PV260 Software Quality

NAME, SURNAME and UČO: _____

Q1. You are requested to write a simple application that given a dataset of all students in a room will compute the number of students that have the same birthdate *(day, month, NOT also same year, so two students considered born on the 28th Feb of different years will be counted)*.

```
[...]
if (students[i].getBirthDateDayMonth().equals(students[j].getBirthDateDayMonth()) ) {
  myCounter.increase();
}
[...]
```

Suppose that `myCounter` has NOT been initialized, so the method will crash with a `NullPointerException`.

If you consider a room of 50 students, how likely are you of getting a failure by running the system?

☐ **A. between 0-50% of the runs**

☒ **B. In more than 51% of the runs**

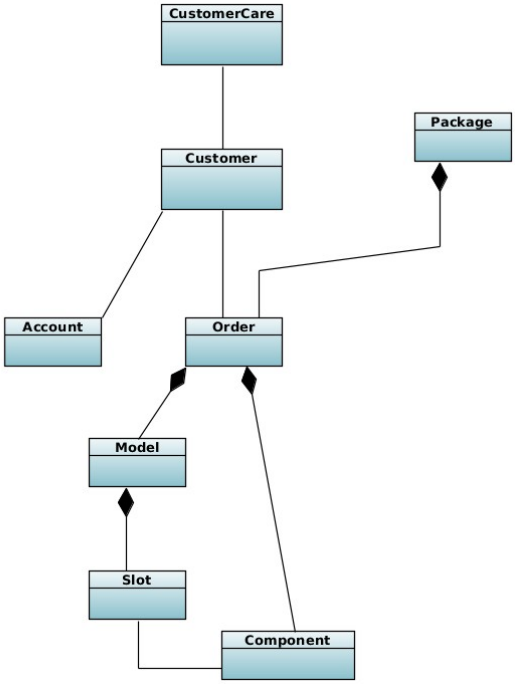
This question was to check a bit the intuition about failures that sometimes can be off quite considerably. This is exactly the same example as the birthday paradox (an explanation with solution here [1]). Contrary to what one would expect just by observing the numbers, p(x) that the failure happens is near 0.97 or in other terms something like 97 failures on 100 runs.
By intuition looking at the number of comparisons might give a better idea, as one student needs to be compared with 49, the second one with 48 and so on... However it can be difficult to quantify the overall number of all comparisons (there are some approximations, like here [2], but clearly it gets outside from pure intuition). The simplest solution is to look at students that do not have a birthday in common, so that P(x) = 1-P(¬x) so the first one will be 365/365 (for sure he will not have), the second one 364/365 and so on until 316/365. So all multiplied together 365/365 * 364/365 * …. 316/365 give 0.029 so that P(x) = 1-0.029=0.97

The code is just a sample, but it also contained two defects:
→ one missing ')' that has been corrected in this version - sorry for this;
→ there is no check that i≠j so P(x) would be 1.0 in that case;

We assigned one point for this question independently from the outcome, this was more an exercise to reason about failures and their probability of occurrence - in this case just with a simple nested loop it is quite difficult to quantify how often the faulty line could be called.

[1] http://salempress.com/store/pdfs/birthday.pdf
[2] http://stanford.edu/~maureenh/quals/html/stats/node27.html

Q2. Look at the following diagram *(from Pezzè & Young book)*, you can use it to answer the question by making examples out of it.



If we are starting development and we are following an incremental integration testing strategy, how is a threading integration different from a sandwich integration? How would you run them on the example given? (PLEASE ANSWER IN ENGLISH)

This example class diagram derives from the Pezzè and Young book [3] - the class diagram can be used to better clarify the two different strategies.

Sandwich or backbone integration would follow both a top-down and bottom-up strategy as it is more convenient case-by-case. For example, modules *CustomerCare* and *Customer* could be integrated top-down while *Account* and *Order* are stubbed. At the same time *Model*, *Slot* and *Component* could be integrated with *Order* using *Customer* and *Package* as drivers, and so on.

Threading integration looks at a specific functionality for the system, some examples that can be derived from the class diagram above could be the *finalization of orders, completing a payment process, checking a valid configuration for the order, and so on*. So each of the feature threads are incrementally tested by usually involving different classes out from the different modules.

[3] M. Pezzè and M. Young, *Software Testing And Analysis: Process, Principles And Techniques*. Hoboken, N.J.: John Wiley & Sons Inc, 2007.