```
 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     //First, we will need two variables: one that represnets  the
   minimum number of noodles that need to be eaten
40     // for a professor to be considered done, and the other the
   minimum number of lines that need to be written
41     // for a professor to be considered done. This is so the program
   actually has an end and doesn't go on forever.
42     private static final int NUM_NOODLES_TO_EAT = 85;
43     private static final int NUM_LINES_TO_WRITE = 200;
44     //Next, we will need two variables to track the total number of
```

```
44 noodles eaten and lines written for the professor.
45     //This is in order to keep track of the professors "progress" to
   being considered finished. Once these two
46     //variables values equal or exceeds their respective minimum
   numbers, the professor will be considered finished.
47     private int noodlesEaten;
48     private int linesWritten;
49     //Now, lets define two more variables to represent the bib basket
   and fork basket. These will be used later in the
50     //program in order to use a shared bib basket and fork basket
   between all professor threads.
51     private Basket BibBasket;
52     private Basket ForkBasket;
53 
54     /*
55     Constructor for Prof Class.
56      */
57     Prof( Basket BibBasket, Basket ForkBasket) {
58         this.BibBasket = BibBasket;
59         this.ForkBasket = ForkBasket;
60     }
61 
62     /*
63     Method for running Prof thread, implements diners and thinkers
   process.
64      */
65     @Override
66     public void run(){
67         int linesWrittenInTurn;
68         int noodlesEatenInTurn;
69         Random rand = new Random();
70         while( (this.noodlesEaten < NUM_NOODLES_TO_EAT) && (this.
   linesWritten < NUM_LINES_TO_WRITE)) {
71 
72             System.out.println( this.getName() + " is sleeping!");
73             try {
74                 sleep( rand.nextInt( 100));
75             } catch (InterruptedException e) {
76                 e.printStackTrace();
77             }
78 
79             linesWrittenInTurn = writeCode();
80             this.linesWritten += linesWrittenInTurn;
81             System.out.println( this.getName() + " wrote " +
   linesWrittenInTurn + " lines of code, total: " + this.linesWritten);
82 
83             System.out.println( this.getName() + " is hungry!");
84             noodlesEatenInTurn = eatNoodles();
85             this.noodlesEaten += noodlesEatenInTurn;
86                     System.out.println( this.getName() + " ate " +
```

```
 86 noodlesEatenInTurn + " noodles, total: " + noodlesEaten);
 87         }
 88         System.out.println("===============>" + this.getName() + " is
    done!");
 89     }
 90     /*
 91     * Simulates a professor writing a random number of lines between
   5 and 20
 92     */
 93     private int writeCode() {
 94         Random rand = new Random();
 95         int linesWritten = rand.nextInt( 16) + 5;
 96         try {
 97             sleep( linesWritten);
 98         } catch (InterruptedException e) {
 99             e.printStackTrace();
100         }
101         return linesWritten;
102     }
103     /*
104     * Determines the number of noodles a professor eats between 2 and
    10. This method also is where the
105     * professor attempts to obtain a form and a bib (in order to eat
   ). Therefore, it will need to run methods from
106     * the Basket file(defined in this file as BibBasket and
   ForkBasket) in order to get and return forks and
107     * bibs to a basket shared by all other professor threads.
108     */
109     private int eatNoodles() {
110         Random rand = new Random();
111         boolean pickBibFirst;
112         boolean obtainedFork = false;
113         boolean obtainedBib = false;
114 
115         while( !( obtainedBib && obtainedFork)) {
116             pickBibFirst = rand.nextBoolean();
117             if (pickBibFirst) {
118                 if (this.BibBasket.getItem(1000, getName())) {
119                     obtainedBib = true;
120                     if (this.ForkBasket.getItem(1100, getName())) {
121                         obtainedFork = true;
122                     } else {
123                         obtainedBib = false;
124                         this.BibBasket.returnItem(getName());
125                     }
126                 }
127             }
128             else{
129                 if (this.ForkBasket.getItem(1075, getName())) {
130                     obtainedFork = true;
```

```
131                     if (this.BibBasket.getItem(1025, getName())) {
132                         obtainedBib = true;
133                     } else {
134                         obtainedFork = false;
135                         this.ForkBasket.returnItem(getName());
136                     }
137                 }
138             }
139         }
140         int noodlesEaten = rand.nextInt( 9) + 2;
141 
142         try {
143             sleep( noodlesEaten * 100);
144         } catch (InterruptedException e) {
145             e.printStackTrace();
146         }
147         pickBibFirst = rand.nextBoolean();
148         if( pickBibFirst){
149             this.BibBasket.returnItem( getName());
150             this.ForkBasket.returnItem( getName());
151         }
152         else {
153             this.ForkBasket.returnItem( getName());
154             this.BibBasket.returnItem( getName());
155         }
156 
157         return noodlesEaten;
158     }
159 }
160 
```

```
 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 available and then decrementing numItems.
39     A timeout is used to ensure there is no deadlock.
40     Method is synchronized to ensure no data races, as only one thread
    can access this method at a time.
41      */
42     synchronized boolean getItem( int timeout, String profName) {
43         long startWait;
44         long waitMilliSeconds;
45         System.out.println("===" +
46             profName + " wants a " + this.itemName);
47         System.out.println("===there are " + this.numItems + " " +
```

```
47 this.itemName);
48             if (this.numItems < 1) {
49                 System.out.println("===" +
50                         profName + " is waiting for a " + this.
   itemName);
51                 startWait = System.nanoTime();
52                 try {
53                     this.wait( timeout);
54                 } catch (InterruptedException e) {
55                     e.printStackTrace();
56                 }
57                 waitMilliSeconds = (System.nanoTime() - startWait) /
   1000000;
58                 System.out.println("===" +
59                         profName + " waited " + waitMilliSeconds + "ms
    for a " + this.itemName);
60             }
61             if (this.numItems > 0) {
62                 System.out.println("===" +
63                         "A " + this.itemName + " was lent to " +
   profName);
64                 this.numItems--;
65                 return true;
66             }
67             return false;
68     }
69 
70     /*
71     Simulates returning an item to the basket, by incrementing
   numItems and notifying other threads of this change.
72     Method is synchronized to ensure no data races, as only one thread
    can access this method at a time.
73      */
74     synchronized void returnItem( String profName) {
75         System.out.println("===" +
76             "A " + this.itemName + " was returned by " + profName);
77             this.numItems++;
78             this.notifyAll();
79     }
80 }
81 
```

```java
/*
DinersAndThinkers.java
    - contains main method, starting point
    - recommended size: ~30 lines

Steps performed (all prompts assume return positive int):
    - prompt number of profs and their names
    - prompt number of forks
    - prompt number of bibs
    - create prof threads AS CLOSE IN TIME AS POSSIBLE

Authors: Philippe Nadon, Jack Shea
 */
package dinersandthinkers;
import java.util.Scanner;

public class DinersAndThinkers {

    public static void main( String[] args) {
        Scanner userIn = new Scanner( System.in);

        System.out.println( "Enter the number of profs: ");
        int numProfs = userIn.nextInt();
        userIn.nextLine();

        System.out.println( "Enter the names of the profs, separated by commas (, ): ");
        String[] profNames = userIn.nextLine().split(", ");

        System.out.println( "Enter the number of forks: ");
        int numForks = userIn.nextInt();

        System.out.println( "Enter the number of bibs: ");
        int numBibs = userIn.nextInt();

        Basket BibBasket = new Basket( numBibs, "bib");
        Basket ForkBasket = new Basket( numForks, "fork");

        Prof[] profArray = new Prof[numProfs];
        for( int i = 0; i < numProfs; i++) {
            profArray[i] = new Prof( BibBasket, ForkBasket);
            profArray[i].setName( profNames[i]);
        }
        // for loop split to start threads as closely as possible.
        for( int i = 0; i < numProfs; i++) {
            profArray[i].start();
        }
    }
}

```