Objective: practice using arrays in common programming tasks.

1. ArrayMath class

Create a public class named `ArrayMath` in package **`util`** containing the methods described below.

Include Javadoc comments for the class with your name (@author) and all method Javadoc.

Here is an example of Javadoc. You should try to write complete descriptions for each method. In BlueJ you can view your Javadoc as HTML (demo in class).

```
/**
 * Mathematical functions involving arrays.
 *
 * @author Cleve Molar  -- use your own name
 */
public class ArrayMath {
    /**
     * Compute and return the average of an array of values.
     * @param x array of double values. May have length 0.
     * @return the average of values in x. If array has
     *         length 0 then the average is 0.0.
     */
    public static double average(double[] x)
```

Methods

These are all **static** methods (just like methods in the Math class). You must decide what the return type should be.

`dotProduct(double[] x, double[] y)`	Returns the inner product (dot product) of two arrays. dot product is x[0]*y[0] + x[1]*y[1] + ... If the arrays have different lengths, then compute the inner product for the *smaller of the two* lengths. That is, ignore extra elements in the longer array. If an array has length 0, then the dotProduct is 0.
`average(double[] x)`	Returns the average value of elements in x. If x has length 0, the average is 0.
`norm(double[] x)`	Returns the Euclidean norm of an array. The Euclidean norm is the square-root of the sum of squares of elements. sqrt(x[0]*x[0] + x[1]*x[1]...) **Don't write duplicate code**: this method requires only one or two lines of code. :-)
`max(double[] x)`	Return the maximal element in x. If x is empty, return -Infinity (this is a constant in Double class).
`polyval(double x, double[] a)`	Return the value of a polynomial with coefficients in array a, evaluated at x. See details below.

average(double[ ] x)

Compute and return the average (mean) value of data in the x array using the formula:

$$\bar{x} = \frac{1}{n}\sum_k x_k$$ *where n is the length of the array.*

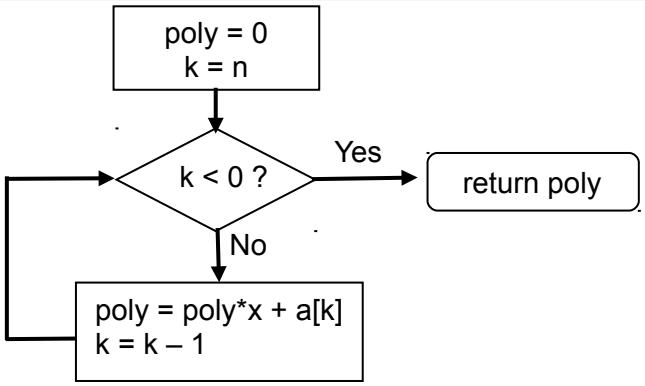
polyval(double x, double[ ] a)

Evaluate a polynomial of degree n at the given value of x. The coefficients of the polynomial are in the array a[ ], with a[k] as coefficient of x to the k-th power. If we let *p(x)* represent the polynomial, then

Formula for a polynomial $p(x)$:

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} \cdots + a_1 x + a_0$$
$$= (\cdots((a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0$$

the coefficient of x^k is array element **`a[k]`**.



Use the second formula for $p(x)$ in your Java code, as shown in the flowchart at right. This formula is faster and usually more accurate. It is faster because you don't need to compute powers if x. The flowchart can be implemented as an indexed **`for`** loop.

```
// syntax: for(initializer; test_condition; increment)
for(int k=n; k >= 0; k--) {
    body of loop
}
```

Notice the **space** around operators. This is the recommended coding style in both Java and Python. Don't write like this: `for(int k=n;k>=0;k--)`