COMP3111: Software Engineering

Unit Testing

Learning Outcomes

- Be able to debug using the Eclipse debugger
- Be able to write unit tests
- Be able to generate and understand coverage reports

Supervised Lab Exercises

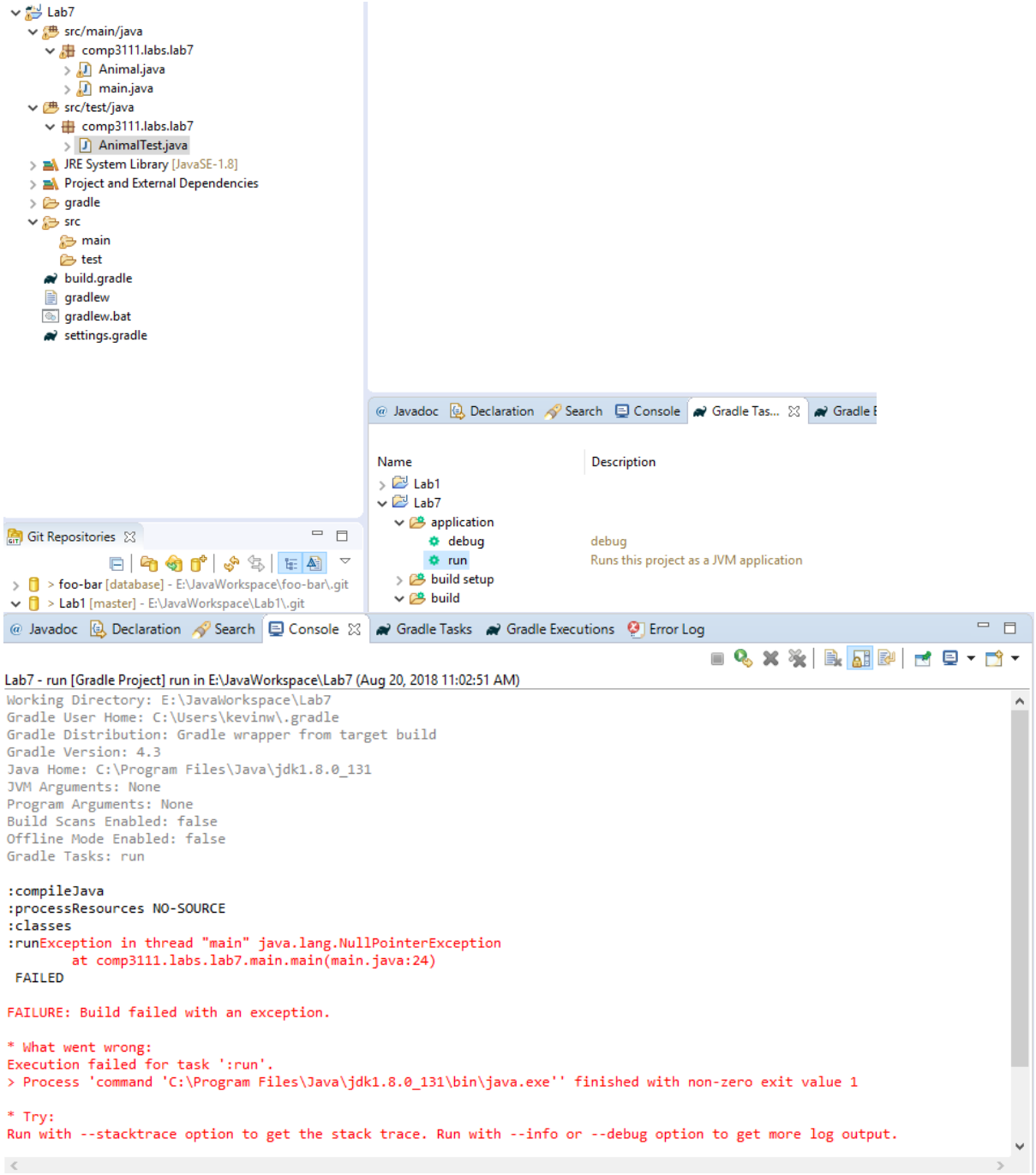
Environment: Eclipse (Version: Photon RC3 (4.8.0RC3)) with Java Development Kit (JDK 8 64-bits) installed on a Windows 10. The steps may be slightly difference if you are using other versions of Eclipse or Mac

Setup: Clone the Unit Testing lab repository from https://github.com/khwang0/comp3111-lab7 into Eclipse.

Exercise 1: Locate and fix a bug

Step 1.1: Go to the Gradle Task windows and click run. You should encounter some errors.

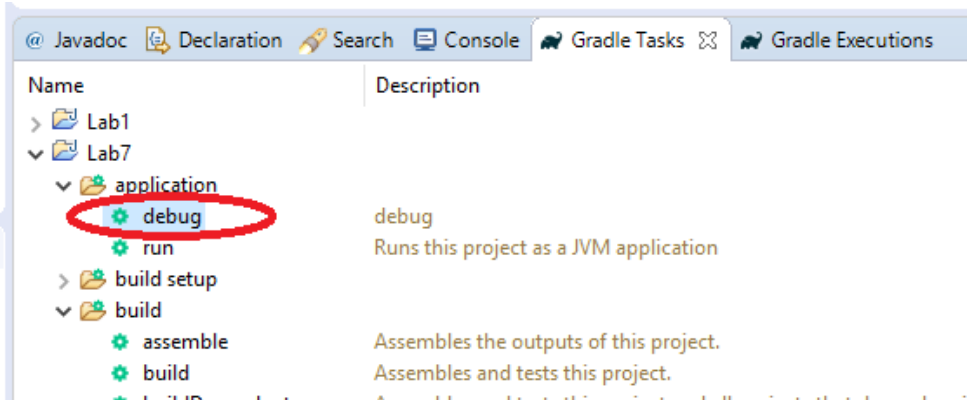
Note: if you cannot find the Gradle task windows, open it from the the menu bar > Windows > Show View > Others.



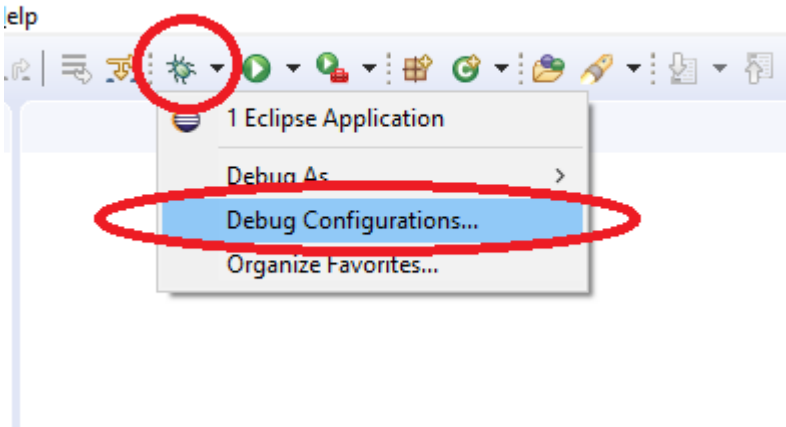
Despite this time the console states pretty clear what has happened, sometime you can have a very lengthy error in Java. So we try to enable the debugger by the following ways

Step 1.2: Go to the Gradle Task windows and click debug. The program is now prepared to be debugged.

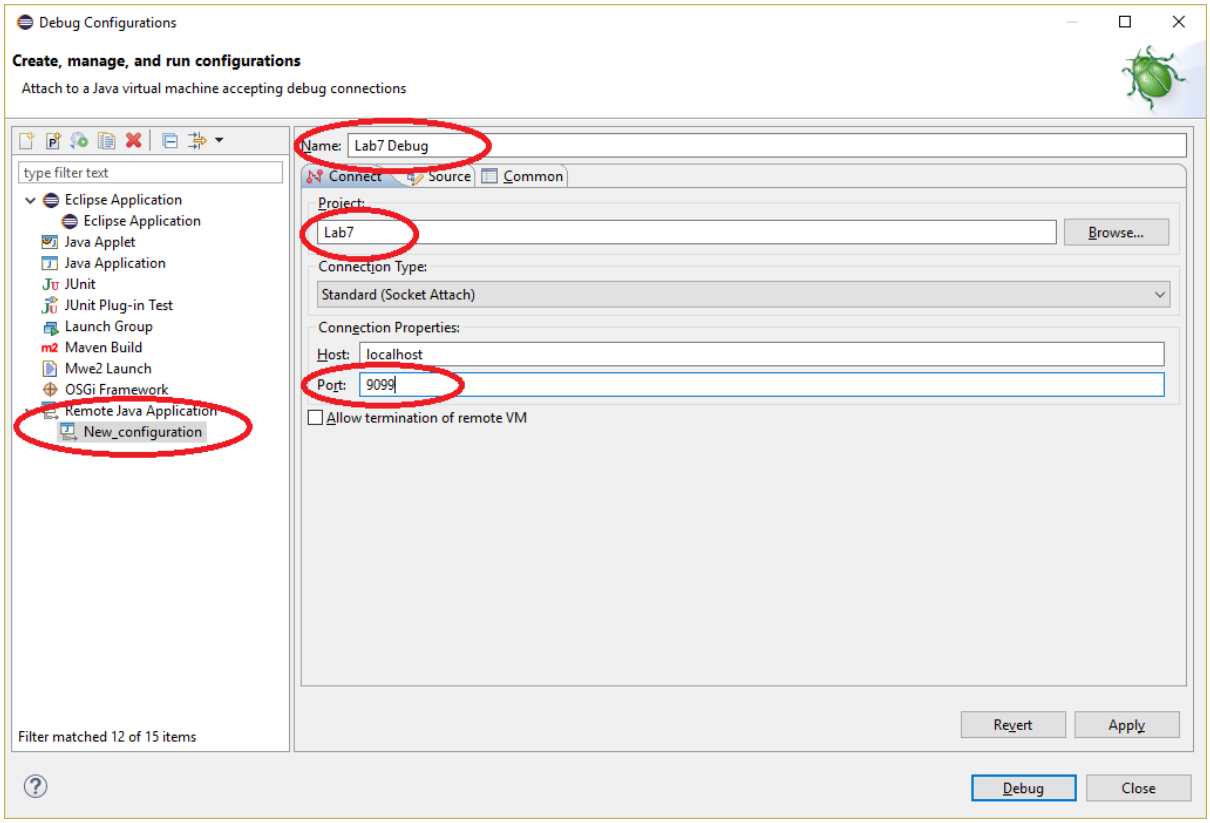
Note: this debug task is a user created task and it does not come with the default setting. In this task we bind the port 9099 for a debugger to attach to. For details, please look at the build.gradle.



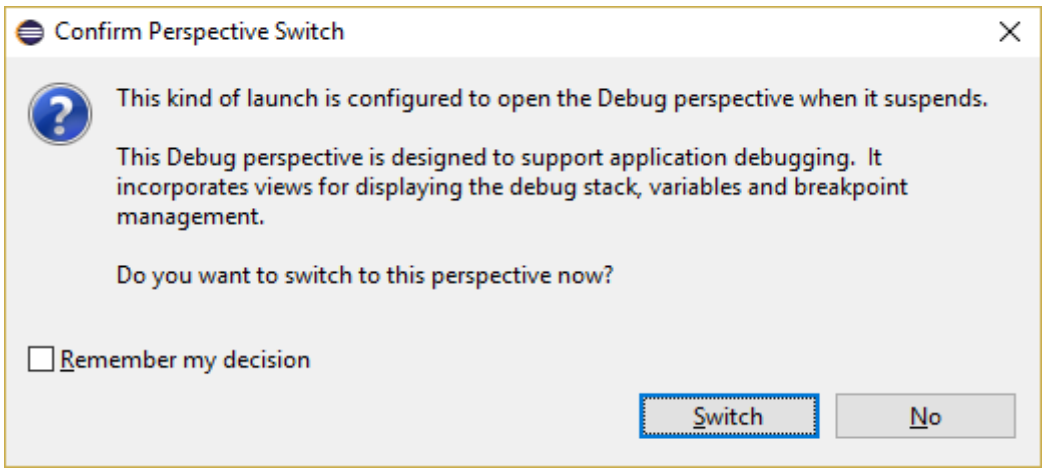
Step 1.3: Click the Debug icon on the menu bar and select Debug Configurations.



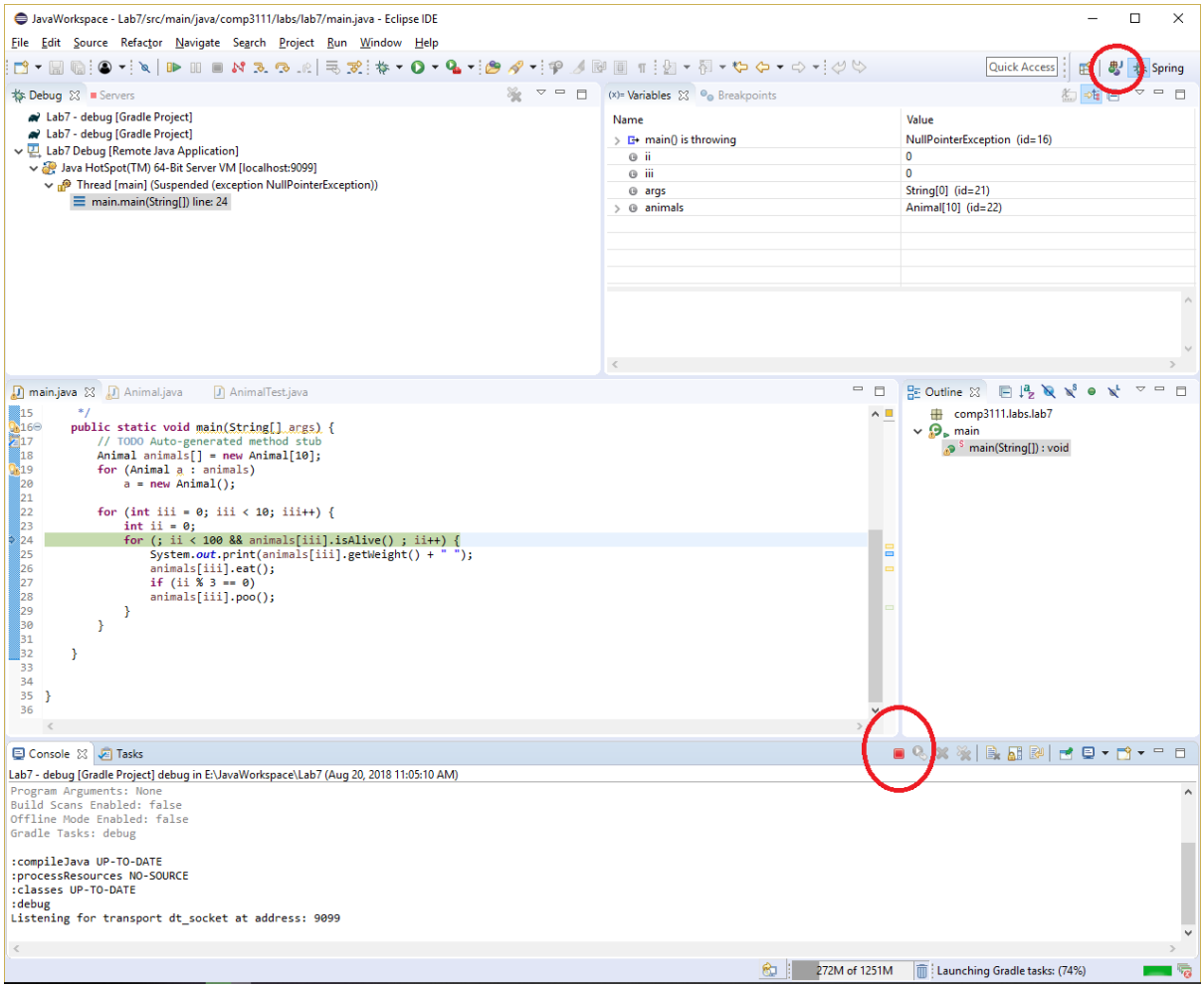
Step 1.4: Double click on “Remote Java Application” to add a new configuration. Type the info as below. Click **Debug**.



Step 1.5: You will see the following dialog, click “Switch”. This will change to the debug perspective.



Now you should see an entire different screen and pause the program at the point of the error. **Click stop button** and **click the icon that circled below** to switch back to the Java perspective.


JavaWorkspace - Lab7/src/main/java/comp3111/labs/lab7/main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Spring
Debug
Servers
Lab7 - debug [Gradle Project]
Lab7 - debug [Gradle Project]
Lab7 Debug [Remote Java Application]
Java HotSpot(TM) 64-Bit Server VM [localhost:9099]
Thread [main] (Suspended (exception NullPointerException))
main.main(String[]) line: 24
Variables
Breakpoints
Name
Value
main() is throwing
NullPointerException (id=16)
ii
0
iii
0
args
String[0] (id=21)
animals
Animal[10] (id=22)
main.java
Animal.java
AnimalTest.java
*/
public static void main(String[] args) {
// TODO Auto-generated method stub
Animal animals[] = new Animal[10];
for (Animal a : animals)
a = new Animal();
for (int iii = 0; iii < 10; iii++) {
int ii = 0;
for (; ii < 100 && animals[iii].isAlive() ; ii++) {
System.out.print(animals[iii].getWeight() + " ");
animals[iii].eat();
if (ii % 3 == 0)
animals[iii].poo();
}
}
}
}
Outline
comp3111.labs.lab7
main
main(String[]) : void
Console
Tasks
Lab7 - debug [Gradle Project] debug in E:\JavaWorkspace\Lab7 (Aug 20, 2018 11:05:10 AM)
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: debug
:compileJava UP-TO-DATE
:processResources NO-SOURCE
:classes UP-TO-DATE
:debug
Listening for transport dt_socket at address: 9099
272M of 1251M
Launching Gradle tasks: (74%)

Step 1.6: Now we need to insert a breakpoint to your program. We know the program will be stopped at line 25 or main.java. To assure the constructor of Animal has been executed, we insert a breakpoint inside the constructor of Animal.java.

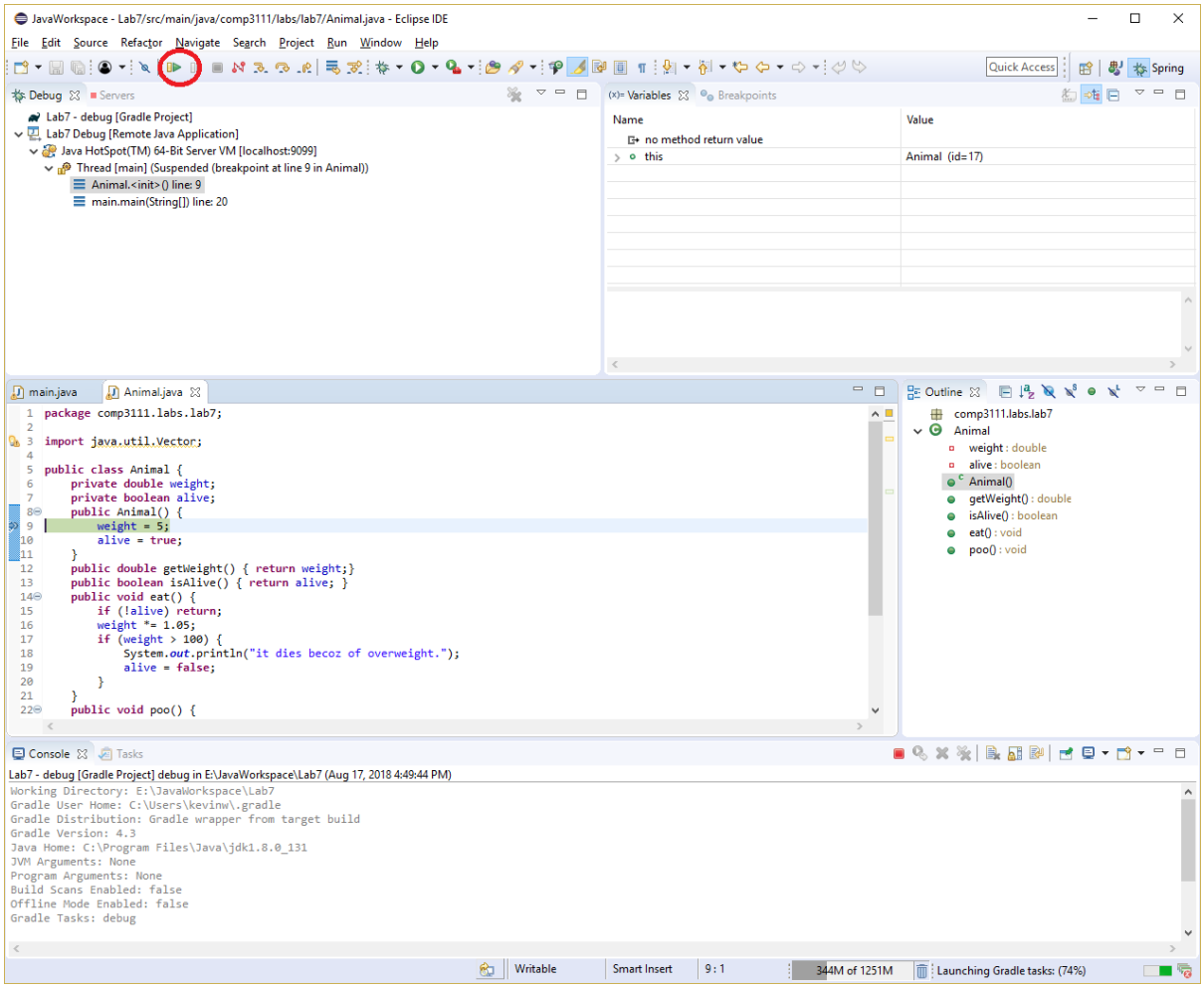
Open Animal.java and double click the line-number 9 to insert a breakpoint there.

main.java | Animal.java

```
1  package comp3111.labs.lab7;
2
3  import java.util.Vector;
4
5  public class Animal {
6      private double weight;
7      private boolean alive;
8      public Animal() {
9          weight = 5;
10         alive = true;
11     }
12     public double getWeight() { return weight;}
13     public boolean isAlive() { return alive; }
14     public void eat() {
15         if (!alive) return;
16         weight *= 1.05;
17         if (weight > 100) {
18             System.out.println("it dies becoz of overweight.");
19             alive = false;
20         }
21     }
22     public void poo() {
23         if (!alive) return;
24         weight -= 1;
25         if (weight < 0) {
26             System.out.println("it dies becoz of underweight.");
```

Note: Your program will stop at a breakpoint when you run it in debug mode. To remove the breakpoint, simply double click it again.

Step 1.7: Redo Step 1.2 and rerun the Debug from menu bar, your program should be stopping at the breakpoint. You can click the resume botton (F8) to continue the program. After clicking for 10 times, you will encounter the same error. That means the bug is still there.


JavaWorkspace - Lab7/src/main/java/comp3111/labs/lab7/Animal.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Spring
Debug
Servers
Lab7 - debug [Gradle Project]
Lab7 Debug [Remote Java Application]
Java HotSpot(TM) 64-Bit Server VM [localhost:9099]
Thread [main] (Suspended (breakpoint at line 9 in Animal))
Animal.<init>() line: 9
main.main(String[]) line: 20
Variables
Breakpoints
Name
Value
no method return value
this
Animal (id=17)
main.java
Animal.java
package comp3111.labs.lab7;
import java.util.Vector;
public class Animal {
private double weight;
private boolean alive;
public Animal() {
weight = 5;
alive = true;
}
public double getWeight() { return weight;}
public boolean isAlive() { return alive; }
public void eat() {
if (!alive) return;
weight *= 1.05;
if (weight > 100) {
System.out.println("it dies becoz of overweight.");
alive = false;
}
}
public void poo() {
Outline
comp3111.labs.lab7
Animal
weight : double
alive : boolean
Animal()
getWeight() : double
isAlive() : boolean
eat() : void
poo() : void
Console
Tasks
Lab7 - debug [Gradle Project] debug in E:\JavaWorkspace\Lab7 (Aug 17, 2018 4:49:44 PM)
Working Directory: E:\JavaWorkspace\Lab7
Gradle User Home: C:\Users\kevinw\.gradle
Gradle Distribution: Gradle wrapper from target build
Gradle Version: 4.3
Java Home: C:\Program Files\Java\jdk1.8.0_131
JVM Arguments: None
Program Arguments: None
Build Scans Enabled: false
Offline Mode Enabled: false
Gradle Tasks: debug
Writable
Smart Insert
9 : 1
344M of 1251M
Launching Gradle tasks: (74%)

Exercise 2: Writing unit tests

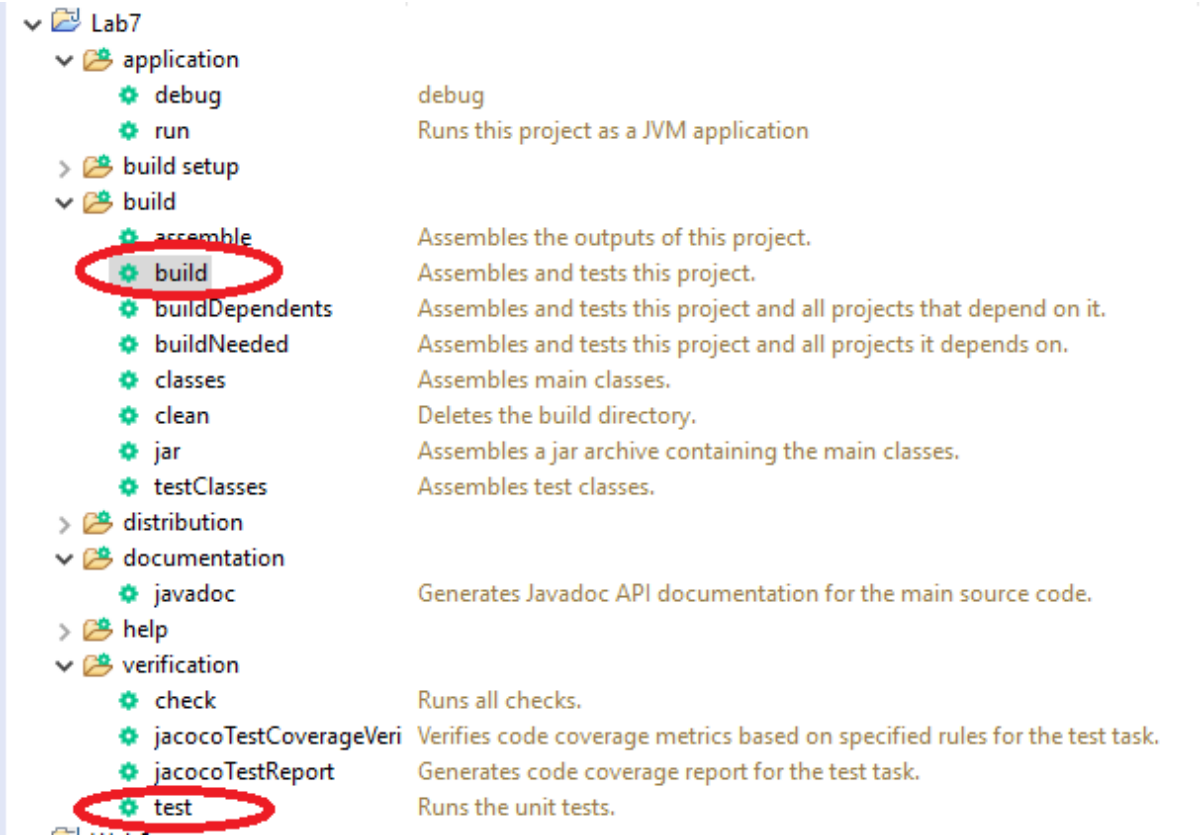
Step 2.1: Open AnimalTeset.java from src/text/java.

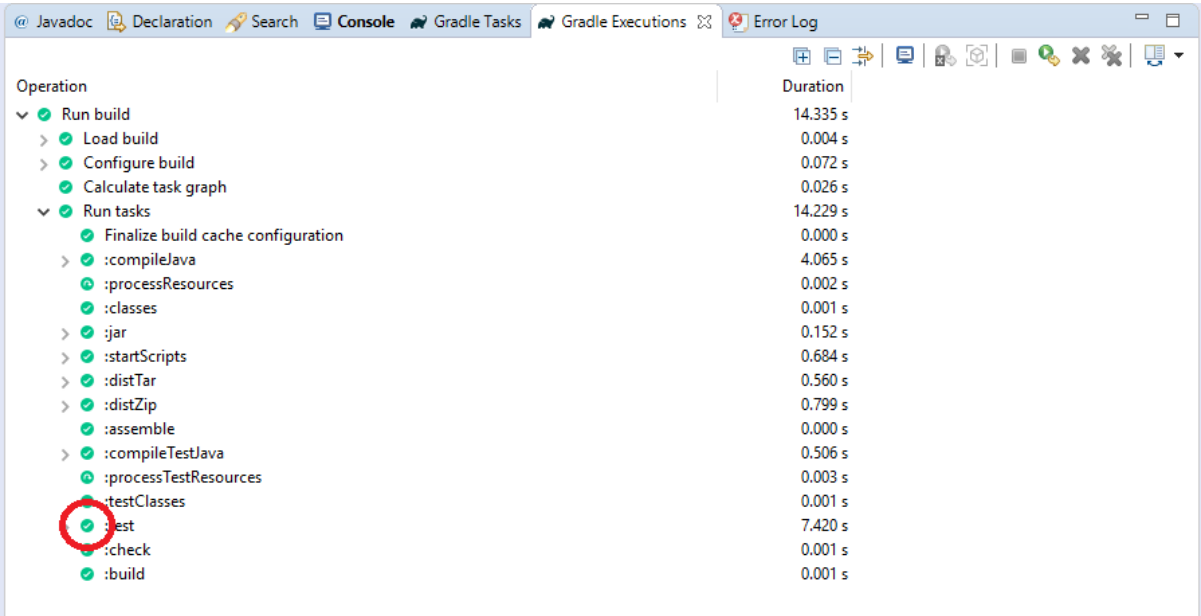
```java
package comp3111.labs.lab7;

import static org.junit.Assert.*;


public class AnimalTest {

    @Test
    public void testInitWeight() {
        Animal a = new Animal();
        assertEquals((int)a.getWeight(), 5);
    }

    @Test
    public void testDieByFeedingTooMuch() {
        Animal a = new Animal();
        for (int i = 0; i < 100; i++)
            a.eat();
        assertEquals(a.isAlive(), false);
    }

}
```

Further explanation:
This is a JUnit test.

- *Each of the function annotated with @Test will be run independently during the unit test.*
- *assertEquals is used to check whether or not the class behaves as expected. In the function testInitWeigth(), we check whether the initial weight of an animal is equals to 5.*
- *JUnit provides many assert methods. These take an expected value and an actual value, reporting a failure if these two values do not match.*

Step 2.2: Click “Gradle Tasks” > “build” and then Click “Gradle Tasks” > “test” (In most of the cases build will also do the task “test”. We click test again just to confirm the test is executed).





Note: The task test was executed successfully.

Exercise 2.3: Now change line 13 of AnimalTest.java to

```
assertEquals((int)a.getWeight(), 50);
```

Repeat Step 2.2 and you should witness a fail of test case as follows. **Undo the change.**

Operation	Duration
:jar UP-TO-DATE	0.007 s
:startScripts UP-TO-DATE	0.048 s
:distTar UP-TO-DATE	0.029 s
:distZip UP-TO-DATE	0.023 s
:assemble UP-TO-DATE	0.001 s
:compileTestJava	0.254 s
:processTestResources	0.001 s
:testClasses	0.000 s
:test	2.171 s
Resolve dependencies of :jacocoAgent	0.030 s
Resolve files of :jacocoAgent	0.002 s
Resolve files of :jacocoAgent	0.001 s
Resolve files of :jacocoAgent	0.000 s
Resolve files of :jacocoAgent	0.000 s
Resolve files of :jacocoAgent	0.001 s
Resolve dependencies of :testRuntimeClasspath	0.020 s
Resolve files of :testRuntimeClasspath	0.000 s
Execute unnamed action for :test	0.001 s
Execute executeTests for :test	2.069 s
comp3111.labs.lab7.AnimalTest	0.184 s
testDieByFeedingTooMuch	0.007 s
testInitWeight	0.005 s

Exercise 2.4: You can also browse the error report from <your project folder>\build\reports\report\tests\test\index.html. So for example my project folder is e:\JavaWorkspace\Lab7. The report is stored at e:\JavaWorkspace\Lab7\build\reports\tests\test\index.html

file:///E:/JavaWorkspace/Lab7/build/reports/tests/test/index.html

Test Summary

2	1	0	0.012s
tests	failures	ignored	duration

50% successful

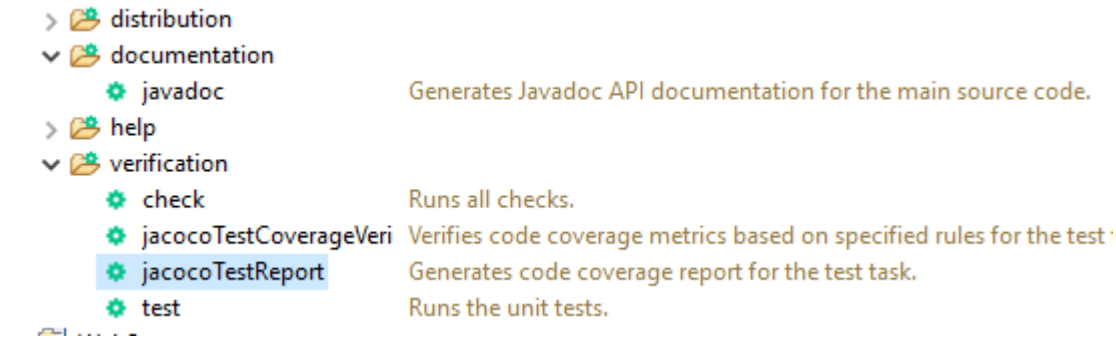
Failed tests Packages Classes

AnimalTest. testInitWeight

Generated by Gradle 4.3 at Aug 20, 2018 10:02:18 AM

Exercise 3: Generating coverage reports

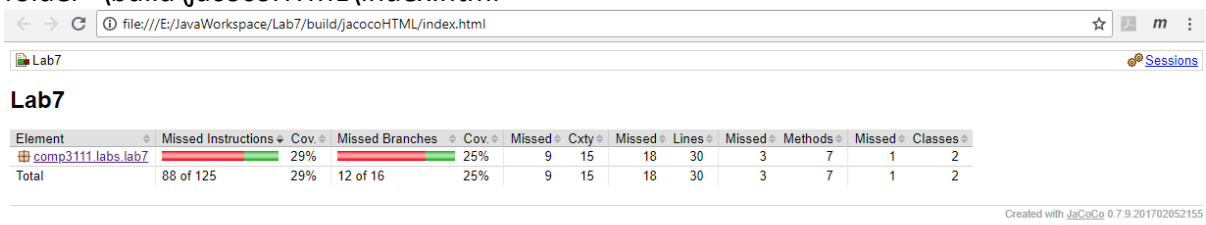
Exercise 3.1: Click “Gradle Task” > “jacocoTestReport”



And you should see this

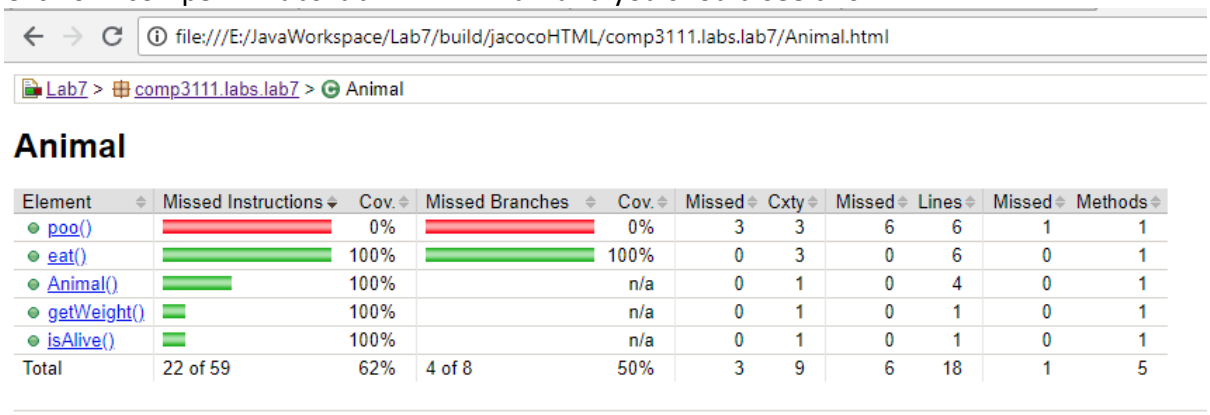
Operation	Duration
Run build	2.444 s
Load build	0.015 s
Configure build	0.100 s
Calculate task graph	0.020 s
Run tasks	2.307 s
Finalize build cache configuration	0.000 s
:compileJava UP-TO-DATE	0.251 s
:processResources	0.001 s
:classes UP-TO-DATE	0.001 s
:jacocoTestReport	2.041 s

Exercise 3.2: Go to your project folder and open <project folder>\build\jacocoHTML\index.html



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
comp3111.labs.lab7		29%		25%	9	15	18	30	3	7	1	2
Total	88 of 125	29%	12 of 16	25%	9	15	18	30	3	7	1	2

Click on “comp3111.labs.lab7” > “Animal” and you should see this:



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
poo()		0%		0%	3	3	6	6	1	1
eat()		100%		100%	0	3	0	6	0	1
Animal()		100%		n/a	0	1	0	4	0	1
getWeight()		100%		n/a	0	1	0	1	0	1
isAlive()		100%		n/a	0	1	0	1	0	1
Total	22 of 59	62%	4 of 8	50%	3	9	6	18	1	5

Further explanation:
There are two types of coverage: statement coverage and branch coverage.

- *A statement is covered if there is a test case that executes that statement*

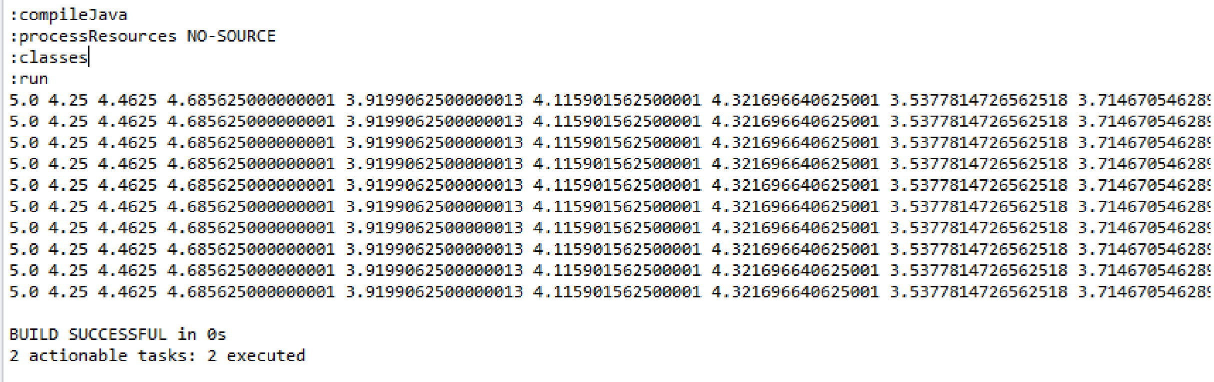
- *A branch is covered if there are test cases that evaluate the condition as true and the condition as false.*
- *As you can see the function poo() has not been tested.*

Note that 100% coverage does not mean that your code is bug free! For example, your test cases may only cover a small range of values. To make sure your code is bug free, you should always consider testing a wide range of values even if your coverage no longer changes. But, if the coverage is low (<50%), it implies there are not enough effort in testing.

Lab Activity and Assessment

Lab Activity

1) Fix the bug in the main.java so that the program ends outputs



```
:compileJava
:processResources NO-SOURCE
:classes
:run
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628
5.0 4.25 4.4625 4.685625000000001 3.9199062500000013 4.115901562500001 4.321696640625001 3.5377814726562518 3.71467054628

BUILD SUCCESSFUL in 0s
2 actionable tasks: 2 executed
```

   ,where each line ends with

   … `0.5353828341528297 0.5621519758604713 it dies becoz of underweight.`

2) Write a unit test that for the function Animal.poo(). The branch coverage of the animal class should reach 100%.

Assessment

Show your TA the following:

1) The correct executation of the program.
2) 100% pass of the tests.
3) JacocoReport that says 100% branch coverage on the class Animal.