System and Software Testing
Rendszer és szoftvertesztelés
(GKN/LM_INTA/INTM057)

Csapó Ádám - csapo.adam@sze.hu
Varjasi Norbert – norbert.varjasi@sze.hu

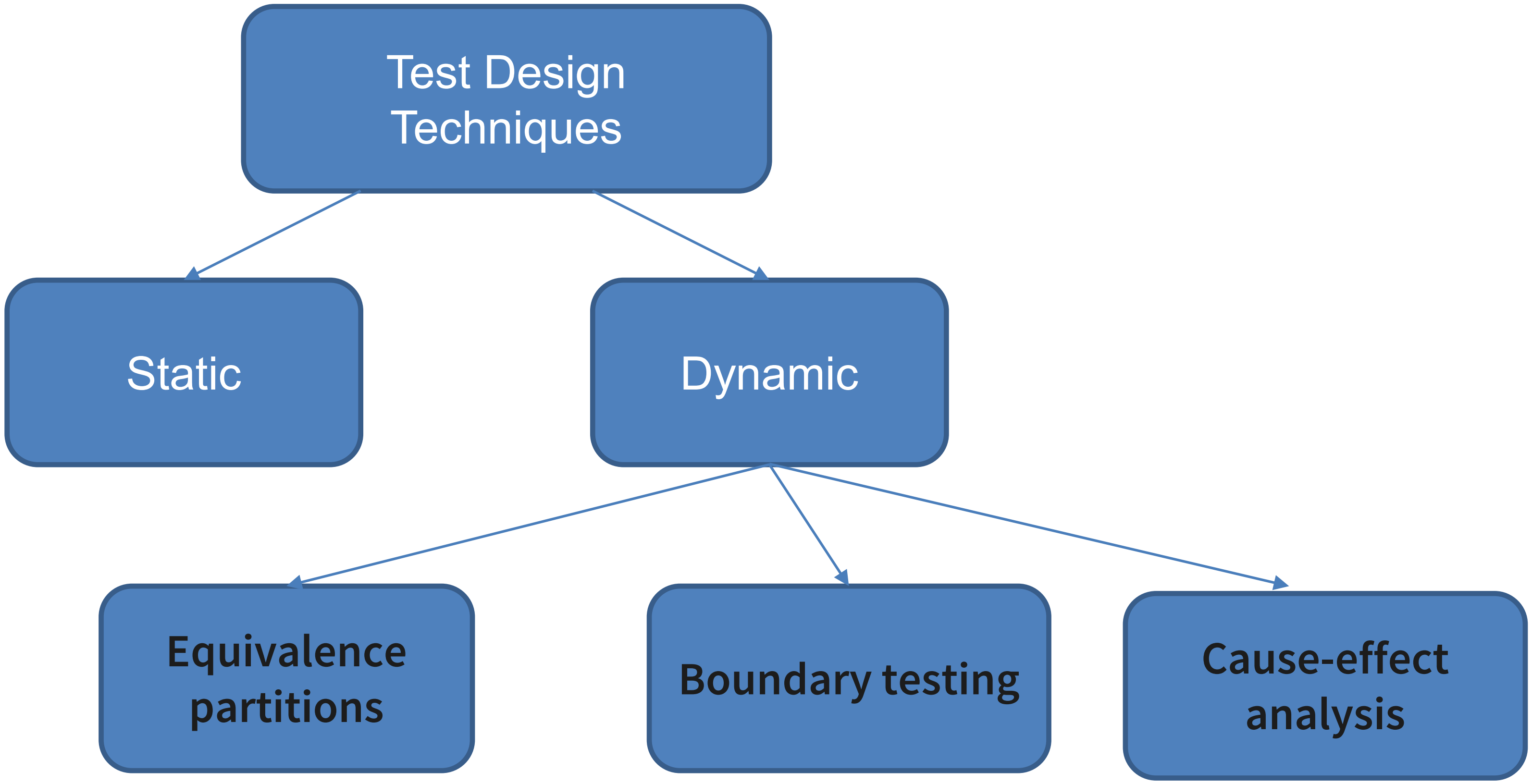
Széchenyi István University
Department of Information Technology

Topics for today

Types of dynamic testing:

- Specification-based testing
  - Equivalence partitions.
  - Boundary testing.
  - Cause-effect analysis.
- Model-based and Structure-based techniques
  - We will discuss about these in another lecture

Testing design techniques



Dynamic testing

- Dynamic Testing is a software testing method used to test the dynamic behaviour of software code.

- Requires execution of the software under test.
    - Can be both white-box (= informed by the code itself) or black-box (we pretend to not have access to the code)

- The aim is to define the weakest areas in the software runtime environment.

Equivalence partitions

- The basic idea is to divide a set of test conditions into subgroups or subsets (partitions) that can be considered the same.
  - (From the point of view of the fault model we are assuming)
- We need to test only one condition from each partition.
- … Assuming that all the conditions in one partition will be treated in the in the same way by the software.

Equivalence partitions

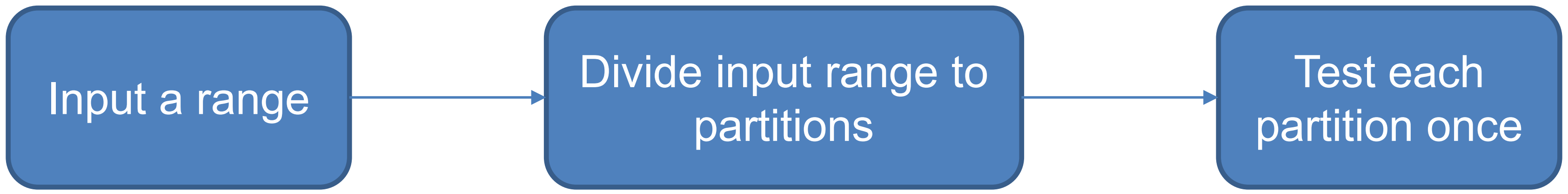
- When looking for equivalence partitions, think about ways to group:
  - similar input
  - similar outputs
  - similar operation of the software

These groups are the equivalence partitions.

Equivalence partitions

- The partitioning is done such that the behavior of the program is similar for every input data belonging to the same equivalence class.
- The main idea behind defining the equivalence classes is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class.
- Equivalence classes for a software can be designed by examining the input data and output data.

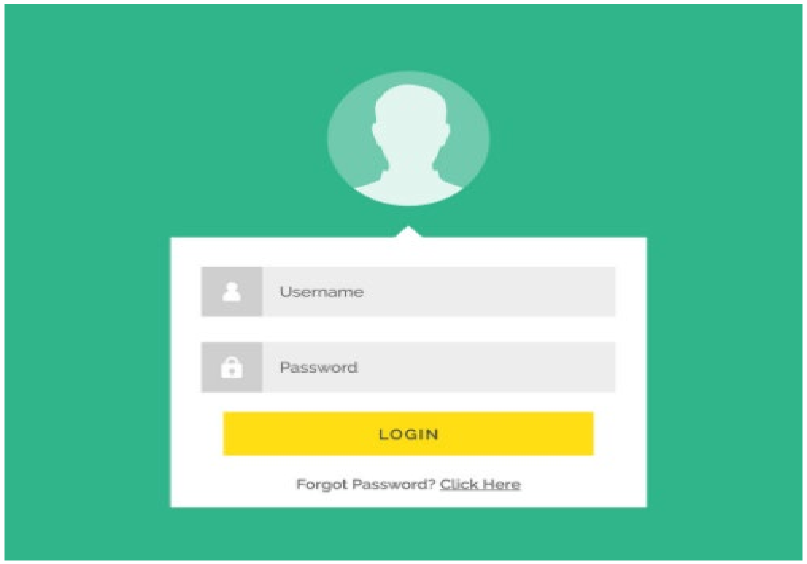
Equivalence partitions

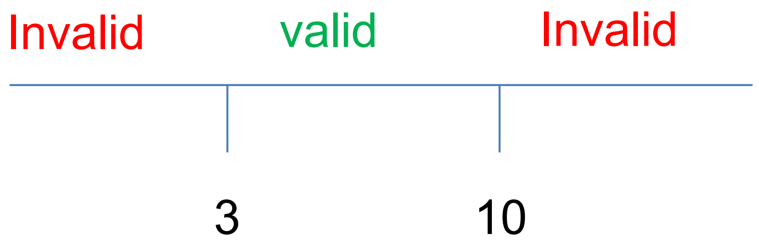


Equivalence partitions

Example:

- Testing a login module that requires the password to be between 3 and 10 characters.

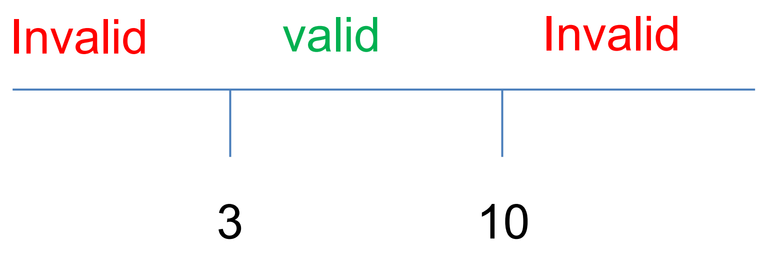




Equivalence partitions

Test Cases:

1. Verify that user enters a valid password (e.g. 4 characters).
2. Verify that user enters an invalid password (e.g. 2 characters).
3. Verify that user enters an invalid password(e.g. 12 characters).



Equivalence partitions

Example 2:
you are testing a login module that requires the password to be between 3 and 10 characters.
Which of the following sets of test data would provide coverage for all the equivalence classes with the least number of tests? A) – 333, 1234564789
B) – 22, AAA, 123456789, 1234567890
C) – -1, 0, 55555,123456789a!
D) – 1, 55555, 12345678901

Equivalence partitions

Example 2:
you are testing a login module that requires the password to be between 3 and 10 characters.
Which of the following sets of test data would provide coverage for all the equivalence classes with the least number of tests? A) – 333, 1234564789

B) – 22, AAA, 123456789, 1234567890

C) – -1, 0, 55555,123456789a!

[✓ D) – 1, 55555, 12345678901]

Equivalence partitions

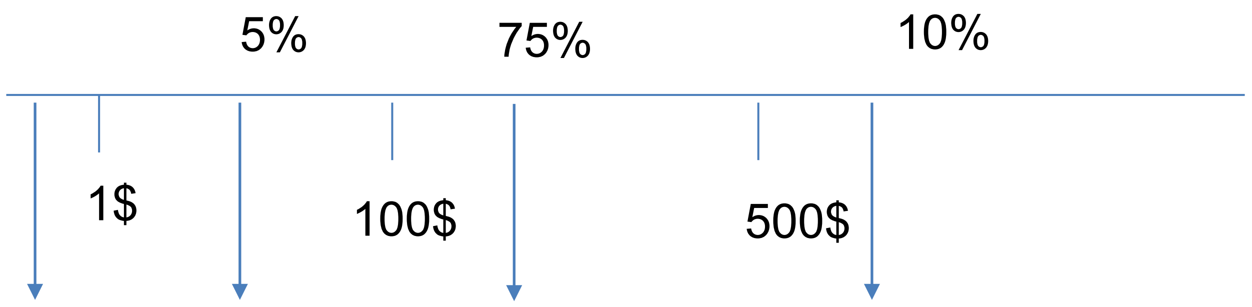
Example 3:
You are a Test Analyst for a company that provides incentives to its customers based on how much money they spend online shopping:

Category 1: \$1 - \$100 = 5%
Category 2: \$100.01 - \$500 = 7.5%
Category 3: Over \$500 = 10%

Using the equivalence partitioning test design technique, which of the following is the correct set of test cases to achieve 100% coverage with the minimum number of test cases?

Equivalence partitions

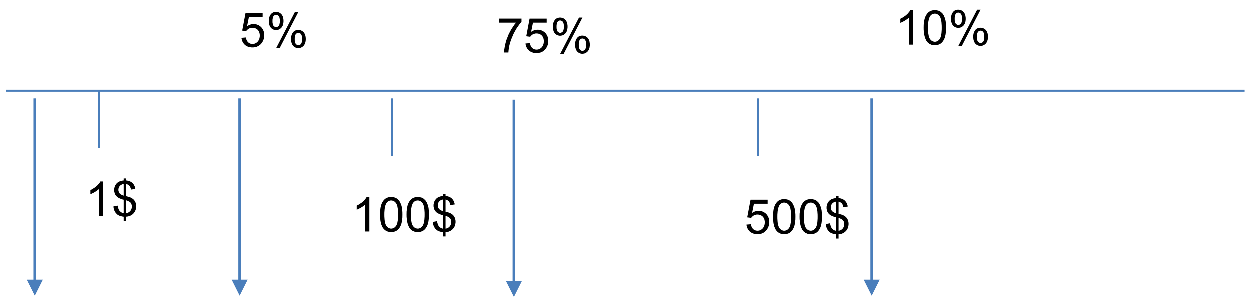
Category 1: $1 - $100 = 5%
Category 2: $100.01 - $500 = 7.5%
Category 3: Over $500 = 10%



a. $50, $150
b. $0, $50, $150, $500
c. $0, $1, $100, $500, $550
d. $0, $50, $150, $550, max amount + $50

Equivalence partitions

Category 1: $1 - $100 = 5%
Category 2: $100.01 - $500 = 7.5%
Category 3: Over $500 = 10%



a. $50, $150
b. $0, $50, $150, $500
c. $0, $1, $100, $500, $550
d. $0, $50, $150, $550, max amount + $50

Equivalence partitions

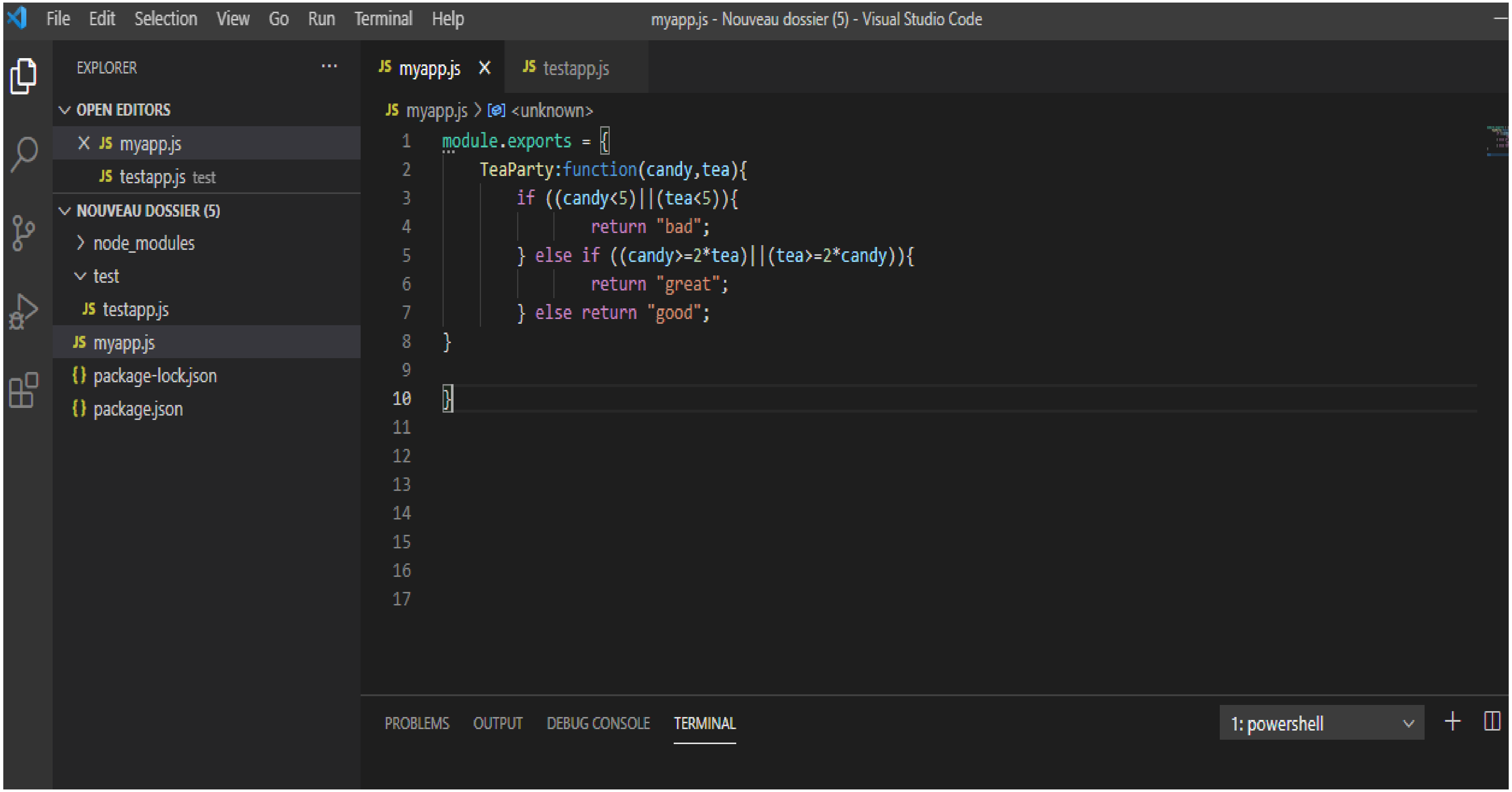
Tea Party Example: We are having a party with sound amount of tea and candy. A party is good if both tea and candy are available in at least 5 units. However, if either amount of tea or candy is at least double the amount of the other one, the party is great.
However, in all cases, if either tea or candy is available in a quantity smaller than 5 units, the party is always bad.

We are going to use Equivalence partitioning to define our test cases.

Equivalence partitions

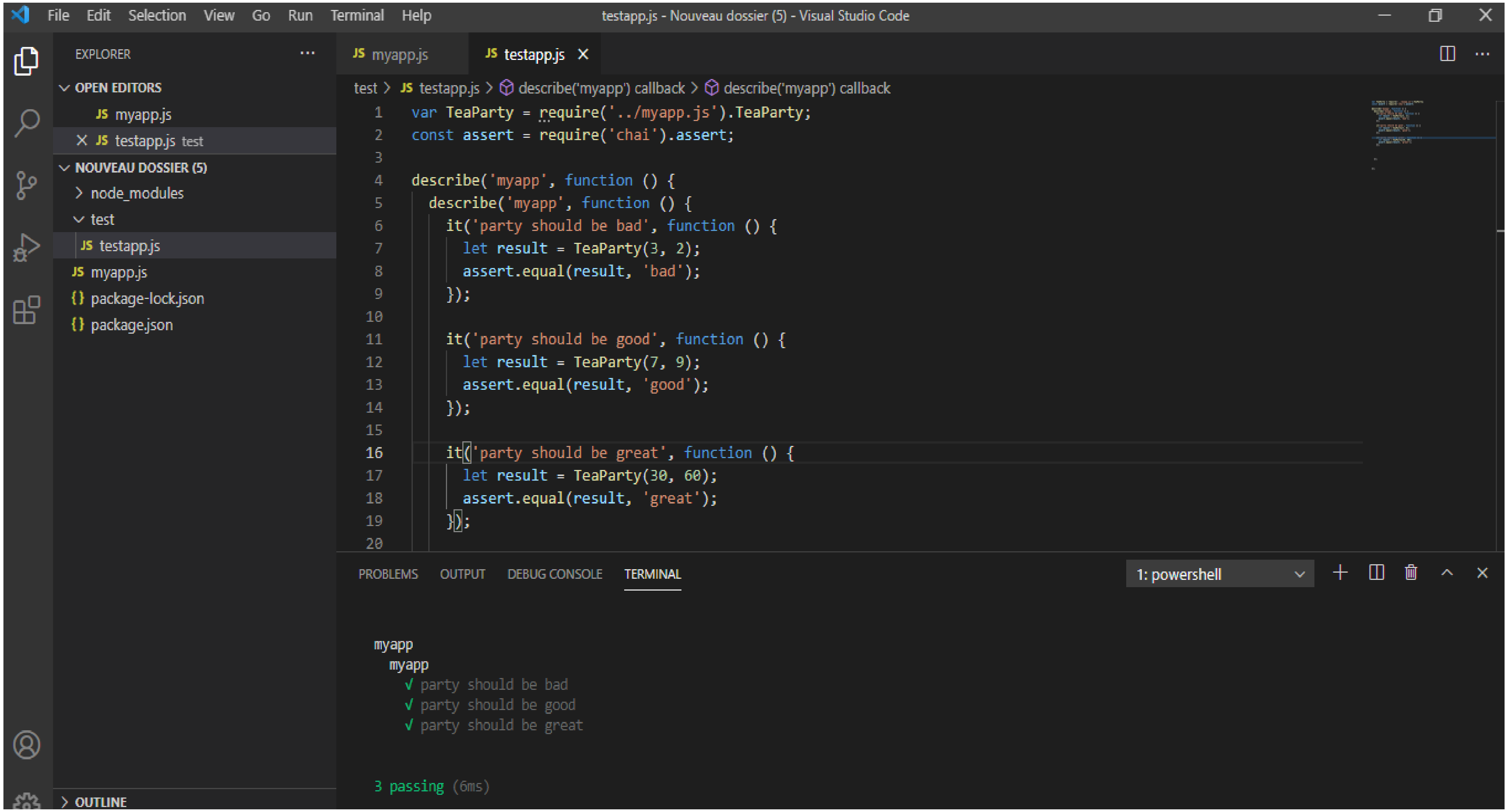
- Steps for this exercise:
  - Let’s use Javascript for this example.
  - Standalone javascript programs can be run using e.g. the Node.js environment
  - Use npm to install the package requirements.
  - Install mocha and chai to implement the testing. (other packages are also available such as jest)
  - Figure out the partitions you want to use.
  - Run different cases to test the app.

Equivalence partitions



```
module.exports = {
    TeaParty:function(candy,tea){
        if ((candy<5)||(tea<5)){
                return "bad";
        } else if ((candy>=2*tea)||(tea>=2*candy)){
                return "great";
        } else return "good";
}

}
```

Equivalence partitions



```javascript
var TeaParty = require('../myapp.js').TeaParty;
const assert = require('chai').assert;

describe('myapp', function () {
  describe('myapp', function () {
    it('party should be bad', function () {
      let result = TeaParty(3, 2);
      assert.equal(result, 'bad');
    });

    it('party should be good', function () {
      let result = TeaParty(7, 9);
      assert.equal(result, 'good');
    });

    it('party should be great', function () {
      let result = TeaParty(30, 60);
      assert.equal(result, 'great');
    });

```

```
  myapp
    myapp
      √ party should be bad
      √ party should be good
      √ party should be great


  3 passing (6ms)
```

Equivalence partitions

Advantages:

- ✓ With the help of equivalence class testing, the number of test cases gets greatly reduced while maintaining the same test coverage.
- ✓ This testing technique helps in delivering a quality product within a minimal time period.
- ✓ It is perfectly suitable for software projects with time and resource constraints.

Equivalence partitions

Limitations:

- ✓ In the case of complex applications, it is very difficult to identify all set of equivalence classes and requires a great deal of expertise from the tester’s side.
- ✓ Incorrectly identified equivalence classes can lead to lesser test coverage and the possibility of defect leakage.

Boundary values analysis

Boundary values analysis

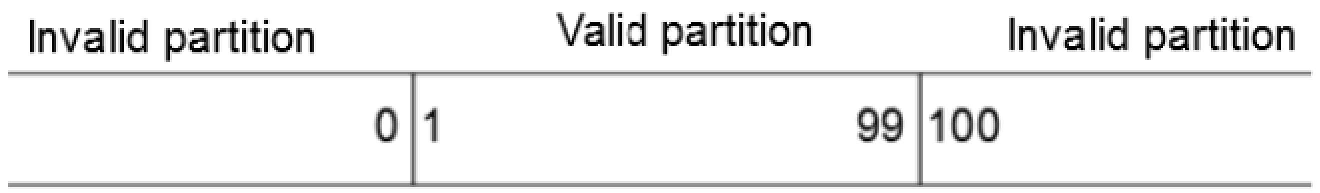
- This method checks the “border” of the equivalence classes. On every boundary, the exact boundary value & both nearest adjacent values (inside & outside the equivalence class) are tested.
- In general, three test cases result from every boundary
- If the upper boundary of one equivalence class equals the lower boundary of the adjacent class, then the respective test cases coincide.

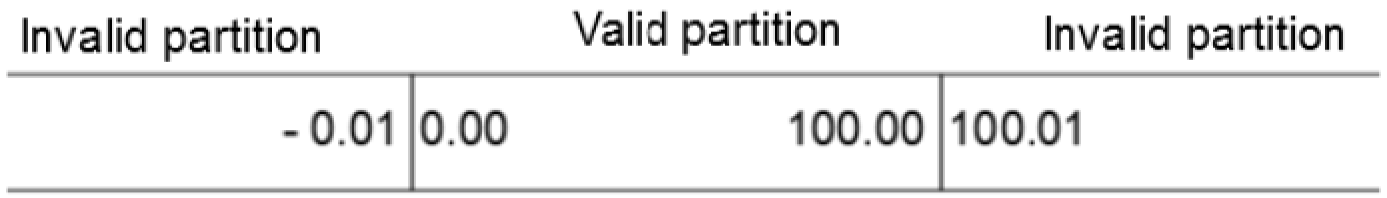
Boundary values analysis

- When there is no real boundary value, it is sufficient to test the boundary with two values: one which is just inside & another just outside the class.
- Think about the following characteristics: First/Last Min/Max and Empty/Full.

- Test: First–1/Last+1, Less than Empty/More than Full and Min–1/Max+1

Boundary values analysis





Boundary values analysis

- Example 1: You are testing a machine that scores exam papers and assigns grades. Based on the score achieved the grades are as follows:
  1-49 = F, 50-59 = D-, 60-69 = D, 70-79 = C, 80-89 = B, 90-100=A

If you apply boundary value analysis, how many test cases will you need to achieve minimum test coverage?

A) 8
B) 10
C) 12
D) 14

Boundary values analysis

- Example 1: You are testing a machine that scores exam papers and assigns grades. Based on the score achieved the grades are as follows:
  1-49 = F, 50-59 = D-, 60-69 = D, 70-79 = C, 80-89 = B, 90-100=A

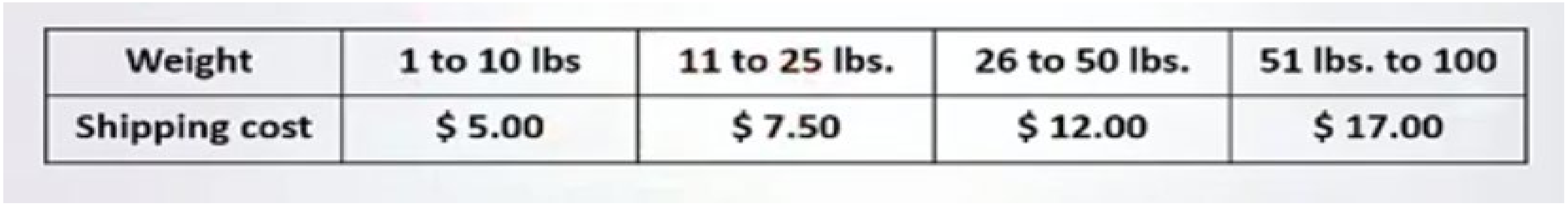
If you apply boundary value analysis, how many test cases will you need to achieve minimum test coverage?

A) 8
B) 10
C) 12
[✓ D) 14]

You need to test all values listed above, + 0 and 101.

Boundary values analysis

- Example 2: You are testing a scale system that determines shipping rates for a regional web-based auto parts distributor. Due to regulations, shipments cannot exceed 100 lbs. You want to include boundary value analysis as part of your black-box test design.



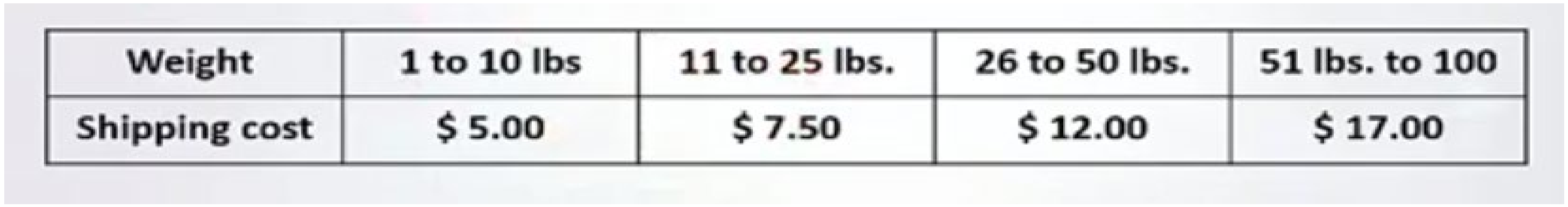
Weight	1 to 10 lbs	11 to 25 lbs.	26 to 50 lbs.	51 lbs. to 100
Shipping cost	$ 5.00	$ 7.50	$ 12.00	$ 17.00

How many tests will you need to execute to achieve 100% boundary value analysis?

A) 4
B) 8
C) 10
D) 12

Boundary values analysis

- Example 2: You are testing a scale system that determines shipping rates for a regional web-based auto parts distributor. Due to regulations, shipments cannot exceed 100 lbs. You want to include boundary value analysis as part of your black-box test design.



Weight	1 to 10 lbs	11 to 25 lbs.	26 to 50 lbs.	51 lbs. to 100
Shipping cost	$ 5.00	$ 7.50	$ 12.00	$ 17.00

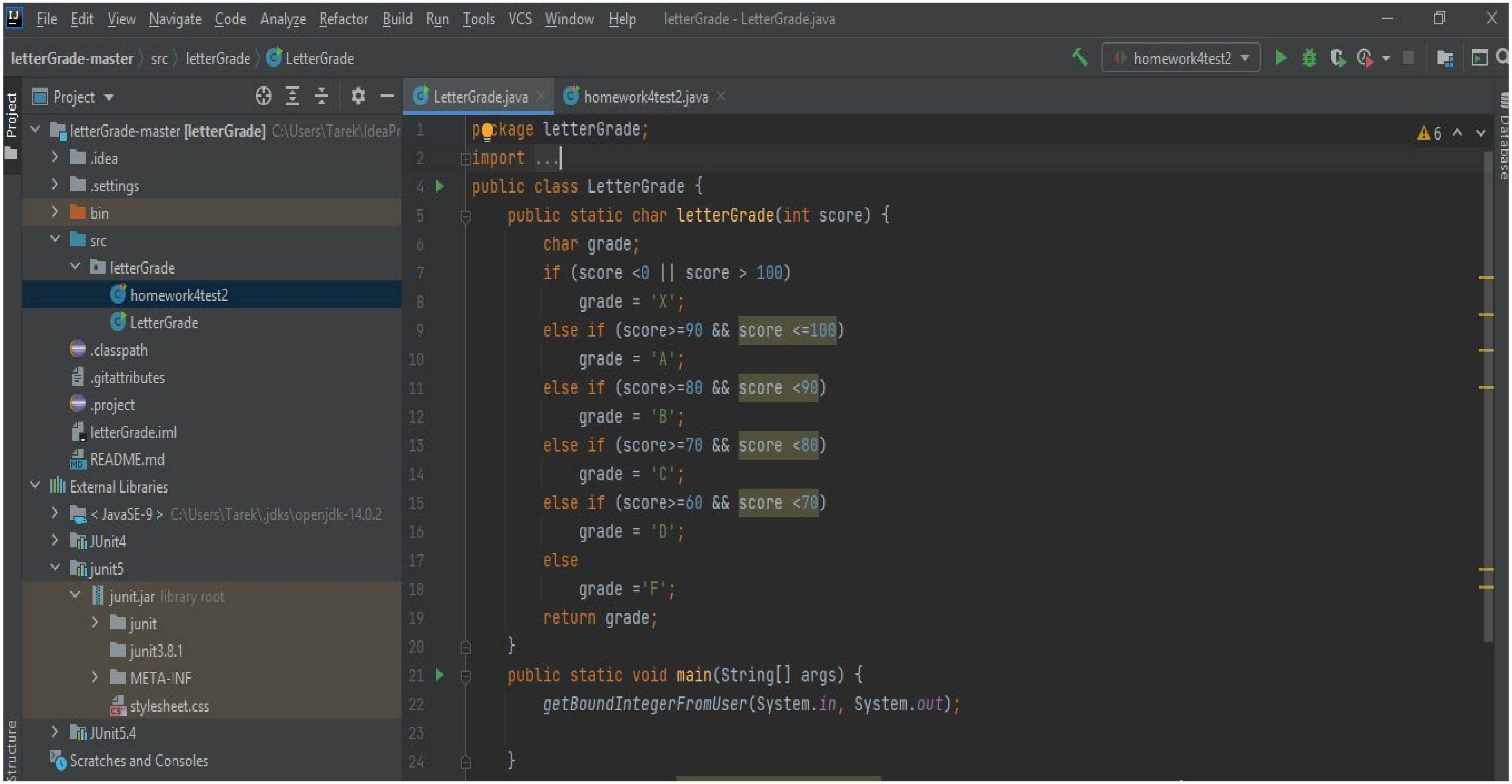
How many tests will you need to execute to achieve 100% boundary value analysis?

A) 4
B) 8
C) [✓ 10]
D) 12

Boundary values analysis

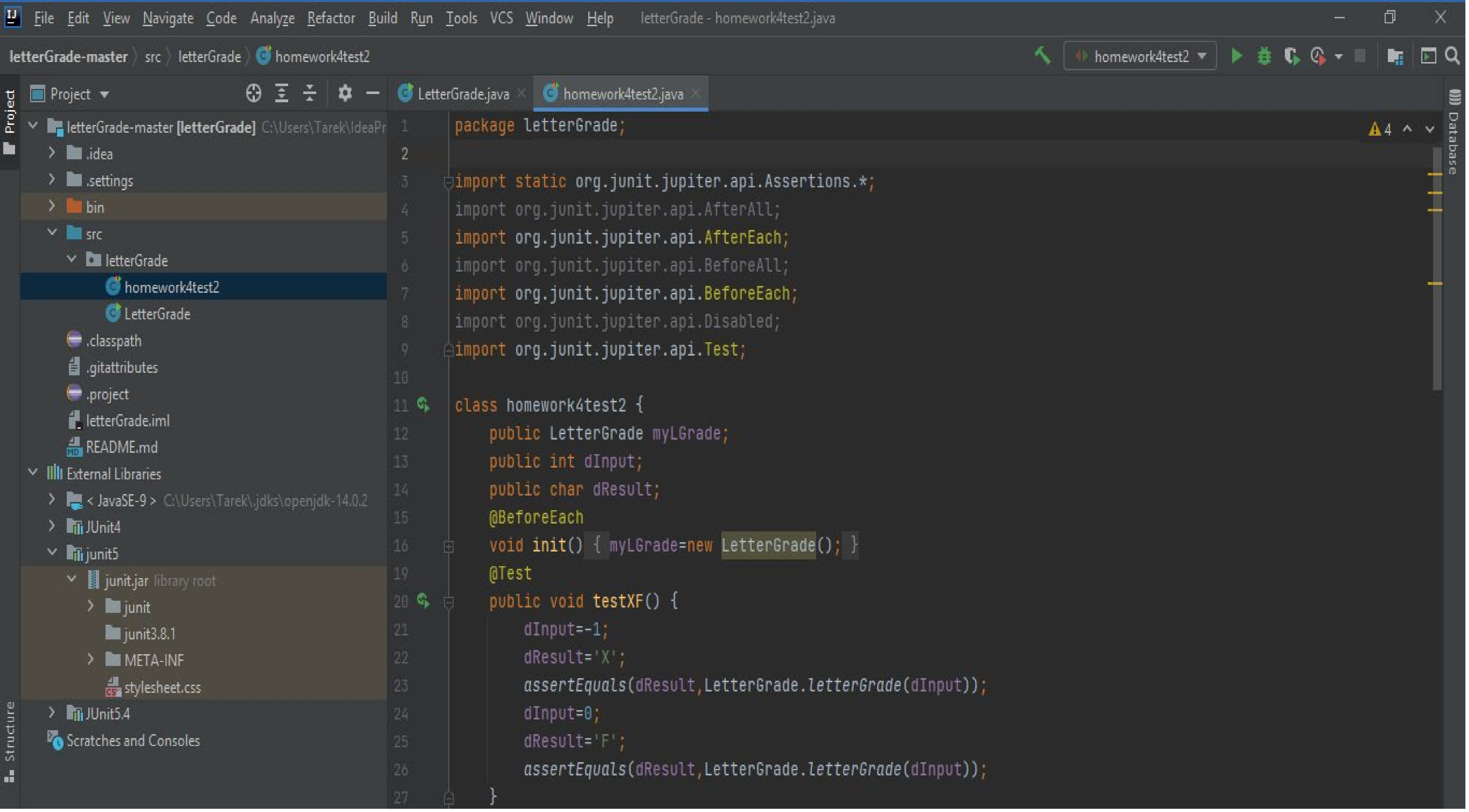
- Lets implement the example of Grade Score:
- Let’s use JAVA and JUNIT.

Boundary values analysis



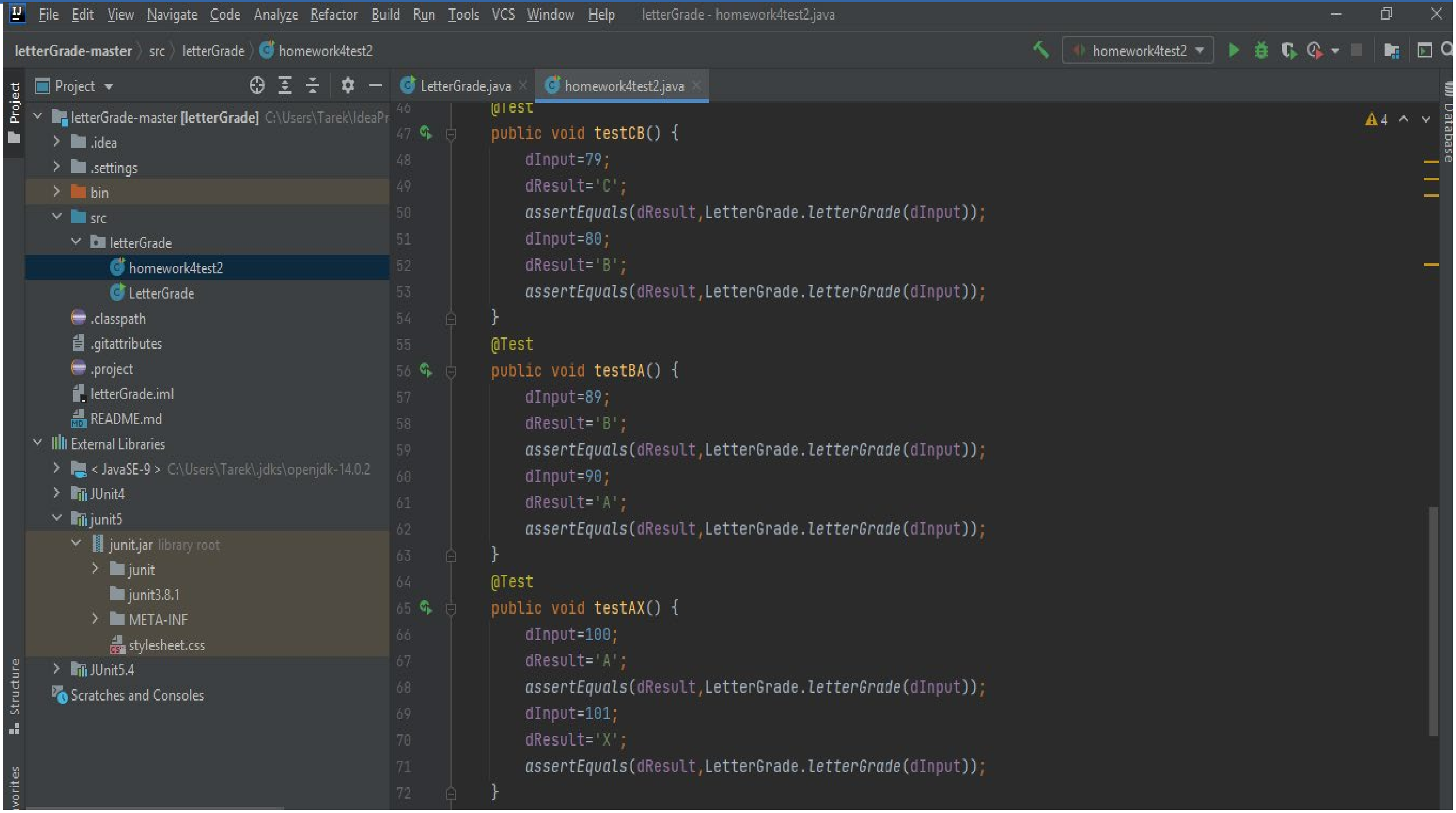
```java
package letterGrade;
import ...
public class LetterGrade {
    public static char letterGrade(int score) {
        char grade;
        if (score <0 || score > 100)
            grade = 'X';
        else if (score>=90 && score <=100)
            grade = 'A';
        else if (score>=80 && score <90)
            grade = 'B';
        else if (score>=70 && score <80)
            grade = 'C';
        else if (score>=60 && score <70)
            grade = 'D';
        else
            grade ='F';
        return grade;
    }
    public static void main(String[] args) {
        getBoundIntegerFromUser(System.in, System.out);

    }
```

Boundary values analysis



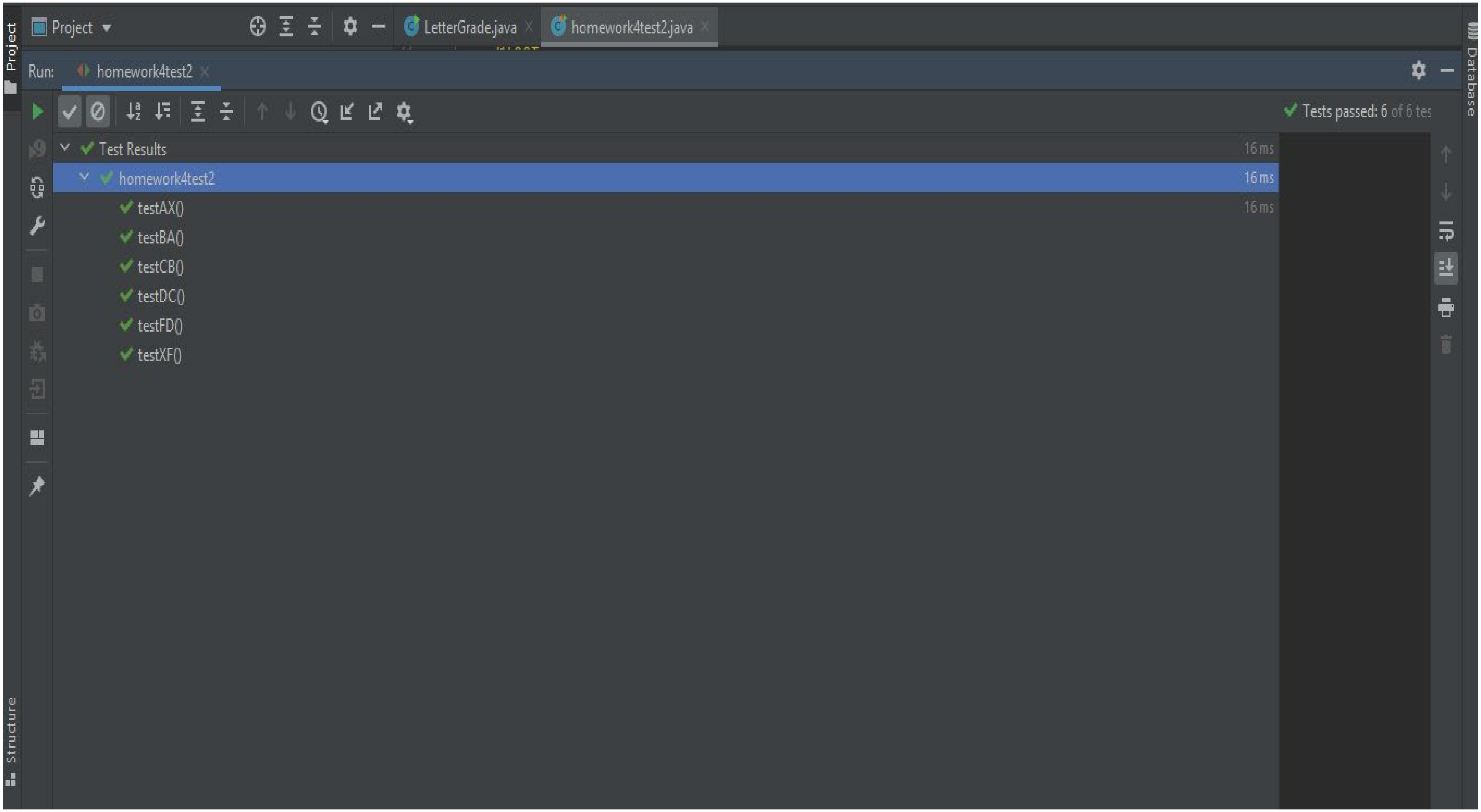
```java
package letterGrade;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;

class homework4test2 {
    public LetterGrade myLGrade;
    public int dInput;
    public char dResult;
    @BeforeEach
    void init() { myLGrade=new LetterGrade(); }
    @Test
    public void testXF() {
        dInput=-1;
        dResult='X';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
        dInput=0;
        dResult='F';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
    }
```

Boundary values analysis



```
    @Test
    public void testCB() {
        dInput=79;
        dResult='C';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
        dInput=80;
        dResult='B';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
    }
    @Test
    public void testBA() {
        dInput=89;
        dResult='B';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
        dInput=90;
        dResult='A';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
    }
    @Test
    public void testAX() {
        dInput=100;
        dResult='A';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
        dInput=101;
        dResult='X';
        assertEquals(dResult,LetterGrade.letterGrade(dInput));
    }
```

Boundary values analysis


Project | LetterGrade.java | homework4test2.java

Run: homework4test2

Tests passed: 6 of 6 tes

```
Test Results          16 ms
  homework4test2      16 ms
    testAX()          16 ms
    testBA()
    testCB()
    testDC()
    testFD()
    testXF()
```

Boundary values analysis

Advantages:

- ✓ It is easier and faster to find defects as the density of defects at boundaries is usually higher.

- ✓ The overall test execution time is reduced as the number of test data points are greatly reduced.

Boundary values analysis

Limitations:

- ✓ Applications with open boundaries are not suitable for use of this method.
- ✓ The success of the testing using boundary value analysis depends on the equivalence classes identified

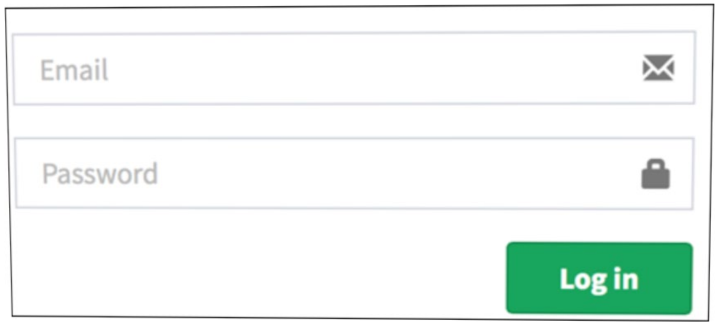
Cause-effect analysis

Cause-effect analysis

- Cause-effect analysis usually involves creating a table that describes the relationship between test cases and specific failure types that are covered by those test cases
  - The columns are the test cases
  - The rows represent different kinds of faulty scenarios
- The goal is to ensure that all scenarios of interest are covered by the test cases
  - If they aren't, this is a way to guide the test engineers in developing new test cases

Cause-effect analysis – Example 1

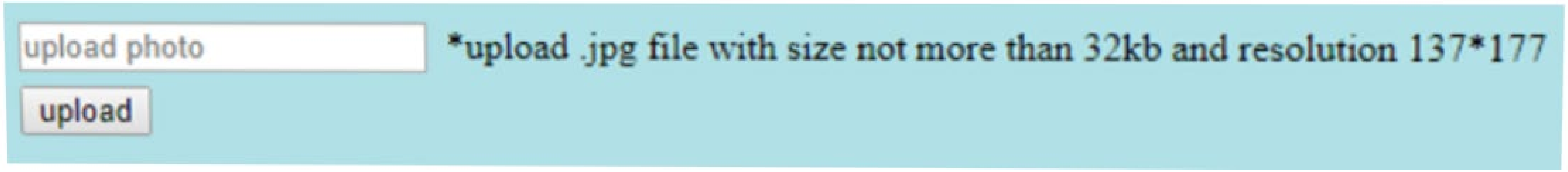
- Consider a classic login screen, for example
- In the most basic case, we will need for test cases:



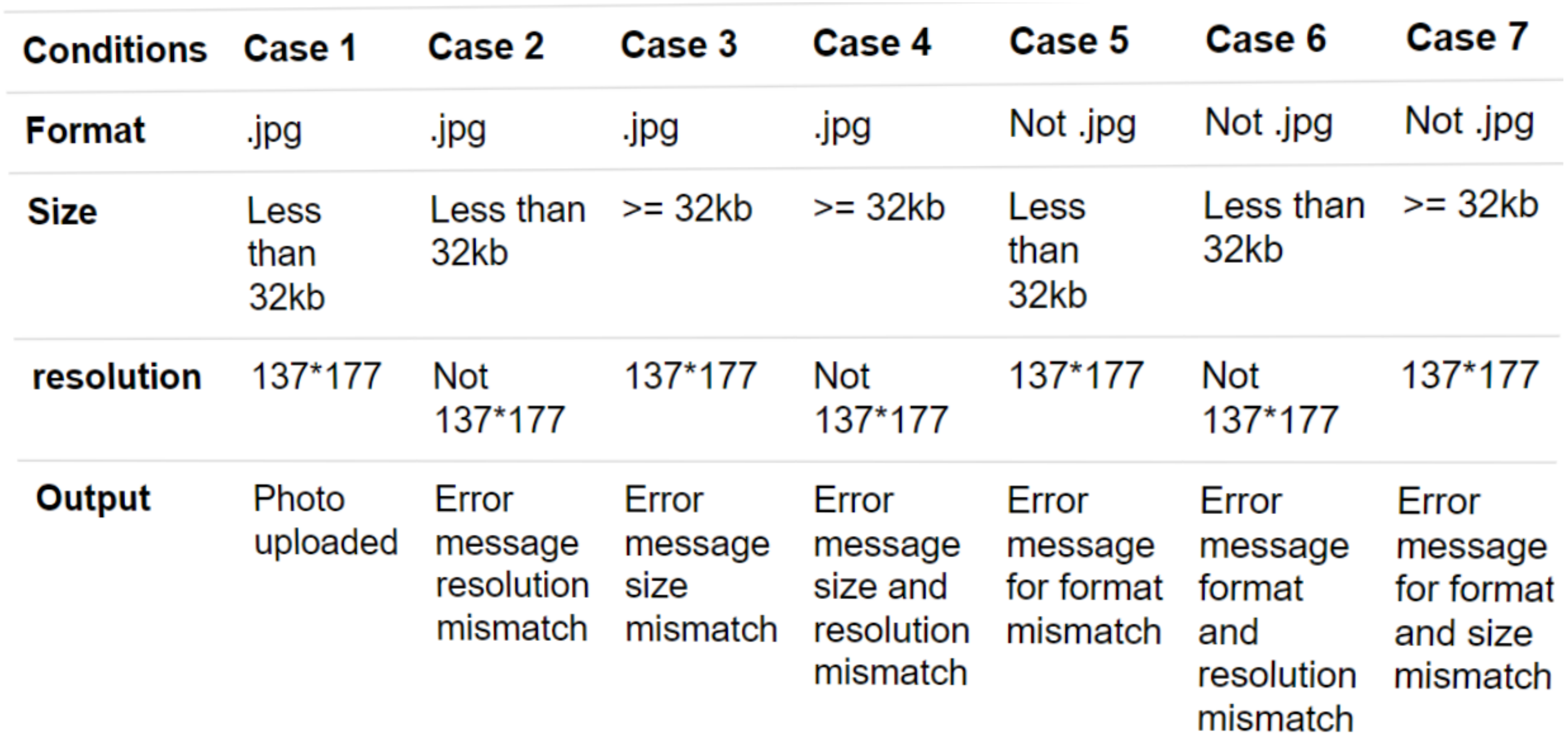
Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

- T – Correct username/password
- F – Wrong username/password
- E – Error message is displayed
- H – Home screen is displayed

Cause-effect analysis – Example 2



*upload .jpg file with size not more than 32kb and resolution 137*177



Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb	Less than 32kb	Less than 32kb	>= 32kb
resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch