Design:

I started with implementing the Huffman coding. My design started with creating a class for a Huffman node that had a symbol, probability, and a left and right pointer. Next I would build a tree using these nodes, and then use the tree to encode the message. Lastly I would use the tree to decode the message. I started building the tree using a hash map, but quickly found that this was to difficult and fiddly. I ended using a priority queue which allowed me to extract the lowest probability nodes and create a parent node. I continued in this manner until my priority queue only held one node. After creating my tree I then had to encode the message. To do this I walked down the tree till I reached a leaf node appending 1s or 0s depending on which way I went. I then had the code for that specific source symbol which I stored in a hash map. The only problem I ran into here was forgetting to delete a character when I went up the tree. After I had all the symbols encoded it was just a matter of iterating through the message encoding each symbol with their respective codes. Lastly I decoded by following the encoded message down the tree until I reached a leaf, then resetting.

In order to implement the arithmetic coding I used a combination of the algorithm set out in the lecture slides with the one in the suggested reading. I started in a similar manner to the Huffman, by creating a class to hold each symbol with its probability range. I then I had to convert the source symbols with their probabilities into arithmetic symbols. Then encoded the message, and lastly decode it. The most difficult part of the arithmetic coding was attempting to do bitwise calculations. I eventually gave up on this and instead hard coded a high, three quarter, half, and quarter value for the range. With these values I was able to check where the high end and low end of the range fell, for instance if the low end was over or equal to half then a 1 could be added to the encoding. The decoding was a matter of reverse engineering using a buffer of the encoded message. I decided to use the first symbol of the information source as my end of data symbol so I knew where the message ended.

Examples:

with the information source:
source symbols: a b c d

probabilities: .01 .25 .5 .24

Source Message:
dbbcbddbddbdbbbbccbdccbdcdcccddbbbdddbbcbdbbcbdccccbdbbbbcddcbbdcbdcbbcbb.
bdbdbbcbddbdbbbbcbcddbddddbbddbcbbcbbbbbbcdddbbbdcbbcddbccbcdbcbbbbcdcbbd
dbba

Huffman Coding:
Encoded Message:
000010110100000001000000010000101010111010001101000100011100000001010100000
000010110100001011010001111010000101010110000001010100010100010101101010100
010000101101000000010000101010110110000000100000000000001010000000110101101
101010101100000000001010100010101100000001110110000110101010110001010100000
0101001
Decoded Message:
dbbcbddbddbdbbbbccbdccbdcdcccddbbbdddbbcbdbbcbdccccbdbbbbcddcbbdcbdcbbcbb
bdbdbbcbddbdbbbbcbcddbddddbbddbcbbcbbbbbbcdddbbbdcbbcddbccbcdbcbbbbcdcbbd
dbba
Average Length: 2.0733333333333333

Arithmetic Coding:
Encoded Message:
110001001100111000001000101011101010010111100111010110010001100100010100101 1
110100011001111100111110111101011011110111011111111010011001001101100001110 11
011100100111011000010011111000011001111001010011110011111011100100011111101 10
001100111000101101010110011010100011010011000100000000000000000000000000
Decoded Message:
dbbcbddbddbdbbbbccbdccbdcdcccddbbbdddbbcbdbbcbdccccbdbbbbcddcbbdcbdcbbcbb
bdbdbbcbddbdbbbbcbcddbddddbbddbcbbcbbbbbbcdddbbbdcbbcddbccbcdbcbbbbcdcbbd
dbba
Average Length: 1.98

We can see that the Arithmetic coding outperformed the Huffman coding producing an average

length of 1.98 compared to 2.07. With the same information source:

Source Message: cdbdbbcbcbdbbbddccdbbbcbcccbdda

Huffman Coding:
Encoded Message:
100001000010110110100001010100000011000010101101111010000000001
Decoded Message: cdbdbbcbcbdbbbddccdbbbcbcccbdda
Average Length: 2.0

Arithmetic Coding:
Encoded Message:
101010100000101011110010010111011011111000101000100111000110000000000000000
00000
Decoded Message: cdbdbbcbcbdbbbddccdbbbcbcccbdda
Average Length: 2.6129032258064515

We can see that as the source message gets smaller the Arithmetic coding is outperformed by the Huffman coding.

Conclusion:

On the whole I found that this practical was difficult because of java and would have possibly been easier in C where more bit control is made available. I feel that if I could do the practical over again I would have read more of the literature on it and drawn out a full game plan before jumping in, as I found myself doing a lot of backtracking.