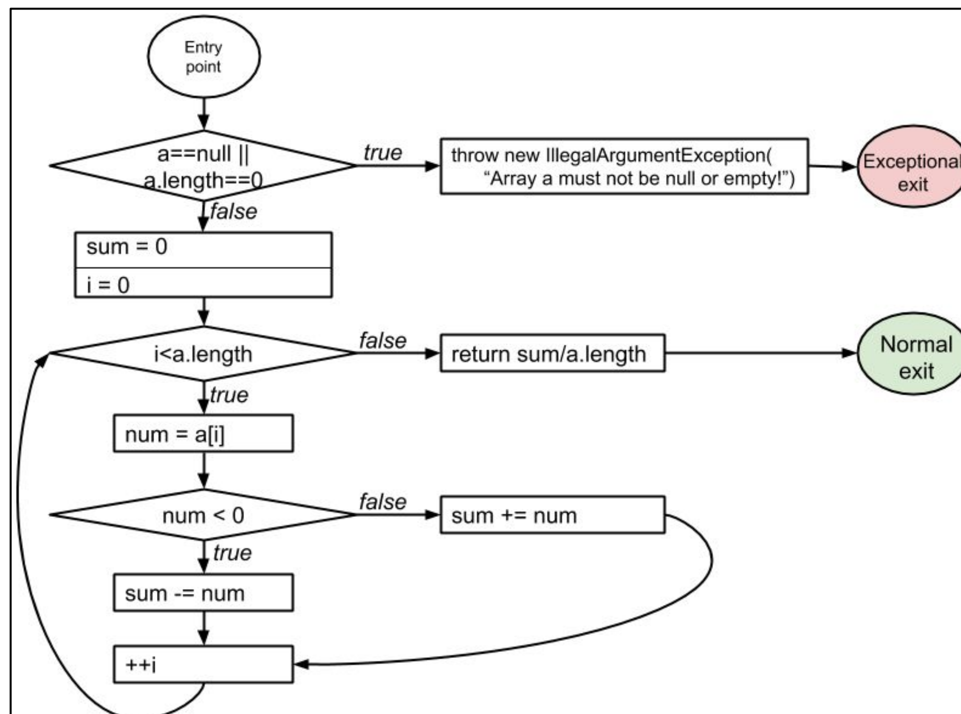
Statement coverage

- **Every statement** must be **executed at least once.**
- = **node coverage** in the control-flow graph (CFG)

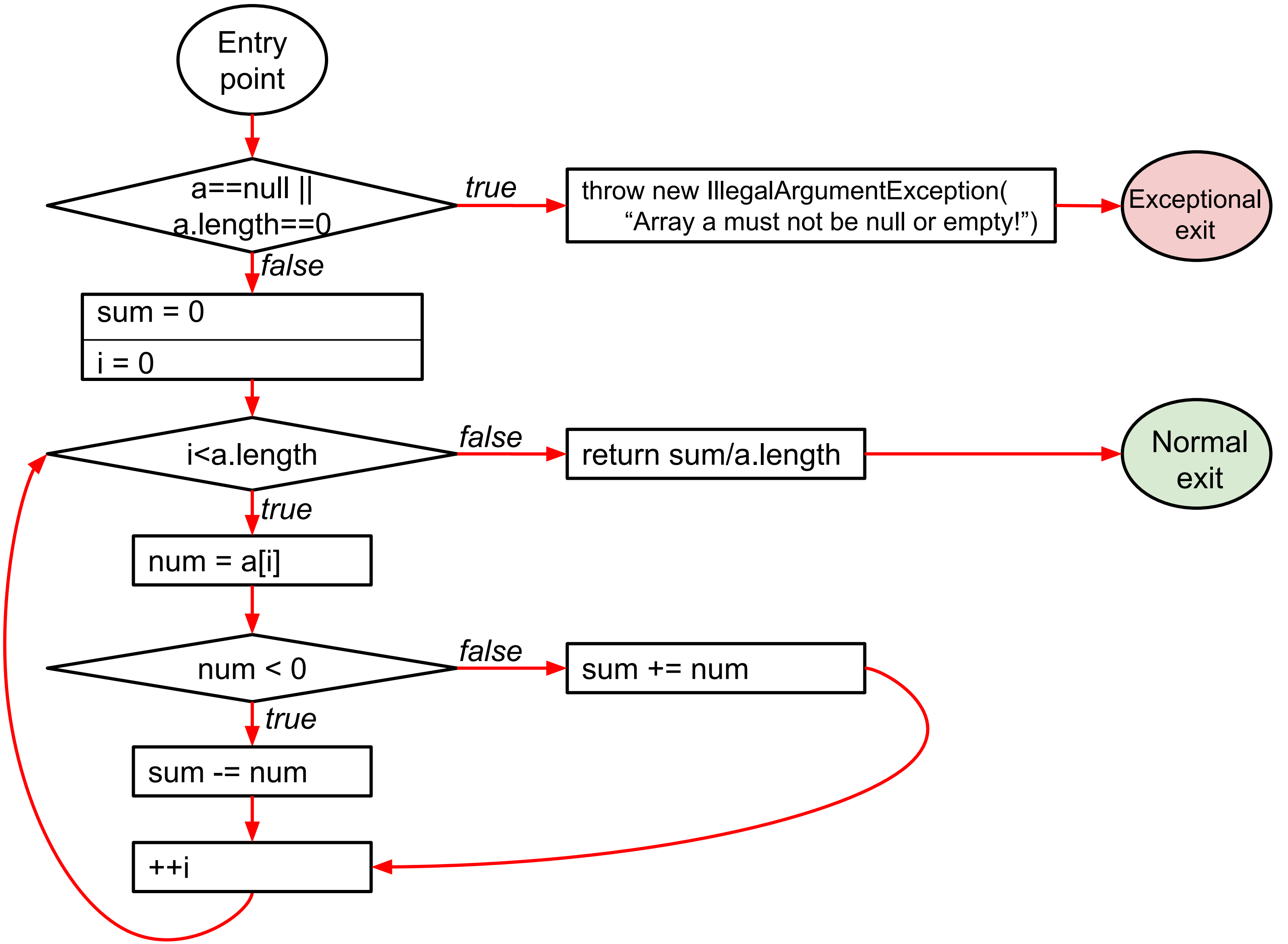


Decision coverage, a.k.a. branch coverage

- **Every decision** must evaluate to **every possible outcome** (true/false) **at least once.**
- decision = maximal boolean expression
  - may have boolean subexpressions

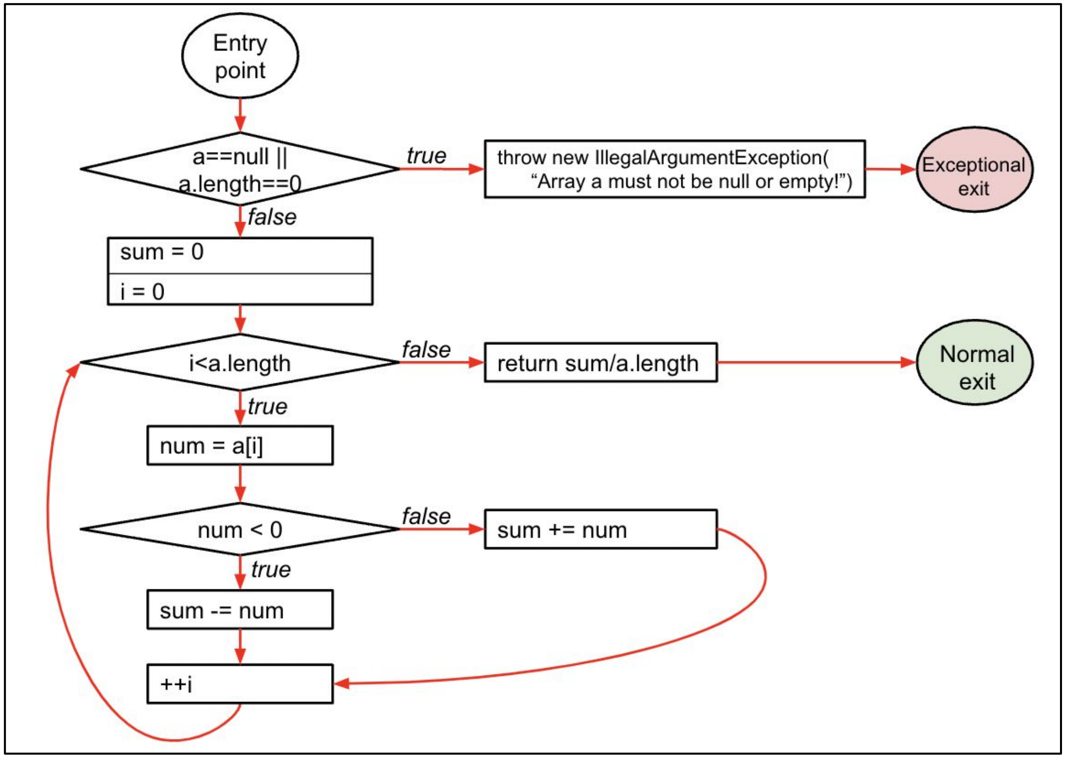


Decision coverage



Decision coverage

- **Every decision** must evaluate to **every possible outcome** (true/false) **at least once.**
- = **edge coverage** in the control-flow graph (CFG)



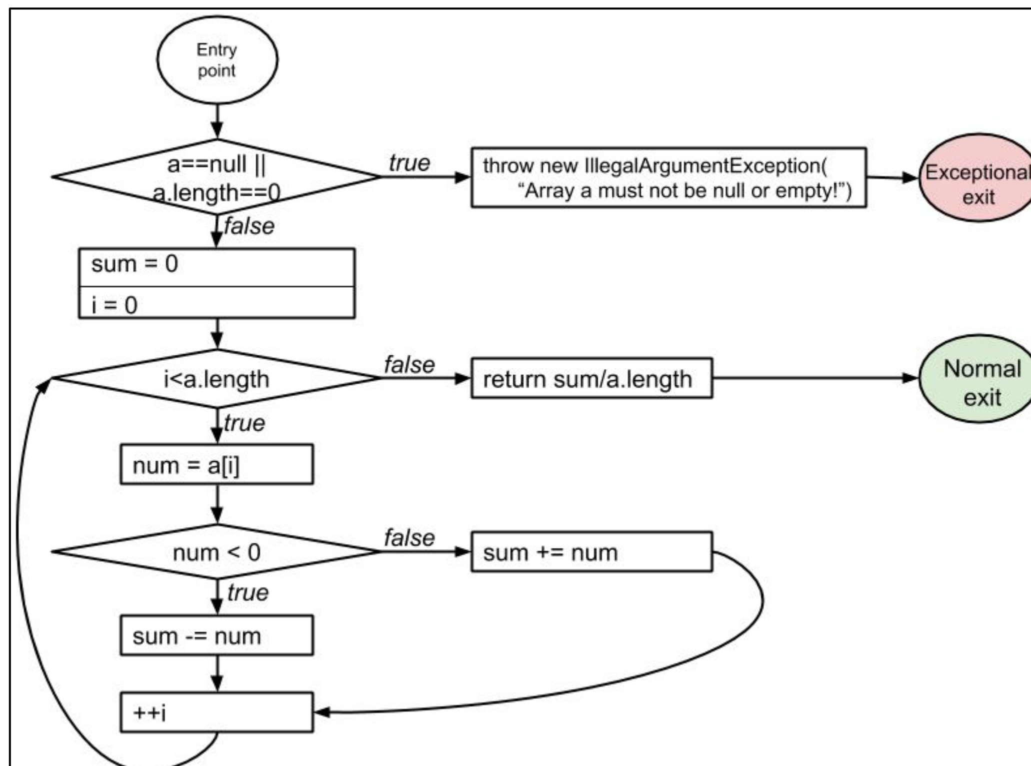
Condition coverage vs. decision coverage

Terminology

- **Condition**: a boolean expression that cannot be decomposed into simpler boolean expressions (atomic).
- **Decision**: a maximal boolean expression. It is composed of 1 or more conditions, using 0 or more logical connectors.
- **Example:** if (*a* || *b*) { … }
    - *a* and *b* are *conditions.*
    - The boolean expression *a* || *b* is a *decision.*

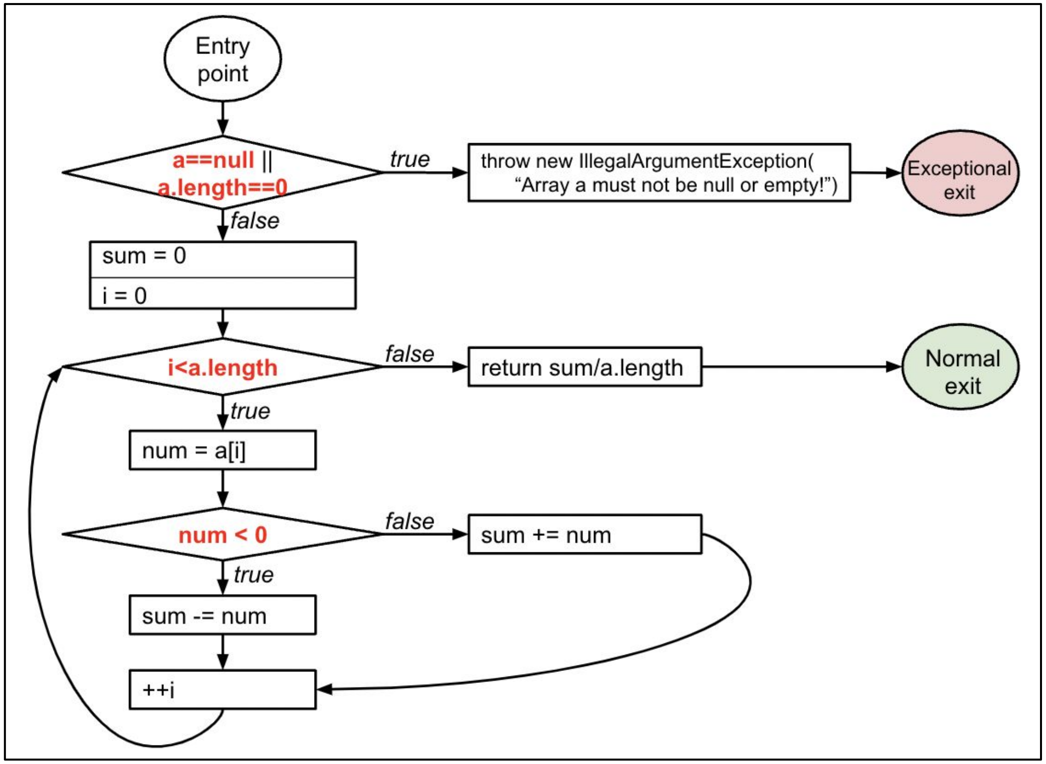
Condition coverage

- **Every condition** in the program must take on **every possible outcome** (true/false) **at least once.**
- condition = minimal boolean expression



Condition coverage

- **Every condition** in the program must take on **every possible outcome** (true/false) **at least once.**
- = **edge coverage in the assembly program**



Structural code coverage: subsumption

Given two coverage criteria A and B,
A subsumes B iff **satisfying A implies satisfying B**

Subsumption relationships:

1. Does statement coverage subsume decision coverage?
2. Does decision coverage subsume statement coverage?
3. Does decision coverage subsume condition coverage?
4. Does condition coverage subsume decision coverage?

Structural code coverage: subsumption

Given two coverage criteria A and B,
A subsumes B iff **satisfying A implies satisfying B**

Subsumption relationships:

1. **Statement** coverage **does not subsume** **decision** coverage
2. **Decision** coverage **subsumes** **statement** coverage
3. **Decision** coverage **does not subsume** **condition** coverage
4. **Condition** coverage **does not subsume** **decision** coverage

There are more coverage criteria, including MC/DC.
(MC/DC is required for safety-critical systems -- DO-178B/C.)

Decision coverage vs. condition coverage

4 possible tests for the decision *a* || *b*:

1. *a* = 0, *b* = 0
2. *a* = 0, *b* = 1
3. *a* = 1, *b* = 0
4. *a* = 1, *b* = 1

a	*b*	*a* || *b*
0	0	0
0	**1**	**1**
1	**0**	**1**
1	1	1

The test suite in the red box satisfies **condition coverage** but **not decision coverage**

a	*b*	*a* || *b*
0	**0**	**0**
0	**1**	**1**
1	0	1
1	1	1

The test suite in the red box does **not** satisfy **condition coverage** but does **decision coverage**

Neither coverage criterion subsumes the other!

Code coverage: summary

Classes in this File	Line Coverage		Branch Coverage		Complexity
Avg	100%	10/10	100%	8/8	6

```java
package avg;

public class Avg {

    /*
     * Compute the average of the absolute values of an array of doubles
     */
    public double avgAbs(double ... numbers) {
        // We expect the array to be non-null and non-empty
        if (numbers == null || numbers.length == 0) {
            throw new IllegalArgumentException("Array numbers must not be null or empty!");
        }

        double sum = 0;
        for (int i=0; i<numbers.length; ++i) {
            double d = numbers[i];
            if (d < 0) {
                sum -= d;
            } else {
                sum += d;
            }
        }
        return sum/numbers.length;
    }
}
```

- Code coverage is easy to compute.
- Code coverage has an intuitive interpretation.
- Code coverage in industry: [Code coverage at Google]
- Code coverage itself is not sufficient!

Evaluating a test suite: maximize a metric

Input: a test suite and a metric

Output: a measurement of the metric

Metrics:

- Lines of code executed = code coverage
- Decisions evaluated to true and false = branch coverage
- …
- Mutation score
    - A **mutant** is a slightly changed variant of the code

Mutation: a concrete example

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 1:

```
public int min(int a, int b) {
    return a;
}
```

Mutation: another example

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 2:

```
public int min(int a, int b) {
    return b;
}
```

Mutation: yet another example

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 3:

```
public int min(int a, int b) {
    return a >= b ? a : b;
}
```

Mutation: last example (I promise)

Original program:

```
public int min(int a, int b) {
    return a < b ? a : b;
}
```

Mutant 4:

```
public int min(int a, int b) {
    return a <= b ? a : b;
}
```

Mutation score

Input: a test suite and a set of mutants

Metric: number of test failures

(Jargon: to “kill” a mutant is for the test to fail)

Example: test suite fails for 3 of the 4 mutants; score = .75

Why is a test failure good?

Modified Condition/Decision Coverage (MC/DC)

MCDC: Modified condition and decision coverage

- **Every decision** in the program must take on **every possible outcome** (true/false) **at least once**
- **Every condition** in the program must take on **every possible outcome** (true/false) **at least once**
- **Each condition** in a decision has been shown to **independently affect** that decision's **outcome**.
  (A condition is shown to independently affect a decision's outcome by: varying just that condition while holding fixed all other possible conditions.)

Required for safety critical systems (DO-178B/C)

MC/DC: an example

```
if (a | b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Which tests (combinations of a and b) satisfy MCDC?

MC/DC: an example

```
if (a | b)
```

a	b	Outcome
0	**0**	**0**
0	1	1
1	0	1
1	**1**	**1**

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Does this test suite satisfy MCDC?

MC/DC: an example

```
if (a | b)
```

a	b	Outcome
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

MCDC is still cheaper than testing all possible combinations.

MC/DC: another example

```
if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Why is this example different?

MC/DC: another example

```
if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	--	1
1	--	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Short-circuiting operators may not evaluate all conditions.

MC/DC: yet another example

```
if (!a) ... if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
1	0	1
1	1	1

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

What about this example?

MC/DC: another example

```
if (!a) ... if (a || b)
```

a	b	Outcome
0	0	0
0	1	1
X	X	X
X	X	X

MCDC

- **Decision** coverage
- **Condition** coverage
- **Each condition** shown to **independently affect outcome**

Not all combinations of conditions may be possible.

MCDC: complex expressions

Provide an MCDC-adequate test suite for:

1. `a | b | c`
2. `a & b & c`

a | b | c

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

a & b & c

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1