```java
package hyperDap.base.testHelpers;

import static org.junit.Assert.assertEquals;
import java.util.ArrayList;
import org.junit.jupiter.api.Test;
import hyperDap.base.types.dataSet.ValueDataSet;

public class TestCalcDerivDepth {

  private ValueDataSet<Double> makeSet() {
    return new ValueDataSet<Double>(0, 1, 0.001, d -> Double.valueOf(d));
  }

  // simple polynomials
  // *****************************************************************************************************************

  @Test
  void constant() {
    double value = 5.0;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(value);
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      assertEquals(0, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void linear() {
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + i);
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(1, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void square() {
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, 2));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(2, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void cubic() {
    int power = 3;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void quad() {
    int power = 4;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void polynom5() {
    int power = 5;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void polynom6() {
    int power = 6;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  // @Test
  void polynom7() {
    int power = 7;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  // @Test
  void polynom8() {
    int power = 8;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  // @Test // here the maxDepth is reached and Integer.MAX_VALUE is assigned
  void polynom9() {
    int power = 9;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    for (int i = 0; i < set.size(); i++) {
      // System.out.println(set.getDerivDepthsByIndex(i));
      assertEquals(Integer.MAX_VALUE, set.getDerivDepthsByIndex(i));
    }
  }

  // changes
  //
  ************************************************************************************
  ********************************

  @Test
  void constantToLinear() {
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0);
    }
    for (int i = 1; i < 51; i++) {
      set.add(5.0 + i);
    }
    set.calcDerivDepths();
    // System.out.println("\nderivDepths:");
    for (int i = 0; i < 49; i++) {
      // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
      assertEquals(0, set.getDerivDepthsByIndex(i));
    }
    // System.out.println(String.format("%s: %s", 49, set.getDerivDepthsByIndex(49)));
    assertEquals(-1, set.getDerivDepthsByIndex(49));
    for (int i = 50; i < set.size(); i++) {
      // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
      assertEquals(1, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void constantToSquare() {
    int power = 2;
    ValueDataSet<Double> set = makeSet();
    for (int i = 0; i < 50; i++) {
      set.add(5.0);
    }
    for (int i = 1; i < 51; i++) {
      set.add(5.0 + Math.pow(i, power));
    }
    set.calcDerivDepths();
    // System.out.println("\nderivDepths:");
    for (int i = 0; i < 49; i++) {
      // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
      assertEquals(0, set.getDerivDepthsByIndex(i));
    }
    // System.out.println(String.format("%s: %s", 49, set.getDerivDepthsByIndex(49)));
    assertEquals(-1, set.getDerivDepthsByIndex(49));
    for (int i = 50; i < set.size(); i++) {
      // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
      assertEquals(power, set.getDerivDepthsByIndex(i));
    }
  }

  @Test
  void squareToConstant() {
    int power = 2;
    ValueDataSet<Double> set = makeSet();
    ArrayList<Integer> depthsExpected = new ArrayList<Integer>();
    double temp = 0;
    for (int i = 0; i < 25; i++) {
      temp = Math.pow(i, power);
      set.add(temp);
      depthsExpected.add(power);
    }
    depthsExpected.remove(depthsExpected.size() - 1);
```

```java
      depthsExpected.add(-1);
      for (int i = 0; i < 25; i++) {
        set.add(temp);
        depthsExpected.add(0);
      }
      for (int i = 0; i < set.size(); i++) {
        // System.out.println(String.format("%s: %s ?= %s", i,
        set.getDerivDepthsByIndex(i),
        // depthsExpected.get(i)));
        assertEquals(depthsExpected.get(i).intValue(), set.getDerivDepthsByIndex(i));
      }
    }

    @Test
    void linearToSquare() {
      int power = 2;
      double base = 0.0;
      double step = 1.0;
      ValueDataSet<Double> set = makeSet();
      ArrayList<Integer> depthsExpected = new ArrayList<Integer>();
      double temp = 0;
      for (int i = 0; i < 25; i++) {
        temp = i;
        set.add(temp);
        depthsExpected.add(1);
      }
      depthsExpected.add(-1);
      for (int i = 1; i < 25; i++) {
        set.add(temp + Math.pow(i, power));
        depthsExpected.add(power);
      }
      for (int i = 0; i < set.size(); i++) {
        // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
        assertEquals(depthsExpected.get(i).intValue(), set.getDerivDepthsByIndex(i));
      }
    }

    @Test
    void biasInConstant() {
      ValueDataSet<Double> set = makeSet();
      for (int i = 0; i < 50; i++) {
        set.add(5.0);
      }
      for (int i = 1; i < 51; i++) {
        set.add(10.0);
      }
      set.calcDerivDepths();
      // System.out.println("\nderivDepths:");
      for (int i = 0; i < 49; i++) {
        // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
        assertEquals(0, set.getDerivDepthsByIndex(i));
      }
      // System.out.println(String.format("%s: %s", 49, set.getDerivDepthsByIndex(49)));
      assertEquals(-1, set.getDerivDepthsByIndex(49));
      for (int i = 50; i < set.size(); i++) {
        // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
        assertEquals(0, set.getDerivDepthsByIndex(i));
      }
    }

    // further polynomial
    //
    *************************************************************************************
    **********************************

    @Test
    void sqareWithinConstant() {
      int power = 2;
      double base = 0;
      double step = 1;
      double value = 5.0;
      ValueDataSet<Double> set = makeSet();
      for (int i = 0; i < 30; i++) {
        set.add(value);
      }
      double temp = 0;
      for (int i = 0; i < 15; i++) {
        temp = value + Math.pow(base + i * step, power);
        set.add(temp);
      }
      for (int i = 0; i < 30; i++) {
        set.add(temp);
      }
      for (int i = 0; i < 30; i++) {
        assertEquals(0, set.getDerivDepthsByIndex(i));
      }
      assertEquals(-1, set.getDerivDepthsByIndex(30));
      for (int i = 31; i < 30 + 14; i++) {
        assertEquals(power, set.getDerivDepthsByIndex(i));
      }
      assertEquals(-1, set.getDerivDepthsByIndex(30 + 14));
      for (int i = 30 + 15; i < set.size(); i++) {
        assertEquals(0, set.getDerivDepthsByIndex(i));
      }
      // for (int i = 0; i < set.size() - 10; i++) {
      // System.out.println(String.format("%s: %s", i, set.getDerivDepthsByIndex(i)));
      // }
    }

    @Test
    void squareOverZero() {
      int power = 2;
      double base = -10;
      double step = 0.1;
      ValueDataSet<Double> set = makeSet();
      for (int i = 0; i < 500; i++) {
        set.add(Math.pow(base + i * step, power));
      }
      for (int i = 0; i < set.size(); i++) {
        // System.out.println(set.getDerivDepthsByIndex(i));
        assertEquals(power, set.getDerivDepthsByIndex(i));
      }
    }

  }
  // end of class

```