# Blog definition:

This blog is designed for people who would like to build their own JAVA web projects. We will take base project and try to implement the whole solution step by step.

# Project definition

This is the first application we will work with. It was selected for it simple functionality and could be easily expanded.

Main Use cases:

1. Login / Registration
2. After login user can see all his boards (both owned and assigned to him)
3. User can see general statistics for owned boards
4. User can assign owned boards to other users.
5. After selection board user can see Lanes and cards on them.
6. User can move Cards between lanes
7. User can create cards
8. Create/Move lanes
9. Comment on available boards and cards

This indeed is not full list of available use cases and as can be seen is expandable.

Tools:

1. Intelij Idea 15
2. Java 8

Tools list will also be expanded in parallel with project growth.

So)) Invite you to start. Hope this will be usefull for you.

# Step 0 Primitive console application

## Use Case:

1. User see all available boards;
2. User select board he wants expand;
3. User see all lanes for selected board;
4. User select lane he wants to expand;
5. User see all cards for selected lane.

## What do we intend to do:

We will define data structures for boards, lanes and cards. For now we will work with arrays: 1D for boards, 2D for lanes and 3D for cards. First index corresponds to Board number, second for Lane and third for Cards. After we have defined all data structures we will display all data and flow for our use cases.

## Implemetation

1. Data structure definition.

String[] boards = **new** String[]{**"board1"**, **"board2"**, **"board3"**};  
  
String[][] lanes = **new** String[][]{  
 {**"lane11"**, **"lane12"**, **"lane13"**},  
 {**"lane21"**, **"lane22"**, **"lane23"**},  
 {**"lane31"**, **"lane32"**, **"lane33"**},  
};  
  
String[][][] cards = **new** String[][][]{  
 {  
 {**"card111"**, **"card112"**, **"card113"**},  
 {**"card121"**, **"card122"**, **"card123"**},  
 {**"card131"**, **"card132"**, **"card133"**},  
 },  
 {  
 {**"card211"**, **"card212"**, **"card213"**},  
 {**"card221"**, **"card222"**, **"card223"**},  
 {**"card231"**, **"card232"**, **"card233"**},  
 },  
 {  
 {**"card311"**, **"card312"**, **"card313"**},  
 {**"card321"**, **"card322"**, **"card323"**},  
 {**"card331"**, **"card332"**, **"card333"**},  
 }  
};

1. Show all boards (UC1):

System.***out***.println(**"Here is list of all your boards"**);  
  
**for**(String board : boards){  
 System.***out***.println(board);  
}

1. User select board he wants expand (UC2):

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.println(**"Enter number of board you want to display lanes"**);  
**int** boardNumber = scanner.nextInt() - 1;

1. User see all lanes for selected board (UC3)

**for**(String lane : lanes[boardNumber]){  
 System.***out***.println(lane);  
}

1. User select lane he wants to expand;

System.***out***.println(**"Enter number of lane you want to display cards"**);  
**int** laneNumber = scanner.nextInt() - 1;

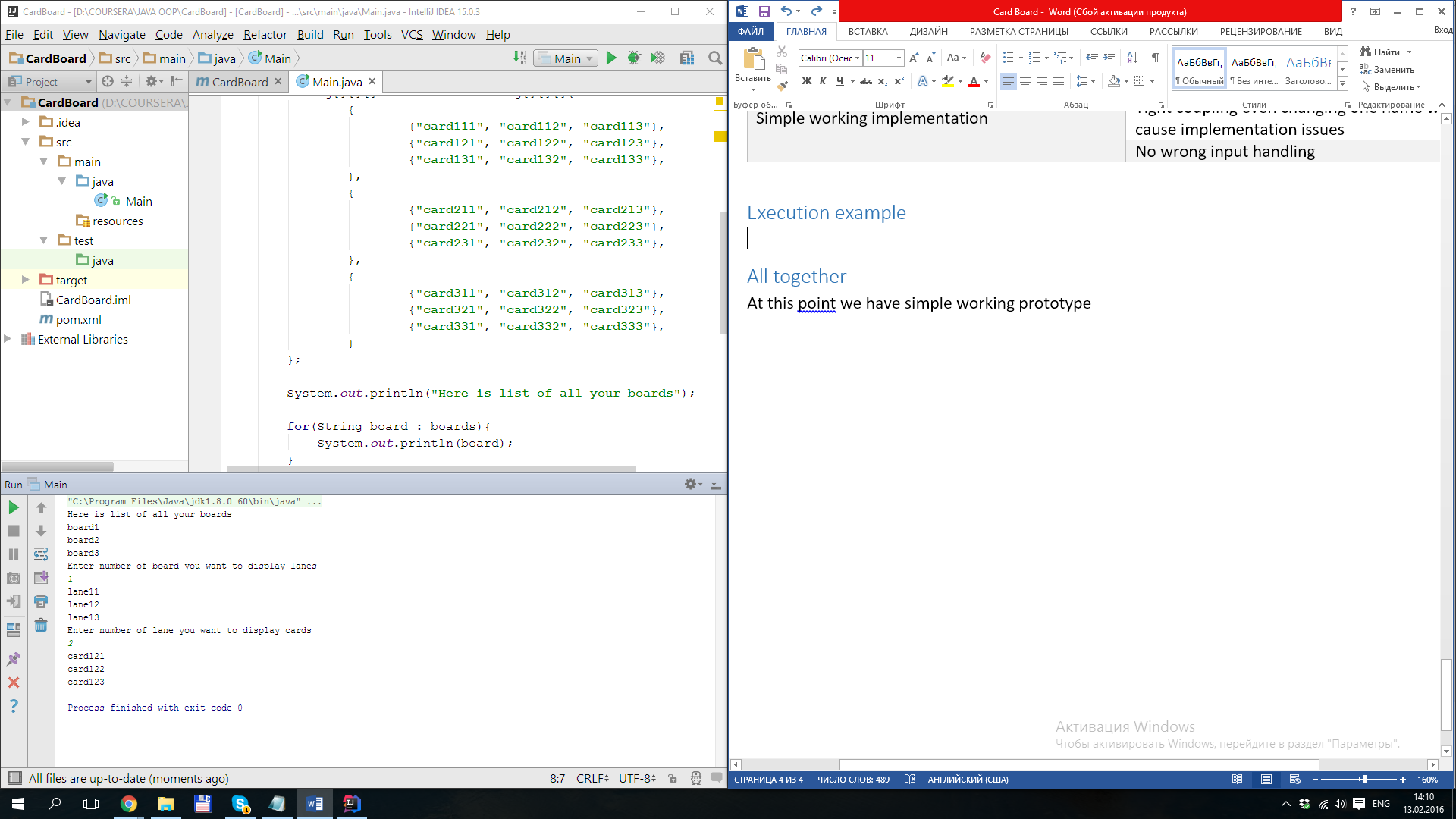
1. User see all cards for selected lane.

**for**(String card : cards[boardNumber][laneNumber]){  
 System.***out***.println(card);  
}

## Pros & Cons

|  |  |
| --- | --- |
| + | - |
| Simple working implementation | Static predefined data |
| Tight coupling even changing one name will cause implementation issues |
| No wrong input handling |

## Execution example



## Conclusion

At this point we have simple working prototype. This implementation is not the best, it just working in 1 flow.

## GitHub code

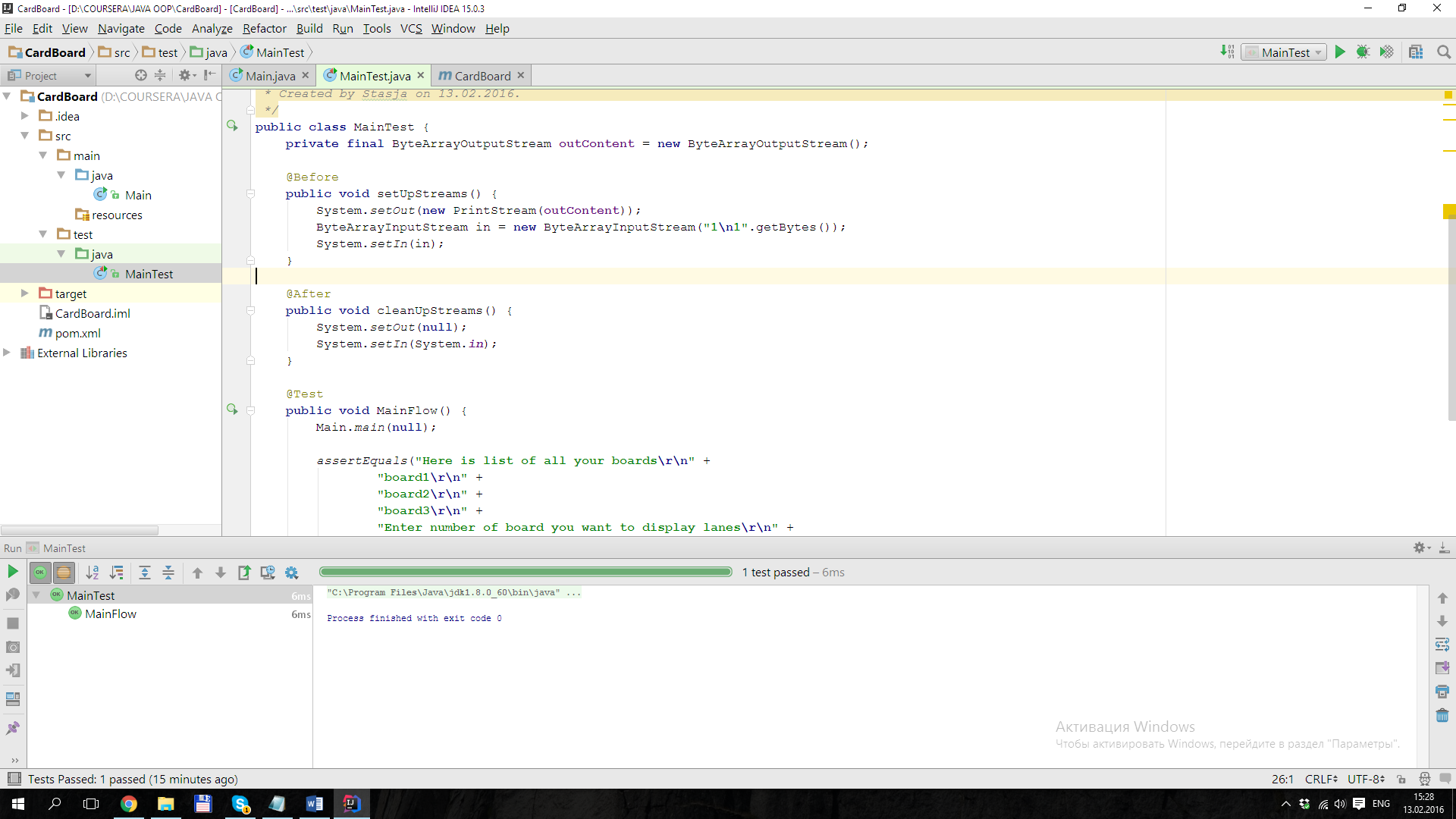
# Step 1 Unit test before refactoring

## Use Case:

1. For being able refactor code without scare broke your functionality.

## What do we intend to do:

Create one additional class in test directory called MainTest. That will contain test case for our code.

Also we will add maven framework for easier dependencies download.

## Implemetation

Main problems in implementation here is to mock System.in for entering predefined data into input console stream and getting results from System.out – ouput console stream.

In our MainTest we will create field for getting out content

**private final** ByteArrayOutputStream **outContent** = **new** ByteArrayOutputStream();

in @Before method we will define and set both ByteArrayOutputStream and ByteArrayInputStream. And in @After method we will properly close those statements.

@Before  
**public void** setUpStreams() {  
 System.*setOut*(**new** PrintStream(**outContent**));  
 ByteArrayInputStream in = **new** ByteArrayInputStream(**"1\n1"**.getBytes());  
 System.*setIn*(in);  
}  
  
@After  
**public void** cleanUpStreams() {  
 System.*setOut*(**null**);  
 System.*setIn*(System.***in***);  
}

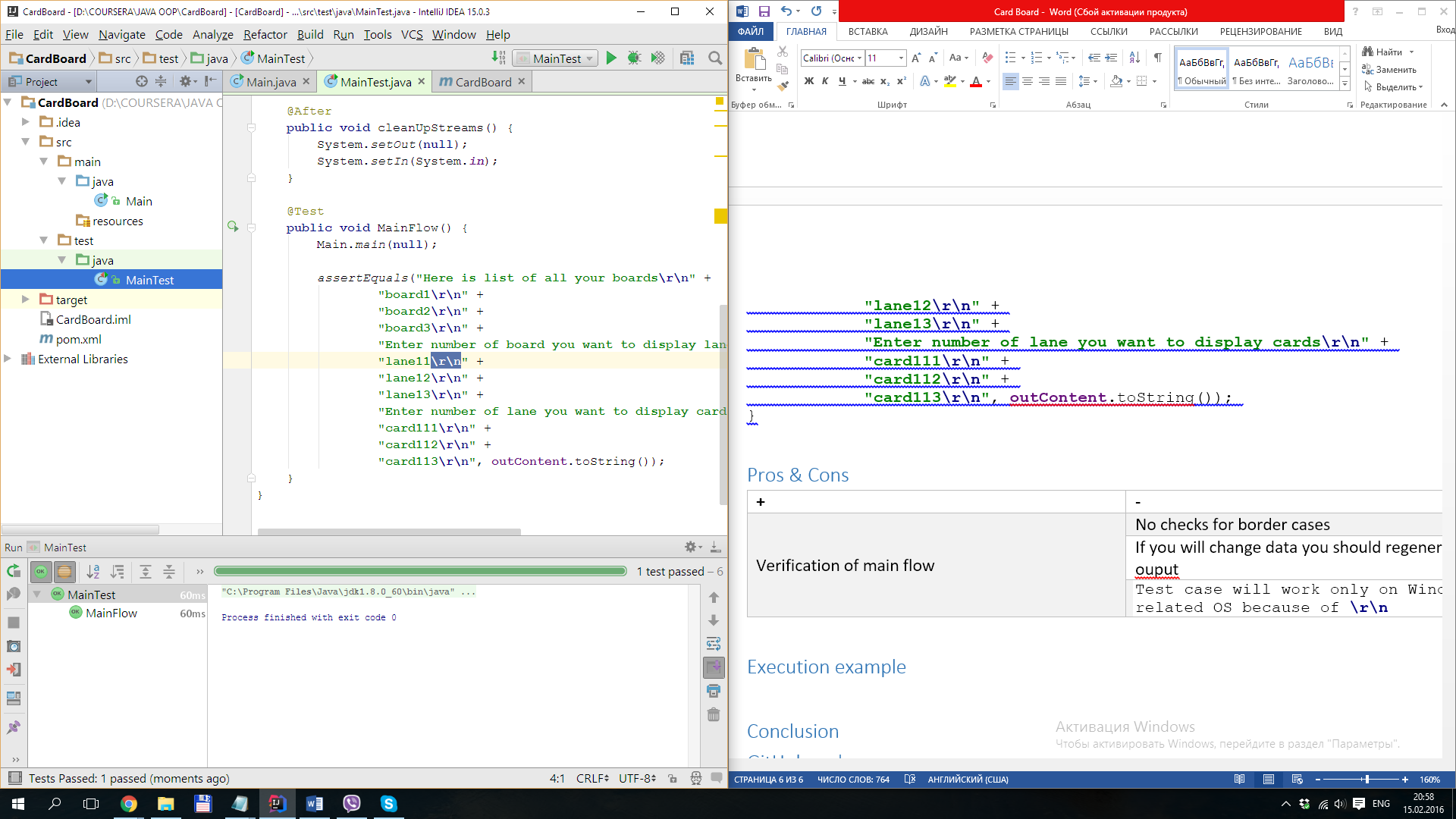
Main method will contain 2 parts: calling main (we can do it as it is static method) and check outContent if it corresponds to output for predefined user input in in.

@Test  
**public void** MainFlow() {  
 Main.*main*(**null**);  
  
 *assertEquals*(**"Here is list of all your boards\r\n"** +  
 **"board1\r\n"** +  
 **"board2\r\n"** +  
 **"board3\r\n"** +  
 **"Enter number of board you want to display lanes\r\n"** +  
 **"lane11\r\n"** +  
 **"lane12\r\n"** +  
 **"lane13\r\n"** +  
 **"Enter number of lane you want to display cards\r\n"** +  
 **"card111\r\n"** +  
 **"card112\r\n"** +  
 **"card113\r\n"**, **outContent**.toString());  
}

## Pros & Cons

|  |  |
| --- | --- |
| + | - |
| Verification of main flow | No checks for border cases |
| If you will change data you should regenerate ouput |
| Test case will work only on Windows related OS because of **\r\n** |

## Execution example



## Conclusion

For now this test case is only covers main use case that was implemented in previous step. It is not perfect and should be rewritten after code refactoring. But it gives us big benefit for being confident that initial refactoring will not harm our use case.

## GitHub code