boolean[] gotRight = new boolean[questionsAsked];

The reason why I made gotRight a boolean array instead of just a regular boolean is because the nature of the use of gotRight is to note if the student got the answer to the question right every time. If I just used a regular boolean, I would have had to create a ‘gotRight’ boolean for every question asked. Not only is this exceptionally spatially inefficient code (as it would have made programs more challenging to read), it is also impossible because the user could input that they want to be asked hundreds of thousands of questions. Creating boolean primitives in real-time for the user would be undoable. Therefore, I made a boolean array for gotRight. It declares as many elements as questions are asked, which are then filled in order according to the for loop that runs for as many times as there are questions (gotRight[count[]). The strengths of having gotRight as a boolean array include: having the ability to create as many elements as needed to store data and decreasing the amount of code on screen, and making the program easier to read. There are no limitations to using a boolean array over a regular boolean primitive with gotRight.

double percentCorrect = 0;

The reason why I made percentCorrect a double instead of an integer is because of how it has values assigned to it. The way it does this is through division, where percentCorrect = numberCorrectDouble/questionsAskedDouble. The reason why it’s so vital that percentCorrect is a double is because numberCorrectDouble is ALWAYS less than or equal to questionsAskedDouble. This means that percentCorrect will most likely be a number under 1.0. If percentCorrect = 0.99 (which represents 99% accuracy on the test) and percentCorrect has been declared as an integer, it will truncate the 0.99 to a 0; what was supposed to be an A+ would have become an F. In short, the strengths of having percentCorrect as a double are that the player can get a grade that isn’t an A+ or an F, and that the program doesn’t crash because of lossy compression that an integer would face when trying to be assigned a non-truncated double. The limitation of having percentCorrect as a double is that it takes up double the space that an integer would on the computer’s long term memory (doubles take up 8 bytes of storage whereas integers take up 4 bytes of storage).

int difficulty = 0;

The reason why I made difficulty an integer instead of a double is because of value assignment and how it affects if statements. If difficulty was a double, the if statements seeing if difficulty was between 1 to 4 would no longer work because with integers, it only had to deal with four numbers. With doubles, though, it deals with functionally infinite numbers. This completely breaks the program because there’s functionally no filter with values of difficulty. In short, the strengths of having difficulty as an integer is that there are only four possible difficulties that are allowed to be inputted, the exact amount of difficulties that I need. Another benefit is that integers take up half the memory that doubles do. There is no limitation to having a double over an integer in this case.