1. Kodo fragmentas (1-2 dalys):

```
class Counter {
      public static void executeThreads () {
            Thread t1 = new Thread(1, 1, mon);
            Thread t2 = new Thread(2, 2, mon);
            Thread t3 = new Thread(3, 3, mon);

            Thread t4 = new Thread(4, 4, mon);
            g1.start(); g2.start(); g3.start(); g4.start();
      }
      public static void main(String[] args) throws IOException {
            //sukuriamas monitorius
            Monitor mon = new Monitor();
            //sukuriamos gijos
            executeThreads(Monitor mon);
            System.out.println("Programa baige darba\n");
      }
}

class Monitor {
      //skaitliukas
      protected long count = 100;

      //naudojami sinchronizuoti metodai
      public synchronized long count() {return count;}
      public synchronized void inc() throws InterruptedException {...}
      public synchronized void dec() throws InterruptedException {...}
      //naudojami nesinchronizuoti metodai
      protected void setCount (long newValue) {...}
      protected void awaitUnderMax() throws InterruptedException {...}
      protected void awaitOverMin() throws InterruptedException {...}
}

public synchronized void inc(long value) throws InterruptedException {
      //laukia, kol skaitiklis nebuvo sumažintas
      awaitUnderMax();
      //padidina skaitiklio reikšmę vienetu
      setCount(count + value);
}
public synchronized void dec(long value) throws InterruptedException {
      //laukia, kol skaitiklis nebuvo padidintas
      awaitOverMin();
      //sumažina skaitiklio reikšmę vienetu
      setCount(count - value);
}

protected void setCount (long newValue) {
```

```
      //pakeičia skaitiklio reikšmę
      count = newValue;
      //išspausdina reikšmę
      System.out.println("Reikšmė: " + count);
      //pažadina visas gijas priklausančias nuo naujos reikšmės
      notifyAll();
}

protected void awaitUnderMax() throws InterruptedException {
      //kol skaitliukas yra maksimalus, laukti kol sumažės
      while (count >= 200) wait();
}

protected void awaitOverMin() throws InterruptedException {
      //kol skaitliukas yra minimalus, laukti kol padidės
      while (count <= 20) wait();
}

class CounterThread extends Thread {
      protected long count;
      protected long thread_id;
      Monitor mon;

      public CounterThread(long value, long thread_id, Monitor monitor)
{
            this.count = value;
            this.thread_id = thread_id;
            Monitor mon = monitor;
      }
      public void run() {
            switch(thread_id) {
                  case 0:
                        mon.inc(count);
                        break;
                  case 1:
                        mon.inc(count);
                        break;
                  case 2:
                        mon.inc(count);
                        break;

                  case 3:
                        mon.dec(count);
                        break;

            }
      }

}
```

2. (pateikta 1 dalyje)
3. Žinučių perdavimas vyksta, kai vienas procesas siunčia žinutes, o kitas priėma žinutes. Yra keli žinučių perdavimo modeliai, iš jų populiariausias MPI, kuris vadinamas pranešimų perdavimo modeliu, jame programa yra sudaryta iš lygiagrečiai veikiančių procesų, kurie komunikuoja tarpusavyje be tarpininkų. Šiame modelyje galima realizuoti ir sinchroninį ir asinchroninį perdavimo būdą. Kitas modelis yra CSP. Jame visi procesai naudoja sinchroninius perdavimo kanalus, kuriais perduoda duomenis tarpusavyje per tarpininką (*channel - kanalą*).

   Dažniausiai žinutės perduodamos per kanalus, jie būna kelių tipų:

One2One perduoda duomenis stabiliai tik tarp dviejų procesų, vienas siunčia kitas gauna.
One2Many siunčia duomenis vienas procesas, priėma - keli.
Many2One siunčia duomenis daug procesų, priėma – vienas.
Many2Many siunčia duomenis daug procesų, priėma duomenis daug procesų.

4. Sąlyginė sinchronizacija JAVA yra atliekama pasinaudojant *wait, notify, notifyAll* funkcijomis. *Wait* funkcija nusako, kad gija turi laukti kol bus pažadinta su *notify* arba *notifyAll* metodu. *Notify* ir *notifyAll* praneša vienam konkrečiam arba visiems procesams (gijoms). Metoduose reiktų naudoti *synchronized* raktažodį, kuris nurodo, kad veiksmai tame metode turi būti vykdomi vienos gijos.

   Pavyzdys:

```
class SimpleBoundedCounter {
      //skaitliukas
      protected long count = MIN;

      //naudojami sinchronizuoti metodai
      public synchronized long count() {return count;}
      public synchronized void inc() throws InterruptedException {...}
      public synchronized void dec() throws InterruptedException {...}
      //naudojami nesinchronizuoti metodai
      protected void setCount (long newValue) {...}
      protected void awaitUnderMax() throws InterruptedException {...}
      protected void awaitOverMin() throws InterruptedException {...}
}

public synchronized void inc() throws InterruptedException {
      //laukia, kol skaitiklis nebuvo sumažintas
      awaitUnderMax();
      //padidina skaitiklio reikšmę vienetu
      setCount(count + 1);
}
public synchronized void dec() throws InterruptedException {
```

```
      //laukia, kol skaitiklis nebuvo padidintas
      awaitOverMin();
      //sumažina skaitiklio reikšmę vienetu
      setCount(count - 1);
}

protected void setCount (long newValue) {
      //pakeičia skaitiklio reikšmę
      count = newValue;
      //pažadina visas gijas priklausančias nuo naujos reikšmės
      notifyAll();
}

protected void awaitUnderMax() throws InterruptedException {
      //kol skaitliukas yra maksimalus, laukti kol sumažės
      while (count == MAX) wait();
}

protected void awaitOverMin() throws InterruptedException {
      //kol skaitliukas yra minimalus, laukti kol padidės
      while (count == MIN) wait();
}
```

5. Kodo fragmentas:

```
// kintamasis CPU
int a;

//kintamasis GPU

int *dev_a;

//paskiriama atmintis kintamajam GPU

cudaMalloc((void**)&dev_a, sizeof(int));

//kopijuojamas kintamasis iš a į dev_a (iš CPU į GPU)

cudaMemcpy(dev_a,a,dydis,cudaMemcpyHostToDevice);

//atliekami veiksmai su dev_a (GPU a)

veiksmas<<<1, 1>>>(dev_a);

//kopijuojamas kintamasis iš dev_a į a (iš GPU į CPU)

cudaMemcpy(c,dev_c,dydis,cudaMemcpyDeviceToHost);

//atlaisvinama kintamojo atmintis GPU
```

```
cudaFree(dev_a);
```