```java
/**
 * Class that contains data on a baby name.
 * Contains the name and gender of the baby and the year and the number of
 * babies born in that year
 *
 * @author Silas Agnew
 * @version November 9, 2017
 */

public class BabyName implements Comparable
{
    String name     = null;
    boolean gender  = false;
    int count       = -1;
    int year        = -1;

    //-Constructor(s)-----------------------------------------------//

    /**
     * Constructs a {@code BabyName} object that contains info about a baby name
     * @param name The name of the baby
     * @param gender Probable gender of the name
     * @param count Number of babies born in {@code year}
     * @param year Year the babies were born
     */
    public BabyName(String name, boolean gender, int count, int year)
    {
        setName(name);
        setGender(gender);
        setCount(count);
        setYear(year);
    }

    /**
     * Takes a line of CSV data and parses it into usable data in
     * {@code BabyName}.
     * @param csvLine Line of CSVs
     * @return A constructed {@code BabyName} from parsed csv data
     * @throws IllegalArgumentException 1) if there is not 4 CSVs in the string
     *  2) if there is an invalid numeric value (i.e. negative)
     */
    public static BabyName BabyNameBuilder(String csvLine, int lineNum)
            throws IllegalArgumentException
    {
        boolean gender = false;
        int count = 0;
        int year = 0;

        String[] csv = csvLine.split("[,]+");
        if (csv.length != 4)
            throw new IllegalArgumentException(
                "CSV line " + lineNum + " is ill-formatted: must contain 4 values");

        // Gender
        if (csv[1].equalsIgnoreCase("M")) gender = false;
        else if (csv[1].equalsIgnoreCase("F")) gender = true;
```

```
 57         else throw new IllegalArgumentException(
 58                 "Gender (line: " + lineNum + ", col: 2) contains neither M or F");
 59
 60         // Count
 61         count = Integer.parseInt(csv[2]);
 62         if (count <= 0) throw new IllegalArgumentException(
 63                 "Birth count (line: " + lineNum +
 64                         ", col: 2) cannot be a negative number.");
 65
 66         // Year
 67         year = Integer.parseInt(csv[3]);
 68         if (year <= 0) throw new IllegalArgumentException(
 69                 "Birth year (line: " + lineNum +
 70                         ", col: 2) cannot be a negative number.");
 71
 72         return new BabyName(csv[0], gender, count, year);
 73     }
 74
 75     //-Accessors-----------------------------------------------------//
 76
 77     /**
 78      * @return If the baby name is a female name.
 79      */
 80     public boolean isFemale() { return gender; }
 81
 82     /**
 83      * @return The baby's name.
 84      */
 85     public String getName() { return name; }
 86
 87     /**
 88      * @return The birth count of the name.
 89      */
 90     public int getCount() { return count; }
 91
 92     /**
 93      * @return The year of the name.
 94      */
 95     public int getYear() { return year; }
 96
 97     //-Mutators------------------------------------------------------//
 98
 99     /**
100      * Sets the baby name.
101      */
102     public void setName(String name) { this.name = name; }
103
104     /**
105      * Sets the baby gender.
106      */
107     public void setGender(boolean gender) { this.gender = gender; }
108
109     /**
110      * Sets the baby birth count.
111      */
112     public void setCount(int count) { this.count = count; }
```

```
113
114     /**
115      * Sets the baby birth year.
116      */
117     public void setYear(int year) { this.year = year; }
118
119     /**
120      * @return A formatted sentence interpreting the data in the class
121      */
122     @Override
123     public String toString()
124     {
125        return count + (isFemale() ? " girls" : " boys") +
126                " named " + name + " in " + year;
127     }
128
129     /**
130      * Compares 2 {@code BabyName} objects by popularity (birth count).
131      * @param other {@code BabyName} to compare this to.
132      * @return {@code n > 0} if {@code other} is greater than {@code this}
133      */
134     @Override
135     public int compareTo(Object other)
136     {
137         return ((BabyName)other).count - count;
138     }
139 }
140
```