Q1

Não irá haver um deadlock já que os objetos a e b de ambas as threas reiniciam o objeto, então, caso uma das threads obter o controle do monitor do objeto b a outra thread não podera ter e quem irá ter o controle do monitor de b, também irá ter o controle do monitor de a.
Um outro fator é que não se poderá conter uma chamada para uma outra thread já que uma thread não tem referência para a outra thread, resumindo, nesse programa necessariamente uma thread irá obter o controle de b, e então sera a única que tentara ter o controle de a, então executara o bloco onde não podera invocar a outra thread para gerar o deadlock e acabara a sua execucao e então a outra thread irá etrar em execução.

Q2

```
// Aluno Saulo Alexandre de Barros (SAB2)
public class ex2 {

public static void main(String[] args) {

   // Descricao da ideia do codigo
   // Uma fila bloqueante onde um produtor insere numeros aleatórios e o consumidor irá
retirar esse numero depois

   int tamanhoMinhaFila = 10;

   filaDeInteiros fila = new filaDeInteiros(tamanhoMinhaFila);

   Produtor inserirNaFila = new Produtor(fila);
   Consumidor retiraDaFila = new Consumidor(fila);

Thread threadProdutor = new Thread(inserirNaFila);
Thread threadConsumidor = new Thread(retiraDaFila);
threadProdutor.start();
threadConsumidor.start();
}

}

class Produtor implements Runnable {

private filaDeInteiros produtor;

public Produtor(filaDeInteiros produtor) {
this.produtor = produtor;
}

public void run( ) {
   try {
      while(true) {
this.produtor.push((int) (Math.random() * 100));
Thread.sleep(500);
}
} catch (InterruptedException e) {
```

```
e.printStackTrace();
}
}
}

class Consumidor implements Runnable {

private filaDeInteiros consumidor;

public Consumidor(filaDeInteiros consumidor) {
this.consumidor = consumidor;
}

public void run( ) {
   try {
    while(true) {
       this.consumidor.pop();
    Thread.sleep(1500);
}
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}


class filaDeInteiros {
private int[] fila;
private int tamanho;

public filaDeInteiros(int tamanho) {
this.fila = new int[tamanho];
this.tamanho = 0;
}

public synchronized void push(int valor) throws InterruptedException {
while (this.tamanho == this.fila.length) {
   System.err.println("Fila bloqueada por estar cheia");
wait();
}
System.out.println("Colocando o inteiro " + valor + " na fila");
this.fila[this.tamanho++] = valor;
notifyAll();
}

public synchronized int pop() throws InterruptedException {
while (this.tamanho == 0) {
System.out.println("Fila vazia");
wait();
}
int numeroRetirado = this.fila[0];
```

```
System.out.println("Retirando o inteiro " + numeroRetirado + " da fila");
for (int i = 0; i < this.tamanho - 1; i++) {
this.fila[i] = this.fila[i + 1];
}
this.tamanho--;
notifyAll();
return numeroRetirado;
}

public synchronized int getSize() {
return this.tamanho;
}
public synchronized boolean estaCheio() {
if(this.tamanho == fila.length) {
   return true;
} else {
   return false;
}
}
}
```

Q3

```
-- Aluno Saulo Alexandre de Barros (SAB2)
-- Não consegui implementar tudo devido ao tempo mas o programa contem a lógica que idealizei para a execução do mesmo
import Control.Concurrent
import Control.Concurrent.STM


type Buffer meuSanduiche = TVar [meuSanduiche]

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

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

put :: Buffer meuSanduiche -> meuSanduiche -> STM()
put buffer meuSanduiche = do
sanduiche <- readTVar buffer
writeTVar buffer (meuSanduiche:sanduiche)

produtor :: Buffer Int->Buffer Int->Buffer Int->Buffer Int->IO ()
produtor buffer meuSanduiche = do
threadDelay 1000000
atomically (put buffer meuSanduiche)
produtor buffer (meuSanduiche)
```

```
consumidor1 :: Buffer Int->Buffer Int->Buffer Int->Buffer Int->IO()
consumidor faca pao tomate carne = do
threadDelay 500000
a1 <- atomically (get faca)
a2 <- atomically (get pao)
a3 <- atomically (get tomate)
a4 <- atomically (get carne)
print("O produtor 1 estah fazendo o senduiche")
consumidor buffer

consumidor2 :: Buffer Int->Buffer Int->Buffer Int->Buffer Int->IO()
consumidor faca pao tomate carne = do
threadDelay 700000
a1 <- atomically (get faca)
a2 <- atomically (get pao)
a3 <- atomically (get tomate)
a4 <- atomically (get carne)
print("O produtor 2 estah fazendo o senduiche")
consumidor buffer

main :: IO()
main = do
buffer <- newBuffer []
forkIO $ consumidor1 buffer
forkIO $ consumidor2 buffer
forkIO $ produtor buffer 30
readLn
```

Q4

```
-- Aluno Saulo Alexandre de Barros (SAB2)
function newpoly(lista, x)

final = 0
arraynumeros = {}

for i,v in pairs(lista) do
arraynumeros[i] = v
end
final = ( arraynumeros[1] * x * x ) + ( arraynumeros[2] * x ) + arraynumeros[3]

return final

end

listaComOsPolinomios = {3,0,1}

function f(x)
return newpoly(listaComOsPolinomios, x)
end
```

print(f(0)) --> 1
print(f(5)) --> 76
print(f(10)) --> 301