Coverage

Lecture Objectives:

1) Practice translating source code into a control flow graph.
2) Compare and contrast the number of test cases necessary meet the associated testing criteria.

In Class Exercise

- In this exercise, we will be using a tool to help us understand the relationship between the different coverage criteria.
- You will be translating source code into a graph.
- Entering that graph into a tool.
- Using the tool to determine the paths you must test.

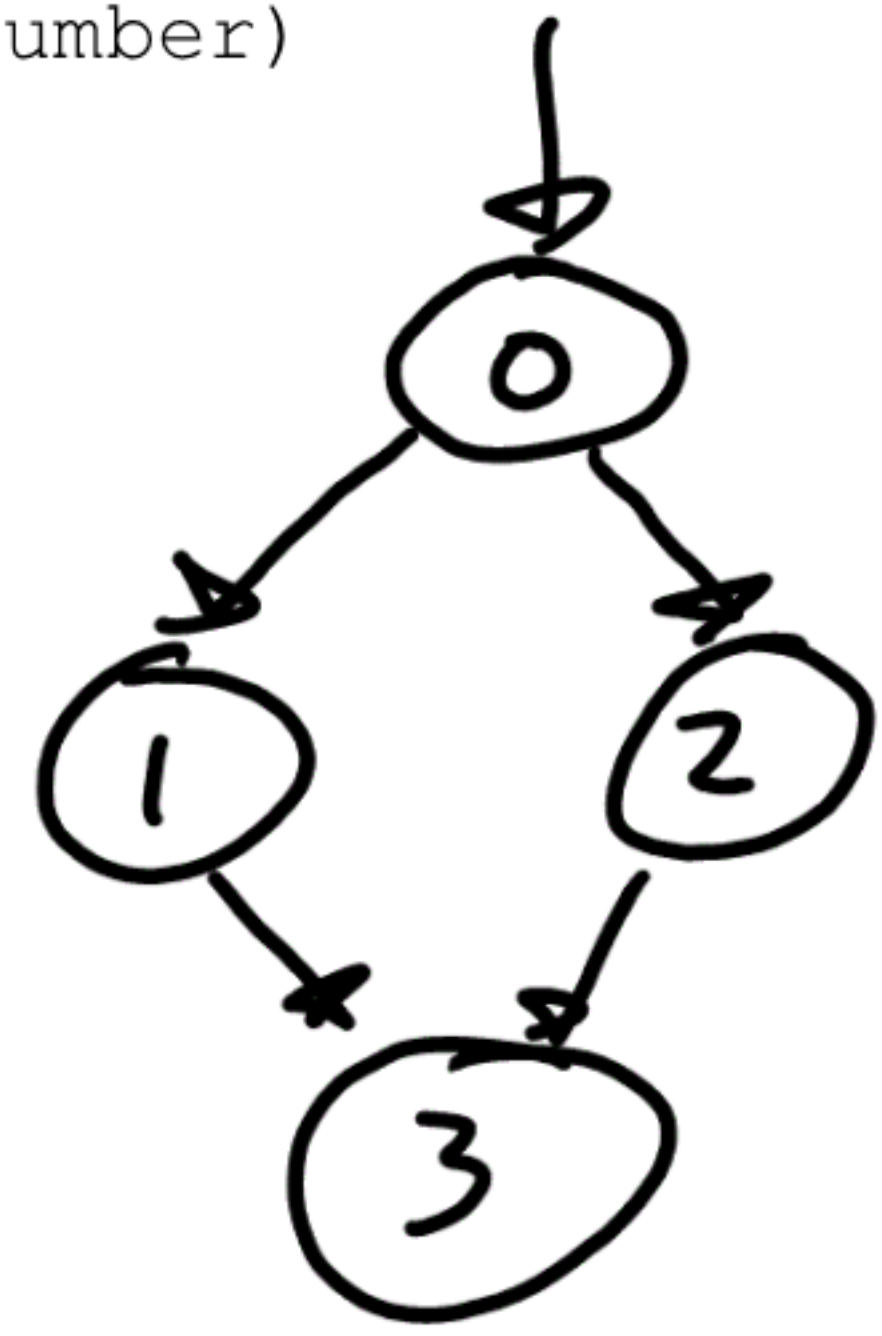
online

"Values to pass in"

Example

- Here is a segment of code
  - (Tool: http://cs.gmu.edu:8080/offutt/coverage/GraphCoverage)

```
boolean isOdd(int number)
{
   boolean retVal;
   if (number %2 ==0)
   {
     retVal = false;
   }
   else
   {
     retVal = true;
   }
}
```



Lets go…

- I will give you a segment of code
- With your next door neighbor, draw a control flow graph
- Using the test criteria I give you, figure out how to obtain the necessary coverage
- Enter the results into the tool
- I will call on some groups to present

How does this relate to the real world?

- What does this mean from last week's lab?
- Which coverage types does EMMA provide?

Going Back to code

```
public static void printit(int value) {
    value = value % 5;
    switch (value) {
            case 1:
            System.out.println("1");
            break;
            System.out.println("Not good.");
            case 2:
            System.out.println("2");
            break;
            case 3:
            System.out.println("3");
            break;
            case 4:
            System.out.println("4");
            break;
            case 5:
            System.out.println("5");
            break;
            default:
            break;
     }
    return;
}
```

Dealing with loops

(Hint: There is a bug in this code…)

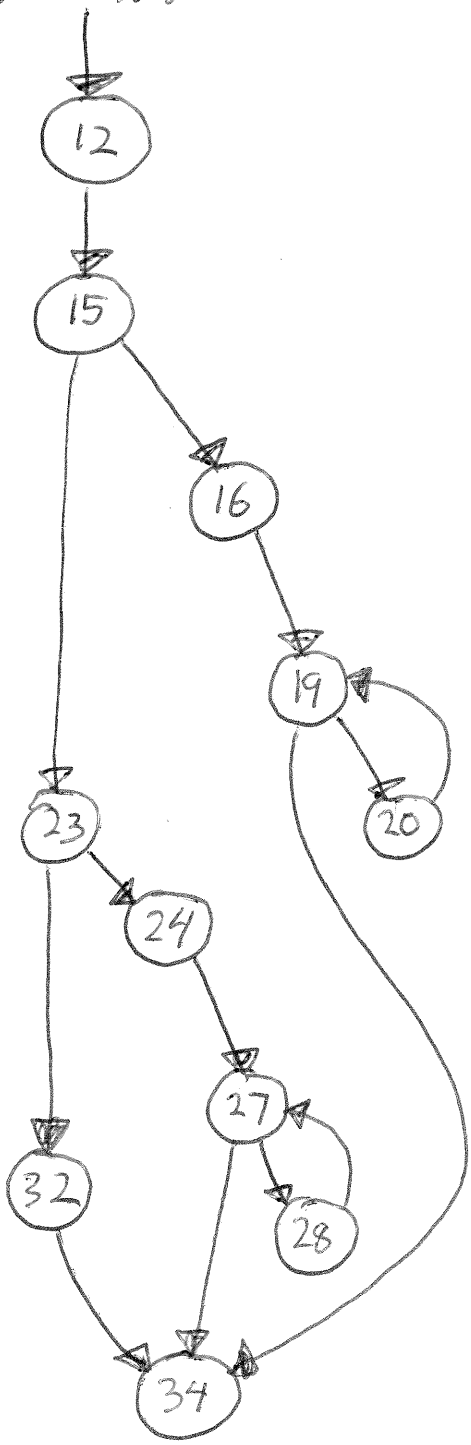
```
/**
 * The method will sum the numbers between 1 and the actual
number.
 * @param value This is the number that is to be counted up to
 * @return
 */
public static int sumNumbers(int value)
{
int retValue = 0;

  for (int index = -1; index < value; index++)
  {
     retValue += index;
  }
return retValue;
}
```

```
1  public class PowerCalculator {
2
3      /**
4       * This method will raise a given number to a given power.
5       *
6       * @param number
7       *            This is any number.
8       * @param power
9       *            This is the power that the number is to be raised to.
10      * @return The return value will be 2^n
11      */
12     public static double power(double number, int power) {
13        double retValue;
14
15        if (power > 0) {
16            int loopCount = power - 1;
17            retValue = number;
18
19            while (loopCount > 0) {
20                retValue *= number;
21                loopCount--;
22            }
23        } else if (power < 0) {
24            int loopCount = power + 1;
25            retValue = 1.0 / number;
26
27            while (loopCount < 0) {
28                retValue /= number;
29                loopCount++;
30            }
31        } else {
32            retValue = 1.0;
33        }
34        return retValue;
35     }
36
```

Power Calculator CFG

Note: Node #'s are code line #'s.



If we want Node Coverage:

3 Test cases:

Value that would generate

[12, 15, 23, 32, 34] ⇒ 0

[12, 15, 16, 19, 20, 19, 34] ⇒ 2

[12, 15, 23, 24, 27, 28, 27, 34] ⇒ [redacted] -2

Node Coverage Also gives us edge coverage

[redacted]

Edge Pair Coverage

Value That would Generate

[12 15 16 19 34] [redacted] 1

[12 15 16 19 20 19 34] 2

[12 15 16 19 20 19 20 19 34] 3

[12 15 23 24 27 34] -1

[12 15 23 24 27 28 27 34] -2

[12 15 23 24 27 28 27 28 27 34] -3

[12 15 23 32 34] 0