WORKED EXAMPLE 11.1 Analyzing Baby Names

© Nancy Ross/iStockphoto.

Problem Statement The Social Security Administration publishes lists of the most popular baby names on their web site `http://www.ssa.gov/OACT/babynames/`. When querying the 1,000 most popular names for a given decade, the browser displays the result on the screen (see the *Querying Baby Names* figure below).

To save the data as text, one simply selects it and pastes the result into a file. The book's code contains a file `babynames.txt` with the data for the 1990s.

Each line in the file contains seven entries:

- The rank (from 1 to 1,000)
- The name, frequency, and percentage of the male name of that rank
- The name, frequency, and percentage of the female name of that rank

For example, the line

```
10  Joseph  260365  1.2681  Megan  160312  0.8168
```

shows that the 10th most common boy's name was Joseph, with 260,365 births, or 1.2681 percent of all births during that period. The 10th most common girl's name was Megan. Why are there many more Josephs than Megans? Parents seem to use a wider set of girl's names, making each one of them less frequent.

Your task is to test that conjecture, by determining the names given to the top 50 percent of boys and girls in the list. Simply print boy and girl names, together with their ranks, until you reach the 50 percent limit.

Step 1 Understand the processing task.

To process each line, we first read the rank. We then read three values (name, count, and percentage) for the boy's name. Then we repeat that step for girls. To stop processing after reaching 50 percent, we can add up the frequencies and stop when they reach 50 percent.

We need separate totals for boys and girls. When a total reaches 50 percent, we stop printing. When both totals reach 50 percent, we stop reading.

The following pseudocode describes our processing task.

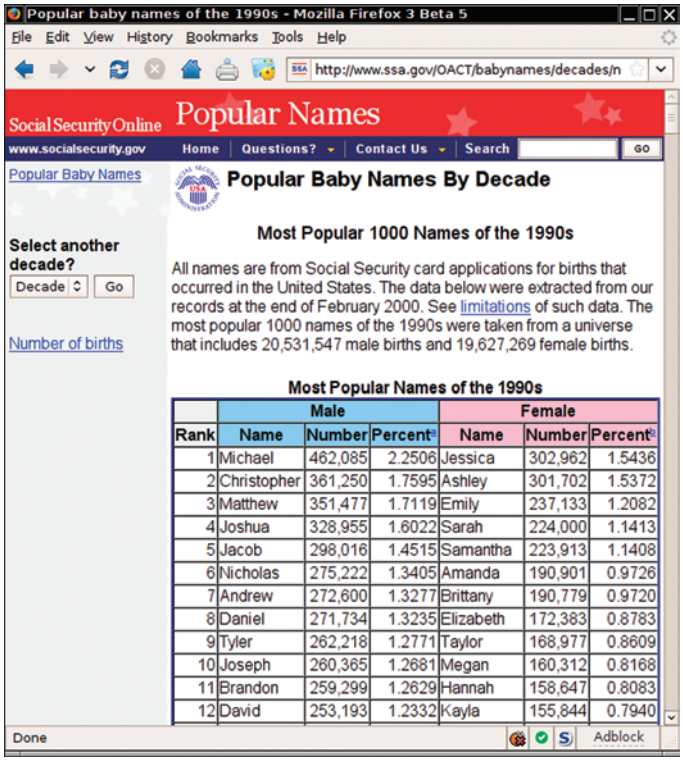
```
boyTotal = 0
girlTotal = 0
While boyTotal < 50 or girlTotal < 50
   Read the rank and print it.

   Read the boy name, count, and percentage.
   If boyTotal < 50
      Print boy name.
   Add percentage to boyTotal.

   Repeat for girl part.
```

Step 2 Determine which files you need to read and write.

We only need to read a single file, `babynames.txt`. We were not asked to save the output to a file, so we will just send it to `System.out`.

Step 3 Choose a mechanism for obtaining the file names.

We do not need to prompt the user for the file name.



Querying Baby Names

Step 4 Choose between line, word, and character-based input.

The Social Security Administration data do not contain names with spaces, such as "Mary Jane". Therefore, each data record contains exactly seven entries, as shown in the screen capture above. This input can be safely processed by reading words and numbers.

Step 5 With line-oriented input, extract the required data.

We can skip this step because we don't read a line at a time.

But suppose you decided in Step 4 to choose line-oriented input. Then you would need to break the input line into seven strings, converting five of them to numbers. This is quite tedious and it might well make you reconsider your choice.

Step 6 Use classes and methods to factor out common tasks.

In the pseudocode, we wrote **Repeat for girl part**. Clearly, there is a common task that calls for a helper method. It involves three tasks:

Read the name, count, and percentage.
Print the name if the total is less than 50 percent.
Add the percentage to the total.

We use a helper class `RecordReader` for this purpose and construct two objects, one each for processing the boy and girl names. Each `RecordReader` maintains a separate total, updates it by adding the current percentage, and prints names until the limit has been reached. Our main processing loop then becomes

```
RecordReader boys = new RecordReader(LIMIT);
RecordReader girls = new RecordReader(LIMIT);
```

```
while (boys.hasMore() || girls.hasMore())
{
   int rank = in.nextInt();
   System.out.print(rank + " ");
   boys.process(in);
   girls.process(in);
   System.out.println();
}
```

Here is the code of the process method:

```
/**
   Reads an input record and prints the name if the current total is less than the limit.
   @param in the input stream
*/
public void process(Scanner in)
{
   String name = in.next();
   int count = in.nextInt();
   double percent = in.nextDouble();

   if (total < limit) { System.out.print(name + " "); }
   total = total + percent;
}
```

The complete program is shown below.

Have a look at the program output. Remarkably, only 69 boy names and 153 girl names account for half of all births. That's good news for those who are in the business of producing personalized doodads. Exercise E11.12 asks you to study how this distribution has changed over the years.

worked_example_1/BabyNames.java

```
1  import java.io.File;
2  import java.io.FileNotFoundException;
3  import java.util.Scanner;
4
5  public class BabyNames
6  {
7     public static final double LIMIT = 50;
8
9     public static void main(String[] args) throws FileNotFoundException
10    {
11       try (Scanner in = new Scanner(new File("babynames.txt")))
12       {
13          RecordReader boys = new RecordReader(LIMIT);
14          RecordReader girls = new RecordReader(LIMIT);
15
16          while (boys.hasMore() || girls.hasMore())
17          {
18             int rank = in.nextInt();
19             System.out.print(rank + " ");
20             boys.process(in);
21             girls.process(in);
22             System.out.println();
23          }
24       }
25    }
26 }
```

worked_example_1/RecordReader.java

```
1  import java.util.Scanner;
2
3  /**
4     This class processes baby name records.
5  */
6  public class RecordReader
7  {
8     private double total;
9     private double limit;
10
11    /**
12       Constructs a RecordReader with a zero total.
13    */
14    public RecordReader(double aLimit)
15    {
16       total = 0;
17       limit = aLimit;
18    }
19
20    /**
21       Reads an input record and prints the name if the current total is less
22       than the limit.
23       @param in the input stream
24    */
25    public void process(Scanner in)
26    {
27       String name = in.next();
28       int count = in.nextInt();
29       double percent = in.nextDouble();
30
31       if (total < limit) { System.out.print(name + " "); }
32       total = total + percent;
33    }
34
35    /**
36       Checks whether there are more inputs to process.
37       @return true if the limit has not yet been reached
38    */
39    public boolean hasMore()
40    {
41       return total < limit;
42    }
43 }
```