```java
import java.io.*;
// import java.util.Scanner;

public class Parser
{
  private BufferedReader readFile;
  private String currentLine;
  private String nextLine;

  /**
   * Constructor / initializer: Creates a Parser and opens the source text file
   * @throws IOException
   */
  public Parser(File f) throws IOException
  {
    // check if empty
    if (f == null)
      throw new IllegalArgumentException("File is empty.");

    // open and read file
    try
    {
      readFile = new BufferedReader(new FileReader(f));
    }
    catch (FileNotFoundException e)
    {
      throw new IllegalArgumentException("Missing file.");
    }

    this.currentLine = null;
    this.nextLine = this.getNextLine();

    advance();
  }

  private String getNextLine() throws IOException
  {
    String nextLine;

    do
    {
      nextLine = this.readFile.readLine();

      if (nextLine == null)
      {
        return null;
      }
    } while (nextLine.trim().isEmpty() || this.isComment(nextLine));

    int commentIndex = nextLine.indexOf("//");
    if (commentIndex != -1)
    {
      nextLine = nextLine.substring(0, commentIndex - 1);
    }

    return nextLine;
  }

  private boolean isComment(String s)
```

```
60    {
61      return s.trim().startsWith("//");
62    }
63
64    /** Checks if there is more work to do (boolean) */
65    public Boolean hasMoreCommands()
66    {
67      return this.nextLine != null;
68    }
69
70    /** Gets the next instruction and makes it the current instruction (string)
71     * @throws IOException*/
72    public void advance() throws IOException
73    {
74      this.currentLine = this.nextLine;
75      this.nextLine = this.getNextLine();
76
77    }
78
79    /**
80     * Returns the current instruction type (constant):
81     *
82     * A_INSTRUCTION for @ xxx, where xxx is either a decimal number or a symbol
83     * C_INSTRUCTION for dest = comp ; jump L_INSTRUCTION for (label)
84     */
85    public CommandType instructionType()
86    {
87      String trimmedLine = this.currentLine.trim();
88
89      if (trimmedLine.startsWith("(") && trimmedLine.endsWith(")"))
90      {
91        return CommandType.L_COMMAND;
92      }
93      else if (trimmedLine.startsWith("@"))
94      {
95        return CommandType.A_COMMAND;
96      }
97      else
98      {
99        return CommandType.C_COMMAND;
100     }
101
102   }
103
104   /**
105    * Returns the instruction’s symbol (string)
106    *
107    * ex: @sum --> "sum" and (LOOP) --> "LOOP"
108    */
109   public String symbol()
110   {
111     String trimmedLine = this.currentLine.trim();
112
113     if (this.instructionType().equals(CommandType.L_COMMAND))
114     {
115       return trimmedLine.substring(1, this.currentLine.length() - 1);
116     }
117     else if (this.instructionType().equals(CommandType.A_COMMAND))
118     {
119       return trimmedLine.substring(1);
```

```
120     }
121     else
122     {
123       return null;
124     }
125   }
126
127   public enum CommandType
128   {
129     A_COMMAND, C_COMMAND, L_COMMAND;
130   }
131
132   /*
133    * D=D+1;JLE current instruction
134    *
135    * dest() returns "D" comp() returns "D+1" jump() returns "JLE"
136    */
137
138   /** Returns the instruction’s dest field (string) */
139   public String dest()
140   {
141     String trimmedLine = this.currentLine.trim();
142     int equalInd = trimmedLine.indexOf("=");
143
144     if (equalInd == -1)
145     {
146       return null;
147     }
148     else
149     {
150       return trimmedLine.substring(0, equalInd);
151     }
152   }
153
154   /** Returns the instruction’s comp field (string) */
155   public String comp()
156   {
157     String trimmedLine = this.currentLine.trim();
158     int equalInd = trimmedLine.indexOf("=");
159     if (equalInd != -1)
160     {
161       trimmedLine = trimmedLine.substring(equalInd + 1);
162     }
163     int semiInd = trimmedLine.indexOf(";");
164
165     if (semiInd == -1)
166     {
167       return trimmedLine;
168     }
169     else
170     {
171       return trimmedLine.substring(0, semiInd);
172     }
173   }
174
175   /** Returns the instruction’s jump field (string) */
176   public String jump()
177   {
178     String trimmedLine = this.currentLine.trim();
179     int semiInd = trimmedLine.indexOf(";");
```

```
180
181     if (semiInd == -1)
182     {
183       return null;
184     }
185     else
186     {
187       return trimmedLine.substring(semiInd + 1);
188     }
189   }
190
191
192 }
193
```