**CITS5501 Software Testing and Quality Assurance**

1. **project**

22302319 Yi Zhou

**JUnit tests**

1. **preconditions and postconditions**
   1. TravelStop class
      1. Preconditions:

1) In TravelStop constructor, 4 parameters are passed in.

2) The first parameter latitude is a double, the second parameter longitude is a double, the third parameter street is a string, the fourth parameter suburb is a string.

3) Street and suburb should not be null, and should exist in the on-disk file

* + 1. Postconditions:

1) getLatitude should return a double, getLongitude should return a double, getStreet should return a string, getSuburb should return a string.

2) the returned street and suburb should exist in the on-disk file

* 1. RoutePlanner class
     1. Preconditions:

1) In constructor RoutePlanner: parameter startLongitude, startLatitude, destinationLongitude, destinationLatitude should be double. Parameter leaveHour, leaveMinute, arriveHour, arriveMinute should be Integer.

2) arriveHour, arriveMinute(arrive time) and leaveHour, leaveMinute(leave time) cannot both be none-null.

3) arriveHour and arriveMinute should be null together or none-null together

4) leaveHour and leaveMinute should be null together or none-null together

5) In method getDirections, one String array parameter should be passed in, this String array should be empty.

* + 1. Postconditions:

1) getDirections method should return a list, which contains the instance of TravelStop class

2) the list contains at least 1 TravelStop

1. **identify test cases (process and two example test cases for each class)**

In these two classes, I will use input space partitioning to identify test cases and following is my process.

* 1. TravelStop class
     1. Identify testable functions: constructor TravelStop , getLatitude(), getLongitude(), getStreet(), getSuburb()
     2. Identify all parameters to the functions: double latitude, double longitude, String street, String suburb
     3. Model the input domain in terms of characteristics, each of which can be partitioned.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **variable** | **Valid partition** | **Invalid partition** | | | |
| latitude | Is double && latitude >= -90.0 &&latitude <= 90.0 | Is none-double | latitude < -90 | | latitude>90 |
| longitude | Is double && longitude >= -180 &&longitude <= 180 | Is none-double | Longitude<-180 | | Longitude>180 |
| street | Is String && street and suburb are both in the on-disk file | Is none-String | | not in the on-disk file | |
| suburb | Is String && street and suburb are both in the on-disk file | Is none-String | | not in the on-disk file | |

* + 1. Choose particular partitions, and values from within those partitions, and generate test cases

\* There is no on-disk file in this project, so that I will use four values to simulate this situation, “mystreet” , “mysuburb” is in this on-disk file and “nostreet”, “nosuburb” is not in this file.

Test case 1:

I selected one of the invalid partition block, the street is not in the on-disk file block for this test case.

test input:

TravelStop tstest1 = TravelStop(44.4, 55.5, “nostreet”, “mysuburb”);

expected result:

should throw an IllegalArgumentException

Test case 2:

I selected one of the invalid partition block, latitude>90 for this test case.

test input:

TravelStop tstest2 = TravelStop(92 , 55.5, “mysuburb”, “mysuburb”);

expected result:

should throw an IllegalArgumentException

* 1. RoutePlanner class
     1. Identify testable functions: constructor RoutePlanner, getStartLongitude(), getStartLatitude(), getLeaveHour(), getLeaveMinute(), getDestinationLongitude(), getDestinationLatitude(), getArriveHour(), getArriveMinute(), getDirections(String description[])
     2. Identify all parameters to the functions:

RoutePlanner: double startLongitude, double startLatitude, Integer leaveHour, Integer leaveMinute, double destinationLongitude, double destinationLatitude, Integer arriveHour, Integer arriveMinute

getDirections: String description[]

* + 1. Model the input domain in terms of characteristics, each of which can be partitioned.

Class constructor:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **variable** | **Valid partition** | | **Invalid partition** | | | | | |
| stratLongitude | Is double && startLatitude >= -90.0 & startlatitude <= 90.0 | | Is none-double | | startLatitude < -90 | | | startLatitude>90 |
| startLatitude | Is double && startLongitude >= -180 & startLongitude <= 180 | | Is none-double | | startLongitude<-180 | | | startLongitude>180 |
| destinationLongtitude | Is double && destinationLatitude >= -90.0 & destinationLatitude <= 90.0 | | Is none-double | | destinationLongitude<-90 | | | destinationLatitude>90 |
| destinationLatitude | Is double && destinationLongitude >= -180 & destinationLongitude <= 180 | | Is none-double | | destinationLongitude<-180 | | | destinationLongitude>180 |
| leaveHour | Is Integer &&  0 <= leaveHour <= 24 | leaveHour=null  && leaveMinute=null | Not  Integer | leaveHour<0 | | leaveHour>24 | leaveHour, leaveMinute, arriveHour,arriveMinute are none-null  together | |
| leaveMinute | Is Integer &&  0 <= leaveMinute <= 60 | leaveHour=null  && leaveMinute=null | Not  Integer | leaveMinute<0 | | leaveMinute>60 | leaveHour, leaveMinute, arriveHour,arriveMinute are none-null  together | |
| arriveHour | Is Integer &&  0 <= arriveHour <= 24 | arriveHour=null  && arriveMinute=null | Not  Integer | ArriveHour<0 | | ArriveHour>24 | leaveHour, leaveMinute, arriveHour,arriveMinute are none-null  together | |
| arriveMinute | Is Integer &&  0 <= arriveMinute <= 60 | arriveHour=null  && arriveMinute=null | Not  Integer | ArriveMinute<0 | | ArriveMinute>60 | leaveHour, leaveMinute, arriveHour,arriveMinute are none-null  together | |

getDirections method:

|  |  |  |
| --- | --- | --- |
| **variable** | **Valid partition** | **Invalid partition** |
| description[] | is empty | is none-empty |

test case 3:

I selected one of the invalid partition block, leaveHour, leaveMinute, arriveHour,arriveMinute are none-null together for this test case.

expected input:

RoutePlanner rptest1 = new RoutePlanner(22.33, 44.55, 6, 22, 87.88, 55.55, 8, 55 );

expected result:

an IllegalArgumentException should be thrown

test case 4:

I selected one of the invalid partition block of getDirections method, description[] is none-empty.

test input:

RoutePlanner rptest2 = new RoutePlanner(22.33, 44.55, null, null, 87.88, 55.55, 8, 55 );

String description[] = {“ssss”, “ssssd”};

rptest2.getDirctions()

expected result:

an IllegalArgumentException should be thrown

**test type: these tests are unit tests, because they only test one single method of the whole class.**

1. **three more test cases for each class**

TravelStop class

test case 5:

description: I selected one of the invalid partition block, longitude is none-double for this test case.

            test input:

TravelStop tstest3 = TravelStop(333.77 , “String”, “mystreet”,  “mysuburb”);

      expected result:

should throw an data-type error

test case 6:

     description: test the getStreet() method, return right value.

                 test input:

     TravelStop tstest4 = TravelStop(333.77 , 6666, “mystreet”,  “mysuburb”);

      tstest4.getStreet();

                  expected result:

                  should return “mystreet”

test case 7:

     description: test the getSuburb() method, return right value

                 test input:

     TravelStop tstest5 = TravelStop(333.77 , 6666, “mystreet”,  “mysuburb”);

      tstest4.getSubrub();

                  expected result:

      should return “mysubrub”

RoutePlanner class

I selected one of the invalid partition block, arriveHour and arriveMinute are not both null for this test case.

             test case 8:

      test input:

       RoutePlanner rptest3 = new RoutePlanner(22.33, 44.55, 6, 22, 87.88, 55.55, 8, null );

expected result:

an IllegalArgumentException is thrown

test case 9:

description: test the getStartLatitude() method, return right value

   test input:

     RoutePlanner rptest4 = new RoutePlanner(22.33, 44.55, null, null, 87.88, 55.55, 8, 55 );

  rptest4.getStartLatitude()

  expected result:

  should return 44.55

test case 10:

description: test the getStartLongitude() method, return right value

   test input:

     RoutePlanner rptest5 = new RoutePlanner(22.33, 44.55, null, null, 87.88, 55.55, 8, 55 );

  rptest4.getStartLongitude()

  expected result:

  should return 22.33

1. **Junit tests (5 test cases)**

**In the java file folder**

1. **When to stop testing and how to identify further tests**

I will check the following points in my project, if most of these answers are yes, I will decide I have written enough tests:

* Are all test cases executed at least once?
* Is the test case pass rate (may be 95%) as defined?
* Is complete test coverage achieved?
* Are all functional flows executed at least once?
* Is the decided defect count reached?
* Are all Major High Priority Defects fixed and closed?
* Have all Defects been retested and closed?
* Has regression been done for all open defects?

I can also use fault seeding method. Deliberately implant several faults in my project. If my tests reveal 100% of the implanted faults, we are more confident that our tests are adequate.

Fault seeding method can also be used to identify further tests. So, if some of the implanted faults are not found, it means I need write more tests for these defects.

Mutation test can also be used to identify further tests, if I mutate my part of the code in my program, and it seems not to be wrong with this part of my code, it means that I need more tests in this part.

For instance, there is an on-disk file which is used to check the existence of suburb and street. I will change or delete some suburb, street names in this file and test again to find whether it becomes wrong with constructing some not-exist suburbs and streets.

**Alloy model**

**In the planner.als file**