UNIVERSITY OF ANTWERP
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Software Testing Lab

Assignment 2 - Category Partitioning and Boundary Values

Submission Deadline: **March 10th, 20:00**

1 ASSIGNMENT

Important Note: *Create an archive for JPacman system after performing all the exercises that require modifications to the files. Submit it along with your report. Refer to the "Assignments General" document for introductory information.*

CATEGORY-PARTITION AND BOUNDARY VALUES

In this assignment, you will be building and testing an implementation `Board.withinBorders(int x, int y)` method, which simply checks that `x` and `y` integer values fall within the borders (width × height) of the board, i.e. $0 <= x <$ width, and $0 <= y <$ height. Our approach follows the approach from Forgács (Practical test design : selection of traditional and automated test design techniques), chapter 5: p57-89.

- **Exercise 1.** List at least three possible errors that an implementor of this method could make. **(Required, 9 points)**

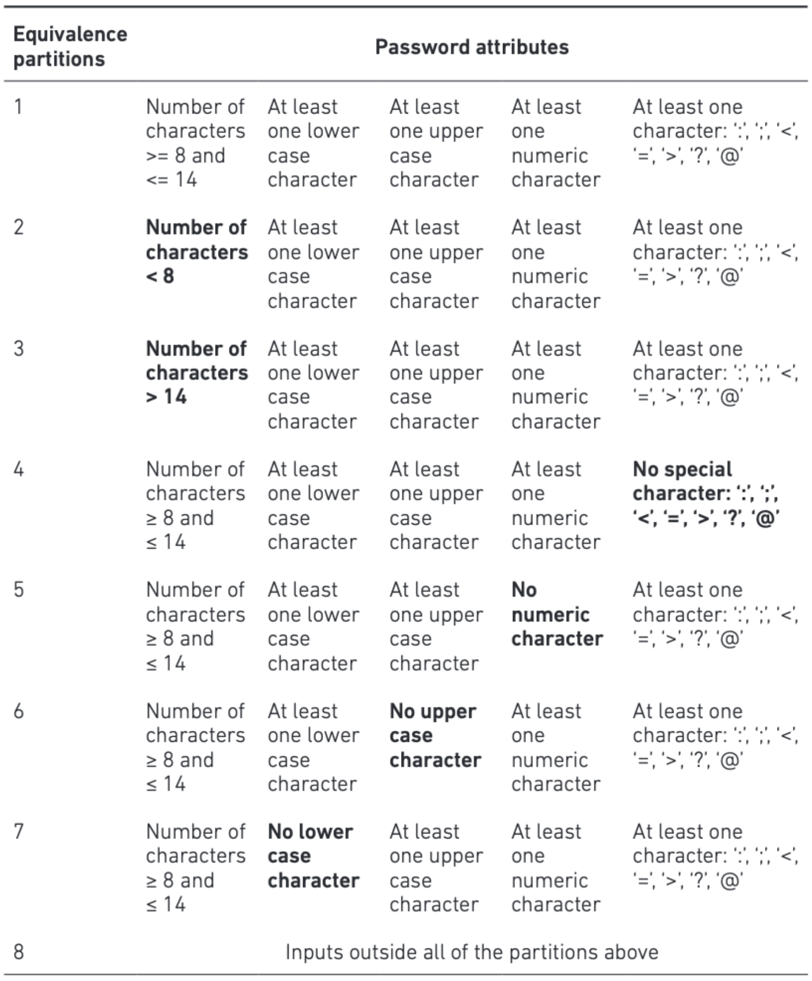


Table 5.1 Equivalence partitioning for the authorisation example

Equivalence partitions	Password attributes				
1	Number of characters >= 8 and <= 14	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ':', ';', '<', '=', '>', '?', '@'
2	**Number of characters < 8**	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ':', ';', '<', '=', '>', '?', '@'
3	**Number of characters > 14**	At least one lower case character	At least one upper case character	At least one numeric character	At least one character: ':', ';', '<', '=', '>', '?', '@'
4	Number of characters ≥ 8 and ≤ 14	At least one lower case character	At least one upper case character	At least one numeric character	**No special character: ':', ';', '<', '=', '>', '?', '@'**
5	Number of characters ≥ 8 and ≤ 14	At least one lower case character	At least one upper case character	**No numeric character**	At least one character: ':', ';', '<', '=', '>', '?', '@'
6	Number of characters ≥ 8 and ≤ 14	At least one lower case character	**No upper case character**	At least one numeric character	At least one character: ':', ';', '<', '=', '>', '?', '@'
7	Number of characters ≥ 8 and ≤ 14	**No lower case character**	At least one upper case character	At least one numeric character	At least one character: ':', ';', '<', '=', '>', '?', '@'
8	Inputs outside all of the partitions above				

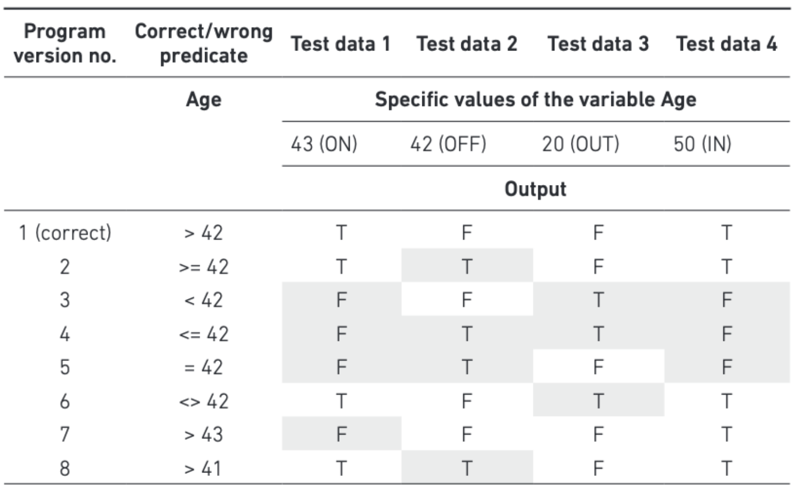
- **Exercise 2.** Identify the equivalence partitions for the withinBorders method. You should provide a table similar to Table 5.1 (see Forgács). You may assume the single fault model. **(Required, 10 points)**

- **Exercise 3.** Similarly, identify the equivalence partitions for the withinBorders method, but for the multiple fault model. **(Required, 10 points)**

- **Exercise 4.** Next, develop a test design against predicate faults (see Forgács, Table 5.2). You should have a table for predicate *x* and a table for predicate *y*. **(Required, 10 points)**

- **Exercise 5.** Implement your test case specs into `BoardTest` class and run the test suite. **(Required, 10 points)**

Table 5.2 Test design against predicate faults. BVA for predicate Age > 42



Program version no.	Correct/wrong predicate	Test data 1	Test data 2	Test data 3	Test data 4
	Age	Specific values of the variable Age			
		43 (ON)	42 (OFF)	20 (OUT)	50 (IN)
		Output			
1 (correct)	> 42	T	F	F	T
2	>= 42	T	T	F	T
3	< 42	F	F	T	F
4	<= 42	F	T	T	F
5	= 42	F	T	F	F
6	<> 42	T	F	T	T
7	> 43	F	F	F	T
8	> 41	T	T	F	T

- **Exercise 6.** Implement `withinBoarders` method. Then re-run the test suite. Inspect code coverage results, and explain your findings. **(Required, 10 points)**
- **Exercise 7.** Which code coverage metric did you use (Branch coverage, Statement coverage or others)? Why? **(Only when late, -5 to 0 points)**
- **Exercise 8.** Would your test suite reveal all the faults you proposed in Exercise 1? If not, explain why the equivalence partitioning approach missed it, and add appropriate test cases separately to your JUnit implementation. **(Required, 8 points)**
- **Exercise 9.** Are your test cases dependent on your method implementation? Did you need to change your test cases while developing the method? Does this have a positive or a negative impact on the design of the test cases? **(Only when late, -5 to 0 points)**
- **Exercise 10.** Repeat the exercises for the method `Game.addGuestFromCode(char code, int x, int y)`. How many test cases do you end up with? **(Required, 25 points)**
- **Exercise 11.** Imagine a more complex Board and an additional parameter that describes the shape of the Guest (like occupying multiple cells). How does an equivalence partitioning approach scale? **(Required, 8 points)**
- **Exercise 12.** Create an equivalence partition table where a guest can vary in size. The guest is a 2D being that can occupy 1 or more cells. You may limit the shape of the guest to rectangles. Does your equivalence partitioning table scale like you predicted? **(Only when late, -10 to 0 points)**