```kotlin
package math

import Point
import math.InterpolationSolver.LeastSquaresType.*
import kotlin.math.E
import kotlin.math.ln
import kotlin.math.pow

class InterpolationSolver {

    /**
     * Возвращает интерполирующую функцию для заданного набора точек методом полинома Ньютона.
     * @param points набор точек по котором будет вычисляться функция.
     */
    internal fun newtonPolynomial(points: List<Point>): MathFunction<Double> {
        val order = points.size
        val finiteDiffs = Array(order) { DoubleArray(order) { 0.0 } }
        finiteDiffs[0] = points.map { it.y }.toDoubleArray() //Δy0 - нулевые конечные разности = значениям функции
        for (i in 1 until order)
            for (j in 0 until order - i)
                finiteDiffs[i][j] = finiteDiffs[i - 1][j + 1] - finiteDiffs[i - 1][j]
        val p0 = points[0]
        val h = points[1].x - p0.x
        return MathFunction<Double> { x ->
            var res = p0.y
            val q = (x[0]!! - p0.x) / h
            var product = 1.0
            for (i in 1 until order) {
                product = (product * (q + 1 - i)) / i
                res += product * finiteDiffs[i][0]
            }
            res
        }
    }

    /**
     * Находит аппроксимирующую функцию для заданного набора точек методом наименьших квадратов.
     * @property LINEAR линейная аппроксимация.
     * @property QUADRATIC квадратичная аппроксимация.
     * @property POW степенная аппроксимация.
     * @property EXPONENTIAL экспоненциальная аппроксимация.
     * @property LOGARITHMIC логарифмическая аппроксимация.
     */
    enum class LeastSquaresType {

        LINEAR {
            override fun approximate(points: List<Point>): MathFunction<Double> {
                val (a, b) = LeastSquaresType.getLinearApproxCoefs(points)
                return MathFunction<Double> { x -> a * x[0] + b }
            }
            override fun toString() = "Линейная"
        },

        QUADRATIC {
            override fun approximate(points: List<Point>): MathFunction<Double> {
                val sumX = points.map { it.x }.sum()
                val sumY = points.map { it.y }.sum()
                val sumXSquares = points.map { it.x.pow(2) }.sum()
                val sumXY = points.map { it.x * it.y }.sum()
                val sumXCubes = points.map { it.x.pow(3) }.sum()
                val sumXTesseracts = points.map { it.x.pow(4) }.sum()
                val sumXSquaresY = points.map { it.x.pow(2) * it.y }.sum()
                val n = points.size
                val matrix = arrayOf(doubleArrayOf(n.toDouble(), sumX, sumXSquares),
                    doubleArrayOf(sumX, sumXSquares, sumXCubes), doubleArrayOf(sumXSquares, sumXCubes, sumXTesseracts))
                val resVector = doubleArrayOf(sumY, sumXY, sumXSquaresY)
                var (a, b, c) = LinearSystemSolver.gaussian(matrix, resVector)
                a = c.also { c = a } //пришлось поменять, почему-то в неправильной последовательности возвращает
                return MathFunction<Double> { x -> a * x[0].pow(2) + b * x[0] + c }
            }
            override fun toString() = "Квадратичная"
        },

        POW {
            override fun approximate(points: List<Point>): MathFunction<Double> {
                val (x, y) = points.map { Pair(ln(it.x), ln(it.y)) }.toList().unzip()
                val modifiedPoints = x.zip(y) { xi, yi -> Point(xi, yi)  }.toList()
```

```
                val (b, a0) = getLinearApproxCoefs(modifiedPoints)
                val a = E.pow(a0)
                return MathFunction<Double> { x -> a * x[0].pow(b) }
            }
            override fun toString() = "Степенная"
        },

        EXPONENTIAL {
            override fun approximate(points: List<Point>): MathFunction<Double> {
                val (x, y) = points.map { Pair(it.x, ln(it.y)) }.toList().unzip()
                val modifiedPoints = x.zip(y) { xi, yi -> Point(xi, yi)  }.toList()
                val (b, a0) = getLinearApproxCoefs(modifiedPoints)
                val a = E.pow(a0)
                return MathFunction<Double> { x -> a * E.pow(b  * x[0]) }
            }
            override fun toString() = "Экспоненциальная"
        },

        LOGARITHMIC {
            override fun approximate(points: List<Point>): MathFunction<Double> {
                val (x, y) = points.map { Pair(ln(it.x), it.y) }.toList().unzip()
                val modifiedPoints = x.zip(y) { xi, yi -> Point(xi, yi)  }.toList()
                val (a, b) = getLinearApproxCoefs(modifiedPoints)
                return MathFunction<Double> { x-> a * ln(x[0]) + b }
            }
            override fun toString() = "Логарифмическая"
        };

        private companion object {
            fun getLinearApproxCoefs(points: List<Point>): Pair<Double, Double> {
                val sumX = points.map { it.x }.sum()
                val sumY = points.map { it.y }.sum()
                val sumXSquares = points.map { it.x.pow(2) }.sum()
                val sumXY = points.map { it.x * it.y }.sum()
                val n = points.size
                val matrix = arrayOf(doubleArrayOf(sumXSquares, sumX), doubleArrayOf(sumX, n.toDouble
    ()))
                val resVector = doubleArrayOf(sumXY, sumY)
                val (a, b) = LinearSystemSolver.gaussian(matrix, resVector)
                return Pair(a, b)
            }
        }

        /**
         * @return аппроксимирующую функцию для набора точек [points].
         */
        abstract fun approximate(points: List<Point>): MathFunction<Double>
    }
}
```