CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Exercise Session 2: Unit Testing

Gregory Gay
DIT635 - February 7, 2020

Enter… The Planning System

- Code on Canvas:
  - https://canvas.gu.se/courses/25762/files/folder/Misc?preview=2280199
- Everybody likes meetings.
  - Not true - but we need to book them.
- We don't want to double-book rooms or employees for meetings.
- System to manage schedules and meetings.

The Planning System

Offers the following high-level features:

1. Booking a meeting
2. Booking vacation time
3. Checking availability for a room
4. Checking availability for a person
5. Printing the agenda for a room
6. Printing the agenda for a person

Develop a Test Plan

In groups, come up with a test plan for this system.

- Given the above features and the code documentation, plan out a series of test cases to ensure that these features can be performed without error.

Food for Thought

- What are the “testable units”?
  - Your tests may use any of the classes in the system, and may be at the method, class, or system level.
- Think about both normal execution and illegal inputs/actions.
  - How many things can go wrong?
  - You will probably be able to add a normal meeting, but can you add a meeting for February 35th?
  - Try it out - you have the code.

Develop Unit Tests

- If a test is supposed to cause an exception to be thrown. Make sure you check for that exception.
- Make sure that your expected output is detailed enough to ensure that - if something is supposed to fail - that it fails for the correct reasons.

Find Any Faults?

Did You Find the Faults?

1: getMeeting and removeMeeting perform no error checking on dates.

```
public Meeting getMeeting(int month, int day, int index){
    return occupied.get(month).get(day).get(index);
}

public void removeMeeting(int month, int day, int index){
    occupied.get(month).get(day).remove(index);
}
```

Did You Find the Faults?

2: Calendar has a 13th month.

```
public Calendar(){
         occupied = new ArrayList<ArrayList<ArrayList<Meeting>>>();

         for(int i=0;i<=13;i++){
              // Initialize month
              occupied.add(new ArrayList<ArrayList<Meeting>>());
              for(int j=0;j<32;j++){
                   // Initialize days
                   occupied.get(i).add(new ArrayList<Meeting>());
              }
         }
}
```

Did You Find the Faults?

3: November has 30 days.

Oh - and we just added a meeting to a day with a date that does not match that date.

```
occupied.get(11).get(30).add(new Meeting(11,31,"Day does not
exist"));
```

Did You Find the Faults?

4: Used a >= in checking for illegal times. December no longer exists.

```
if(mMonth < 1 || mMonth >= 12){
           throw new TimeConflictException("Month does not 
exist.");
}
```

Did You Find the Faults?

5: We should be able to start and end a meeting in the same hour.

```
if(mStart >= mEnd){
    throw new TimeConflictException("Meeting starts before it ends.");
}
```

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

What Other Faults Did You Find?

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY