PRIMEIRA

```
import java.util.concurrent.atomic.AtomicInteger;

public class ProdutorConsumidor {

    Produtor p;
    Consumidor c;
    AtomicInteger ai;

    public ProdutorConsumidor(int n) {
        this.ai = new AtomicInteger(n);
        this.p = new Produtor(this.ai);
        this.c = new Consumidor(this.ai);
    }

}

class Consumidor implements Runnable {

    AtomicInteger ai;

    public Consumidor(AtomicInteger ai) {
        this.ai = ai;
    }

    public void run() {
        while (true) {

            try {
                // Dar um tempo para consumir
                Thread.sleep(10000);
            } catch (Exception e) {}
            System.out.println("Consumindo");
            this.ai.getAndDecrement();
        }
    }
}

class Produtor implements Runnable {

    AtomicInteger ai;

    public Produtor(AtomicInteger ai) {
        this.ai = ai;
    }
```

```
    public void run() {
        while (true) {

            try {
                // Dar um tempo para produzir
                Thread.sleep(4000);
            } catch (Exception e) {}
            System.out.println("Produzindo");
            System.out.println(this.ai.get());
            this.ai.getAndIncrement();
        }
    }
}
```

SEGUNDA

```
public class BlockingQueue {

    private int indexPut;
    private int tamFila;
    private int indexTake;
    private int qtItens;
    public int[] fila;



    public BlockingQueue(int qt) {
        this.indexPut = 0;
        this.tamFila = qt;
        this.indexTake = 0;
        this.qtItens = 0;
        this.fila = new int[qt];
    }

    public synchronized void put(int n) {
        try {
            // Se a fila estiver cheia espera ficar livre.
            while(this.qtItens == this.tamFila) {
                wait();
            }
            this.qtItens++;
            this.fila[this.indexPut++ % this.tamFila] = n;
            notifyAll();
        } catch(Exception e) {}

        // this.t[this.indexPut++ % this.qtFila] = n;
    }
```

```
    public synchronized int get() {
        int retorno = 0;
        try {
            while(this.qtItens == 0) {
                wait();
            }
            this.qtItens--;
            retorno = this.fila[this.indexTake % this.tamFila];
            this.fila[this.indexTake++ % this.tamFila] = 0; //
Limpa a casa retirada
            notifyAll();

        } catch (Exception e) {}
        return retorno;
    }
}
```

TERCEIRA

```
import Control.Concurrent
import Control.Concurrent.MVar
import Control.Concurrent.STM



produtor :: TVar Int -> String -> IO()
produtor componente nome = do

    atomically (do
        tempComponente <- readTVar componente
        --putStrLn ("Produzindo " ++ nome)
        --putStrLn (show tempComponente)
        writeTVar componente (tempComponente + 1))

    produtor componente nome


produtorCaixa :: TVar Int -> TVar Int -> TVar Int -> IO()
produtorCaixa parafusos porcas caixaPar = do

    atomically (do
        qtParafuso <- readTVar parafusos
```

```
        qtPorcas <- readTVar porcas

        if qtParafuso > 0 && qtPorcas > 0 then do
            qtCaixa <- readTVar caixaPar
            --putStrLn "Produzindo um par"
            --putStrLn (show qtCaixa + 1)
            writeTVar caixaPar (qtCaixa + 1)
            writeTVar parafusos (qtParafuso - 1)
            writeTVar porcas (qtPorcas - 1)
        else retry)

    produtorCaixa parafusos porcas caixaPar

main :: IO()
main = do

    porcas <- newTVarIO 0
    parafusos <- newTVarIO 0
    caixaPar <- newTVarIO 0

    forkIO (produtor porcas "porca")
    forkIO (produtor parafusos "parafuso")
    forkIO (produtorCaixa parafusos porcas caixaPar)
    threadDelay 50000000000 -- Fazer com que o programa não acabe
    return ()
```

QUARTA

```
newpoly = function(fatores)
    qtFatores = #fatores
    novaFuncao = function(x)
        fatores = fatores
        it = qtFatores
        total = 0
        for i=1,it do
            total = total + fatores[i] * (x ^ (it - i))
        end
        return total
```

```
    end
    return novaFuncao
end

f = newpoly({3,0,1})
print(f(0))
print(f(5))
print(f(10))
```

QUINTA

```
content.function receive()
    local status, value = coroutine.resume(producer)
    return value
end

function send (x)
    print("enviando valor")
    coroutine.yield(producer, x)
end

consumer = coroutine.create(
    function ()
        while true do
            local x = receive()
            io.write(x, "\n")
        end
    end)

producer = coroutine.create(
    function ()
        local i = 0
        while true do
            send(i)
            i = i + 1
        end
    end)

coroutine.resume(consumer)
```