1. Monitoriaus klasė:

```
public class Monitor {

    private int a;

    //metodas skaičiaus pridėjimui prie a
    public synchronized void add(int x) {
        a += x;
    }

    //metodas skaičiaus atėmimui iš a
    public synchronized void sub(int x) {
        a -= x;
    }

    //metodas a spausdinimui
    public void print() {
        System.out.println(a);
    }

    Monitor(int a) {
        this.a = a;
    }
}
```

Naudojimo pavyzdys: dvi gijos, kurių viena šimtą kartų prided atsitiktinį skaičių prie a, kita – atima. Abi kas 10 operacijų išspausdina a reikšmę.

```
public static void main(String[] args) {
        // sukuriamas monitorius ir atsitiktinių skaičių generatorius
        Monitor m = new Monitor(0);
        Random r = new Random();

        //pirmas procesas, kuris šimtą kartų prideda atsitiktinį skaičių
        Runnable r1 = new Runnable() {
            @Override
            public void run() {
                for (int i = 1; i <= 100; i++) {
                    int num = r.nextInt(100);
                    m.add(num);
                    System.out.println("Thread 1: adding " + num );

                    if(i % 10 == 0) {
                        m.print();
                    }
                }
            }
        };

        //antras procesas, kuris šimtą kartų atima atsitiktinį skaičių
```

```
        Runnable r2 = new Runnable() {
            @Override
            public void run() {
                for (int i = 1; i <= 100; i++) {
                    int num = r.nextInt(100);
                    m.sub(num);
                    System.out.println("Thread 2: subtracting " + num );

                    if(i % 10 == 0) {
                        m.print();
                    }
                }
            }
        };

        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch(Exception e) {
            System.err.println("failed to join threads");
        }
    }
```

2. Pakeičiame Monitor klasės metodus add() ir sub() į tokius:

```
    //metodas skaičiaus pridėjimui prtie a
    public void add(int x) {
        synchronized(lock) {
            while(a >= 100) {
                try {
                    lock.wait();
                } catch (InterruptedException ex) {
                    break;
                }
            }

            a += x;

            lock.notifyAll();
        }
    }

    //metodas skaičiaus atėmimui iš a
    public void sub(int x) {
        synchronized(lock) {
```

```
            while(a <= 0) {
                try {
                    lock.wait();
                } catch(InterruptedException ex) {
                    break;
                }
            }

            a -= x;

            lock.notifyAll();
        }
    }
```

Monitoriaus panaudojimas išlieka toks pat.

3. Jeigu turime matricą $\begin{bmatrix} a_{00} & \dots & a_{0n} \\ \dots & \ddots & \dots \\ a_{m0} & \dots & a_{mn} \end{bmatrix}$ ir vektorių $\begin{bmatrix} x_0 \\ \dots \\ x_n \end{bmatrix}$, tai jų sandauga yra vektorius $\begin{bmatrix} a_{00}x_0 + a_{01}x_1 + \dots + a_{0n}x_n \\ \dots \\ a_{m0}x_0 + a_{m1}x_1 + \dots + a_{mn}x_n \end{bmatrix}$.
   Turime n*m gijų ir n kanalų, kur gija$_{ij}$ suskaičiuoja sandaugą a_{ij} *x_j ir siunčia ją į j-ąjį kanalą. Toliau yra dar n gijų, kur gija$_i$ suskaičiuoja k-ojo kanalo skaičių sumą ir padeda į k-ąją rezultatų vektoriaus poziciją.
4. Yra n tiekėjų ir m pirkėjų. Masyvuose A_i, yra tiekėjo T_i tiekiami produktai, o masyvuose B_j yra pirkejų P_j perkami produktai (i ∈ [0; n), j ∈ [0; m)); Tiekėjai vienu metu tiekia prekes parduotuvei, pirkėjai – karto nurodytus pirkimus. Veiksmai atliekami tol, kol galimi pasikeitimai parduotuvėje; Parduotuvė turi maksimalų galimų prekių kiekį. Parašykite programą, kuri parodytų, kaip laikui bėgant parduotuvėje kinta prekės ir jų kiekiai.

   Tokiame uždavinyje gamintojams reikia numatyti, kada vartotojai nebegalės nieko suvartoti ir nustoti gaminti, o vartotojams reikia numatyti, kada gamintojai baigs gaminti ir nustoti vartoti, kai suvartojama viskas, kas įmanoma. Trumpai tariant, reikia išvengti *deadlock* situacijos – gamintojai laukia, kol vartotojai kažką suvartos ir atlaisvins vietą bendroje atmintyje, o vartotojai neturi, ką vartoti ir laukia, kol gamintojai ką nors pagamins.
5. V11, v12, v21, v22, v31, v32;
   V11, v12, v31, v32, v21, v22;
   V21, v22, v11, v12, v31, v32;
   V21, v22, v31, v32, v11, v12;
   V31, v32, v11, v12, v21, v22;
   V31, v32, v21, v22, v11, v12;