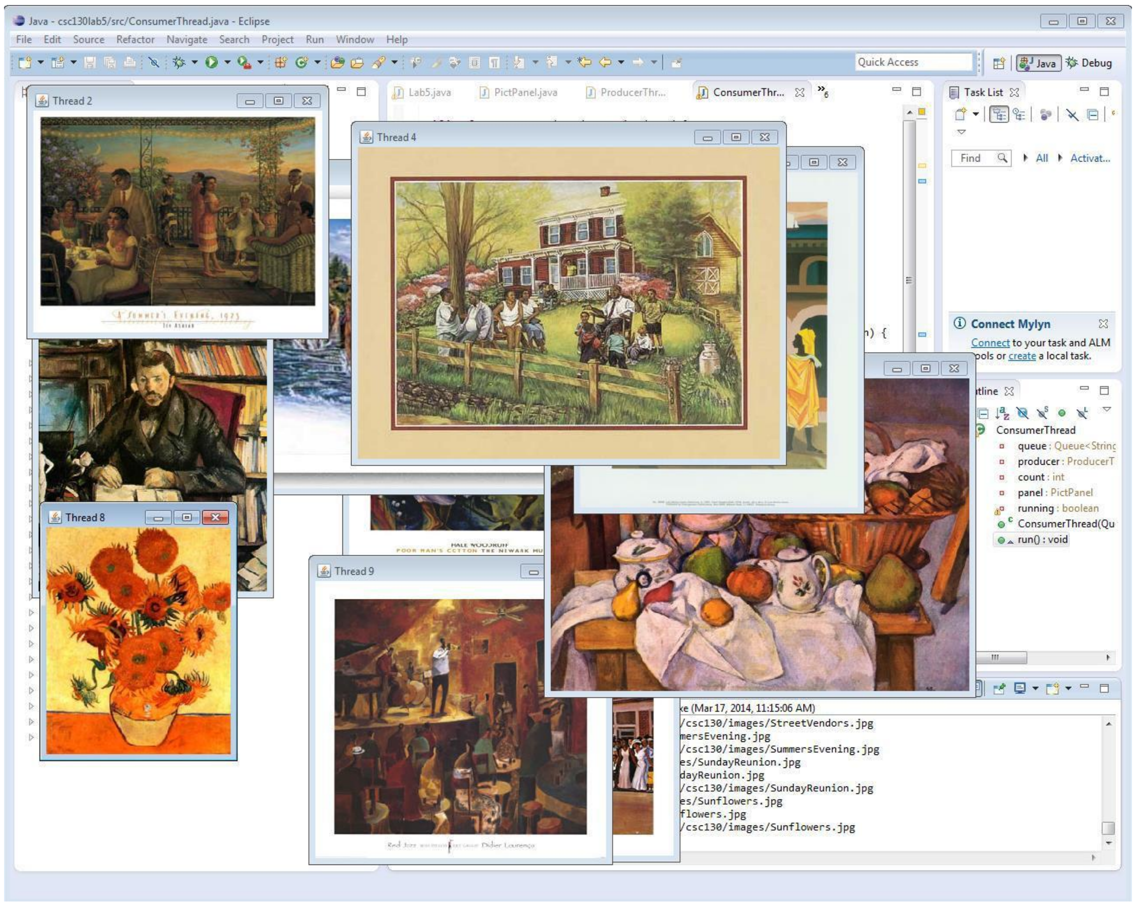
This lab is a simplified version of the producer-consumer model. The producer will place (push) the names of images into a queue; several consumers will retrieve (pop) the names and display the images in frames for a random amount of time.

Threads will be used to simulate the simultaneous access to the queue. The producer will run as a thread as well as the consumers. As the producer places items into the queue, the queue can become full. Therefore, the producer should wait until that condition changes before continuing. As the consumers remove items from the queue, an isEmpty condition can occur. Therefore, the consumers should also wait until that condition changes. The variable **numThreads,** in the application class, will store the number of consumers and the corresponding number of frames. Most of the code is prewritten. You must add the appropriate code where specified by comments with TODO. You should read through this entire document and understand what should take place after each step in the programs prior to writing your code.

The focus is not on the producer-consumer model and the synchronization issues that can occur (see Wikipedia page - https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem), but on the use of a queue to simulate such problems.

For more information on Threads, see the Wikipedia page, https://en.wikipedia.org/wiki/Thread_(computing) and the document included on Blackboard.



```
e (Mar 17, 2014, 11:15:06 AM)
/csc130/images/StreetVendors.jpg
mersEvening.jpg
/csc130/images/SummersEvening.jpg
es/SundayReunion.jpg
dayReunion.jpg
/csc130/images/SundayReunion.jpg
es/Sunflowers.jpg
flowers.jpg
/csc130/images/Sunflowers.jpg
```

The lab consists of a producer thread that places the names of images into a ***Queue<String>***.
Consumer threads, defined in an array and stored in a **ThreadGroup**, remove the images from the Queue and display them in frames that are associated with each thread, for a random amount of time.

The ***PictPanel*** class is complete, but you must modify the ***ProducerThread***, ***ConsumerThread*** and the ***Application*** class[1]. Use the Queue and Exception classes, but add the ***synchronized*** modifier to the all methods in the Queue class, except the constructor:

enqueue, dequeue, front, rear, isEmpty, isFull, getSize, toString

As stated, you must document your programs.

Lab5bApp
-numThreads : int
-queue : Queue
-tg : ThreadGroup
-consumer : Thread[]
-producer : ProducerThread
-panel : PictPanel[]
-pictureFrame : JFrame[]
-picture : String[]
+Lab5App()
+static void main(String[])
+void loadImages()
+void loadPictures()

ProducerThread
-queue : Queue
-picture : String[]
+ProducerThread(Queue, String[])
+void run()

$$\begin{tikzcd}\text{Thread} \\ \text{ProducerThread} \arrow[u]\end{tikzcd}$$

ConsumerThread
-queue : Queue
-producer : ProducerThread
-count : int
-panel : PictPanel
+ConsumerThread(Queue, ProducerThread, PictPanel)
+ void run()

$$\begin{tikzcd}\text{Thread} \\ \text{ConsumerThread} \arrow[u]\end{tikzcd}$$

PictPanel
-pict : Image
-parent : JFrame
+PictPanel(JFrame)
+void paintComponent(Graphics)
+synchronized void drawPict(String)

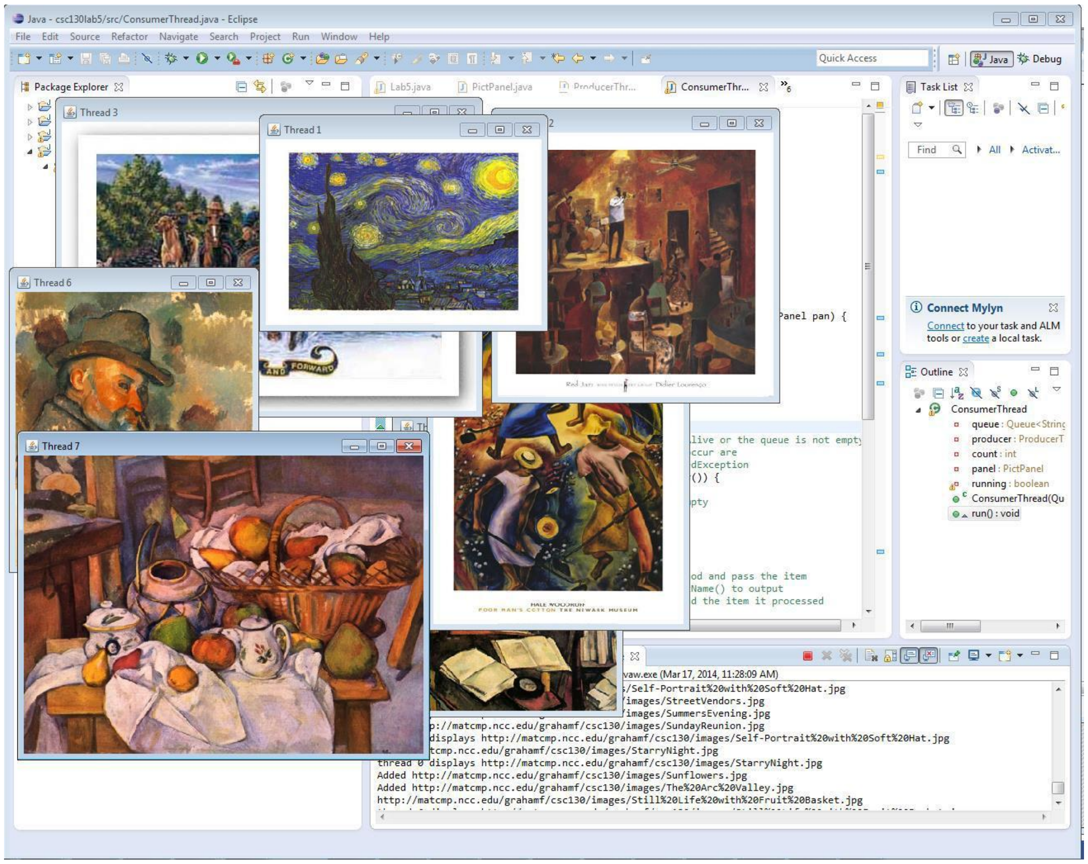
$$\begin{tikzcd}\text{JPanel} \\ \text{PictPanel} \arrow[u]\end{tikzcd}$$

****
You can download free-to-use images from the Internet and place them in an images folder.
You can obtain pictures from: http://www.pics4learning.com *http://www.bigfoto.com*

You will have to copy and use the JSoup jar files (*jsoup-1.8.1.jar, jsoup-1.8.1-javadoc.jar, jsoup-1.8.1-sources.jar*). Add them as external jar files (see the documentation file on Blackboard).

[1] **Note**: add code wherever there is a TODO message in the program

Example: Program running with 7 consumer threads.


Java - csc130lab5/src/ConsumerThread.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Package Explorer
Lab5.java
PictPanel.java
ProducerThr...
ConsumerThr...
Task List
Find
All
Activat...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
Thread 3
Thread 1
Thread 6
Thread 7

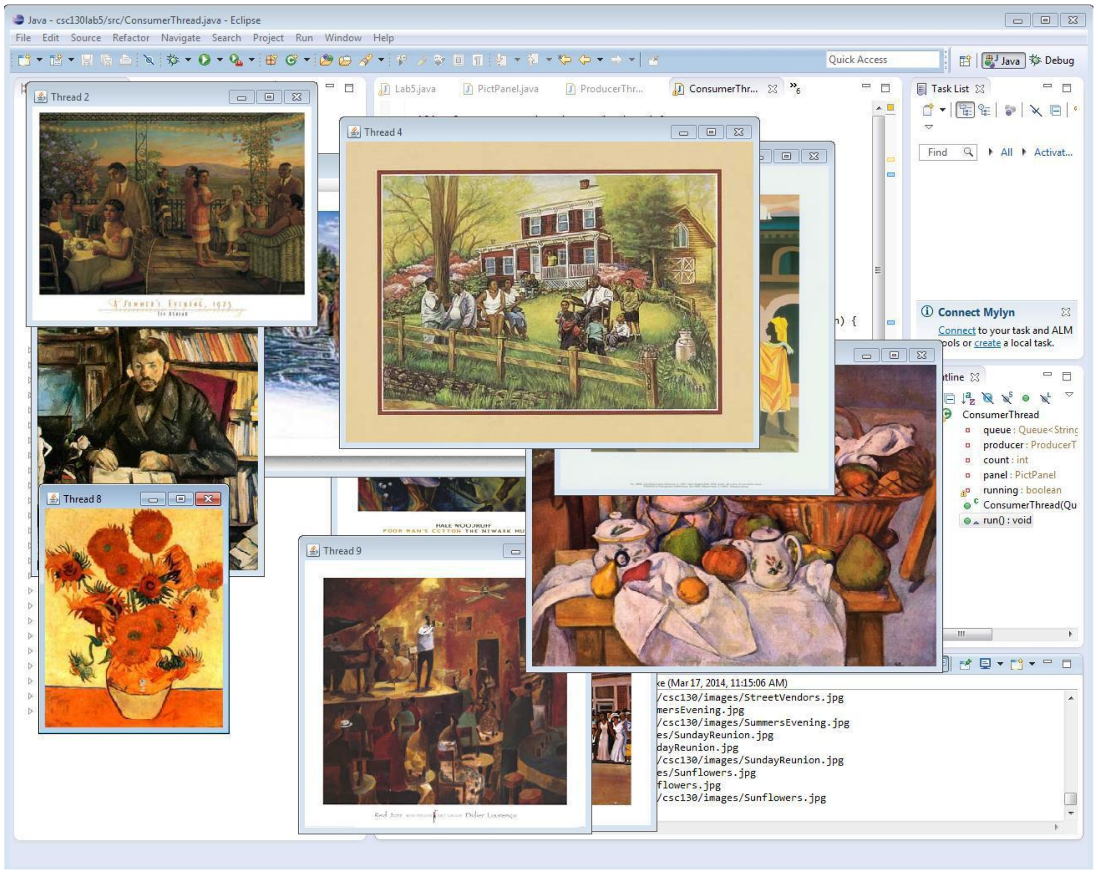
```
Panel pan) {
```

```
live or the queue is not empty
ccur are
dException
()) {

pty


od and pass the item
Name() to output
d the item it processed
```

ConsumerThread
- queue : Queue<String
- producer : ProducerT
- count : int
- panel : PictPanel
- running : boolean
- ConsumerThread(Qu
- run() : void

```
vaw.exe (Mar 17, 2014, 11:28:09 AM)
s/Self-Portrait%20with%20Soft%20Hat.jpg
/images/StreetVendors.jpg
/images/SummersEvening.jpg
p://matcmp.ncc.edu/grahamf/csc130/images/SundayReunion.jpg
displays http://matcmp.ncc.edu/grahamf/csc130/images/Self-Portrait%20with%20Soft%20Hat.jpg
tcmp.ncc.edu/grahamf/csc130/images/StarryNight.jpg
thread 0 displays http://matcmp.ncc.edu/grahamf/csc130/images/StarryNight.jpg
Added http://matcmp.ncc.edu/grahamf/csc130/images/Sunflowers.jpg
Added http://matcmp.ncc.edu/grahamf/csc130/images/The%20Arc%20Valley.jpg
http://matcmp.ncc.edu/grahamf/csc130/images/Still%20Life%20with%20Fruit%20Basket.jpg
```

Example: Program running with 10 consumer threads.


Java - csc130lab5/src/ConsumerThread.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
Lab5.java
PictPanel.java
ProducerThr...
ConsumerThr...
Task List
Find
All
Activat...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
Thread 2
Thread 4
Thread 8
Thread 9

ConsumerThread
- queue : Queue<String
- producer : ProducerT
- count : int
- panel : PictPanel
- running : boolean
- ConsumerThread(Qu
- run() : void

```
e (Mar 17, 2014, 11:15:06 AM)
/csc130/images/StreetVendors.jpg
mersEvening.jpg
/csc130/images/SummersEvening.jpg
es/SundayReunion.jpg
dayReunion.jpg
/csc130/images/SundayReunion.jpg
es/Sunflowers.jpg
flowers.jpg
/csc130/images/Sunflowers.jpg
```

```java
import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

import java.net.*;
import java.applet.*;
import java.io.*;

import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;

public class Lab5bApp {

   // TODO: define an integer to store the number of consumer threads
   // variable name - numThreads

   // TODO: define a reference to a Queue<String> (use the default constructor)
   // variable name - queue


   // define a reference to a ThreadGroup
   private ThreadGroup tg = new ThreadGroup("consumers");

   // TODO: define a reference to an array of Threads (data type isThread)
   // variable name - consumer

   // TODO: define a reference to a ProducerThread
   // variable name - producer


   // TODO: define a reference to an array of PictPanel objects (to display each image)
   // variable name - panel

   // TODO: define a reference to an array of JFrame objects (to place each Panel)
   // variable name - pictureFrame

   // TODO: define a reference to an array of Strings (to store the names of the images)
   // variable name - picture


   public static void main(String[] args) {

      // TODO: create an instance of this class and call its constructor

      // exit the program
      System.exit(0);
   }
```

```
// constructor for the class
public Lab5bApp() {
   // TODO: permit the user to input the number of consumer threads (3 - 5)
   // Use JOptionPane.showInputDialog


   // TODO: initialize all arrays to the size of the number of consumer threads


   // TODO: call the appropriate method to load the pictures


   // TODO: create a loop based on the number of consumer threads
   //    for each JFrame and PictPanel object
   {
      // TODO: create an instance of JFrame with the title Thread and the number
         // of threads, example: (Thread 1) and store it in the array

      // TODO: use the setSize method to set the JFrame's size to 200 by 200

      // Set X,Y coordinate of upper left corner of the frame
      pictureFrame[i].setLocation(80 * i, 80 * i);

      // TODO: create an instance of PictPanel and store it in the panel
      // array

      // add the panel to the frame
      pictureFrame[i].getContentPane().add(panel[i]);

      // Use the setVisible method to show the
      frame pictureFrame[i].setVisible(true);
   }

   // TODO: create an instance of ProducerThread
   // store it in the variable created
   // the ProducerThread constructor needs a Queue and an array of Strings


   // TODO: call the start method of the producer thread


   // producer.setPriority(Thread.MAX_PRIORITY);

   // create the consumer threads and add them to the ThreadGroup
   for (int i = 0; i < numThreads; i++) {
      consumer[i] = new Thread(tg, new ConsumerThread(queue,
            producer, panel[i]), "thread " + i);
      consumer[i].start();
   }

   // TODO: loop until the producer thread is no longer alive
   // use the isAlive() method of the producer thread
   // the loop pauses the program until the producer thread ends
```

```
      // TODO: after the loop ends, display a message using ShowMessageDialog
      // indicating  the producer thread has finished



      // TODO: pause the program until the consumer threads end
      // Loop while ThreadGroup's activeCount > 0


      // TODO: after the loop ends, display a message using ShowMessageDialog
      // indicating that the consumer threads have finished


      // TODO: display an end of program message using ShowMessageDialog

   }

   /**
    *  Method loads the pictures from URL
    */
   public void loadImages() {
     Document doc;
     String url = "http://venus.cs.qc.cuny.edu/~aabreu/cs212/images/";
     int j = 0;

      try {
         System.out.println("Reading URL directory");
         doc = Jsoup.connect(url).get();
         Elements files = doc.getElementsByTag("a");

         // initialize the array size for the file names
         picture = new String[files.size()];

         for (Element file : files){
            String filename = file.attr("href");
            if(filename.indexOf(".jpg") >= 0
               || filename.indexOf(".gif") >= 0){
               System.out.println(filename);
               picture[j++] = url + filename;
            }
         }
      } catch (IOException ioe){
              ioe.printStackTrace();
       }
   }
```

```
    /**
     *  Method loads the pictures from a local folder
     */
    public void loadPictures() {
       // Ceate a reference to an array of Files (type is File)
      File[] pictures;

      // create a reference String for the image directory
       String directory = "images/";

       // create a reference to the directory
       File imgDirectory = new File(directory);

       // store the directory listing in the File array
       pictures = imgDirectory.listFiles();

       // initialize the array size for the file names
       picture = new String[pictures.length];

       // add each image file to the image file array
       for (int i = 0, j = 0; i < pictures.length; i++) {
          File file = pictures[i];

          // add .jpg and .gif file names to the String array
          // you can add other image types
          if (file.isFile()
                && (file.getName().toLowerCase().endsWith(".jpg") ||
                   file.getName().toLowerCase().endsWith(".gif"))) {
             // display the name of the image file
             System.out.println((j + 1) + ": " + pictures[i].getName());

             // add the name to the array
             picture[j] = pictures[i].getName();

             // increment the number of images
             j++;
          }
       }
    }
}
```

```
import javax.swing.*;
import java.awt.*;
import java.net.*;

public class PictPanel extends JPanel {

   private static final long serialVersionUID = 1L;
   private Image pict;
   private JFrame parent;

   public PictPanel(JFrame frame) {
      parent = frame;
      try {

         ImageIcon imgIcon = new ImageIcon(new URL("http://www.google.com/images/logo.gif"));
         pict = imgIcon.getImage();
         parent.setVisible(false);
         this.setSize(pict.getWidth(this), pict.getHeight(this));
         parent.setSize(pict.getWidth(this), pict.getHeight(this));
         parent.setVisible(true);
         repaint();
      }
      catch(java.net.MalformedURLException e) {}
   }

    public void paintComponent(Graphics g)  {
      super.paintComponent(g);
      if(pict != null)
         g.drawImage(pict,0,0,getWidth(), getHeight(),this);
    }

    public synchronized void drawPict(String fileName) {
      try{
         System.out.println(fileName);

         // FileName is a local file
         //ImageIcon imgIcon = new ImageIcon(this.getClass().getResource(fileName));

         // FileName is an URL
         ImageIcon imgIcon = new ImageIcon(new URL(fileName));

         pict = imgIcon.getImage();
         if(pict != null) {
            MediaTracker imageTracker = new MediaTracker(this);
            imageTracker.addImage(pict, 0);

            try {
                    imageTracker.waitForAll();
            }
            catch(InterruptedException e){ }

            parent.setVisible(false);
            this.setSize(pict.getWidth(this), pict.getHeight(this));
            parent.setSize(pict.getWidth(this), pict.getHeight(this));

            parent.setVisible(true);
            repaint();
         }
      }catch(MalformedURLException mue){
         System.out.println("Oh oh, something bad happened");
      }
    }
}
```

```java
public class ProducerThread extends Thread {

   private Queue<String> queue;
   private String[] picture;

   // TODO: constructor
   //    initialize the queue and the array
   //    using the 2 parameters


   public void run() {
      // TODO loop for each picture and the picture is not null
      // Note: possible exceptions that can occur are
      //    QueueFullException and InterruptedException
      {
            // TODO loop while the queue is full


            // TODO add an item from the array to the queue
            //   display the item added to the queue
            //   sleep for 100 - 1000 milliseconds

      }
      // TODO When the thread stop's running,
      // display a message indicating the thread has stopped
   }
}
```

```
public class ConsumerThread extends Thread {

   private Queue<String> queue;
   private ProducerThread producer;
   private int count;
   private PictPanel panel;
   private boolean running;

   // TODO: constructor
   //    initialize the queue, producer thread, and the panel
   //    using the 3 parameters


   public void run() {
      // TODO loop while the producer thread isAlive or the queue is not empty
      // Note: possible exceptions that can occur are
      //    QueueEmptyException and InterruptedException
      {
            // TODO loop while the queue is empty


            // TODO if the queue is not empty
            {
               // take an item from the queue
               // call the panel's drawPict method and pass the item
               // Use Thread.currentThread().getName() to output
               //    the name of this thread and the item it processed
               // add 1 to the count
               // sleep for 1000 to 5000 milliseconds (random)
            }
      }
      // TODO When the thread stop's running,
      //    display its name and the number of items processed


   }

}
```