TÖL304G
Forritunarmál
Verkefnablað 13

Snorri Agnarsson

13. nóvember 2022

Íslenska

Verkefni

Hópverkefni

Skrifið Java forrit þar sem tveir þræðir, hvor um sig, framleiða heiltölurnar 1^2, 2^2, ..., 10000^2, senda þær hverja eftir aðra áleíðis í aðra tvo þræði, gegnum sömu samskiptarás. Þræðirnir sem taka á móti tölunum skulu hvor um sig sækja 10000 tölur úr samskiptarásinni og leggja þær saman og skrifa summuna. Þið munuð þurfa long breytu til að reikna summuna.

Skilið forritstextanum og sýnið einnig úttakið úr þremur keyrslum á forritinu ásamt meðaltali talnanna tveggja sem skrifaðar voru. Allir ættu alltaf að fá sama meðaltal en ekki endilega sömu tvær skrifaðar tölur.

Inni í forritinu skuluð þið hafa þrjár gerðir hluta, sem eiga að vera innri klasar í ykkar aðalklasa, og skulu heita Producer, Container og Consumer. Producer og Consumer eiga að vera þræðir, þ.e. undirklasar klasans Thread, en Container á að vera ílát (samskiptarás) sem getur annað hvort verið tómt eða innihaldið eitt gildi. Einfaldast er að nota static breytu til að vísa á þann Container hlut sem notaður er sem samskiptarás, því margir þræðir geta notað sömu static breytu.

Munið að skrifa skýra og rétta fastayrðingu fyrir sérhverja lykkju.

Container klasinn skal vera einhvern veginn svona:

```
// Hvert tilvik af þessum klasa er ílát fyrir
// eina heiltölu (int).  Fleiri en einn þráður
// getur notað ílátið án þess að spilla
```

```
// því að ílátið sé í réttu ástandi.
static class Container
{
    boolean isEmpty = true;
    int theValue;
    // Fastayrðing gagna:
    //   Ílátið er tómt þþaa isEmpty sé true.
    //   Ef ílátið er ekki tómt þá inniheldur
    //   theValue gildið í ílátinu.

    // Notkun: Container c = new Container();
    // Fyrir:  Ekkert.
    // Eftir:  c vísar á nýjan tóman Container.
    // Ath.:   Ekki þarf að forrita þennan smið,
    //         hann verður sjálfkrafa til.

    // Notkun: c.put(x);
    // Eftir:  Búið er að setja x í ílátið c.
    //         Ef til vill þurfti að bíða eftir
    //         að ílátið tæmdist áður en það
    //         tókst.
    public synchronized void put( int x )
        throws InterruptedException
    {
        while( !isEmpty )
            // Fastayrðing gagna er sönn.
            // Þessi þráður hefur núll sinnum eða
            // oftar kallað á wait() til að reyna
            // að fá pláss til að setja gildið x
            // í hlutinn. Á undan sérhverju slíku
            // kalli var ekki pláss, þ.e. isEmpty
            // var ósatt.
            wait();
        isEmpty = false;
        theValue = x;
        notifyAll();
    }

    // Notkun: x = c.get();
    // Eftir:  Búið er að sækja x úr ílátinu c.
    //         Ef til vill þurfti að bíða eftir
    //         að ílátið fylltist áður en það
```

```
    //          tókst.
    public synchronized int get()
    throws InterruptedException
    {
        while( isEmpty )
            // Fastayrðing gagna er sönn.
            // Þessi þráður hefur núll sinnum eða
            // oftar kallað á wait() til að reyna
            // að gildi í hlutinn til að fjarlægja.
            // Á undan sérhverju slíku kalli var
            // ekki gildi í hlutnum, þ.e. isEmpty
            // var satt.
            wait();
        int x = theValue;
        isEmpty = true;
        notifyAll();
        return x;
    }
}
```

Athugið notkunina á boðunum wait() og notifyAll() og virkni þeirra í samhengi við lykilorðið synchronized. Þessi tvö boð eru skilgreind í klasanum Object og eru því til staðar í sérhverjum Java hlut. Við ræðum þessa virkni í fyrirlestri.

Einstaklingsverkefni

Skrifið Morpho forrit eða fall sem gerir sama og Java forritið að ofan. Þið megið skrifa forritið sem Morpho script eða þýðingarhæft forrit eftir því hvað ykkur hentar. Einfaldast er að nota channel sem samskiptarás, sem virkar þá svipað og Container að ofan. Fallið makeChannel býr til nýjan channel. Til að senda gildi inn í channel má nota *tvíundaraðgerðina* `<-`. Til að sækja gildi úr channel má nota *einundaraðgerðina* `<-`. Þessar aðgerðir hafa sama nafn en eru ekki sama aðgerðin. Ef breytan `c` inniheldur channel þá mun segðin `c<-x` senda (skrifa) gildið `x` í þann channel. Segðin `x = <-c` mun sækja (lesa) gildi úr þeim channel og setja í breytuna `x`.

Einföld leið til að koma í veg fyrir yfirflæði er að frumstilla breytuna, sem summerað er í, með gildinu `bigInteger(0)`.

English

Assignments

Group Assignment

Write a Java program where two threads, each by itself, produce the integers 1^2, 2^2, ..., 10000^2, sends them, one after the other forward to two other threads, through the same communication channel. The receiving threads shall each fetch 10000 numbers from the communication channel and add them together and write out the sum. You will need a long variable to compute the sum.

Turn in the source code and also show the output from three runs of the program along with the average of the two written numbers. All should always get the same average but not necessarily the same two written numbers.

In the code you should have three types of objects, which should be inner classes in your main class, and should be named Producer, Container, and Consumer. Producer and Consumer should be threads, i.e. subclasses of the Thread class, but Container should be a container (a communication channel) which can either be empty og contain one value. The simplest way of connecting is to use a static class variable to refer to the Container used, since many threads can use the same static variable.

Remember t-o write a clear and correct invariant for each loop.

The Container class should be along these lines:

```
// Each instance of this class is a container
// for one integer (int). More than one thread
// can use the container without spoiling the
// correct state of the container.
static class Container
{
    boolean isEmpty = true;
    int theValue;
    // Data invariant:
    //   The container is empty iff isEmpty is
    //   true. If the container is not empty
    //   then theValue is the value in the
    //   container.

    // Use:   Container c = new Container();
    // Pre:   Nothing.
    // Post:  c refers to a new empty Container.
    // Note:  This constructor does not have to
    //        be programmed, it will be generated
    //        automatically.
```

```
// Use:   c.put(x)
// Post:  x has been put into the container c.
//        Perhaps this involved waiting for the
//        container to become empty before this
//        was successful.
public synchronized void put( int x )
    throws InterruptedException
{
    while( !isEmpty )
        // The data invariant is true.
        // This thread has zero times or more
        // often called wait() to attempt to
        // get space in the container to insert
        // x. Before each such call the space
        // was not available, i.e. isEmpty was
        // false.
        wait();
    isEmpty = false;
    theValue = x;
    notifyAll();
}

// Use:   x = c.get();
// Post:  x has been fetched from the container c.
//        Perhaps this involved waiting for the
//        container to become non-empty before
//        this succeeded.
public synchronized int get()
throws InterruptedException
{
    while( isEmpty )
        // The data invariant is true.
        // This thread has zero times or more
        // often called wait() to attempt to
        // get a value into the container to
        // fetch. Before each such call no
        // value was available, i.e. isEmpty was
        // true.
        wait();
    int x = theValue;
    isEmpty = true;
```

```
            notifyAll();
            return x;
        }
}
```

Note the use of wait() and notifyAll() and their functioning in relation to the keyword synchronized. These two methods are defined in the class Object and are therefore available in every Java object. We will discuss this functioning in lecture.

Individual Assignment

Write a Morpho program or function that does the same as the Java program above. You may write the program as a Morpho script or as a compilable program, at your convenience. Simplest is to use a channel for communication, which then works similarly to the Container above. The function makeChannel() creates a new channel. To send a value into a channel the binary operation `<-` can be used. To fetch a value from a channel, use the unary operation `<-`. These operation have the same name but are not the same operation. If a variable `c` contains a channel then the expression `c<-x` will send (write) the value `x` into that channel. The expression `x = <-c` will fetch (read) a from the channel and put it in the variable `x`.

A simple way to prevent overflow is to initialize the variable being summed into with the value `bigInteger(0)`.