CSE 338 Software Testing, Validation & Verification

Course Project

FIFA TEAM RANKING

LINK TO PREOJECT REPOSITORY

Submitted to:

Prof. Dr. Islam Elmadah

Eng. Adham Nour

Submitted by:

Youssef George Fouad	19P9824
Mostafa Nasrat Metwally	19P4619
Kerollos Wageeh Youssef	19P3468
Anthony Amgad Fayek	19P9880

Table of Contents

List of Figures

1. INTRODUCTION

National football teams are ranked each month according to some criteria, while the club teams are ranked annually according to another criteria. This desktop JAVA application is developed to view national teams ranking from FIFA's ranking history and clubs' rankings from UEFA's ranking history and are updated according to the latest changes. The application is developed based on testing methodologies to ensure functional and non-functional accuracy of the system using automated testing tools and frameworks, like Junit, Testfx, Awaitility, etc.

2. APPLICATION STRUCTURE

The application consists of several modules and layers that forms the desktop app full functionality and interface.

2.1. Backend data module

This module represents the backend layer of the system that gets the data using two methods, scrapping or from an API.

2.1.1. API calls

The NatScrapping class is responsible for getting the national teams' data in JSON file format from FIFA's API. This API was found when looking at the network packets incoming to the FIFA | COCACOLA RANKINGS website. This API has a URL that consists of "https://www.fifa.com/api/ranking-overview?locale=en&dateId=id" followed by the id of the month.

This ID has the following format:

1. Any date after 14 Feb 2007 has an id of the day difference between your chosen date and 14 Feb 2007 plus 8079
2. Any date before that is has an id with its order in the choices and a number is skipped on 18 Dec 2002 going up.

2.1.2. Scrapping

The UefScrapping class is responsible for getting the UEFA clubs' data using scrapping implemented to collect the data directly from the UEFA.com website. The main website has another embedded static html page within it that was found in the main website's network packets. The static html link is the one scraped and its URL is made up of "https://www.uefa.com/nationalassociations/uefarankings/club/libraries//years/" followed by the year of the required data. Jsoup library is used to connect to the website and get its HTML elements by any of their attributes to collect the required data about the club.

2.2. Middleware layer

After getting data from the API or using scrapping, data should be casted to objects of custom classes before being manipulated or viewed to the user. Two custom classes are created to define a suitable blueprint of the retrieved or the scraped data.

2.2.1. NatTeam

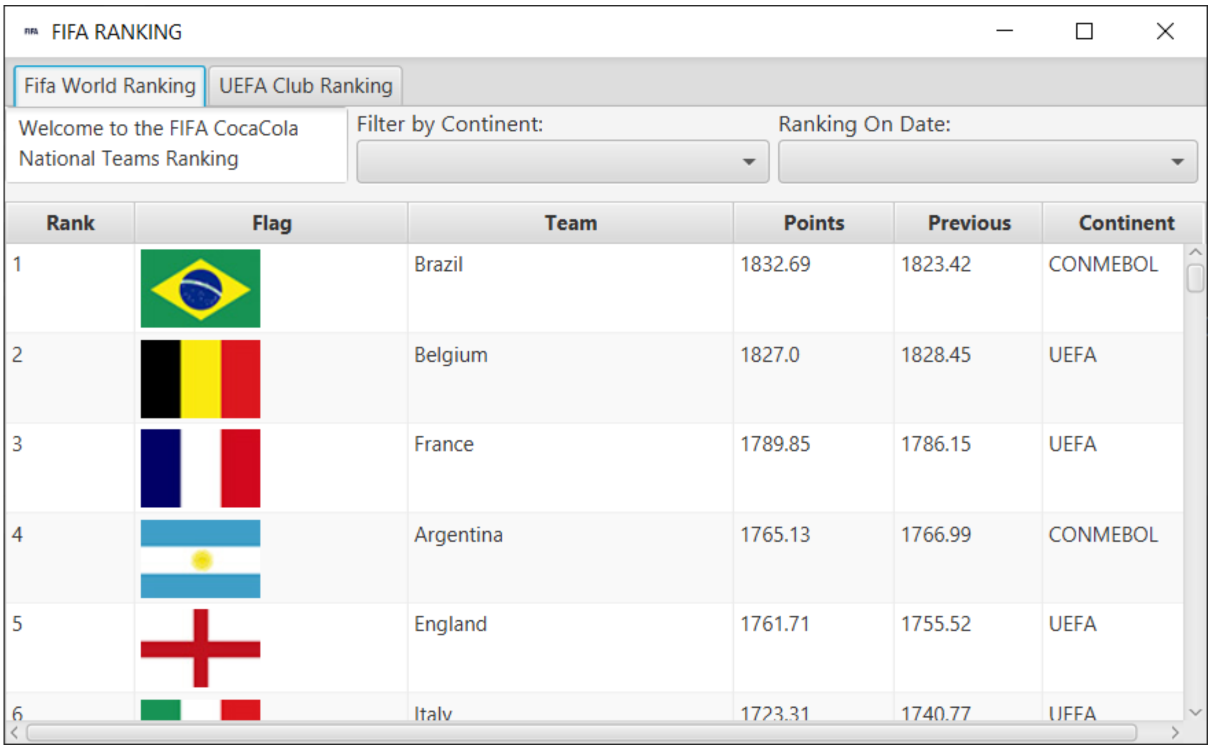
NatTeam class defines the template which the retrieved national teams' data must be casted to, to be used in our application. It consists of several attributes: national team name, location to specify the country's continent, country code that resembles the country name's abbreviation, total points, previous points, rank, and the national flag.

2.2.2. UefTeam

UefTeam class defines the template which the scraped European club teams' data must be casted to, in order to be used in our application. It consists of several attributes: team name, country, team code, current points in addition to points history of the past 5 seasons, current rank, and flag.

2.3. GUI module

This module is responsible for the presentation of the data to the user. It consists of 2 pages, one for the national teams and the other for the European clubs. Below are sample screenshots of the GUI of the app.


FIFA RANKING

Fifa World Ranking | UEFA Club Ranking

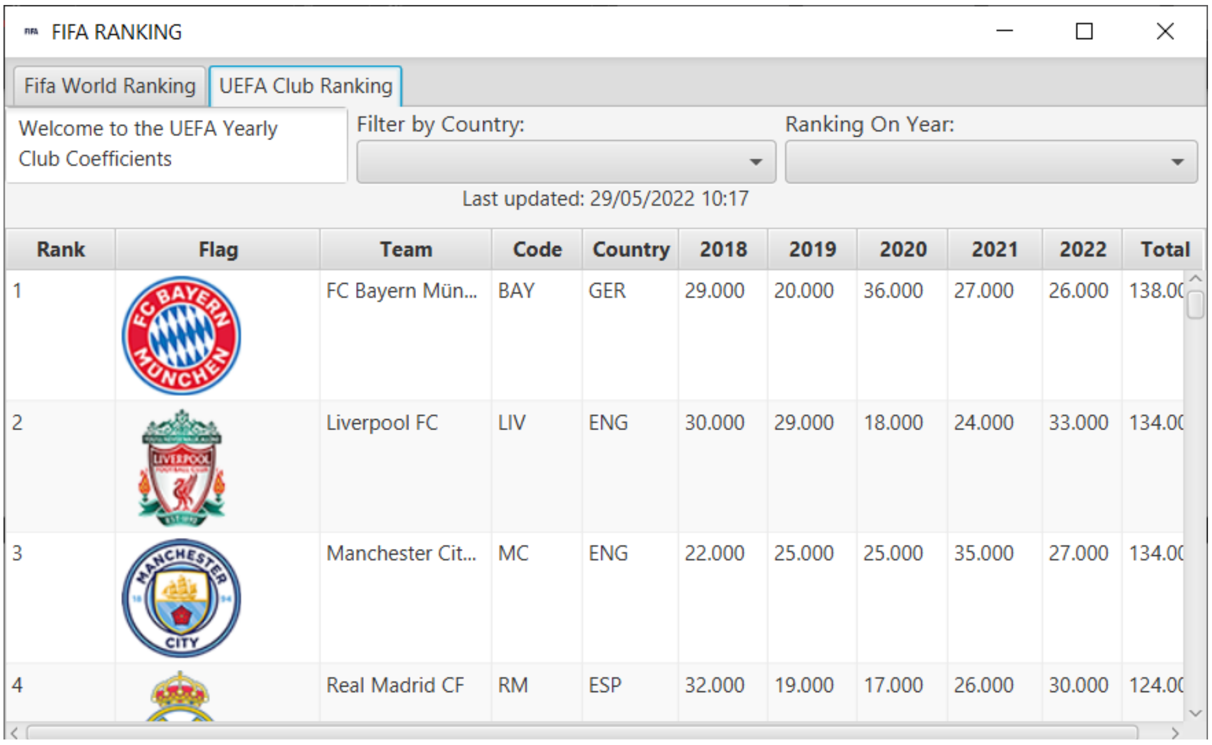
Welcome to the FIFA CocaCola National Teams Ranking

Filter by Continent:

Ranking On Date:

Rank	Flag	Team	Points	Previous	Continent
1		Brazil	1832.69	1823.42	CONMEBOL
2		Belgium	1827.0	1828.45	UEFA
3		France	1789.85	1786.15	UEFA
4		Argentina	1765.13	1766.99	CONMEBOL
5		England	1761.71	1755.52	UEFA
6		Italy	1723.31	1740.77	UEFA

Figure 1: FIFA National Teams World Ranking Tab


FIFA RANKING

Fifa World Ranking | UEFA Club Ranking

Welcome to the UEFA Yearly Club Coefficients

Filter by Country:

Ranking On Year:

Last updated: 29/05/2022 10:17

Rank	Flag	Team	Code	Country	2018	2019	2020	2021	2022	Total
1		FC Bayern Mün...	BAY	GER	29.000	20.000	36.000	27.000	26.000	138.0(
2		Liverpool FC	LIV	ENG	30.000	29.000	18.000	24.000	33.000	134.0(
3		Manchester Cit...	MC	ENG	22.000	25.000	25.000	35.000	27.000	134.0(
4		Real Madrid CF	RM	ESP	32.000	19.000	17.000	26.000	30.000	124.0(

Figure 2: UEFA Yearly Club Coefficients

This module uses the 2 modules mentioned before it. However, it runs the scrapping module in a different thread than the main GUI thread. In addition to that the GUI displays a loading screen on the table until its corresponding scrapping module finishes its fetching and converting that fetched data into a JavaFX Observable Array List.

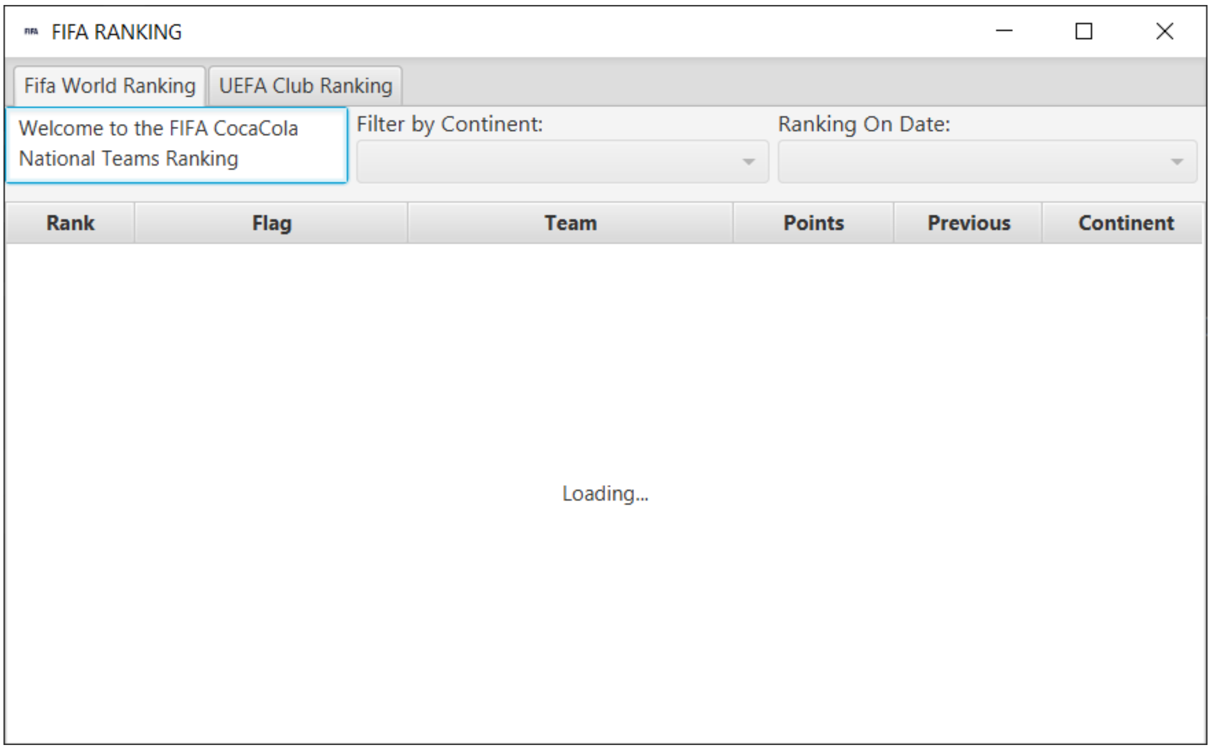


Figure 3: Application Loading Screen

3. UNIT TESTING

To develop this application, we should test it on many levels and using different methods to fully test the application's functional and non- functional requirements. Testing techniques include but not limited to unit testing, integration testing, GUI testing, and performance testing.

Unit testing is required to test single entities in our code as a single function, class, or module; therefore, in this section we design and used tests to test our application's main functions where these tests can be found in 2 classes: NatScrappingUnitTest and UefScrappingUnitTest, under the Test module as shown in the following figure.

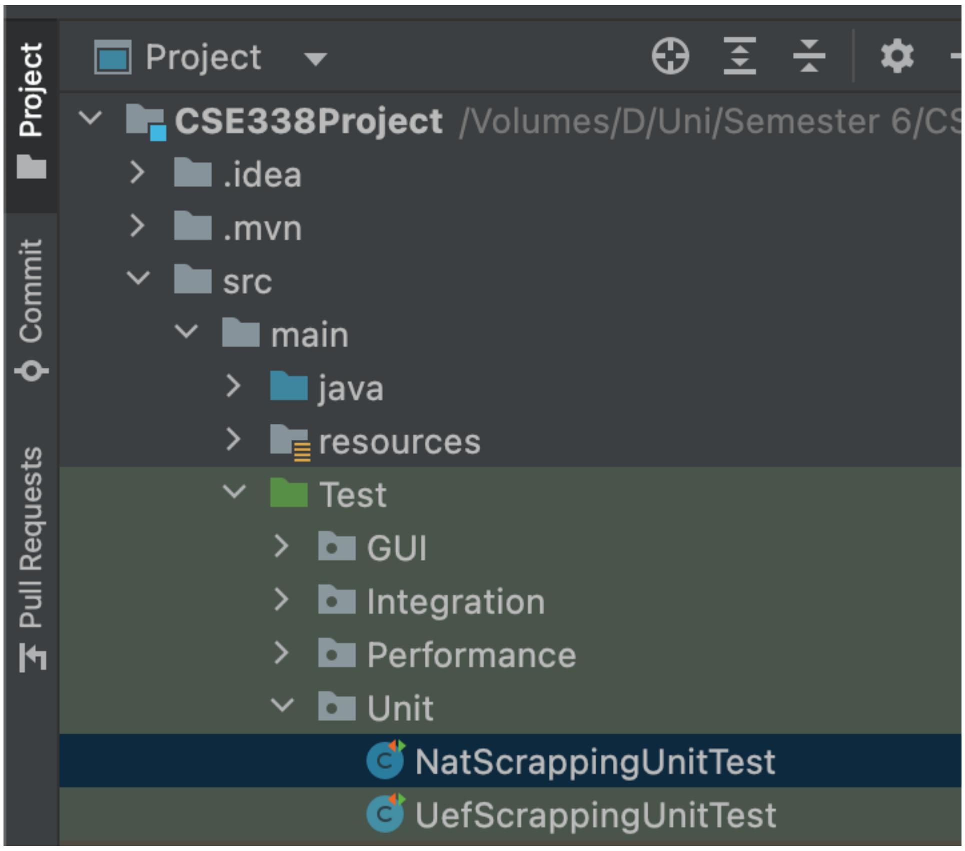


Figure 4: Unit Testing Package

These tests were designed to test functions based on equivalence partitioning where each category is tested by a test case. Categories can be boundary values, middle values or values that are just before or after the boundaries.

FYI: Any testing class decorated with `@ExtendWith(ApplicationExtension.class)` requires the following Java Runtime parameters when started:

```
--add-exports javafx.graphics/com.sun.javafx.application=ALL-UNNAMED
```

3.1. *NatScrapping methods testing*

NatScrapping class includes 2 methods, ***getDates***() – ***getRanking***(), that need to be tested on their own as units. Therefore, ***NatScrappingUnitTest*** class include tests for the 2 mentioned methods as follows. Each of the two methods' test cases are grouped in a nested class.

3.1.2. GetDatesTests class

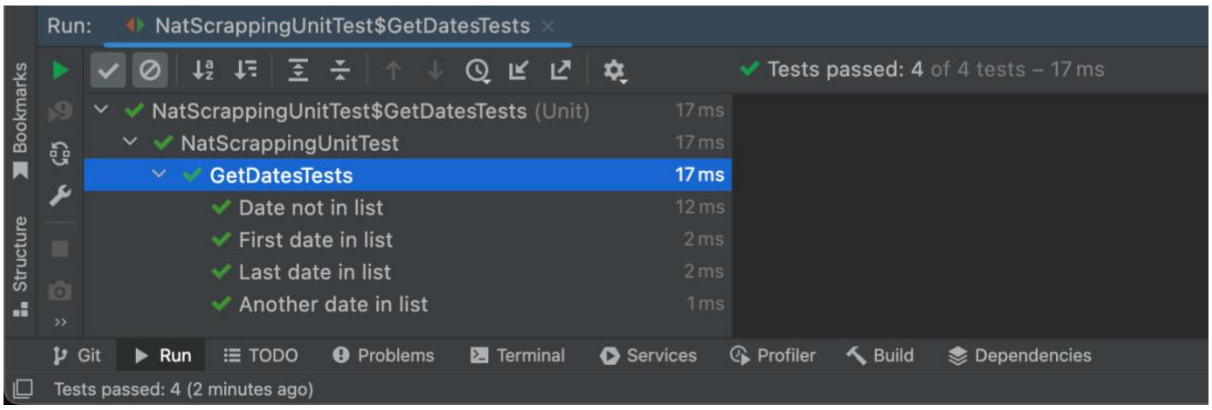
Initially, we have to setup the nested class using @Nested and use the @BeforeAll to get the dates list that we would run our test cases on before running them.

```
@Nested
class GetDatesTests {
    static ObservableList<String> datesObservableList;
    @BeforeAll
    public static void beforeAll() throws FileNotFoundException {
        datesObservableList = NatScrapping.getDates();}
```

Four test cases in our test suite for unit testing of this method.

```
@Test
@DisplayName("First date in list")
public void firstDateTest() {
    assertEquals("31 Mar 2022", datesObservableList.get(0));}

@Test
@DisplayName("Another date in list")
public void anotherDateTest() {
    assertEquals("03 Oct 2012", datesObservableList.get(100));}

@Test
@DisplayName("Last date in list")
public void lastDateTest() {
    assertEquals("31 Dec 1992", datesObservableList.get(318));}

@Test
@DisplayName("Date not in list")
public  void dateNotInList(){
    assertFalse(datesObservableList.contains("01 01 2001"));}
```

Resulting in four successes.



```
Run:   NatScrappingUnitTest$GetDatesTests
✓ Tests passed: 4 of 4 tests – 17 ms
✓ NatScrappingUnitTest$GetDatesTests (Unit)   17 ms
  ✓ NatScrappingUnitTest                       17 ms
    ✓ GetDatesTests                            17 ms
      ✓ Date not in list                       12 ms
      ✓ First date in list                      2 ms
      ✓ Last date in list                       2 ms
      ✓ Another date in list                    1 ms
Tests passed: 4 (2 minutes ago)
```

Lastly, we would free the list after finishing all the tests using **@AfterAll**.

```
@AfterAll
public static void afterAll() {
    datesObservableList = null;}
```

3.1.3. GetRankingTests class

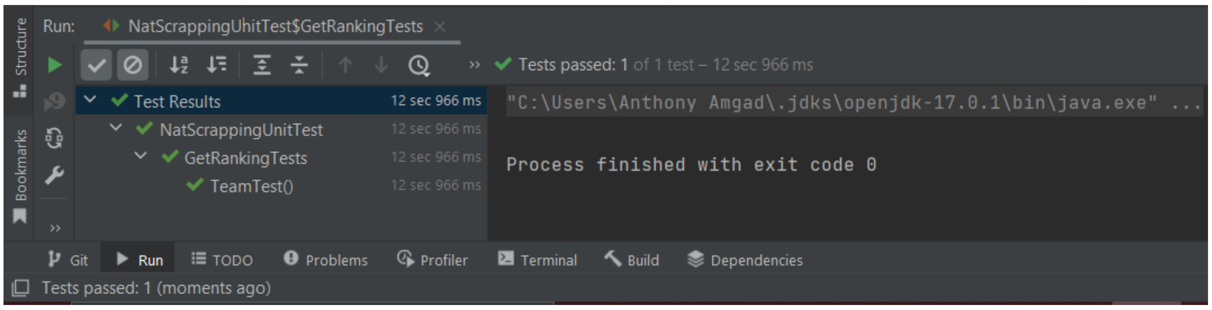
We setup the nested class using ***@Nested***, ***@ExtendWith(ApplicationExtension.class)***, before using ***@Start*** to get the list we are going to test.

```
@Nested
@ExtendWith(ApplicationExtension.class)
class GetRankingTests {
    static ObservableList<NatTeam> teamsObservableList;
    @Start
    public void start(Stage stage) throws Exception {
        teamsObservableList = NatScrapping.getRanking(13603);}
```

Test case in our test suite for unit testing of this method.

```
@Test
public void TeamTest() throws JSONException, IOException {
    assertEquals("Brazil", teamsObservableList.get(0).getName());}
```

Resulting in a success.



```
Tests passed: 1 of 1 test – 12 sec 966 ms

Test Results                    12 sec 966 ms
  NatScrappingUnitTest          12 sec 966 ms
    GetRankingTests             12 sec 966 ms
      TeamTest()                12 sec 966 ms

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...

Process finished with exit code 0

Tests passed: 1 (moments ago)
```

3.2. UefScrapping methods testing

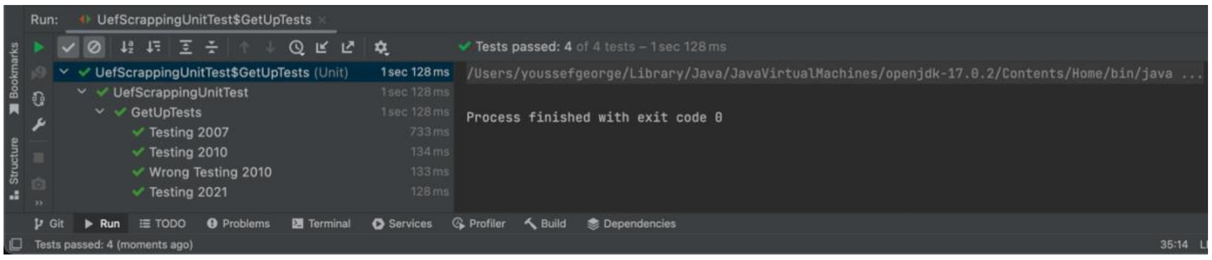
UefScrapping class includes 3 methods, ***getUp***() – ***getCountries***() – ***getUefRank***(), that need to be tested on their own as units. Therefore, ***UefScrappingUnitTest*** class include tests for the 3 mentioned methods as follows. Each of the three methods' test cases are grouped in a nested class.

3.2.1. GetUpTests class

Initially, we must create the nested class using @Nested. 3 test cases in our test suite for unit testing of this method.

```
@Nested
class GetUpTests {

    @DisplayName("Testing 2021")
    @Test
    public void getUpTest() throws IOException {
        Assertions.assertEquals("Last updated: 20/10/2021 09:14",
UefScrapping.getUp(2021));
    }

    @DisplayName("Testing 2007")
    @Test
    public void getUpTest2() throws IOException {
        assertEquals("Last updated: 19/06/2018 09:33",
UefScrapping.getUp(2007));
    }

    @DisplayName("Testing 2010")
    @Test
    public void getUpTest3() throws IOException {
        assertEquals("Last updated: 19/06/2018 09:32",
UefScrapping.getUp(2010));
    }

    @DisplayName("Wrong Testing 2010")
    @Test
    public void getUpTest4() throws IOException {
        assertNotEquals("Last updated: 19/06/2010 09:32",
UefScrapping.getUp(2010));
    }}
```

Resulting in four successes.



```
Run:  UefScrappingUnitTest$GetUpTests
✓ Tests passed: 4 of 4 tests – 1 sec 128 ms
✓ UefScrappingUnitTest$GetUpTests (Unit)   1 sec 128 ms
  ✓ UefScrappingUnitTest                   1 sec 128 ms
    ✓ GetUpTests                           1 sec 128 ms
      ✓ Testing 2007                        733 ms
      ✓ Testing 2010                        134 ms
      ✓ Wrong Testing 2010                  133 ms
      ✓ Testing 2021                        128 ms

/Users/youssefgeorge/Library/Java/JavaVirtualMachines/openjdk-17.0.2/Contents/Home/bin/java ...

Process finished with exit code 0

Tests passed: 4 (moments ago)
```

3.2.2. GetCountriesTests class

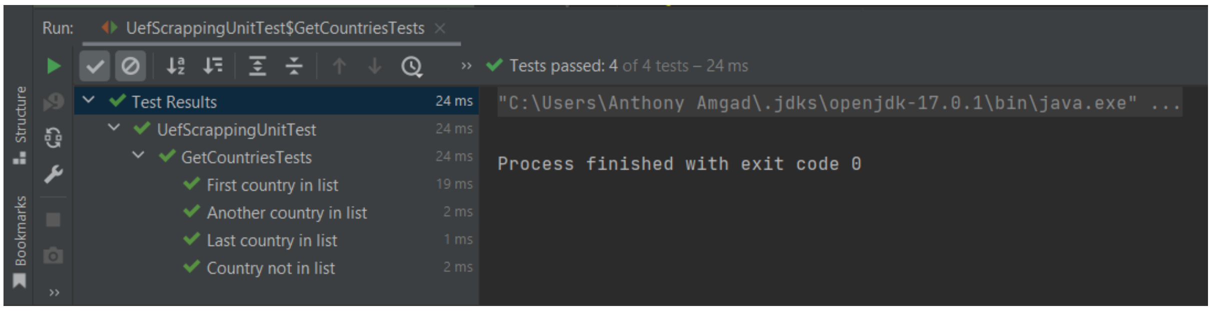
We setup the nested class using ***@Nested*** before using ***@BeforeAll*** to get the list we are going to test.

```
@Nested
class GetCountriesTests{
    static ObservableList<String> countriesObservableList;

    @BeforeAll
    public static void start() throws FileNotFoundException {
        countriesObservableList = UefScrapping.getCountries();
    }
```

4 test cases in our test suite for unit testing of this method.

```
@Test
@DisplayName("First country in list")
public void firstCountryTest() {
    assertEquals("All", countriesObservableList.get(0));
}

@Test
@DisplayName("Another country in list")
public void anotherCountryTest() {
    assertEquals("ENG", countriesObservableList.get(14));
}

@Test
@DisplayName("Last country in list")
public void lastCountryTest() {
    assertEquals("WAL", countriesObservableList.get(55));
}

@Test
@DisplayName("Country not in list")
public void countryNotInListTest(){
    assertFalse(countriesObservableList.contains("KKK"));
}
```

Resulting in 4 successes.



```
Run:  UefScrappingUnitTest$GetCountriesTests
✓ Tests passed: 4 of 4 tests – 24 ms

✓ Test Results                      24 ms
  ✓ UefScrappingUnitTest            24 ms
    ✓ GetCountriesTests             24 ms
      ✓ First country in list       19 ms
      ✓ Another country in list      2 ms
      ✓ Last country in list         1 ms
      ✓ Country not in list          2 ms

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...

Process finished with exit code 0
```

3.2.3. GetUefRankTests class

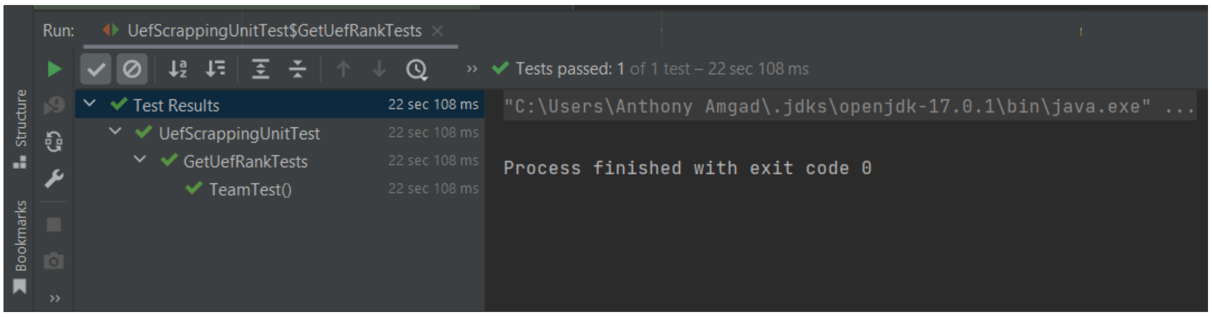
We setup the nested class using ***@Nested***, ***@ExtendWith(ApplicationExtension.class)***, before using ***@Start*** to get the list we are going to test.

```
@Nested
@ExtendWith(ApplicationExtension.class)
class GetUefRankTests{
    static ObservableList<UefTeam> teamsObservableList;
    @Start
    public static void start(Stage stage) throws IOException {
        teamsObservableList = UefScrapping.getUefRank(2007);}
```

Test case in our test suite for unit testing of this method.

```
@Test
public void TeamTest(){
    assertEquals("AC Milan", teamsObservableList.get(0).getName());
}
```

Resulting in a success.



Run: UefScrappingUnitTest$GetUefRankTests

Tests passed: 1 of 1 test – 22 sec 108 ms

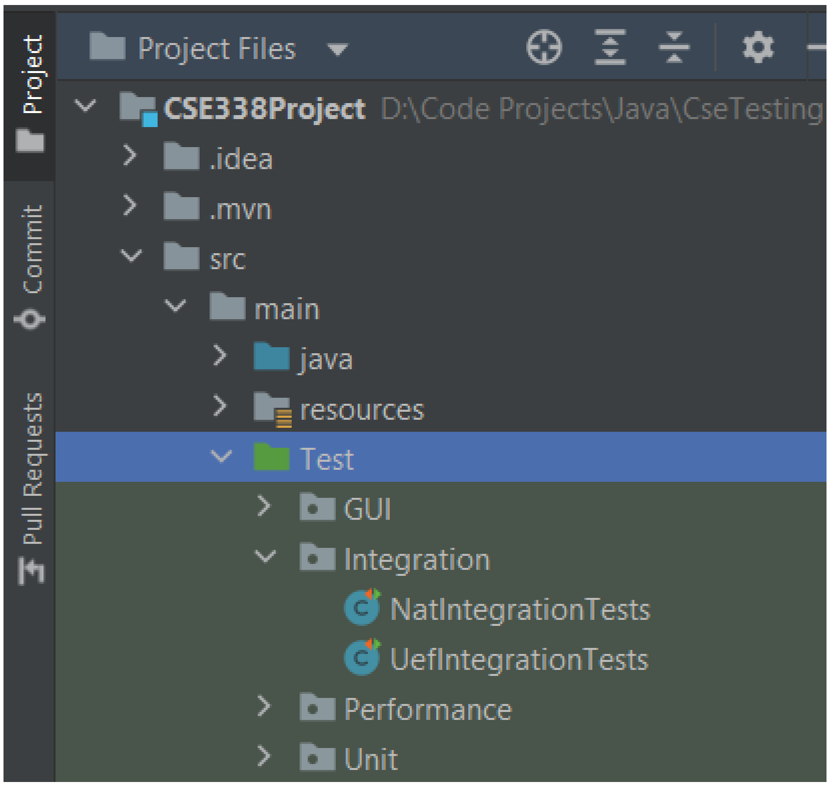
```
Test Results                 22 sec 108 ms
  UefScrappingUnitTest       22 sec 108 ms
    GetUefRankTests          22 sec 108 ms
      TeamTest()             22 sec 108 ms
```

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...

Process finished with exit code 0
```

4. INTEGRATION TESTING

After testing each unit alone using unit testing, integration testing comes in hand to test whether these units interact with each other as expected. Integration testing can be done in 2 ways, top-down using stubs, or bottom-up using drivers.

These tests can be found in 2 classes, NatIntegrationTests and UefIntegrationTests, under the Test module as shown in the following figure.



```
CSE338Project D:\Code Projects\Java\CseTesting
  .idea
  .mvn
  src
    main
      java
      resources
      Test
        GUI
        Integration
          NatIntegrationTests
          UefIntegrationTests
        Performance
        Unit
```

Figure 5: Integration Testing Package

Just like in unit testing, any testing class decorated with:

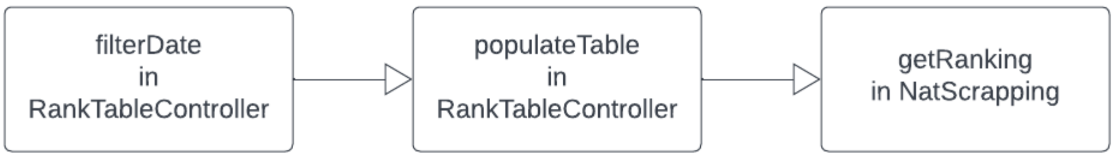
```
@ExtendWith(ApplicationExtension.class)
```

requires the following Java Runtime parameters when started:

```
--add-exports javafx.graphics/com.sun.javafx.application=ALL-UNNAMED
```

4.1. NatIntegrationTests

The following is the hierarchical dependency between functions that we will test.



4.1.1. Top-Down Testing

Top-Down testing of the national teams' part of the project will go as the following. A class for top-down testing is created:

```
@Nested
@ExtendWith(ApplicationExtension.class)
class TopDown{

    RankTableController rtc;

    @Start
    public void start(Stage stage) throws Exception {
        rtc = new RankTableController();
        rtc.datesFilter = new ComboBox<>();
        rtc.dates = NatScrapping.getDates();
        rtc.rankingTable = new TableView<>();
        rtc.continentFilter = new ComboBox<>();
}
```

First, we will test the filterDate function alone using a stub in place of the populateTable function. Stubs in our program are made so that if an underscore is placed after the intended function's name it will be replaced with its corresponding stub. Ex: function() --> function_(). This will result in changing the source code to look like the following:

```
public void filterDate() throws Exception {
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd MMM yyyy");
    LocalDate dateChoice = LocalDate.parse(datesFilter.getValue(), dtf);
    LocalDate feb2007 = LocalDate.parse("14 Feb 2007", dtf);
    LocalDate dec2002 = LocalDate.parse("18 Dec 2002", dtf);
    ArrayList<String> arr = new ArrayList<>(dates);
    int index = arr.indexOf(datesFilter.getValue());
    int datediff1 = Math.toIntExact(ChronoUnit.DAYS.between(feb2007,
dateChoice));
    int datediff2 = Math.toIntExact(ChronoUnit.DAYS.between(dec2002,
dateChoice));

    if (datediff1 >= 0) {
        populateTable_(datediff1 + 8079);
    } else if (datediff2 > 0) {
        populateTable_(arr.size() - index + 1);
    } else {
        populateTable_(arr.size() - index);}}
```

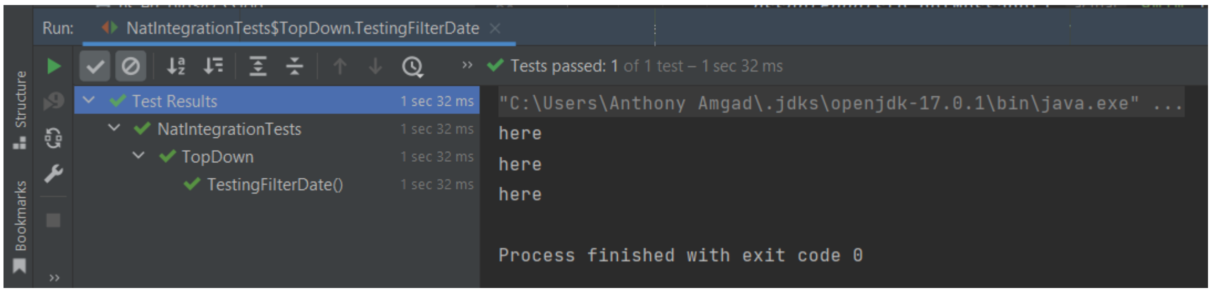
This is the created stub:

```
public void populateTable_(int i) throws Exception {
    throw new Exception(String.valueOf(i));}
```

This is our testing function:

```
@Test
void TestingFilterDate(){

    rtc.datesFilter.setValue("07 Feb 2019");
    try{
        rtc.filterDate();
    }catch (Exception e){
        assertEquals(e.getMessage(),"12455");
    }
    rtc.datesFilter.setValue("02 Sep 2009");
    try{
        rtc.filterDate();
    }catch (Exception e){
        assertEquals(e.getMessage(),"9010");
    }
    rtc.datesFilter.setValue("14 May 1997");
    try{
        rtc.filterDate();
    }catch (Exception e){
        assertEquals(e.getMessage(),"39");}}
```

Resulting in a success:


Run:
NatIntegrationTests$TopDown.TestingFilterDate
Tests passed: 1 of 1 test – 1 sec 32 ms
Test Results 1 sec 32 ms
NatIntegrationTests 1 sec 32 ms
TopDown 1 sec 32 ms
TestingFilterDate() 1 sec 32 ms
Structure
Bookmarks

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
here
here
here

Process finished with exit code 0
```

Next, we return the populateTable function in its position and replace the getRanking function with a stub. This will result in changing the source code to look like the following:

```
public void populateTable(int i){
    Task task = new Task() {
        @Override
        protected Object call() {
            try{
                rankingTable.getItems().clear();
                System.out.println("bef");
                rankingTable.setPlaceholder(new Label("Loading..."));
                datesFilter.setDisable(true);
                continentFilter.setDisable(true);
                ObservableList<NatTeam> li = NatScrapping.getRanking_(i);
                datesFilter.setDisable(false);
                continentFilter.setDisable(false);
                System.out.println("aft");
                MainLi = new ArrayList<>(li);
                rankingTable.setItems(li);
            }catch (Exception e){
                e.printStackTrace();
                rankingTable.setPlaceholder(new Label("Make sure you have a
stable internet connection."));
                datesFilter.setDisable(false);
                continentFilter.setDisable(false);
            }
            return null;}};
            new Thread(task).start();
}
```

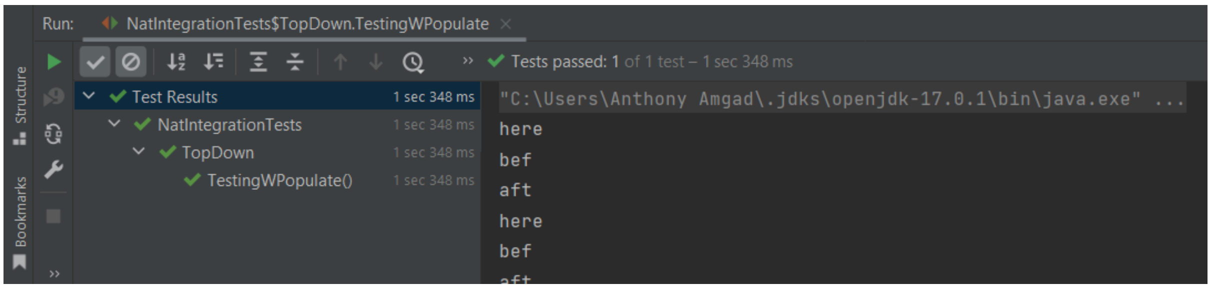
The created stub can be found in the NatScrapping class:

```
public static ObservableList<NatTeam> getRanking_(int i) throws
IllegalArgumentException {
    ObservableList<NatTeam> ls = observableArrayList();
    ls.add(new NatTeam(i));
    return ls;}
```

This is our Testing function:

```
@Test
void TestingWPopulate() throws Exception {

    rtc.datesFilter.setValue("07 Feb 2019");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"12455");

    rtc.datesFilter.setValue("02 Sep 2009");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"9010");

    rtc.datesFilter.setValue("14 May 1997");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"39");}
```

Resulting in a success:



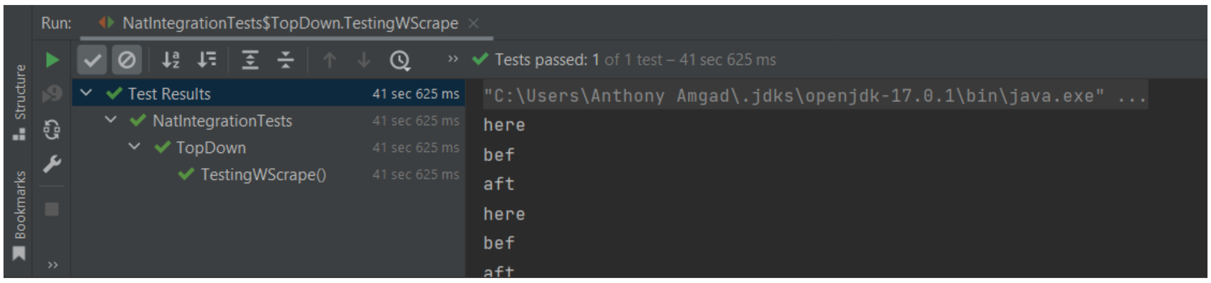
```
Run:  NatIntegrationTests$TopDown.TestingWPopulate
Tests passed: 1 of 1 test – 1 sec 348 ms

Test Results                 1 sec 348 ms
  NatIntegrationTests        1 sec 348 ms
    TopDown                  1 sec 348 ms
      TestingWPopulate()     1 sec 348 ms

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
here
bef
aft
here
bef
aft
```

The final step is returning any replaced function with a stub to its original state and testing one last time.

This is our testing function:

```java
@Test
void TestingWScrape() throws Exception {
    rtc.datesFilter.setValue("07 Feb 2019");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Belgium");

    rtc.datesFilter.setValue("12 Jan 2011");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Spain");

    rtc.datesFilter.setValue("14 May 1997");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Brazil");
}
```

Resulting in a success:



```
Run:  NatIntegrationTests$TopDown.TestingWScrape
Tests passed: 1 of 1 test – 41 sec 625 ms

Test Results                 41 sec 625 ms
  NatIntegrationTests        41 sec 625 ms
    TopDown                  41 sec 625 ms
      TestingWScrape()       41 sec 625 ms

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
here
bef
aft
here
bef
aft
```

4.1.2. Bottom-Up Testing

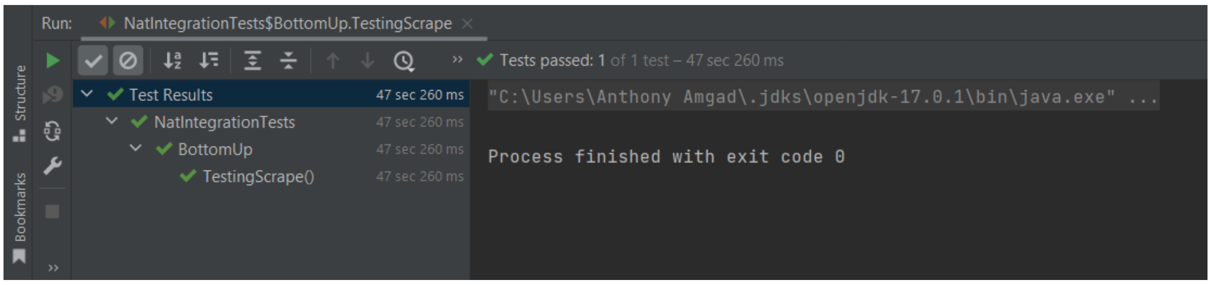
Bottom-up testing has its own nested class:

```
@Nested
@ExtendWith(ApplicationExtension.class)
class BottomUp{

    RankTableController rtc;


    @Start
    public void start(Stage stage) throws Exception {
        rtc = new RankTableController();
        rtc.datesFilter = new ComboBox<>();
        rtc.dates = NatScrapping.getDates();
        rtc.rankingTable = new TableView<>();
        rtc.continentFilter = new ComboBox<>();
    }
```

First, we will test the getRanking() function alone, using the test code as a driver for the function. This is our testing function:

```
@Test
void TestingScrape() throws Exception {
    ObservableList<NatTeam> obsli = NatScrapping.getRanking(39);
    assertEquals(obsli.get(0).getName(),"Brazil");
    obsli.clear();
    obsli = NatScrapping.getRanking(9507);
    assertEquals(obsli.get(0).getName(),"Spain");
    obsli.clear();
    obsli = NatScrapping.getRanking(12455);
    assertEquals(obsli.get(0).getName(),"Belgium");
}
```

Resulting in a success:



Run: NatIntegrationTests$BottomUp.TestingScrape

Tests passed: 1 of 1 test – 47 sec 260 ms

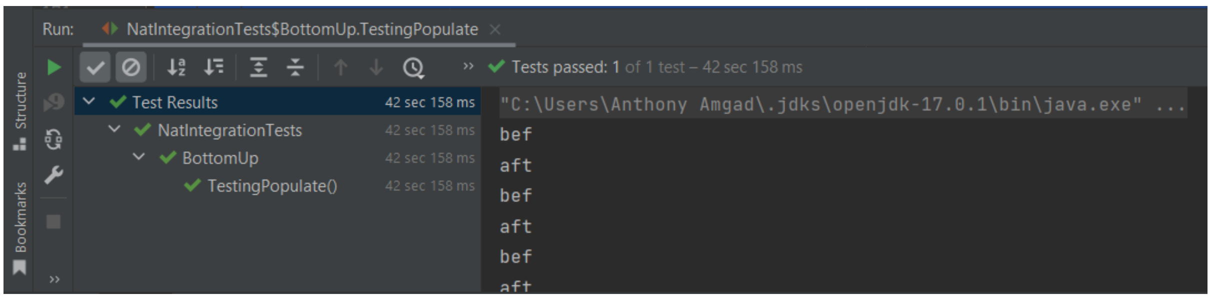
```
Test Results                 47 sec 260 ms
  NatIntegrationTests        47 sec 260 ms
    BottomUp                 47 sec 260 ms
      TestingScrape()        47 sec 260 ms
```

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...

Process finished with exit code 0
```

Next, we test the populateTable function which depends on the previous function we tested. This is our testing function code:

```java
@Test
void TestingPopulate(){
    rtc.populateTable(39);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Brazil");

    rtc.populateTable(9507);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Spain");

    rtc.populateTable(12455);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Belgium");

}
```

Resulting in a success:



Tests passed: 1 of 1 test – 42 sec 158 ms

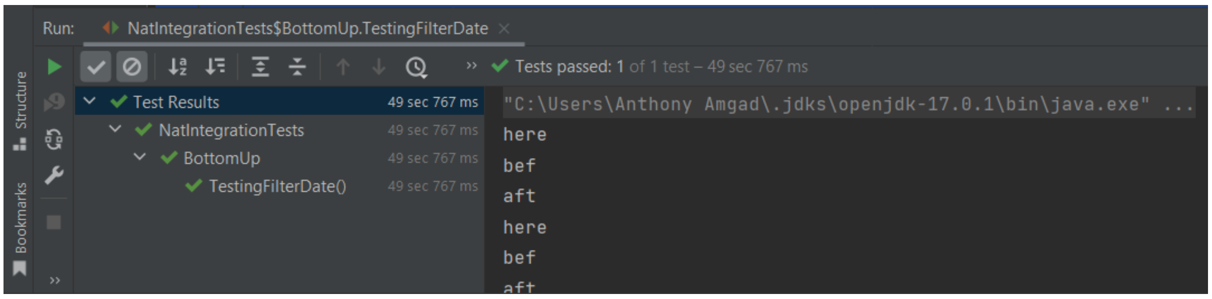
```
Test Results                42 sec 158 ms
  NatIntegrationTests       42 sec 158 ms
    BottomUp                42 sec 158 ms
      TestingPopulate()     42 sec 158 ms
```

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
bef
aft
bef
aft
bef
aft
```

Finally, we test the filterDate function ensuring that the system works.

```java
@Test
void TestingFilterDate() throws Exception {
    rtc.datesFilter.setValue("07 Feb 2019");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Belgium");

    rtc.datesFilter.setValue("12 Jan 2011");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Spain");

    rtc.datesFilter.setValue("14 May 1997");
    rtc.filterDate();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    assertEquals(rtc.rankingTable.getItems().get(0).getName(),"Brazil");
}
```

Resulting in a success:


Run: NatIntegrationTests$BottomUp.TestingFilterDate

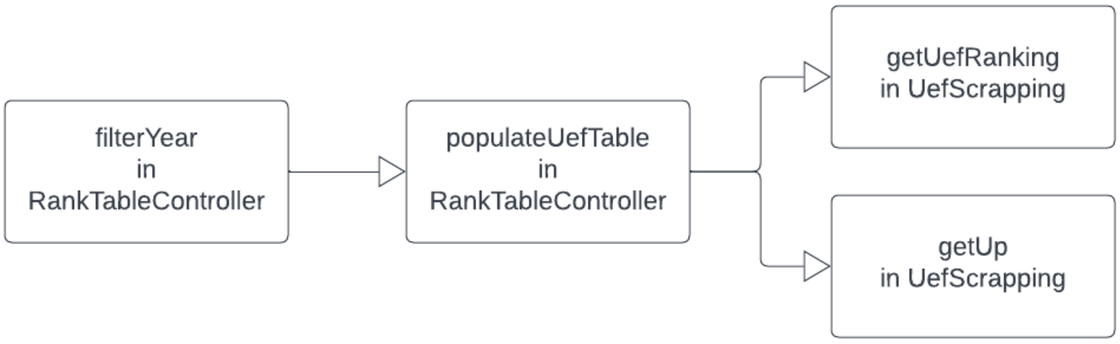
Tests passed: 1 of 1 test – 49 sec 767 ms

```
Test Results                 49 sec 767 ms
  NatIntegrationTests        49 sec 767 ms
    BottomUp                 49 sec 767 ms
      TestingFilterDate()    49 sec 767 ms
```

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
here
bef
aft
here
bef
aft
```

4.2. UefIntegrationTests

The following is the hierarchical dependency between functions that we will test.


filterYear in RankTableController
populateUefTable in RankTableController
getUefRanking in UefScrapping
getUp in UefScrapping

4.2.1. Top-Down Testing

A nested class for top-down testing is created:

```
@Nested
@ExtendWith(ApplicationExtension.class)
class TopDown{

    RankTableController rtc;

    @Start
    public void start(Stage stage) throws Exception {
        rtc = new RankTableController();
        rtc.yearFilter = new ComboBox<>();
        rtc.countries = UefScrapping.getCountries();
        rtc.UefarankingTable = new TableView<>();
        rtc.countryFilter = new ComboBox<>();
        rtc.labelLastUp = new Label();
        rtc.Utcp = new TableColumn<>();
        rtc.Utcp1 = new TableColumn<>();
        rtc.Utcp2 = new TableColumn<>();
        rtc.Utcp3 = new TableColumn<>();
        rtc.Utcp4 = new TableColumn<>();
    }
```

First, we will test the filterYear function alone using a stub in place of the populateUefTable function. Just like the NatIntegrationTests, stubs in our program are made so that if an underscore is placed after the intended function's name it will be relaced with its corresponding stub. Ex: function() --> function_(). This will result in changing the source code to look like the following:

```
public void filterYear() throws Exception {
    int yr = Integer.parseInt(yearFilter.getValue());
    populateUefTable_(yr);
}
```

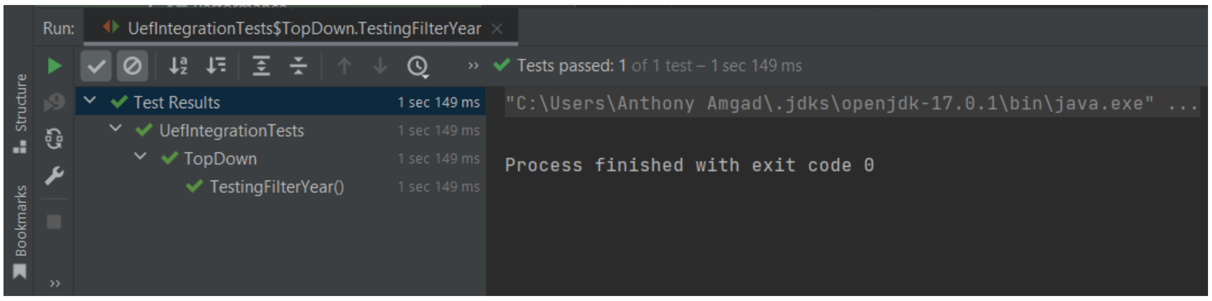
This is the created stub:

```
public void populateUefTable_(int i) throws Exception {
    throw new Exception(String.valueOf(i));
}
```

This is our test function:

```
@Test
void TestingFilterYear(){
    rtc.yearFilter.setValue("2020");
    try{
        rtc.filterYear();
    }catch (Exception e){
        assertEquals(e.getMessage(),"2020");
    }
    rtc.yearFilter.setValue("2010");
    try{
        rtc.filterYear();
    }catch (Exception e){
        assertEquals(e.getMessage(),"2010");
    }
    rtc.yearFilter.setValue("2005");
    try{
        rtc.filterYear();
    }catch (Exception e){
        assertEquals(e.getMessage(),"2005");
    }
}
```

Resulting in a success:



```
Run: UefIntegrationTests$TopDown.TestingFilterYear
Tests passed: 1 of 1 test – 1 sec 149 ms

Test Results                    1 sec 149 ms
  UefIntegrationTests           1 sec 149 ms
    TopDown                     1 sec 149 ms
      TestingFilterYear()       1 sec 149 ms

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...

Process finished with exit code 0
```

Next, we return the populateUefTable as it was by removing the '_' and then we replace the UefScrapping.getUp and UefScrapping.getUefRank with stubs changing the code of the populateUefTable function to look like this:

```
public void populateUefTable(int i){
    try{
        labelLastUp.setText(UefScrapping.getUp_(i));
    }catch (Exception e){
        labelLastUp.setText("Error");
    }
    Utcp.setText(String.valueOf(i));
    Utcp1.setText(String.valueOf(i-1));
    Utcp2.setText(String.valueOf(i-2));
    Utcp3.setText(String.valueOf(i-3));
    Utcp4.setText(String.valueOf(i-4));
    Task task = new Task() {
        @Override
        protected Object call() {
            try{
                UefarankingTable.getItems().clear();
                System.out.println("befff");
                UefarankingTable.setPlaceholder(new Label("Loading..."));
                yearFilter.setDisable(true);
                countryFilter.setDisable(true);
                ObservableList<UefTeam> li = UefScrapping.getUefRank_(i);
                yearFilter.setDisable(false);
                countryFilter.setDisable(false);
                System.out.println("afttt");
                MainUefLi = new ArrayList<>(li);
                UefarankingTable.setItems(li);
            }catch (Exception e){
                e.printStackTrace();
                UefarankingTable.setPlaceholder(new Label("Make sure you
have a stable internet connection."));
                labelLastUp.setText("###");
                yearFilter.setDisable(false);
                countryFilter.setDisable(false);
                Utcp.setText("-");
                Utcp1.setText("-");
                Utcp2.setText("-");
                Utcp3.setText("-");
                Utcp4.setText("-");
            }
            return null;
        }
    };

    new Thread(task).start();

}
```

The stubs can be found in the UefScrapping class:

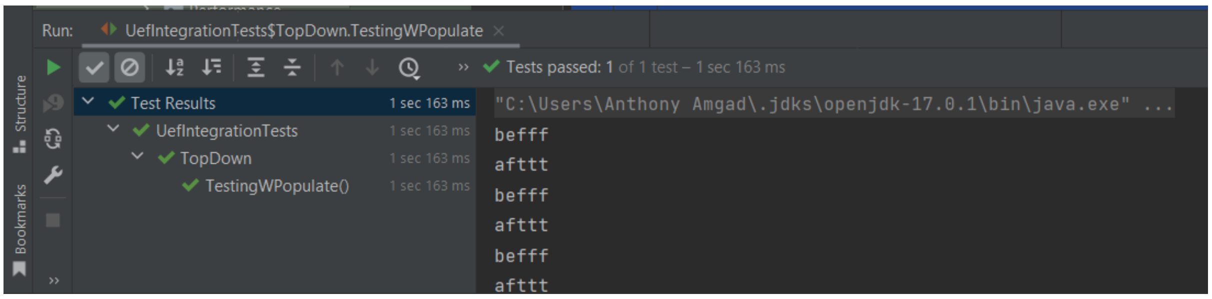
```
public static ObservableList<UefTeam> getUefRank_(int year) throws
IOException {
    ObservableList<UefTeam> li = FXCollections.observableArrayList();
    li.add(new UefTeam(year));
    return li;
}
```

```java
public static String getUp_(int year) throws IOException {
    return String.valueOf(year);
}
```

This is our test function:

```java
@Test
void TestingWPopulate() throws Exception {

    rtc.yearFilter.setValue("2020");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"2020");
    assertEquals(rtc.labelLastUp.getText(),"2020");

    rtc.yearFilter.setValue("2010");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"2010");
    assertEquals(rtc.labelLastUp.getText(),"2010");

    rtc.yearFilter.setValue("2005");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"2005");
    assertEquals(rtc.labelLastUp.getText(),"2005");

}
```

Resulting in a success:



UefIntegrationTests$TopDown.TestingWPopulate

Tests passed: 1 of 1 test – 1 sec 163 ms

```
Test Results                1 sec 163 ms
  UefIntegrationTests       1 sec 163 ms
    TopDown                 1 sec 163 ms
      TestingWPopulate()    1 sec 163 ms
```

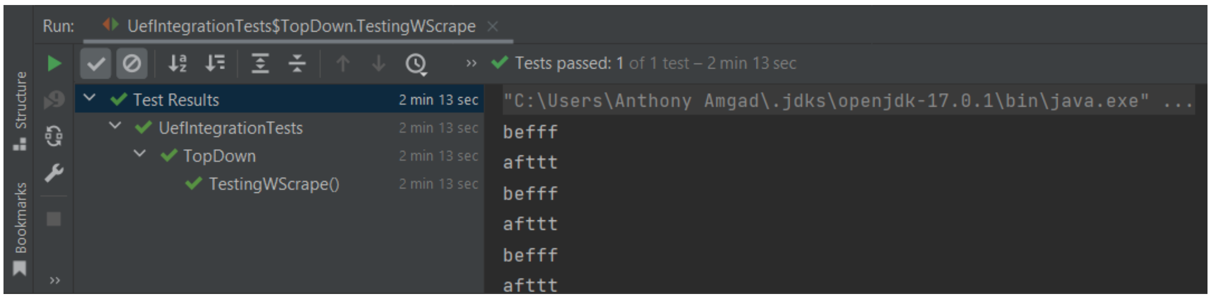
```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
befff
afttt
befff
afttt
befff
afttt
```

Finally, all stubs are removed, and the system is tested one final time with the following function:

```java
@Test
void TestingWScrape() throws Exception {
    rtc.yearFilter.setValue("2017");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"Real
Madrid CF");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:52");
```

```
    rtc.yearFilter.setValue("2010");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"FC
Barcelona");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:32");

    rtc.yearFilter.setValue("2005");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(1).getName(),"Valencia
CF");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:34");

}
```

Resulting in a success:



```
Run: UefIntegrationTests$TopDown.TestingWScrape
Tests passed: 1 of 1 test – 2 min 13 sec

Test Results                 2 min 13 sec
  UefIntegrationTests        2 min 13 sec
    TopDown                  2 min 13 sec
      TestingWScrape()       2 min 13 sec

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
befff
afttt
befff
afttt
befff
afttt
```

4.2.2. *Bottom-Up Testing*

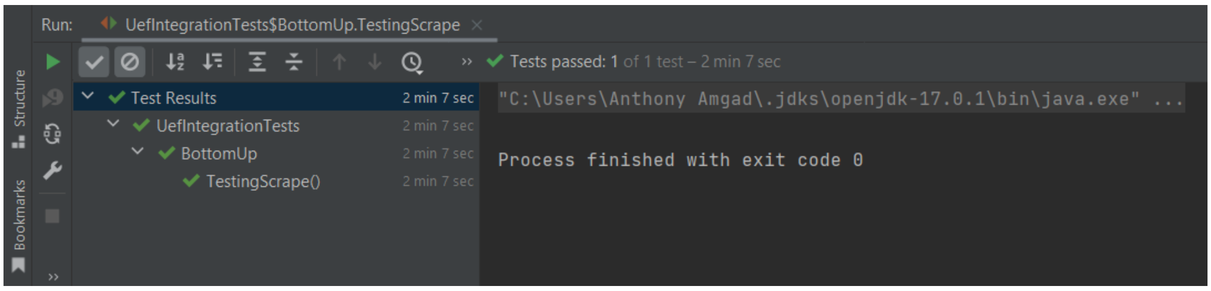
Bottom-up testing has its own nested class:

```
@Nested
@ExtendWith(ApplicationExtension.class)
class BottomUp{

    RankTableController rtc;


    @Start
    public void start(Stage stage) throws Exception {
        rtc = new RankTableController();
        rtc.yearFilter = new ComboBox<>();
        rtc.countries = UefScrapping.getCountries();
        rtc.UefarankingTable = new TableView<>();
        rtc.countryFilter = new ComboBox<>();
        rtc.labelLastUp = new Label();
        rtc.Utcp = new TableColumn<>();
        rtc.Utcp1 = new TableColumn<>();
        rtc.Utcp2 = new TableColumn<>();
        rtc.Utcp3 = new TableColumn<>();
        rtc.Utcp4 = new TableColumn<>();
    }
```

First, we will test the getUefRanking and getUp functions alone, using the test code as a driver for the function. This is our testing function:

```
@Test
void TestingScrape() throws Exception {
    ObservableList<UefTeam> obsli = UefScrapping.getUefRank(2017);
    assertEquals(obsli.get(0).getName(),"Real Madrid CF");
    obsli.clear();
    obsli = UefScrapping.getUefRank(2010);
    assertEquals(obsli.get(0).getName(),"FC Barcelona");
    obsli.clear();
    obsli = UefScrapping.getUefRank(2005);
    assertEquals(obsli.get(1).getName(),"Valencia CF");
    assertEquals(UefScrapping.getUp(2017),"Last updated: 19/06/2018
09:52");
    assertEquals(UefScrapping.getUp(2010),"Last updated: 19/06/2018
09:32");
    assertEquals(UefScrapping.getUp(2005),"Last updated: 19/06/2018
09:34");
}
```

Resulting in a success:

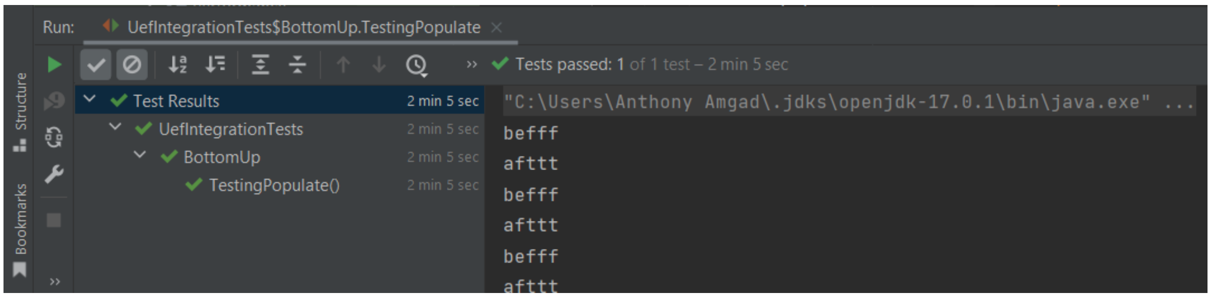


```
Run:   UefIntegrationTests$BottomUp.TestingScrape
Tests passed: 1 of 1 test – 2 min 7 sec

Test Results                2 min 7 sec
  UefIntegrationTests       2 min 7 sec
    BottomUp                2 min 7 sec
      TestingScrape()       2 min 7 sec

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...

Process finished with exit code 0
```

Next, we test the populateUefTable function which depends on the previous function we tested. This is our testing function code:

```
@Test
void TestingPopulate(){
    rtc.populateUefTable(2017);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"Real
Madrid CF");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:52");
    rtc.populateUefTable(2010);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"FC
Barcelona");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:32");
    rtc.populateUefTable(2005);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(1).getName(),"Valencia
CF");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:34");
}
```

Resulting in a success:



```
UefIntegrationTests$BottomUp.TestingPopulate
Tests passed: 1 of 1 test – 2 min 5 sec

Test Results                2 min 5 sec
  UefIntegrationTests       2 min 5 sec
    BottomUp                2 min 5 sec
      TestingPopulate()     2 min 5 sec

"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
befff
afttt
befff
afttt
befff
afttt
```

Finally, we test the filterYear function ensuring that the system works.

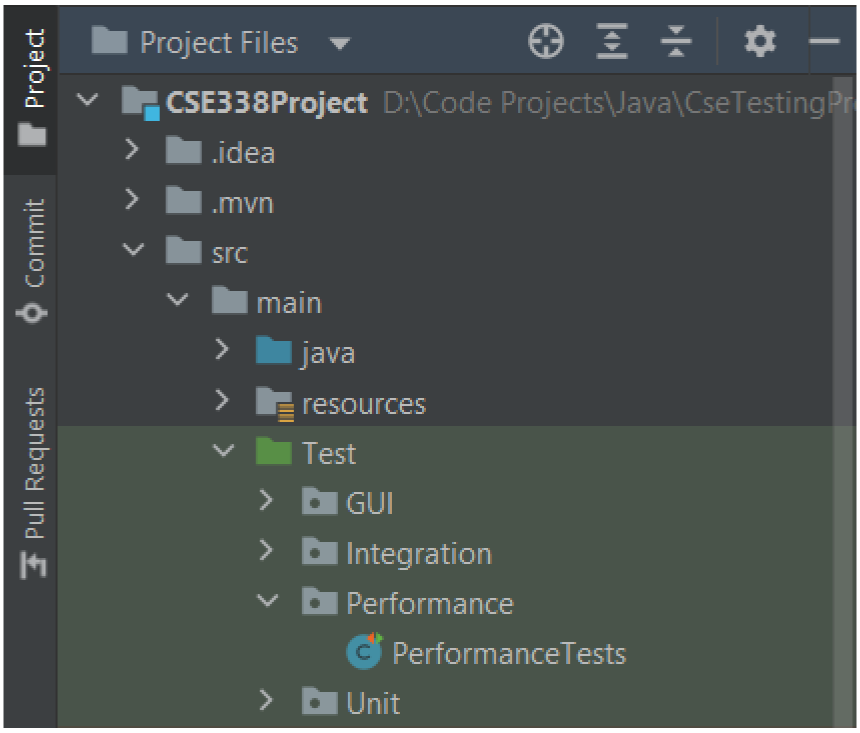
```
@Test
void TestingFilterYear() throws Exception {
    rtc.yearFilter.setValue("2017");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"Real
Madrid CF");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:52");

    rtc.yearFilter.setValue("2010");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(0).getName(),"FC
Barcelona");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:32");

    rtc.yearFilter.setValue("2005");
    rtc.filterYear();
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    assertEquals(rtc.UefarankingTable.getItems().get(1).getName(),"Valencia
CF");
    assertEquals(rtc.labelLastUp.getText(),"Last updated: 19/06/2018
09:34");

}
```

Resulting in a success:

5. PERFORMANCE TESTING

Any development process needs testing for the response time as well as data stress testing of the development product to be released. Since our data is set to be limited by FIFA and UEFA's databases then the amount of data for the program to handle has a set and fixed size. In addition to that, even if the size of data were to increase it will not be that much of an increase that it would break our developed product as it is currently in a safe and stable enough state.

Therefore, in this section we design and used tests to test our application's response time of fetching data from FIFA and UEFA's databases. These tests can be found in the PerformanceTests class, under the Performance module as shown in the following figure.



```
CSE338Project D:\Code Projects\Java\CseTestingPr
  .idea
  .mvn
  src
    main
      java
      resources
    Test
      GUI
      Integration
      Performance
        PerformanceTests
      Unit
```

Figure 6: Performance Testing Package

The class is created with the following structure:

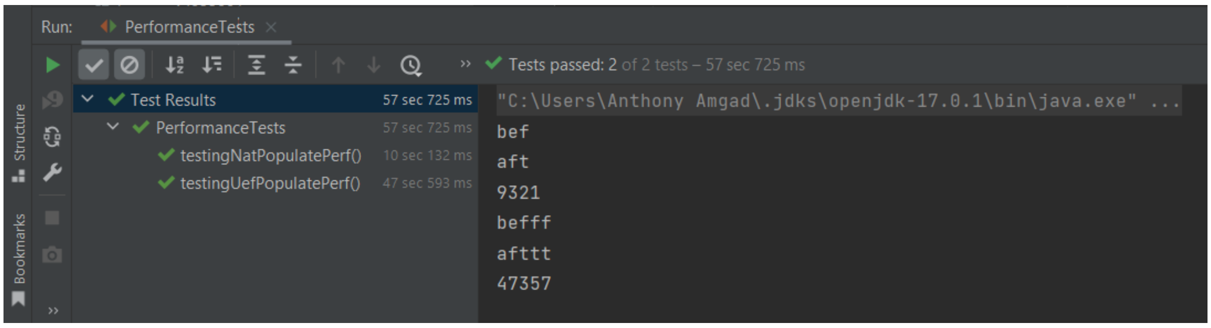
```
@ExtendWith(ApplicationExtension.class)
public class PerformanceTests {

    RankTableController rtc;

    @Start
    public void start(Stage stage) throws Exception {
        rtc = new RankTableController();
        rtc.datesFilter = new ComboBox<>();
        rtc.dates = NatScrapping.getDates();
        rtc.rankingTable = new TableView<>();
        rtc.continentFilter = new ComboBox<>();
        rtc.yearFilter = new ComboBox<>();
        rtc.countries = UefScrapping.getCountries();
        rtc.UefarankingTable = new TableView<>();
        rtc.countryFilter = new ComboBox<>();
        rtc.labelLastUp = new Label();
        rtc.Utcp = new TableColumn<>();
        rtc.Utcp1 = new TableColumn<>();
        rtc.Utcp2 = new TableColumn<>();
        rtc.Utcp3 = new TableColumn<>();
        rtc.Utcp4 = new TableColumn<>();
    }
```

With 2 test cases:

```
@Test
void testingNatPopulatePerf(){
    long startTime = System.currentTimeMillis();
    rtc.populateTable(1);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.rankingTable.getItems().size()!=0);
    long endTime = System.currentTimeMillis();
    long time = endTime - startTime;
    System.out.println(time);
    assertTrue(time < 30000);
}

@Test
void testingUefPopulatePerf(){
    long startTime = System.currentTimeMillis();
    rtc.populateUefTable(2022);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
rtc.UefarankingTable.getItems().size()!=0);
    long endTime = System.currentTimeMillis();
    long time = endTime - startTime;
    System.out.println(time);
    assertTrue(time < 60000);
}
```

The first test (testingNatPopulatePerf) is used to see the application's response time of fetching the data from FIFA's API and converting them to the application's class structure and making sure that time doesn't exceed 30 seconds. On the other hand, the second test (testingUefPopulatePerf) is used for testing the response time of fetching the data from UEFA's html and converting its scrapped data into the application's class structure, and since scrapping html is of course slower than a JSON API call, it is made sure that the time doesn't exceed 60 seconds.

These tests result in success:



```
Tests passed: 2 of 2 tests – 57 sec 725 ms

Test Results                      57 sec 725 ms
  PerformanceTests                57 sec 725 ms
    testingNatPopulatePerf()      10 sec 132 ms
    testingUefPopulatePerf()      47 sec 593 ms
```

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
bef
aft
9321
befff
afttt
47357
```

The tests also show that the numbers are way within range with FIFA's database process being finished in 9 seconds and UEFA's processing is finished in 47 seconds.

Obviously, these numbers vary from a device to another. However, if the device where the application is running is fairly modern, these numbers would still pass.

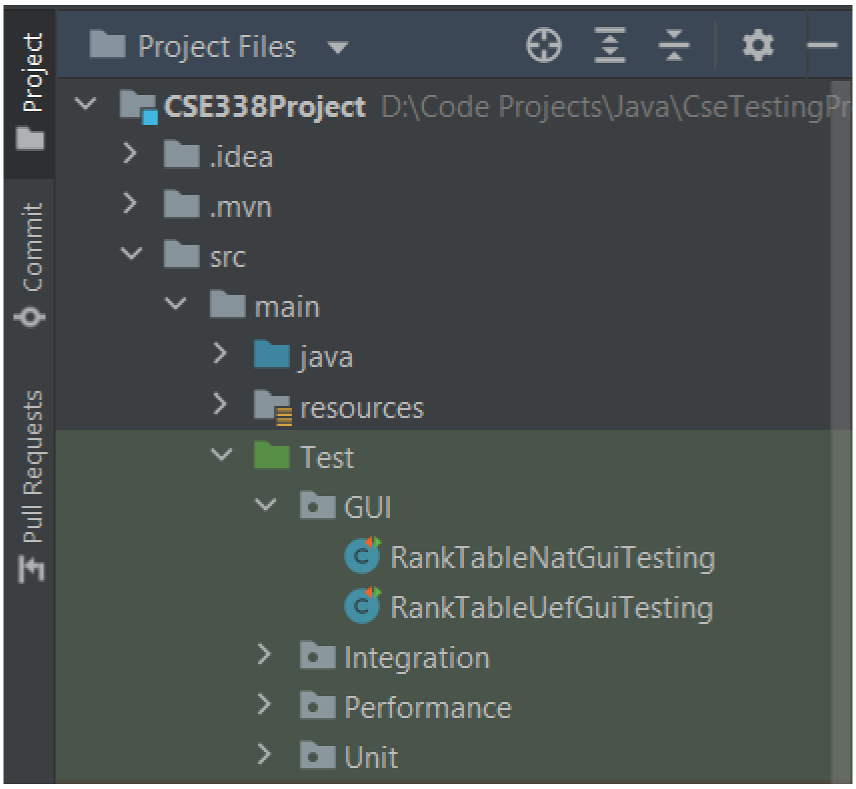
On older devices the program would still function, but the loading screen would appear for a longer time.

The numbers are considered acceptable, so the program passes the performance tests.

6. GUI TESTING

Any development process requires insurance that the GUI operates as intended using its inputs and outputs. This is done by starting a stage using the TestFx framework and then using the TestFx robot to interact with that GUI.

These tests can be found in the RankTableNatGuiTesting and RankTableUefGuiTesting classes, under the GUI module as shown in the following figure.

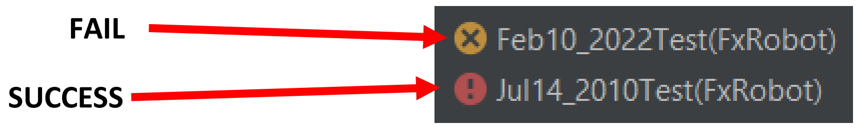


```
CSE338Project  D:\Code Projects\Java\CseTestingPr
  .idea
  .mvn
  src
    main
      java
      resources
      Test
        GUI
          RankTableNatGuiTesting
          RankTableUefGuiTesting
        Integration
        Performance
        Unit
```

Figure 7: GUI Testing Package

Another piece of information needed to be known is that there is a thread issue when changing the date fetched where JavaFX elements are altered in a normal Java Thread causing an exception to be raised. However, this exception can be completely disregarded as it does not affect the program's functionality.

This exception however will cause tests where the date is changed to "fail", but this can be differentiated from actually failing due to misfunctioning and that is by looking at the emblem that appears when testing. The '!' means that it's a success because of our small issue, and the 'x' means that it's a failure.



```
Feb10_2022Test(FxRobot)
Jul14_2010Test(FxRobot)
```

6.1. RankTableNatGuiTesting

This class tests the FIFA World Ranking part of the program. This class has the following structure:

```
@ExtendWith(ApplicationExtension.class)
public class RankTableNatGuiTesting {

    @Start
    private void start(Stage stage) throws IOException {
        stage.setScene(new
Scene(FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().ge
tResource("RankTable.fxml")))));
        stage.show();
    }
```

We have 2 inputs that we must test. First, the continent combo box. The following test function is made to test that:

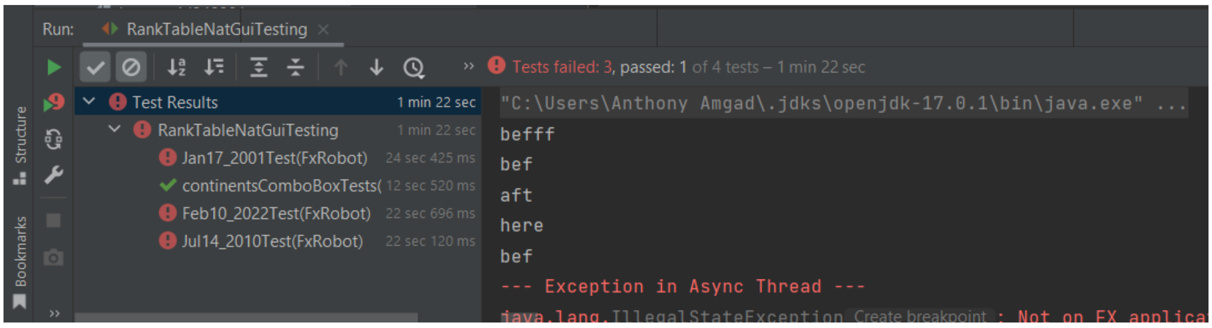
```
@Test
void continentsComboBoxTests(FxRobot robot) {
    TableView<NatTeam> ttb =
robot.lookup("#rankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> cbb =
robot.lookup("#continentFilter").queryAs(ComboBox.class);
    assertEquals(cbb.getItems().size(),7);
    int currtabsize = ttb.getItems().size();
    robot.interact(() -> cbb.getSelectionModel().select(0));
    assertEquals(currtabsize,ttb.getItems().size());
    for(int i = 1; i < 7; i++){
        final int k = i;
        robot.interact(() -> cbb.getSelectionModel().select(k));
        for(int j = 0; j < ttb.getItems().size(); j++){

assertEquals(ttb.getItems().get(j).getLocation(),cbb.getItems().get(i));
        }
    }
}
```

In addition to that, there are 3 test functions for the date combo box:

```
@Test
void Feb10_2022Test(FxRobot robot) {
    TableView<NatTeam> ttb =
robot.lookup("#rankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> mcb =
robot.lookup("#datesFilter").queryAs(ComboBox.class);
    assertEquals(mcb.getItems().size(),319);
    assertEquals(mcb.getItems().get(1), "10 Feb 2022");
    robot.interact(() -> mcb.getSelectionModel().select(1));
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size() != 0);
    assertEquals(ttb.getItems().get(0).getCountryCode(),"BEL");
    assertEquals(ttb.getItems().get(0).getTotalPoints(),1827.45);}
```

```
@Test
void Jul14_2010Test(FxRobot robot) {
    TableView<NatTeam> ttb =
robot.lookup("#rankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> mcb =
robot.lookup("#datesFilter").queryAs(ComboBox.class);
    assertEquals(mcb.getItems().size(),319);
    assertEquals(mcb.getItems().get(127), "14 Jul 2010");
    robot.interact(() -> mcb.getSelectionModel().select(127));
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size() != 0);
    assertEquals(ttb.getItems().get(8).getCountryCode(),"EGY");
    assertEquals(ttb.getItems().get(8).getTotalPoints(),1053);
}

@Test
void Jan17_2001Test(FxRobot robot) {
    TableView<NatTeam> ttb =
robot.lookup("#rankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> mcb =
robot.lookup("#datesFilter").queryAs(ComboBox.class);
    assertEquals(mcb.getItems().size(),319);
    assertEquals(mcb.getItems().get(238), "17 Jan 2001");
    robot.interact(() -> mcb.getSelectionModel().select(238));
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size() != 0);
    assertEquals(ttb.getItems().get(8).getCountryCode(),"YUG");
    assertEquals(ttb.getItems().get(8).getTotalPoints(),707);
}
```

These tests are made to check the data against FIFA's official website, and they result in success:



```
Tests failed: 3, passed: 1 of 4 tests – 1 min 22 sec

Test Results                              1 min 22 sec
  RankTableNatGuiTesting                  1 min 22 sec
    Jan17_2001Test(FxRobot)               24 sec 425 ms
    continentsComboBoxTests(              12 sec 520 ms
    Feb10_2022Test(FxRobot)               22 sec 696 ms
    Jul14_2010Test(FxRobot)               22 sec 120 ms
```

```
"C:\Users\Anthony Amgad\.jdks\openjdk-17.0.1\bin\java.exe" ...
befff
bef
aft
here
bef
--- Exception in Async Thread ---
java.lang.IllegalStateException Create breakpoint : Not on FX applica
```

6.2. *RankTableUefGuiTesting*

This class tests the UEFA Ranking part of the program. This class has the following structure:

```
@ExtendWith(ApplicationExtension.class)
public class RankTableUefGuiTesting {

    @Start
    private void start(Stage stage) throws IOException {
        stage.setScene(new
Scene(FXMLLoader.load(Objects.requireNonNull(getClass().getClassLoader().ge
tResource("RankTable.fxml")))));
        stage.show();}
```

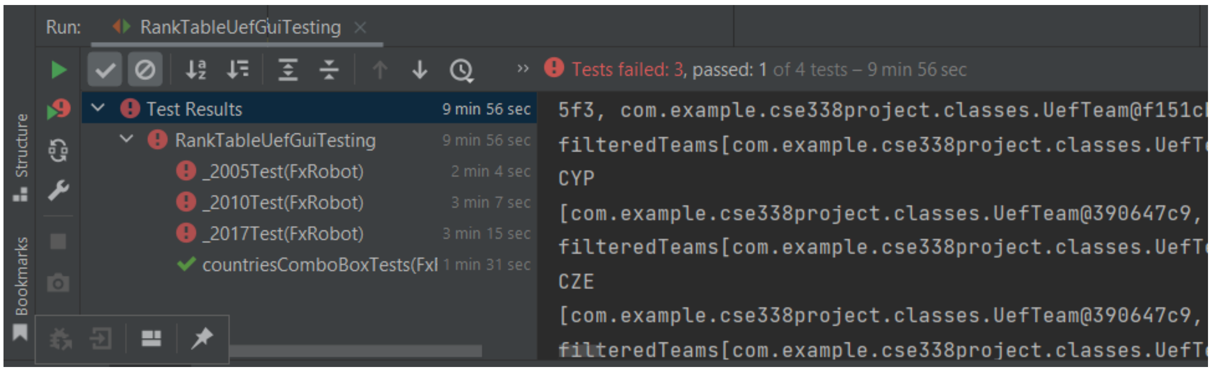
We have 2 inputs that we must test. First, the country combo box. The following test function is made to test that:

```
@Test
void countriesComboBoxTests(FxRobot robot) {
    robot.clickOn("#uefTab");
    TableView<UefTeam> ttb =
robot.lookup("#UefarankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> cbb =
robot.lookup("#countryFilter").queryAs(ComboBox.class);
    assertEquals(cbb.getItems().size(),56);
    int currtabsize = ttb.getItems().size();
    robot.interact(() -> cbb.getSelectionModel().select(0));
    assertEquals(currtabsize,ttb.getItems().size());
    for(int i = 1; i < 56; i++){
        final int k = i;
        robot.interact(() -> cbb.getSelectionModel().select(k));
        for(int j = 0; j < ttb.getItems().size(); j++){

assertEquals(ttb.getItems().get(j).getCountry(),cbb.getItems().get(i));
        }}}
```

In addition to that, there are 3 test functions for the year combo box:

```
@Test
void _2017Test(FxRobot robot) {
    robot.clickOn("#uefTab");
    TableView<UefTeam> ttb =
robot.lookup("#UefarankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> mcb =
robot.lookup("#yearFilter").queryAs(ComboBox.class);
    assertEquals(mcb.getItems().size(),18);
    assertEquals(mcb.getItems().get(5), "2017");
    robot.interact(() -> mcb.getSelectionModel().select(5));
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size() != 0);
    assertEquals(ttb.getItems().get(0).getName(),"Real Madrid CF");
    assertEquals(ttb.getItems().get(0).getTotalPoints(),"156.000");}
@Test
void _2010Test(FxRobot robot) {
    robot.clickOn("#uefTab");
    TableView<UefTeam> ttb =
robot.lookup("#UefarankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> mcb =
robot.lookup("#yearFilter").queryAs(ComboBox.class);
    assertEquals(mcb.getItems().size(),18);
    assertEquals(mcb.getItems().get(12), "2010");
    robot.interact(() -> mcb.getSelectionModel().select(12));
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size() != 0);
    assertEquals(ttb.getItems().get(0).getName(),"FC Barcelona");
    assertEquals(ttb.getItems().get(0).getTotalPoints(),"121.000");
}

@Test
void _2005Test(FxRobot robot) {
    robot.clickOn("#uefTab");
    TableView<UefTeam> ttb =
robot.lookup("#UefarankingTable").queryAs(TableView.class);
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size()!=0);
    ComboBox<String> mcb =
robot.lookup("#yearFilter").queryAs(ComboBox.class);
    assertEquals(mcb.getItems().size(),18);
    assertEquals(mcb.getItems().get(17), "2005");
    robot.interact(() -> mcb.getSelectionModel().select(17));
    Awaitility.await().atMost(2, TimeUnit.MINUTES).until(() ->
ttb.getItems().size() != 0);
    assertEquals(ttb.getItems().get(1).getName(),"Valencia CF");
    assertEquals(ttb.getItems().get(1).getTotalPoints(),"99.000");
}
```

These tests are made to check the data against UEFA’s official website, and they result in success:



```
Tests failed: 3, passed: 1 of 4 tests – 9 min 56 sec

Test Results                          9 min 56 sec
  RankTableUefGuiTesting              9 min 56 sec
    _2005Test(FxRobot)                2 min 4 sec
    _2010Test(FxRobot)                3 min 7 sec
    _2017Test(FxRobot)                3 min 15 sec
    countriesComboBoxTests(FxI        1 min 31 sec
```

```
5f3, com.example.cse338project.classes.UefTeam@f151c
filteredTeams[com.example.cse338project.classes.UefT
CYP
[com.example.cse338project.classes.UefTeam@390647c9,
filteredTeams[com.example.cse338project.classes.UefT
CZE
[com.example.cse338project.classes.UefTeam@390647c9,
filteredTeams[com.example.cse338project.classes.UefT
```