Group Project Report - Test Cases

CS3343

Group 13



Group Members:

|  |  |  |
| --- | --- | --- |
| Full Name | SID | Role |
| Luka Moderc | 55594415 | Project Manager |
| Andela Basic | 55594403 | Assistant Project Manager |
| Uros Cvijanovic | 55304127 | Testing Engineer |
| Phudis Dawieang | 55411086 | Programmer |
| Balaji Varun Aditya | 55304510 | Programmer |
| Jeffers Chan | 55606049 | Testing Engineer |

Test Report

# Table of Contents

Test Cases

FlowHierarchy Diagram

Methodology

Coverage Analysis

Unit Testing

1. FloydWarshall.java

1.1. Coverage for Unit Testing for FloydWarshall.java

1.2. Test cases for Unit testing for FloydWarshall.java

1.3. Test cases Explanation

#### 2. User.java

1.1. Coverage for Unit Testing for User.java

1.2. Test cases for Unit testing for User.java

1.3. Test cases Explanation

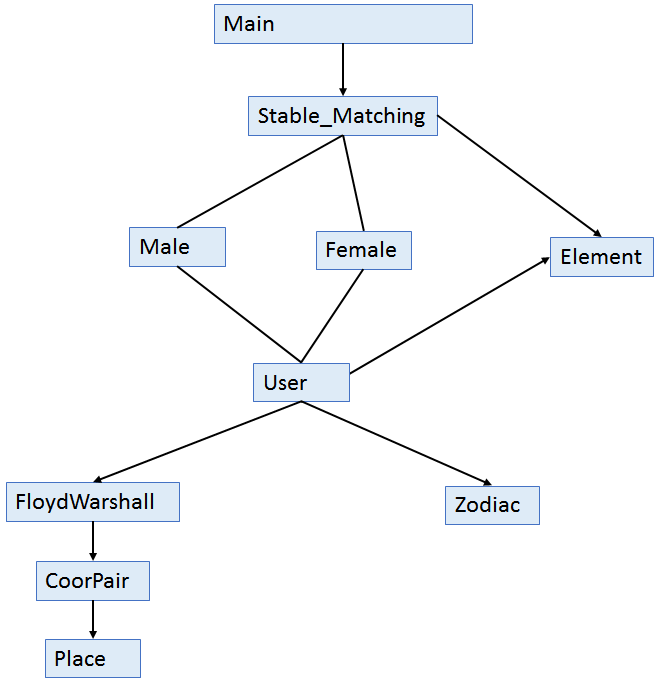
Integration Testing

1. Coverage for Integration testing

2. Test Cases for Integration testing

3. Test Cases Explanation

### FlowHierarchy Diagram

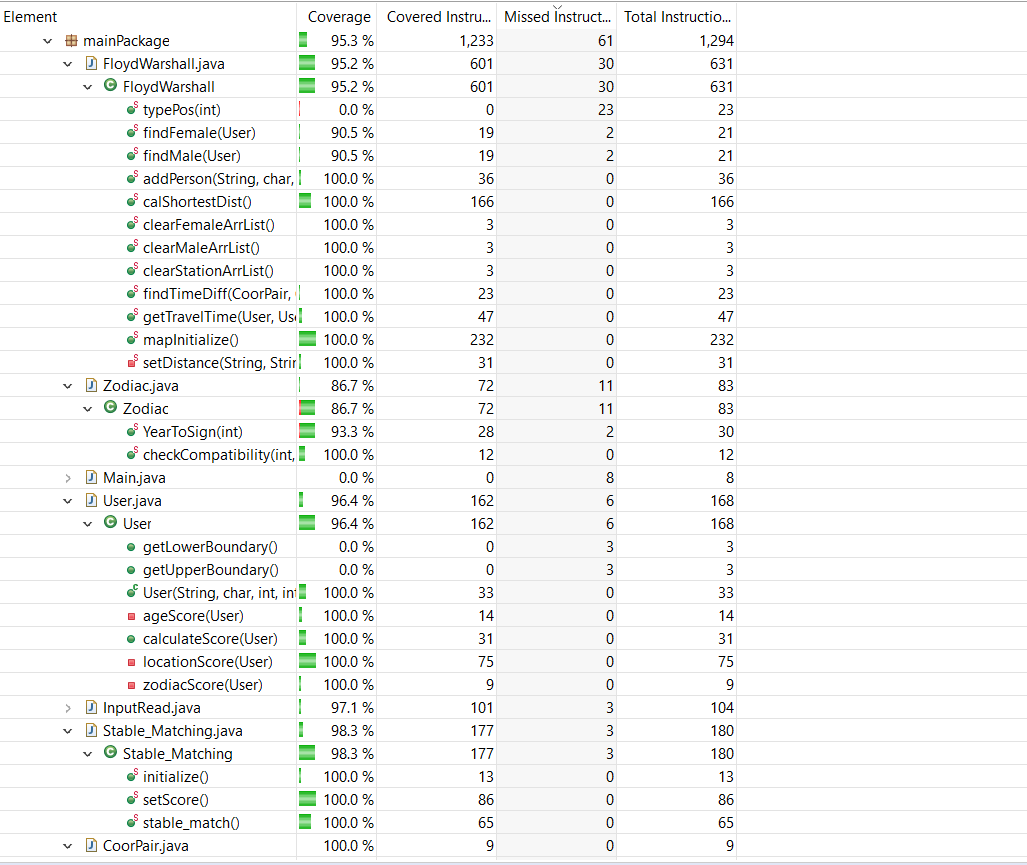


### Methodology

We chose the bottom-up testing method for the testing. The main reason for choosing this method is the generation of test cases and the observation of the results. The main disadvantage of the bottom-up approach is that the whole project has to be completed before the testing is done. However, in our case, the main class containing the stable matching algorithm is the last method developed and the whole project was sent for testing after it is done. For this reason and also the simplicity we chose the Bottom-Up method.

### 

### Coverage Analysis



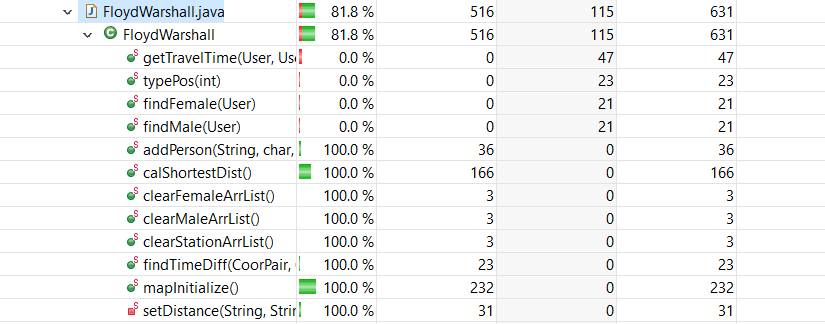
### Unit Testing

1. FloydWarshall.java

The first class-tested is FloydWarshall.java class explained in the previous section. The main idea was to extensively test the calShortestDist() method that calculates the shortest distance between the pairs. The coverage for unit testing of FloydWarshall.java is not 100% as some methods from that classes are implemented and tested in the later testing process (eg. getTravelTime(User, User) the method called from the method in User.java).

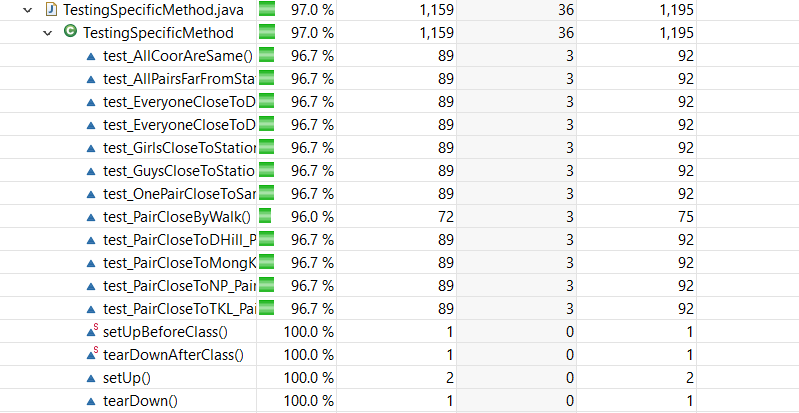
1.1 The coverage for Unit Testing of FloydWashall.java:





1.2. Test cases for FloydWarshall.java:

**All test cases contain the real coordinates of residential buildings based in Hong Kong taken from Google Maps.**



1.3. Test Cases Explanation:

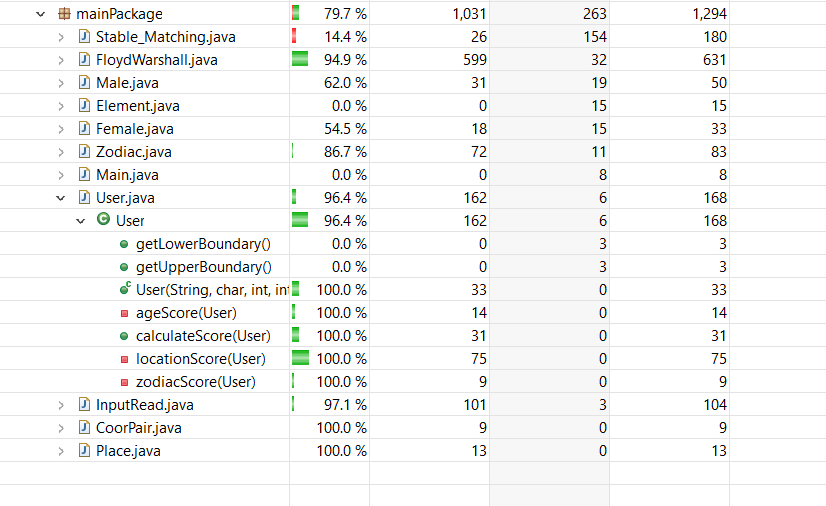
* test\_AllCoorAreSame() - this test case is testing the distance when all the pairs have the same coordinates
* test\_GuysCloseToStationGirlsFar() - this test case is testing when both boys are close to MTR stations but girls are far
* test\_GirlsCloseToStationGuysFar() - this test case is testing when girls are close to MTR station but boys are far
* test\_OnePairCloseToSameStationOneFar() - this test case is testing when one pair is close to same station and other is far from the same station
* test\_AllPairsFarFromStation() - this test case is testing when all pairs are far from MTR station
* test\_PairCloseByWalk() - this test case is testing when the pair is closer by walk than the MTR station
* test\_PairCloseToMongKok\_PairCloseToCentral() - this test case is testing when one pair is close to Mong Kok MTR station and the other is close to Central station
* test\_PairCloseToDHill\_PairCloseToKT() - this test case is testing when one pair is close to Diamond Hill MTR station and the other is close to Kowloon Tong station
* test\_PairCloseToNP\_PairCloseToLK() - this test case is testing when one pair is close to North Point MTR station and the other is close to Lai King station
* test\_PairCloseToTKL\_PairCloseToHH() - this test case is testing when one pair is close to Tiu Keng LengMTR station and the other is close to Hung Hum station
* test\_EveryoneCloseToDifferentStation() - this test case is testing when everyone is close to different station
* test\_EveryoneCloseToDifferentStation1() - this test case is testing when everyone is close to different station (different combination of stations)
* testCase\_GirlsAndBoysHaveSameCoordinates() - this test case is testing when girls and boys have same coordinates
* test\_OneBoyAndOneGirlSameCoordinates() - this test case is testing when one pair have the same coordinates and others also have the same coordinates

#### 

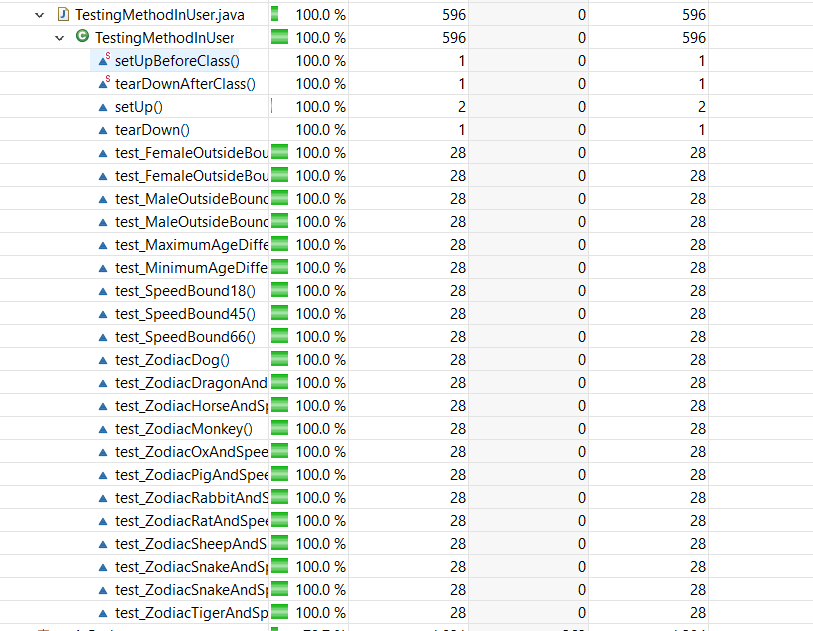
2. User.java

The second class I decided to do unit testing on was the User.java class. The method I focused on is calculateScore() method that is called by a User on another User and calculates the score of compatibility based on parameters. At first, that was one big function but after code refactoring, it was separated into several methods inside User.java but all of those methods are called from calculateScore(), therefore testing that function allowed me to cover all methods in User.java.

2.1. Coverage for Unit Testing of User.java is:



2.2. Test Cases For User.java:



2.3. Test Cases Explanation:

User.calculateScore(User):

Cases when One User is outside the age preference for others:

* test\_FemaleOutsideBoundForMale\_lower() - this test case is testing when Female is outside the age boundary for Male, is younger than the girl prefers
* test\_FemaleOutsideBoundForMale\_higher() - this test case is testing when Female is outside the age boundary for Male, is older than the girl prefers
* test\_MaleOutsideBoundForFemale\_higher() - this test case is testing when Male is outside the age boundary for Female, is older than the girl prefers
* test\_MaleOutsideBoundForFemale\_lower() - this test case is testing when Male is outside the age boundary for Female, is younger than the girl prefers

Cases for the maximum and minimum difference allowed by bounds:

* test\_MaximumAgeDifferenceAllowedByAgeBounds() - this test case is testing the Maximum difference allowed by age boundaries
* test\_MinimumAgeDifferenceAllowedByAgeBounds() - this test case is testing the Minimum difference allowed by age boundaries

User.calculateScore(User), User.ocationScore(User), User.zodiacScore(User) - These are combinations between the Zodiac signs and Boundary Values for Age that affect the speed of the person, hence affecting the overall score. As the distance between the pairs will take up the largest percentage of the score, to test how other parameters affect the score it is assumed that location scores are uniform, furthermore, the distance has already been tested in previous Unit testing.

* test\_ZodiacHorseAndSpeedBound30() - this test case is testing when the Zodiac sign is Horse and age is 30
* test\_ZodiacSnakeAndSpeedBound19() - this test case is testing when the Zodiac sign is Snake and age is 19
* test\_ZodiacMonkey() - this test case is testing when the Zodiac sign is Monkey
* test\_ZodiacDog() - this test case is testing when the Zodiac sign is Dog
* test\_ZodiacPigAndSpeedBound25() - this test case is testing when the Zodiac sign is Pig and age is 25
* test\_ZodiacRatAndSpeedBound24() - this test case is testing when the Zodiac sign is Rat and age is 24
* test\_ZodiacOxAndSpeedBound23() - this test case is testing when the Zodiac sign is Ox and age is 23
* test\_ZodiacSnakeAndSpeedBound31() -this test case is testing when the Zodiac sign is Snake and age is 31
* test\_ZodiacSheepAndSpeedBound65() -this test case is testing when the Zodiac sign is Sheep and age is 65
* test\_ZodiacDragonAndSpeedBound32() -this test case is testing when the Zodiac sign is Dragon and age is 32
* test\_ZodiacRabbitAndSpeedBound69() -this test case is testing when the Zodiac sign is Rabbit and age is 69
* test\_ZodiacTigerAndSpeedBound70() -this test case is testing when the Zodiac sign is Tiger and age is 70
* test\_SpeedBound18() - this test case is testing when age is 18
* test\_SpeedBound45() - this test case is testing when age is 45
* test\_SpeedBound66() -this test case is testing when age is 66

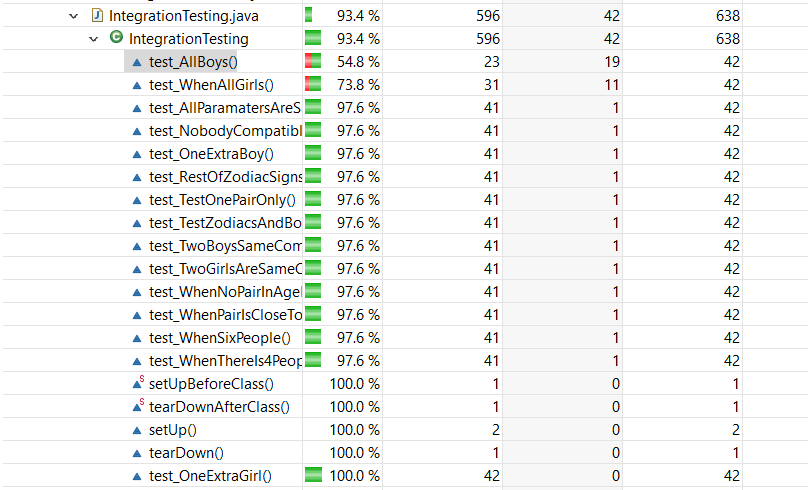
### Integration Testing

For the Integration Testing, we decided to test the StableMaching.java class as that method calls all the important methods from other classes therefore basically going through all processes of calculating distance (FloydWarshall.calculateShortestDist()), calculates Score for each pair (User.calculateScore()) and finally does the stable matching algorithm and returns most suitable pairs. I used a bottom-up approach for the simple reason that it is the easiest to implement starting from the bottom and going up without the need for use of stub classes as the code was already complete when I got it for testing.

1. Coverage for Integration Testing:

#### 

1. Test Cases For Integration Testing:



1. Test cases Explanation:

* test\_WhenThereIs4People() - this test case is testing when there is 4 people in the input file
* test\_WhenAllGirls() - this test case is testing when there is all girls in the input file
* test\_WhenSixPeople() - this test case is testing when there is 6 people in the input file
* test\_OneExtraBoy() - this test case is testing when there is one extra boy in the input file
* test\_OneExtraGirl() - this test case is testing when there is one extra girl in the input file
* test\_TestZodiacsAndBoundaryValuesForSpeed\_10People() -
* test\_RestOfZodiacSignsAndBoundariesForSpeed\_8People() -
* test\_WhenPairIsCloseToEachOtherButNotInsideAgeBoundary() -
* test\_WhenNoPairInAgeBoundaryAndAllPeopleFarFromEachOther() -
* test\_AllBoys() - this test case is testing when there is all boys in the input file
* test\_TwoGirlsAreSameCompatibleForOneGuy() - test case when two girls have the same parameters that are compatible with one guy but not other
* test\_TwoBoysSameCompatibleForOneGirls() - test case when two boys have the same parameters that are compatible with one girl but not other
* test\_NobodyCompatibleWithEachOther() - this test case is testing when nobody have parameters that are compatible with others
* test\_AllParamatersAreSameExceptNames() - this test case is testing when all parameters are same for all people except names
* test\_TestOnePairOnly() - this test case is testing when there is only one boy and one girl in the input file