MAKERERE UNIVERSITY

SOFTWARE TESTING AND VERIFICATION COURSEWORK INFORMAL TEST PLAN

LECTURER : Dr. Alex Mwotil

Name	**Registration Number**	**Student Number**
Katwebaze Emmanuel	21/U/0618/PS	2100706815
Nammiiro Zaharah Bidin	21//U/1614	2100701614
Nabbona Prossy	21/U/11450/PS	2100711450
Mwegyesa Saul	21/U/0567	2100700567
Ssenteza Emmanuel	21/U/13955/PS	2100713955

1. Informal Testing Plan

a) Calendar class:

a. Default constructor:

- Check if the calendar is initialized correctly i.e. Not Null. Create new calendar instance and ensure it returns “not null”.
- Check if the calendar is initialized with days that do not exist. Create a new calendar instance and get the different meetings in the days that should not exist, in the different months. Ensure that their descriptions equal “Day does not exist”.

b. isBusy method:

- Test for a time frame that is not busy. Create a new calendar instance and check the busy status for a random non-busy month, day, start time and end time. Ensure the status is false.
- Test for a time frame that overlaps with an existing meeting. Create a new calendar instance and check the busy status for a random non-busy month, day, start time and end time. Ensure the status is false.
- Test for invalid month, but valid day, start time, and end time. Create a new calendar instance, pass an invalid month e.g. 13 to the isBusy function and expect a TimeConflictException to be thrown.
- Test for invalid day, but valid month, start time, and end time. Create a new calendar instance, pass an invalid day e.g. 32 to the isBusy function and expect a TimeConflictException to be thrown.
- Test for invalid start time, but valid day, month, and end time. Create a new calendar instance, pass an invalid start time e.g. -1 to the isBusy function and expect a TimeConflictException to be thrown.
- Test for invalid end time, but valid day, month, and start time. Create a new calendar instance, pass an invalid end time e.g. 24 to the isBusy function and expect a TimeConflictException to be thrown.

- Test for end time before start time. Create a new calendar instance, pass an end time that is before a start time e.g. start time as 10 and end time as 8 to the isBusy function and expect a TimeConflictException to be thrown.
- Test for start time being equal to end time. Create a new calendar instance, pass an start time that is equal to the end time e.g. start time as 8 and end time as 8 to the isBusy function and expect a TimeConflictException to be thrown.

c. **Test checkTimes method:**

- Test for valid month, day, start time, and end time. Call the checkTimes method through the Calendar class since it's a static method passing in valid inputs. The call shouldn't throw a TimeConflictException else it fails.
- Test for invalid month, but valid day, start time, and end time. Call the checkTimes method through the Calendar class since it's a static method passing in an invalid month. Expect the call to throw a TimeConflictException else it fails.
- Test for invalid day, but valid month, start time, and end time. Call the checkTimes method through the Calendar class since it's a static method passing in an invalid day. Expect the call to throw a TimeConflictException else it fails.
- Test for invalid start time, but valid day, month, and end time. Call the checkTimes method through the Calendar class since it's a static method passing in an invalid start time. Expect the call to throw a TimeConflictException else it fails.
- Test for invalid end time, but valid day, month, and start time. Call the checkTimes method through the Calendar class since it's a static method passing in an invalid end time. Expect the call to throw a TimeConflictException else it fails.
- Test for end time before start time. Call the checkTimes method through the Calendar class since it's a static method passing in inputs where the

end time is before the start time. Expect the call to throw a TimeConflictException else it fails.

- Test for start time being equal to end time. Call the checkTimes method through the Calendar class since it's a static method passing in inputs where the start time is equal to the end time. Expect the call to throw a TimeConflictException else it fails.

d. Test addMeeting method:

  - Test adding a meeting on a non-busy day. Create a new calendar instance, create a new meeting with valid inputs, and add the meeting to the calendar. Check the busy status for the calendar on the particular day set when creating the meeting. Ensure the status is True.
  - Test adding a meeting with a busy time frame. Create a new calendar instance, create a new meeting with valid inputs, and add the meeting to the calendar. Then create another meeting with the same input parameters as the first one and try adding it to the calendar. It should throw a TimeConflictException.
  - Test adding a meeting with invalid month, and valid day, start time, and end time.

- Test adding a meeting for invalid month, but valid day, start time, and end time. Create a new calendar instance. Then create a new meeting instance and pass an invalid month e.g. 35 to the meeting constructor. Add the meeting to the calendar and expect a TimeConflictException to be thrown.
- Test adding a meeting for invalid day, but valid month, start time, and end time. Create a new calendar instance. Then create a new meeting instance and pass an invalid day e.g. 32 to the meeting constructor. Add the meeting to the calendar and expect a TimeConflictException to be thrown.
- Test adding a meeting for invalid start time, but valid day, month, and end time. Create a new calendar instance. Then create a new meeting instance and pass an invalid start time e.g. -1 to the meeting constructor. Add the meeting to the calendar and expect a TimeConflictException to be thrown.

- Test adding a meeting for invalid end time, but valid day, month, and start time. Create a new calendar instance. Then create a new meeting instance and pass an invalid end time e.g. 24 to the meeting constructor. Add the meeting to the calendar and expect a TimeConflictException to be thrown.
- Test adding a meeting for end time before start time. Create a new calendar instance. Then create a new meeting instance and pass an end time that is less than the start time e.g. start time – 12, end time -10, to the meeting constructor. Add the meeting to the calendar and expect a TimeConflictException to be thrown.
- Test adding a meeting for start time being equal to end time. Create a new calendar instance. Then create a new meeting instance and pass an end time that is equal to the start time e.g. start time – 8, end time - 8, to the meeting constructor. Add the meeting to the calendar and expect a TimeConflictException to be thrown.

e. **Test clearSchedule method:**

- Test clearing schedule for a valid month and day. Create a new calendar instance. Then create a new valid meeting instance and add the meeting to the calendar. Clear the schedule for that particular month and day of the newly added meeting then try to access the meetings on that particular month and day and expect an IndexOutOfBoundsException to be thrown.
- Test clearing schedule for an invalid month and day. Create a new calendar instance. Then without adding any new meeting, clear a schedule for a meeting for an invalid month and invalid day. Expect an IndexOutOfBoundsException.

f. **Test printAgenda method:**

- Test printing agenda for a valid month. Create new calendar and room instances. Then create a list of attendees to which different attendees will be added. Create a new meeting passing in a particular valid month, and other time inputs including the attendees, room and meeting description. Then print the agenda for that particular month alone and expect the agenda to contain the description of the meeting earlier entered.

- Test printing agenda for a valid month and day. Create new calendar and room instances. Then create a list of attendees to which different attendees will be added. Create a new meeting passing in a particular valid month and day, and other time inputs including the attendees, room and meeting description. Then print the agenda for that particular month and day and expect the agenda to contain the description of the meeting earlier entered.
- Test printing agenda for free month. Create a new calendar instance and print an agenda for a random month. The agenda should equal "Agenda for {month}:" where {month} is the entered month.
- Test printing agenda for free month and day. Create a new calendar instance and print an agenda for a random month and day. The agenda should equal "Agenda for {month}/{day}:" where {month} and {day} are the entered month and day respectively.
- Test printing agenda for an invalid month. Create a new calendar instance and print an agenda for a random invalid month. Expect an IndexOutOfBoundsException to be thrown.
- Test printing agenda for an invalid month and valid day. Create a new calendar instance and print an agenda for a random invalid month and valid day. Expect an IndexOutOfBoundsException to be thrown.
- Test printing agenda for a valid month and invalid day. Create a new calendar instance and print an agenda for a random valid month and invalid day. Expect an IndexOutOfBoundsException to be thrown.

g. Test getMeeting method:

- Test retrieving a meeting for occupied month and day. Create new calendar and meeting instances for a particular month and day. Add the meeting to the calendar. Retrieve the newly created meeting and assert that its description matches the description that was passed during creation of the meeting.
- Test retrieving a meeting for free month and day. Create new calendar instance and get a meeting for a random month and day that’s valid that doesn’t contain a meeting. Expect an IndexOutOfBoundsException.

- Test retrieving a meeting for a valid month and invalid day. Create new calendar instance and get a meeting for a random valid month and day that’s invalid that doesn’t contain a meeting. Expect an IndexOutOfBoundsException.
- Test get a meeting for invalid month and valid day. Create new calendar instance and get a meeting for a random invalid month and day that’s valid that doesn’t contain a meeting. Expect an IndexOutOfBoundsException.
- Test removing a meeting for valid month and invalid day. Create new calendar instance and get a meeting for a random valid month and day that’s invalid that doesn’t contain a meeting. Expect an IndexOutOfBoundsException.
- Test retrieve a meeting for occupied month, day and invalid index. Create new calendar and meeting instances for a particular month and day. Add the meeting to the calendar and try to get the newly created meeting but pass in a wrong index. Expect an IndexOutOfBoundsException.

h. Test removeMeeting method:

- Test removing a meeting for occupied month and day. Create new calendar and meeting instances for a particular month and day. Add the meeting to the calendar, remove the meeting and try to get the meeting description for the previously removed meeting. Expect an IndexOutOfBoundsException.
- Test removing a meeting for free month and day. Create new calendar instance and remove a meeting for a random month and day that’s valid that doesn’t contain a meeting. Expect an IndexOutOfBoundsException.
- Test removing a meeting for invalid month and valid day. Create new calendar instance and remove a meeting for a random invalid month and day that’s valid that doesn’t contain a meeting. Expect an IndexOutOfBoundsException.
- Test removing a meeting for valid month and invalid day. Create new calendar instance and remove a meeting for a random valid month and day

that's invalid that doesn't contain a meeting. Expect an IndexOutOfBoundsException.

- Test remove a meeting for occupied month, day and invalid index. Create new calendar and meeting instances for a particular month and day. Add the meeting to the calendar and try to remove the newly created meeting but pass in a wrong index. Expect an IndexOutOfBoundsException.
- Test removing a meeting for free month and day and invalid index. Create new calendar instance and remove a meeting for a random month and day that's valid that doesn't contain a meeting but pass in a wrong index. Expect an IndexOutOfBoundsException.

b) Meeting class:

Initial setup for the test involves creating a meeting that is initialized with attendees and is assigned to a particular room.

a. Test meeting and room initialized correctly:

- Given the setup has been initialized, assert that the meeting and room that have been initialized are not null.

b. Test add attendee:

- Given the setup has been initialized, a new instance of a person is created and then added to the meeting. Assert that the attendees list in the meeting that is created in the setup contains the name of the newly add attendee.

c. Test remove attendee:

- Given the setup has been initialized, retrieve the attendee to remove from the already made list of attendees. Make sure that checking whether attendees list contains the attendee that was just removed returns false.

d. Test toString method:

- Given the setup has been initialized, assert that the string form of the meeting equals the expected string pattern.

e. Test getters and setters:

- Given the setup has been initialized, carry out different assertions on the month, day, start time, end time, description and room id. Verifying the ability to use the setter by assigning new parameters to the meeting and asserting whether the new parameters are equal to their intended new values.

c) **Person class:**

Setup for before each test involves creating a person instance.

a. **Test add meeting:**
   - Given the setup has been initialized, create a new meeting for a particular month and day and add it to the person. Assert the busy status of the person for that particular month and day to be true.

b. **Test print agenda for month:**
   - Given the setup has been initialized, print the agenda for the particular person for a particular month and it should not be null.

c. **Test print agenda for day:**
   - Given the setup has been initialized, print the agenda for the particular person for a particular month and day and it should not be null.

d. **Test isBusy method when not busy:**
   - Given the setup has been initialized, check the busy state of the initialized person and it should be false.

e. **Test isBusy method when busy:**
   - Given the setup has been initialized, create a new check the busy state of the initialized person and it should be false.

f. **Test getMeeting method:**
   - Given the setup has been initialized, create a new meeting for a particular month and day, add it to the person. Retrieve the meeting from the person for that particular month and day, plus valid index. Assert that it isn't null.

g. **Test removeMeeting method:**
   - Given the setup has been initialized, create a new meeting for a particular month and day, add it to the person. Remove the meeting from the person for that particular month and day, plus valid index. Assert the busy status of the person for that particular month and day to be false.

d) **Room class:**

   a. **Test default constructor:**
      - Create a new instance of a room and assert that the room is not null and the room id is an empty string.

   b. **Test parametrized constructor:**
      - Create a new instance of a room passing in a certain id and assert that the room is not null and that the room id is equal to the string that was passed on creation of the id.

   c. **Test add meeting:**
      - Create new instances of a room and meeting, add the meeting to the room and assert that the busy state for the room is true for the specific time period of the added meeting.

   d. **Test print agenda by month:**
      - Create new instances of a room and assert that the agenda for the created room for a particular month is not null.

   e. **Test print agenda by month and day:**
      - Create new instances of a room and assert that the agenda for the created room for a particular month and day is not null.

   f. **Test isBusy method when not busy:**
      - Create new instances of a room and assert that the busy state for the room is false for a specific time period.

   g. **Test isBusy method when busy:**

- Create new instances of a room and meeting, add the meeting to the room and assert that the busy state for the room is true for the specific time period of the added meeting.

h. **Test getMeeting method:**
   - Create new instances of a room and meeting, add the meeting to the room and assert that the newly added meeting to the room is equal to the one that was just initially created.

i. **Test removeMeeting method:**
   - Create new instances of a room and meeting, add the meeting to the room and remove the same meeting. Assert that the busy state for the room is false for the specific time period of the just removed meeting.

e) **Organization class:**

Initial setup for the test involves creating an organization instance.

a. **Test getEmployees method:**
   - Get the employees list from the organization, assert that it's not null and assert that the size of the list is equal to 5.

b. **Test getRooms method:**
   - Get the rooms list from the organization, assert that it's not null and assert that the size of the list is equal to 5.

c. **Test getRoom method for valid room:**
   - Get a particular room from the organization by passing in a valid room id. Assert that the room is not null.

d. **Test getRoom method for invalid room:**
    - Get a particular room from the organization by passing in an invalid room id. Expect an exception to occur and assert that the exception message is equal to “Requested room does not exist”.

e. **Test getEmployee method for valid employee:**
    - Get a particular employee from the organization by passing in a valid employee name. Assert that the employee is not null.

f. **Test getEmployee method for invalid employee:**
    - Get a particular employee from the organization by passing in an invalid employee name. Expect an exception to occur and assert that the exception message is equal to “Requested employee does not exist”.