Listing 1: UE04_Download/src/processing/Processor.java

```java
package processing;

import data.Data;

/**
 * This interface represents processing objects that take a <code>Data</code> object as
 * input, processes it, and returns a new, processed <code>Data</code> object.
 */
public interface Processor {

  /**
   * Processes the specified <code>Data</code> object and returns a new, processed
   * <code>Data</code> object.
   *
   * @param data The <code>Data</code> object to process.
   * @return A new, processed <code>Data</code> object
   */
  Data process(Data data);

  /**
   * Returns the name of this <code>Processor</code> object. This should be a
   * human-readable string representation.
   *
   * @return The name.
   */
  String getName();

}
```

Listing 2: UE04_Download/src/processing/Processors.java

```java
package processing;

import data.Data;
import java.util.ArrayList; // import the ArrayList class

/**
 * This static-only class provides several factory methods and classes for
 * creating {@link Processor} objects.
 */
public class Processors {

  // static only class that should not be instantiated; hide constructor
  private Processors() {
  }

  // 1) oeffentliche, abstrakte, statische, innere Klasse "Scaler"
  public static abstract class Scaler implements Processor {
    public abstract double getMin();

    public abstract double getMax();

    public Data process(final Data data) {
      final double dataMin = DataUtil.min(data);
      final double dataMax = DataUtil.max(data);
      final ArrayList<Double> newValues = new ArrayList<Double>();

      data.forEach((val) -> {
        final double scaled = (val - dataMin) / (dataMax - dataMin);
        final double newVal = scaled * (getMax() - getMin()) + getMin();
        newValues.add(newVal);
      });
      // as Data() constructor requires a double[] and i work with a arraylist i
      // convert it back to primitive double arry
      // i could also use a double[] because of data.size() function but this seems
      // acceptable too
      // in addition this fullfills all requirements of the task given
      return new Data(newValues.stream().mapToDouble(Double::doubleValue).toArray());
    }

    @Override
    public String getName() {
      return "Scaler(min=" + this.getMin() + ",␣max=" + this.getMax() + ")";
    }
  }

  // 2) oeffentliche, statische Methode "Processor scale(double min, double max)"
  public static Processor scale(final double min, final double max) {
    return new Scaler() {

      @Override
      public double getMin() {
        return min;
      }

      @Override
      public double getMax() {
        return max;
      }

    };

  }

  // 3) oeffentliche, statische, innere Klasse "PercentScaler"
  public static class PercentScaler extends Scaler {

    @Override
    public double getMin() {
      return 0;
    }

    @Override
    public double getMax() {
      return 100;
    }

  }

  // 4) oeffentliche, statische Methode "Processor standardize()"
  public static Processor standardize() {
    return new Processor() {

      @Override
      public Data process(final Data data) {
        final ArrayList<Double> newValues = new ArrayList<Double>();
        final double avg = DataUtil.avg(data);
        final double std = DataUtil.std(data);

        data.forEach((val) -> {
          final double newVal = (val - avg) / std;
          newValues.add(newVal);
        });
        // see above why i use arraylist
        return new Data(newValues.stream().mapToDouble(Double::doubleValue).toArray());
      }
      @Override
      public String getName() {
        return "Standardizer";
      }
    };
  }

  // 5) private, statische, innere Klasse "Clipper"

  private static class Clipper implements Processor {
    private final boolean clipLower;
    private final boolean clipUpper;
    private final double lower;
    private final double upper;

    public Clipper(final boolean clipLower, final boolean clipUpper, final double
        lower, final double upper) {
      this.clipLower = clipLower;
      this.clipUpper = clipUpper;
      this.lower = lower;
      this.upper = upper;
    }

    @Override
    public Data process(final Data data) {
      final ArrayList<Double> newValues = new ArrayList<Double>();

      data.forEach((val) -> {
        if (this.clipLower && val < this.lower) {
          newValues.add(this.lower);
        } else if (this.clipUpper && val > this.upper) { // use else if, because this
            saves the second if check
                                      // if we hit the first if
          newValues.add(this.upper);
        } else {
          newValues.add(val);
        }

      });
      // see above why i use arraylist
      return new Data(newValues.stream().mapToDouble(Double::doubleValue).toArray());
    }

    @Override
    public String getName() {
      String val = "Clipper";
      if (this.clipLower && this.clipUpper) {
        val += "(lower=" + this.lower + ",␣upper=" + this.upper + ")";
      } else if (this.clipLower) {
        val += "(lower=" + this.lower + ")";
      } else if (this.clipUpper) {
        val += "(upper=" + this.lower + ")";
      }
      return val;
    }

  }
  // 6) oeffentliche, statische Methode "Processor clip(double lower, double
  // upper)"

  public static Processor clip(double lower, double upper) {
    return new Clipper(true, true, lower, upper);
  }
  // 6) oeffentliche, statische Methode "Processor clipLower(double lower)"
  public static Processor clipLower(double lower) {
    return new Clipper(true, false, lower, 0);
  }
  // 6) oeffentliche, statische Methode "Processor clipUpper(double upper)"

  public static Processor clipUpper(double upper) {
    return new Clipper(false, true, 0, upper);
  }

  // static helper class for statistical measures of Data objects
  private static class DataUtil {

    /**
     * Returns the minimum of the specified <code>Data</code> object.
     */
    public static double min(final Data data) {
      double min = Double.POSITIVE_INFINITY;
      for (final double d : data) {
        if (d < min) {
          min = d;
        }
      }
      return min;
    }

    /**
     * Returns the maximum of the specified <code>Data</code> object.
     */
    public static double max(final Data data) {
      double max = Double.NEGATIVE_INFINITY;
      for (final double d : data) {
        if (d > max) {
          max = d;
        }
      }
      return max;
    }

    /**
     * Returns the average (mean) of the specified <code>Data</code> object.
     */
    public static double avg(final Data data) {
      double sum = 0;
      for (final double d : data) {
        sum += d;
      }
      return sum / data.size();
    }

    /**
     * Returns the standard deviation of the specified <code>Data</code> object.
     */
    public static double std(final Data data) {
      final double avg = avg(data);
      double sum = 0;
      for (final double d : data) {
        final double deviation = d - avg;
        sum += deviation * deviation;
      }
      return Math.sqrt(sum / data.size());
    }

  }
```

```
}
```

Listing 3: UE04_Download/src/data/Data.java

```java
package data;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Iterator;

/**
 * This class represents an immutable data object whose content are double
 * values, i.e., it is basically an immutable double array. New
 * <code>Data</code> objects can be created either by using the constructor
 * {@link #Data(double[])} or the static factory method
 * {@link #readFromFile(String)}.
 * <p>
 * The individual double values can be accessed only via an iterator, for
 * example, by using the foreach-loop:
 *
 * <pre>
 *     for (double d: myDataObject) {
 *         ...
 *     }
 * </pre>
 *
 * Exactly {@link #size()} values will be returned by this iteration.
 */
public class Data implements Iterable<Double> {

  private final double[] values;

  /**
   * Creates a new immutable <code>Data</code> object using the specified
   * <code>values</code>.
   *
   * @param values The double values of this <code>Data</code> object.
   */
  public Data(double[] values) {
    this.values = values;
  }

  /**
   * Returns the size of the stored values, i.e., the number of elements this
   * <code>Data</code> object contains. The iterator will have exactly this many
   * iterations.
   *
   * @return The size of the stored values.
   */
  public int size() {
    return values.length;
  }

  /**
   * Returns an iterator for iterating over the double values of this
   * <code>Data</code> object.
   *
   * @return The iterator for the double values.
   */
  @Override
  public Iterator<Double> iterator() {
    return new DataIterator();
  }

  @Override
  public String toString() {
    return toCsvString();
  }

  /**
   * Creates a new <code>Data</code> object from the contents of the CSV-file
   * specified by the given <code>path</code>.
   *
   * @param path The path of the file whose contents should be used for the new
   *             <code>Data</code> object.
   * @return A new <code>Data</code> object.
   * @throws IOException Thrown when anything goes wrong when reading the file.
   */
  public static Data readFromFile(String path) throws IOException {
    String[] parts = Files.readString(Path.of(path)).split(",");
    double[] values = new double[parts.length];
    for (int i = 0; i < parts.length; i++) {
      values[i] = Double.parseDouble(parts[i]);
    }
    return new Data(values);
  }

  /**
   * Write the double values of this <code>Data</code> object as comma separated
   * values to a file.
   *
   * @param path The path for the output file.
   * @throws IOException Thrown when anything goes wrong when writing to the file.
   */
  public void writeToFile(String path) throws IOException {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(path))) {
      writer.write(toCsvString());
    }
  }

  /**
   * Returns a string containing the double values of this <code>Data</code>
   * object separated by commas.
   */
  private String toCsvString() {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < values.length - 1; i++) {
      sb.append(values[i]).append(",");
    }
    sb.append(values[values.length - 1]);
    return sb.toString();
  }

  private class DataIterator implements Iterator<Double> {

    private int i = 0;

    @Override
    public boolean hasNext() {
      return i < values.length;
    }

    @Override
    public Double next() {
      double d = values[i];
```

```java
            i++;
            return d;
        }

    }

}
```

Listing 4: UE04_Download/src/app/Main.java

```java
package app;

import data.Data;
import processing.Processor;
import processing.Processors;

import java.io.IOException;

public class Main {

    public static void main(String[] args) throws IOException {
        Data data = Data.readFromFile("data.csv");
        Processor[] processors = { new Processors.PercentScaler(), Processors.scale(-20,
             123), Processors.standardize(),
              Processors.clipLower(-1), Processors.clipUpper(1.1), Processors.clip(-0.7, 1)
                  };
        for (Processor p : processors) {
            System.out.println(String.format("processing␣data␣with␣'%s'", p.getName()));
            System.out.println(String.format("␣before:␣%s", data));
            data = p.process(data);
            System.out.println(String.format("␣after:␣%s", data));
        }
        data.writeToFile("data_processed.csv");
    }

    // MY OUTPUT

    /**
     * processing data with 'Scaler(min=0.0, max=100.0)' before:
     * 2.0,4.0,4.0,4.0,5.0,5.0,7.0,9.0 after:
     *
        0.0,28.57142857142857,28.57142857142857,28.57142857142857,42.857142857142854,42.85714285714
     * processing data with 'Scaler(min=-20.0, max=123.0)' before:
     *
        0.0,28.57142857142857,28.57142857142857,28.57142857142857,42.857142857142854,42.85714285714
     * after:
     *
        -20.0,20.857142857142854,20.857142857142854,20.857142857142854,41.285714285714285,41.285714
     * processing data with 'Standardizer' before:
     *
        -20.0,20.857142857142854,20.857142857142854,20.857142857142854,41.285714285714285,41.285714
     * after: -1.4999999999999998,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0
     * processing data with 'Clipper (lower=-1.0)' before:
     * -1.4999999999999998,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0 after:
     * -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0 processing data with
     * 'Clipper (upper=0.0)' before:
     * -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,2.0 after:
     * -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,1.1 processing data with
     * 'Clipper(lower=-0.7, upper=1.0)' before:
     * -1.0,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,1.1 after:
     * -0.7,-0.5,-0.5,-0.5,0.0,0.0,0.9999999999999998,1.0
     */

}
```