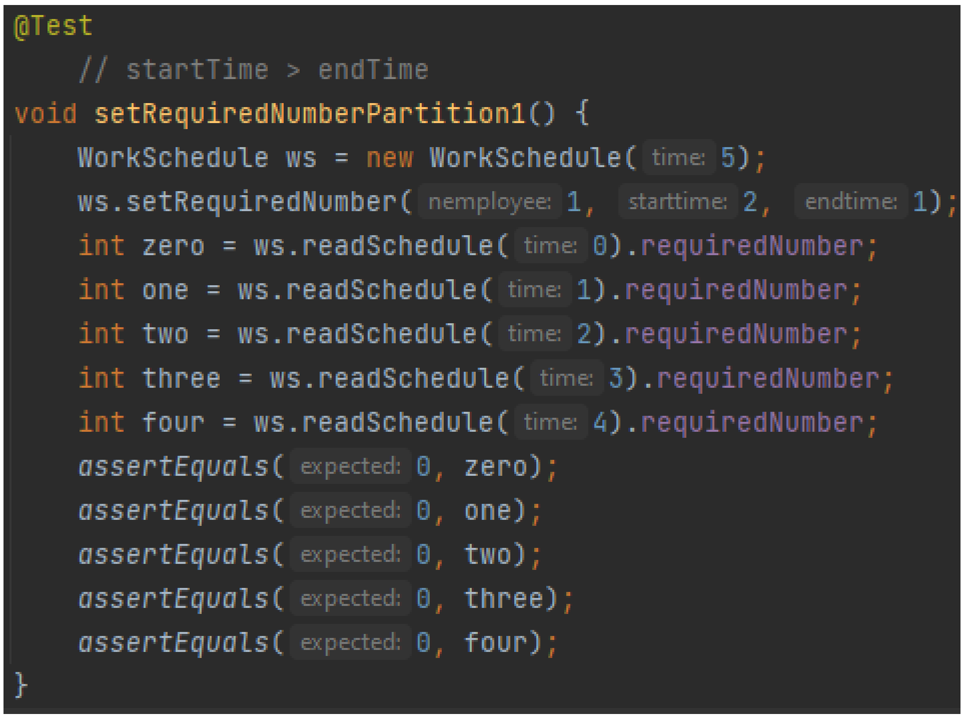
Method a)

Input space: nemployee >= 0, starttime >= 0, endttime >= 0

Partition #1: starttime > endtime, schedule

test_setRequiredNumber_part1: input: “starttime = 2, endtime = 1”, expected: No changes made to requiredNumber on any WorkSchedules.

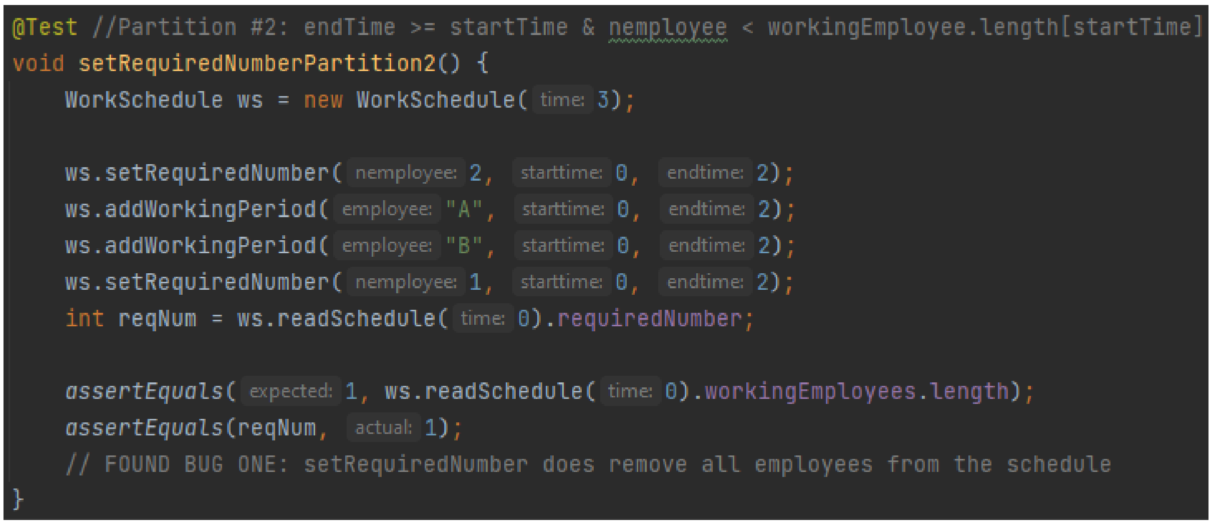


```java
@Test
    // startTime > endTime
void setRequiredNumberPartition1() {
    WorkSchedule ws = new WorkSchedule( time: 5);
    ws.setRequiredNumber( nemployee: 1,  starttime: 2,  endtime: 1);
    int zero = ws.readSchedule( time: 0).requiredNumber;
    int one = ws.readSchedule( time: 1).requiredNumber;
    int two = ws.readSchedule( time: 2).requiredNumber;
    int three = ws.readSchedule( time: 3).requiredNumber;
    int four = ws.readSchedule( time: 4).requiredNumber;
    assertEquals( expected: 0, zero);
    assertEquals( expected: 0, one);
    assertEquals( expected: 0, two);
    assertEquals( expected: 0, three);
    assertEquals( expected: 0, four);
}
```

Explanation for test for partition **#1:** If endtime is greater than startTime no changes should be made to the schedule.

Partition #2: endtime >= starttime & nemployee < workingEmployee.length[starttime]

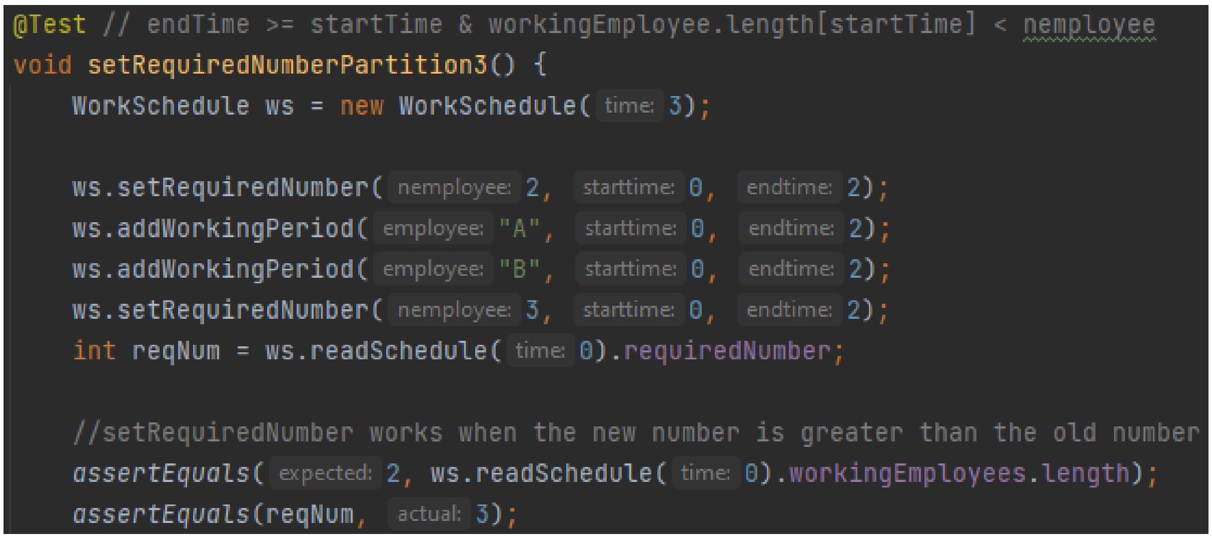
test_setRequiredNumber_part3: input: “nemployee = x, starttime = 0, endtime = 2”,
expected: this.schedule[starttime].requiredNumber = x
workingEmployee.length[starttime] = x


```
@Test //Partition #2: endTime >= startTime & nemployee < workingEmployee.length[startTime]
void setRequiredNumberPartition2() {
    WorkSchedule ws = new WorkSchedule( time: 3);

    ws.setRequiredNumber( nemployee: 2,  starttime: 0,  endtime: 2);
    ws.addWorkingPeriod( employee: "A",  starttime: 0,  endtime: 2);
    ws.addWorkingPeriod( employee: "B",  starttime: 0,  endtime: 2);
    ws.setRequiredNumber( nemployee: 1,  starttime: 0,  endtime: 2);
    int reqNum = ws.readSchedule( time: 0).requiredNumber;

    assertEquals( expected: 1, ws.readSchedule( time: 0).workingEmployees.length);
    assertEquals(reqNum,  actual: 1);
    // FOUND BUG ONE: setRequiredNumber does remove all employees from the schedule
}
```

Explanation for test for partition **#2:** The method setRequiredNumber removes all employees from the schedule when we are using it to <u>decrease</u> the number of working employees. requiredNumber is still updated correctly though.

Partition #3: endtime >= starttime & workingEmployee.length[starttime] < nemployee

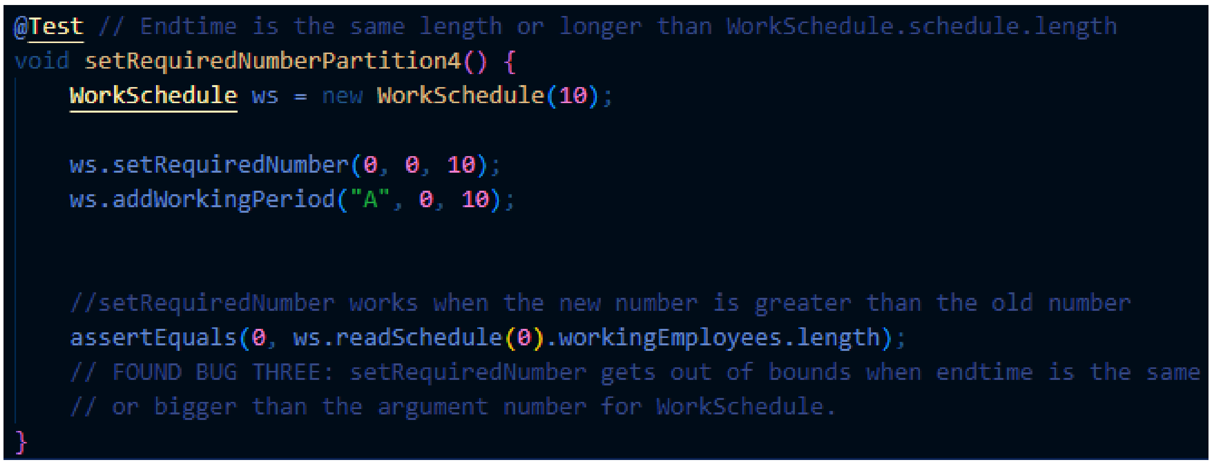
test_setRequiredNumber_part4: input: “nemployee = x, starttime = 0, endtime = 2”,
expected: this.schedule[starttime].requiredNumber = x


```
@Test // endTime >= startTime & workingEmployee.length[startTime] < nemployee
void setRequiredNumberPartition3() {
    WorkSchedule ws = new WorkSchedule( time: 3);

    ws.setRequiredNumber( nemployee: 2,  starttime: 0,  endtime: 2);
    ws.addWorkingPeriod( employee: "A",  starttime: 0,  endtime: 2);
    ws.addWorkingPeriod( employee: "B",  starttime: 0,  endtime: 2);
    ws.setRequiredNumber( nemployee: 3,  starttime: 0,  endtime: 2);
    int reqNum = ws.readSchedule( time: 0).requiredNumber;

    //setRequiredNumber works when the new number is greater than the old number
    assertEquals( expected: 2, ws.readSchedule( time: 0).workingEmployees.length);
    assertEquals(reqNum,  actual: 3);
```

Explanation for test for partition **#3:** The method setRequiredNumber works when we <u>increase</u> the number of required people to work where we can clearly see that requiredNumber is updated.

Partition #4: WorkSchedule.schedule.length <= endTime

test_setRequiredNumber_part5: input: “nemployee = 0, starttime = 0, endtime = 10, Workschedule(10)”
expected: this.schedule[starttime].requiredNumber = 0



```java
@Test // Endtime is the same length or longer than WorkSchedule.schedule.length
void setRequiredNumberPartition4() {
    WorkSchedule ws = new WorkSchedule(10);

    ws.setRequiredNumber(0, 0, 10);
    ws.addWorkingPeriod("A", 0, 10);


    //setRequiredNumber works when the new number is greater than the old number
    assertEquals(0, ws.readSchedule(0).workingEmployees.length);
    // FOUND BUG THREE: setRequiredNumber gets out of bounds when endtime is the same
    // or bigger than the argument number for WorkSchedule.
}
```

Explanation for test for partition **#4:** The method setRequiredNumber for-loop got out of bounds when we tried setting endtime to the same or bigger value than the WorkSchedules time slots.

Border cases:
test_setRequiredNumber_1: input: “nemployee = endtime = starttime = 0”, “schedule”
expected: “WorkSchedule.schedule = schedule”

test_setRequiredNumber_2: input: “nemployee = endtime = starttime = MAX_INT”, “schedule”
expected: “WorkSchedule.schedule = schedule”

test_setRequiredNumber_3: input: “nemployee = x, endtime = MAX_INT, starttime = 0”, “schedule”
expected: this.schedule[0..MAX_INT].requiredNumber = x
(IF working_employees > nemployees working_employees are set to nemployees otherwise working_employees are left unchanged)

test_setRequiredNumber_4: input: “ nemployee = x, endtime = 0,
starttime = MAX_IN ”, ”schedule”
expected: “WorkSchedule.schedule = schedule”

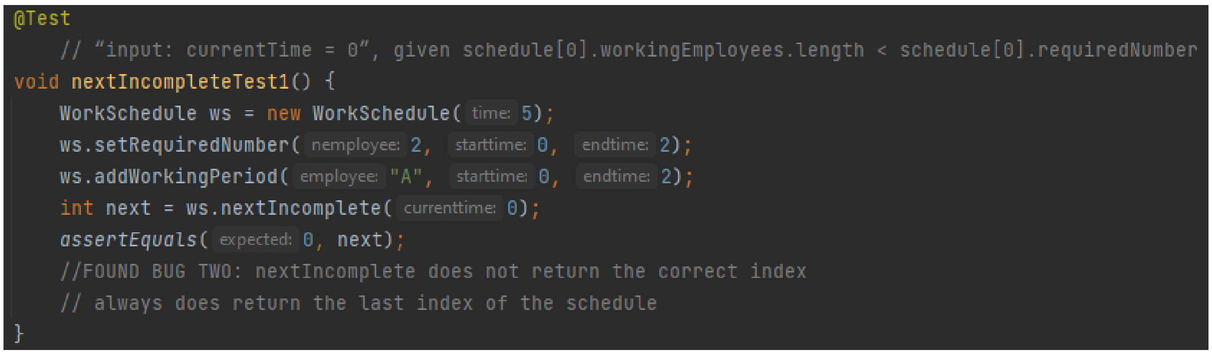
Explanation of test cases: We made our test cases to cover all of the different situations that can occur in setRequiredNumber and did them within the bounds of our input space.

<u>Method b)</u>

Input space: currentTime >= 0 and currenttime < size

Partition #1: currentTime < size

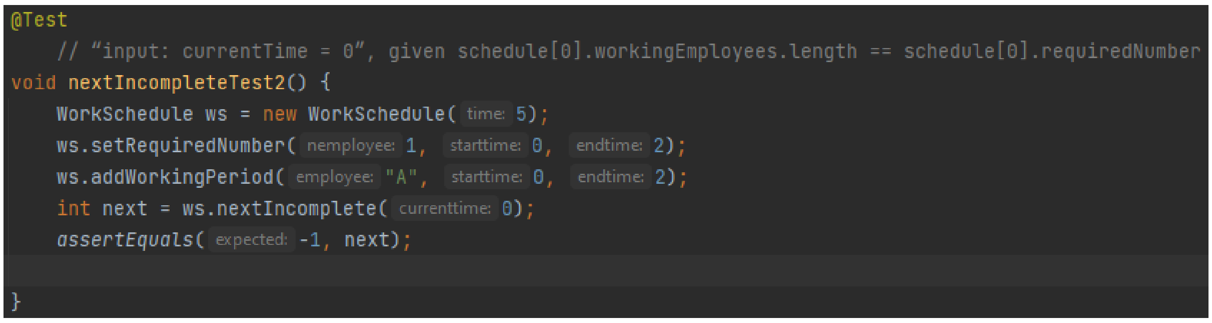
test_nextIncomplete_part1: “input: currentTime = 0”, given schedule[0].workingEmployees.length < schedule[0].requiredNumber
expected: nextIncomplete(0) = 0,



```
@Test
    // “input: currentTime = 0”, given schedule[0].workingEmployees.length < schedule[0].requiredNumber
void nextIncompleteTest1() {
    WorkSchedule ws = new WorkSchedule( time: 5);
    ws.setRequiredNumber( nemployee: 2,  starttime: 0,  endtime: 2);
    ws.addWorkingPeriod( employee: "A",  starttime: 0,  endtime: 2);
    int next = ws.nextIncomplete( currenttime: 0);
    assertEquals( expected: 0, next);
    //FOUND BUG TWO: nextIncomplete does not return the correct index
    // always does return the last index of the schedule
}
```

Explanation of test1 for partition **#1:** We found that the method nextIncomplete will always return the last index of the schedule, this is because the program gets stuck in the for-loop until it is finished traversing.

test_nextIncomplete_part2: “input: currentTime = 0”, given schedule[0..size].workingEmployees.length == schedule[0..size].requiredNumber
expected: nextIncomplete(0) = -1,



```
@Test
    // “input: currentTime = 0”, given schedule[0].workingEmployees.length == schedule[0].requiredNumber
void nextIncompleteTest2() {
    WorkSchedule ws = new WorkSchedule( time: 5);
    ws.setRequiredNumber( nemployee: 1,  starttime: 0,  endtime: 2);
    ws.addWorkingPeriod( employee: "A",  starttime: 0,  endtime: 2);
    int next = ws.nextIncomplete( currenttime: 0);
    assertEquals( expected: -1, next);

}
```

Explanation of test2 for partition **#1:** The tests checks that if nemployees is equal to the requiredNumber for all

Border case:
test_nextIncomplete_1: “input: currentTime = MAX_INT - 1”

```
        if (schedule[MAX_INT - 1].workingEmployees.length <  schedule[MAX_INT -
        1].requiredNumber)
                expected: nextIncomplete(MAX_INT - 1) = MAX_INT - 1
        else
                expected: nextIncomplete(MAX_INT - 1) = - 1
```