CSI3105 - Software Testing

Assignment 2 - Software Testing Project

Edith Cowan University – Joondalup

Authors

Stout-Spykers, Q. 10223002

Wagner, C. P. 10196193

Wimbridge, J. J. 10143966

Contents

[Testing Strategy 4](#_Toc484414441)

[Approach 4](#_Toc484414442)

[Process 4](#_Toc484414443)

[Stages of testing progress 5](#_Toc484414444)

[How to execute test cases 7](#_Toc484414445)

[1. Go to configure build paths 7](#_Toc484414446)

[2. Add archives 7](#_Toc484414447)

[3. Select and Run test 8](#_Toc484414448)

[Test plan 9](#_Toc484414449)

[Unit test suite 9](#_Toc484414450)

[Meeting - Jarryd 9](#_Toc484414451)

[Set 1 14](#_Toc484414452)

[Set 2 15](#_Toc484414453)

[Calendar – Jarryd 16](#_Toc484414454)

[Room - QJ 25](#_Toc484414455)

[Person - QJ 26](#_Toc484414456)

[Organization - QJ 27](#_Toc484414457)

[Planner - QJ 28](#_Toc484414458)

[Integration test suite 34](#_Toc484414459)

[Subsystem: Calendar & Meeting - Carl 34](#_Toc484414460)

[Subsystem: Calendar & Meeting & Person - Carl 38](#_Toc484414461)

[Subsystem: Meeting, Calendar & Room - Jarryd 42](#_Toc484414462)

[Test Cases - Jarryd 49](#_Toc484414463)

[Results 51](#_Toc484414464)

[Unit Test Executions 51](#_Toc484414465)

[Meeting 51](#_Toc484414466)

[Calendar 52](#_Toc484414467)

[Planner 53](#_Toc484414468)

[CheckTimes Method for Calendar Class 54](#_Toc484414469)

[Test Calendar add meetings to short months 55](#_Toc484414470)

[Integration Test Executions 56](#_Toc484414471)

[Calendar Meeting class integration test 56](#_Toc484414472)

[Calendar Meeting Room class integration test 57](#_Toc484414473)

[Calendar Meeting Person class integration test 58](#_Toc484414474)

[Defect Report 59](#_Toc484414475)

[Discussions and conclusions 65](#_Toc484414476)

[Descriptions of errors 65](#_Toc484414477)

[StackOverFlowErrors 65](#_Toc484414478)

# Testing Strategy

## Approach

White-Box techniques used include unit testing (Static program analysis and Data-flow analysis on Methods and Classes), integration testing (Classes and Methods), inter-class and intra-class testing using control flow graphing (CFG). Black-Box techniques used include boundary value analysis (BVA), equivalence partitioning (EP), Category partitioning (CP) and Combinatorial testing (CT) where appropriate. Code and Path coverage metrics were used to ensure a complete test suite that covers all branches.

## Process

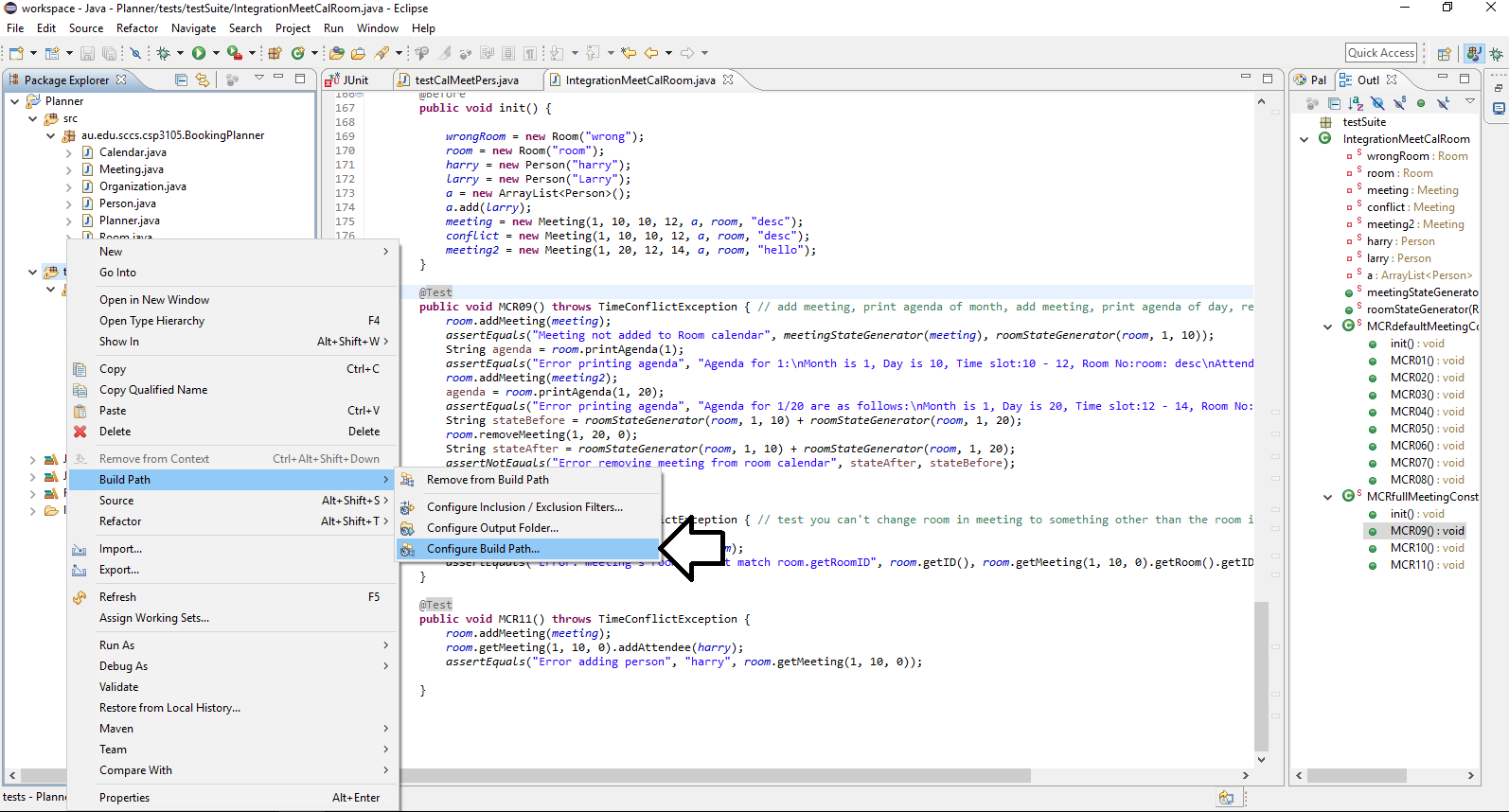
Develop class diagram as a stubbed hierarchy to determine the base class for object orientated testing. Develop CFGs for each method of each class and intra-class diagrams for each class for white-box analysis during the integration testing phase. Identify all definition-clear paths as DU pairs. Identify values for BVA, EP, CP, and CT on methods for black-box testing. Identify appropriate software packages for executing test cases. Develop test plan using analysis of the above data and generate a test suite using software packages that covers all branches and values. Implement the finished test suite in Junit and Eclipse, and finally capture screen shots of each test case proceeding execution.

# Stages of testing progress

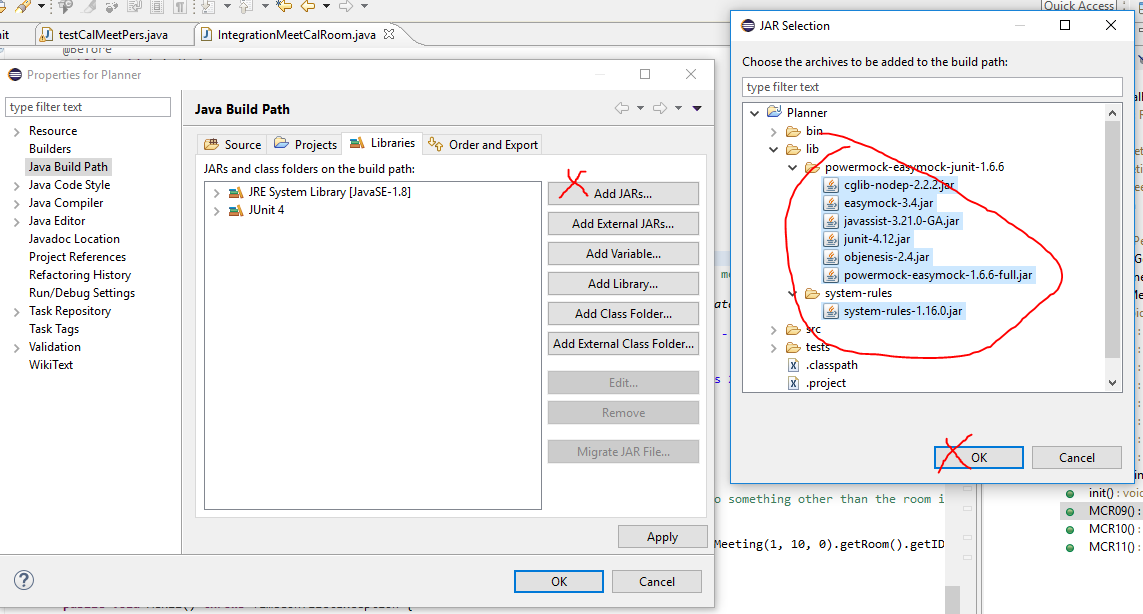
|  |  |
| --- | --- |
| Stage (Testing Artefacts) | Comments |
| Class diagram as a stubbed hierarchy. | Base class was determined as Meeting and tiers of integration testing were identified as aggregates of classes. Tier 1 {Meeting}, Tier 2 {Meeting, Calendar}, Tier 3.1 {Meeting, Calendar, Room}, Tier 3.2 {Meeting, Calendar, Person}, Tier 4 {Meeting, Calendar, Room, Person, Organization}, Tier 5 {Meeting, Calendar, Room, Person, Organization, Planner}. Note that tier 5 was identified for system testing while tier 1 was identified as the unit test for the base class. However, it was determined at a later stage that the base class could be Organization.class. The reasoning for this was the fact that Organization instantiates all the Room and Person objects that it needs in its constructor and only relies on the functionality of those objects that it instantiates. Prior to the instantiation of Organization, the Room and Person objects do not need to exist while the Meeting class requires an existing Room object and ArrayList of Person objects to be passed to its constructor. Further, there are methods in the Meeting class (e.g toString()) that rely on the functionality of Room and Person. The “TimeConflictException” class was disregarded in this diagram due to the class being an alias for built in exceptions. |
| Control flow graphs for each method of each class. | All methods of class were identified from the class diagram and subsequent control flow graphs of each method were developed to identify BVA, EP, CP and CT values to be used in developing black box test cases. Creating these diagrams ensured complete code coverage. |
| Intra-class diagrams for each class. | Intra-class diagrams allowed DU pairs for white-box integration testing to be identified. Creating these diagrams ensure complete path coverage. |
| Identify required software packages for developing a complete test plan. | Eclipse IDE with the Junit plugin were requirements of the case study. However further software packages to assist in generation of the test suite were identified as additional requirements. The “EasyMock” software package and its extension “PowerMock” to generate class stubs and reflection to access private methods (specifically the “Planner” class) whose method “inputOutput” is a private method. |
| Test plan | The test plan begins with a generally applicable test suite of BlackBox techniques Boundary Value Analysis (BVA) and Equivalence Partitioning (EP) that have been applied to method parameters when deemed necessary (e.g method testing the methods we determined required such attention to ensure correct validation or correct output.  Next, the unit tests of classes are presented beginning with the base class. The team determined that unit testing is required for all classes before the commencement of integration testing. Integration testing begins from the base class and then progresses up the hierarchy tier by tier focusing on the method call chains with control flow that moves between classes. We tried to implement test cases that covered all interactions possible between the classes in the sub-system under test. This included using the classes own methods to change its state before calling the method of another class to cause an interaction. In these test suites we tested with the assumption that the classes should be loosely coupled, and that any class that allows the changing of its attributes via methods, either intraclass or interclass, should either (1) not allow transitions to invalid states or (2) handle any invalid attribute within interaction methods in ALL instances using exception handling. For example, if the Meeting.toString() method is called on a meeting without an ArrayList of Person objects, the program should handle and react rather than cause an error. |
| Unit test suite | We started off by trying to method test using traditional procedural unit testing with BVA and EP. This approached changed to object orientated methods, unit testing classes, when we started using instance variables with inspection. A multitude of stack overflow errors in planner. They are present in every method of planner other than input output. The default meeting constructor does not initialize an ArrayList of type person variable name attendees. This error allows for multiple null pointer exceptions when attempted to access via getAttendees or toString methods. Additionally Calendar.checkTimes has a logic error that makes the month of December completely inaccessible. Further logic errors were found in addMeeting. The constructor of calendar blocks off the 30th of November making that day inaccessible. The same constructor also leaves several invalid dates unblocked. |
| Integration test suite | We began by testing the subsystem of Meeting and Calendar. To do this we identified the DU-pairs for instances of the objects and where they were used in other classes. We tracked these uses through a call chain in the interclass diagram. We built test cases based on the DU-pairs in terms of how we could call the methods of use on the instances in question and what effect this had on the state of the objects and the subsequent transitions. We then repeated this by integrating the next class to the subsystem, in this case, we integrated the Room and Person classes to the subsystem separately (i.e Meeting, Calendar, Room and then Meeting, Calendar and Person. Following this, we moved on to integrating Organization with all of the preceding classes. By doing this we were able to test how classes actually interact with each other in terms of methods that use and redefine objects or attributes from a different class. |
| System test suite | System testing was considered however due to time constraints, final system testing was infeasible. |

# How to execute test cases

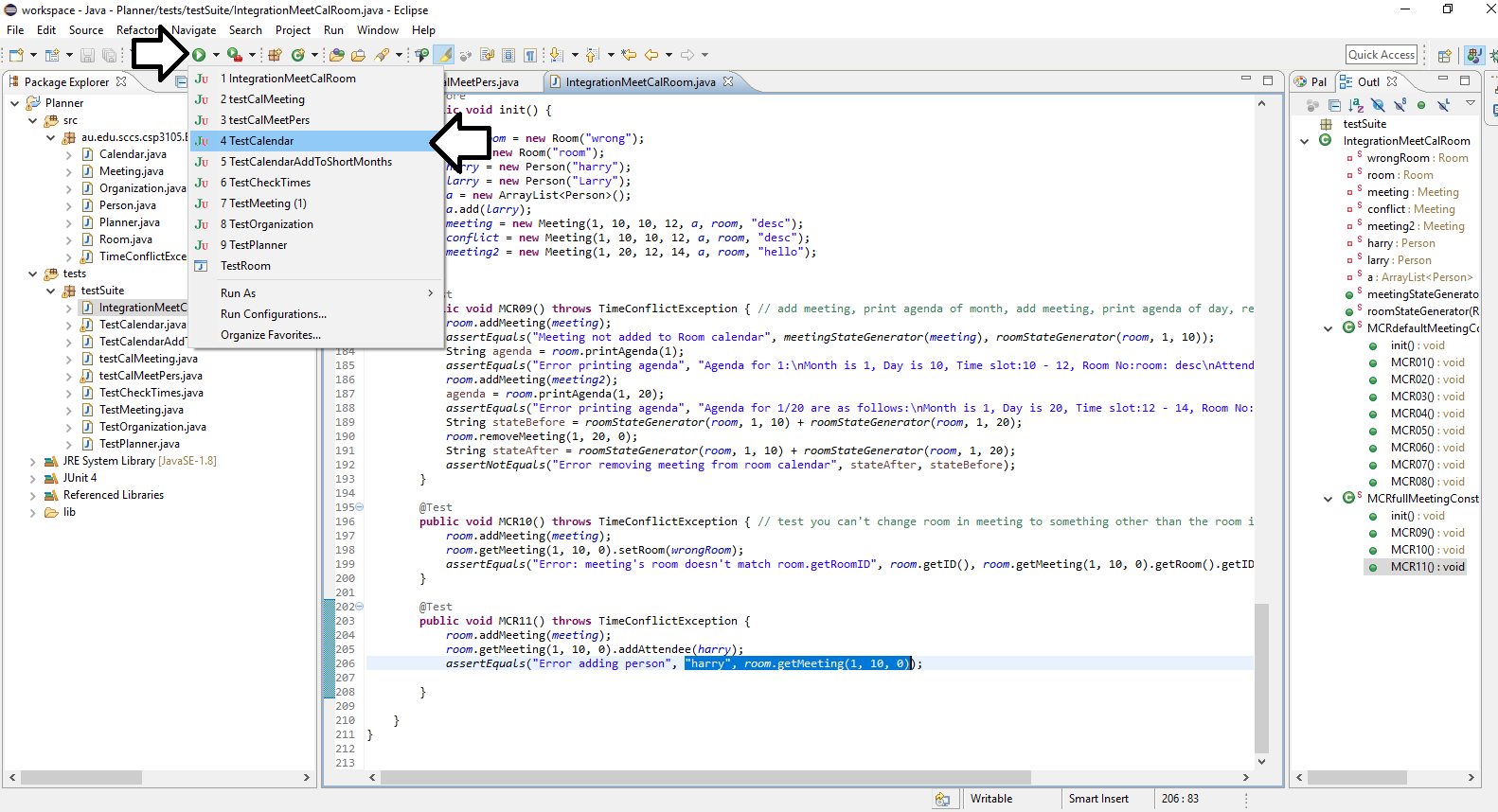
### 1. Go to configure build paths



### 2. Add archives



### 3. Select and Run test



# Test plan

## Unit test suite

### Meeting - Jarryd

Define the attributes of the class and establish the states that they can be in.

|  |  |
| --- | --- |
| Attribute | States |
| month : int | 1. Uninitialized 2. Initialized valid 3. Initialized invalid |
| day : int | 1. Uninitialized 2. Initialized valid 3. Initialized invalid |
| start : int | 1. Uninitialized 2. Initialized valid 3. Initialized invalid |
| end : int | 1. Uninitialized 2. Initialized valid 3. Initialized invalid |
| attendees : ArrayList<Person> | 1. Uninitialized 2. Initialized but empty 3. Initialized & >=1 Person object |
| room : Room | 1. Uninitialized 2. Initialized valid 3. Initialized invalid |
| description : String | 1. Uninitialized 2. Initialized valid 3. Initialized invalid |

#### Definitions and Uses (Intraclass)

Use the Intraclass Control Flow Graph for Meeting class to determine all definitions of instance variables and then where they are used throughout the class from method to method. These “Def-use” pairs will form the basis for derivation of test cases. The test cases goal will be to cover these paths by calling the appropriate methods in sequence.

##### Legend

|  |  |
| --- | --- |
| m | Month |
| d | Day |
| sh | Start Hour |
| eh | End Hour |
| p | Person |
| AL | ArrayList |
| r | Room |
| desc | Description |

Nodes in the following table refer to the nodes on the ***intraclass*** CFG for Meeting class in relevant attached diagram.

##### Month

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[2.2] Meeting(***m, d***) | Meeting[8.2] toString(),  Meeting[9.2] getMonth(),  Meeting[10.2] setMonth() | Inspector  Inspector  Modifier |
| Meeting[3.2] Meeting(***m, d, desc***) | Meeting[8.2] toString(),  Meeting[9.2] getMonth(),  Meeting[10.2] setMonth() | Inspector  Inspector  Modifier |
| Meeting[4.2] Meeting(***m, d, sh, eh***) | Meeting[8.2] toString(),  Meeting[9.2] getMonth(),  Meeting[10.2] setMonth() | Inspector  Inspector  Modifier |
| Meeting[5.2] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.2] toString(),  Meeting[9.2] getMonth(),  Meeting[10.2] setMonth() | Inspector  Inspector  Modifier |

##### Day

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[2.3] Meeting(***m, d***) | Meeting[8.2] toString(),  Meeting[11.2] getDay(),  Meeting[12.2] setDay() | Inspector  Inspector  Modifier |
| Meeting[3.3] Meeting(***m, d, desc***) | Meeting[8.2] toString(),  Meeting[11.2] getDay(),  Meeting[12.2] setDay() | Inspector  Inspector  Modifier |
| Meeting[4.3] Meeting(***m, d, sh, eh***) | Meeting[8.2] toString(),  Meeting[11.2] getDay(),  Meeting[12.2] setDay() | Inspector  Inspector  Modifier |
| Meeting[5.3] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.2] toString(),  Meeting[11.2] getDay(),  Meeting[12.2] setDay() | Inspector  Inspector  Modifier |

##### Start Hour

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[2.4] Meeting(***m, d***) (defaulted 0) | Meeting[8.2] toString(),  Meeting[13.2] getStartTime(),  Meeting[14.2] setStartTime() | Inspector  Inspector  Modifier |
| Meeting[3.4] Meeting(***m, d, desc***) (defaulted 0) | Meeting[8.2] toString(),  Meeting[13.2] getStartTime(),  Meeting[14.2] setStartTime() | Inspector  Inspector  Modifier |
| Meeting[4.4] Meeting(***m, d, sh, eh***) | Meeting[8.2] toString(),  Meeting[13.2] getStartTime(),  Meeting[14.2] setStartTime() | Inspector  Inspector  Modifier |
| Meeting[5.4] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.2] toString(),  Meeting[13.2] getStartTime(),  Meeting[14.2] setStartTime() | Inspector  Inspector  Modifier |

##### End Hour

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[2.5] Meeting(***m, d***) (defaulted 0) | Meeting[8.2] toString(),  Meeting[15.2] getEndTime(),  Meeting[16.2] setEndTime() | Inspector  Inspector  Modifier |
| Meeting[3.5] Meeting(***m, d, desc***) (defaulted 0) | Meeting[8.2] toString(),  Meeting[15.2] getEndTime(),  Meeting[16.2] setEndTime() | Inspector  Inspector  Modifier |
| Meeting[4.5] Meeting(***m, d, sh, eh***) | Meeting[8.2] toString(),  Meeting[15.2] getEndTime(),  Meeting[16.2] setEndTime() | Inspector  Inspector  Modifier |
| Meeting[5.5] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.2] toString(),  Meeting[15.2] getEndTime(),  Meeting[16.2] setEndTime() | Inspector  Inspector  Modifier |

##### AL<p>

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[5.6] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.4, 8.5] toString(),  Meeting[6.2] addAttendee(p),  Meeting[7.2] removeAttendee(p),  Meeting[17.2] getAttendees() | Inspector  Inspector  Modifier |

##### Room

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[5.7] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.2] toString(),  Meeting[18.2] getRoom(),  Meeting[19.2] setRoom() | Inspector  Inspector  Modifier |

##### Description

|  |  |  |
| --- | --- | --- |
| **Defined** [node] method() | **Use(s)** [node] method() | Type |
| Meeting[3.6] Meeting(***m, d, desc***) | Meeting[8.2] toString(),  Meeting[20.2] getDescription(),  Meeting[21.2] setDescription() | Inspector  Inspector  Modifier |
| Meeting[5.8] Meeting(m, d, sh, eh, AL<p>, r, desc) | Meeting[8.2] toString(),  Meeting[20.2] getDescription(),  Meeting[21.2] setDescription() | Inspector  Inspector  Modifier |

### Set 1

#### Mock Objects Required

|  |  |
| --- | --- |
| ID | constructor |
| p1 | Person(“Jeff”) |
| r1 | Room() |

#### Test Cases

|  |  |  |
| --- | --- | --- |
| ID | Call - method(x) | Expected outcome |
| M00 | addAttendee(p1) | getAttendees().get(0) == p1 |
| M01 | removeAttendee(p1) | getAttendees() == null |
| M02 | toString() | toString() != null |
| M03 | getMonth() | getMonth == 0 |
| M04 | setMonth(1) | getMonth() == 1 |
| M05 | getDay() | getDay() == 0 |
| M06 | setDay(2) | getDay() == 2 |
| M07 | getStartTime() | getStartTime() == 0 |
| M08 | setStartTime(3) | getStartTime() == 3 |
| M09 | getEndTime() | getEndTime() == 0 |
| M10 | setEndTime(4) | getEndTime() == 4 |
| M11 | getAttendees() | getAttendees().get(0) == null |
| M12 | getRoom() | getRoom() == null |
| M13 | setRoom(r1) | getRoom() == r1 |
| M14 | getDescription() | getDescription() == null |
| M15 | setDescription("Test description") | GetDescription() == "Test description" |

Notes:

M01, M02 & M03 are expected to fail in our runs.

### Set 2

#### Mock Objects Required

|  |  |
| --- | --- |
| ID | constructor |
| p2 | Person("Jeff") |
| r2 | Room("JO.123") |
| people | ArrayList<Person> [p2] |
| meeting | Meeting(1, 2, 3, 4, people, r2, "description"  ) |

|  |  |  |
| --- | --- | --- |
| ID | Call - method(x) | Expected outcome |
| M16 | addAttendee(p2) | getAttendees().size() == 2 |
| M17 | removeAttendee(p2) | getAttendees().size() == 0 |
| M18 | toString() | toString() == "Month is 1, Day is 2, Time slot:3 - 4, Room No:JO.123: description\nAttending: Jeff" |

\*\* Didn't retest setters and getters for this constructor as they were tested in set 1. We have validated that work.

### Calendar – Jarryd

|  |  |
| --- | --- |
| Attributes | States |
| ArrayList<ArrayList<ArrayList<Meeting>>> occupied | Initialized empty, initialized set |

|  |  |
| --- | --- |
| Methods | Type |
| bool isBusy(int month, int day, int start, int end) | inspector |
| void checkTimes(int month, int day, int start, int end) | inspector |
| void addMeeting(Meeting toAdd) | modifier |
| void clearSchedule(int month, int day) | modifier |
| String printAgenda(int month) | inspector |
| String printAgenda(int month, int day) | inspector |
| Meeting getMeeting(int month, int day, int index) | inspector |
| void removeMeeting(int month, int day, int index) | modifier |

#### checkTimes() - Jarryd

This method performs validation on the month, day, start hour and end hour inputs and is a vital component of the system. We have decided that it should be tested individually before any other method in the class for the following reasons;

* It is called by other methods in the class.
* It does not call any other methods of the class.

We will use our test cases designed to test Boundary Value and Equivalence Partitions with ***weak*** equivalence partitioning. This is due to the fact that they are individually checked inputs where the invalid input to just one parameter will raise an exception. As per our assumption that ***start hour*** must be ***greater*** than ***end hour*** (cannot be equal to) we will test the combinations of these inputs. Testing for a valid input of ***start hour*** = 23 should be infeasible, as it would mean there is no valid input for ***end hour***, the same would occur when testing the valid input of ***end hour*** = 0.

##### Legend

|  |  |
| --- | --- |
| Lb | Lower Boundary |
| Ub | Upper Boundary |

##### Equivalence Partitions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| month | day | sh | eh | Combinatorial Invalids |
| !(int) | !(int) | SH < 0 [22] | EH < 0 [24] | SH > EH [12] |
| M < 1 [17] | D < 1 [19] | SH > 23 [23] | EH > 23 [25] | SH == EH [15] |
| M > 12 [18] | D > 31 [20] | !(int) | !(int) |  |
| 1 <= M <= 12 [16] | 1 <= D <= 31 [16] | 0 <= SH <= 23 [16] | 1 <= EH <= 23 [16] |  |

##### Boundaries

Example Format:

LbM(1) 0 [02]

Lb = Lower Boundary

M = Month

(1) = Part of boundary (i.e the lower part, the actual boundary value, or upper part of the boundary)

0 = The actual value of boundary part

[02] = The corresponding test case which uses it

|  |  |  |  |
| --- | --- | --- | --- |
| Month | Day | Start hour | End Hour |
| LbM(1) 0  [02] | LbD(1) 0 [07] | LbSH(1) -1 [10] | LbEH(1) -1 [14] |
| LbM(2) 1 [01] | LbD(2) 1 [01] | LbSH(2) 0 [01] | LbEH(2) 0 [15] |
| LbM(3) 2 [03] | LbD(3) 2 [08] | LbSH(3) 1 [04] | LbEH(3) 1 [01] |
| UbM(1) 11 [04] | Ub31D(1) 30 [04] | UbSH(1) 22 [03] | UbEH(1) 22 [04] |
| UbM(2) 12 [05] | Ub31D(2) 31 [05] | UbSH(2) 23 [12] | UbEH(2) 23 [03] |
| UbM(3) 13 [06] | Ub31D(3) 32 [09] | UbSH(3) 24 [11] | UbEH(3) 24 [13] |

BVA & EP of checkTimes()

 The checkTimes method is void and so there is no expected output. The following legend details the criteria for test cases passing/failing.

##### Legend:

|  |
| --- |
| Passes without Exception |
| TCE Thrown |
| Infeasible due to combination of values |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Tech | Tested | Month | Day | Start hour | End hour |
| CL01 | bva | LbM(2), LbD(2),  LbSH(2), LbEH(3) | 1 | 1 | 0 | 1 |
| CL02 | bva | LbM(1) | 0 | 2 | 2 | 3 |
| CL03 | bva | LbM(3),  UbSH(1), UbEH(2) | 2 | 27 | 22 | 23 |
| CL04 | bva | UbM(1), Ub31D(1), UbEH(1),  LbSH(3) | 11 | 30 | 1 | 22 |
| CL05 | bva | UbM(2), Ub31D(2), | 12 | 31 | 1 | 2 |
| CL06 | bva | UbM(3) | 13 | 5 | 3 | 4 |
| CL07 | bva | LbD(1) | 3 | 0 | 7 | 8 |
| CL08 | bva | LbD(3) | 8 | 2 | 3 | 4 |
| CL09 | bva | Ub31D(3) | 5 | 32 | 10 | 11 |
| CL10 | bva | LbSH(1) | 8 | 13 | -1 | 4 |
| CL11 | bva | UbSH(3) | 9 | 8 | 24 | 20 |
| CL12 | bva, combo | UbSH(2),  com[SH > EH] | 6 | 23 | 23 | 18 |
| CL13 | bva | UbEH(3) | 4 | 25 | 9 | 24 |
| CL14 | bva | LbEH(1) | 2 | 5 | 2 | -1 |
| CL15 | Bva, combo | LbEH(2),  Com[SH==EH] | 9 | 21 | 0 | 0 |
| CL16 | ep | Ev(M), Ev(D), Ev(SH), Ev(EH) | 4 | 20 | 5 | 7 |
| CL17 | ep | Ei(M)[M<1] | -5 | 14 | 5 | 12 |
| CL18 | ep | Ei(M)[M>12] | 15 | 17 | 10 | 13 |
| CL19 | ep | Ei(D)[D<1] | 8 | -7 | 15 | 16 |
| CL20 | ep | Ei(31D)[D>31] | 5 | 40 | 11 | 14 |
| CL21 | ep | Ev(31D)[D==31 | 7 | 31 | 13 | 15 |
| CL22 | ep | Ei(SH)[SH<0] | 3 | 22 | -6 | 3 |
| CL23 | ep | Ei(SH)[SH>23] | 4 | 6 | 29 | 21 |
| CL24 | ep | Ei(EH)[EH<1] | 5 | 14 | 3 | -4 |
| CL25 | ep | Ei(EH)[EH>24] | 12 | 9 | 10 | 31 |

#### Testing the validity of ArrayList<ArrayList<ArrayList<Meeting>>> occupied - Jarryd

This set of test cases will check that the calendar successfully fills the unrequired days in months with less than 31 days with an all day meeting described as "Day does not exist". It was determined that strong EP testing was required for testing these days as it was identified as an area of code highly likely to contain errors.

|  |  |  |
| --- | --- | --- |
| Test-ID | Called method(s) | Expected result |
| CL26 | Calendar.isBusy(2, 28, 10, 15) | IsBusy() returns false |
| CL27 | Calendar.isBusy(2, 29, 10, 12) | IsBusy() returns true |
| CL28 | Calendar.isBusy(2, 30, 0, 1) | IsBusy() returns true |
| CL29 | Calendar.isBusy(2, 31, 22, 23) | IsBusy() returns true |
| CL30 | Calendar.isBusy(4, 30, 12, 13) | IsBusy() returns false |
| CL31 | Calendar.isBusy(4, 31, 0, 23) | IsBusy() returns true |
| CL32 | Calendar.isBusy(6, 30, 10, 12) | IsBusy() returns false |
| CL33 | Calendar.isBusy(6, 31, 13, 16) | IsBusy() returns true |
| CL34 | Calendar.isBusy(9, 30, 10, 12) | IsBusy() returns false |
| CL35 | Calendar.isBusy(9, 31, 10, 22) | IsBusy() returns true |
| CL36 | Calendar.isBusy(11, 30, 13, 14) | IsBusy() returns false |
| CL37 | Calendar.isBusy(11, 31, 11, 12) | IsBusy() returns true |

#### Modifying the Calendar - Jarryd

These test cases test the scenarios in which the methods of *Calendar* can modify its attribute, *occupied. Assumption that should be able to add adjacent meetings*

All mock meetings set to return testID as description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test-ID | Testing | Mock(s) | Method(s) called | Expected outcome |
| CL38 | Adding meeting to blocked out non-day | M Meeting  (2, 30, 10, 12) | addMeeting(meeting), | Throws ***TCE -***  Meeting not added |
| CL39 | Adding meeting to free slot in valid day | M Meeting  (5, 10, 13, 14) | addMeeting(meeting),  getMeeting(10, 5, 0).getDescription() | Description at index 0 == CL39  Meeting added |
| CL40 | Add to free slot  immediately preceding booked slot | M Meeting1  (2, 2, 14, 15)  M Meeting2  (2, 2, 13, 14) | addMeeting(meeting1),  addMeeting(meeting2),  getMeeting(2, 2, 1).getDescription() | Two adjacent meetings added. Description of meeting in index 1 == CL40b |
| CL41 | Add to free slot  immediately following booked slot | M Meeting1  (3, 3, 14, 15)  M Meeting2  (3, 3, 15, 16) | addMeeting(meeting1),  addMeeting(meeting2),  getMeeting(3, 3, 1).getDescription() | Two adjacent meetings added. Agenda for day shows both |
| CL42 | Add to conflicted slot  (new overlaps start time of existing) | M Meeting1  (1, 1, 12, 15)  M Meeting2  (1, 1, 11, 13) | addMeeting(meeting1),  addMeeting(meeting2), | Throws ***TCE*** -  Meeting2 not added |
| CL43 | Add to conflicted slot  (new is subsumed by existing) | M Meeting1  (4, 4, 12, 17)  M Meeting2  (4, 4, 13, 16) | addMeeting(meeting1),  addMeeting(meeting2), | Throws ***TCE*** -  Meeting2 not added |
| CL44 | Add to conflicted slot (new overlaps end of existing) | M Meeting1  (5, 5, 12, 17)  M Meeting2  (5, 5, 15, 20) | addMeeting(meeting1),  addMeeting(meeting2), | Throws ***TCE*** -  Meeting2 not added |
| CL45 | Add to conflicted slot (new subsumes existing) | M Meeting1  (6, 6, 12, 17)  M Meeting2  (6, 6, 10, 20) | addMeeting(meeting1),  addMeeting(meeting2), | Throws ***TCE*** -  Meeting2 not added |
| CL46 | Add to conflicted slot (new matches existing) | M Meeting1  (7, 7, 12, 17)  M Meeting2  (7, 7, 12, 17) | addMeeting(meeting1),  addMeeting(meeting2), | Throws ***TCE*** -  Meeting2 not added |
| CL47 | Add to invalid date (out of bounds) | M meeting  (-3, 4, 0, 5) | addMeeting(meeting), | Throws TCE -  Meeting not added |
| CL48 | Remove a meeting from free slot | - | RemoveMeeting(1, 1, 0) | Throw TCE |
| CL49 | Remove meeting from invalid date | - | RemoveMeeting(-3, 1, 0) | Throw TCE |
| CL50 | Remove meeting from blocked off day in shorter month - (Should not be possible) | - | RemoveMeeting(4, 31, 0)  GetMeeting(4, 31, 0).getDescription() | GetMeeting() != null |
| CL51 | ClearSchedule of day with meeting(s) | M Meeting  (1, 1, 1, 2) | AddMeeting(meeting),  IsBusy(1, 1, 1, 2),  ClearSchedule(1, 1),  getMeeting(1, 1, 0) | GetMeeting() == null |
| CL52 | ClearSchdedule of day outside bounds | - | ClearSchedule(1, 45) | Throws TCE |
| CL53 | Call isBusy() on times overlapping start of existing meeting | M Meeting  (1, 1, 10, 14) | AddMeeting(meeting),  isBusy(1, 1, 9, 11) | isBusy == true |
| CL54 | Call isBusy() on times subsumed by existing meeting | M Meeting  (1, 1, 10, 14) | AddMeeting(meeting),  isBusy(1, 1, 11, 13) | isBusy == true |
| CL55 | Call isBusy() on times overlapping end of existing meeting | M Meeting  (1, 1, 10, 14) | AddMeeting(meeting),  isBusy(1, 1, 13, 15) | isBusy == true |
| CL56 | Call isBusy() on times that subsume existing meeting | M Meeting  (1, 1, 10, 14) | AddMeeting(meeting),  isBusy(1, 1, 9, 15) | isBusy == true |
| CL57 | Call isBusy() on free slot preceding existing meeting | M Meeting  (1, 1, 10, 14) | AddMeeting(meeting),  isBusy(1, 1, 9, 10) | isBusy == false |
| CL58 | Call isBusy() on free slot following existing meeting | M Meeting  (1, 1, 10, 14) | AddMeeting(meeting),  isBusy(1, 1, 14, 15) | isBusy == false |

Test case CL38 failed, this suggests addMeeting can be performed on days intended to be blocked off as “Day does not exist”. This has led to the need for further testing on the rest of the days intended to be blocked off.

|  |  |  |  |
| --- | --- | --- | --- |
| Test-ID | Mock(s) | Method(s) called | Expected Outcome |
| CL59 | Meeting (2, 28, 10, 15, "CL59") | AddMeeting(Meeting),  GetMeeting(2, 28, 0) | GetDescription() == "CL59" |
| CL60 | Meeting (2, 29, 10, 12, "CL60") | AddMeeting(Meeting),  GetMeeting(2, 29, 1) | Throw IndexOutOfBoundsException |
| CL61 | Meeting (2, 30, 0, 1, "CL61") | AddMeeting(Meeting),  GetMeeting(2, 30, 1) | Throw IndexOutOfBoundsException |
| CL62 | Meeting (2, 31, 22, 23, "CL62") | AddMeeting(Meeting),  GetMeeting(2, 31, 1) | Throw IndexOutOfBoundsException |
| CL63 | Meeting (4, 30, 12, 13, "CL63") | AddMeeting(Meeting),  GetMeeting(4, 30, 0) | GetDescription() == "CL63" |
| CL64 | Meeting (4, 31, 0, 23, "CL64") | AddMeeting(Meeting),  GetMeeting(4, 31, 1) | Throw IndexOutOfBoundsException |
| CL65 | Meeting (6, 30, 10, 12, "CL65") | AddMeeting(Meeting),  GetMeeting(6, 30, 0) | GetDescription() == "CL65" |
| CL66 | Meeting (6, 31, 13, 16, "CL66") | AddMeeting(Meeting),  GetMeeting(6, 31, 1) | Throw IndexOutOfBoundsException |
| CL67 | Meeting (9, 30, 10, 12, "CL67") | AddMeeting(Meeting),  GetMeeting(9, 30, 0) | GetDescription() == "CL67" |
| CL68 | Meeting (9, 31, 10, 22, "CL68") | AddMeeting(Meeting),  GetMeeting(9, 31, 1) | Throw IndexOutOfBoundsException |
| CL69 | Meeting (11, 30, 13, 14, "CL69") | AddMeeting(Meeting),  GetMeeting(11, 30, 0) | GetDescription() == "CL69" |
| CL70 | Meeting (11, 31, 11, 12, "CL70") | AddMeeting(Meeting),  GetMeeting(11, 31, 1) | Throw IndexOutOfBoundsException |

### Room - QJ

Room is not evaluated in the unit testing phase as it is too tightly coupled with the Calendar class for any value to be gained from testing it individually.

#### Rationale:

* Room() & Room(String) are not tested as they are constructors with no control flow.
* getID() is not tested as it is a simple getter with no control flow.
* The following Room methods are not tested here as they are too heavily tied to the Calendar class for their coverage, in unit testing, to have any value. They are evaluated in integration testing.
* addMeeting(Meeting)
* printAgenda(int)
* printAgenda(int, int)
* isBusy(int, int, int, int)
* getMeeting(int, int, int)
* removeMeeting(int, int, int)

#### Problems:

* It is possible for getID() to return null, if null is passed to the Room(String) constructor. While not a problem currently, the regression bug potential should be noted.

### Person - QJ

Person is not evaluated in the unit testing phase as it is too tightly coupled with the Calendar class for any value to be gained from testing it in isolation.

#### Rationale:

* Both person() and person(String) are not unit tested as they are constructors with no control flow.
* getName() is not unit tested as it is a simple getter with no control flow.
* The following person methods are not tested here as they are too heavily tied to the Calendar class for their coverage, in unit testing, to have any value. They are evaluated in integration testing.
* addMeeting(Meeting)
* printAgenda(int)
* printAgenda(int, int)
* isBusy(int, int, int, int)
* getMeeting(int, int, int)
* removeMeeting(int, int, int)

### Organization - QJ

Organization is highly coupled with the Person and Room objects, however there is some test worthy control flow in the getRoom(String) & getEmployee(String) methods.

##### Test cases:

|  |  |  |
| --- | --- | --- |
| ID | Call - method(x) | Expected outcome |
| O01 | getRoom("A real room") | getRoom("A real room").getID() == "ML13.213" |
| O02 | getRoom("Not a real room") | Throws Exception |
| O03 | getEmployee("A real employee") | getEmployee("A real employee").getName() == "Justin Gardener" |
| O04 | getEmployee("Not a real employee") | Throws Exception |

##### Object instances:

|  |  |
| --- | --- |
| ID | Overridden/new methods |
| testOrganization | setEmployees(ArrayList<Person>)  setRooms(ArrayList<Room>) |

###### Rationale:

* The Organization() constructor is not unit tested as it contains no control flow and on inspection, no failure conditions were identified as occurring when isolated.
* getEmployees() and getRooms() are not unit tested as they are simple getters with no control flow.
* While getRoom(String) and getEmployee(String) are both tightly coupled with the Room and Person classes respectively, there is some test worthy control flow. Mock Room & Person classes will be used to assist in isolating Organization.

### Planner - QJ

#### Methods not under test:

* Planner()
* main(String[] args)
* inputOutput(String message)

#### Methods under test:

* mainMenu()
* scheduleMeeting()
* scheduleVacation()
* checkRoomAvailability()
* checkEmployeeAvailability()
* checkAgendaRoom()
* checkAgendaPerson()

#### Test cases:

Method under test – mainMenu()

|  |  |  |
| --- | --- | --- |
| ID | Call - method(x) | Expected outcome |
| P01 | inputOutput – return "1"; | Planner.scheduleMeeting();  PowerMock.expectLastCall();  verify(Planner); |
| P02 | inputOutput – return "2"; | Planner.scheduleVacation();  PowerMock.expectLastCall();  verify(Planner); |
| P03 | inputOutput – return "3"; | Planner.checkRoomAvailability();  PowerMock.expectLastCall();  verify(Planner); |
| P04 | inputOutput – return "4"; | Planner.checkEmployeeAvailability();  PowerMock.expectLastCall();  verify(Planner); |
| P05 | inputOutput – return "5"; | Planner.checkAgendaRoom();  PowerMock.expectLastCall();  verify(Planner); |
| P06 | inputOutput – return "6"; | Planner.checkAgendaPerson();  PowerMock.expectLastCall();  verify(Planner); |
| P07 | inputOutput – return "0"; | Planner.System.exit(0);  PowerMock.expectLastCall();  verify(Planner); |
| P08 | inputOutput – return "7" | Planner.mainMenu();  PowerMock.expectLastCall();  verify(Planner); |
| P09 | inputOutput – return "not numeric" | Planner.mainMenu();  PowerMock.expectLastCall();  verify(Planner); |
| P10 | While(true) {InputOutput – return "7"} | !StackOverFlowError |

###### Rationale:

* P10 – demonstrating StackOverFlowError. This error is present in many classes & methods

Method under test – scheduleMeeting()

|  |  |  |
| --- | --- | --- |
| ID | Inputs | Expected outcome |
| P11 | month = "1";  day = "1"  startTime "1"  endTime "Non-numeric" | Throws NumberFormatException |
| P12 | month = "1";  day = "1"  startTime "Non-numeric"  endTime "1" | Throws NumberFormatException |
| P13 | month = "1";  day = "Non-numeric"  startTime "1"  endTime "1" | Throws NumberFormatException |
| P14 | month = "Non-numeric";  day = "1"  startTime "1"  endTime "1" | Throws NumberFormatException |
| P15 | m = stub "1"  d = stub "1"  sT = stub "1"  eT = stub "1"    org.getRooms() return mockRooms  r.isBusy() - throws TimeConflictException    - mainMenu()-  inputOutput() - stub return "1" | !StackOverFlowError |
| P16 | m = stub "1"  d = stub "1"  sT = stub "1"  eT = stub "1"  org.getRooms() - stub return mockRooms  r.isBusy() - stub return false  r.getID() - stub return "ID: mockRoom"    id = inputOutput() - stub return "cancel"    - mainMenu() -  inputOutput() - stub return "1" | !StackOverFlowError |
| P17 | m = stub "1"  d = stub "1"  sT = stub "1"  eT = stub "1"  org.getRooms() - stub return mockRooms  r.isBusy() - stub return false  r.getID() - stub return "ID: mockRoom"    id = inputOutput() - stub return "ID: mockRoom"  org.getRoom("ID: mockRoom") - stub return mockRoom    org.getEmployees() - return mockEmployees  p.isBusy() - stub throw TimeConflictException    - mainMenu() -  inputOutput() - stub return "1" | !StackOverFlowError |
| P18 | m = stub "1"  d = stub "1"  sT = stub "1"  eT = stub "1"  org.getRooms() - stub return mockRooms  r.isBusy() - stub return false  r.getID() - stub return "ID: mockRoom"    id = inputOutput() - stub return "ID: mockRoom"  org.getRoom("ID: mockRoom") - stub return mockRoom    org.getEmployees() - return mockEmployees  p.isBusy() - false  p.getName - stub return "Name: mockEmployee"    inputOutput() - return "Name: mockEmployee"  org.getEmployee("Name: mockEmployee")                                       - stub return mockEmployee  inputOutput() - stub return "done"    inputOutput() - stub return "Description: mockMeeting"    r.addMeeting() - stub expectPreviousCall()  p.addMeeting() - stub expectPreviousCall()    - mainMenu() -  inputOutput() - stub return"1" | !StackOverFlowError |

###### Object instances:

|  |  |
| --- | --- |
| ID | constructor |
| mockRoom | Room() |
| mockRooms | ArrayList<Room>[mockRoom] |
| mockEmployee | Person() |
| mockEmployees | ArrayList<Person>[mockEmployee] |
| mockOrg | MockOrg(mockRooms, mockEmployees) \*representation, not literal |

###### Rationale:

* No BVA on m/d/sT/eT as validation is handled in Calendar.checkTimes
* Attempted to do all paths testing
* Using stub returns as they can be trigger repeatedly in EasyMock. Did this for automating the StackOverFlowError cases.

###### Problems:

* P01 – Something wrong with testcase implementation. Why does it take >200ms to execute?

Note:

The rest of the Planner class test plan was not completed.

## Integration test suite

Please note: all nodes specified in [node] in tables below refer to nodes on the intraclass CFG for the respective class. CFG’s have been included in the zip file.

### Subsystem: Calendar & Meeting - Carl

Required mock object(s): Person ArrayList, Room

#### Class Meeting:

Valid constructors.

|  |  |
| --- | --- |
| Valid | Invalid |
| Meeting(***m, d***) | Meeting() |
| Meeting(*m, d, desc*) | Any instance with ***m*** or ***d*** or ***sh*** or ***eh*** out of bounds |
| Meeting(m, d, sh, eh) |  |
| Meeting(***m***,***d****,****sh****,****eh***, AL<***p***>, ***r***, ***desc***) |  |

#### Class Calendar:

Has only one valid constructor, the default constructor.

Format: Class[node]: Method

M = month, d = day, sh = start, eh = end, p = person, a = attendees, r = room, desc = description

AL<> = ArrayList

|  |  |
| --- | --- |
| Defined | Use |
| Meeting(***m, d***) | Calendar[4]: addMeeting(**meeting**) |
|  | Calendar[7]: getMeeting(***m, d, i***) |
|  | Calendar[8]: removeMeeting(***m, d, i***) |
|  | Calendar[6]: printAgenda(***m***) |
|  | Calendar[2]: isBusy(***m, d, sh, eh***) |
|  | Calendar[5]: clearSchedule(***m, d***) |
|  |  |
| Meeting(*m, d, desc*) | Calendar[4]: addMeeting(**meeting**) |
|  | Calendar[7]: getMeeting(***m, d, i***) |
|  | Calendar[8]: removeMeeting(***m, d, i***) |
|  | Calendar[6]: printAgenda(***m***) |
|  | Calendar[2]: isBusy(***m, d, sh, eh***) |
|  | Calendar[5]: clearSchedule(***m, d***) |
|  |  |
| Meeting(m, d, sh, eh) | Calendar[4]: addMeeting(**meeting**) |
|  | Calendar[7]: getMeeting(***m, d, i***) |
|  | Calendar[8]: removeMeeting(***m, d, i***) |
|  | Calendar[6]: printAgenda(***m***) |
|  | Calendar[2]: isBusy(***m, d, sh, eh***) |
|  | Calendar[5]: clearSchedule(***m, d***) |
|  |  |
| Meeting(***m***,***d****,****sh****,****eh***, AL<***p***>, ***r***, ***desc***) | Calendar[4]: addMeeting(**meeting**) |
|  | Calendar[7]: getMeeting(***m, d, i***) |
|  | Calendar[8]: removeMeeting(***m, d, i***) |
|  | Calendar[6]: printAgenda(***m***) |
|  | Calendar[2]: isBusy(***m, d, sh, eh***) |
|  | Calendar[5]: clearSchedule(***m, d***) |

|  |  |  |
| --- | --- | --- |
| Test Case ID | Procedure Input | Expected Output |
| CM01 | * Create new meeting (1,1) * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert printAgenda | "No Meetings booked on this date.\n\n" |
| CM02 | * Create new meeting (1,1,0,1) * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert printAgenda | "No Meetings booked on this date.\n\n" |
| CM03 | * Create new meeting (1,1,"Testdesc") * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert printAgenda | "No Meetings booked on this date.\n\n" |
| CM04 | * Create new meeting (1,1,0,1,ArrayList<Person>,Room("testID","Testdesc") * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert printAgenda | "No Meetings booked on this date.\n\n" |
| CM05 | * Create new meeting (1,1)Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert isBusy | false |
| CM06 | * Create new meeting (1,1,0,1) * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert isBusy | false |
| CM07 | * Create new meeting (1,1,"Testdesc") * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert isBusy | false |
| CM08 | * Create new meeting (1,1,0,1,ArrayList<Person>,Room("testID","Testdesc") * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Remove meeting from calendar(1,1,0) * Assert isBusy | false |
| CM09 | * Create new meeting (1,1) * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Clear schedule * Assert isBusy | false |
| CM10 | * Create new meeting (1,1, "Testdesc") * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Clear schedule * Assert isBusy | false |
| CM11 | * Create new meeting (1,1,0,1) * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Clear schedule * Assert isBusy | false |
| CM12 | * Create new meeting (1,1,0,1,ArrayList<Person>,Room("testID","Testdesc") * Create new calendar * Add meeting to calendar * Get meeting from calendar(1,1,0) * Clear schedule * Assert isBusy | false |

### Subsystem: Calendar & Meeting & Person - Carl

|  |  |
| --- | --- |
| Defined | Use |
| Person() | Calendar[4]: addMeeting(**meeting**) |
|  | Calendar[7]: getMeeting(***m, d, i***) |
|  | Calendar[8]: removeMeeting(***m, d, i***) |
|  | Calendar[6]: printAgenda(***m***) |
|  | Calendar[2]: isBusy(***m, d, sh, eh***) |
|  | Calendar[5]: clearSchedule(***m, d***) |
|  |  |
| Person(***name***) | Calendar[4]: addMeeting(**meeting**) |
|  | Calendar[7]: getMeeting(***m, d, i***) |
|  | Calendar[8]: removeMeeting(***m, d, i***) |
|  | Calendar[6]: printAgenda(***m***) |
|  | Calendar[2]: isBusy(***m, d, sh, eh***) |
|  | Calendar[5]: clearSchedule(***m, d***) |
|  |  |
| Person() | Meeting[7]: removeAttendee(***attendee***) |
|  | Meeting[8]: addAttendee(***attendee***) |
|  | Meeting[5]: Meeting(***m***,***d****,****sh****,****eh***, AL<***p***>, ***r***, ***desc***) |
|  |  |
| Person(***name***) | Meeting[7]: removeAttendee(***attendee***) |
|  | Meeting[8]: addAttendee(***attendee***) |
|  | Meeting[5]: Meeting(***m***,***d****,****sh****,****eh***, AL<***p***>, ***r***, ***desc***) |

|  |  |  |
| --- | --- | --- |
| Test Case ID | Procedure Input | Assert |
| CM01 | * Create new room * Create new person (default constructor) * Create new meeting * Add person to meeting * Create new calendar * Add meeting to calendar * Get meeting from calendar * Remove meeting from calendar * Assert printAgenda for calendar | "No Meetings booked on this date.\n\n" |
| CM02 | * Create new room * Create new person (default constructor) * Create new meeting * Add person to meeting * Create new calendar * Add meeting to calendar * Get meeting from calendar * Remove meeting from calendar * Assert isBusy calendar | false |
| CM03 | * Create new room * Create new person (default constructor) * Create new meeting * Add person to meeting * Create new calendar * Add meeting to calendar * Get meeting from calendar * Clear schedule for calendar * Assert isBusy calendar | false |
| CM04 | * Create new room * Create new person (String name) * Create new meeting * Add person to meeting * Create new calendar * Add meeting to calendar * Get meeting from calendar * Remove meeting from calendar * Assert printAgenda for calendar | "No Meetings booked on this date.\n\n" |
| CM05 | * Create new room * Create new person (String name) * Create new meeting * Add person to meeting * Create new calendar * Add meeting to calendar * Get meeting from calendar * Remove meeting from calendar * Assert isBusy calendar | false |
| CM06 | * Create new room * Create new person (String name) * Create new meeting * Add person to meeting * Create new calendar * Add meeting to calendar * Get meeting from calendar * Clear schedule for calendar * Assert isBusy calendar | false |
| CM07 | * Create new room * Create new person (String name) * Create new meeting * Add attendee to meeting * Remove attendee from meeting * Create new calendar * Add meeting to calendar * Assert Calendar->Meeting->getAttendees | Empty ArrayList |
| CM08 | * Create new room * Create new person (String name) * Create new meeting * Add attendee to meeting * Create new calendar * Add meeting to calendar * Remove attendee from meeting * Assert Calendar->Meeting->getAttendees | Empty ArrayList |
| CM09 | * Create new room * Create new person (String name) * Create new meeting * Add attendee to meeting * Add same attendee to meeting * Create new calendar * Add meeting to calendar * Assert Calendar->Meeting->getAttendees->Get(**0**)->getName NOT equals Calendar->Meeting->getAttendees->Get(**1**)->getName | True |

### Subsystem: Meeting, Calendar & Room - Jarryd

Required mock object(s): Person x2 into ArrayList

#### Class Meeting:

From the perspective of What is a valid state to add to a **Calendar**? What is an invalid state to add to a C**alendar**?

|  |  |
| --- | --- |
| Valid | Invalid |
| Meeting(***m, d***) | Meeting() |
| Meeting(*m, d, desc*) | Any instance with ***m*** or ***d*** or ***sh*** or ***eh*** out of bounds |
| Meeting(m, d, sh, eh) |  |
| Meeting(***m***,***d****,****sh****,****eh***, AL<***p***>, ***r***, ***desc***) |  |

We are only concerned with instantiating valid meeting objects for adding to the calendar(s) as unit testing has covered the attempted addition of invalid meetings to calendar objects extensively. We will include one test case that tests the setters and getters of the meeting class in order to change its state after the interaction with the rest of the subsystem. The test suite below was derived using the following process;

* + Locate the instantiations of the objects of subsystem.
  + Locate the uses of the object in question in other class methods.
  + Follow the call chain using the CFG’s for the classes.
  + Construct test cases that test the transitions between state of the objects and the interactions between objects.

#### Legend:

|  |  |
| --- | --- |
| m | Month |
| d | Day |
| sh | Start Hour |
| eh | End Hour |
| p | Person |
| AL | ArrayList |
| r | Room |
| desc | Description |
|  | Denotes method of same class as object |

Meeting

Format: Class[node]: Method

|  |  |  |
| --- | --- | --- |
| Defined | Use | Call chain |
| Meeting[1]: Meeting() | Room[4]: addMeeting(***meeting***) | Calendar.addMeeting(***meeting***), |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***),  Meeting.toString() |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, I***) | Calendar.removeMeeting***(m, d, i***) |
|  | Meeting[10]: setMonth(***m***), [9]: getMonth() |  |
|  | Meeting[12]: setDay(***d***), [11]: getDay() |  |
|  | Meeting[14]: setStartTime(***sh***) [13]: getStartTime() |  |
|  | Meeting[16]: setEndTime(***eh***) [15]: getEndTime() |  |
|  | Meeting[19]: setRoom(***r***), [18]: getRoom() |  |
| Meeting[2]: Meeting(***m, d***) | Room[4]: addMeeting(***meeting***) | Calendar.addMeeting(***meeting***), |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***),  Meeting.toString() |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, I***) | Calendar.removeMeeting***(m, d, i***) |
|  | Meeting[10]: setMonth(***m***), [9]: getMonth() |  |
|  | Meeting[12]: setDay(***d***), [11]: getDay() |  |
|  | Meeting[14]: setStartTime(***sh***) [13]: getStartTime() |  |
|  | Meeting[16]: setEndTime(***eh***) [15]: getEndTime() |  |
|  | Meeting[19]: setRoom(***r***), [18]: getRoom() |  |
| Meeting[3]: Meeting(***m, d, desc***) | Room[4]: addMeeting(meeting) | Calendar.addMeeting(***meeting***), |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***),  Meeting.toString() |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, I***) | Calendar.removeMeeting***(m, d, i***) |
|  | Meeting[10]: setMonth(***m***), [9]: getMonth() |  |
|  | Meeting[12]: setDay(***d***), [11]: getDay() |  |
|  | Meeting[14]: setStartTime(***sh***) [13]: getStartTime() |  |
|  | Meeting[16]: setEndTime(***eh***) [15]: getEndTime() |  |
|  | Meeting[19]: setRoom(***r***), [18]: getRoom() |  |
| Meeting[4]: Meeting(***m, d, sh, eh***) | Room[4]: addMeeting(meeting) | Calendar.addMeeting(***meeting***), |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***),  Meeting.toString() |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, I***) | Calendar.removeMeeting***(m, d, i***) |
|  | Meeting[10]: setMonth(***m***), [9]: getMonth() |  |
|  | Meeting[12]: setDay(***d***), [11]: getDay() |  |
|  | Meeting[14]: setStartTime(***sh***) [13]: getStartTime() |  |
|  | Meeting[16]: setEndTime(***eh***) [15]: getEndTime() |  |
|  | Meeting[19]: setRoom(***r***), [18]: getRoom() |  |
| Meeting[5]: Meeting(*m, d, sh, eh,*AL<*p*> *a, r, desc*) | Room[4]: addMeeting(meeting) | Calendar.addMeeting(***meeting***), |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***),  Meeting.toString() |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, I***) | Calendar.removeMeeting***(m, d, i***) |
|  | Meeting[10]: setMonth(***m***), [9]: getMonth() |  |
|  | Meeting[12]: setDay(***d***), [11]: getDay() |  |
|  | Meeting[14]: setStartTime(***sh***) [13]: getStartTime() |  |
|  | Meeting[16]: setEndTime(***eh***) [15]: getEndTime() |  |
|  | Meeting[19]: setRoom(***r***), [18]: getRoom() |  |

#### Class Calendar:

From the perspective of What state can the **Calendar** be in at the exact position (represented by parameters of meeting) that a meeting is trying to be added to?

##### Conflicted || Not Conflicted

First level of integration testing (Meeting & Calendar) covered the definition/instantiation of a Calendar object in its own constructor, at this level we are concerned with the instantiation/definition of Calendar in the next tier in the hierarchy (which is Room).

A ***Calendar*** object is instantiated in the ***Room*** class. For this reason during the integration testing of this subsystem there will be no method calls directly to a standalone instantiation of the ***Calendar*** class. The integration of the ***Calendar*** class will come from making method calls to the attribute ***Calendar*** in the instantiated ***Room***.

|  |  |  |
| --- | --- | --- |
| Defined | Use | Call chain |
| Room[1]: Room() | Room[3]: getID() |  |
|  | Room[4]: addMeeting(***meeting***) | Calendar.addMeeting(***meeting***) |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(*m*),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[7]: isBusy(***m, d, sh, eh***) | Calendar.isBusy(***m, d, sh, eh***) |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, i***) | Calendar.removeMeeting(***m, d, i***) |
| Room[2]: Room(id) | Room[3]: getID() |  |
|  | Room[4]: addMeeting(***meeting***) | Calendar.addMeeting(***meeting***) |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m***),  Meeting.toString() |
|  | Room[7]: isBusy(***m, d, sh, eh***) | Calendar.isBusy(***m, d, sh, eh***) |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
|  | Room[9]: removeMeeting(***m, d, i***) | Calendar.removeMeeting(***m, d, i***) |

\*\* while integration testing we found that calendar is never actually instantiated except for in room, this was not detectable in unit testing.

###### Class Room:

From the perspective of What state can the **Room**'s calendar be in at the exact position (represented by parameters of meeting) that a meeting is trying to be added to?

###### Conflicted || Not Conflicted

A ***Calendar*** object is instantiated in the ***Room*** class. For this reason, during the integration testing of this subsystem there will be no method calls directly to a standalone instantiation of the ***Calendar*** class. The integration of the ***Calendar*** class will come from making method calls to the attribute ***Calendar*** in the instantiated ***Room***.

###### Room

|  |  |  |
| --- | --- | --- |
| Defined | Use | Call chain |
| Room[1]: Room() | Meeting[5]: Meeting(***m, d, sh, eh,***AL<***p***> ***a, r, desc***) |  |
|  | Meeting[8]: toString() |  |
|  | Meeting[18]: getRoom() |  |
|  | Meeting[19]: setRoom(***r***) |  |
|  | Room[4]: addMeeting(***meeting***) | Calendar.addMeeting(***meeting***) |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***) |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***) |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |
| Room[2]: Room(id) | Meeting[5]: Meeting(***m, d, sh, eh,***AL<***p***> ***a, r, desc***) |  |
|  | Meeting[8]: toString() |  |
|  | Meeting[18]: getRoom() |  |
|  | Meeting[19]: setRoom(***r***) |  |
|  | Room[4]: addMeeting(***meeting***) | Calendar.addMeeting(***meeting***) |
|  | Room[5]: printAgenda(***m***) | Calendar.printAgenda(***m***) |
|  | Room[6]: printAgenda(***m, d***) | Calendar.printAgenda(***m, d***) |
|  | Room[8]: getMeeting(***m, d, i***) | Calendar.getMeeting(***m, d, i***) |

Instances created in init()

|  |  |
| --- | --- |
| room | Room(“room”) |
| wrongRoom | Room(“wrong”) |
| harry | Person(“harry”) |
| larry | Person(“larry”) |
| a | ArrayList<Person> (harry, larry) |
| meeting | Meeting(1, 10, 10, 12, a, room, "desc") |
| meeting2 | Meeting(1, 20, 12, 14, a, room, "hello") |
| conflict | Meeting(1, 10, 10, 12, a, room, "desc") |

### Test Cases - Jarryd

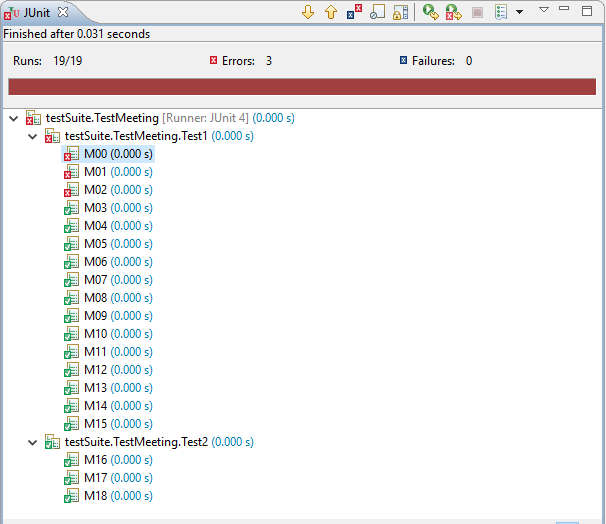
|  |  |  |  |
| --- | --- | --- | --- |
| Test-ID | Transition(s) | Method call(s) | Expected output |
| MCR01 | Default constructor into room calendar | Room.addMeeting(meeting) | State of meeting in the calendar == state of original meeting. |
| MCR02 | (m, d) constructor into room calendar and printAgena(m) | Room.addMeeting(meeting), Room.printAgenda(m) | Agenda != "No Meetings booked on this date.\n\n" |
| MCR03 | (m, d) constructor into room calendar and printAgena(m,d) | Room.addMeeting(meeting), Room.printAgenda(m, d) | Agenda != "No Meetings booked on this date.\n\n" |
| MCR04 | Add (m, d) constructor and then remove | Room.addMeeting(meeting), Room.removeMeeting(1, 10, 0) | State of room before remove != state of room after. |
| MCR06 | Add (m, d) meeting to room, then set room | Room.addMeeting(meeting), meeting.setRoom(room) | Room.getMeeting().getID() == “room” |
| MCR07 | Add (m, d) meeting and then set day to invalid int | Room.addMeeting(meeting), meeting.setDay(100) | TCE thrown. |
| MCR08 | Add meeting, add conflicted meeting | Room.addMeeting(meeting), room.addMeeting(conflict) | TCE thrown. |
| MCR09 | Complete constructor, printAgenda(m), add another, printAgenda(m, d), remove last | Room.addMeeting(meeting), room.printAgenda(m), room.addMeeting(meeting2), room.removeMeeting(1, 20, 0) | Assert correct agenda  Assert correct agenda  stateBefore == stateAfter |
| MCR10 | Add meeting then set room to wrong room | Room.addMeeting(meeting), room.getMeeting(1, 10, 0).setRoom(wrongRoom) | Room.getID() == room.getRoom().getID() |
| MCR11 | Add meeting then add an attendee afterward | Room.addMeeting(meeting), room.getMeeting(1, 10, 0).addAttendee(harry) | Room.getMeeting(1, 10, 0).getAttendees().get(1).getName() == “harry” |

 \*\* This test suite could not be finished due to time constraints. The intention was to traverse all of the DU pairs in one way or another in different sequences of method calls.

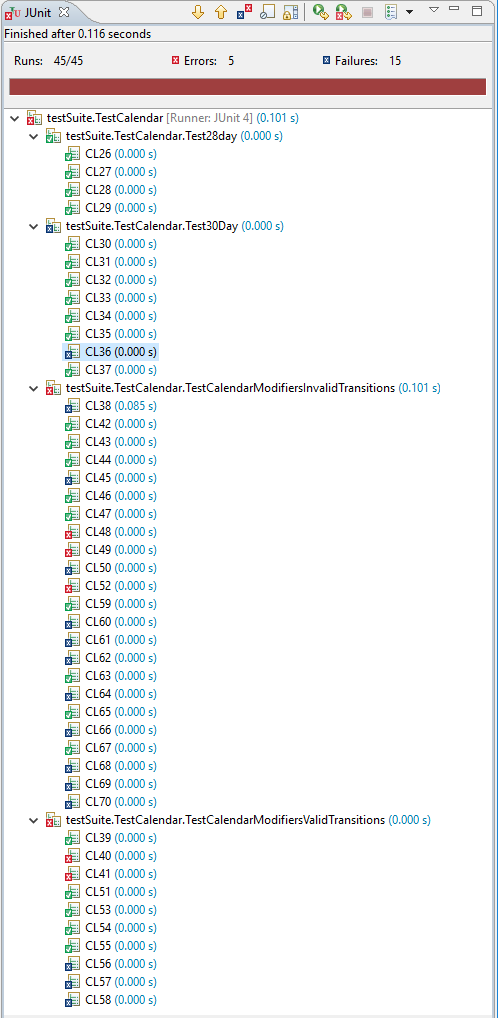
# Results

## Unit Test Executions

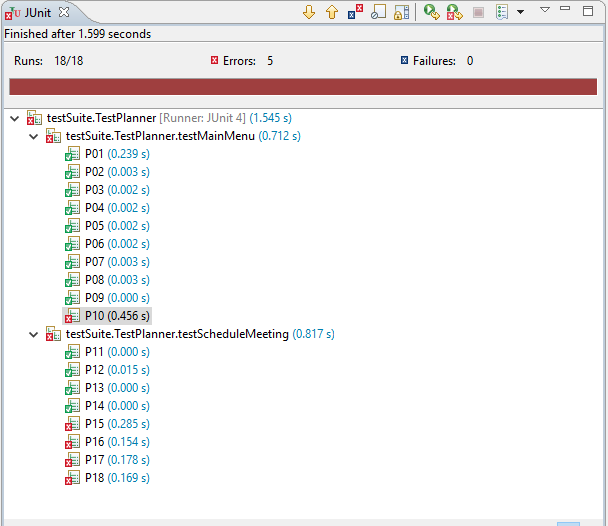
### Meeting



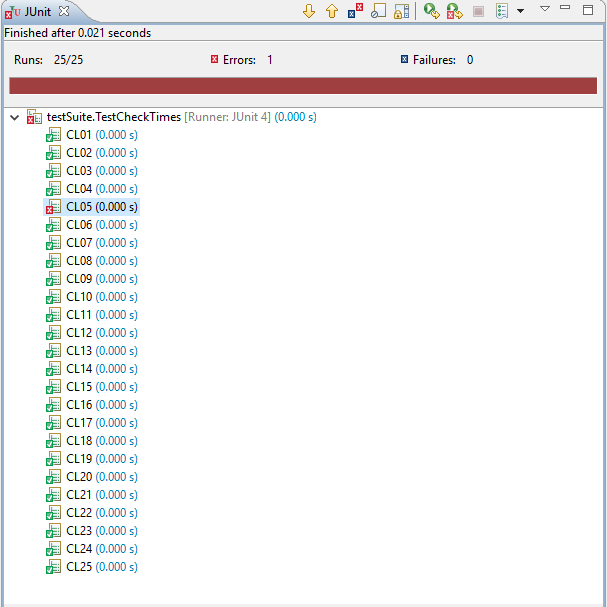
### Calendar



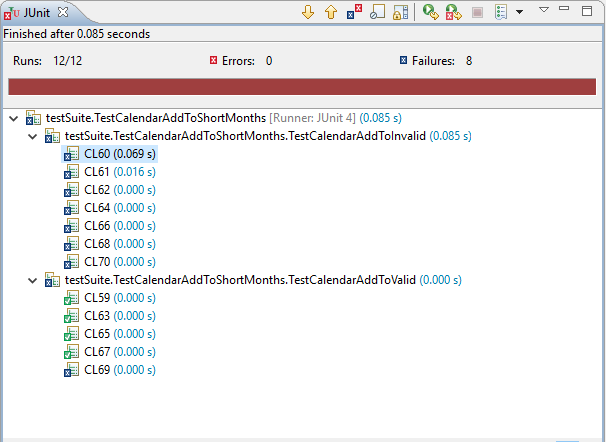
### Planner



### CheckTimes Method for Calendar Class

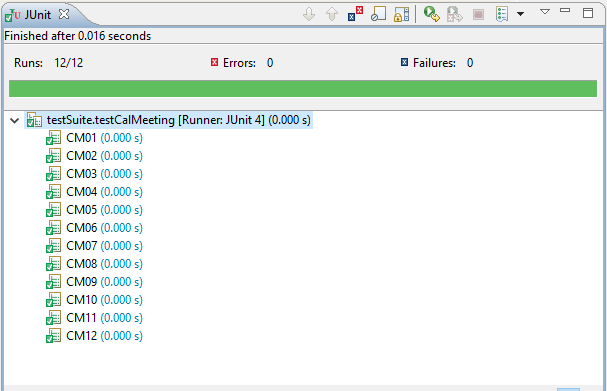


### Test Calendar add meetings to short months

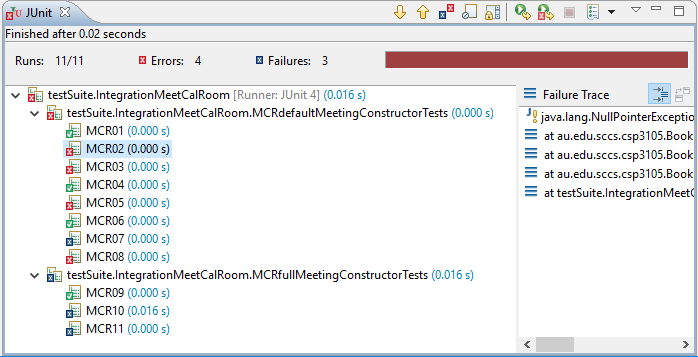


## Integration Test Executions

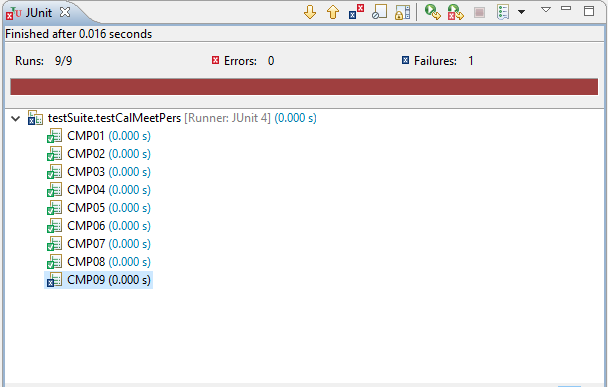
### Calendar Meeting class integration test



### Calendar Meeting Room class integration test



### Calendar Meeting Person class integration test



# Defect Report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test id | Class/Method | Expected Result | Actual Result | Description of Defect/fault | Comments |
| P10 | Planner.mainMenu() | Does not throw a StackOverflowError | Throws StackOverFlowError | See section: StackOverFlowErrors | Demonstrates a StackOverFlowError in Planner. This defect type is present in multiple methods of Planner. |
| P15 | Planner.scheduleMeeting() | Does not throw a StackOverflowError | Throws StackOverFlowError | See section: StackOverFlowErrors |  |
| P16 | Planner.scheduleMeeting() | Does not throw a StackOverflowError | Throws StackOverFlowError | See section: StackOverFlowErrors |  |
| P17 | Planner.scheduleMeeting() | Does not throw a StackOverflowError | Throws StackOverFlowError | See section: StackOverFlowErrors |  |
| P18 | Planner.scheduleMeeting() | Does not throw a StackOverflowError | Throws StackOverFlowError | See section: StackOverFlowErrors |  |
| M00 | Meeting.addAttendee(p1) | getAttendees().get(0) == p1 | NullPointerException | Adds a Person object to an instance of Meeting created with the default constructor. | The default Meeting constructor does not initialize ArrayList<Person> attendees. |
| M01 | Meeting.removeAttendee(p1) | getAttendees() == null | NullPointerException | Attempts to remove a Person from an instance of Meeting created with the default constructor. | The default Meeting constructor does not initialize ArrayList<Person> attendees. |
| M02 | Meeting.toString() | toString() != null | NullPointerException | Calls toString() on an instance of Meeting created with the default constructor. | The default Meeting constructor does not initialize ArrayList<Person> attendees. |
| CL05 | calendar.checkTimes(int, int, int, int) | valid | TimeConflictException: Month does not exist | Calls checkTimes() on a valid date, i.e. 12, 31, 1, 2. | Calendar.checkTimes() throws a TimeConflictException on any date in the month of December. |
| CMP09 | Meeting.addAttendee(attendee) Meeting.addAttendee(attendee) | Should be invalid | Is valid | Duplicate Person is successfully added to the same Meeting. | Meeting.addAttendee() should not allow multiple entrees of an attendee at a given Meeting. |
| CL60 | Calendar.addMeeting(Meeting) Calendar.getMeeting(2, 29, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL61 | Calendar.addMeeting(Meeting) Calendar.getMeeting(2, 30, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL62 | Calendar.addMeeting(Meeting) Calendar.getMeeting(2, 31, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL64 | Calendar.addMeeting(Meeting) Calendar.getMeeting(4, 31, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL66 | Calendar.addMeeting(Meeting) Calendar.getMeeting(6, 31, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL68 | Calendar.addMeeting(Meeting) Calendar.getMeeting(9, 31, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL70 | Calendar.addMeeting(Meeting) Calendar.getMeeting(11, 31, 1) | Throws IndexOutOfBoundException | Meeting is added. | Invalid Meeting is successfully added to Calendar. | Logic error in Calendar line: 125. if(!toCheck.getDescription().equals("Day does not exist")) |
| CL69 | Calendar.addMeeting(Meeting) Calendar.getMeeting(11, 30, 0) | “CL69” | “Day does not exist” | Unsuccessful attempt to add a valid Meeting to Calendar. | Error in constructor of Calendar. Blocks off a valid day. |
| CL36 | Calendar.isBusy(11,30,13,14) | Calendar.isBusy() == false | Calendar.isBusy() == true | Attempts to call isBusy() on a date that should be blocked. | Error in constructor of Calendar. Leaves invalid date unblocked. |
| CL38 | Calendar.addMeeting(meeting(2, 30, 10, 12)) | Throws TimeConflictException | Successfully adds Meeting. | Successfully adds a Meeting to a non-existent day. | Error in constructor of Calendar. Leaves invalid date unblocked. |
| CL45 | Calendar.addMeeting(Meeting(6,6,12,17) Calendar.add(Meeting(6,6,10,20)) | Throws TimeConflictException | Successfully adds Meeting | Successfully adds a Meeting to a conflicted slot | The conflict checking in addMeeting allows a meeting that consumes an existing meeting to be booked. |
| CL48 | Calendar.removeMeeting(1,1,0) | Throws TimeConflictException | Throws IndexOutOfBoundsException | Removes a Meeting for free slot | If you remove meeting when day is empty an error occurs, should handle |
| CL49 | Calendar.removeMeeting(-3,1,0) | Throws TimeConflictException | Throws IndexOutOfBoundsException | Remove Meeting from invalid date | Should check date/time before executing |
| CL50 | Calendar.removeMeeting(4, 31, 0) | Calendar.getMeeting(4, 31, 0).getDescription()  != null | Calendar.getMeeting(4, 31, 0).getDescription() == “Day-blocking meeting was removed from calendar” | Remove meeting from blocked off day in shorter month - (Should not be possible) | Should not allow the removal of blocked off day in a shorter month |
| CL52 | ClearSchedule(1, 45) | Throws TimeConflictException | Throws IndexOutOfBoundsException | Calendar.clearSchdedule of day outside bounds | Should check date/time before executing |
| CL60 | Calendar.addMeeting(2, 29, 10, 12, “CL60”) Calendar.getMeeting(2, 29, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL61 | Calendar.addMeeting(2, 30, 0, 1, “CL61”) Calendar.getMeeting(2, 30, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL62 | Calendar.addMeeting(2, 31, 22, 23, “CL62”) Calendar.getMeeting(2, 31, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL64 | Calendar.addMeeting(4, 31, 0, 23, “CL64”) Calendar.getMeeting(4, 31, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL66 | Calendar.addMeeting(6, 31, 13, 16, “CL66”) Calendar.getMeeting(6, 31, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL68 | Calendar.addMeeting(9, 31, 10, 22, “CL68”) Calendar.getMeeting(9, 31, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL69 | Calendar.addMeeting(11, 30, 13, 14, “CL69”) Calendar.getMeeting(11, 30, 0) | Calendar.getMeeting(11, 30, ).getDescription() == “CL69” | Calendar.getMeeting(11, 30, 0).getDescription() == “Day does not exist” | Test meeting CAN be added to this day | Day was incorrectly blocked off by calendar |
| CL70 | Calendar.addMeeting(11, 31, 11, 12, “CL70”) Calendar.getMeeting(11, 31, 1) | Throws IndexOutOfBoundsException | No Exception | Test that meeting can’t be added to this day | The meeting gets added. Bad validation in addMeeting |
| CL40 | Calendar.addMeeting(2, 2, 14, 15),  Calendar.addMeeting(2, 2, 13, 14) | Calendar.getMeeting(2, 2, 1).getDescription() == “CL40b” | Throws TimeConflictException | Add to free slot immediately preceding booked slot | Should be able to have adjacent meetings. Current validation means hour gap required. |
| CL41 | Calendar.addMeeting(3, 3, 14, 15),  Calendar.addMeeting(3, 3, 15, 16) | Calendar.getMeeting(3, 3, 1).getDescription() == “CL41b” | Throws TimeConflictException | Add to free slot immediately following booked slot | Should be able to have adjacent meetings. Current validation means hour gap required. |
| CL56 | Calendar.addMeeting(1, 1, 10, 14) | isBusy(1, 1, 9, 15) == true | isBusy(1, 1, 9, 15) == false | Call isBusy() on times that subsume existing meeting | If new meeting times consume the booked times, meeting is added |
| CL57 | Calendar.addMeeting(1, 1, 10, 14) | isBusy(1, 1, 9, 10) == false | isBusy(1, 1, 9, 10) == true | Call isBusy() on free slot preceding existing Meeting. | Should be able to have adjacent meetings. Current validation means hour gap required. |
| CL58 | Calendar.addMeeting(1, 1, 10, 14) | isBusy(1, 1, 14, 15) == false | isBusy(1, 1, 14, 15) == true | Call isBusy() on free slot preceding existing Meeting. | Should be able to have adjacent meetings. Current validation means hour gap required. |
| MCR02 | Room.addMeeting(1, 10) | printAgenda(1) != "No Meetings booked on this date.\n\n" | Throws NullPointer Exception | State transition from (m, d) constructor into room calendar and printAgena(m) |  |
| MCR03 | Room.addMeeting(1, 10) | printAgenda(1,10) != "No Meetings booked on this date.\n\n" | Throws NullPointer Exception | State transition from (m, d) constructor into room calendar and printAgena(m, d) |  |
| MCR05 | Meeting.addAttendee(Larry) | meeting.getAttendees().get(0).getName()) == “larry” | Throws NullPointer Exception | Adding attendee to meeting when it was constructed without an ArrayList of type <people> |  |
| MCR07 | Room.addMeeting(meeting)  Meeting.setDay(100) | Throws TimeConflictException | No Exception | Test to make sure you can't change meetings attributes to invalid state AFTER you have added it to room calendar in valid state |  |
| MCR08 | Room.addMeeting(meeting)  Room.addMeeting(conflict) | Throws TimeConflictException | Throws NullPointer Exception | still should not be able to add conflicting meetings |  |
| MCR10 | Rom.addMeeting(meeting)  Room.getMeeting(1,10,0).setRoom(wrongRoom) | room.getID() == room.getMeeting(1, 10, 0).getRoom().getID()) | room.getID() != room.getMeeting(1, 10, 0).getRoom().getID()) | test you can't change room in meeting to something other than the room its booked to |  |
| MCR11 | room.addMeeting(meeting)  room.getMeeting(1, 10, 0).addAttendee(harry) | "harry" == room.getMeeting(1, 10, 0) | "harry" != room.getMeeting(1, 10, 0) | Should not be able to add attendee after getMeeting |  |

Control flow cannot reach end of method through normal use. Each menu option in mainMenu() ends with a new instance of mainMenu(). Instances of Planner methods never leave the stack, excluding inputOutput(String).

# Discussions and conclusions

## Descriptions of errors

### StackOverFlowErrors

Control flow cannot reach the end of Planner.mainMenu() through normal use, although it can be reached through certain paths that throw exceptions. All possible paths in Planner.mainMenu(), other than System.exit(), result in either a new instance of Planner.mainMenu() or another Planner method that calls a new instance of Planner.mainMenu(). Other than Planner.inputOutput(String), all method calls in Planner result in a new stackframe that cannot be removed except through a few paths where exceptions are thrown. An example of a path non-numeric Strings entered to inputOutput() in nodes 1-4 of scheduleMeeting() and scheduleVacation().

Each menu option in mainMenu() ends with a new instance of mainMenu(). Instances of Planner methods never leave the stack, excluding inputOutput(String).

Control flow cannot reach the end of Planner.mainMenu() through normal use, although it can be reached through certain paths that throw exceptions. All possible paths in Planner.mainMenu(), other than System.exit(), result in either a new instance of Planner.mainMenu() or another Planner method that calls a new instance of Planner.mainMenu(). Other than Planner.inputOutput(String), all method calls in Planner result in a new stackframe that cannot be removed except through a few paths where exceptions are thrown. An example of a path that can remove a stackframe, entering a non-numeric String at any one of the inputOutput() nodes 1-4 of scheduleMeeting() or scheduleVacation(), will result in an exception that is caught in mainMenu() and allows both of the topmost stackframes to be removed.

### Tier 3 anomaly

When a complete meeting is instantiated it includes a room object and an arraylist of persons. Each of these contain their own calendar which may/may not contain already existing meetings. Each time a meeting with room and person attributes is added to a room and person, another layer of meeting/calendar is created. Essentially, we end up with a room that holds a meeting, that contains a room attribute that has its own calendar that holds meetings that have a room and so on and so forth. This can occur an infinite number of times.

## Testing process

### Original Plan - Procedural testing approach

Early on in this undertaking, our plan was to systematically test each individual function of the classes, using a procedural testing approach. A diagram of the hierarchy was created and Meeting was determined to be the base class. We set about attempting to perform exhaustive procedural testing on each method of each class, in the order shown in the hierarchy. It was intended that we would then move onto 'integration testing', performed in a similar manner. Attempting to do integration testing with this approach revealed this procedural testing method to be inappropriate for object oriented testing. The plan was revised.

### Updated plan - Object oriented testing approach

Following is a description of the object oriented testing approach adopted towards the end of the project. At this stage the class hierarchy developed in the original plan continued to be used, maintaining Meeting as the identified base class.

#### Unit testing

The following process was intended to be used starting from the base class, Meeting, and working up the hierarchy. The approach primarily involves performing path testing

1. Identify the instance variables – In the case of unit testing, these are the attributes of the class under test.
2. Identify defs - in unit testing, are any instantiations or re-defs of class attributes. As is the case in Planner. They can also be instantiations of a method on to the stack, I.e. Planner.mainMenu() calling this.mainMenu().
3. Identify DU-pairs - of class attributes, this refers to any use of a given def. For the defs of method instantiations, instead of finding a DU-pair you are looking for new definitions.
4. Implementation – determine paths and write test cases to traverse them.

#### Integration testing

The following process was intended to be used starting from the second tier of the hierarchy and working up to and including the fourth.

1. Identify the object instances – in the case of integration testing, the object instances are defs.
2. Identify inter-class DU pairs – in integration testing, a DU pair is any use of a object instance occurring from another class in the current tier.
3. Implementation – identify possible states and paths between them. This can include paths like instantiating a valid state then transition to an invalid state before traversing a use path.

## Conclusion

We endeavored to test the program in an object-oriented manner, not just in terms of techniques but also in terms of test requirements. This means that we tested under the assumption that the classes should be loosely coupled enough that they do not rely on validation at a higher level of functionality in order for the program to successfully run. What we found was that we gravitated towards "method testing" for too long a time before implementing an approach that focuses on the state of the class object and the possible transitions between states. After adopting an OO approach, black-box testing techniques were employed in areas that we deemed them to be necessary. Similarly, areas were identified where we would use weak EP testing as well as likely problem areas that would require strong EP testing. We can advise many recommendations for this software. The prevailing issue is the lack of cohesion and prevalence of coupling between classes, primarily between the attributes of Organization, Room, Person, and Meeting.