

CIS573 -Software Engineering

1st Homework

Ning Xu 20139217

Yayang Tian 24963298

Overview

In this assignment, our group (Yayang Tian , Ning Xu) conducted Junit testing on the GPXstats library. Then, we used testing strategies cobertura and muJava to analysis and enhance the effectiveness of our tests.

(1)**Unit Test:** There are 10 methods to be tested. We used black-box and white-box testing strategies together to create the unit tests. For black box testing, we identified the equivalence classes and boundary conditions based on specifications. For white box testing, we examined the code to make sure most of the paths had been executed.

(2)**Measuring Coverage:** Actually, we used three ways to measure our coverage, namely eCobertura, Ant Build Cobertura, and Command-Line-Based Cobertura. Although eCobertura is rather time-efficient, we still prefer Ant Build Cobertura because it could provide details about branch coverage, as a way to modify previous Junit test code to get more coverage. See the following picture: We've almost achieve 100% Line Coverage and 100% Branch Coverage, while some lines are impossible to cover which we would discuss later.

Coverage Report - All Packages



Package 	# Classes	Line Coverage		Branch Coverage		Complexity
(default)	11	98%	292/296	97%	115/118	2.686
All Packages	11	98%	292/296	97%	115/118	2.686
Classes in this Package 		Line Coverage		Branch Coverage		Complexity
GPXcalculator		98%	53/54	100%	12/12	2
GPXchecker		100%	53/53	100%	50/50	45
GPXformat		87%	7/8	N/A	N/A	1
GPXlink		100%	6/6	N/A	N/A	1
GPXmetadata		100%	7/7	N/A	N/A	1
GPXObject		100%	31/31	87%	7/8	2
GPXparser		97%	65/67	100%	8/8	7
GPXtime		100%	40/40	100%	32/32	2.9
GPXtrk		100%	11/11	75%	3/4	1.6
GPXtrkpt		100%	10/10	N/A	N/A	1
GPXtrkseg		100%	9/9	75%	3/4	1.75

Figure 1

(3)**Measure adequacy:** We then Use muJava to perform mutation analysis on our test set. We provide picture capture for every method to vividly show the percentage of mutants been killed. For most methods, we've killed a big percentage, yet there are still some mutants survive, some of which are unreachable, some are equivalent mutants, while some are generated by test case that do not pass.

Next, we will evaluate the effectiveness of test case in terms of each method.

GPXchecker.checkFormat

Given the name of a file, determine whether the file is a well-formed GPX file (according to the specification above). The return value should be a GPXformat object that indicates whether the file is valid: if it is not, the object should contain an appropriate error message; if it is, the object should contain a listing of all the XML tags in the file (which are used to help parse the file).

1. Test case generation:

Black-box testing : --- Equivalent classes:

- (1) The GPXfiles should contain all elements listed in the HW1 website such as metadata, trk and their whole attribute. (The result should return a valid GPXformat, whose isValid() function should be true.)
- (2) The GPXfiles have only trk and its attribute without the metadata element.(the result should return a valid GPXformat, whose isValid() function should be true.)
- (3) The GPXfiles have more than one trksegment. (the result should return a valid GPXformat, whose isValid() function should be true.)
- (4) The GPXfiles have all the element listed in the website, but the order of them is wrong according to the specification. (the result should return a invalid GPXformat, whose isValid() function should be false.)
- (5) The GPXfiles have no trk or have no essential attributes listed in the website such as the name of trk, etc. (the result should return a invalid GPXformat, whose isValid() function should be false.)

Black-box testing : --- Boundary conditions

- (1)The GPXfiles have no metadata and trk elements, (the result should return a invalid GPXformat, whose isValid() function should be false.)
- (2)The GPXfiles have two trk elements, (the result should return a invalid GPXformat, whose isValid() function should be false.)
- (3)The GPXfiles have only one trkpoints, (the result should return a valid GPXformat, whose isValid() function should be true.)

White-box testing:

Through reading GPXchecker.java, we find the following faults:

- (1)There is no trksegment loop, so GPXfiles with more than one trksegments will return GPXformat with false valid.
- (2)The name of trk could be null in the code while the specification says that the trk must have exactly one name.
- (3)There is a rule about the order of the attribute lat and lon while there are no constraints about that in the code.
- (4)There are no constraints about the number of the href attribute while the specification says there should be only one.

2. Adequacy Criteria

For this method, we create total 26 test cases with 4 test cases failures. Each generating method listed above has one or more test cases. We obtain the **100% statement coverage and 100% branch coverage**.

Since we generate many test cases according to the while box principle to obtain a high statement & branch coverage, we feel it's adequate enough to reveal all the faults in the code. Thus we stop testing this method.

3. Failure Discovered

The 4 of the 26 test cases failed.

- (1) Failure 1: test case 1 have two trksegments, however, there is no loop sentence for generating trksegments. Therefore the checker function will misunderstand the valid test case as invalid one.
- (2) Failure 2: test case 2 have two href attributes, it should return an invalid format while since there is no judgment about the number of href, the checker function will misunderstand the invalid test case as valid one.
- (3) Failure 3: test case 3 reverse the order of lat and lon attributes, it should return an invalid format while since there is no judgment about the order of lat and lon, the checker function will misunderstand the invalid test case as valid one.
- (4) Failure 4: test case 4 have two name for one trk element, it should return an invalid format while since there is no judgment about the number of name attribute, the checker function will misunderstand the invalid test case as valid one.

4. Coverage

"As shown in Figure 1, our tests achieved **100% statement coverage and 100% branch coverage** in the GPXchecker class.

5. Mutation Analysis

We performed mutation analysis and found that our test sets killed 98% of the 144 Traditional mutants and 0% of 1 Class mutants. (see Figure above). Since it's a relatively high kill-rate, we don't generate special test case to get rid of AODS_22 and AORS_58.

TestCase RunnerTraditional Mutants ViewerClass Mutants Viewer

☐ Execute only class mutants

☐ Execute only traditional mutants

☒ Execute all mutants

Class : GPXchecker

Method : GPXformat_checkFormat(java.lang.String)

TestCase: MucheckerTest

Time-Out : 3 seconds

RUN

Op	#
AORB	0
AORS	60
AOIU	2
AOIS	4
AO...	0
AODS	20
ROR	18
COR	0
COD	0
COI	18
SOR	0
LOR	0
LOI	22
LOD	0
ASRS	0

Total : 144

Op	#
IHI	0
IHD	0
IOD	0
IOP	0
IOR	0
ISI	0
ISD	0
IPC	0
PNC	0
PMD	0
PPD	0
PCI	0
PCC	0
PCD	0
PRV	0
OMR	0
OMD	0
OAN	0
JTI	0
JTD	0
JSI	0
JSD	0
JID	0
JDC	0
EOA	0
EOC	0
EAM	1
EMM	0

Total : 1

Traditional Mutants Result

Live Mutants #	2
Killed Mutants #	142
Total Mutants #	144
Mutant Score	98.0%

Live

AODS_22
AORS_58

Killed

AODS_1
AODS_10
AODS_12
AODS_13
AODS_14
AODS_15
AODS_16
AODS_17
AODS_18
AODS_19
AODS_20
AODS_21
AODS_3
AODS_4
AODS_5
AODS_6
AODS_7
AODS_8
AODS_9

Class Mutants Result

Live Mutants #	1
Killed Mutants #	0
Total Mutants #	1
Mutant Score	0.0%

Live

EAM_1

Killed

GPXparser.parser

Given the name of a file and a GPXformat object, create a GPXObject that holds all the data from the file. Return null if the file is not valid or if an error occurs in parsing it.

1. Test case generation:

Black-box testing --- Equivalent classes:

(1)When GPXformat or filename is null, the returned result should be null also. (The GPXObject should be null)

(2)When GPXformat and filename are both valid, the returned GPXObject should contain all the information in the file. (The GPXObject should be valid.)

(3)When GPXformat is invalid which means the GPXfile is bad-formatted, the returned result should be null.(The GPXObject should be null)

Black-box Testing--- Boundary conditions:

(1)The GPXfile has two trksegments which is valid should be transformed into a valid GPXObject. (The GPXObject should be valid.)

(2)The GPXfile has wrong format (such as the wrong order of lat and lon attributes, etc., the failures pointed out in the GPXchecker section) should returned null.(The GPXObject should return null.)

White-box testing:

(1)The code to read the “lon” attribute is wrong since it reads from 6 to (end-1), which omits the first character in the corresponding lon string. (The GPXObject is not the same as in the GPXfiles).

(2)The code to read name attribute is wrong since the code identify the ending of a string by the symbol “<”. When the name contains the symbol “<”, it will misunderstand the end of the name string. (The GPXObject is not the same as in the GPXfiles).

(3)The code to generate the final GPXObject is wrong since it renew the metadata, therefore the metadata of GPXObject is null.

2. Adequacy Criteria

For this method, we created 8 test cases for each of the equivalence classes and boundary conditions, which achieved **97% statement and 100% branch coverage**.

we feel it's adequate enough to reveal all the faults in the code. Thus we stop testing this method.

3. Failure Discovered

There are 5 errors and 4 failures for the 6 test cases.

(1)**error 1:** test case 1 try to compare the original GPXObject.toString() (which is valid formatted) and the returned GPXObject.toString(). Since the the code renew the metadata of returned GPXObject in the end, the returned GPXObject.toString() points to Null. Therefore it results to an error.

(2)**error 2:** test case 2 contain a name attribute as "Walking ahead,<>Go on!" which should be considered to be valid in the normal life. But since the code identify the ending of a string by the symbol "<". When the name contains the symbol "<", it will misunderstand the end of the name string.

(3)**error 3:** test case 8 contains no metadata. This would render a NullPointerException. Because the code wrongly renew the object twice near the end of the program. Hence the generated meatdata, becomes a null pointer.

(4)**error 4:** test case 9 get wrong filename input. No matter non-existent or void input filename, it would also return a NullPointerException. The reason is similar to the analysis above. This is fatal error, because all the things generated would result in a null pointer, which is of no use to other methods.

(5)**error 5:** test case 3 contains two trksegments, since the code doesn't consider this condition, it will result to the error when construct the GPXObject in the parser function..

(6)**failure 1 & 2:** test case 4,5 compareS the original GPXObject.trk().toString() (which is valid formatted) and the returned GPXObject.trk().toString(), since the code of reading the "lon" attribute is wrong which omits the first character in the corresponding lon string (reads from 6 to (end-1)). Therefore the results should have difference, which cause the failure.

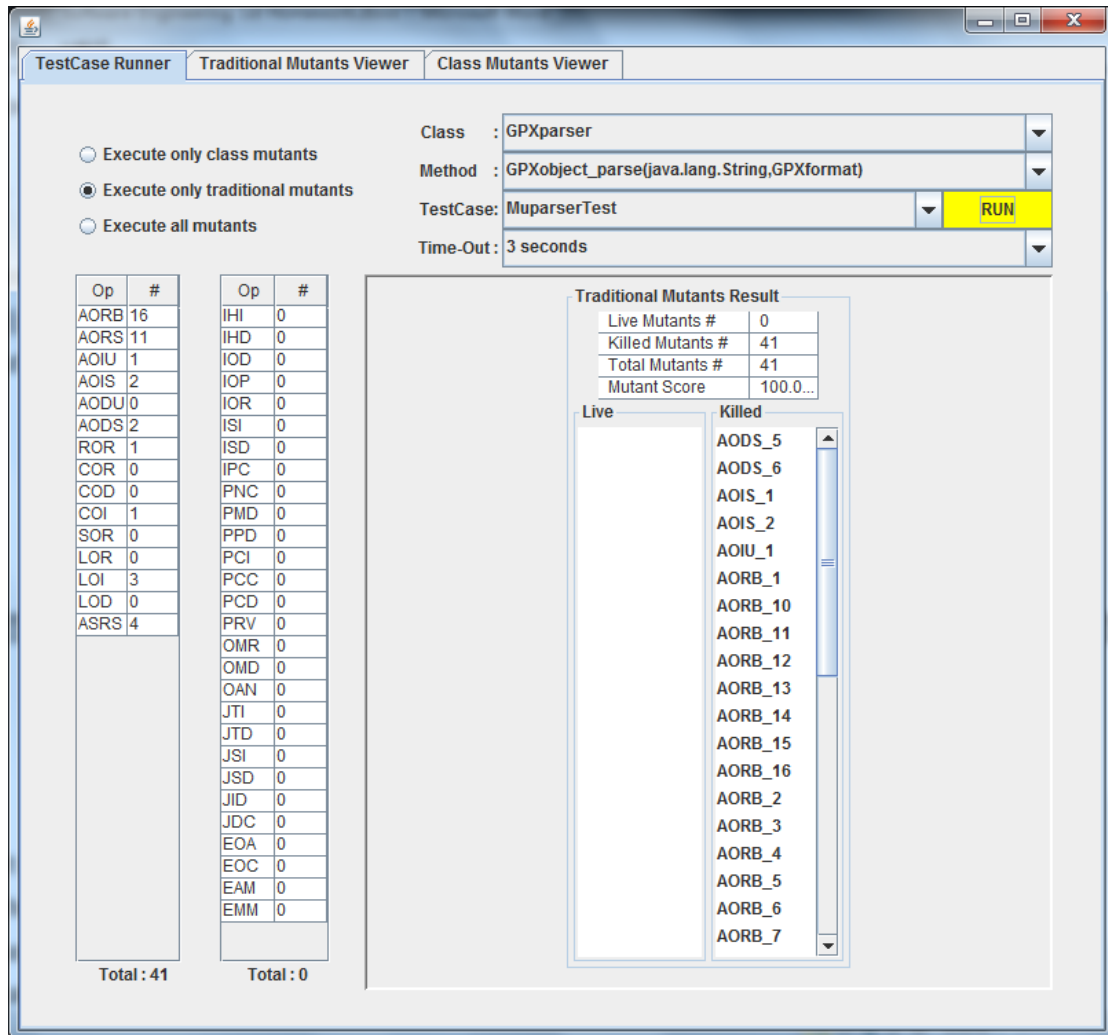
(5)**failure3:** test case 6 compare the original GPXObject.metadata().toString() (which is valid formatted) and the returned GPXObject.metadata().toString()(which is null since it's renewed in the end of the code), thus it results to a failure.

(6)**failure 4:** test case 7 contain an empty name attribute, since it should return null while the GPXchecker misunderstand it's a well-formatted GPXfile and let it construct the GPXObject, it also result in a failure.

4. Coverage

"As shown in Figure 1, our tests achieved **97% statement coverage and 100% branch coverage**. The only lines that have not been covered is line 170 and 171: an exception sentence. The reason is that there is no way to throw a exception for it to catch. We've tried the worst situation to randomly input an irregular non-existent filename, or void fillname, both of these situations don't render the method throw an exception.

5. Mutation Analysis



We performed mutation analysis and found that our test sets killed 100% of the 41 traditional mutants. (See Figure above)

GPXcalculator.calculateElapsedTime(Seg):

1. Test case generation:

Black-box testing: ---Equivalent classes:

(1)The valid input trksegment (here we only care about the time order is valid) should return the accurate time difference. (the returned result should be accurate.)

(2)The null input should return -1.

(3)The invalid input trksegment should return -1.

Black-box testing:---Boundary classes:

(1)The input trksegment has a descending time order. (The output should be -1)

(2)The input trksegment has a large time difference (such as one year period) between the first and the last point. (The output should be accurate)

(3)The input trksegment only has one trkpoint. (The output should be 0)

White-box testing:

The code doesn't consider the condition that the input is null, so we create a test case for it.

2. Adequacy Criteria

For this method, we created 6 test cases for each of the equivalence classes and boundary conditions, which achieved **100%** statement & branch coverage. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

3. Failure Discovered

There are 1 error and 4 failures for the 7 test case.

(1) **Error 1:** test case 1 inputs an empty trksegment, which should return -1. While this condition isn't defined in the code, it results to an error.

(2) **Failure 1 & 2:** test case 2 & 3 input a descending time-order trksegment, the difference is test 2 have 3 points and the middle point is earlier than the first one while test 3 have 2 points and the time difference is negative. These two cases we all expect to be -1, while the function doesn't, since it doesn't check the order time.

(3) **Failure 3:** test 4 have a large time difference which is one year. Since the type of result is "long", the time difference exceeds its length. Therefore it result to a failure.

(The function returns results with millisecond unit. It contradicts the specification which needs the results with second unit. Hence, rigidly speaking, the test will have no right output. But in the implementation of Junit, we omit this fault of the specification.)

4. Coverage

“As shown in Figure 1, our tests achieved **100% statement coverage and 100% branch coverage**.

5. Mutant Analysis

The screenshot shows the 'Traditional Mutants Viewer' window. It has three tabs: 'TestCase Runner', 'Traditional Mutants Viewer' (selected), and 'Class Mutants Viewer'. The window contains the following elements:

- Execution Options:**
 - ☐ Execute only class mutants
 - ☒ Execute only traditional mutants
 - ☐ Execute all mutants
- Configuration Fields:**
 - Class : GPXcalculator
 - Method : long_calculateElapsedTime(GPXtrkseg)
 - TestCase: MucalculateElapSegTest
 - Time-Out : 3 seconds
 - A yellow 'RUN' button is next to the TestCase field.
- Mutant Lists:**
 - Left List:** A table with columns 'Op' and '#'. It lists 19 mutants with their operators and counts. The total count is 19.

Op	#
AORB	8
AORS	0
AOIU	1
AOIS	8
AODU	0
AODS	0
ROR	0
COR	0
COD	0
COI	0
SOR	0
LOR	0
LOI	2
LOD	0
ASRS	0

Total : 19
 - Right List:** A table with columns 'Op' and '#'. It lists 19 mutants with their operators and counts. The total count is 0.

Op	#
IHI	0
IHD	0
IOD	0
IOP	0
IOR	0
ISI	0
ISD	0
IPC	0
PNC	0
PMD	0
PPD	0
PCI	0
PCC	0
PCD	0
PRV	0
OMR	0
OMD	0
OAN	0
JTI	0
JTD	0
JSI	0
JSD	0
JID	0
JDC	0
EOA	0
EOC	0
EAM	0
EMM	0

Total : 0
- Traditional Mutants Result:** A summary table showing:

Live Mutants #	4
Killed Mutants #	15
Total Mutants #	19
Mutant Score	78.0%

 - Live Mutants:** AOIS_17, AOIS_18, AOIS_21, AOIS_22
 - Killed Mutants:** AOIS_15, AOIS_16, AOIS_19, AOIS_20, AOIU_3, AORB_1, AORB_2, AORB_3, AORB_4, AORB_5, AORB_6, AORB_7, AORB_8, LOI_6, LOI_7

We performed mutation analysis and found that our test sets killed 78% of the 19 traditional mutants. (see Figure above).

GPXcalculator.calculateElapsedTime(Trk):

1. Test case generation:

Black-box testing:Equivalent classes:

(1)The valid (here we only care about the time order is valid) input trk should return the accurate time difference. (The returned result should be accurate.)

(2)The null input should return -1.

(3)The invalid input trk should return -1.

Black-box testing :Boundary classes:

(1)The input trk contains invalid trksegments which have a descending time order. (The output should be -1)

(2)The input trk contains trksegments which have large time difference (such as one year period) between the first and the last point. (The output should be accurate)

(3)The input trk only has one trkpoint. (The output should be 0)

White box Testing:

We notice that it uses the calculator.calculateElapsedTime(trkseg) function, we generate test cases which results in failures in that function.

2. Adequacy Criteria

For this method, we created 5 test cases for each of the equivalence classes and boundary conditions, which achieved 100% statement & branch coverage. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

3. Failure Discovered

There are 3 failures for the 6 test case.

(1)**Failure 1:** test case 1 input a trk whose trksegments are in descending order, since there are no sentences in calculator.calculateElapsedTime(trkseg) function to judge the time order, it causes the failure.

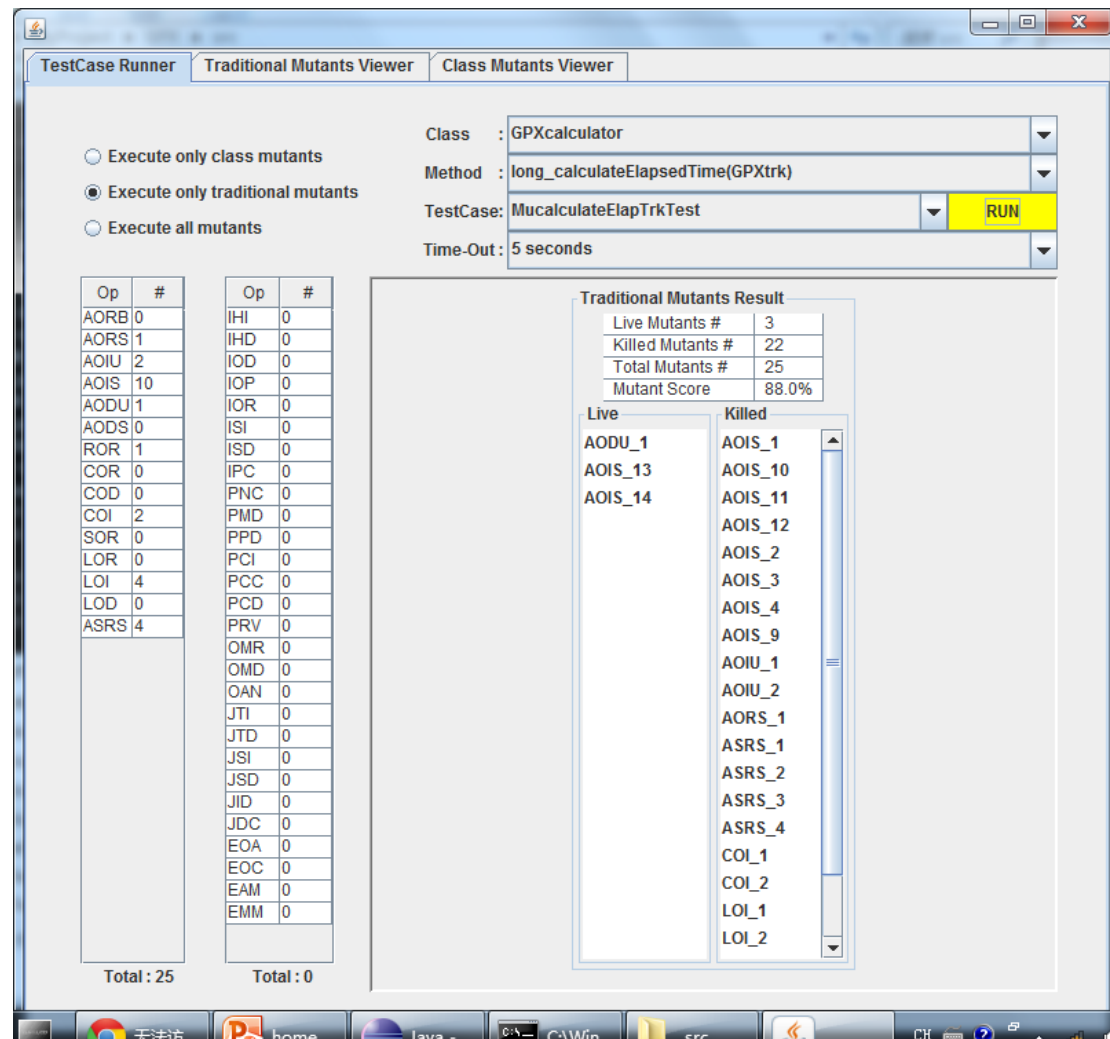
(2)**Failure 2:** test case 2 input a trk whose trksegments have large time difference. The fault is the same as the analysis in calculator.calculateElapsedTime(trkseg) section. It's the wrong type "long" resulting in the failure.

(The function returns results with millisecond unit. It contradicts the specification which needs the results with second unit. Hence, rigidly speaking, the test will have no right output. But in the implementation of Junit, we omit this fault of the specification.)

4. Coverage

As shown in Figure 1, our tests achieved **100% statement coverage** and **100% branch coverage**.

5 .Mutant Analysis



We performed mutation analysis and found that our test sets killed 88% of the 25 traditional mutants. (See Figure above)

GPXcalculator.calculateDistanceTraveled(Seg):

If the argument is a GPXtrk (track), return the total distance traveled for all track segments. *Note: latitude and longitude are in degrees, elevation is in meters.*

1. Test case generation:

Black-box testing: Equivalent classes:

(1)The valid input trksegment (here we only care about the lat, lon and ele are valid) should return the accurate distance (we regard the totally distance should be the sum of sub-segment distance between any two points). (The returned result should be accurate.)

(2)The null input should return -1.

(3)The input trksegment only has one point, which should return 0.

Boundary classes:

(1)The input trksegment has three trkpoints. (The output should be the sum of sub-segment distance between any two points)

(2)The input trksegment has the same lat, lon and ele attribute for every points.

White-box testing:

The code doesn't consider the condition that the input is null, so we create a test case for it.

2. Adequacy Criteria

For this method, we created 5 test cases for each of the equivalence classes and boundary conditions, which achieved **100% statement & branch coverage**. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

3. Failure Discovered

There is 1 error for the 5 test cases.

(1)**error1:** test case 1 inputs an empty trksegment, which represents the input is null. Since this condition isn't defined in the code, it results to an error.

(The function returns results with kilometer unit. It contradicts the specification which needs the results with meter unit. Hence, rigidly speaking, the test will have no right output. But in the implementation of Junit, we omit this fault of the specification.)

4. Coverage

As shown in Figure 1, our tests achieved **100% statement coverage and 100% branch coverage**.

5 Mutant Analysis

The screenshot shows the 'Traditional Mutants Viewer' window. It has three tabs: 'TestCase Runner', 'Traditional Mutants Viewer' (selected), and 'Class Mutants Viewer'. The window contains the following elements:

- Execution Options:** Three radio buttons: 'Execute only class mutants', 'Execute only traditional mutants' (selected), and 'Execute all mutants'.
- Configuration Fields:**
 - Class: GPXcalculator
 - Method: double_calculateDistanceTraveled(GPXtrkseg)
 - TestCase: MuGPXcalculateDistSegTest
 - Time-Out: 5 seconds
- Run Button:** A yellow button labeled 'RUN'.
- Mutant Lists:** Two tables on the left showing mutant counts for various operators (Op) and their frequencies (#).
- Summary Table:** A table titled 'Traditional Mutants Result' showing overall statistics.
- Mutant Lists:** Two scrollable lists on the right, 'Live' and 'Killed', showing specific mutant identifiers.

Mutant Count Tables:

Op	#
AORB	88
AORS	1
AOIU	12
AOIS	52
AO...	0
AODS	0
ROR	0
COR	0
COD	0
COI	1
SOR	0
LOR	0
LOI	9
LOD	0
ASRS	4
Total : 167	

Op	#
IHI	0
IHD	0
IOD	0
IOP	0
IOR	0
ISI	0
ISD	0
IPC	0
PNC	0
PMD	0
PPD	0
PCI	0
PCC	0
PCD	0
PRV	0
OMR	0
OMD	0
OAN	0
JTI	0
JTD	0
JSI	0
JSD	0
JID	0
JDC	0
EOA	0
EOC	0
EAM	0
EMM	0
Total : 0	

Traditional Mutants Result

Live Mutants #	20
Killed Mutants #	147
Total Mutants #	167
Mutant Score	88.0%

Live Mutants:

- AOIS_56
- AOIS_59
- AOIS_60
- AOIS_63
- AOIS_64
- AOIS_67
- AOIS_68
- AOIS_73
- AOIS_74
- AOIS_77
- AOIS_78
- AOIS_81
- AOIS_82
- AOIS_85
- AOIS_86
- AOIS_87
- AOIS_88
- AORB_65
- AORB_66

Killed Mutants:

- AOIS_23
- AOIS_24
- AOIS_25
- AOIS_26
- AOIS_31
- AOIS_32
- AOIS_33
- AOIS_34
- AOIS_35
- AOIS_36
- AOIS_45
- AOIS_46
- AOIS_47
- AOIS_48
- AOIS_49
- AOIS_50
- AOIS_51
- AOIS_52
- AOIS_53

We performed mutation analysis and found that our test sets killed 88% of the 167 traditional mutants. (See Figure above) The reason why we don't obtain a high kill-rate may be because the most of our Junit test cases aim to reveal faults which results in only small amount of test cases can pass.

GPXcalculator.calculateDistanceTraveled(Trk):

If the argument is a GPXtrk (track), return the total distance traveled for all track segments. *Note: latitude and longitude are in degrees, elevation is in meters.*

1. Test case generation:

Black-box testing: Equivalent classes:

(1)The valid input trk (here we only care about the lat, lon and ele are valid) should return the accurate distance (we regard the totally distance should be the sum of sub-segment distance between any two points). (the returned result should be accurate.)

(2)The null input should return -1.

(3)The input trk has only one point, which should return 0.

Boundary classes:

(1)The input trk has three trksegments of two trkppts each. (The output should be the sum of sub-segment distance between any two points)

(2)The input trk has two trksegments of one trkppts each. (The output should be 0)

(3)The input trk has only one trkppts.

White-box testing:

The code doesn't consider the condition that the input is null, so we create a test case for it.

2. Adequacy Criteria

For this method, we created 5 test cases for each of the equivalence classes and boundary conditions, which achieved **100% statement & branch coverage**. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

3. Failure Discovered

There is 1 error for the 4 test cases.

(1)**Error 1:** test case 1 inputs an empty trk, which represents the input is null. Since this condition isn't defined in the code, it results to an error.

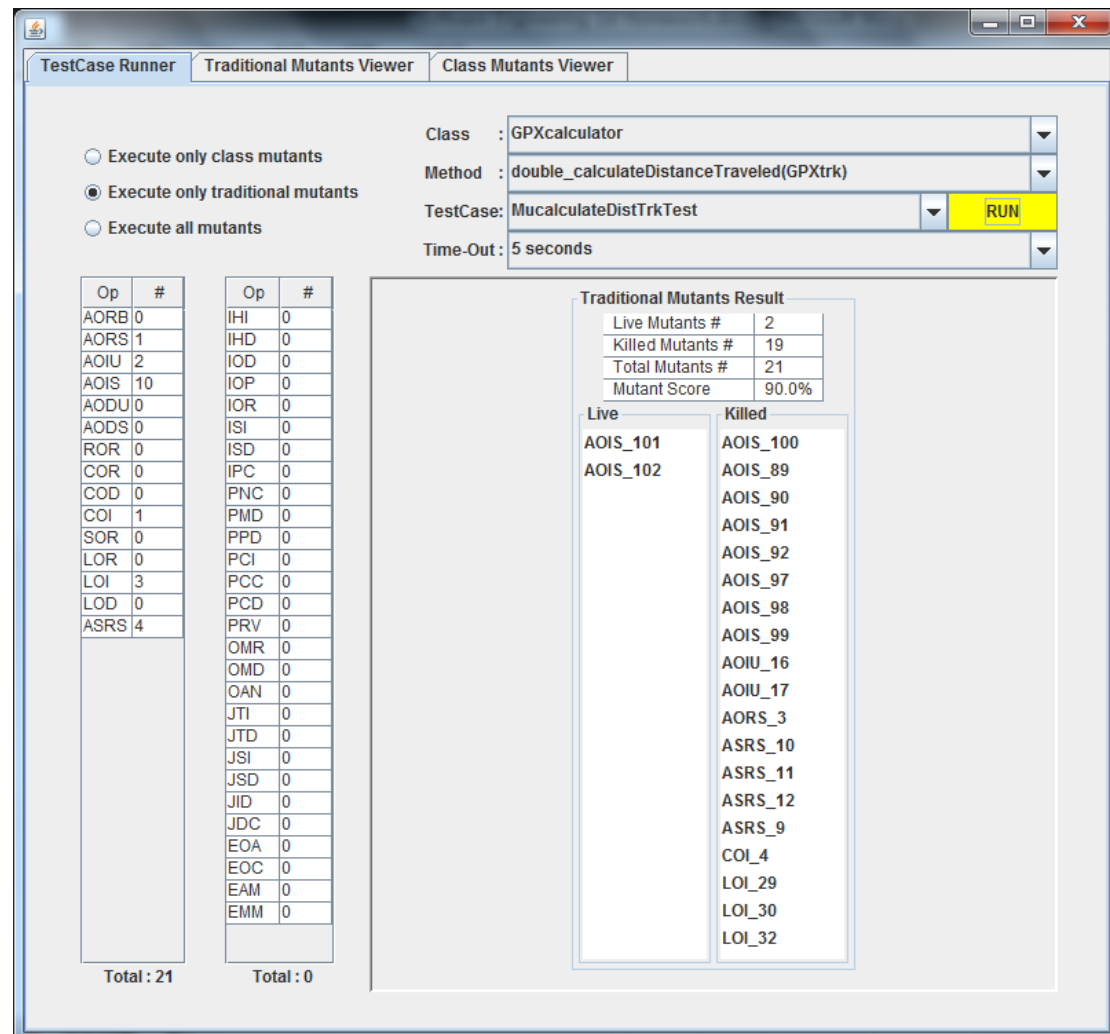
(The function returns results with kilometer unit. It contradicts the specification which needs the results with meter unit. Hence, rigidly speaking, the test will have no right output. But in the implementation of Junit, we omit this fault of the specification.)

4. Coverage

As shown in Figure 1, our tests achieved **100% statement coverage and 100% branch**

coverage.

5 Mutant Analysis



We performed mutation analysis and found that our test sets killed 90% of the 21 traditional mutants. (See Figure above)

GPXcalculator.calculateAverageSpeed:

compute the total distance traveled and total elapsed time, and return the average speed in meters per second

1. Test case generation:

Black-box testing---Equivalent classes:

- (1) The valid input trk (here we care about both distance attributes and time attributes) should return a correct average speed. (The output should be correct.)
- (2) The null input should return -1.
- (3) The input trk which only have one trkpts should return 0.
- (4) The invalid input trk should return -1.

Boundary conditions:

- (1) The input trk have have a large time difference in each two trksegments. (The output should be correct.)
- (2) The input trk have a descending time order, which should return -1. (The output should be -1.)

White-box testing:

The code doesn't consider the condition that the input is null, so we create a test case for it.

4. Adequacy Criteria

For this method, we created 5 test cases for each of the equivalence classes and boundary conditions, which achieved **100% statement & branch coverage**. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

5. Failure Discovered

There are 1 error and 3 failures of 5 test cases.

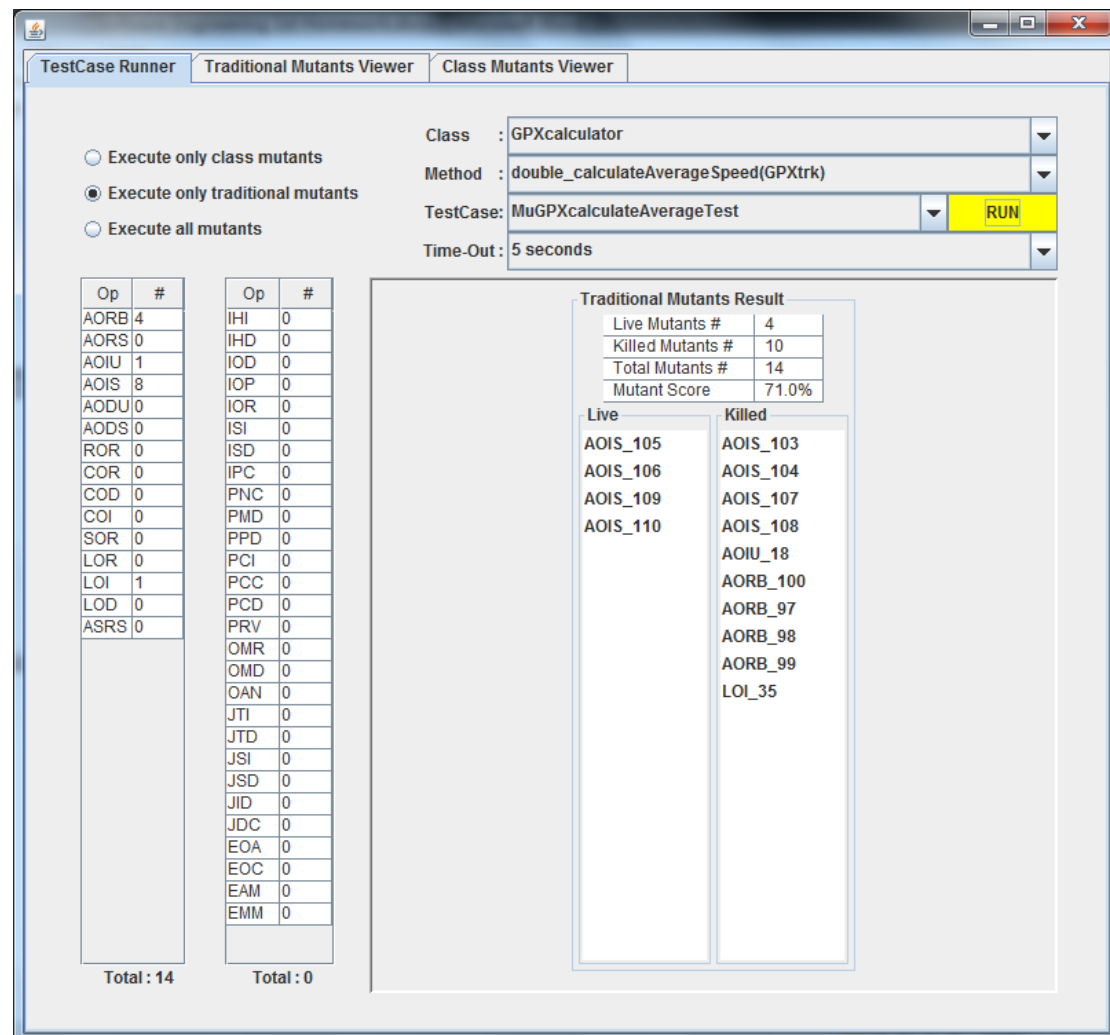
- (1) Error 1: test case 1 inputs an empty trk, which represents the input is null. Since this condition isn't defined in the code, it results to an error.
- (2) Failure 1: test case 2 inputs the trk which has a descending time order, since there GPXcalculator.calculateElapsedTime() won't check the time order of the input, it causes the failure.
- (3) Failure 2: test case 3 inputs the trk only has one trkpts, since the GPXcalculator.calculateElapsedTime() function returns 0, so the returned result is NaN while it's expected to return 0.
- (4) Failure 3: test case 4 inputs the trk which has long time difference in each trksegment,

since the results returned by `GPXcalculator.calculateElapsedTime()` function is wrong due to the wrong type “long”, the final result is also wrong.

4. Coverage

As shown in Figure 1, our tests achieved **100% statement coverage** and **100% branch coverage**.

5 Mutation Analysis



We performed mutation analysis and found that our test sets killed 71% of the 14 traditional mutants. (See Figure above) The reason why we don't obtain a high kill-rate may be because the most of our Junit test cases aim to reveal faults which results in only small amount of test cases can pass.

GPXcalculator.calculateBearing:

Determine the bearing (direction) from the first track point to the last track point, across all track segments. The return value should be between 0 and 360 degrees, with 90 degrees being due east, 180 degrees being due south, etc.

1. Test case generation:

Black-box testing---Equivalent classes:

(1)The valid input trk (here we care only distance attributes: lat, lon and ele) should return the correct bearing. (The output should be correct.)

(2)The null input should return -1.

(3)The input trk which only have one trkpts should return 0.

Black-box testing ---Boundary conditions:

(1) The input trk has 2 trksegments of one trkpts each.(the output should return 0.)

White box Testing:

Through checking the equations to calculate the bearing in one trk, we find the equation of the code is wrong which the "+" should be "-", therefore the result will always be wrong. We thus generate normal test cases which is able to reveal this fault.

2. Adequacy Criteria

For this method, we created 4 test cases for each of the equivalence classes and boundary conditions, which achieved **100% statement & branch coverage**. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

3. Failure Discovered

There is one failure and one error of the 4 test cases.

(1) **Error 1:** test case 1 inputs an empty trk, which represents the input is null. Since this condition isn't defined in the code, it results to an error.

(2) **Failure 1:** test case 2 input s a normal case which contains a valid trk. As mentioned above, since the equation of the code is wrong which the "...sin(lat2). + Math.sin..." should be "...sin(lat2). - Math.sin...", therefore the result will always be wrong.

4. Coverage

As shown in Figure 1, our tests achieved **100% statement coverage and 100% branch coverage**.

5 Mutation Analysis

The screenshot shows the 'Traditional Mutants Viewer' window. It displays configuration settings for a mutation analysis run on the 'GPXcalculator' class, specifically for the 'double_calculateBearing(GPXtrk)' method using the 'MuGPXcalculateBearingTest' test case. The 'Execute only traditional mutants' option is selected. The results show 154 total mutants, with 127 killed and 27 live, resulting in a mutant score of 82.0%.

Configuration:

- Class: GPXcalculator
- Method: double_calculateBearing(GPXtrk)
- TestCase: MuGPXcalculateBearingTest
- Time-Out: 5 seconds
- Execute only traditional mutants (selected)

Traditional Mutants Result:

Op	#	Op	#
AORB	96	IHI	0
AORS	0	IHD	0
AOIU	11	IOD	0
AOIS	42	IOP	0
AO...	0	IOR	0
AODS	0	ISI	0
ROR	0	ISD	0
COR	0	IPC	0
COD	0	PNC	0
COI	0	PMD	0
SOR	0	PPD	0
LOR	0	PCI	0
LOI	5	PCC	0
LOD	0	PCD	0
ASRS	0	PRV	0
		OMR	0
		OMD	0
		OAN	0
		JTI	0
		JTD	0
		JSI	0
		JSD	0
		JID	0
		JDC	0
		EOA	0
		EOC	0
		EAM	0
		EMM	0

Total : 154 Total : 0

Traditional Mutants Result Summary:

Live Mutants #	Killed Mutants #
27	127
Total Mutants #	154
Mutant Score	82.0%

Live Mutants List:

- AOIS_139
- AOIS_140
- AOIS_143
- AOIS_144
- AOIS_147
- AOIS_148
- AOIS_151
- AOIS_152
- AOIS_155
- AOIS_156
- AOIS_159
- AOIS_160
- AOIU_21
- AOIU_24
- AOIU_25
- AOIU_26
- AORB_103
- AORB_133
- AORB_134

Killed Mutants List:

- AOIS_119
- AOIS_120
- AOIS_121
- AOIS_122
- AOIS_123
- AOIS_124
- AOIS_125
- AOIS_126
- AOIS_127
- AOIS_128
- AOIS_129
- AOIS_130
- AOIS_131
- AOIS_132
- AOIS_133
- AOIS_134
- AOIS_135
- AOIS_136
- AOIS_137

We performed mutation analysis and found that our test sets killed 82% of the 154 traditional mutants. (See Figure above) The reason why we don't obtain a high kill-rate may be because the most of our Junit test cases aim to reveal faults which results in only small amount of test cases can pass.

GPXcalculator.calculateFastestSegment:

Return the 0-based index of the track segment with the fastest average speed.

1. Test case generation:

Black-box testing Equivalent classes:

(1) The valid input trk (here we care distance attributes: lat, lon and ele) should return the correct index of fastest trksegment. (The output should be as expected.)

(2) The null input should return -1.

(3) The input trk which only have one trkpts should return -1.

Boundary conditions:

(1) The input trk has 2 trksegments of one trkpts each. (The output should be -1.)

White box Testing:

(1) The code doesn't consider the condition that the input trk is null, we generate the test cases for it.

(2) The fastest segment speed in the code isn't given the right value while it just increase as the the "i" increases. We also generate two test cases for it.

2. Adequacy Criteria

For this method, we created 5 test cases for each of the equivalence classes and boundary conditions, which achieved **98% statement & branch coverage**. As we think it's adequate to reveal all the faults in this method, therefore we stopped here.

3. Failure Discovered

There are 1 error and 3 failures of the 5 test cases.

(1) **Error 1:** test case 1 inputs an empty trk, which represents the input is null. Since this condition isn't defined in the code, it results to an error.

(2) **Failure 2:** test case 2 inputs a trk which only contain one trkpts. It should return -1 while the result is 0. Since in the code this condition hasn't been taken into account.

(3) **Failure 3 & 4:** test case 3 & 4 are the normal cases which contain both enough trkpts and trksegments. Since the fastest segment speed in the code isn't given the right value while it just increase as the the "i" increases, the output result will always be wrong.

4. Coverage

As shown in Figure 1, our tests achieved **98% statement coverage and 100% branch coverage**. The only line that have not been covered is line 7, which is the class title of

GPXcalculator. We believe this is due to a bug in Ant build cobertura, because if we use eCobertura, the statement and branch coverage is 100%.

5 Mutation Analysis

The screenshot shows the 'Traditional Mutants Viewer' window. It displays configuration settings for a mutation analysis run on the 'GPXcalculator' class, specifically for the 'int_calculateFastestSegment(GPXtrk)' method using the 'MucalculateFastSegTest' test case. The 'Execute only traditional mutants' option is selected. The results show 40 total mutants, with 10 live and 30 killed, resulting in a 75.0% mutant score.

Configuration:

- Class: GPXcalculator
- Method: int_calculateFastestSegment(GPXtrk)
- TestCase: MucalculateFastSegTest
- Time-Out: 5 seconds
- Execution Options:
 - ☐ Execute only class mutants
 - ☒ Execute only traditional mutants
 - ☐ Execute all mutants

Mutants Summary:

Op	#	Op	#
AORB	4	IHI	0
AORS	1	IHD	0
AOIU	2	IOD	0
AOIS	20	IOP	0
AODU	0	IOR	0
AODS	0	ISI	0
ROR	5	ISD	0
COR	0	IPC	0
COD	0	PNC	0
COI	2	PMD	0
SOR	0	PPD	0
LOR	0	PCI	0
LOI	6	PCC	0
LOD	0	PCD	0
ASRS	0	PRV	0
		OMR	0
		OMD	0
		OAN	0
		JTI	0
		JTD	0
		JSI	0
		JSD	0
		JID	0
		JDC	0
		EOA	0
		EOC	0
		EAM	0
		EMM	0

Total : 40

Traditional Mutants Result:

Live Mutants #	Killed Mutants #
10	30
Total Mutants #	40
Mutant Score	75.0%

Live Mutants:

- AOIS_165
- AOIS_167
- AOIS_173
- AOIS_175
- AOIS_177
- AOIS_187
- AOIS_188
- AORB_197
- AORB_199
- ROR_2

Killed Mutants:

- AOIS_166
- AOIS_168
- AOIS_174
- AOIS_176
- AOIS_178
- AOIS_179
- AOIS_180
- AOIS_181
- AOIS_182
- AOIS_183
- AOIS_184
- AOIS_185
- AOIS_186
- AOIU_30
- AOIU_31
- AORB_198
- AORB_200
- AORS_4
- COI_5

Total : 0

We performed mutation analysis and found that our test sets killed 75% of the 40 traditional mutants. (See Figure above) The reason why we don't obtain a high kill-rate may be because the most of our Junit test cases aim to reveal faults which results in only small amount of test cases can pass.

GPXObject.toString:

/ render the GPXObject as a well-formed GPX file, containing all data. */*

1. Test case generation:

Black-box testing --- Equivalent classes:

- (1) When the input is an object manually created from the standard GPXformat posted by Mr. Murhphy (it should return a string equals to this GPXformat)
- (2) When there is no metadata of the object (Should also return a GPXformat)
- (3) When there is no trkseg of the object (should return true)
- (4) When there is no trkpt of the object (should return true)

Boundary conditions:

White-box testing:

- (1) We found that the method doesn't examine the time order of the points. Time-inversed points are certainly invalid.
 - (2) We also found that there is no constraint about the time order between segments. This flaw is also very crucial.
- So, we generate test case to justify them.

2. Adequacy Criteria

For this method, we created 5 test cases for the equivalence classes and boundary conditions, and 2 test case for the white-box approach to reach each branch, for a total of 7 test cases, which achieved **100% statement coverage and 87% Branch coverage**. Since we felt that we had found all the equivalence classes and have illustrated the only branch that have not been covered, we stopped here.

3. Failure Discovered

4 of the 26 test cases failed.

(1)**Failure1:** We generate a test case without segments. It should return a well-formed GPXformat. But instead, it throw an NullPointerException. We use try-catch sentences to catch this exception, so it becomes a failure. The null pointer problem is because the internal flaws in the definition of class GPXtrgseg. Specifically, GPXtrkseg.trkseg().

(2)**Failure2:** We generate a test case without track points. It should return GPXformat, but instead it also throw an NullPointerException. This is because the incomplete definition of class trgseg. These two problems are rather crucial because the object without trgs and trgs are valid according to specification while it is impossible to create an object using this method.

(3)**Failure3:** If we generate a test case with 2 trkpts, whose time sequence is inverse. This cannot occur in our daily life, thus should return null. But, for this method, it instead returns

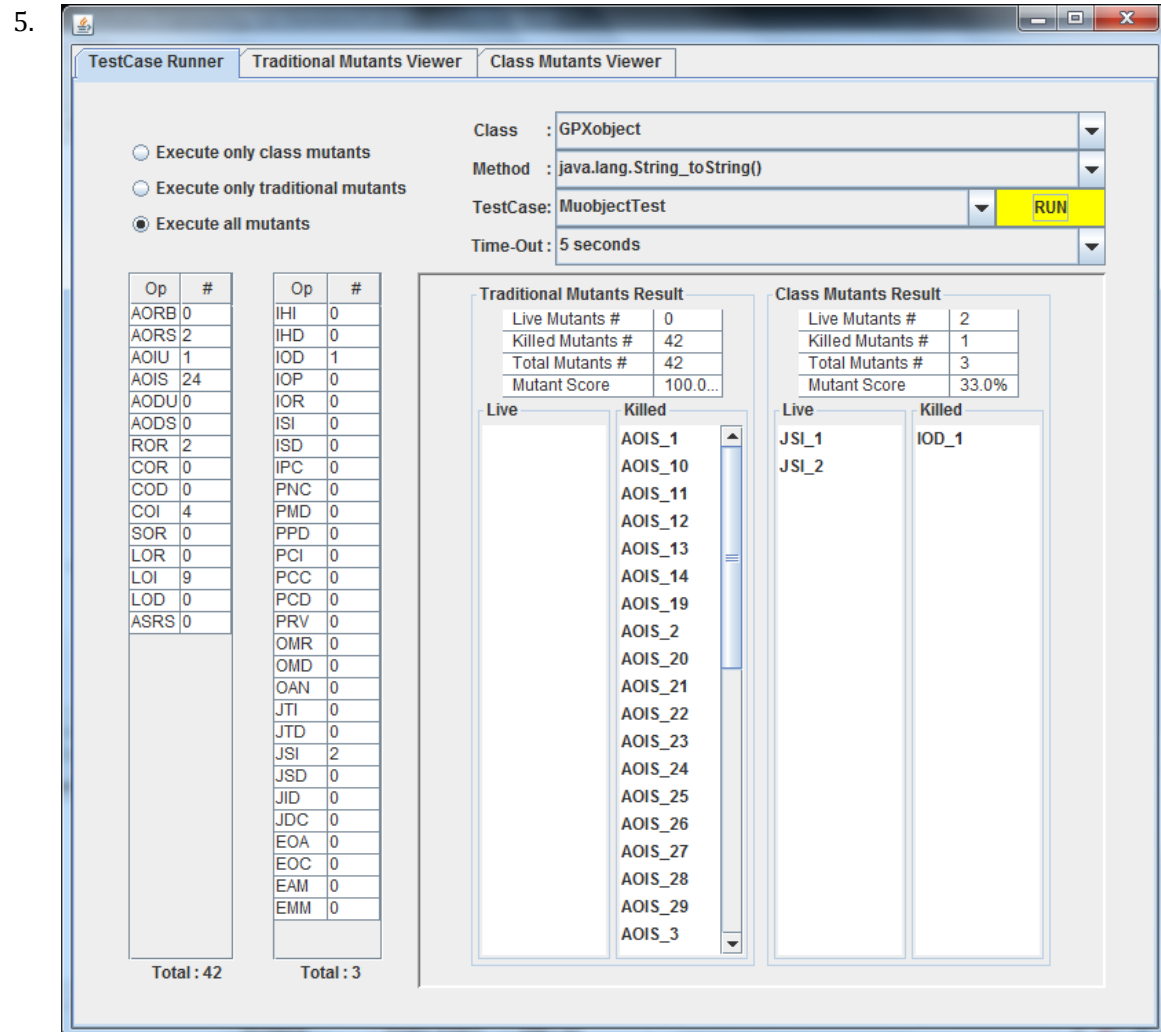
an object regardless of this question. This is a fatal problem because when the method `GPXparser.parse()` is calling the method `object.toString()`, the error would be transmitted.

(4)**Failure4:** If we generate a test case with 2 trksegs, whose general time sequence is also inverse. This cannot happen in our daily life, thus should return null. But, for this method, it instead returns an object regardless of this question. This is a fatal problem because when the method `GPXparser.parse()` is calling the method `object.toString()`, the error would also be transmitted.

(5)**Failure5:** If both without metadata and trksegs, for the test case5, the method should a well-formed GPXformat. However, it throws a `NullPointerException` error. (We use try-catch sentences to transformed it into a failure.) This has the similar reason as Failure 1 and 2.

4. Coverage

“As shown in Figure 1, our tests achieved **100% statement coverage and 87% Branch coverage**. The only code not covered was lines 52 for the method of `GPXObject.toString()`. The reason is that this line of code is certainly unreachable: in order to reach that code, the array `trksegs[]` on line 50 would have been null of with elements; however, when it is executing `_trk.trksegs()`, it throws a `NullPointerException` before the branch arriving at line 52 could be executed.



We performed mutation analysis and found that our test sets killed 100% of the 42 traditional mutants and 33% of 3 class mutants. (See Figure above)

GPXtime.createGPXtime:

given a string in the format "YYYY-MM-DDThh:mm:ssZ", parse it and create a GPXtime object with the corresponding fields populated. Return null if the string is not well-formed or if an error occurs.

1. Test case generation:

Black-box testing --- Equivalent classes:

- (1) Given any day in standard format "YYYY-MM-DDThh:mm:ssZ" (should return an object)
- (2) If the attributes of year, month, hour and minute is apparently impractical (should return null. Like there is more than 29 days of February.)
- (3) If input equals to null (should return null too)

Boundary conditions:

- (1) If the data equals to Java start year. (should return 0)

White-box Testing:

In order to examine the justifying sentence on the line of 58, we generate 11 test case to test each "or" condition.

2. Adequacy Criteria

For this method, we created 4 test cases for equivalence classes and boundary conditions, and 11 test case for the white-box approach to reach each branch, for a total of 15 test cases, which achieved **100%** statement coverage and **100%** branch coverage. Since we felt that we had tried our best to find all the equivalence classes with full branch and statement coverage, we stopped here.

3. Failure Discovered

1 of the 15 test cases failed.

Failure1: In this test case testcreateGPXtime3(), there are 30 days for February. The method should consider it invalid and return null but instead returns a GPXtime object. This is because the method doesn't take the special days of February into consideration. This failure is rather critical, because it will impose a great thread on lots of methods from other classes, such as GPXCalculatorElapsedTime, GPXparser.parse and GPXObject.toString().

4. Coverage

"As shown in Figure 1, our tests achieved **100%** statement coverage and **100%** branch coverage. That is, we've attained full coverage.

5. Mutation Analysis

Test Case Runner | **Traditional Mutants Viewer** | **Class Mutants Viewer**

☐ Execute only class mutants
☐ Execute only traditional mutants
☒ Execute all mutants

Class : GPXtime
 Method : GPXtime_createGPXtime(java.lang.String)
 Test Case: MuCreatetimeTest **RUN**
 Time-Out : 5 seconds

Op	#	Op	#
AORB	0	IHI	0
AORS	0	IHD	0
AOIU	6	IOD	1
AOIS	56	IOP	0
AO...	0	IOR	0
AODS	0	ISI	0
ROR	25	ISD	0
COR	20	IPC	0
COD	0	PNC	0
COI	21	PMD	0
SOR	0	PPD	0
LOR	0	PCI	0
LOI	17	PCC	0
LOD	0	PCD	0
ASRS	0	PRV	0
		OMR	0
		OMD	0
		OAN	0
		JTI	0
		JTD	0
		JSI	7
		JSD	0
		JID	0
		JDC	0
		EOA	0
		EOC	0
		EAM	0
		EMM	0
Total : 145		Total : 8	

Traditional Mutants Result

Live Mutants #	27
Killed Mutants #	118
Total Mutants #	145
Mutant Score	81.0%

Live	Killed
AOIS_81	AOIS_37
AOIS_82	AOIS_38
AOIS_83	AOIS_39
AOIS_84	AOIS_40
AOIS_85	AOIS_41
AOIS_86	AOIS_42
AOIS_87	AOIS_43
AOIS_88	AOIS_44
AOIS_89	AOIS_45
AOIS_90	AOIS_46
AOIS_91	AOIS_47
AOIS_92	AOIS_48
COR_10	AOIS_49
COR_12	AOIS_50
COR_14	AOIS_51
COR_16	AOIS_52
COR_2	AOIS_53
COR_4	AOIS_54
COR_6	AOIS_55

Class Mutants Result

Live Mutants #	7
Killed Mutants #	1
Total Mutants #	8
Mutant Score	12.0%

Live	Killed
JSI_1	IOD_1
JSI_2	
JSI_3	
JSI_4	
JSI_5	
JSI_6	
JSI_7	

We performed mutation analysis and found that our test sets killed 81% of the 145 traditional mutants and 12% of 8 class mutants. (See Figure above) The reason why we don't obtain a high kill-rate may be because the most of our Junit test cases aim to reveal faults which results in only small amount of test cases can pass.

GPXtime.convertToJavaTime

Given a GPXtime object, return a long that represents it in "Java time", i.e. milliseconds since midnight on January 1, 1970.

1. Test case generation:

Black-box testing --- Equivalent classes:

- (1) When the input is a standard year (should return the difference in milliseconds)
- (2) When input is before Java time (should return null)
- (3) When the date is nonsense, like 30 days for February (should return null)
- (4) When input year contains n leap years (results should add n more day)
- (5) When input is null, should return null.

Black-box testing --- Boundary conditions:

- (1) If the input is a standard Java time. (Should return 0)

White-box testing:

- (1) On the line of 96, it is apparent that the boundary of index for month is wrong.
 - (2) On the line of 101, it is apparent that the counting day should firstly minus 1.
- So, we generate two test cases to justify them.

2. Adequacy Criteria

For this method, we created 5 test cases for equivalence classes and boundary conditions, and 2 test case for the white-box approach to reach each branch, for a total of 7 test cases, which achieved **100% statement coverage and 98% line coverage**. Since we felt that we had fought all the equivalence classes with high branch and statement coverage, we stopped here.

3. Failure Discovered

5 of the 7 test cases failed.

- (1) **Failure1:** If we input a date that is one day later, it should have simply returned $24 \times 60 \times 60 \times 1000$. But instead, it returns another value. This is because that it forgets to minus the Java day by 1.
- (2) **Failure2:** If the input day is after at least one leap year such as 1972, it should add one more day. But instead it returns another value, which is greatly different with expected value. This is because that it wrongly calculate the total month with respect to January, and wrongly calculate the days with respect to the first day.
- (3) **Failure3:** If the input day is after at least one leap year such as 2000 (alternative form of leap year), it should add one more day for each leap year. But instead it returns a totally

different value. Because the wrong counting of months and days difference.

(4)**Failure4:** The forth test case is simple. We input the same day as Java time. The return value should certainly be zero, but instead, it returns a very large number. This is because on the code line of 101,

(5)**Failure5:** The fifth test case we generate is nonsense data. This is actually related to the GPXtime.createGPXtime() method, which does not take (>29 days of February) into consideration. It should return null, but instead, returns a real number. This is also fatal because this flaw will certainly to other methods in class GPXcalculator.java.

4. Coverage

“As shown in Figure 1, our tests achieved **100% statement coverage and 98% branch coverage**. The only line of code not covered was lines 8 “public class GPXcalculator”. This is really annoying, because this line is actually not a sentence. The probably reason behind this is this method have never called itself, so cobertura consider it uncovered.

5. Mutation Analysis

The screenshot shows the TestCase Runner application with the following configuration:

- Class:** GPXtime
- Method:** long_convertToJavaTime()
- TestCase:** MuConverttimeTest
- Time-Out:** 5 seconds
- Execution Options:** Execute all mutants (selected)

The application displays two main result panels:

Traditional Mutants Result

Live Mutants #	46
Killed Mutants #	194
Total Mutants #	240
Mutant Score	80.0%

Live Mutants List:

- AOIS_107
- AOIS_109
- AOIS_111
- AOIS_131
- AOIS_132
- AOIS_135
- AOIS_136
- AOIS_139
- AOIS_140
- AOIS_143
- AOIS_144
- AOIS_145
- AOIS_146
- AOIU_22
- AOIU_23
- AOIU_24
- AORB_101
- AORB_102
- AORB_29

Killed Mutants List:

- AOIS_100
- AOIS_101
- AOIS_102
- AOIS_103
- AOIS_104
- AOIS_105
- AOIS_106
- AOIS_108
- AOIS_110
- AOIS_112
- AOIS_113
- AOIS_114
- AOIS_115
- AOIS_116
- AOIS_117
- AOIS_118
- AOIS_119
- AOIS_120
- AOIS_121

Class Mutants Result

Live Mutants #	8
Killed Mutants #	0
Total Mutants #	8
Mutant Score	0.0%

Live Mutants List:

- IOD_1
- JSI_1
- JSI_2
- JSI_3
- JSI_4
- JSI_5
- JSI_6
- JSI_7

Summary:

- Traditional Mutants:** Total: 240
- Class Mutants:** Total: 8

We performed mutation analysis and found that our test sets killed 80% of the 240 traditional mutants and 0% of 8 class mutants. (See Figure above) The reason why we don't obtain a high kill-rate may be because the most of our Junit test cases aim to reveal faults which results in only small amount of test cases can pass.