1(a). There are many characteristics that make code good. Some characteristics include; efficient, reliable, maintainable, readable, testable, well structured, simple, follows standards, and easy to change. If I were to pick out a top three I would say first and foremost it does have to be efficient. You could produce the most beautifully written code but if it can’t accomplish the task in a timely manner what good is it? For most things if it takes more than a few seconds user become annoyed. If your task list app takes 5 seconds to open user will find an alternative or be unhappy with the product. Secondly, I would choose readability. It’s important especially for larger systems with a long-life span to be readable by other programmers. And yourself for that matter. You may not touch the code for a year. Things will be forgotten and you will find yourself in a nightmare trying to remember what, why or how you did some task. Other programmers may need to maintain your code and you will want to make it easier for them to follow by using good naming conventions, sensible and simple logic and my third point well structured. There are multiple ways to do things but there is probably one way that makes the most sense.

1(b). A java program that can be understood is very much open to interruption. Working with the lousy java code for the Amazing project I was able to understand it. I had to be or else I wouldn’t be able to refactor it. However, it took me hours to grasp its logic and I’m still not completely sure of all of it. Understanding goes back to readability. Naming variable and methods with accurate description where there is no ambiguity is a great start. Keeping your classes small and having them focus on specific objects and not allowing them to be a catch all for your methods. Same goes with long functions, keep them small and performing only one task. When functions perform more than one task that is a sign they can probably be split in two or more separate functions. Some people just naturally understand the code better than others and will take shortcuts. This will not help down the road. There are principle and rules to follow to optimize your code so that it can be well understood by others and yourself when looking back. You should strive to meet all of the rules to put it in a position where it can be easily understood. But by just saying easily understood that is general and very open. Just strive to incorporate all the principle of good code writing.

2. I feel as if this code was intentionally written poorly. They did everything they could to not follow good coding principles. First none of the variable had names that made any sense. They were single letter variables such as x, c, q, z, h, and v. This gave you no indication of what they are for. The authors specifically created a function (goto) that was not included in the language because the way it created program flow makes it near impossible to follow. The entire logic was contained in a single function that was about 500 lines long in a single switch statement and loop. I can’t say they made a lot of mistakes and poor choices, they just made one very big poor choice. The overall design was hard to follow and understand. They were either lazy or really smart that they could remember and follow program execution. This had to be drawn up on paper prior to writing it. I don’t know how anyone can sit down and just write and remember the logic from previous steps. It did follow some order by starting at the top (I think 270 was the beginning) and doing various checks before finally writing its result (case 1130 was the end). If the programmer had named his variable with a descriptive name, did not use got statements and had 2 or 3 classes with small functions then the code would be magnitudes better. Those are the only real problems I see with the way it was written but those are big smells.