第3週

- 復習：スレッド同期１（共有バッファ(size = 1)）
- 説明：スレッド同期２（共有バッファ(size > 1)）
- 実世界のアプリケーション
  - 演習３：**Mother feeds baby**
  - 課題３：ex1: **Coffee Shopのシミュレーション:**
    コーヒーを作るのと飲むのを仮想的に実現させ
    ex2: Video Shopのシミュレーション:
    ビデオのレンタルを仮想的に実現させる。
  - 発展課題：（第4週）

回転寿司のシミュレーション：寿司を握るのと食べる仮想的に実現させる。

スレッド同期の例

An example of multithreading: **Producer/Consumer**

With thread synchronization using 2 monitoring variables

Producer sets **i** → IntegerStore → **Consumer** gets **j** (i=j)

```
class Producer  extends Thread {
  private IntegerStore  pStore;
  public Producer( IntegerStore iS )
  {    pStore = iS;   }
  public void run()
  {  pStore.setMore Data(true)
    for ( int i = 0; i < 10;   i++ ) {
        pStore.setSharedInt( i);
        try { Thread.sleep(1000+ (int) ( Math.random() * 3000 ) );   }
        catch( InterruptedException e ) {; }
      }
    pStore.setMoreData( false ); }}
```

```
class Consumer  extends Thread {
  private IntegerStore cStore;
  public Consumer( IntegerStore iS)
  {     cStore = iS;   }
  public void run()
  { int val;
    while ( cStore.hasMoreData() ) {
        val = cStore.getSharedInt();
       try {  Thread.sleep( 1000+(int) ( Math.random() * 3000 ) );   }
       catch( InterruptedException e ) {; }
 } }}
```

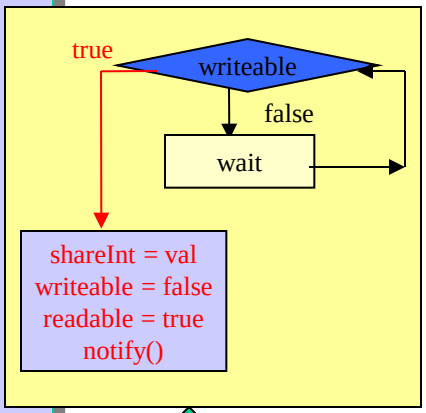
```
public class SharedStore {
  public static void main( String args[] )
  {     IntegerStore is = new IntegerStore();
        Producer p = new Producer(is );
        Consumer c = new Consumer( is );
        p.start();      c.start();   }}
```

```
class IntegerStore {
  private int sharedInt = -1;
  private boolean moreData = false;
  private boolean writeable = true;
  private bollean readable = false;
  public synchronized void setSharedInt( int val ) {
    while(!writeable) {
      try {
        wait();       }
      catch (InterruptedException e){;
      }
    }
    System.out.println( "Producer set sharedInt to " + val );
    sharedInt = val;
    writeable = false;
    readable = true;
    notify();
  }
  public synchronized int getSharedInt() {
    while (!readable){
      try{
        wait();
      }
      catch(InterruptedException e){;
      }
    }
    System.out.println( "Consumer retrieved " + sharedInt );
    writeable = true;
    readable = false;
    notify();
    return sharedInt;
  }
  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData() { return moreData; }
}
```

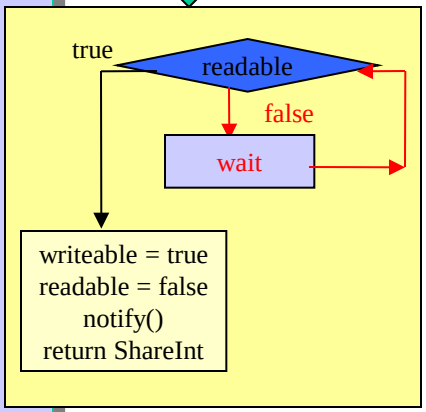
オブジェクトisを生成する時、条件変数writeableがtrueです。

Producerスレッドからの setSharedIntメソッドを呼び出す。

Consumerスレッドからの getSharedIntメソッドを呼び出す。



writeableとreadableはモニタ変数(trueとfalse)



並行スレッドが共有値のアクセス

Shared value's access by parallel threads

IntegerStoreの最大値は一つ整数です

```
class Producer  extends Thread {
    ……
  public void run()
  {
    for ( int count = 0; count < 10; count++ ) {
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      ……
      pStore.setSharedInt( count );
      ……
    }
    pStore.setMoreData( false );
  }
}
```

```
class Consumer  extends Thread {
    ……
  public void run()
  {
    while ( cStore.hasMoreData() ) {
      try {
        Thread.sleep( (int) ( Math.random() * 3000 ) );
      }
      ……
      int val = cStore.getSharedInt();
      ……
    }
  }
}
```

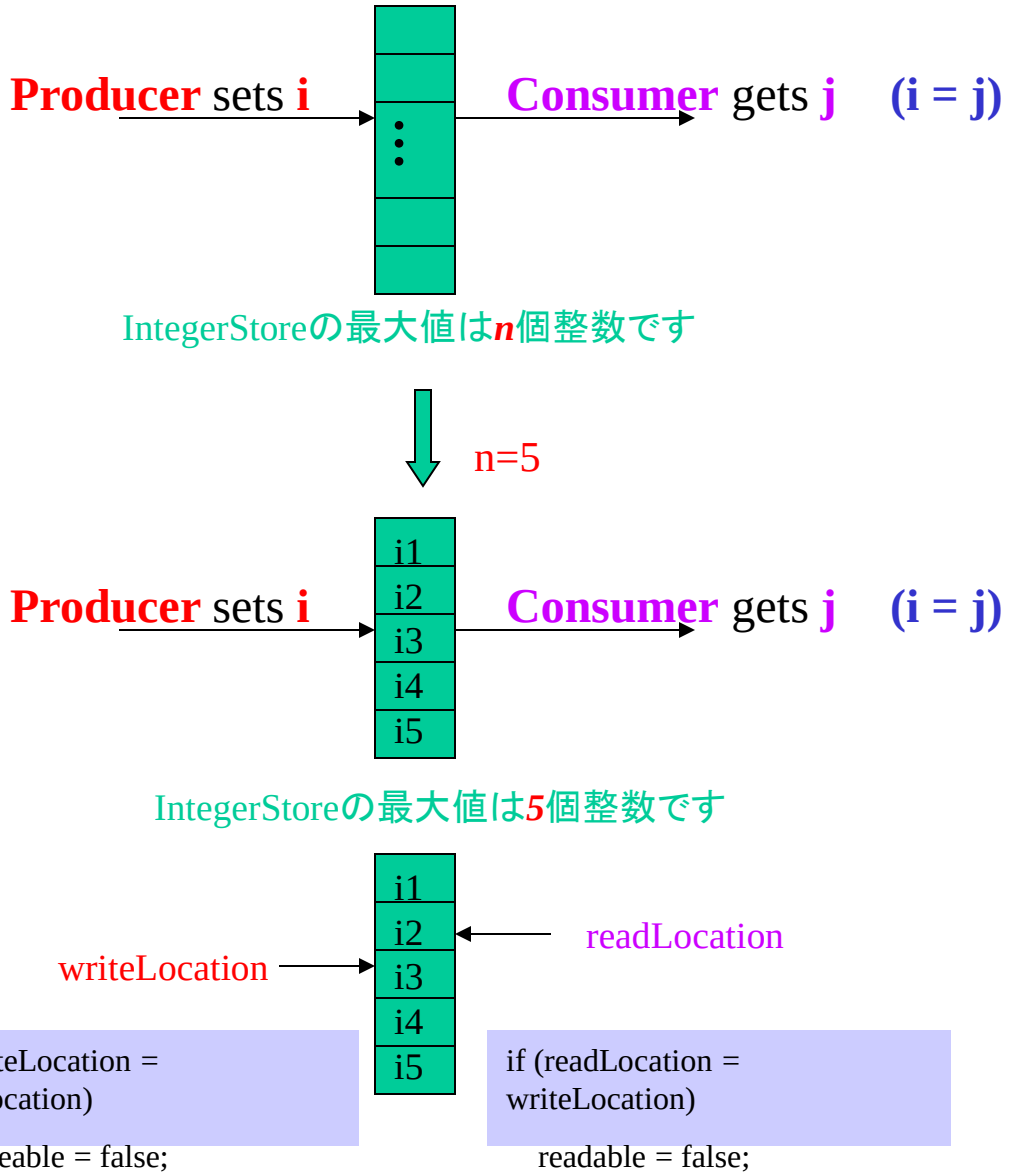
問題：　生産者の速度　≠　消費者の速度　＝＞　　パフォーマンスが悪くなる。

(速い/遅い)　　（遅い/速い）　（スレッド待っている時間が長くなるので）

解決法：　大きいバッファ（size > 1）を使うと問題を解く。

並行スレッドが共有バッファ(size > 1)のアクセス

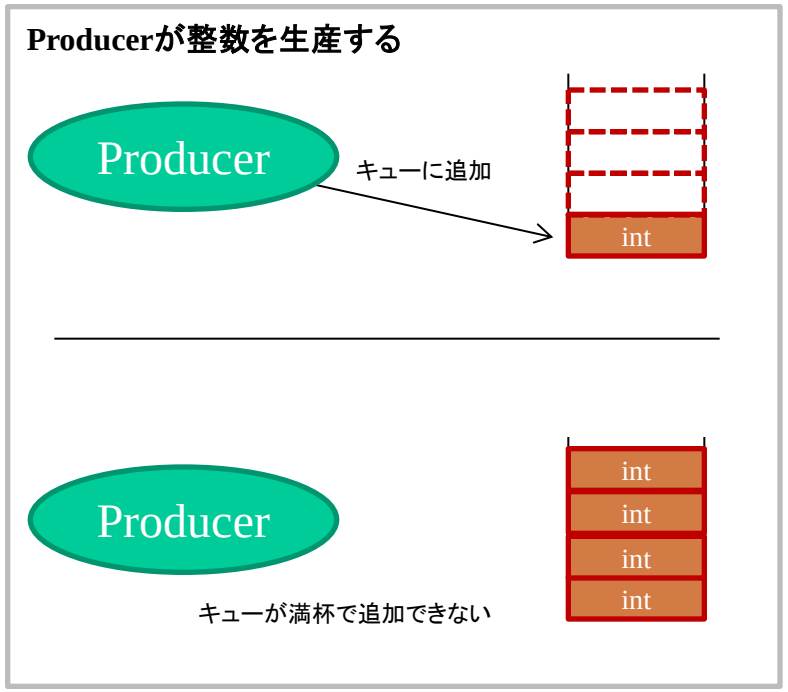
Shared buffer's access by parallel threads

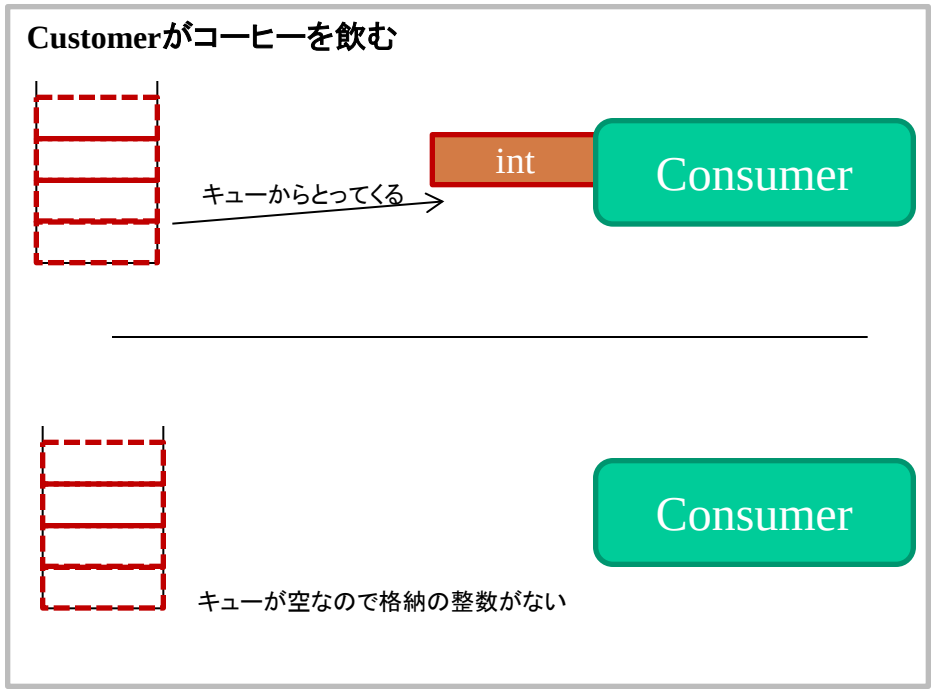


- 5個整数を格納することができるようなバッファを使います。
- もし共有バッファには空き場所があれば、生産者は共有バッファに値を書き込むことができます。
- もし共有バッファにはまだ読んでない値があれば、消費者は共有バッファから値を読み出すことができます。
- 読み出す時、書き込む同じ順番に、読み出します。そのことができるように、変数writeLocation（書き込み位置）とreadLocation（読み出し位置）を使います。
- 読み出す時、readLocation = writeLocationの場合はバッファが空っぽなので、読み出すことができないをセットします。
- 書き込む時、writeLocation = readLocationの場合はバッファが満杯なので、書き込むことができないをセットします。

もう一つの方法

- 前ページに説明したやり方でもできるが、もう1つのやり方を紹介
  - 整数を配列ではなく、キューに格納する

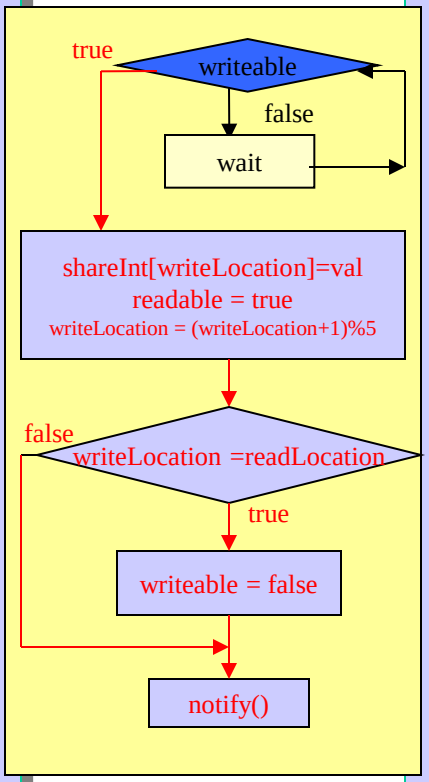




並行スレッドが共有巡回バッファ(size > 1)のアクセスの例

Source Code

```
class IntegerStore {
  private int sharedInt = -1;
  private boolean moreData = true;
  private boolean writeable = true;
  public synchronized void setSharedInt( int val ) {
    while(!writeable) {
      try {
        wait();
      }
      catch (InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    sharedInt = val;
    writeable = false;
    notify();
  }
  public synchronized int getSharedInt() {
    while (writeable){
      try{
        wait();
      }
      catch(InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    writeable = true;
    notify();
    return sharedInt;
  }
  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData() { return moreData; }
}
```




```
class IntegerStore4 {
  private int sharedInt[] = {-1, -1,-1, -1, -1};
  private boolean moreData = true;
  private boolean writeable = true;
  private boolean readable = false;
  private int readLocation = 0, writeLocation = 0;

  public synchronized void setSharedInt( int val ) {
    while(!writeable) {
      try {
        System.out.println("Producer is waiting... to set " + val);
        wait();
      }
      catch (InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    sharedInt[writeLocation++] = val;
    readable = true;
    System.out.print("Producer writes " + val + " to index "
                     + writeLocation);
    writeLocation = writeLocation % 5;
    if (writeLocation == readLocation)
    {
      writeable = false;
      System.out.println("Buffer is full");
    }
    notify();
  }
```

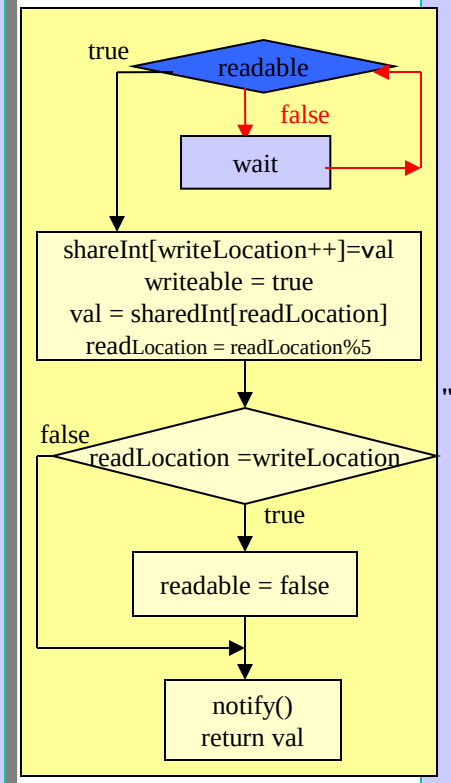
To the next page

並行スレッドが共有巡回バッファ(size > 1)のアクセスの例

Source Code (continue …)

from the previous page

```
class IntegerStore {
  private int sharedInt = -1;
  private boolean moreData = true;
  private boolean writeable = true;
  public synchronized void setSharedInt( int val ) {
    while(!writeable) {
      try {
        wait();
      }
      catch (InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    sharedInt = val;
    writeable = false;
    notify();
  }
  public synchronized int getSharedInt() {
    while (writeable){
      try{
        wait();
      }
      catch(InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    writeable = true;
    notify();
    return sharedInt;
  }
  public void setMoreData( boolean b ) { moreData =
b; }
  public boolean hasMoreData() { return moreData; }
}
```




```
public synchronized int getSharedInt() {
    int val;
    while (!readable){
      try{
        System.out.println("Consumer is waiting... to get " );
        wait();
      }
      catch(InterruptedException e){
        System.err.println("Exception: " + e.toString());
      }
    }
    writeable = true;
    val = sharedInt[readLocation++];
    System.out.print("Consumer reads " + val + " from index
"
                    + readLocation);
    readLocation = readLocation %5;
    if (readLocation == writeLocation)
    {
      readable = false;
      System.out.println("Buffer is empty");
    }
    notify();
    return val;
  }

  public void setMoreData( boolean b ) { moreData = b; }
  public boolean hasMoreData()
  {
    if (moreData == false && readLocation == writeLocation)
      return false;
    else
      return true;   }
}
```

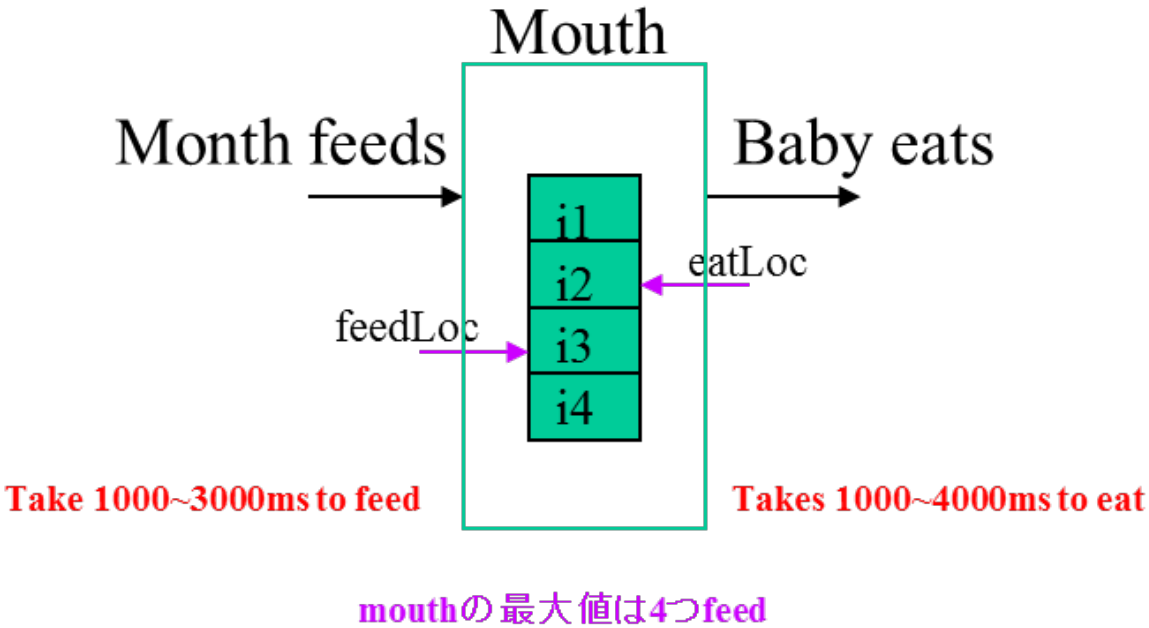
Work in Class

Run all examples （one example）
in this lecture notes

演習３

How to change it to 回転すし
（１）二人職人
（２）４人客
（３）テーブルsize = 10

MotherがBabyにa sequence of foodsを与えると仮定せよ.
Mouthの最大値は４つfeedを仮定して、i=0～9で、Babyは以下に従う。



mouthの最大値は4つfeed

課題2：Ex1の実行例

```
母親は food_0 を食べさせた
赤ちゃんは food_0 を食べた
母親は food_1 を食べさせた
赤ちゃんは food_1 を食べた
母親は food_2 を食べさせた
赤ちゃんは food_2 を食べた
母親は food_3 を食べさせた
赤ちゃんは food_3 を食べた
母親は food_4 を食べさせた
赤ちゃんは food_4 を食べた
母親は food_5 を食べさせた
赤ちゃんは food_5 を食べた
母親は food_6 を食べさせた
赤ちゃんは food_6 を食べた
母親は food_7 を食べさせた
赤ちゃんは food_7 を食べた
母親は food_8 を食べさせた
赤ちゃんは food_8 を食べた
母親は food_9 を食べさせた
赤ちゃんは food_9 を食べた
```

母親は赤ちゃんが食べ終わってから食べ物を与える

MotherスレッドとBabyスレッドが交互になっている

演習3の実行例

```
母親は 0 を食べさせた
赤ちゃんは 0 を食べだ！
食べるものは無かった
母親は 1 を食べさせた
赤ちゃんは 1 を食べだ！
食べるものは無かった
母親は 2 を食べさせた
赤ちゃんは 2 を食べだ！
食べるものは無かった
母親は 3 を食べさせた
母親は 4 を食べさせた
赤ちゃんは 3 を食べだ！
母親は 5 を食べさせた
母親は 6 を食べさせた
赤ちゃんは 4 を食べだ！
母親は 7 を食べさせた
赤ちゃんは 5 を食べだ！
母親は 8 を食べさせた
母親は 9 を食べさせた
赤ちゃんの口はいっぱいだ！
赤ちゃんは 6 を食べだ！
母親は 10 を食べさせた
赤ちゃんの口はいっぱいだ！
赤ちゃんは 7 を食べだ！
母親は 11 を食べさせた
赤ちゃんの口はいっぱいだ！
赤ちゃんは 8 を食べだ！
母親は 12 を食べさせた
赤ちゃんの口はいっぱいだ！
赤ちゃんは 9 を食べだ！
母親は 13 を食べさせた
赤ちゃんの口はいっぱいだ！
赤ちゃんは 10 を食べだ！
赤ちゃんは 11 を食べだ！
赤ちゃんは 12 を食べだ！
赤ちゃんは 13 を食べだ！
食べるものは無かった
```

赤ちゃんは4つまで口に入る。
4つ以上食べ物を入れると口がいっぱいで入らない。

口がいっぱいなので、食べ終わるのを待つ。
（Motherスレッドはwait状態になる）

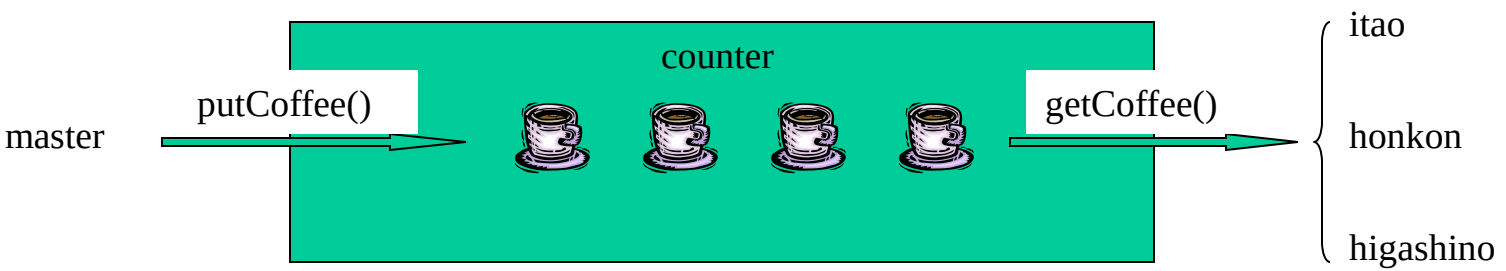
母親が与えた食べ物を全て食べ終わったので終了

課題3：select one from Ex1 and Ex2

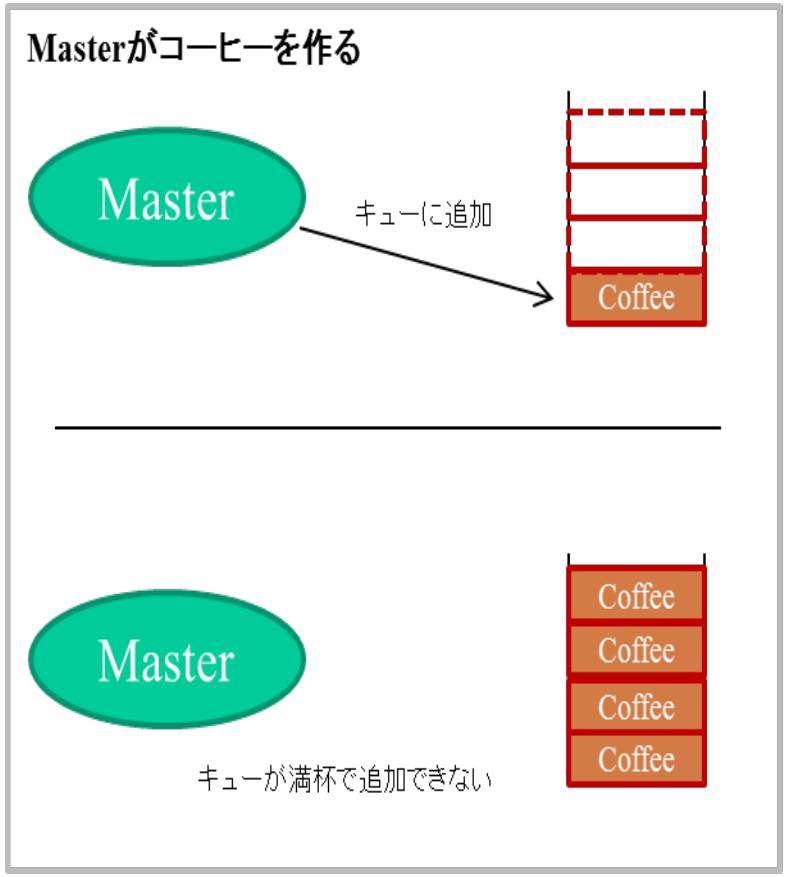
Ex1：CoffeeShop

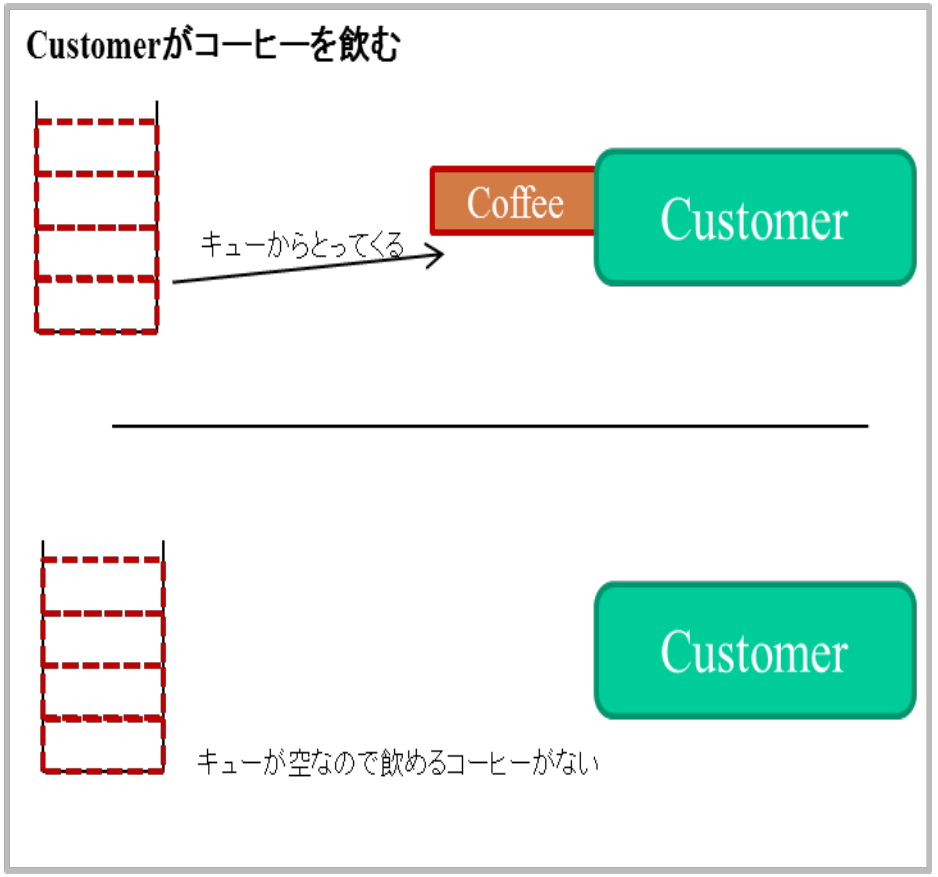
class CoffeeShop

class ShopMaster **class Counter** **class CoffeeDrinker**



コーヒーを配列ではなく、キューに格納する





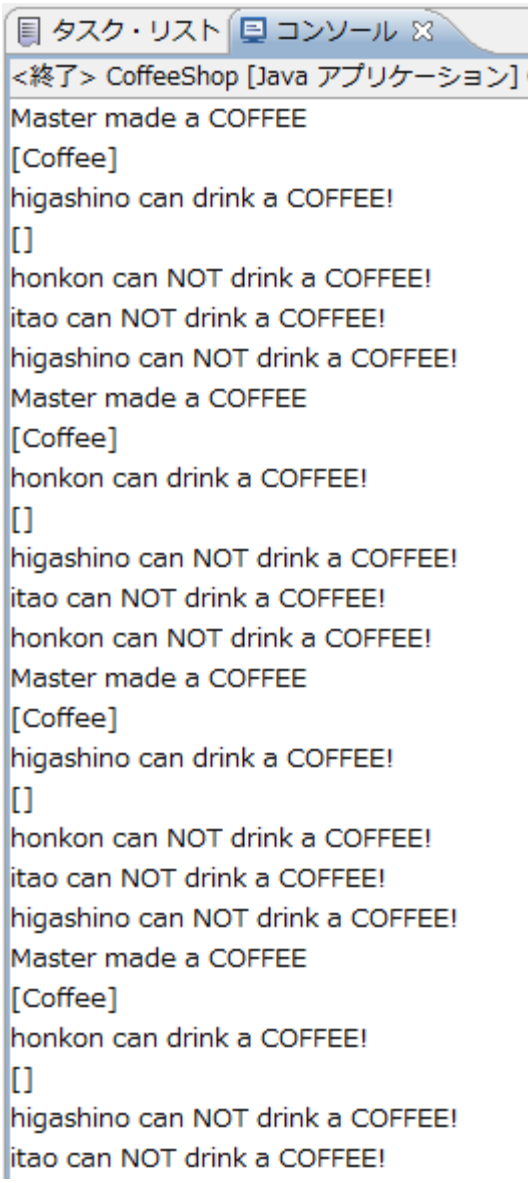
Queueの扱い方はhttp://java.sun.com/j2se/1.5.0/ja/docs/ja/api/java/util/Queue.html
を参考にしてください。

擬似コード（Counterクラス)

```
class Counter {

  Queue<String> coffees = new LinkedList<String>();

  public synchronized void getCoffee(String name) throws InterruptedException{     //コーヒーを飲む
    while(コーヒーがない){
       客はコーヒー飲めない
    }
    客はコーヒーを飲んだよ
      ・・・

    if(もしコーヒーが4つ作られていたら)
      待っているお客さん全員が飲める→ということは？
  }


   public synchronized void putCoffee() throws InterruptedException{     //コーヒーをカウンターに置く
     while(コーヒーが4つ以上作られていたら){
       赤字になってしまう。
     }
     コーヒーを作る

    if(マスターがコーヒー作った（coffees.size )
      待っていた客がコーヒー飲めるようになる。
   }
}
```

Queueの扱い方はhttp://java.sun.com/j2se/1.5.0/ja/docs/ja/api/java/util/Queue.htmlを参考にしてください。

CoffeShop 実行例



Master made a COFFEE
[Coffee]
higashino can drink a COFFEE!
[]
itao can NOT drink a COFFEE!
Master made a COFFEE
[Coffee]
itao can drink a COFFEE!
[]
higashino can NOT drink a COFFEE!
Master made a COFFEE
[Coffee]
higashino can drink a COFFEE!
[]
higashino can NOT drink a COFFEE!
Master made a COFFEE
[Coffee]
higashino can drink a COFFEE!
[]

課題3：select one from Ex1 and Ex2

Ex2：VideoShop

class VideoShop

class Rack

class User

A thread class

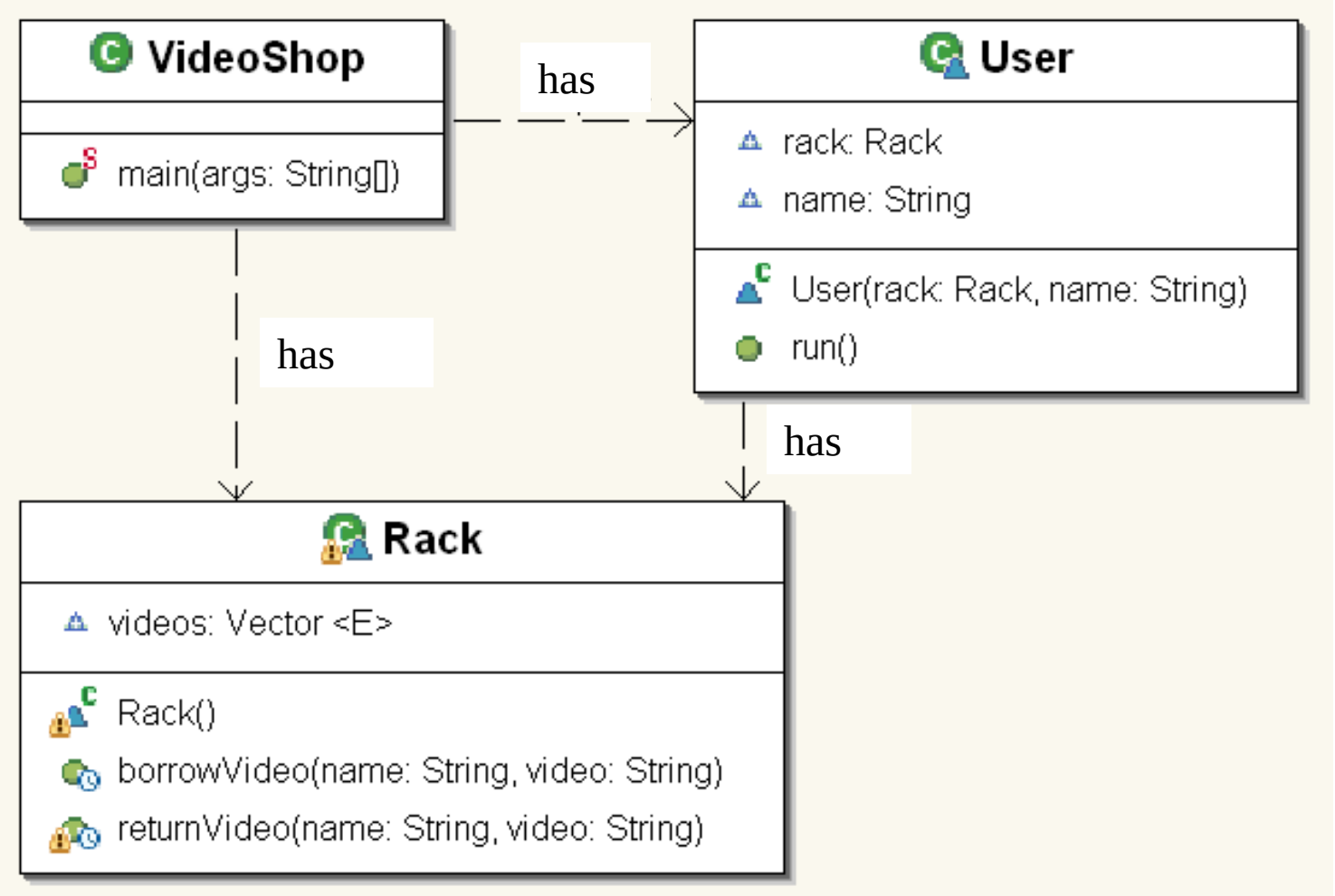
- itao
- honkon
- higashino

Hints:

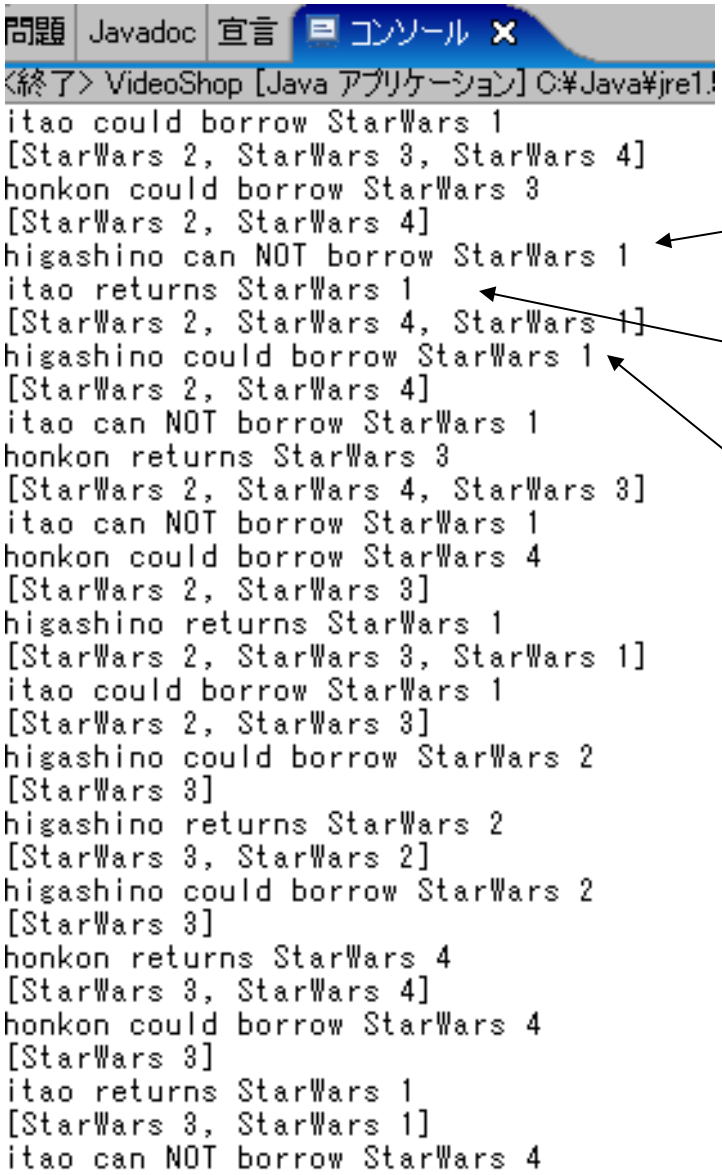
- プログラム「CoffeeShop」を改良して、プログラム「VideoShop」を作る。
- プログラム「VideoShop」は、ビデオのレンタルを仮想的に実現させる。

- プログラム「VideoShop」は以下の3つのクラスで成り立つ:
  - VideoShop
    - mainメソッドを持ち、RackとUserを生成後、Userをスタートさせる。
  - Rack
    - Vectorとしてビデオ郡を保持する。コンストラクタで、適当にビデオを登録しておく。「借りる」メソッドと「返す」メソッドを提供する。
  - User
    - Threadとして稼動する。乱数によって借りるビデオを決め、Rackから「借りる」。乱数によって決まった時間分Sleepし、Rackにビデオを「返す」。これを繰り返す。

Class Diagram



VideoShop 実行例:



板尾が「スターウォーズ１」を借りているため、東野は借りれない（waitする）。

板尾が「スターウォーズ１」を返した(notifyAllしてwaitを解除させる)。

東野は無事、「スターウォーズ１」を借りることができた。

VideoShopEx.java

発展問題

回転寿司