Assignment 2 - G03P02

Group information

- Ana Inês Oliveira de Barros - `up201806593@fe.up.pt`;
- João de Jesus Costa - `up201806560@fe.up.pt`

Function Selection Process

The aim of this assignment is to perform black-box testing. This is problematic because none of methods in the code are documented (e.g. javadoc). In order to find the purpose of each method, we needed to follow our intuition about the names of the methods, arguments, and classes. Having more extensive documentation would allow for better black-box testing.

Since we aren't using mocks, we tried to test methods that didn't depend on other objects of the project. We discarded functions belonging to the `gui` package due to its dependence on *swing*. The `misc` package was also discarded since it only contains one function (not enough for the completion of the assignment). Functions related to elapsed time were also ignored.

The selected package for testing was the `de.dominik_geyer.jtimesched.project` package.

Method 1

Method: `public static int parseSeconds(String strTime)` in `ProjectTime.java` line 36.

Purpose: This function receives a string representing time, in `hh:mm:ss` format, and returns the total number of seconds it represents.

Reason for selection: This method deals with parsing of user input, which needs to be robust.

Steps

1. Identify the parameters:
    - `strTime` - string representing time.
2. Characteristics of the parameters
    - The string should represent a valid time in the format: `hh:mm:ss`
    - *hh* represents the hours
    - *mm* represents the minutes
    - *ss* represents the seconds
    - It should be possible to pass single digits for each component
3. Add constraints
    - Negative time is not allowed.
    - Seconds lie within the interval [0, 59].
    - Minutes lie within the interval [0, 59].

4. Generate combinations

Partition	Input	Expected outcome
Negative hours	`-1:00:00`	Thrown exception
Negative minutes	`00:-1:00`	Thrown exception
Negative seconds	`00:00:-1`	Thrown exception
Minutes out of bounds	`00:60:00`	Thrown exception
Seconds out of bounds	`00:00:60`	Thrown exception
Not a number	`a:b:c`	Thrown exception
Missing component	`00:00`	Thrown exception
One digit minute	`00:1:00`	60
One digit second	`00:00:01`	1
Two digit minutes	`00:59:00`	3540
Two digit seconds	`00:00:59`	59

Unit Tests

We created three tests: one for valid inputs, one for missing components, and one for invalid inputs.

- Valid inputs test:
  - We pass a string that should be considered valid and result in an output;
  - E.g. "00:59:01".
- Missing components test:
  - We pass a string that is missing one of that components;
  - We noticed that the parsing fails, so we consider it the expected behaviour;
  - The method could be adapted to accept this king of input as valid;
  - E.g. "00:00".
- Invalid inputs test:
  - Test inputs that are known to be invalid: invalid characters (e.g. letters), negative time, minutes/seconds out-of-range, etc...
  - The parsing of this should always result on a failure;
  - E.g. "00:60:00".

Results

All the test-cases pass successfully.

Method 2

Method: `public void adjustSecondsToday(int secondsToday)` in `Project.java` line 192.

Purpose: This function receives an integer representing the number of seconds that it took to complete a task. Then, the function updates the number of seconds spent on the task today, as well as the overall time spent on it.

Reason for selection: Throughout the usage of the application, the user is able to set the time spent on a task. As such, this is a method that deals with user input, so it needs to be reliable.

Steps

1. Identify the parameters:
    - `secondsToday` - integer representing the number of seconds.
2. Characteristics of the parameters
    - The integer should represent a positive number between 0 and infinite.
3. Add constraints
    - Negative time is not allowed.
4. Generate combinations

Partition	Input	Expected outcome
Negative seconds	-1	0
Immediately below secondsToday complement	- (secondsToday - 1)	0
Negative time today	- secondsToday	0
Immediately above secondsToday complement	- (secondsToday + 1)	0
Zero	0	0
Immediately below secondsToday	secondsToday - 1	secondsToday - 1
Equal to secondsToday	secondsToday	secondsToday
Immediately above secondsToday	secondsToday + 1	secondsToday + 1

Unit Tests

We created two tests: one for valid inputs and another one for invalid inputs.

- Valid inputs test:
    - We try setting the value to 0 and to the values around the current number of *seconds spent today* on the task;
    - This should be valid and result on the number of seconds today being adjusted to the given number of seconds;
    - Furthermore, the overall time should be adjusted accordingly;
    - E.g. `10`.
- Invalid inputs test:

- We try adjusting the time to negative values, and cause the time delta to be negative;
- This tests whether the method can handle negative values, and negative time deltas;
- This should result in the *seconds today* being 0 and the overall time being adjusted accordingly (consider the input as 0);
- E.g. `-1`.

Results

All the test-cases pass successfully.

Method 3

Method: `public Object getValueAt(int row, int column)` in `ProjectTableModel.java` line 65.

Purpose: Given two integers row and column, this function returns the value at the given column for the project at the given row.

Reason for selection: This function takes care of the selection of values present on the table. As the user interacts with the application, it is important that the selected value is the correct one. It is required for the application to work as intended.

Steps

1. Identify the parameters:
    - `row` - integer representing a project
    - `column` - integer representing a project's displayed property
    - `arPrj` - list of projects present on the table model
2. Characteristics of the parameters
    - The row integer should represent a positive number between 0 and the total number of projects;
    - The column integer should represent a positive number between 0 and the total number of categories.
3. Add constraints
    - Negative rows are not allowed;
    - Negative columns are not allowed;
    - No single row is valid when there are no projects;
    - No single column is valid when there are no projects;
    - Rows must point to a project;
    - Columns must point to a category;
    - Lists of projects mustn't be empty.
4. Generate combinations

Partition	Input (List, row, column)	Expected output
Title singleton list	One project, Row 0, Title column	Project's title

Partition	Input (List, row, column)	Expected output
Time overall singleton list	One project, Row 0, Time overall column	Project's time overall
Time created singleton list	One project, Row 0, Time created column	Project's time crated
Checked singleton list	One project, Row 0, Checked column	Project's checked
Time today singleton list	One project, Row 0, Time today column	Project's time today
Start/Pause singleton list	One project, Row 0, Start/pause column	Project's start/pause
Title two projects list	Two projects, Row 1, Title column	Second project's title
Title empty list	No projects, Row 0, Title column	Thrown exception
Out of lower bound project	One project, Row -1, Title column	Thrown exception
Out of upper bound project	One project, Row 1, Title column	Thrown exception
Out of lower bound column	One project, Row 0, -100	Default case
Out of upper bound column	One project, Row 0, 100	Default case
Out of upper bound column	One project, Row 0, Delete column	Default case

Unit Tests

We created three tests: one for valid inputs, one for out-of-bounds inputs, and another one for invalid inputs.

- Valid inputs test:
    - In this test we pass valid rows (where there is a project) and valid columns;
    - We expect to get the value of the project corresponding to the column chosen;
    - E.g. "One project, Line 0, Title column".
- Out-of-bounds inputs test:
    - In this test we play with the list of projects;
    - Trying to get a project from an empty list;
    - Passing a negative row value;
    - Choosing a row where there isn't a project (non-empty project list);
    - E.g. "One project, Line -1, Title column".
- Invalid inputs test:

- In this test we try getting invalid column numbers from a project;
- This includes negative column numbers, and column numbers that don't correspond to any column;
- In our opinion, the *delete column* should be considered since it doesn't show any value;
- The expected output of this test is a default output. In this case, the output is "wtf?". We believe the test uses a default value instead of an exception, because Swing doesn't handle that.

Results

As discussed previously, when we try getting the value of columns that aren't valid, the method returns the string "wtf?". We believe that, in order to be compatible with Swing, this method isn't allowed to explicitly throw exceptions. As such, it returns this default case/object when handling unknown/invalid `column` values.

Since projects don't contain a property to display on the *delete column*, we consider that this method should return the default case/object for this column. For this reason, the invalid inputs test fails for the input: `one project list, row 0, delete column`. From what we can gather, this combination of inputs returns whether the project is currently running or not, which doesn't appear to make sense.

The other test-cases pass successfully.

Method 4

Method: `public void setSecondsOverall(int secondsOverall)` in `Project.java` line 178.

Function's purpose: This function sets the *seconds overall* of a project as the value it receives as an argument (if valid).

Reason for selection: It is important that this function works as expected since other methods depend on it.

Steps

1. Identify the parameters:
   - `secondsOverall` - integer representing the number of seconds overall.
2. Characteristics of the parameters
   - The integer should represent a positive number between 0 and infinite.
3. Add constraints
   - Negative time is not allowed.
4. Generate combinations

Partition	Input	Expected output
Negative seconds	-100	0
Negative seconds 2	-1	0

Partition	Input	Expected output
Zero seconds	0	0
Positive seconds	1	1
Positive seconds 2	100	100

Unit Tests

We created one test for all inputs. The test receives the inputs from a stream of arguments from an auxiliary method. The inputs tested are the same as the ones present on the previous table.

- We try setting the value of secondsOverall to 0 and to the values around;
- For positive numbers (including 0), this should valid and result on the number of seconds overall being adjusted to the given number of seconds;
- For invalid inputs (negative times), the value of seconds overall should be set to 0.

Results

All the tests pass successfully.

Method 5

Method: `public void setSecondsToday(int secondsToday)` in `Project.java` line 185.

Purpose: This function sets the seconds today of a project as the value it receives as an argument (if valid).

Reason for selection: It is important that this function works as expected since other methods depend on it.

Steps

1. Identify the parameters:
    - `secondsToday` - integer representing the number of seconds today.
2. Characteristics of the parameters
    - The integer should represent a positive number between 0 and infinite.
3. Add constraints
    - Negative time is not allowed.
4. Generate combinations

Partition	Input	Expected output
Negative seconds	-100	0
Negative seconds 2	-1	0
Zero seconds	0	0

Partition	Input	Expected output
Positive seconds	1	1
Positive seconds 2	100	100

Unit Test

We created one test for all inputs. The test receives the inputs from a stream of arguments from an auxiliar method. The inputs tested are the same as the ones present on the previous table.

- We try setting the value of secondsToday to 0 and to the values around;
- For positive numbers, this should valid and result on the number of seconds today being adjusted to the given number of seconds;
- For invalid inputs (negative times), the value of seconds today should be set to 0.

Results

All the tests pass successfully.

Note about method 4 and 5

We noticed that the function `setSecondsToday(int secondsToday)` does not update the value of *seconds overall* of a project. If the author of the project is not careful with keeping both values updated accordingly, it might lead to problems. For example, a task could have 10 seconds today and 0 seconds overall, which does not make sense. For this reason, we think that this function shouldn't be publicly exposed, and that this update should have been implemented differently.