```kotlin
import kotlin.math.abs

/**
 * Содержит результат решения интеграла.
 * @property resValue значение интеграла.
 * @property infelicity погрешность вычислений.
 * @property blocks количество разбиений.
 * @version 1.0
 */
data class IntegralAnswer(val resValue: Double, val infelicity: Double, val blocks: Int)

/**
 * Находит численное значение интеграла разными методами.
 * @author Артемий Кульбако.
 * @version 1.0
 */
class IntegralSolver {

    /**
     * Константы, определяющие варианты решения методом прямоугольников.
     * @property LEFT метод левых прямоугольников.
     * @property CENTER метод средних прямоугольников.
     * @property RIGHT метод правых прямоугольников.
     */
    enum class RectangleMethodType { LEFT, CENTER, RIGHT }

    private interface ApproximationRule { fun findValue(step: Double, i: Int): Double }

    companion object {

        /**
         * Находит значение интеграла методом трапеций.
         * @param integral интеграл, который нужно посчитать.
         * @param precision точность вычислений.
         * @return IntegralAnswer.
         * @version 1.1
         */
        fun integrateByTrapezoid(integral: Integral, precision: Double): IntegralAnswer {
            val rule = object: ApproximationRule {
                override fun findValue(step: Double, i: Int) =
                    step * 0.5 * (integral.f.func(integral.limits.low + i * step) +
                            (integral.f.func(integral.limits.low + (i + 1) * step)))
            }
            return approximate(integral, precision, rule)
        }

        /**
         * Находит значение интеграла методом прямоугольников.
         * @param integral интеграл, который нужно посчитать.
         * @param precision точность вычислений.
         * @return IntegralAnswer.
         * @version 1.0
         */
        fun integrateByRectangle(integral: Integral, precision: Double, type: RectangleMethodType):
IntegralAnswer {
            val rule = when (type) {
                RectangleMethodType.LEFT -> object: ApproximationRule {
                    override fun findValue(step: Double, i: Int) =
                        step * integral.f.func(integral.limits.low + i * step)
                }
                RectangleMethodType.CENTER -> object: ApproximationRule {
                    override fun findValue(step: Double, i: Int) =
                        (step * integral.f.func(integral.limits.low + i * step) +
                            step * integral.f.func(integral.limits.low + (i + 1) * step)) / 2
                }
                RectangleMethodType.RIGHT -> object: ApproximationRule {
                    override fun findValue(step: Double, i: Int) =
                        step * integral.f.func(integral.limits.low + (i + 1) * step)
                }
            }
            return approximate(integral, precision, rule)
        }

        private fun approximate(integral: Integral, precision: Double, rule: ApproximationRule):
IntegralAnswer {
            fun findArea(integral: Integral, step: Double): Double {
                var area = 0.0
                for (i in 0 until ((integral.limits.high - integral.limits.low) / step).toInt()) {
                    area += rule.findValue(step, i)
                    if (area.isNaN() || area.isInfinite()) throw Exception("Функция не определена на
```

```
78 заданном отрезке.")
79                 }
80                 return area
81             }
82 
83             val limits = integral.limits
84             var step = limits.high - limits.low
85             var error: Double
86             var integralN: Double
87             var integral2N = findArea(integral, step)
88             do {
89                 integralN = integral2N
90                 step /= 2
91                 integral2N = findArea(integral, step)
92                 error = calcError(integral2N, integralN)
93             } while (error > precision)
94             if (limits.isSwitchedRange) integral2N = - integral2N
95             return IntegralAnswer(integral2N, error, ((limits.high - limits.low) / step).toInt())
96         }
97 
98         private fun calcError(integralN: Double, integral2N: Double) = abs(integral2N - integralN) / 3
99     }
100 }
```