```
package A01_Grundlage_der_Bioinformatik;

import java.io.*;
import java.util.ArrayList;

/**
 * Created by JWP on 04.22.2018
 */

public class FastA {
	private enum State { HEADER,
		SEQ,
		HEADERSEQ }

	// Constants
	private final String HEADERSTART = ">";
	private final int BLOCKLEN = 80;

	private java.util.List<String> headers = new ArrayList<String>();
	private java.util.List<String> sequences = new ArrayList<String>();

	public void read(Reader r) throws IOException {

		// Processes FastA file line by line
		BufferedReader br = new BufferedReader(r);

		// Used for saving sequence lines
		StringBuilder sb = new StringBuilder();

		// Keeps track of state (remember the state diagram from the tutorial)
		State currentState = State.HEADER;

		String line;
		while( (line = br.readLine()) != null ) {

			// Ignore leading and trailing whitespace of line
			line = line.trim();

			// Ignore empty lines
			if ("".equals(line)) {

				continue;
			}
```

```
            // Switch on current State and how line starts
            if (currentState.equals(State.HEADER) && line.startsWith(HEADERSTART)) {

                // Add header to header list and change State
                headers.add(line);
                currentState = State.SEQ;
            } else if (currentState.equals(State.SEQ) && !line.startsWith(HEADERSTART)) {

                // Assume here that line is sequence, append to StringBuilder and change
State
                sb.append(line);
                currentState = State.HEADERSEQ;

            } else if (currentState.equals(State.HEADERSEQ)) {

                if (line.startsWith(HEADERSTART)) {

                    // Add sequence of String builder to list and empty String Builder
                    sequences.add(sb.toString());
                    sb.setLength(0);

                    headers.add(line);
                    currentState = State.SEQ;

                    // Assume line is sequence
                } else {

                    sb.append(line);
                }
            } else {

                throw new IllegalArgumentException("The FASTA file is malformed");
            }
        }
        br.close();

        // Add remaining sequence to sequence list
        sequences.add(sb.toString());

        if(this.headers.size() != this.sequences.size()) {

            throw new IllegalArgumentException("Number of headers and sequences is not
equal!");
        }
```

```
}


/**
 *
 * Writes all sequences containes in this object to the provided Writer.
 *
 * @param w The Writer to which the FASTA records should be written to.
 */
public void write(Writer w) throws IOException {

	BufferedWriter bw = new BufferedWriter(w);

	for(int i = 0; i < this.size(); i++) {

		// Write header line
		bw.write(this.getHeader(i));
		bw.newLine();

		String seq = this.getSequence(i);
		// Write associated sequence, trim to 80 characters
		for (int start = 0; start < seq.length(); start += BLOCKLEN) {

			bw.write(seq.substring(start, Math.min(seq.length(), start + BLOCKLEN)));
			bw.newLine();
		}
		bw.newLine();
	}
	bw.close();
}

public int size() {

	return headers.size();
}

public String getHeader(int i) {

	return headers.get(i);
}

String getSequence(int i) {

	return sequences.get(i);
	}

	public void add(String header, String sequence) {

		this.headers.add(header);
		this.sequences.add(sequence);
	}
}
```

```
package A01_Grundlage_der_Bioinformatik;

import java.io.*;

/**
 * Created by JWP on 04.22.2018
 */
public class ReverseComplement {

    public static void main(String[] args) throws IOException {

        if (args.length == 2) { // need exactly two commandline arguments: infile and outfile


            System.out.println("Constructing reverse complement DNA Seqs...");


            String infile = args[0];
            Reader r = new FileReader(infile);

            A01_Grundlage_der_Bioinformatik.FastA          inputFasta          =          new
A01_Grundlage_der_Bioinformatik.FastA();
            inputFasta.read(r);
            r.close();

            A01_Grundlage_der_Bioinformatik.FastA          resultFasta          =          new
A01_Grundlage_der_Bioinformatik.FastA();

            for (int i = 0; i < inputFasta.size(); i++) {
                resultFasta.add(inputFasta.getHeader(i) + " (Reverse Complemented)",
                        Construct_Reverse_Complement(inputFasta.getSequence(i)));
            }

            String outfile = args[1];
            Writer w = new FileWriter(outfile);
            resultFasta.write(w); // write the result
            w.close(); // finished writing, close
        } else {
            System.out.println("Please set up your arguments");
        }
    }


    public static String Construct_Reverse_Complement(String sequence) {
        String reverseComplement = "";


        for (int i = 0; i < sequence.length(); i++)
            switch (sequence.charAt(i)) {
                case 'A':
                    reverseComplement = "T" + reverseComplement;
                    break;
                case 'T':
                    reverseComplement = "A" + reverseComplement;
                    break;
                case 'G':
                    reverseComplement = "C" + reverseComplement;
                    break;
                case 'C':
                    reverseComplement = "G" + reverseComplement;
                    break;


            }
        return reverseComplement;
    }
}
```