Lucas Alexandre Figueiredo Albuquerque

1.
Nessa situação não ocorrerá deadlock, pois apesar de utilizar uma estrutura de synchornized aninhados o que poderia promover o deadlock, as duas threads "dão" o synchornized nas mesmas variáveis na mesma ordem, então se t1 delas conseguir o "b" t2 não tenta pegar o "a" ela fica esperando o objeto "b" se tornar disponível, já a t1 pega o a e executa o código . Caso o código fosse:
C t1 = new C(a1,b1)
C t2 = new C(b1,a1)
t1.start();
t2.start();
Aí sim o código podeira entrar em deadlock, pois a thread t1 poderia dar o synchornized em b1 e antes de conseguir dar em a1 a thread t2 daria em a1 assim ocorrendo deadlock pois nenhuma das duas liberaria as suas respectivas variáveis travadas.

2.
```
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class Q2 {
	public static BlockingQueue Numeros;

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Q2 blo = new Q2(100);
		Runnable c = new consumidor(blo.Numeros);
		Runnable p = new produtor(blo.Numeros);

		Thread prod = new Thread(p);
		Thread cons = new Thread(c);
		prod.start();
		cons.start();

	}

	Q2(int x) {
		this.Numeros = new ArrayBlockingQueue<>(x);

	}

	static class consumidor implements Runnable {
		BlockingQueue blo;

		consumidor(BlockingQueue bloq) {
			this.blo = bloq;
		}

		@Override
		public void run() {
			while (true) {
			try {
				System.out.println("Consumido: "+blo.take());
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
			}
```

```
		}

	}

	public static class produtor implements Runnable {
		BlockingQueue blo;

		produtor(BlockingQueue bloq) {
			this.blo = bloq;
		}

		@Override
		public void run() {
			// TODO Auto-generated method stub
			int i = 0;
			while (true) {
				try {
					this.blo.put(i);
					System.out.println("Produzido: "+i);
					Thread.sleep(500);
					i++;
				} catch (InterruptedException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}

			}
		}

	}

}
```

3.

```
module Main where
import Control.Concurrent
import Control.Concurrent.STM

type Buffer myBuffer = TVar [myBuffer]

newBuffer :: [myBuffer] -> IO (Buffer myBuffer)
newBuffer = newTVarIO

get :: Buffer meuBuffer -> STM meuBuffer
get buffer = do
   myBuffer <- readTVar buffer
   if null myBuffer
      then retry
      else return (head myBuffer)

put :: Buffer myBuffer -> myBuffer -> STM()
put buffer myBuffer = do
   xs <- readTVar buffer
   writeTVar buffer (myBuffer:xs)

produtor :: Buffer Int -> Int -> IO ()
produtor buffer myBuffer = do
```

```
    threadDelay 1000000
    atomically (put buffer myBuffer)
    produtor buffer (myBuffer + 1)

consumidor :: Buffer Int->Buffer Int->Buffer Int->Buffer Int-> IO ()
consumidor faca pao carne tomate = do
    threadDelay 1000000
    a <- atomically (get faca)
    b <- atomically (get pao)
    c <- atomically (get carne)
    d <- atomically (get tomate)
    print ("Sanduiche Feito por Funcionario 1")
    atomically (put faca 1)
    consumidor faca pao carne tomate

consumidor2 :: Buffer Int->Buffer Int->Buffer Int->Buffer Int-> IO ()
consumidor2 faca pao carne tomate = do
    threadDelay 100000
    a <- atomically (get faca)
    b <- atomically (get pao)
    c <- atomically (get carne)
    d <- atomically (get tomate)
    print ("Sanduiche Feito por Funcionario 2")
    atomically (put faca 1)
    consumidor2 faca pao carne tomate

main :: IO()
main = do
    pao <- newBuffer []
    carne <- newBuffer []
    tomate <- newBuffer []
    faca <- newBuffer [1]
    forkIO $ consumidor faca pao carne tomate
    forkIO $ consumidor2 faca pao carne tomate
    forkIO $ produtor pao 30
    forkIO $ produtor carne 30
    forkIO $ produtor tomate 30
    readLn
```