UNIVERSITY OF ANTWERP
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

Software Testing Lab

Assignment 2

Submission Deadline: **March 6th, 20:00**

1 INTRODUCTION

1.1 OBJECTIVE

The objective of the lab work of the Software Testing course is to help you learn how you can apply the various testing techniques and test design patterns as discussed during the lectures in practice. You will apply these techniques to a simple Pacman system written in Java. The amount of coding that needs to be done is relatively small: The focus is on testing.

Many testing approaches are based on partitioning the data set into categories, focusing on domain boundaries, and selecting appropriate combinations of inputs, as explained in Chapter 10 of Binder. In this exercise we will build up some experience with these techniques, and learn how to test methods using them.

1.2 APPROACH

The work in the labs is mostly self-study. The handouts contain a chain of tasks, some more practical, others in the form of more philosophical questions reflecting on previous tasks. Programming exercises are in Java. For your Java development, you can use your favorite IDE. All the material needed for the completion of the assignment is available at `http://`

ansymore.uantwerpen.be/courses/software-testing . The JPacman distribution includes source files, test files, and documentation (in the doc directory).

- `pacman-requirements.txt`: A text file describing the JPacman use cases
- `pacman-design.txt`: A text file describing the key JPacman design decisions

1.3 Front heavy course

The course and assignments are front heavy. This means that all assignments will be given in approximately the first half of the semester. This in contrast to most other courses which will require more work at the end of the semester. You are encouraged to complete and submit the assignments on time, to free up ample time in the second half of the semester.

1.4 Grading

Each assignment is graded from 0 to 100. To be eligible for the final exam, you need to score at least 50/100 for each assignment. Most assignments build upon each other. It is possible that some exercises can only be completed if you completed previous exercises. You are encouraged to submit the assignments on time. If you submit late, you should answer the "Only when late" exercises. With these you cannot gain points, you lose point when you do not answer them or answer them incorrectly.

1.5 Report

Please note that the report is the most important part of your answer, so take some time to write an adequate report. Describe your actions, results, and explanations for each exercise in your report with full sentences. If appropriate, provide screenshots to show you fulfilled the exercise. Accompany the report with all of the requested material. All submitted artefacts should be stand-alone, runnable with single commands, and should not require additional packages. Upload all files in a zipped archive with your name, underscore and the assignment number, i.e. <Surname_LastName_2.zip>.

1.6 Questions

There will be a lab session every Monday from 10:45 to 12:45 to answer any questions. Prepare your questions beforehand. Broken packages, links, missing images, etc that prevent the assignment to be completed can be reported to Onur.Kilincceker@uantwerpen.be **and** Mutlu.Beyazit@uantwerpen.be.

2 Assignment

Important Note: *Create an archive from the JPacman system after you perform each exercise that requires modifications to the files. Name the archive according to the exercise number, and submit them along with your report.*

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)**
- **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, generate 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)**
- **Exercise 6.** Actually 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)**

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				

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