

Carter Brezinski

200391111

ENSE 375/475 - Final Documentation

April 27th, 2021

Table of Contents

Table of Contents	2
1. Introduction	3
1.1 Outline of what I completed	3
1.2 My steps to completing certain tasks	5
2. Diagrams, Graphs, and Tables for Exam	8
3. Overview and Comments Written During Exam and Development	
4. References	13-14

Within this documentation are links to my now public github and youtube video below:

Youtube video submission link: https://youtu.be/WD_2dNaDEH0

Or: https://www.youtube.com/watch?v=WD_2dNaDEH0

(If the video is blurry or low Quality email me and I can reupload it.)

Github Repository link:

<https://github.com/CarterBrezinski/ENSE-375-Final/tree/main>

1 Introduction

Hello, my name is Carter Brezinski, my student ID is 200391111, and this is my exam documentation for my ENSE 375 Final. Below you will see diagrams of my completed work, as well as screenshots documenting my work in Jenkins.

1.1: Checklist of what I completed during the exam:

1- Add to Jenkins a “Test Results Analyzer” plugin to help you in visualizing the tests results. ✓

2- Method: (class:Time24 - method: toTime24)✓

- Implement the function.
- draw a CFG for your implementation, and then identify paths required for 100% node coverage, 100% edge coverage, 100% edge-pair coverage and identify all prime paths.
- Build a test case table where each path you identified above corresponds to a testing case.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

3- Method: (class:Time12 – method: subtract)✓

- Carry out an input domain modelling and clearly state the characteristics, domains, blocks, values. Use the pairwise coverage criteria.
- Implement the function by converting the 12-hour time first to 24-hour time.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

4- Method: (class:Date – method: LessThan)✓

- Carry out an input domain modelling and clearly state the characteristics, domains, blocks, values. Use the pairwise coverage criteria.
- Implement the function by converting the 12-hour time first to 24-hour time.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

5- Method: (class:DateTime – method: subtract)✓

- Implement the function.
- draw a CFG for your implementation, and then identify paths required for 100% node coverage, 100% edge coverage, 100% edge-pair coverage and identify all prime paths.
- Build a test case table where each path you identified above corresponds to a testing case.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

COULD NOT COMPLETE THIS PORTION OF THE CODE IN TIME.

6- For the method: (class: Ticket - method: CheckTicket) ✗

- Implement the checkTicket function in the Ticket class.
- draw a CFG for your implementation, and then identify paths required for 100% node coverage, 100% edge coverage, 100% edge-pair coverage and identify all prime paths.
- Build a test case table where each path you identified above corresponds to a testing case.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

2nd Iteration development

7- For the method: (class:Ticket - method: hasCyclicTrip) ✗

- Carry out an input domain modelling and clearly state the characteristics, domains, blocks, values. Use the pairwise coverage criteria.
- Implement the function.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

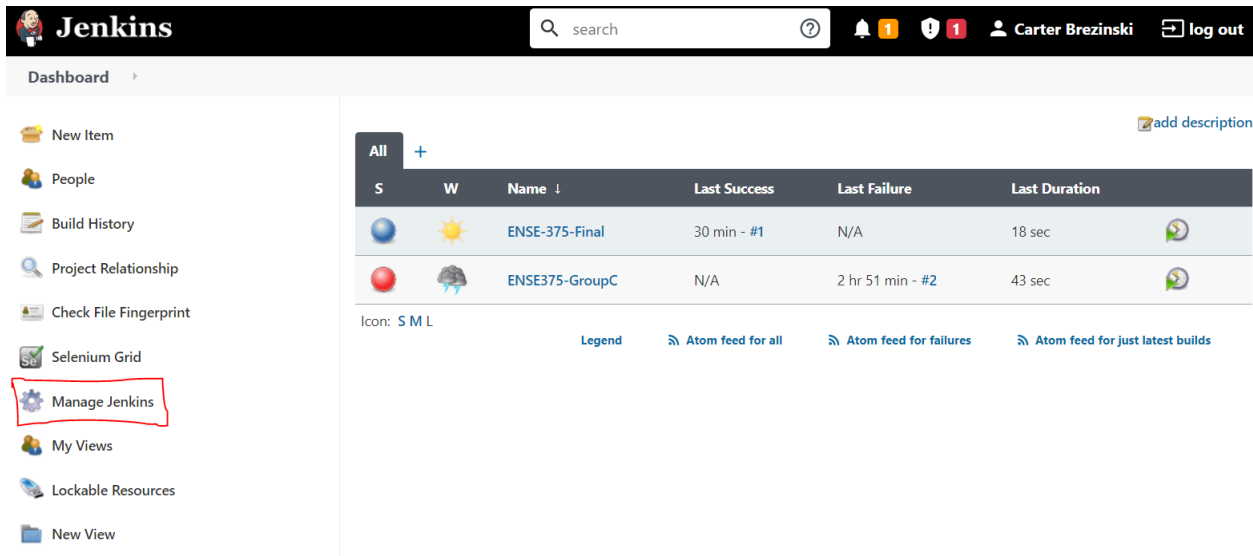
8- Implement functionality to consider different time zones at different airports. ✗

1.2: My steps to completing certain tasks:

Get Jenkins properly working and formatted with Github

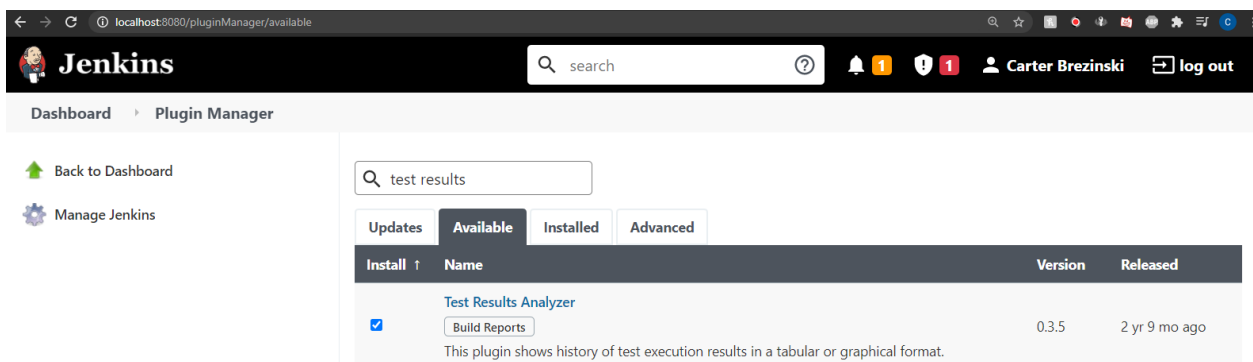
- Biggest issue I had was getting my jenkins properly linked to my private github
 - The reason for this was because I didn't make a commit after my initial commit. Once I had done this 'dummy' commit in a separate branch, I was able to successfully build my jenkins project without issues.

1. Install the plugin instructed by the professor
 - Provide references used



The screenshot shows the Jenkins Dashboard. The top navigation bar includes the Jenkins logo, a search bar, and user information for Carter Brezinski. The left sidebar contains a list of links: New Item, People, Build History, Project Relationship, Check File Fingerprint, Selenium Grid, **Manage Jenkins** (highlighted with a red box), My Views, Lockable Resources, and New View. The main content area displays a table of build jobs. The table has columns for Status (S), Weather (W), Name, Last Success, Last Failure, and Last Duration. Two jobs are listed: 'ENSE-375-Final' and 'ENSE375-GroupC'. Below the table, there are links for 'Legend', 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'.

S	W	Name ↓	Last Success	Last Failure	Last Duration
		ENSE-375-Final	30 min - #1	N/A	18 sec
		ENSE375-GroupC	N/A	2 hr 51 min - #2	43 sec



The screenshot shows the Jenkins Plugin Manager interface. The top navigation bar is the same as the dashboard. The left sidebar has links for 'Back to Dashboard' and 'Manage Jenkins'. The main content area has a search bar with 'test results' entered. Below the search bar are tabs for 'Updates', 'Available', 'Installed', and 'Advanced'. The 'Available' tab is selected, showing a table of available plugins. The table has columns for 'Install', 'Name', 'Version', and 'Released'. One plugin, 'Test Results Analyzer', is listed with a checkbox checked, version 0.3.5, and released 2 yr 9 mo ago. A description below the plugin states: 'This plugin shows history of test execution results in a tabular or graphical format.'

Install	Name	Version	Released
<input checked="" type="checkbox"/>	Test Results Analyzer Build Reports	0.3.5	2 yr 9 mo ago

Installing Plugins/Upgrades

Preparation

- Checking internet connectivity
- Checking update center connectivity
- Success

Test Results Analyzer Success

Loading plugin extensions Success

➡ [Go back to the top page](#)

(you can start using the installed plugins right away)

➡ ☐ Restart Jenkins when installation is complete and no jobs are running


The next step for me was to make another dummy commit, run a test, and see the results via the plugin:

Test Results Analyzer

Options

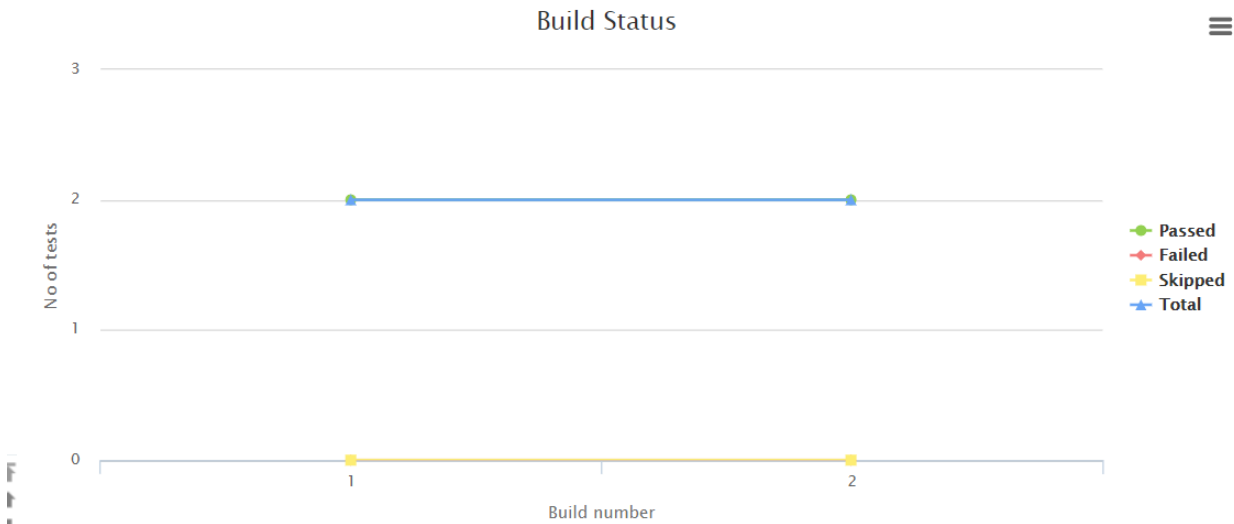
Download Test (CSV)

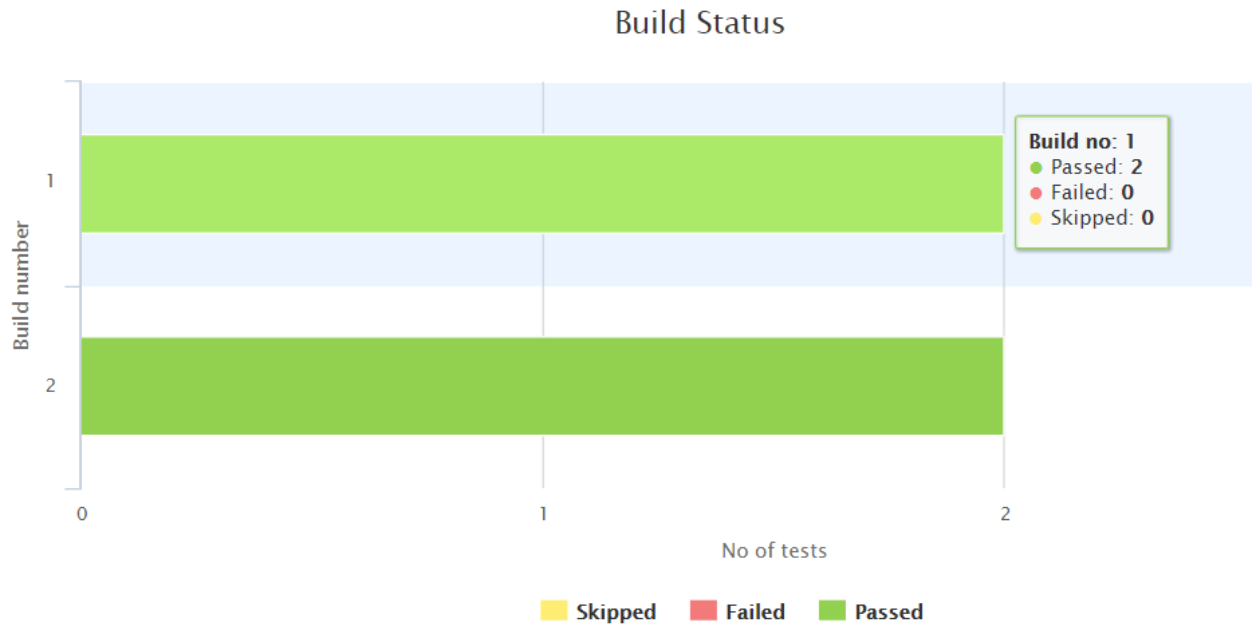
Search:

Chart	Package/Class/Testmethod	Passed	Transitions	2	1
<input type="checkbox"/>	 com.uregina.app	100% (100%)	0	PASSED	PASSED

Top 10 Most Broken Tests

There are no failing tests





2. Evaluate the code given in the zip folder along with the information provided in the exam layout

My first step in evaluating the code was looking through each .java file and seeing what needed to be added/changed, as well as seeing what each function 'needed'.

The layout of the code has 'Todo' in comments.

In my mind it is much easier to search for something like 'QQQ' as it doesn't show up normally in code. So my first step in evaluating the code is adding in these comments for easier searchability.

From there I look deeper in depth at what each function does and what is being passed from method to method.

3. Begin answering the questions provided.

A large portion of my code development involves vigorously commenting my code so that I can reduce as many errors in code and testing as possible before actually developing and writing code.

2 Diagrams, Graphs, and Tables for the Exam

Q2 Test Case Table:

Test No.	Test Case	Inputs	Expected Outputs	Actual Outputs	Success? (y/n)
1	validTime1	Hour = 5 Min = 37 AmPm = pm	17:37	17:37	Y
2	validTime2	Hour = 7 Min = 07 AmPm = am	7:07	7:07	Y
3	validTime3	A = 12 Min = 00 AmPm = am	00:00	00:00	Y
4	validToString	A = 5 B = 7 B = 7	Isosceles	17:37	Y
5	validSubtract	A = 5 B = 12 C = 13	71	71	Y

Q2 CFG:

CF 6 for To Time 24

Def Start (68)

Def min hrs (70, 71)
max hrs

Use am-pm (74)

Use hours (75)

Def hours (76)

Use hours (80)

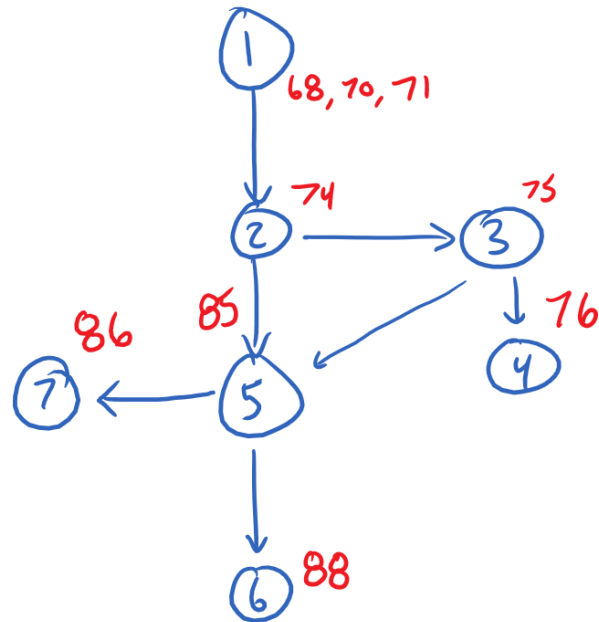
Def hours (81)

Def time (85)

Use exception (86)

Use hours, minutes (85)

Use time (88)



Q3 Input domain model:

Carter Brezinski
200391111

Input domain model for subtraction method

	Characteristic	Block Case 1	Block Case 2
A	Valid Time Entry?	1 or more entries incorrectly entered	all entries entered correctly
B	Valid Subtraction?	No Errors or Exceptions throw during calculations	Valid subtraction takes place and test is successful.
<p>Ideal Case: (A2, B2) Invalid / Impossible* Cases: (A1, B1), (A1, B2), (A2, B1)</p> <p>* It isn't possible for an entry to be entered 'incorrectly' as it will consistently check/ask for valid times, and it should still give a subtraction result even if the answer is negative.</p>			

Q4 Input domain model:

Carter Brezinski
200391111

Input domain model for LessThan method

	Characteristic	Block Case 1	Block Case 2
A	Valid Date Entry	1 or more entries incorrectly entered	all entries entered correctly
B	Valid LessThan test case	Date1 is not < Date2 or Date1 == Date 2	Date1 is < Date2
<p>Ideal Case: (A2, B2) or (A2, B1) Invalid / Impossible* Cases: (A1, B1), (A1, B2)</p> <p>* Invalid entries or a lack of date1 being less than date2 are less than idea and would be considered invalid based on test cases made during development.</p>			

Q5 CFG:

CFG for DateTime Substruct

```
def diff (63)
def boundtest 1 & 2 (66, 67)
use boundtest 1 & 2 (72)
use exception (72)
def date day 1 & 2 (77, 78)
use d1 & d2 (77 & 78)
def datetime
  date 2 time
  date 1 minutes (80 → 85)
  date 2 minutes
  date 1 hours
  date 2 hours
def daydiff (92)
def timediff (93)
use daydiff & timediff (95)
def diff (95)
use diff (98)
```

Q5 Test Case Table:

Test No.	Test Case	Inputs	Expected Outputs	Actual Outputs	Success? (y/n)
1	Failure Case	Date1: 4/29/2021 Time1: 9:00pm Date2: 4/27/2021 Time2: 10:00pm	More than one day exception	More than one day exception	N
2	Success Case	Date1: 4/27/2021 Time1: 9:00pm Date2: 4/27/2021 Time2: 10:00pm	60minutes	60minutes	Y

3 Overview and Comments Written Down During Exam and Development

Here is a brief point by point journal that I made during the development and writing of my exam, so there is an understanding of where I was at and what issues I was experiencing during the exam:

- There was a period of time within my code where I would add my test cases which when I developed and coded them, they made sense and did not pose errors from my point of view or the codes point of view. But Jenkins refused to like how I formatted my code and this brought forth many many errors. This is why in much of my test case code there are lots of comments or commented out test cases.
- From there I progressed and finally made some ground on my time24 and time12 code as well as the test cases for my

- In many of my cases as you would see in my tables, I considered what would be necessary for invalid test cases to occur, but in many of my cases I didn't actually get to fully flesh out invalid cases.
- In many of my test cases, I know it's not good practice, I would consider successfully building an object in one test case as a 'successful build test case' additionally. This isn't good practice nor is it right due to the fact that each test case should only cover 1 specific task or function.

4 Reflection

I would say overall this exam could've easily been spread over the course of 2 days. There were many test cases and hiccups that occurred for me during my development and test case phase that caused major gaps in productive time due to me attempting to solve my current problem.

All this being said, I'm sure there were some students who were more prepared than I was, and didn't take the full exam time. So in that case I understand why the exam was only limited to 12 hours, which is still very gracious.

The exam at first felt very overloaded, but once I got into a groove I began to really clear through objectives. Unfortunately I wasn't able to complete the whole exam outline by the end of it though.

Due to my hiccups during creating test cases many of my test cases were left with only one example of each case, and I mention in my comments had I had more time I would've also added in invalid cases.

5 References

To get an understanding of the plugin I would be working with:

<https://plugins.jenkins.io/test-results-analyzer/>

To better understand enums, as I am unfamiliar with them:

https://www.w3schools.com/java/java_enums.asp

For an error I experienced when testing:

<https://stackoverflow.com/questions/13811020/error-class-x-is-public-should-be-declared-in-a-file-named-x-java>

What I plan(planned) on using for input via the AppTest to test the “user’s input”.

<https://stackoverflow.com/questions/31635698/junit-testing-for-user-input-using-scanner>

For throwing multiple exceptions:

<https://stackoverflow.com/questions/2912565/throwing-multiple-exceptions-in-java/12126931>

My CFGs, test case tables, and Input domain modelling diagrams will all be included in my documentation as I have them poorly drawn and will implement them after recording my video.