```java
package hyperDap.generator.main;

import java.util.List;
import java.util.Random;
import hyperDap.base.types.dataSet.DataSet;
import hyperDap.base.types.dataSet.ValueDataSet;

public class GenMain {

  /**
   * Create a new {@link ValueDataSet} made up of any number of mathematical functions combined.
   *
   * @param functionEncodings The {@code encoding} specifying what functions should be represented.
   *        See {@link GenSegment} for details.
   * @param numberOfBiases The number of times that the value of the functions should abruptly
   *        change, affecting all subsequent values.
   * @param base The {@code base} of the {@link DataSet}.
   * @param step The {@code step} of the {@link DataSet}.
   * @param length A rough number of the data points that is to be generated.
   * @param noise Unused at this time.
   * @return The generated {@link ValueDataSet}
   */
  public static ValueDataSet<Double> newDataSet(List<String> functionEncodings, int numberOfBiases,
      double base, double step, int length, double noise, double precision) {
    // protect from bad arguments
    if (functionEncodings.isEmpty()) {
      throw new IllegalArgumentException(
          String.format("%s was passed an empty list of functionEncodings", GenMain.class));
    }
    if (step == 0.0) {
      throw new IllegalArgumentException(
          String.format("%s has been passed illegal step size of 0.0!", GenMain.class));
    }
    if (length <= 0) {
      throw new IllegalArgumentException(
          String.format("%s has been passed illegal length argument of %s", GenMain.class, length));
    }
    // log and debugging
    System.out.println(String.format("%s.newDataSet(encodings, %s, %s, %s)", GenMain.class,
        numberOfBiases, base, step, length));
    for (String encoding : functionEncodings) {
      System.out.println(encoding);
    }
    System.out.println("generating now");
    // prepare data generation
    Random rand = new Random();
    int number = length / functionEncodings.size(); // the number of data points to be added
    ValueDataSet<Double> set =
        new ValueDataSet<Double>(base, step, precision, d -> Double.valueOf(d));
    set.add(0.0); // add an initial value
    // for each functionEncoding generate and add a list of data points
    GenSegment generator;
    double scale;
    double shiftX;
    double lastVal;
    for (String encoding : functionEncodings) {
      lastVal = set.getByIndex(set.size() - 1);
      scale = (Double.valueOf(rand.nextInt(10)) - 4.0) / 10;
      shiftX = Double.valueOf(rand.nextInt(30)) - 15.0;
      if (encoding.equals("rand")) { // random data is handled differently
        generator = new GenSegment("constant", scale, shiftX, lastVal, step);
        generator.addRandomToDoubleDataSet(set, number);
        continue;
      }
      generator = new GenSegment(encoding, scale, shiftX, lastVal, step);
      generator.addToDoubleDataSet(set, number, noise);
      // add a bias if needed
      if (numberOfBiases != 0) {
        numberOfBiases--;
        // the same function but shifted by the already added data points in X and by the intended
        // bias in Y
        generator =
            new GenSegment(encoding, scale, shiftX + number, lastVal + rand.nextInt(7) + 2, step);
        // for demonstration purposes only use visible and positive bias
        generator.addToDoubleDataSet(set, number, noise); // length is liberally extended here
      }
    }
    // complete
    return set;
  }

}
```