



FIT 3162 User Guide

Group Members Name:

Tan Jin En

Yap Rui Yi

Seow Zheng Hao

=====

Table Of Contents

=====

Table Of Contents	2
End User Guide	3
Section 1	3
Subsection 1.1	3
Subsection 1.2	3
Subsection 1.3	5
Subsection 1.4	6
Subsection 1.5	7
Subsection 1.6	8
Technical Guide	9
Section 1	9
Section 2	12

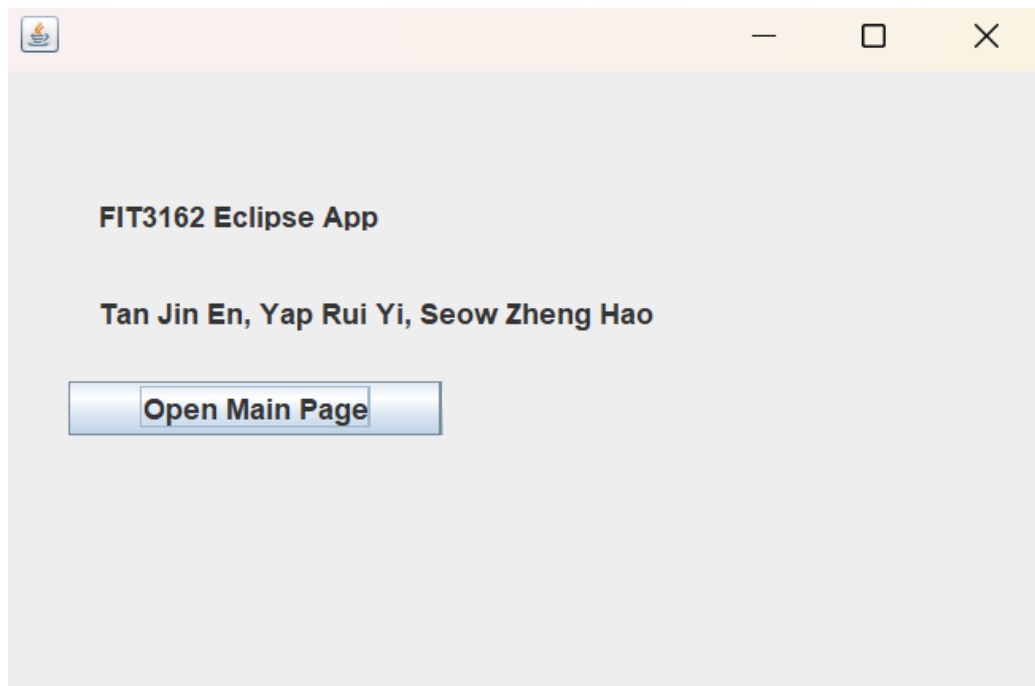
End User Guide

Section 1

Introduction to app user interface and usage flow

Subsection 1.1

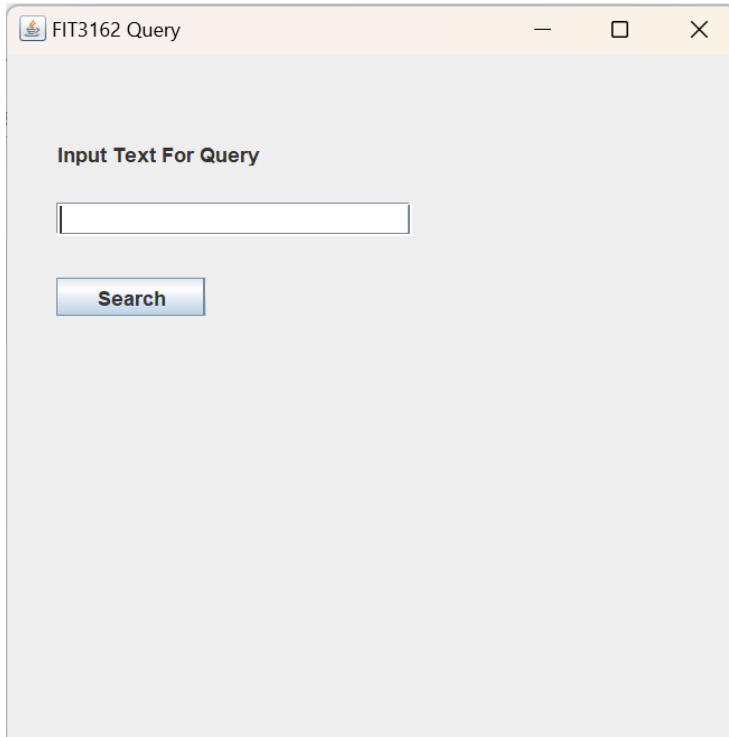
Opening first page:



This initial first page is an introduction to the project and the team members involved. The main page to the app is opened after clicking the “Open Main Page” button.

Subsection 1.2

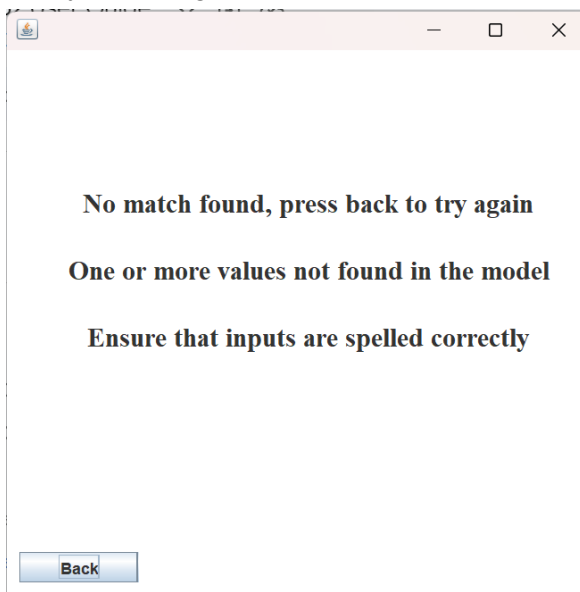
Main query input page:



A screenshot of a web browser window titled "FIT3162 Query". The window has a light gray background. At the top, there is a label "Input Text For Query" in bold. Below this label is a text input field. Underneath the input field is a blue button with the word "Search" in white text.

This page acts as the homepage to query the ontology model and get results based on the input query. The text input is case insensitive, and can take in queries with support for multiple item tags. To query for results that match multiple values, the query takes in values separated by comma, e.g. "nausea,chestpain","chest pain","nausea,gibberish". In cases where one or more of the query input is not found in the model, an error page will be shown to inform the user of the possible changes to be made for the query. A back button is available to return to the query page for another query attempt

Query error page:



A screenshot of a web browser window showing an error message. The window has a light gray background. The error message is displayed in bold text: "No match found, press back to try again", "One or more values not found in the model", and "Ensure that inputs are spelled correctly". At the bottom left of the window is a blue button with the word "Back" in white text.

Subsection 1.3

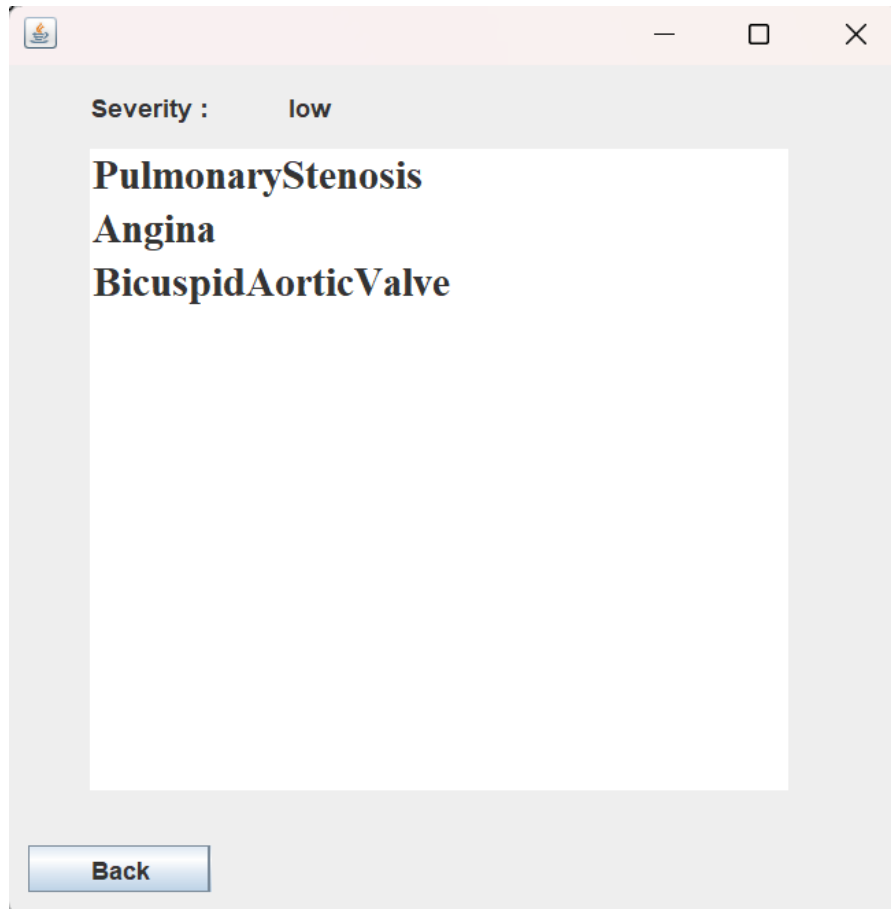
On successful query input match result page:



On successful matching of the query input with values within the ontology model, a result page will be displayed. The "severity rating" shown on the top is a score that is calculated based on the number of returned results with different severities assigned, with a comment based on the rating as well directly below it. The "heart" image in the centre is an illustration of the severity of the returned results. The higher the severity score, the more distorted the image. Below that are 3 different bar chart items that display the results categorised by severity level, Red: high, Orange: medium, Yellow: low threat level. The number on each bar chart indicates the number of results that matched the input query for each severity level. By clicking on the individual bar charts, a new page is opened that displays the results that matched with the input query and have the corresponding severity level. A "back" button is available to return to the query page to query again with a different query input.

Subsection 1.4

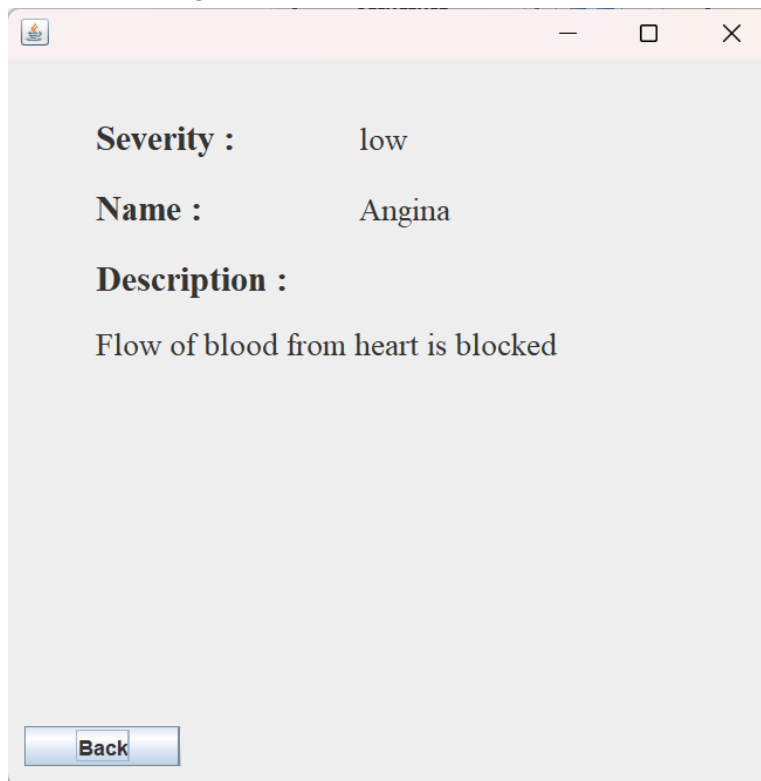
Result display page depending on chosen severity level:



The “severity” on top displays the severity level that the results inside the list belong to. In this example, the input query “fatigue” was used and 3 results with severity level “low” were returned in the list (“PulmonaryStenosis”, “Angina”, “BicuspidAorticValve”). By clicking on the individual results a new page will be opened that displays the description of the selected result. A “back” button is available to return to the result page again.

Subsection 1.5

Description page for selected result:



The "severity" on the top displays the severity level of the current result displayed in the result page. The name of the current result is displayed right below that. The description for the current result is displayed at the bottom of the name. A "back" button at the bottom closes the current page and returns to the previous page that displays the list of results with the same severity level.

Subsection 1.6

1. Limitations

- The spellings for the input query must be exact for the query to function that includes alphabets with no spaces for input queries that contain multiple words
- The data for the query is dependent on a ontology model created in Protege that cannot be edited in the software

2. Starting the software

- Double clicking on the app.exe file will open the opening first page in Subsection 1.1

3. Exiting the software

- At any point of the user usage process, the “x” on the top right of the program will terminate the program

4. Special Situation

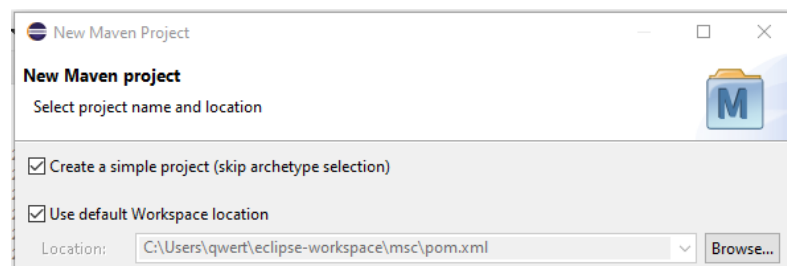
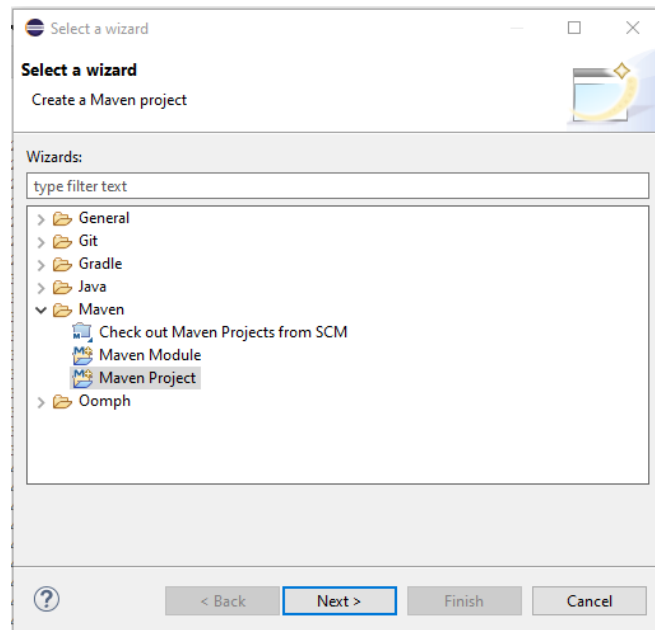
- When no match is made based on the query, users can query again with a different input by pressing the back button and updating the text box
- For situations where the description for the disease is not found, a “missing description” text will be in place.

Technical Guide

Section 1

Set up Eclipse for OWL API development.

1. Install java JDK 21
2. Create a new Maven project (File->New->Other->Maven Project). Check the option “Create simple project”. Click “Next”.



3. Enter Group Id “owlapi.tutorial” and Artifact Id “msc”; click “Finish”

Artifact

Group Id: owlapi.tutorial

Artifact Id: msc

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel

4. Make sure project is referencing the Java 21 Development Kit (Project -> properties -> Java Build Path -> Remove Java JRE -> Add Libraries -> JRE System Library -> Alternate JRE -> JDK 21 -> Finish)

Properties for msc

type filter text

- > Resource
- > Builders
- > Coverage
- > **Java Build Path**
- > Java Code Style
- > Java Compiler
- > Javadoc Location
- > Java Editor
- > Maven
- > Project Natures
- > Project References
- > Refactoring History
- > Run/Debug Settings
- > WikiText

Java Build Path

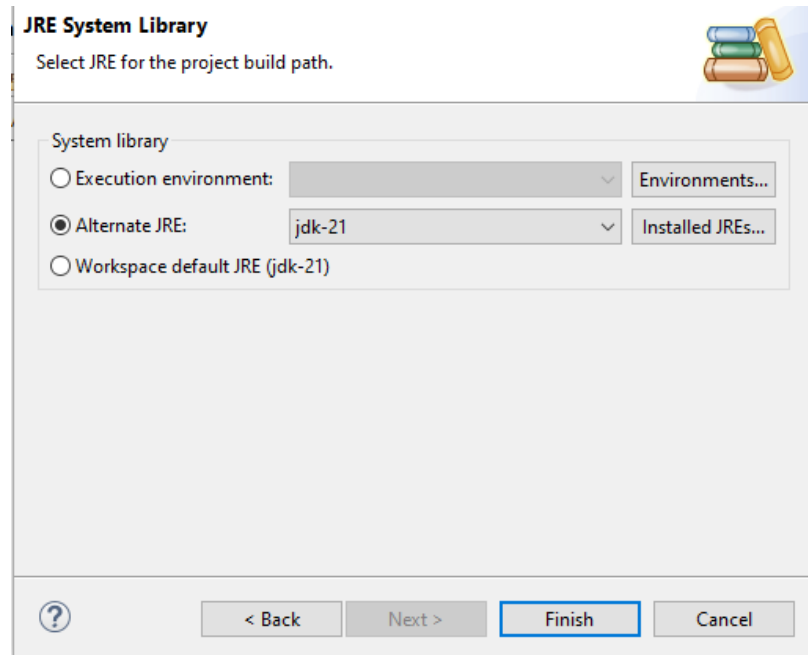
Source Projects Libraries Order and Export Module Dependencies

JARs and class folders on the build path:

- > fyp_tempv2.jar - D:\fypfolder
- > JRE System Library [JavaSE-1.8]
- > Maven Dependencies

Add JARs... Add External JARs... Add Variable... Add Library... Add Class Folder... Add External Class Folder... Edit... Remove Migrate JAR File... Apply

Apply and Close Cancel



5. In the newly created project, double-click on the “pom.xml”, and copy the text below into the pom.xml file, then save the pom.xml file.

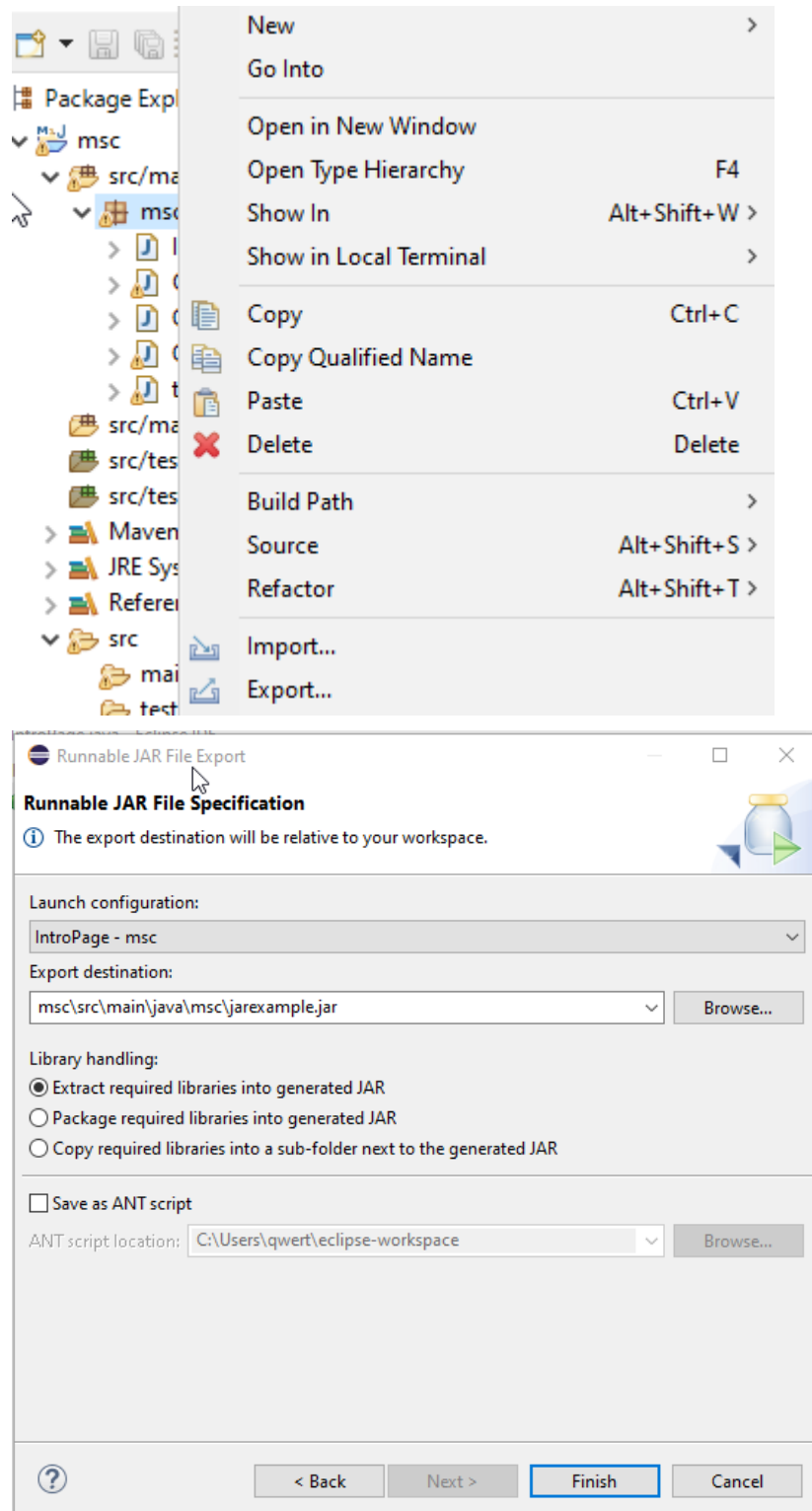
```
*msc/pom.xml x OWLAPIfirst.java IntroPage.java
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi
2   ="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=
3   "http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <artifactId>owlapi-contract</artifactId>
6   <name>OWLAPI :: Tests and Tutorials</name>
7   <parent>
8     <groupId>net.sourceforge.owlapi</groupId>
9     <artifactId>owlapi-parent</artifactId>
10    <version>5.5.0</version>
11  </parent>
12  <dependencies>
13    <dependency>
14      <groupId>${project.groupId}</groupId>
15      <artifactId>owlapi-apibinding</artifactId>
16      <version>${project.version}</version>
17    </dependency>
18    <dependency>
19      <groupId>com.hermit-reasoner</groupId>
20      <artifactId>org.semanticweb.hermit</artifactId>
21      <version>1.3.8.4</version>
22    </dependency>
23  </dependencies>
24 </project>
```

6. Now project is suitable for OWLAPI development, to create a java file simply right click on the package “owlapi.tutorial” (underneath src) and selecting New->Class to create new java class

Section 2

making an executable jar file in Java

1. Click on msc package folder -> export as runnable JAR File -> select launch configuration as "IntroPage - msc" -> specify export destination -> finish



FIT 3162 Test Report

Group Members Name:

Tan Jin En

Yap Rui Yi

Seow Zheng Hao

Introduction

In this report, we will showcase how our software satisfies the set of software requirements specified in the project proposal. To achieve this, we will plan out 2 kinds of testing methods - whitebox test and blackbox test. Black Box test is a test method that only considers the external behaviour of the system; the internal workings of the software is not taken into account, and whitebox testing is a software testing technique that focuses on examining its internal functions instead of its external behaviour. We will carry out black box test by first testing the input text for the query for symptoms. If the input is correct i.e. if the symptom that is in input has at least one corresponding heart disease, then the query result page will pop up after clicking the search button. Inside the query result page, the user will then be presented a heart image that becomes increasingly distorted with higher number of diseases returned from the query result; thereby showing the severity of the symptom inputted from the query. Additionally, a new list view showcasing a list of possible diseases will pop up by clicking on the heart image, so that the user can know what are the possible diseases returned from specifying a specific symptom. As for whitebox testing, we will refactor the code so that all the OWLAPI functions inside main will be moved into another method. The main function will then be calling the method instead of the OWLAPI functions directly, so that we can make the code more modular and easier to test. By doing so, we can now perform white box testing on the methods.

Unit Testing

Unit testing for the whole project is done using JUnit. The main advantage of using JUnit is due to its modularity. It contains several modules that allow us to run different functions of a software application individually; this enables us to perform white box and black box testing for our software application.

White Box Testing

Test scenario	Description	Expected Result	Test Status	Tester
Test 1	Test if loadOntology() returns the expected ontology.	The function successfully loaded the same ontology and returns	Pass	Jin En Tan
Test 2	Test if CreateSymptom NameMapping() returns the expected mapping of symptoms of disease.	The function successfully maps the symptoms and hence no error will be shown.	Pass	Jin En Tan

```
@Test
public void testLoadOntology() throws OWLOntologyCreationException {
    // Arrange
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    File file = new File("C:\\Users\\JET0614\\Downloads\\heartdiseaseontology.owl");

    // Act
    OWLOntology result = OWLAPIFirst.loadOntology(manager, file);

    // Assert
    assertNotNull(result);
}
```

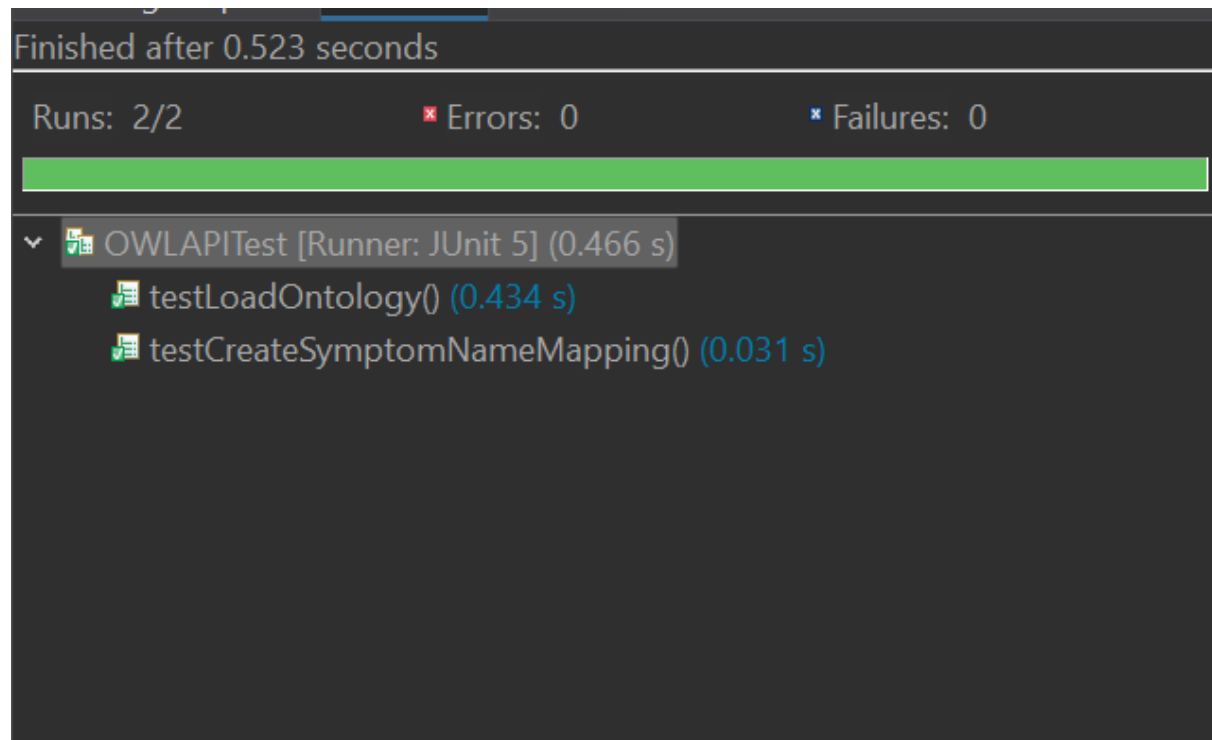
Test 1 screenshot of code

```
@Test
public void testCreateSymptomNameMapping() throws OWLOntologyCreationException {
    // Arrange
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    File file = new File("C:\\Users\\JET0614\\Downloads\\heartdiseaseontology.owl");
    OWLOntology ontology = manager.loadOntologyFromOntologyDocument(file);

    // Act
    Map<String, String> result = OWLAPIFirst.createSymptomNameMapping(ontology);

    // Assert
    assertNotNull(result);
}
```

Test 2 screenshot of code



Screenshot of both test 1 and 2 ran successfully

Black Box Testing

Test scenario	Description	Expected Result	Test Status	Tester
Test 3	Test if queryDiseases WithSymptom does not throw any exception and it will return on console the list of diseases with the symptom.	Console will have a list of diseases that have symptoms of nausea.	Pass	Jin En Tan
Test 4	Test if the function to make the user query lowercase is working.	Console should not output any error	Pass	Jin En Tan

```
@Test
public void testQueryDiseasesWithSymptom() throws OWLOntologyCreationException {
    // Arrange
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    File file = new File("C:\\Users\\JET0614\\Downloads\\heartdiseaseontology.owl");
    OWLOntology ontology = manager.loadOntologyFromOntologyDocument(file);

    ByteArrayInputStream in = new ByteArrayInputStream("nausea\n".getBytes());
    System.setIn(in);

    // Act and Assert
    assertDoesNotThrow(() -> OWLAPIFirst.queryDiseasesWithSymptom(manager, ontology));
}
```

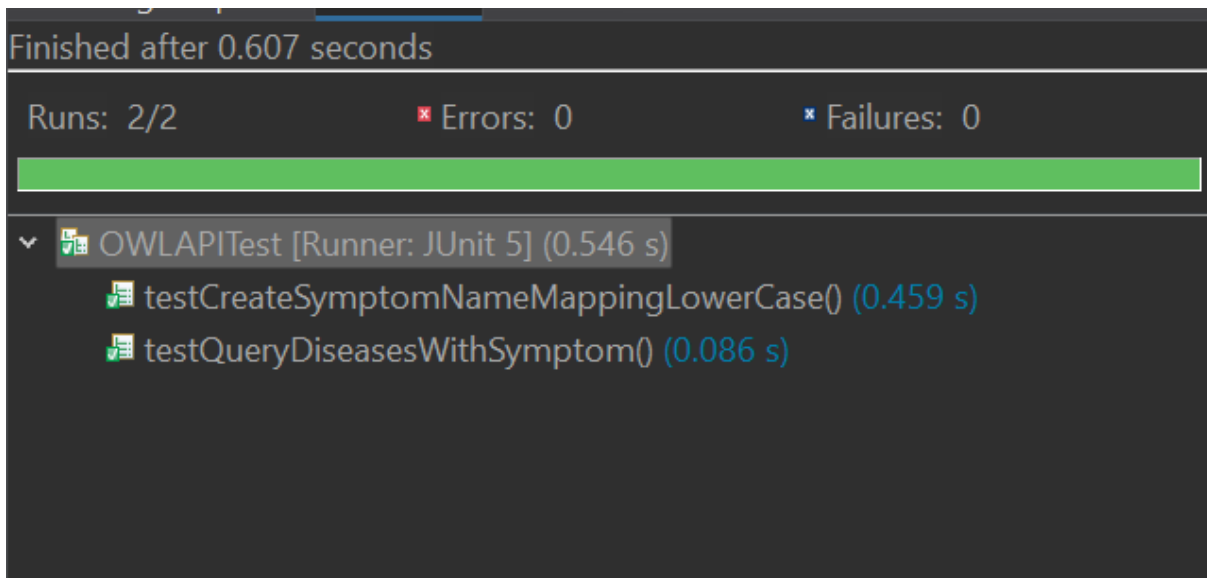
Test 3 screenshot of code

```
@Test
public void testCreateSymptomNameMappingLowerCase() throws OWLOntologyCreationException {
    // Arrange
    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    File file = new File("C:\\Users\\JET0614\\Downloads\\heartdiseaseontology.owl");
    OWLOntology ontology = manager.loadOntologyFromOntologyDocument(file);

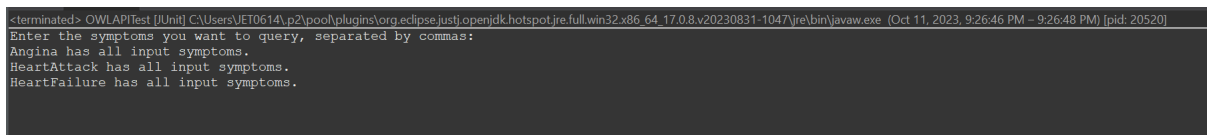
    // Act
    Map<String, String> result = OWLAPIFirst.createSymptomNameMapping(ontology);

    // Assert
    assertTrue(result.containsKey("nausea"));
    assertNotNull(result.get("nausea"));
}
```

Test 4 screenshot of code



Screenshot of both test 3 and 4 ran successfully



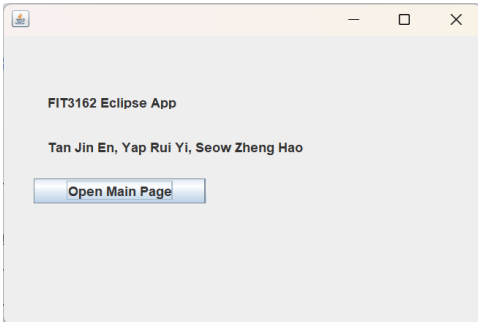
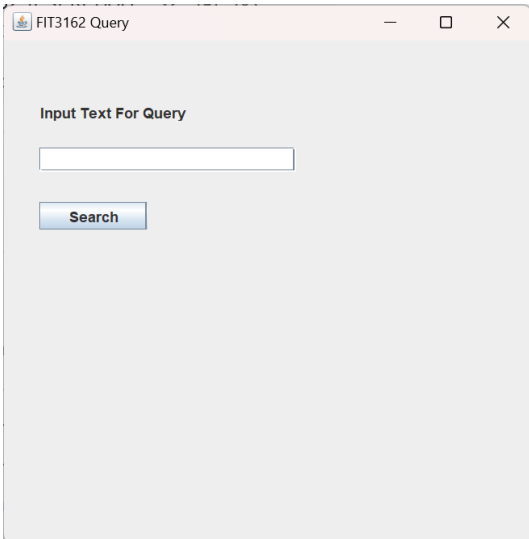
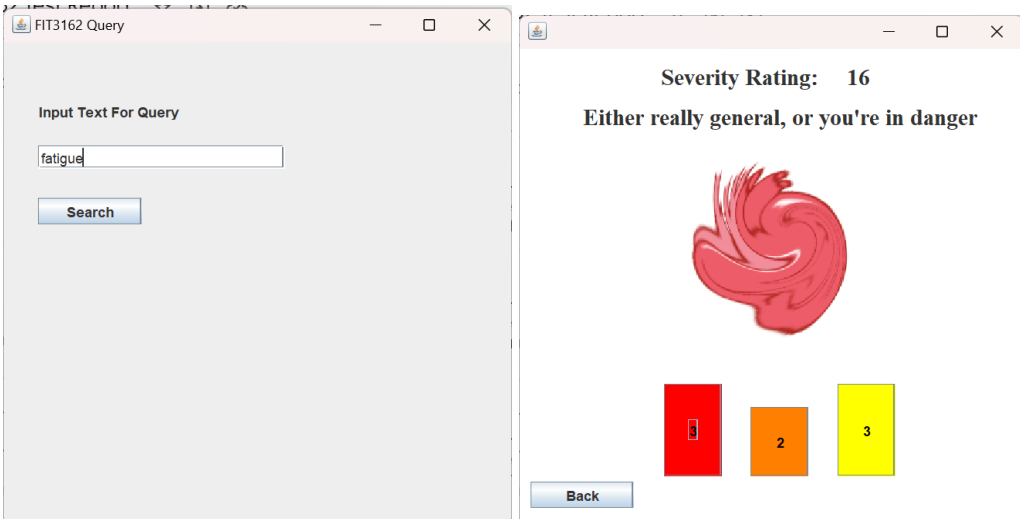
Screenshot of expected output in console for test 3

Integration Testing

Test Scenario	Test Description	Test Steps	Expected Results	Observed
1	Test program run	<ol style="list-style-type: none"> 1. Run app.jar executable jar file 2. Check for application window 	Program successfully runs with opening first page displayed	Executable file ran perfectly with no problems and opened starting page
2	Test open main page button work	<ol style="list-style-type: none"> 1. Run app.jar executable jar file 2. Click "open main page" button 	Programs successfully runs and after button click the query page is displayed	Button was functional and opened the correct page
3	Test query operational	<ol style="list-style-type: none"> 1. When on query page, input query into text box 2. Click "search" button 	After opening query page, input sample query "fatigue", query result page is opened and displays a successful match	Query functionality works, query result page displayed correct number of matches
4	Test query is not case sensitive	<ol style="list-style-type: none"> 1. When on query page, input query with random capitalisations into text box 2. Click "search" button 	After opening query page, input sample query "faTlguE", query result page is opened and displays correct matches for "fatigue"	Query result maintains the same after changing capitalisation of query input
5	Test correct query result for sample query input "chestpain"	<ol style="list-style-type: none"> 1. When on query page, input query "chestpain" into text box 2. Click "search" button 	After opening query page, input sample query "chestpain", query result page is opened and displays correct number of matches for "chestpain" based on	Query result page displays correct results that match the inbuilt dl query for the ontology model in protege

			protege model and correct matches in result list display	
6	Test correct query result for sample query input "nausea"	<ol style="list-style-type: none"> 1. When on query page, input query with "nausea" into text box 2. Click "search" button 	After opening query page, input sample query "nausea", query result page is opened and displays correct number of matches for "nausea" based on protege model and correct matches in result list display	Query result page displays correct results that match the inbuilt dl query for the ontology model in protege
7	Test that all back buttons are fully functional	<ol style="list-style-type: none"> 1. Open the app.exe 2. Conduct a query and open up into one of the result list pages and open a description page 3. Use the back button to return to the result list and ensure that another result description page can be opened without having to restart the app 	After opening query page, use sample query "nausea", click on any of the bar chart to open result list and click on a result to open the description page, click back, the result list of the current query should be intact and another result description page can be opened	After opening up description pages for results, returned to the result list with no issues and query results still intact, can still open another result description page (certain descriptions for some results are not completed and will be completed during live testing)

Test images

1	 <p>The screenshot shows a window titled "FIT3162 Eclipse App". Inside the window, the text "FIT3162 Eclipse App" is displayed at the top, followed by the names "Tan Jin En, Yap Rui Yi, Seow Zheng Hao". Below the names is a button labeled "Open Main Page".</p>
2	 <p>The screenshot shows a window titled "FIT3162 Query". Inside the window, the text "Input Text For Query" is displayed above a text input field. Below the input field is a button labeled "Search".</p>
3	 <p>The screenshot shows two windows. The left window is titled "FIT3162 Query" and contains the "Input Text For Query" field with the text "fatigue" entered, and the "Search" button. The right window displays the results of the search. It shows a "Severity Rating: 16" and the text "Either really general, or you're in danger". Below this text is a red, swirling, abstract graphic. At the bottom of the right window, there are three colored squares: a red square with the number 1, an orange square with the number 2, and a yellow square with the number 3. A "Back" button is located at the bottom left of the right window.</p>

4

FIT3162 Query


Input Text For Query

faTlguE

Search

Severity Rating: 16

Either really general, or you're in danger



3

2

3

Back

5

FIT3162 Query


Input Text For Query

chestpain

Search

Severity Rating: 12

There's a lot of possibilities



3

1

1

Back

Severity : high


AtrialFibrillation
HeartAttack
HeartFailure


Back

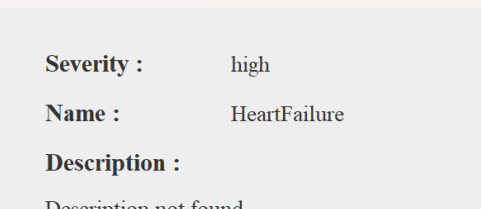
Severity : mid

Arrhythmia

Back

	<div data-bbox="285 253 815 786"> <p>Severity : low</p> <p>Angina</p> <p>Back</p> </div> <div data-bbox="823 210 1327 786"> <p>Query (class expression)</p> <p>hasSymptom some ChestPain</p> <p>Execute Add to ontology</p> <p>Query results</p> <p>Subclasses (6 of 6)</p> <ul style="list-style-type: none"> Angina Arrhythmia AtrialFibrillation HeartAttack HeartFailure owl:Nothing </div>
6	<div data-bbox="285 824 831 1379"> <p>FIT3162 Query</p> <p>Input Text For Query</p> <p>nausea</p> <p>Search</p> </div> <div data-bbox="839 837 1378 1364"> <p>Severity Rating: 7</p> <p>Get That Checked</p>  <div> <div>2</div> <div>0</div> <div>1</div> </div> <p>Back</p> </div> <div data-bbox="285 1391 804 1912"> <p>Severity : high</p> <p>HeartAttack</p> <p>HeartFailure</p> <p>Back</p> </div> <div data-bbox="812 1391 1331 1912"> <p>Severity : low</p> <p>Angina</p> <p>Back</p> </div>

	<div data-bbox="288 217 809 714"> <p>DL query:</p> <p>Query (class expression)</p> <p>hasSymptom some Nausea</p> <p><input type="button" value="Execute"/> <input type="button" value="Add to ontology"/></p> <p>Query results</p> <p>Subclasses (4 of 4)</p> <ul style="list-style-type: none"> ● Angina ● HeartAttack ● HeartFailure ● owl:Nothing </div>
7	<div data-bbox="288 754 833 1308"> <p>FIT3162 Query</p> <p>Input Text For Query</p> <p>nausea</p> <p><input type="button" value="Search"/></p> </div> <div data-bbox="839 754 1378 1308"> <p>Severity Rating: 7</p> <p>Get That Checked</p>  <div> <div>2</div> <div>0</div> <div>1</div> </div> <p><input type="button" value="Back"/></p> </div> <div data-bbox="288 1317 809 1879"> <p>Severity : high</p> <p>HeartAttack</p> <p>HeartFailure</p> <p><input type="button" value="Back"/></p> </div> <div data-bbox="812 1317 1367 1879"> <p>Severity : high</p> <p>Name : HeartAttack</p> <p>Description :</p> <p>Description not found</p> <p><input type="button" value="Back"/></p> </div>



Severity : high

Name : HeartFailure

Description :
Description not found

Back

Useability Test

Limit testing with input

Situation	Description	Expected	Observed
1	Input "FatiGuE"	Correct result page	Correct result page
2	Input value that is not in database	No match page is displayed	No match page is displayed
3	Input multiple values separated by comma with all correct values	Correct result page that shows results that match all query input	Correct result page that shows results that match all query input
4	Input multiple values separated by comma with some incorrect values	No match page is displayed	No match page is displayed

Testing Limitations & Recommendations for improvement

Our testing process is limited by 3 factors: 1) our ontology model only consists of classes that are specific to heart diseases and symptoms related to heart diseases, 2) our query is not flexible as it only returns a set of diseases that is based on conjunctions of symptoms as input e.g., Nausea, ChestPain as input will return a set of diseases that have Nausea AND ChestPain not Nausea OR ChestPain, and 3) JUnit - the unit testing framework we are using is limited in the way it reports errors, which can make it difficult to identify the cause of a test failure. As a result, we are unable to test diseases returned from a query that is not related to heart disease symptoms as input. We also cannot query for diseases that are based on disjunctions of symptoms as input. Furthermore, although we can identify which function at a specific line is throwing an error, due to the limitation of JUnit, we cannot identify the exact reason for why a function throws an error or returns an unexpected output. In the event that a wrong symptom is detected from the input, the program will simply loop again asking for the user to enter a correct input.

We can improve our program by first focusing on making our query accept symptoms that are unrelated to heart disease so that we can include more symptoms and diseases that are not related to just heart disease. This can be done by adding more diseases and symptoms unrelated to heart diseases as classes, then establishing a relationship between the domain(disease) and range(symptoms) by the use of the "hasSymptom" object property, then creating individuals for the diseases; the individuals will then be returned as output after the query. Besides that, we can also increase flexibility of our query by making our query to support disjunction of symptoms as input inside the UI. We can achieve this by creating a dropdown option in the query page so that users can have an option to select the type of query they want. On top of that, a better description of diseases returned from the query should also be done so that end users will have a better understanding of the type of diseases returned from the query. For example, the severity rating shown for each disease can be further improved by having a number based system instead of classifying each disease to have only one of three levels of severity - low, medium or high. By doing so, the end user can understand which diseases returned are more or less life-threatening. At last, We should have used TestNG instead of JUnit to perform unit testing, because TestNG supports running tests in parallel, making it possible to speed up complex tests, and also makes our software more scalable if we decide to add on more functionalities in future iterations. It can also provide a more detailed and comprehensive reporting of errors; thus making it easier to identify the cause of a test failure.