```java
/* Features a function that prints the decimal value of a given integer value. */
public class IntegerToBinary {

    public static void main(String[] args) {
        integerToBinary(Integer.parseInt(args[0]));
        System.out.println();
    }

    public static void integerToBinary(int n) {
        if (n == 1 || n == 0) {
            System.out.print(n);
        } else {
            integerToBinary(n / 2);
            System.out.print(n % 2);
        }
    }
}
```

```java
/** Reads a command line string and checks if it's a palindrome. */
public class Palindrome {

    public static void main(String[] args) {
        System.out.println(isPalindrome(args[0]));
    }

    public static boolean isPalindrome(String s) {
        if (s.length() == 1 || s.length() == 0) {
            return true;
        } else if (s.charAt(0) == s.charAt(s.length() - 1)) {
            if (isPalindrome(s.substring(1, s.length() - 1))) {
                return true;
            }
        }
        return false;
    }
}
```

```java
/* Recieves two command line integers, n and k, and returns the respective binomial coefficent.
   Uses memoization to optimize the recursive process. */
public class Binomial {

    public static void main(String[] args) {
        System.out.println(binomial(Integer.parseInt(args[0]), Integer.parseInt(args[1])));
    }

    // Computes and returns the Binomial coefficient
    public static long binomial(int n, int k) {
        long[][] memo = new long[n + 1][k + 1];
        if (k > n) {
            return 0;
        }
        if (k == 0 || n == 0) {
            return 1;
        }
        return (binomial(n - 1, k, memo) + binomial(n - 1, k - 1, memo));
    }

    public static long binomial(int n, int k, long[][] memo) {
        if (k > n) {
            return 0;
        }
        if (k == 0 || n == 0) {
            return 1;
        }
        if (memo[n][k] == 0) {
            memo[n][k] = binomial(n - 1, k, memo) + binomial(n - 1, k - 1, memo);
        }
        return memo[n][k];
    }
}
```

```java
/** Prints ths Sierpinski Triangle fractal. */
public class Sierpinski {

    public static void main(String[] args) {
        sierpinski(Integer.parseInt(args[0]));
    }

    // Draws a Sierpinski triangle of depth n on the standard canvass.
    public static void sierpinski(int n) {
        double s = Math.sqrt(3) / 2;
        StdDraw.line(0, 0, 1, 0);
        StdDraw.line(0.5, s, 0, 0);
        StdDraw.line(1, 0, 0.5, s);
        sierpinski(n, 0, 1, 0.5, 0, 0, s);
    }

    public static void sierpinski(int n, double x1, double x2, double x3,
,           double y1, double y2, double y3) {
        if (n == 0) {
            return;
        }
        StdDraw.line((x1 + x2) / 2, (y1 + y2) / 2, (x1 + x3) / 2, (y1 + y3) / 2); // ! left
diagonal
        StdDraw.line((x1 + x2) / 2, (y1 + y2) / 2, (x2 + x3) / 2, (y1 + y3) / 2); // !
right line
        StdDraw.line((x1 + x3) / 2, (y1 + y3) / 2, (x2 + x3) / 2, (y1 + y3) / 2); // !
stright line
        sierpinski(n - 1, x1, (x1 + x2) / 2, (x1 + x3) / 2, y1, (y1 + y2) / 2, (y1 + y3) /
2);
        sierpinski(n - 1, (x1 + x2) / 2, x2, (x2 + x3) / 2, (y1 + y2) / 2, y2, (y2 + y3) /
2);
        sierpinski(n - 1, (x1 + x3) / 2, (x2 + x3) / 2, x3, (y1 + y3) / 2, (y2 + y3) / 2,
y3);
    }
}
```

```java
/** Prints ths Sierpinski Triangle fractal. */
public class Sierpinski {

    public static void main(String[] args) {
        sierpinski(Integer.parseInt(args[0]));
    }

    // Draws a Sierpinski triangle of depth n on the standard canvass.
    public static void sierpinski(int n) {
        double s = Math.sqrt(3) / 2;
        StdDraw.line(0, 0, 1, 0);
        StdDraw.line(0.5, s, 0, 0);
        StdDraw.line(1, 0, 0.5, s);
        sierpinski(n - 1, 0, 1, 0.5, 0, 0, s);
    }

    public static void sierpinski(int n, double x1, double x2, double x3,
,           double y1, double y2, double y3) {
        if (n == 0) {
            return;
        }
        StdDraw.line((x1 + x2) / 2, (y1 + y2) / 2, (x1 + x3) / 2, (y1 + y3) / 2); // ! left
diagonal
        StdDraw.line((x1 + x2) / 2, (y1 + y2) / 2, (x2 + x3) / 2, (y1 + y3) / 2); // ! right
line
        StdDraw.line((x1 + x3) / 2, (y1 + y3) / 2, (x2 + x3) / 2, (y1 + y3) / 2); // ! stright
line
        sierpinski(n - 1, x1, (x1 + x2) / 2, (x1 + x3) / 2, y1, (y1 + y2) / 2, (y1 + y3) / 2);
        sierpinski(n - 1, (x1 + x2) / 2, x2, (x2 + x3) / 2, (y1 + y2) / 2, y2, (y2 + y3) / 2);
        sierpinski(n - 1, (x1 + x3) / 2, (x2 + x3) / 2, x3, (y1 + y3) / 2, (y2 + y3) / 2, y3);
    }
}
```

```java
/** Draws the Koch curve and the the Koch snowflake fractal. */
public class Koch {

    public static void main(String[] args) {

        // Uncomment the first code block to test the curve function.
        // Uncomment the second code block to test the snowflake function.
        // Uncomment only one block in each test, and remember to compile
        // the class whenever you change the test.
        /*
         * // Tests the curve function:
         * // Gets n, x1, y1, x2, y2,
         * // and draws a Koch curve of depth n from (x1,y1) to (x2,y2).
         */
        // curve(Integer.parseInt(args[0]),
        // Double.parseDouble(args[1]), Double.parseDouble(args[2]),
        // Double.parseDouble(args[3]), Double.parseDouble(args[4]));

        /*
         * // Tests the snowflake function:
         * // Gets n, and draws a Koch snowflake of n edges in the standard canvass.
         */ snowFlake(Integer.parseInt(args[0]));
    }

    /**
     * Gets n, x1, y1, x2, y2,
     * and draws a Koch curve of depth n from (x1,y1) to (x2,y2).
     */
    public static void curve(int n, double x1, double y1, double x2, double y2) {
        if (n == 0) {
            return;
        }
        StdDraw.line(x1, y1, x2, y2);
        double p1x = x1 + (x2 - x1) / 3;
        double p1y = y1 + (y2 - y1) / 3;
        double p2x = x2 - (x2 - x1) / 3;
        double p2y = y2 - (y2 - y1) / 3;
        double p3x = ((Math.sqrt(3) / 6) * (y1 - y2)) + ((x1 + x2) / 2);
        double p3y = ((Math.sqrt(3) / 6) * (x2 - x1)) + ((y1 + y2) / 2);
        // * Drawing the new shape
        StdDraw.line(p1x, p1y, p3x, p3y);
        StdDraw.line(p3x, p3y, p2x, p2y);
        // * Removing the original segment
        StdDraw.setPenColor(StdDraw.WHITE);
        StdDraw.line(p1x, p1y, p2x, p2y);
        StdDraw.setPenColor(StdDraw.BLACK);
        curve(n - 1, x1, y1, p1x, p1y);
        curve(n - 1, p1x, p1y, p3x, p3y);
        curve(n - 1, p3x, p3y, p2x, p2y);
        curve(n - 1, p2x, p2y, x2, y2);
    }

    /** Gets n, and draws a Koch snowflake of n edges in the standard canvass. */
    public static void snowFlake(int n) {
        // A little tweak that makes the drawing look better
        StdDraw.setYscale(0, 1.1);
        StdDraw.setXscale(0, 1.1);
        // Draws a Koch snowflake of depth n
        curve(n, 0.0, 0.85, 1.0, 0.85);
        curve(n, 0.5, 0.0, 0.0, 0.85);
        curve(n, 1.0, 0.85, 0.5, 0.0);
    }
}
```