For testEmptyFile, we tested a file that was empty. We did this to ensure that the code was able to handle a situation where a file was empty. Through try, catch and if statements, we were able to make sure that the empty file did not cause the code to result in an error.

For test1, we tested a simple ‘abcd’ file of different characters. We did this to ensure that the code was able to create a frequencyTable, a huffmanTree, that allowed the code to traverse down the tree in order to create a hash map, which was the code map. With these, the compression method was able to read the characters in the text file using BufferedReader and write in the compressed file using BufferedBitWriter. Compression looked up the character’s code word in the code map and wrote the sequences of 0s and 1s as bits into the compressed out file.The decompression method traversed down the tree until it reached a leaf, which were the characters. It first read the compressed file using BufferedBitReader and the sequences of 0s and 1s, which were boolean bits from the compressed file. It turned to the left node if it was 0, and to the right node if it was 1. Once it reached a leaf, it retrieved the character data from that node and used BufferedWriter to write the character into the out decompressed file.

For test2, we wanted to test the code with different lengths of words and a mixture of single characters.
For testSingleCharacter, we wanted to test a text file with only one character to make sure that the code compressed and decompressed a text file even if it only had one character.
For testRepeatingCharacter, we wanted to test a text file with one character continuously repeating. This was to make sure that the code was able to compress and decompress a file that had a huffman tree of just one leaf.
Finally, we tested the long text files of the USConstitution and WarAndPeace. The compressed size of WarAndPeace was 1.8 MB.