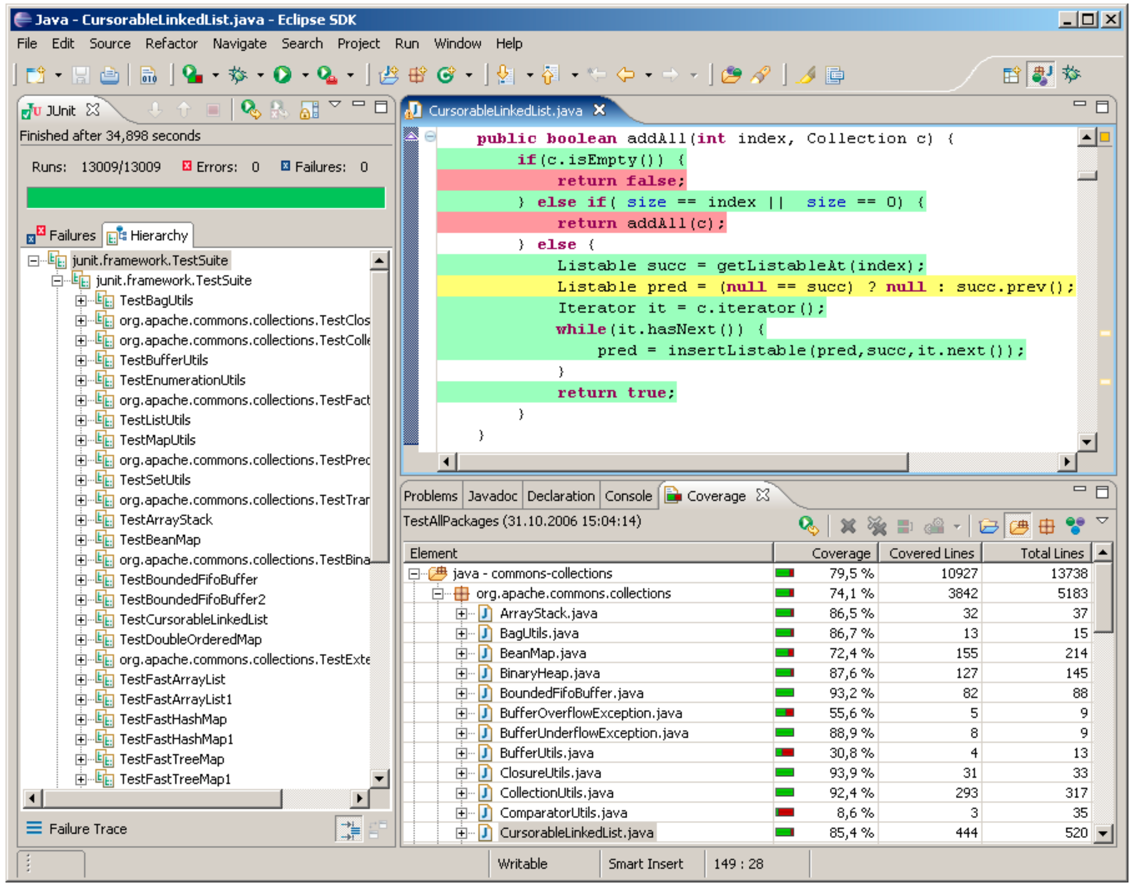
WHITEBOX TESTING

CS-HU 374 Lecture 6

ECLEMMA

EclEmma is an Eclipse plug-in structural coverage tool
https://www.eclemma.org/

- Available only as Eclipse plug-in.
- EclEmma is the Eclipse integration of JaCoCo, which is a free code coverage library for Java.
- To monitor what statements have been executed EclEmma does on-the-fly instrumentation of the code.
- EclEmma can run the programs' main.
- EclEmma can run JUnit test cases.


Java - CursorableLinkedList.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
JUnit
Failures
Hierarchy
Failure Trace
CursorableLinkedList.java
Problems
Javadoc
Declaration
Console
Coverage
Writable
Smart Insert
149 : 28

Finished after 34,898 seconds

Runs: 13009/13009 Errors: 0 Failures: 0

```
junit.framework.TestSuite
  junit.framework.TestSuite
    TestBagUtils
    org.apache.commons.collections.TestClos
    org.apache.commons.collections.TestColle
    TestBufferUtils
    TestEnumerationUtils
    org.apache.commons.collections.TestFact
    TestListUtils
    TestMapUtils
    org.apache.commons.collections.TestPred
    TestSetUtils
    org.apache.commons.collections.TestTrar
    TestArrayStack
    TestBeanMap
    org.apache.commons.collections.TestBina
    TestBoundedFifoBuffer
    TestBoundedFifoBuffer2
    TestCursorableLinkedList
    TestDoubleOrderedMap
    org.apache.commons.collections.TestExte
    TestFastArrayList
    TestFastArrayList1
    TestFastHashMap
    TestFastHashMap1
    TestFastTreeMap
    TestFastTreeMap1
```

```java
    public boolean addAll(int index, Collection c) {
        if(c.isEmpty()) {
            return false;
        } else if( size == index ||  size == 0) {
            return addAll(c);
        } else {
            Listable succ = getListableAt(index);
            Listable pred = (null == succ) ? null : succ.prev();
            Iterator it = c.iterator();
            while(it.hasNext()) {
                pred = insertListable(pred,succ,it.next());
            }
            return true;
        }
    }
```

TestAllPackages (31.10.2006 15:04:14)

Element	Coverage	Covered Lines	Total Lines
java - commons-collections	79,5 %	10927	13738
org.apache.commons.collections	74,1 %	3842	5183
ArrayStack.java	86,5 %	32	37
BagUtils.java	86,7 %	13	15
BeanMap.java	72,4 %	155	214
BinaryHeap.java	87,6 %	127	145
BoundedFifoBuffer.java	93,2 %	82	88
BufferOverflowException.java	55,6 %	5	9
BufferUnderflowException.java	88,9 %	8	9
BufferUtils.java	30,8 %	4	13
ClosureUtils.java	93,9 %	31	33
CollectionUtils.java	92,4 %	293	317
ComparatorUtils.java	8,6 %	3	35
CursorableLinkedList.java	85,4 %	444	520

INITIAL SET UP

Updated CS-HU374-Public

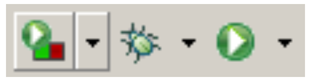
Week3 folder: w3_code and w3_test packages

Run PersonMain with “Chris 81” and “Chris -81” inputs

INSTALLING AND USING ECLEMMA PLUG-IN

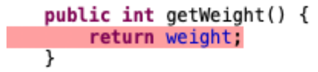
Install it using one of the ways describe at EclEmma website
http://www.eclemma.org/installation.html

- This task will require Eclipse re-start
- Onyx's Eclipse already has it installed

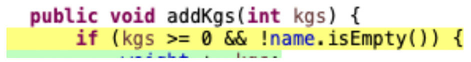
To run EclEmma click on the coverage launcher which is the first icon in the picture 

Run PersonMain on the previous two inputs using the coverage launcher. If successful you will observe that the lines in the java files are highlighted.

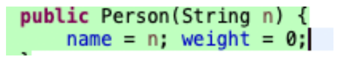
Colors

- Red - did not execute or not "covered" 

```
public int getWeight() {
    return weight;
}
```

- Yellow - partially executed or "covered" conditional statements 

```
public void addKgs(int kgs) {
    if (kgs >= 0 && !name.isEmpty()) {
```

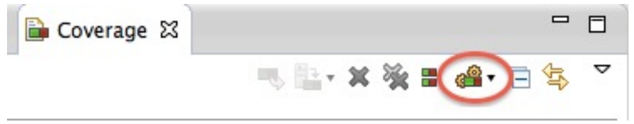
- Green - executed or "covered" 

```
public Person(String n) {
    name = n; weight = 0;
```

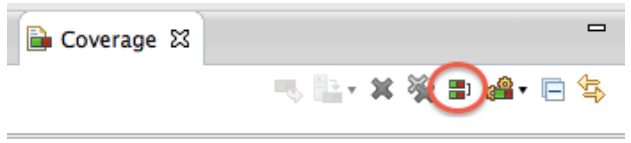
MERGING COVERAGE SESSIONS

The coverage data for each execution is kept as a separate session.

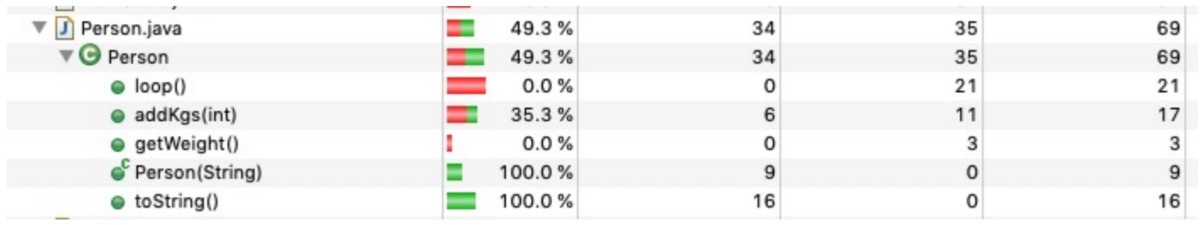
To see the coverage data from a previous run, click the active session selection button and select one that you need.



To merge the data from session together, click on the merge session button and select sessions you want to merge.



Merge the two coverage sessions for PersonMain and take a look at the coverage report on Person.java

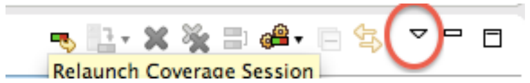


Person.java	49.3 %	34	35	69
Person	49.3 %	34	35	69
loop()	0.0 %	0	21	21
addKgs(int)	35.3 %	6	11	17
getWeight()	0.0 %	0	3	3
Person(String)	100.0 %	9	0	9
toString()	100.0 %	16	0	16

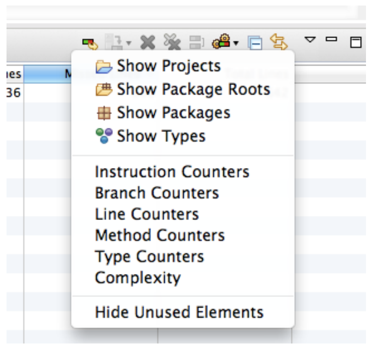
DIFFERENT COVERAGE CRITERIA

EclEmma allows you to select different coverage criteria for the coverage report

- Instruction counter
- Line counter
- Branch counter
- Method counter
- Type counter

Click on Menu button 

Select an appropriate coverage element for the reporting



STATEMENT COVERAGE CRITERION

EclEmma allows you to select different coverage criteria for the coverage report

- Instruction counter
- Line counter
- Branch counter
- Method counter
- Type counter

Counts Java bytecode instructions

Counts lines in the code

bytecode

```
// Method descriptor #33 (Ljava/lang/String;)V
  // Stack: 2, Locals: 2
  public ActiveTestSuite(java.lang.String name);
    0  aload_0 [this]
    1  aload_1 [name]
    2  invokespecial junit.framework.TestSuite(java.lang.String) [3]
    5  return
```

one line of code that invokes a method

Bytecode counter >> line counter

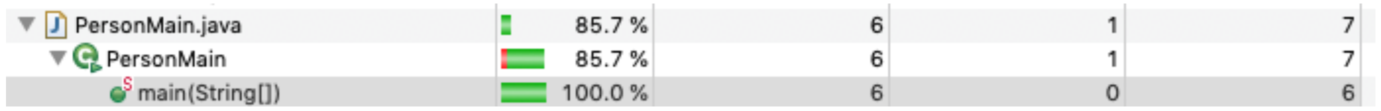
Assigns a “weight” to each line

Commonly the line coverage criterion is used

- industry standard is 85% adequate

UNREACHABLE CODE

Consider the line coverage report for Person



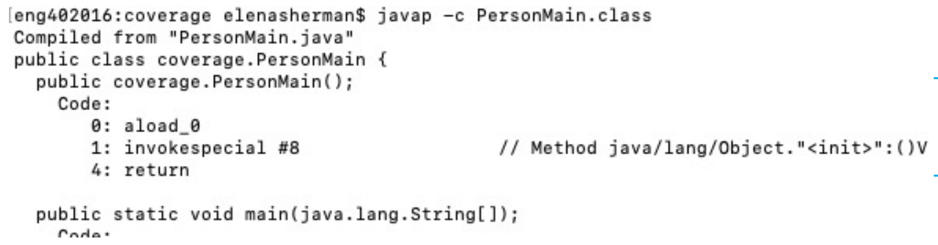
PersonMain.java	🟩	85.7 %	6	1	7
PersonMain	🟥🟩	85.7 %	6	1	7
main(String[])	🟩	100.0 %	6	0	6

The class has only one method and all its 6 lines are executed

However of the class we have the total of 7 lines

- Where this one extra line came from that downgrades our coverage?
- Let’s take a look at the compiled class file.

javap -c PersonMain.class



```
[eng402016:coverage elenasherman$ javap -c PersonMain.class
Compiled from "PersonMain.java"
public class coverage.PersonMain {
  public coverage.PersonMain();
    Code:
       0: aload_0
       1: invokespecial #8                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
```

The default constructor of PersonMain has never been called

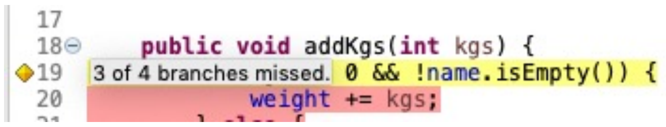
$\rightarrow$ Create PersonMain object and run EclEmma again

BRANCH COVERAGE CRITERION

EclEmma allows you to select different coverage criteria for the coverage report

- Instruction counter
- Line counter
- Branch counter
- Method counter
- Type counter

Basic Condition Coverage + Branch coverage

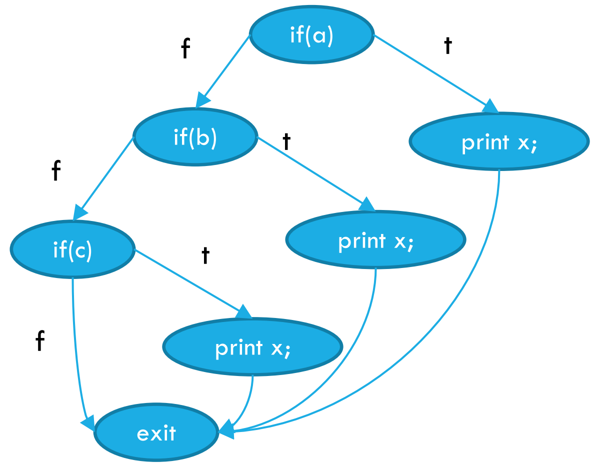


```
    public void addKgs(int kgs) {
3 of 4 branches missed. 0 && !name.isEmpty()) {
            weight += kgs;
```

A complex condition are interpreted as several branches of conditional statements with a basic condition

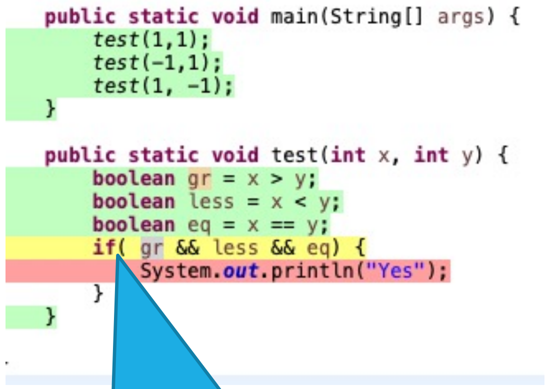
```
if( a || b || c){
  print x;
}
exit
```

```
if(a){
  print x;
} else {
  if(b){
   print x;
  } else {
   if(c) {
     print x;
}
exit
```



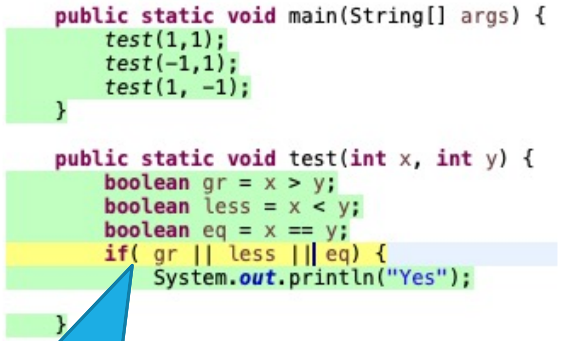
INFEASIBLE BRANCHES

Dependencies between variable could make some branches infeasible



```
public static void main(String[] args) {
        test(1,1);
        test(-1,1);
        test(1, -1);
}

public static void test(int x, int y) {
        boolean gr = x > y;
        boolean less = x < y;
        boolean eq = x == y;
        if( gr && less && eq) {
            System.out.println("Yes");
        }
}
```



```
public static void main(String[] args) {
        test(1,1);
        test(-1,1);
        test(1, -1);
}

public static void test(int x, int y) {
        boolean gr = x > y;
        boolean less = x < y;
        boolean eq = x == y;
        if( gr || less || eq) {
            System.out.println("Yes");

}
```

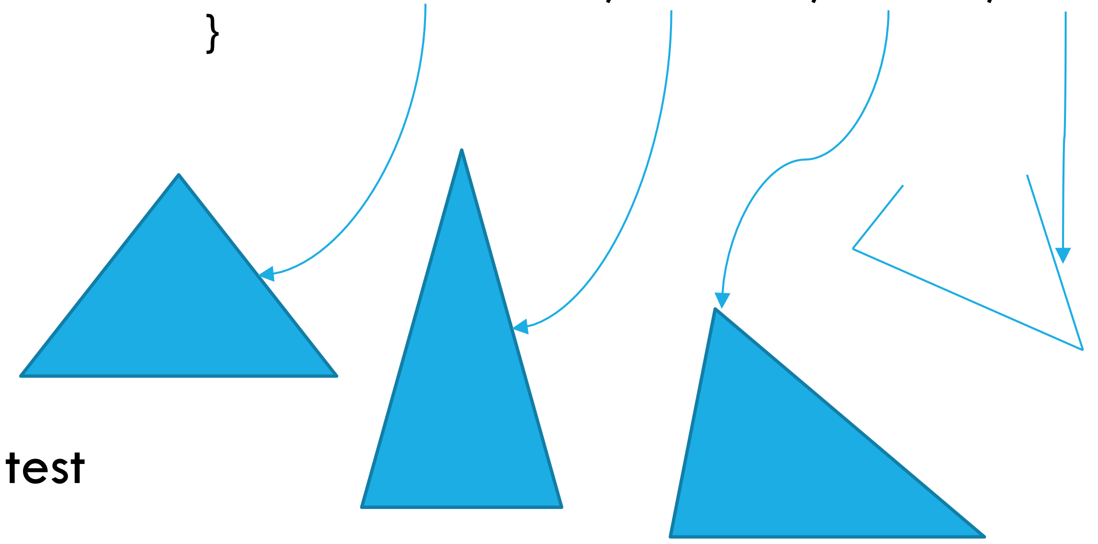
It could never happened that all variables have true values

It could never happened that all variables have false values

- Difficult to identify infeasible branches
- Is it actually infeasible or no test input is generated yet?
- Only used in safety-critical systems for official reporting
- Good internal quality measurement
  - Difficult to sell software with 50% of branches tested.
  - Easier to sell software with 85% of statement tested.
  - Which software would you trust more?
    - 50% of branches covered and 65% of statements covered
    - 25% of branches covered and 85% of statements covered

IN-CLASS WORK 1 – PART 1

```
enum TriangleType{
        EQUILATERAL, ISOSCELES, SCALENE, INVALID
}
```



```
static TtriangleType classify(int a, int b, int c)
```

Consider the source code for TriangleClassifier.java

Create JUnit with a <u>minimum</u> number of test cases that cover all statements T_s

- Does T_s covers all branches?

Add test cases to create a <u>minimum</u> branch adequate test suite T_b

Report the sizes of T_s and T_b

Run T_s and T_b on the faulty version (still jar) of the classifier. Do any of them reveal a fault?

IN-CLASS WORK – CONT.

Evaluate your Blackbox test suite T_{BB} on the TriangleClassifer.java source code

- What is the line coverage?
- What is the branch coverage?

What to show/record

Show T_s, run it and display its EclEmma coverage
Show T_b, run it and display its EclEmma coverage
Compare sizes T_s and T_b
Run T_s on the faulty version (as JUnit without coverage) – any failures?
Run T_b on the faulty version – any failures?
Run your T_{BB}, what are the line and the branch coverage?
Compare T_{BB} with T_s and T_b.

ASSIGNMENT 2

Similar what we did in class but on a larger scale:

- A program with complex code.
- Create a statement adequate test inputs.
- Create a branch adequate test inputs.
- Document your experience.

Due on Monday October 17 before class.

Advice of the week:

Blackbox and Whitebox testing are complimentary

Testing is a best effort activity