Victoria University
of Wellington, New Zealand
Te Whare Wananga o te
Upoko o te Ika a Maui
Aotearoa

SWEN221: Software Development

16: Testing III

Something about A3…

```java
/**
 * An implementation of the "classical" rules of Whist.
 *
 * @author David J. Pearce
 *
 */
public class SingleHandWhist extends AbstractCardGame {

    public SingleHandWhist() {

    }

    public String getName() {
        return "Classic Whist";
    }
```

```java
/**
 * A simple variation of Whist where only a single hand is played.
 *
 * @author David J. Pearce
 *
 */
public class ClassicWhist extends AbstractCardGame {

    public ClassicWhist() {

    }

    public String getName() {
        return "Single Hand Whist";
    }
```

Calculating Code Coverage

- Example *Coverage Criteria*:
  - **Function/Method Coverage:** number of methods invoked / # methods
  - **Statement Coverage**: number of statements executed / # statements
  - **Branch Coverage**: number of branches where both true and false side tested / # branches

Control-Flow Graph

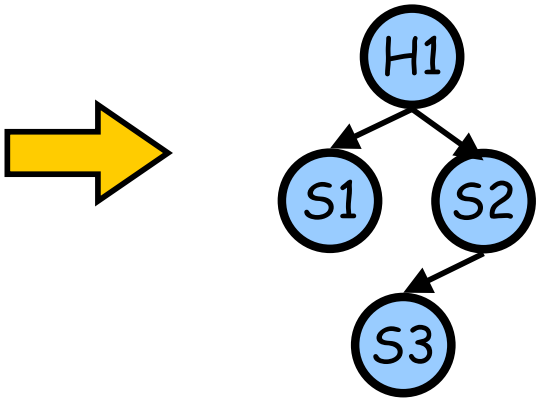
```
if(…) { return; }
else { S2 }
S3
```

```
if(…) { S1 ; S2 }
S3
```

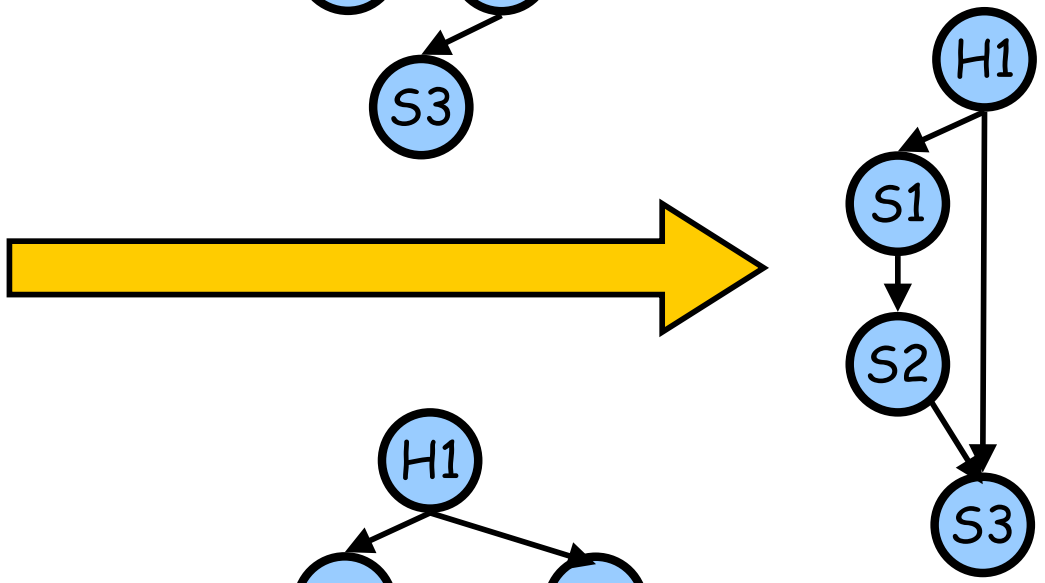
```
if(…) { S1 }
else if(…) { S2 }
else { S3 }
S4
```

Control-Flow Graph

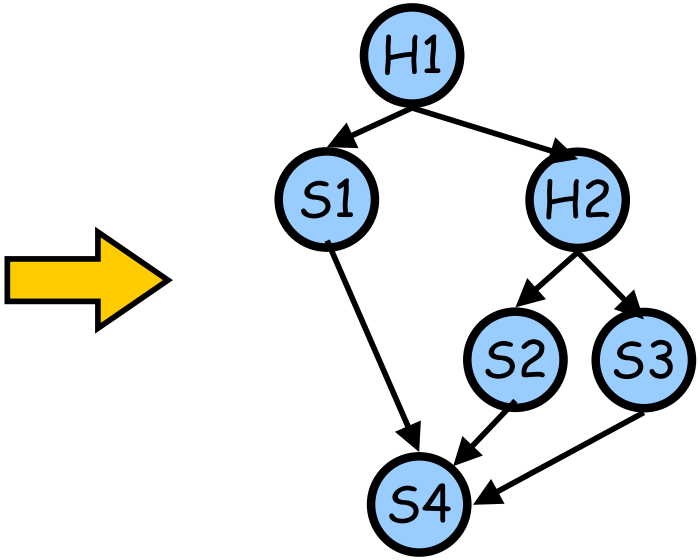
```
if(…) { return; }
else { S2 }
S3
```




```
if(…) { S1 ; S2 }
S3
```




```
if(…) { S1 }
else if(…) { S2 }
else { S3 }
S4
```

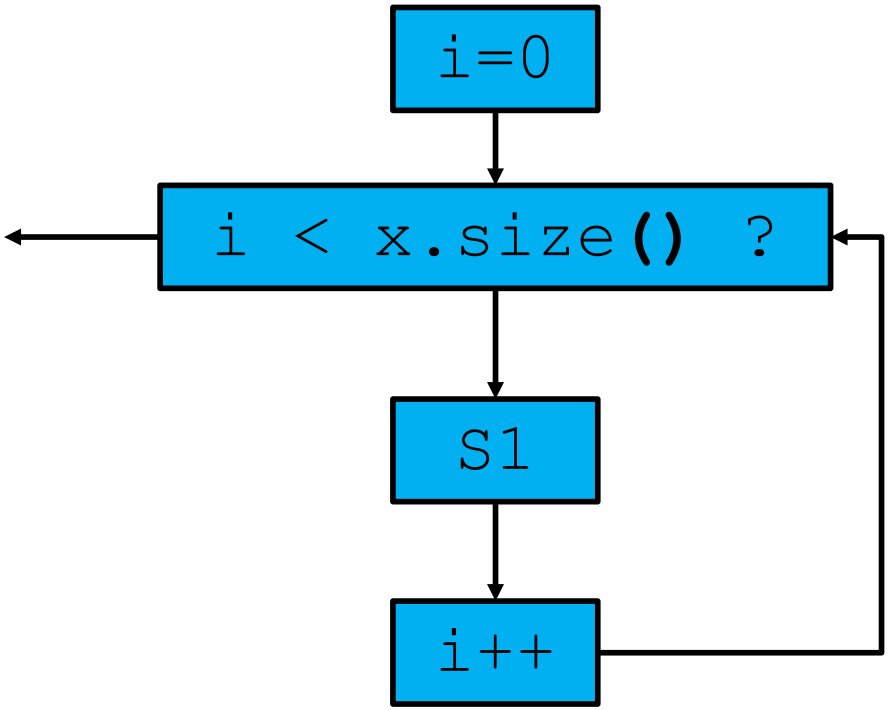


More Control-Flow Graph

```
for(int i=0; i < x.size(); i++) {
 S1;
}
```

More Control-Flow Graph

```
for(int i=0; i < x.size(); i++) {
 S1;
}
```



Calculating Code Coverage

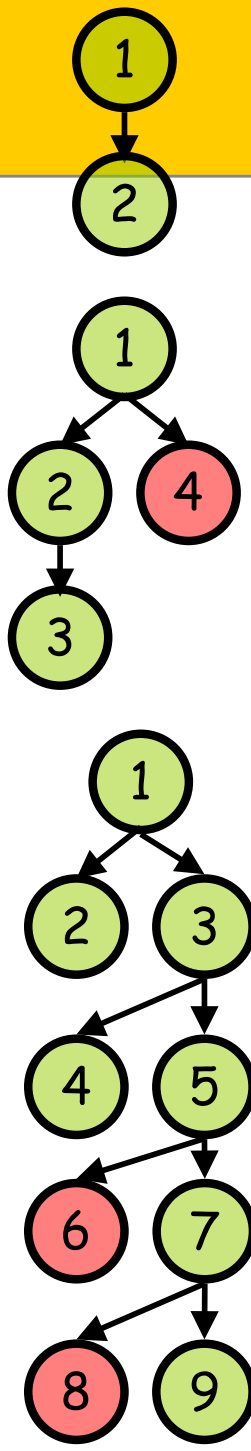
```
class Card {
  private int number, suit;

  public Card(int n, int s) { number = n; suit = s; }

  public boolean equals(Object o) {
    if(o instanceof Card) {
      Card c = (Card) o;
      return c.number == number && c.suit == suit;
    }
  return false;
}

public int compareTo(Card c) {
  if(suit > c.suit) { return -1; }
  else if(suit < c.suit) { return 1; }
  else if(number < c.number) { return -1; }
  else if(number > c.number) { return 1; }
  else { return 0; }
}}
```

Calculating Code Coverage

```
@Test void testEquals() {
  assertTrue(new Card(1,2).equals(new Card(1,2)));
}
@Test void testCompareEquals() {
  assertTrue(new Card(1,2).compareTo(new Card(1,2)) == 0);
}
@Test void testCompareLess() {
  assertTrue(new Card(2,3).compareTo(new Card(2,1)) < 0);
}
@Test void testCompareGreater() {
  assertTrue(new Card(2,1).compareTo(new Card(2,3)) > 0);
}
```

- Based on these, Calculate (as %):
  - Method Coverage
  - Statement Coverage
  - Branch Coverage

```
class Card {
  private int number, suit;

  public Card(int n, int s) { number = n; suit = s; }

  public boolean equals(Object o) {
    if(o instanceof Card) {
      Card c = (Card) o;
      return c.number == number && c.suit == suit;
    }
  return false;
}

public int compareTo(Card c) {
  if(suit > c.suit) { return -1; }
  else if(suit < c.suit) { return 1; }
  else if(number < c.number) { return -1; }
  else if(number > c.number) { return 1; }
  else { return 0; }
}}
```



Method Coverage = 3 / 3 = **100%**
Statement Coverage = 12 / 15 = **80%**
Branch Coverage = 2 / 5 = **40%**

Partial Statement Coverage

```
int sumSmallest(List<Integer> v1) {
 // sum smallest list
  int r = 0;
  for(int i=0; i < v1.size(); i++) {
    r += v1.get(i);
  }

  return r;
}

@Test void test() {
  assertTrue(sumSmallest(null) == 0);
}
```

- In EMMA some statements marked yellow
  - Indicates *partial coverage*
  - Statement corresponds to more than one CFG node
  - Some, but not all, of its nodes were executed

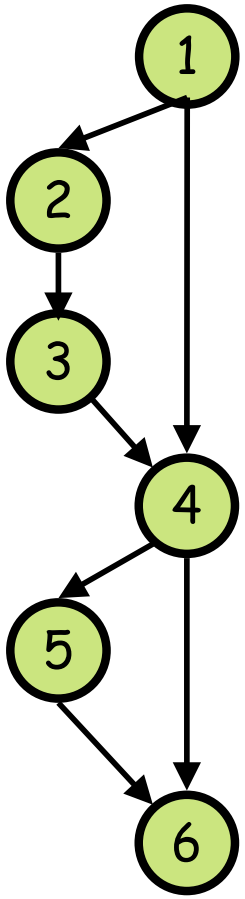
Statement & Branch Coverage

```
class Test {
 static int f(int x, int y) {
  if(x < y && y >= 0) { x = y; y = 0; }
  if(x <= y) { x = x / y; }
  return x;
}}

@Test void tester() {
 assertTrue(Test.f(0,5) == 5);
 assertTrue(Test.f(-4,-2) == 2);
}
```

- Compute (as %):
  - Statement Coverage
  - Branch Coverage
- Q) What's the problem ?

Statement & Branch Coverage

```
class Test {
 static int f(int x, int y) {
  if(x < y && y >= 0) { x = y; y = 0; }
  if(x <= y) { x = x / y; }
  return x;
}}

@Test void tester() {
 assertTrue(Test.f(0,5) == 5);
 assertTrue(Test.f(-4,-2) == 2);
}
```

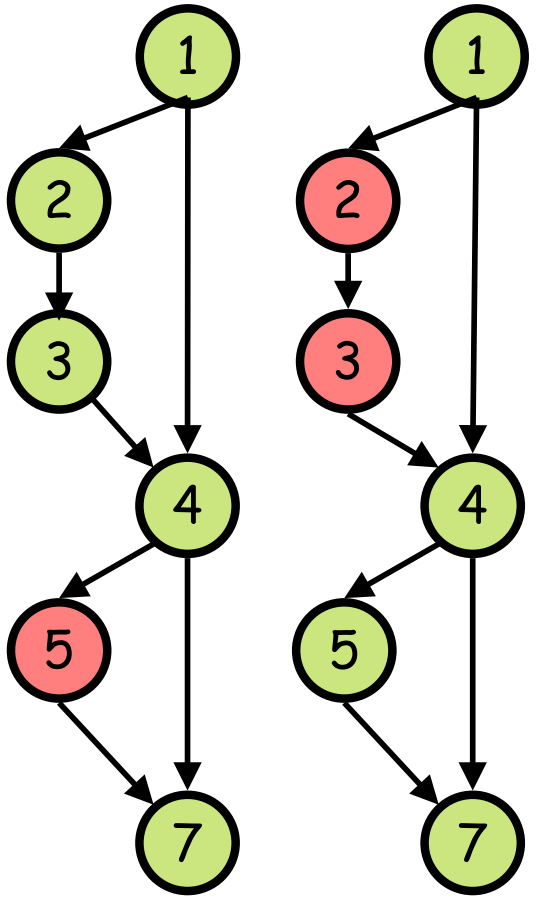


- Compute (as %):
  - Statement Coverage **= 6/6 => 100%**
  - Branch Coverage **= 2/2 => 100%**

- Q) What's the problem ?

Statement & Branch Coverage

```
class Test {
 static int f(int x, int y) {
  if(x < y && y >= 0) { x = y; y = 0; }
  if(x <= y) { x = x / y; }
  return x;
}}

@Test void tester() {
 assertTrue(Test.f(0,5) == 5);
 assertTrue(Test.f(-4,-2) == 2);
}
```

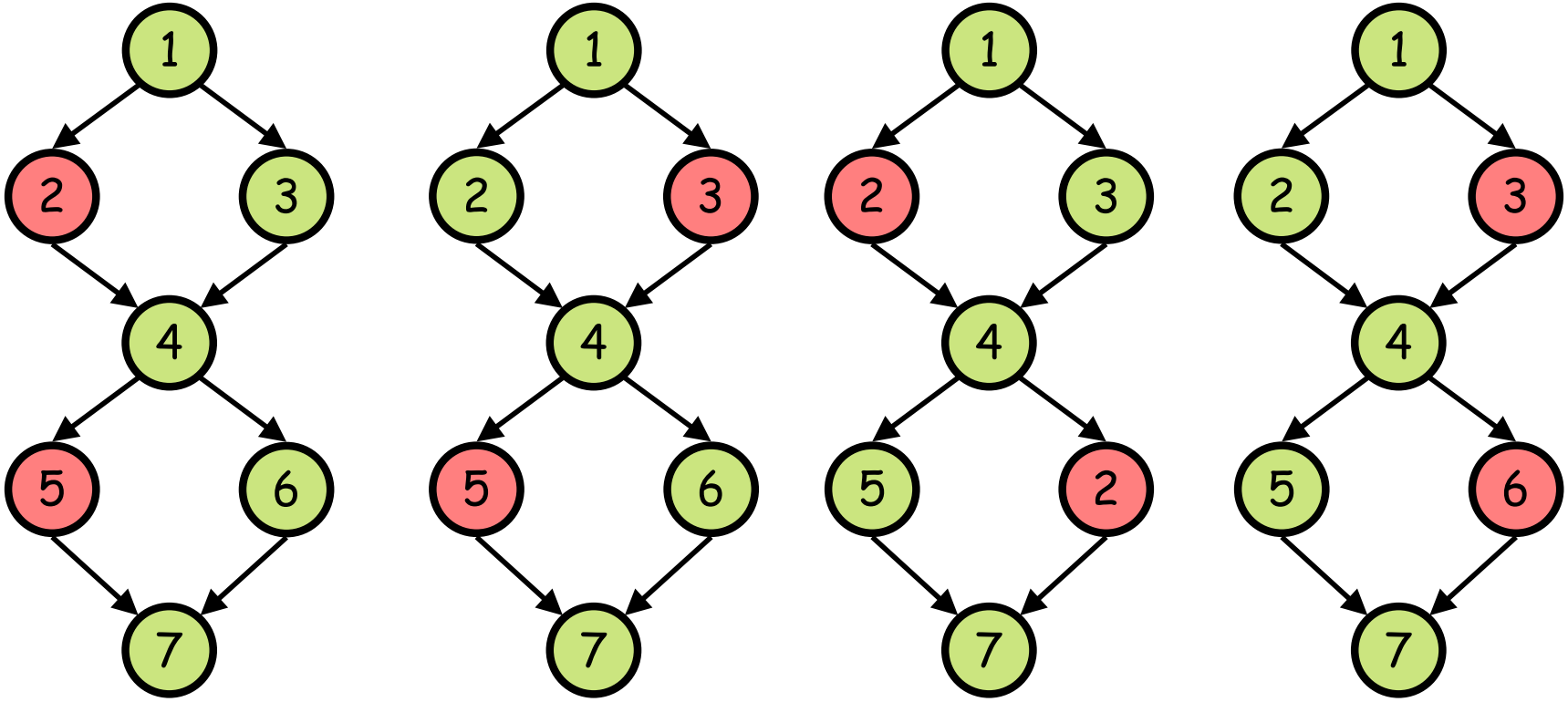


- Compute (as %):
  - Statement Coverage **= 6/6 => 100%**
  - Branch Coverage **= 2/2 => 100%**
- Q) What's the problem ?

1st only 2nd only

Execution Paths

Definition: An **execution path** a path through a method's CFG which corresponds to an execution of that method.



- Here, four distinct paths through CFG
- **100% Path Coverage**: tested all paths through CFG

Infeasible Paths

- Consider this method:

```
class Test {
 static int f(int x, int y) {
  if(x < y) { x = -y;}
  if(x >= y) { x = y; }
  return x;
}}

@Test void tester() {
 assertTrue(Test.f(0,5) == -5);
 assertTrue(Test.f(5,0) == 0);
}
```

- How many execution paths are there here?
- What path coverage is obtained here?

Loops

- Consider this method:

```
class Test {
 static int sum(int x, int y) {
  int s = 0;
  for(int i=x;i<y;++i) {
   s = s + i;
  }
  return s;
}}
```

- Q) How many execution paths are there here?

Loops

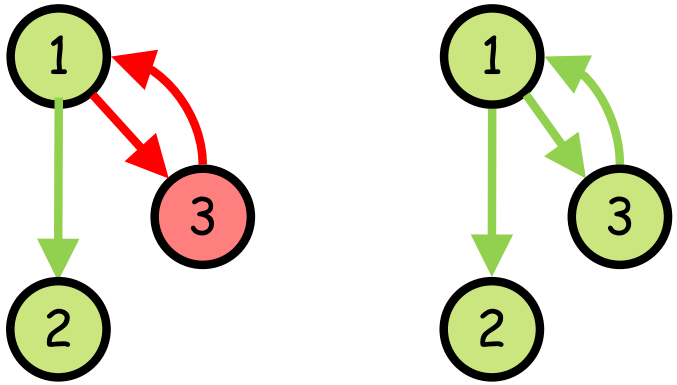
- Consider this method:

```
class Test {
 static int sum(int x, int y) {
  int s = 0;
  for(int i=x;i<y;++i) {
   s = s + i;
  }
  return s;
}}
```

- Q) How many execution paths are there here?
- A) Undefined

Simple Path Coverage

Definition: A **simple execution path** is a path through the method which iterates each loop at most once.



- Simple Path Coverage Criteria:
  - Aim to test all simple paths through a method
  - Helps keep the number of tests manageable
  - Two paths in above loop example

```
int sumSmallest(List<Integer> v1, List<Integer> v2) {
 // sum smallest list
 int r = 0;
 if(v1.size() <= v2.size()) {
  for(int i=0;i != v1.size();++i) { r += v1.get(i); }
 } else { for(int i=0;i != v2.size();++i) { r += v2.get(i); }}
 return r;
}

@Test void tester() {
 List<Integer> EMPTY = new ArrayList<Integer>();
 List<Integer> NONEMPTY = new ArrayList<Integer>();
 NONEMPTY.add(1);
 assertTrue(sumSmallest(EMPTY, EMPTY) == 0);
 assertTrue(sumSmallest(NONEMPTY, EMPTY) == 0);
 assertTrue(sumSmallest(NONEMPTY, NONEMPTY) == 0);
}
```

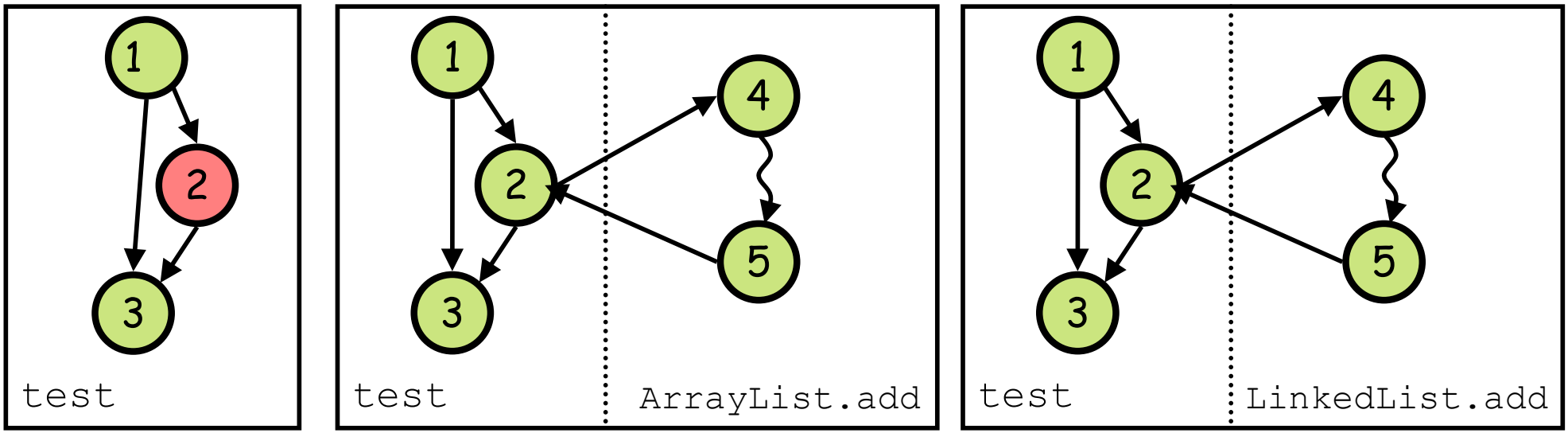
- Calculate (as %):
  - Simple Path Coverage

Coverage & Object Orientation

- Consider this method:

```
public void test(int x, List<String> ls) {
 if(x == 0) { ls.add("Hello"); }
}
```

- Now, consider some execution paths:



- So, how many execution paths are possible?

Coverage & Object Orientation

Definition: A **polymorphic execution path** is a path through one or more dynamically dispatched method calls

- Recall Dynamic Dispatch:
  - Method executed depends on dynamic type of receiver
  - So, providing different instances can have different behaviour
  - i.e. different execution paths
- Polymorphic Code Coverage:
  - Given a fixed set of classes
  - Can determine maximum number of polymorhic paths
  - Hence, can determine polymorphic code coverage

Summary

- Black-Box Testing
- White-Box Testing
- Control-Flow Graph
- Code Coverage Calculation
  - Method coverage
  - Statement coverage
  - Branch coverage
  - Path coverage