Assignment 2

Team 26

Question 1

This requirement is easily solved by labeling the sender of the message that creates the contract as the director of the contract.

Other functionality that should only be executable by the director (e.g. adding shareholders etc.) will then only need to use an appropriate *require()* check to block non-director addresses from using that functionality.

Question 2

Questions can be added using the *addFunction()* method. This function is only callable by the director, as we require the *msg.sender* to be the same as the address that initialized the contract.

A string (in memory) is needed and assigned to the name attribute of a new *Question* struct instance. Moreover, we flag the *Question* instance to be *open* by setting that flag to *true*. Otherwise, the default would be a closed *Question*.

After the *Question* is created, it is pushed to the *Question* array. *Question* instances are retrieved by index, meaning all future interactions with them require an index of the *Question* to be used.

Question 3

The functionalities of adding or removing shareholders are implemented in the *addShareholder()* and *removeShareholder()* functions. Only the director is allowed to perform such actions, thus *msg.sender* is required to be equal to the director.

We want to avoid adding duplicates in the *shareholders* array, so it is required to check if the address has already been added to the *shareholders* array. Moreover, deleting an address from the array object requires it to already be present in the array.

Restricting the viewing of results is done by requiring all non-director addresses to be in the *shareholders* array. Moreover, it is required for a *Question.open* flag to be set to *false* (i.e. the question needs to have been closed by the director) before shareholders can view the results.

Question 4

Per *Question* instance we keep track of a list of addresses of the shareholders who have already voted. This way we can prevent shareholders from voting more than once.

Moreover, we keep track of the result of the vote in a *uint* variable called *Question.voteCount*. We increment this value if a shareholder votes positively, otherwise we decrement it. By doing it here we save the computation later on, as altering an int by one value is much cheaper than having to compute the result from some sort of list of answers.

Question 5

The director can call the *closeQuestion()* function, closing the *Question* instance corresponding to the given index by setting the *Question.open* flag to *false*. Note that only the director can call it, as we require the *msg.sender* to be the director.

Computing the result is easily done by simply checking if the total of *Question.voteCount* is either higher or lower than 0. A string is sent back based on if the question was accepted, rejected or no decision was reached (in the case of a tie - i.e. a *Question.voteCount* equal to 0).

GIven that there is no need to breakdown the results further for inspection this is the simplest and most cost effective way to handle the computation of the results.