Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Software Testing - Project

Target

Classless Inter-Domain Routing Calculator (CIDR Calculator)
https://f-droid.org/packages/us.lindanrandy.cidrcalculator/

Part 1 Software Description

System Description:

Q1. What does the app do?

Ans: CIDR Calculator is a simple IP subnet calculator for network engineers to quickly determine what the address range is of a subnet. Users need to give an IP address (IPv4) first, and then this app will provide the following information:

a) Address range
b) Maximum address
c) Wildcard
d) IP address in binary/Hex formats
e) IP Netmask in binary format

All the history records that users have inputted are maintained for easy reuse. Moreover, a basic IPv6 address calculator is included, which provides the following information:

a) Address range
b) Maximum addresses
c) Extra information about what type of address was given, such as “Mapped IPv4”.

Q2. What are the inputs?

Ans: The main input is the IP address, and users can choose different subnet masks.

Q3. What are the outputs?

Ans: The outputs are the information including address range, maximum address, wildcard, IP address and IP Netmask address in binary format.

Q4. What non-functional characteristics does your app have?

Ans: Non-functional characteristics of this app are the execution performance and reliability.

Q5. What kind of faults could your app have?

Ans: This app could crash due to quick and random inputs, which are relevant to the execution performance and reliability.

Q6. What are the highest risks for the app?

Ans: This app might not pass the monkey testing.

Unit Description:

Q1. What does the unit do?

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Ans: The first class is the “Converter”, which converts the IP address from decimal to binary and hex formats. The second class is the “InetAddress”, which evaluates whether the given address is an “IPv4 mapped” IPv6 address. The third class is the “CIDRCalculator”, which controls the main calculator activity. I chose public functions that are not related to Android behavioral functions (OnCreate, OnResume etc.) and private functions that only focus on the public functions for unit testing.

Q2. What are the inputs and outputs?

Ans: The general inputs are IP addresses including IPv4 and IPv6 in several formats.

Q3. What kind of faults can the unit have?

Ans: It could fail during the bit calculation or the determination of valid and invalid IP addresses.

Part 2 Functional Unit Testing

Approach for Functional Unit Testing:

Q1. Describe how you tested the unit. In particular, you should describe what functional test techniques you used and how they were applied.

Ans: First, the project that I selected was an open-source Android app, and I chose Android Studio to implement the unit tests. Before I started to develop the test cases, JUnit4 needed to be included and integrated into the “dependencies” setting of “build.gradle”. In the unit testing, I used invalid input and boundary testing techniques to generate my test cases. My implementations of test cases are shown in CIDRCalculatorUnitTesting.java.

Functional Test Cases:

Q1. A table of test cases. Each test case should have the following:

- A unique identifier (a number is sufficient).
- A brief description of the test case that describes what test case specification is being satisfied.
- The inputs used.
- The expected output.
- The status of the test: not executed, passed, failed

Ans:

The attached file, Chun-Chi_UnitTesing.xlsx, shows the test results and detailed information of the test cases. In “Unit Testing” sheet, it shows the test results of the unit testing for three classes, CIDRCalculator, Converter, and InetAddresses. In total, there are 29 test cases with a pass rate of 26/29. In the “Test Coverage (All classes)” sheet, it shows the class coverage, method coverage, and line coverage for every class. For example, InetAddress class has one class, 12 methods, and 130 lines. After proceeding with my test suite, the test suite covers 100% of class, 83.3% of methods, and 73.1% of lines.

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Functional Unit Testing Results:

Q1. If necessary, an explanation on why certain tests were not executed.

Ans: In my test cases, all the unit tests could be executed, but some tests failed because some particular functions needed to check the input data and make sure that the data was valid.

Q2. A list of bugs detected. It is not necessary to debug the failures.

Ans: Even though some unit tests failed due to the lack of the judgement. However, these failures have not caused any bugs until now. Currently, as to the test coverage, the following table shows the percentages of the overall and individual classes. The classes marked with the yellow color were chosen to develop the test cases for unit testing. The more detailed information is listed in Chun-Chi_UnitTesting_TestCoverage.zip.

Coverage Summary for Package: us.lindanrandy.cidrcalculator			
Package	**Class, %**	**Method, %**	**Line, %**
us.lindanrandy.cidrcalculator	7.1% (3/ 42)	10.5% (15/ 143)	12.5% (138/ 1106)
Class	Class, %	Method, %	Line, %
R	0% (0/ 8)	0% (0/ 8)	0% (0/ 8)
Preferences	0% (0/ 2)	0% (0/ 5)	0% (0/ 18)
NotifySubnet	0% (0/ 1)	0% (0/ 2)	0% (0/ 22)
InetAddresses	100% (1/ 1)	83.3% (10/ 12)	73.1% (95/ 130)
IPv6Calculator	0% (0/ 5)	0% (0/ 18)	0% (0/ 160)
HistoryProvider	0% (0/ 2)	0% (0/ 13)	0% (0/ 132)
HistoryList	0% (0/ 3)	0% (0/ 14)	0% (0/ 69)
CustomKeyboard	0% (0/ 5)	0% (0/ 16)	0% (0/ 70)
Converter	25% (1/ 4)	18.8% (3/ 16)	15% (25/ 167)
CIDRHistory	0% (0/ 2)	0% (0/ 3)	0% (0/ 3)
CIDRCalculator	12.5% (1/ 8)	5.9% (2/ 34)	5.5% (18/ 325)
BuildConfig	0% (0/ 1)	0% (0/ 2)	0% (0/ 2)

Part 3 Structural Unit Testing and Integration Testing

Structural Unit Testing Results:

Q1. How much statement coverage was obtained (before writing structural tests).

Ans: After executing the unit testing, the statement coverage was 95/130 (73.1%) for th "InetAddresses" class. This value was calculated by the code coverage tool of Android Studio.

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Q2. A description of what was missed during functional testing. If there were deficiencies during functional testing, describe the deficiencies and make the appropriate modifications to the unit functional testing sections in this document.

Ans: For the missing statements of my unit testing suite, those items are all relevant to the invalid inputs. Because I do not create all the test cases including all the conditions which are “return null” and “exception” cases in the beginning of unit testing. An example is shown as follows. In line 106, this “return null” is a missing statement which I do not include a “address.length != IPV4_PART_COUNT” condition. For lines 114 and 115, I do not create a situation which makes “parseOctet” throw a number-format exception.

```
102     private static final int IPV4_PART_COUNT = 4;
103 @       private static byte[] textToNumericFormatV4(String ipString) {
104             String[] address = ipString.split( regularExpression: "\\.", limit: IPV4_PART_COUNT + 1);
105             if (address.length != IPV4_PART_COUNT) {
106                 return null;
107             }
108
109             byte[] bytes = new byte[IPV4_PART_COUNT];
110             try {
111                 for (int i = 0; i < bytes.length; i++) {
112                     bytes[i] = parseOctet(address[i]);
113                 }
114             } catch (NumberFormatException ex) {
115                 return null;
116             }
117             return bytes;
118         }
```

As to these failures, these failed cases should throw exceptions due to the incorrect format of inputs, but they do not. The following unit-testing and function code demonstrate one of these failed cases. When the address of the given input IP includes a negative value, this value should be detected, and this function “stringIPtoInt” should throw an exception. However, this function only checks whether the length of this address is equal to four, whether the length of every single value is greater than 1, whether the value of each sub-string can be parsed to an integer, and whether the value of each sub-string is less than 255. The function “stringIPtoInt” does not check whether the value of each sub-string is a positive value or not. If not, “stringIPtoInt” should throw an exception.

```
67      @Test (expected = Exception.class)
68      public void stringIPtoInt_192_168_1_N1() throws Exception {
69          //arrange
70          String inputIP = "192.168.1.-1";
71          //action
72          int actualOutput = CIDRCalculator.stringIPtoInt(inputIP);
73          //assert
74          // use ""(expected = Exception.class) to catch an exception
75      }
```

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

```
447 @   public static int stringIPtoInt(String ip) throws Exception
448     {
449         String[] quad = ip.split( regularExpression: "\\.", limit: 4);
450         if (quad.length != 4)
451         {
452             throw new Exception();
453         }
454         int ip32bit = 0;
455         for (String value : quad) {
456             if (value.length() < 1) {
457                 throw new Exception();
458             }
459             int octet;
460             try {
461                 octet = Integer.parseInt(value);
462             } catch (NumberFormatException e) {
463                 throw new Exception();
464             }
465
466             if (octet > 255) {
467                 throw new Exception();
468             }
469             ip32bit = ip32bit << 8;
470             ip32bit = ip32bit | octet;
471         }
472         return ip32bit;
473     }
```

Q3. A final measure of statement coverage. If less than 100%, explain why.

Ans: After adding 15 test cases for those invalid input conditions, the statement coverage becomes 100% (129/129) shown in the test coverage report, Chun-Chi_StruturalTesting_TestCoverage.zip.

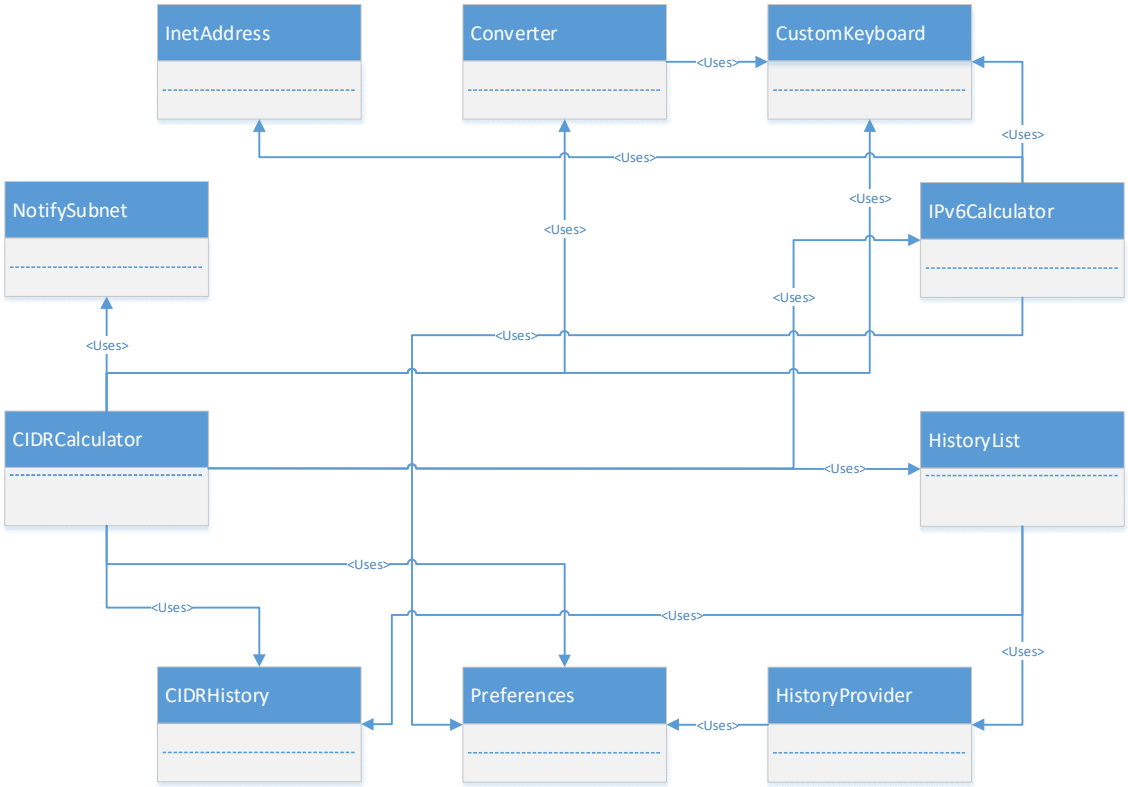
Integration Testing:

Q1. Briefly describe what each unit it interacts with.

Q2. How do the modules interact? Do they call each other's functions? What data do they exchange?

Ans: The following diagram shows the interactions between these units. The "CIDRCalculator" class is the main unit, which uses seven other classes. The unit that I focused on, "InetAddress", is only used by the "IPv6Calculator" class. As to the data exchange between "InetAddress" and "IPv6Calculator", "IPv6Calculator" feeds the IP address to "InetAddress" and receives the Byte or String format of the given IP address from "InetAddress".

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee



Q3. What kind of faults can there be in the integration of those modules?

Ans: In my observation, there can be inadequate error processing in the integration of these modules. Due to the missing consideration of invalid inputs, these modules cannot properly deal with this invalid information causing them to display the incorrect data. For example, when users enter an IP address including a negative value, these modules should identify this error and show some indications such as “Invalid IP address”.

Q4. Describe some specific integration tests that can be carried out. (It is not necessary to execute these tests).

Ans: I chose two combined modules as my target modules for the integration tests. The first combined modules are “CIDRCalculator” and “Converter”. “CIDRCalculator” is the main module to deal with the IPv4 address and to render the calculation results, and “Converter” includes two functions which convert the IP address into binary or hex format strings. I created several test cases, such as correct inputs, invalid inputs, or boundary inputs to test these two modules. The second combined modules are “IPv6Calculator” and “InetAddress”. “IPv6Calculator” is similar to “CIDRCalculator”, but it aims at the IPv6 address. “InetAddress” provides several functions for “IPv6Calculator” to deal with the IPv6 address. The ways to test the first combined modules can be applied to the second combined modules.

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Part 4 System Testing and Conclusions

Exploratory System Testing:

Q1. Mention the (possibly emulated) device you carried out exploratory testing. Please note whether the device is a phone or tablet and whether you carried out the testing on the emulator or the actual device.

Ans: The target device is an Android phone, and it is Sony Xperia Z5 Premium which contains 5.5" panel with Android 7.1.1 version.

Q2. Then, for each "tour", describe the following:

A) How you carried out the tour. For the Back Alley and Lonely Businessman tours, the descriptions should indicate the features you tested. For the Antisocial tour, the description should describe the bad inputs and where they were applied.

Ans:

a) Landmark tour: Select key features to test.

i. Input an IPv4 address and then click the "CALCULATE" button to get address range, maximum addresses, wildcard, IP Binary, and IP Binary Netmask.
ii. Based on the first case, select different CIDR Netmasks in a given IPv4 address.
iii. Based on the first case, select different Netmasks in a given IP Netmask.
iv. After the first case, click the "RESET" button and then check the display.
v. In "History" page to click one of these records listed in History pasge.
vi. In "Converter" page, click the "CONVERT TO BINARY/HEX" button.
vii. In "Converter" page, click the "CONVERT TO DECIMAL/HEX" button.
viii. In "Converter" page, click the "CONVERT TO DECOMAL/BINARY" button.
ix. In "IPv6" page, Input an IPv6 address to calculate address range, maximum addresses, and info.
x. Based on the previous step, select different CIDR Netmasks in a given IPv6 address.
xi. After the previous case, click the "RESET" button and then check the display.

b) FedEx tour: Aim at the data flowing through this app.

i. Input an IPv4 address in "IPv4" page and then click the "CALCULATE" button. Go to "Converter" page. The input IPv4 address should be kept in "Converter" page.
ii. Based on previous step, select "Return IP". The input IPv4 address should be kept in "IPv4" page.

c) Back Alley tour: Focus on the rarely used feature.

i. In "Preference" page, enable "Auto Complete" feature to make sure that this app will automatically calculate the IP address selected in the "History" page.

ii. In “Preference” Page, select “Input Keyboard”, and then go to “IPv4”, “Convertor”, and “IPv6” pages to check whether the keyboard is modified according to the previous selection.

d) Lonely Businessman tour: Find the features which are furthest away from the starting point of this app.
   i. Go to “Preference” page and select 10 as the history entries in “History Entries” option. Continue inputting and calculating more than 10 records in “IPv4” page. Then go to “History” page and check the number of these records.

e) Garbage Collection tour: Check the display of every column.
   i. Input an IPv4 address with a blank value in “IPv4” page and then see the UI display.
   ii. Input an IPv4 address with a blank value in “Converter” page and then see the UI display.
   iii. Input an IPv4 address with a blank value in “IPv6” page and then see the UI display.

f) Supermodel tour: Check whether UI display of this app is appropriate.
   i. Check whether the UI display of every page is good.
   ii. Check whether the representations of the calculated results are proper.
   iii. Check whether the performance of the calculation

g) Couch Potato tour: Do as little as work as possible.
   i. Close this app entirely first and then launch again this app, check whether there is any default value.
   ii. In “IPv4” page, input an invalid IPv4 address and then check whether the result is meaningful.
   iii. In “IPv6” page, input a valid IPv4 mapped address and then check whether the result is meaningful.

h) Antisocial tour: Enter the bad inputs to test.
   i. In “IPv4” page, input an IPv4 address with a negative value and then check the result.
   ii. In “IPv4” page, input an IPv4 address with a punctuation mark and then check the result.
   iii. In “IPv6” page, input an IPv6 address with a negative value and then check the result.
   iv. In “IPv6” page, input an IPv4 address with a punctuation mark and then check the result.

i) After-Hours tour: Check whether the data can be kept.

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

i. Input an IPv4 address and then click the “CALCULATE” button to get the results. Then power off the phone for 1 hour. Come back to check the result.
ii. Input an IPv4 address and then click the “CALCULATE” button to get the results. Then put this app in the background for 1 hour. Come back to check the result.

B) How long it took to carry out the tour.

Ans: It took around 4 hours.

C) Any bugs found during the tour.

Ans: There were two bugs found. The first issue was if users enter an IPv4 address with negative value, this app would show the message “Bad IP format”. However, this app still does the calculations and represents the wrong results. The second issue was if users enter a valid IPv6 address and do calculations and then enter an invalid address, not all of the results are updated.

D) Any additional test cases that came to your head while applying the tour (if appropriate).

Ans: Monkey test can be introduced into this app. I will carry out it later.

Scenario Testing:

Q1. Describe the three tests.

Ans: There are three scenarios shown as follows.

a) IPv4 Calculation
   i. Enter a valid IPv4 address, and then do calculation, followed by checking the results.
   ii. Enter an invalid IPv4 address, and then do calculation, followed by checking the results.
   iii. Enter a valid IPv4 address, and then do calculation with selecting a different IPv4 mask, followed by checking the results.

b) Converter
   i. Enter a valid IPv4 address, and the do calculation.
   ii. Send MENU key, and then select “Converter”.
   iii. Check whether the IP address of Converter is the same with the previous input address, and then check other results.

c) IPv6 Calculation
   i. Send MENU key, and then select “IPv6”.
   ii. Enter a valid IPv6 address, do calculation, and then check the results.
   iii. Select a different IPv6 mask, do calculation, and then check the results.

All the implementations are present in CIDRCalculatorScenarioTesting.java.

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Q2. Note if you found any bugs with the tests.

Ans: In this stage, I did not find any new bugs.

Conclusion

Q1. What did you learn from this project?

Ans: The first thing I learned was how to use Android Studio. This was a good opportunity to learn how to import an open-source project, how to use gradle, and how to build and debug an Android project. Through this project, I learned how to develop a unit test suite by utilizing JUnit and applying several functional testing strategies, such as invalid input testing and boundary testing. In the stage of structural unit testing and integration testing, finding a way to complete 100% code coverage in a selected unit was my goal. By understanding the algorithm and interactions between this selected unit and other relevant units and knowing how to measure the cover coverage, I achieved this goal, and the test cases of integration testing were developed. As for the system testing, I studied how to use Rotobium to develop the scenario testing. Finally, I want to mention that there were many discussion boards talking about how to use JUnit and Rotobium, such as stackoverflow.com that provided me several solutions to complete my test cases.

Q2. What were the biggest challenges in completing this project? How did you overcome these challenges?

Ans: The biggest challenge for me in this project was selecting a proper target Android app. In the beginning of this assignment, I spent a lot of time looking for an app related to the camera because I was familiar with developing camera drivers. Later, I found that it was inappropriate to choose a camera app to proceed with testing because there was almost no algorithms or calculations in these camera apps. The only thing in these apps which was appropriate for testing was GUI. However, this was not our target in this class. After observing more than 50 projects, I found that digit or bit calculation was more appropriate for implementing the test cases. Finally, I found this app, CIDR calculator, which had a lot of calculations within internal classes. It was appropriate for me to use this open-source project to be my target project in this class.

Q3. What would you do differently if you were to test a similar app?

Ans: If I were required to do testing for a similar app, I would start by understanding the algorithms and mechanisms first. The more I understand, the sooner I can generate the test cases and achieve 100% code coverage. I would try to apply more functional testing strategies to the app, such as equivalence class testing, output testing, and catalog based testing. Those testing strategies could make my test suite more comprehensive. Finally, as for system testing, I would try Appium to proceed with the scenario testing instead of Robotium because Appium has been the more popular than Robotium and can be used by a number of programming languages including C# and Python.

Name: Chun-Chi Huang
Course: Software Testing
Professor: Michael McKee

Optional Part Random Testing

Monkey Testing

Q1. Describe some of the events that occurred when running the tests.

Ans: According to the log generated by monkey testing, I found four types of events present in the corresponding log. The first type was the touch event including ACTION_DOWN and ACTION_UP. The second type was the trackball event which could generate ACTION_MOVE. The third type was the flip event. However, when this event was sent, and IOException happened, it did not break the monkey testing. The final type was the rotation event, and apparently, this event caused the screen rotation. The detailed log was recorded in the monkeyTestLog.txt.

Q2. Note if you found any bugs with the tests.

Ans: I found that IOException was caused by flip events, but the monkey testing still passed. No other bugs were found.

Q3. Your personal thoughts on using this form of random testing.

Ans: I knew that monkey testing before because of my previous experiences in the mobile industry. I remember that Android phones needed to pass monkey testing before we got the google logo licenses. If the mobile phone is not reliable, it is easy to fail in the monkey test. By randomly sending events to the mobile phone, anything can happen. People cannot manually test phones in the way performed by monkey testing due to physical limitations. My colleagues usually spent a lot of time trying to pass this test. However, once our products could pass monkey testing, these products became more robust than before. Thus, monkey testing is a useful tool to verify the stability of mobile phones.