```
 1 /*
 2 Prof.java
 3     - class for DinersAndThinkers
 4     - recommended size: ~90 lines
 5
 6 Cycles through 4 steps until goals met:
 7     - Sleep:
 8         - print message (indication)
 9         - sleep for random time between [0, 100)
10     - Program:
11         - print message (indication, # lines, total # lines written)
12         - writes random number of lines [5, 20], 1 millisecond sleep per line
13     - Hungry:
14         - print message (indication)
15     - Eat:
16         - REQUIRES:
17             - (1) fork
18             - (1) bib
19         - attempts to obtain bib and fork IN RANDOM ORDER
20         - once bib & fork obtained:
21             - print message (indication, # noodles, total # noodles eaten)
22             - eats random number of noodles [2, 10], 1 millisecond per noodle
23             - once done, places bib and fork back in RANDOM ORDER
24
25 Completes once the following is satisfied:
26     - 85 noodles eaten
27     - 200 lines written
28
29 prints indication of completion, starts with "==============>"
30
31 Authors: Philippe Nadon, Jack Shea
32  */
33
34 package dinersandthinkers;
35
36 import java.util.Random;
37
38 public class Prof extends Thread{
39     private static final int NUM_NOODLES_TO_EAT = 85;
40     private static final int NUM_LINES_TO_WRITE = 200;
41     private int noodlesEaten;
42     private int linesWritten;
43     private Basket BibBasket;
44     private Basket ForkBasket;
45
46     /*
47     Constructor for Prof Class.
48      */
49     Prof( Basket BibBasket, Basket ForkBasket) {
50         this.BibBasket = BibBasket;
51         this.ForkBasket = ForkBasket;
52     }
53
54     /*
55     Method for running Prof thread, implements diners and thinkers process.
56      */
57     @Override
58     public void run(){
```

```
59          int linesWrittenInTurn;
60          int noodlesEatenInTurn;
61          Random rand = new Random();
62          while( (this.noodlesEaten < NUM_NOODLES_TO_EAT)
63                  && (this.linesWritten < NUM_LINES_TO_WRITE)) {
64
65              System.out.println( this.getName() + " is sleeping!");
66              try {
67                  sleep( rand.nextInt( 100));
68              } catch (InterruptedException e) {
69                  e.printStackTrace();
70              }
71
72              linesWrittenInTurn = writeCode();
73              this.linesWritten += linesWrittenInTurn;
74              System.out.println(
75                      this.getName() + " wrote " + linesWrittenInTurn +
76                              " lines of code, total: " + this.linesWritten);
77
78              System.out.println( this.getName() + " is hungry!");
79              noodlesEatenInTurn = eatNoodles();
80              this.noodlesEaten += noodlesEatenInTurn;
81                      System.out.println(
82                              this.getName() + " ate " + noodlesEatenInTurn +
83                                      " noodles, total: " + noodlesEaten);
84          }
85          System.out.println("===============>" + this.getName() + " is done!");
86      }
87      /*
88      Simulates a professor writing a random number of lines between 5 and 20
89      */
90      private int writeCode() {
91          Random rand = new Random();
92          int linesWritten = rand.nextInt( 16) + 5;
93          try {
94              sleep( linesWritten);
95          } catch (InterruptedException e) {
96              e.printStackTrace();
97          }
98          return linesWritten;
99      }
100     /*
101     Determines the number of noodles a professor eats between 2 and 10.
102     This method also is where the professor attempts to obtain a fork
103       and a bib (in order to eat).
104     Therefore, it will need to run methods from the Basket
105       file(defined in this file as BibBasket and ForkBasket) in order to get and
106       return forks and bibs to a basket shared by all other professor threads.
107     */
108     private int eatNoodles() {
109         Random rand = new Random();
110         boolean pickBibFirst;
111         boolean obtainedFork = false;
112         boolean obtainedBib = false;
113
114         while( !( obtainedBib && obtainedFork)) {
115             pickBibFirst = rand.nextBoolean();
116             if (pickBibFirst) {
```

```
117                 if (this.BibBasket.getItem(1000, getName())) {
118                     obtainedBib = true;
119                     if (this.ForkBasket.getItem(1100, getName())) {
120                         obtainedFork = true;
121                     } else {
122                         obtainedBib = false;
123                         this.BibBasket.returnItem(getName());
124                     }
125                 }
126             }
127             else{
128                 if (this.ForkBasket.getItem(1075, getName())) {
129                     obtainedFork = true;
130                     if (this.BibBasket.getItem(1025, getName())) {
131                         obtainedBib = true;
132                     } else {
133                         obtainedFork = false;
134                         this.ForkBasket.returnItem(getName());
135                     }
136                 }
137             }
138         }
139         int noodlesEaten = rand.nextInt( 9) + 2;
140 
141         try {
142             sleep( noodlesEaten * 100);
143         } catch (InterruptedException e) {
144             e.printStackTrace();
145         }
146         pickBibFirst = rand.nextBoolean();
147         if( pickBibFirst){
148             this.BibBasket.returnItem( getName());
149             this.ForkBasket.returnItem( getName());
150         }
151         else {
152             this.ForkBasket.returnItem( getName());
153             this.BibBasket.returnItem( getName());
154         }
155 
156         return noodlesEaten;
157     }
158 }
159 
```

```
 1 /*
 2 Basket.java
 3     - class for DinersAndThinkers
 4     - recommended size: ~50 lines
 5
 6 Implements the concept of a basket containing one type of item
 7
 8 Contains:
 9     - name for the items
10     - number of items available
11     - getItem, with parameter for the LIMIT ON TIME TO GET ITEM (deadlock)
12     - returnItem, returns the item
13
14 Any message from this class starts with "==="
15     - send message when:
16         - when someone is waiting for a resource
17         - when someone gets a resource
18         - when someone returns a resource
19
20 Authors: Philippe Nadon, Jack Shea
21  */
22
23 package dinersandthinkers;
24
25 class Basket {
26     private int numItems;
27     private String itemName;
28
29     /*
30     Constructor for Basket class.
31      */
32     Basket(int numItems, String itemName) {
33         this.numItems = numItems;
34         this.itemName = itemName;
35     }
36
37     /*
38     Simulates getting an item from the basket, by waiting until an item is
39         available and then decrementing numItems.
40     A timeout is used to ensure there is no deadlock.
41     Method is synchronized to ensure no data races,
42         as only one thread can access this method at a time.
43      */
44     synchronized boolean getItem( int timeout, String profName) {
45         long startWait;
46         long waitMilliSeconds;
47         System.out.println("===" +
48             profName + " wants a " + this.itemName);
49         System.out.println("===" +
50                 "there are " + this.numItems + " " + this.itemName);
51             if (this.numItems < 1) {
52                 System.out.println("===" +
53                         profName + " is waiting for a " + this.itemName);
54                 startWait = System.nanoTime();
55                 try {
56                     this.wait( timeout);
57                 } catch (InterruptedException e) {
58                     e.printStackTrace();
```

```
59                  }
60                  waitMilliSeconds = (System.nanoTime() - startWait) / 1000000;
61                  System.out.println("===" +
62                          profName + " waited " +
63                          waitMilliSeconds + "ms for a " + this.itemName);
64              }
65              if (this.numItems > 0) {
66                  System.out.println("===" +
67                          "A " + this.itemName + " was lent to " + profName);
68                  this.numItems--;
69                  return true;
70              }
71              return false;
72      }
73
74      /*
75      Simulates returning an item to the basket, by incrementing numItems
76          and notifying other threads of this change.
77      Method is synchronized to ensure no data races,
78          as only one thread can access this method at a time.
79       */
80      synchronized void returnItem( String profName) {
81          System.out.println("===" +
82              "A " + this.itemName + " was returned by " + profName);
83              this.numItems++;
84              this.notifyAll();
85      }
86  }
87
```

```
 1 /*
 2 DinersAndThinkers.java
 3     - contains main method, starting point
 4     - recommended size: ~30 lines
 5
 6 Steps performed (all prompts assume return positive int):
 7     - prompt number of profs and their names
 8     - prompt number of forks
 9     - prompt number of bibs
10     - create prof threads AS CLOSE IN TIME AS POSSIBLE
11
12 Authors: Philippe Nadon, Jack Shea
13  */
14 package dinersandthinkers;
15 import java.util.Scanner;
16
17 public class DinersAndThinkers {
18
19     public static void main( String[] args) {
20         Scanner userIn = new Scanner( System.in);
21
22         System.out.println( "Enter the number of profs: ");
23         int numProfs = userIn.nextInt();
24         userIn.nextLine();
25
26         System.out.println(
27                 "Enter the names of the profs, separated by commas (, ): ");
28         String[] profNames = userIn.nextLine().split(", ");
29
30         System.out.println( "Enter the number of forks: ");
31         int numForks = userIn.nextInt();
32
33         System.out.println( "Enter the number of bibs: ");
34         int numBibs = userIn.nextInt();
35
36         Basket BibBasket = new Basket( numBibs, "bib");
37         Basket ForkBasket = new Basket( numForks, "fork");
38
39         Prof[] profArray = new Prof[numProfs];
40         for( int i = 0; i < numProfs; i++) {
41             profArray[i] = new Prof( BibBasket, ForkBasket);
42             profArray[i].setName( profNames[i]);
43         }
44         // for loop split to start threads as closely as possible.
45         for( int i = 0; i < numProfs; i++) {
46             profArray[i].start();
47         }
48     }
49 }
50
```