1. What is the largest value of n (choosing n as a power of 10) that you can use that mostly stays out of deadlock? Your answer only needs to be approximate.
n = 1000

2. What is the smallest value of n (choosing n as a power of 10) that you can use to reliably reach deadlock? Your answer only needs to be approximate.
n = 1

3. How does this system behave if t1 and t2 access their shared resources in the exact same order? Make the necessary changes to the code and test it.
Explain why you see what you see. Does the system still reach deadlock? Why or why not?
The system will never reach deadlock, because two processes can access to a single resource at the same time.

4. Remove all of the synchronization from the code. Remove the "synchronized (resourceX)" coding. What do you see when you run the program?
A different result every time, because two threads will run alternately and randomly.

5. Experimenting only with the value n, does the program work with n = 10, 100, 1000, 10000, 100000, 100000000 ?
It works.

6. Swap the lines "t1.start();" with "t2.start()". Does the program work with n = 10, 100, 1000, 10000, 100000, 100000000 ? Explain what you find.
It works. Nothing changes.

7. Remove the synchronized keywords from the Account class methods. Does the program work with n = 10 ? Describe what happens.
Sometimes, the balance is not 10.

8. With the synchronized keywords removed from the Account class methods. Does the program work with n = 100, 1000, 10000, 100000, 1000000 ? Describe what happens.
The balance might be some values other than 0 (abs<n)

9. Currently, we are adding n elements to the queue and removing n elements from the queue. Explain what happens if we add 2n (1,2,3,...,n,n+1,...,2n) elements to the queue rather than n.The removal thread is left unchanged.
Nothing happened.

10. Currently, we are adding n elements to the queue and removing n elements from the queue. Explain what happens if we only add n-1 elements to the queue rather than n. The removal thread is left unchanged.
The program will never stop. The last elements showed in the screen is n-1. Since the wait() command waits for all background processes to finish.

11. This is the same as question 10 but after adding 1,2,3,...,n-1 to the queue, we would like to put the thread to sleep for 5 seconds and then add n to the queue. The queue ends up taking on 1,2,3,...,n but it does so in two parts. First, it adds 1,2,3,...,n-1 to the queue, and then it sleeps. When it awakes, it adds one value, n, to the queue. Use this code
The program will first output 1,2,3,...,n-1 , and then stop for 5 seconds, and finally show n to the screen.