```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)
  {
    return s.trim().startsWith("//");
  }

  /** Checks if there is more work to do (boolean) */
  public Boolean hasMoreCommands()
  {
    return this.nextLine != null;
  }

  /** Gets the next instruction and makes it the current instruction (string)
   * @throws IOException*/
  public void advance() throws IOException
  {
    this.currentLine = this.nextLine;
    this.nextLine = this.getNextLine();

  }

  /**
   * Returns the current instruction type (constant):
   *
   * A_INSTRUCTION for @ xxx, where xxx is either a decimal number or a symbol
   * C_INSTRUCTION for dest = comp ; jump L_INSTRUCTION for (label)
   */
  public CommandType instructionType()
  {
    String trimmedLine = this.currentLine.trim();

    if (trimmedLine.startsWith("(") && trimmedLine.endsWith(")"))
    {
      return CommandType.L_COMMAND;
    }
    else if (trimmedLine.startsWith("@"))
    {
      return CommandType.A_COMMAND;
    }
    else
    {
      return CommandType.C_COMMAND;
    }

  }

  /**
   * Returns the instruction’s symbol (string)
   *
   * ex: @sum --> "sum" and (LOOP) --> "LOOP"
   */
  public String symbol()
  {
    String trimmedLine = this.currentLine.trim();

    if (this.instructionType().equals(CommandType.L_COMMAND))
    {
      return trimmedLine.substring(1, this.currentLine.length() - 1);
    }
    else if (this.instructionType().equals(CommandType.A_COMMAND))
    {
      return trimmedLine.substring(1);
    }
    else
    {
      return null;
    }
  }

  public enum CommandType
  {
    A_COMMAND, C_COMMAND, L_COMMAND;
  }

  /*
   * D=D+1;JLE current instruction
   *
   * dest() returns "D" comp() returns "D+1" jump() returns "JLE"
   */

  /** Returns the instruction’s dest field (string) */
  public String dest()
  {
    String trimmedLine = this.currentLine.trim();
    int equalInd = trimmedLine.indexOf("=");

    if (equalInd == -1)
    {
      return null;
    }
    else
    {
      return trimmedLine.substring(0, equalInd);
    }
  }

  /** Returns the instruction’s comp field (string) */
  public String comp()
  {
    String trimmedLine = this.currentLine.trim();
    int equalInd = trimmedLine.indexOf("=");
    if (equalInd != -1)
    {
      trimmedLine = trimmedLine.substring(equalInd + 1);
    }
    int semiInd = trimmedLine.indexOf(";");

    if (semiInd == -1)
    {
      return trimmedLine;
    }
    else
    {
      return trimmedLine.substring(0, semiInd);
    }
  }

  /** Returns the instruction’s jump field (string) */
  public String jump()
  {
    String trimmedLine = this.currentLine.trim();
    int semiInd = trimmedLine.indexOf(";");
```

```java

    if (semiInd == -1)
    {
      return null;
    }
    else
    {
      return trimmedLine.substring(semiInd + 1);
    }
  }


}

```