tick3 submission from David Brazdil

Name	David Brazdil (db538)
College	TRINH
Submission contents	uk/ac/cam/db538/fjava/tick3/MessageQueue.java uk/ac/cam/db538/fjava/tick3/ProducerConsumer.java uk/ac/cam/db538/fjava/tick3/UnsafeMessageQueue.java uk/ac/cam/db538/fjava/tick3/SafeMessageQueue.java uk/ac/cam/db538/fjava/tick3/QueueTest.java uk/ac/cam/db538/fjava/tick3/BankSimulator.java
Ticker	Not yet assigned
Ticker signature	

MessageQueue.java

```
 0   package uk.ac.cam.db538.fjava.tick3;
 1
 2   public interface MessageQueue<T> {
 3   public void put(T msg);
 4   public T take();
 5   }
```

ProducerConsumer.java

```
 0   package uk.ac.cam.db538.fjava.tick3;
 1
 2   public class ProducerConsumer {
 3   private MessageQueue<Character> m = new UnsafeMessageQueue<Character>();
 4   private class Producer implements Runnable {
 5   char[] cl = "Computer Laboratory".toCharArray();
 6
 7   public void run() {
 8   for (int i = 0; i < cl.length; i++) {
 9   m.put(cl[i]);
10   try {Thread.sleep(500);} catch (InterruptedException e) {
11   e.printStackTrace();
12   }
13   }
14   }
15   }
16
17   private class Consumer implements Runnable {
18   public void run() {
19   while (true) {
20   System.out.print(m.take());
21   System.out.flush();
22   }
23   }
24   }
25
26   void execute() {
27   new Thread(new Producer()).start();
28   new Thread(new Consumer()).start();
29   }
30
31   public static void main(String[] args) {
32   new ProducerConsumer().execute();
33   }
34   }
```

UnsafeMessageQueue.java

```
 0  package uk.ac.cam.db538.fjava.tick3;
 1
 2  public class UnsafeMessageQueue<T> implements MessageQueue<T> {
 3  private static class Link<L> {
 4  L val;
 5  Link<L> next;
 6  Link(L val) { this.val = val; this.next = null; }
 7  }
 8
 9  private Link<T> first = null;
10  private Link<T> last = null;
11
12  public void put(T val) {
13  Link<T> newLink = new Link<T>(val);
14  if (last != null)
15  last.next = newLink;
16  last = newLink;
17  if (first == null)
18  first = newLink;
19  }
20
21  public T take() {
22  while(first == null) //use a loop to block thread until data is available
23  try {Thread.sleep(100);} catch(InterruptedException ie) {}
24  Link<T> firstLink = first;
25  first = firstLink.next;
26  return firstLink.val;
27  }
28  }
```

SafeMessageQueue.java

```
 0  package uk.ac.cam.db538.fjava.tick3;
 1
 2  public class SafeMessageQueue<T> implements MessageQueue<T> {
 3  private static class Link<L> {
 4  L val;
 5  Link<L> next;
 6  Link(L val) { this.val = val; this.next = null; }
 7  }
 8
 9  private Link<T> first = null;
10  private Link<T> last = null;
11
12  public synchronized void put(T val) {
13  Link<T> newLink = new Link<T>(val);
14  if (last != null)
15  last.next = newLink;
16  last = newLink;
17  if (first == null)
18  first = newLink;
19  this.notify();
20  }
21
22  public synchronized T take() {
23  while(first == null) //use a loop to block thread until data is available
24  try { this.wait(); } catch(InterruptedException ie) {}
25  Link<T> firstLink = first;
26  first = firstLink.next;
27  return firstLink.val;
28  }
29  }
```

QueueTest.java

```
 0  package uk.ac.cam.db538.fjava.tick3;
 1
 2  public class QueueTest {
 3
 4  private class Producer extends Thread {
 5  private int sent = 0;
 6  public void run() {
 7  for (int i = 0; i < 50000; ++i) {
 8  q.put("" + i);
 9  sent++;
10  }
11  }
12  public int numberProduced() {return sent;}
13  }
14
15  private class Consumer extends Thread {
16  private int recv = 0;
17  public void run() {
18  while (!q.take().equals("EOF")) {
19  recv++;
20  }
21  q.put("EOF");
22  }
23  public int numberConsumed() {return recv;}
24  }
25
26  private MessageQueue<String> q;
27  private Consumer[] consumers;
28  private Producer[] producers;
29
30  QueueTest(MessageQueue<String> q, int c, int p) {
31  this.q = q;
32  consumers = new Consumer[c];
33  for (int i = 0; i < c; ++i)
34  consumers[i] = new Consumer();
35  producers = new Producer[p];
36  for (int i = 0; i < p; ++i)
37  producers[i] = new Producer();
38  }
39
40  public void run() {
41
42  for (Consumer c : consumers) c.start();
43  for (Producer p : producers) p.start();
44  for (Producer p : producers) try {p.join();} catch (InterruptedException e) {}
45
46  q.put("EOF");
47  //terminate join at 10 secs since EOF marker may get lost
48  for (Consumer c : consumers) try {c.join(10000);} catch (InterruptedException e) {}
49
50  int recv = 0;
51  for (Consumer consumer : consumers) recv += consumer.numberConsumed();
52  int sent = 0;
53  for (Producer p : producers) sent += p.numberProduced();
54  System.out.println(recv + " / " + sent);
55  }
56
57  public static void main(String[] args) {
58  System.out.println("** UNSAFE ** ");
59  new QueueTest(new UnsafeMessageQueue<String>(), 1, 1).run();
60  new QueueTest(new UnsafeMessageQueue<String>(), 3, 1).run();
61  new QueueTest(new UnsafeMessageQueue<String>(), 1, 3).run();
62  new QueueTest(new UnsafeMessageQueue<String>(), 3, 3).run();
63
64  System.out.println("** SAFE ** ");
65  new QueueTest(new SafeMessageQueue<String>(), 1, 1).run();
66  new QueueTest(new SafeMessageQueue<String>(), 3, 1).run();
67  new QueueTest(new SafeMessageQueue<String>(), 1, 3).run();
68  new QueueTest(new SafeMessageQueue<String>(), 3, 3).run();
69  }
70  }
```

BankSimulator.java

```
 0  package uk.ac.cam.db538.fjava.tick3;
 1
 2  import java.util.Random;
 3
 4  public class BankSimulator {
 5
 6  private class BankAccount {
 7  private int balance;
 8  private int acc;
 9
10  BankAccount(int accountNumber, int deposit) {
11  balance = deposit;
12  acc = accountNumber;
13  }
14
15  public int getAccountNumber() {
16  return acc;
17  }
18
19  public void transferTo(BankAccount b, int amount) {
20  BankAccount lock1 = null, lock2 = null;
21  if (this.acc < b.acc) {
22  lock1 = this;
23  lock2 = b;
24  } else {
25  lock2 = this;
26  lock1 = b;
27  }
28
29  synchronized(lock1) {
30  synchronized(lock2) {
31  balance -= amount;
32  b.balance += amount;
33  }
34  }
35  }
36  }
37
38  private static Random r = new Random();
39  private class RoboTeller extends Thread {
40  public void run() {
41  //Robots work from 9am until 5pm; one customer per second
42  for(int i=9*60*60; i<17*60*60; i++) {
43  int a = r.nextInt(account.length);
44  int b = r.nextInt(account.length);
45  account[a].transferTo(account[b], r.nextInt(100));
46  }
47  }
48  }
49
50  private int capital;
51  private BankAccount[] account;
52  private RoboTeller[] teller;
53
54  public BankSimulator(int capital, int accounts, int tellers) {
55      this.capital = capital;
56      this.account = new BankAccount[accounts];
57      this.teller = new RoboTeller[tellers];
58  for(int i=0; i<account.length; i++)
59  account[i]  = new BankAccount(i,capital/account.length);
60      }
61
62  public int getCapital() {return capital;}
63
64  public void runDay() {
65  for(int i=0; i<teller.length; i++)
66  teller[i] = new RoboTeller();
67  for(int i=0; i<teller.length; i++)
68  teller[i].start();
69
70  int done = 0;
71  while(done < teller.length)
```

```
72   try{teller[done].join();done++;} catch(InterruptedException e) {}
73
74   int finalCapital = 0;
75   for(int i=0; i<account.length; i++)
76   finalCapital += account[i].balance;
77   capital = finalCapital;
78   }
79
80   public static void main(String[] args) {
81   BankSimulator javaBank = new BankSimulator(10000,10,100);
82   javaBank.runDay();
83   System.out.println("Capital at close: ??"+javaBank.getCapital());
84   }
85   }
```