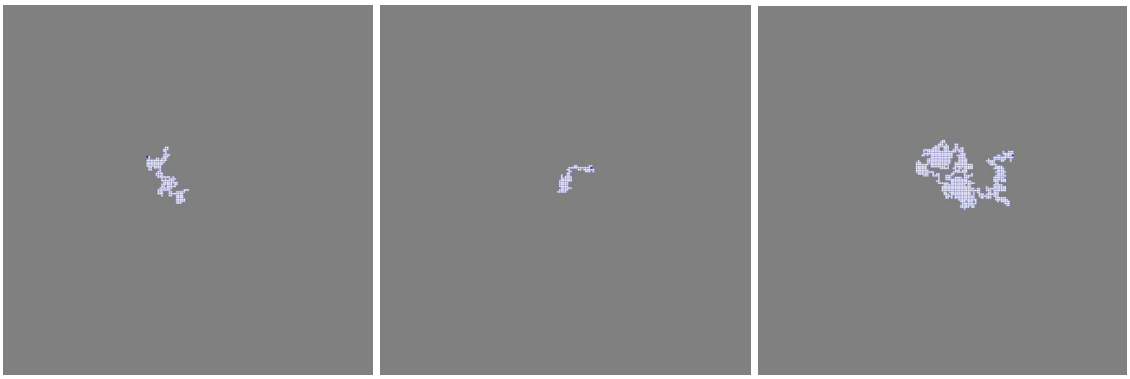
Ex 1: Use and adapt the code PowersOfTwo.java, to print the first 50 powers of 2^N. Include your code as well as the output result.

```
0 1
1 2
2 4
3 8
4 16
5 32
6 64
7 128
8 256
9 512
10 1024
11 2048
12 4096
13 8192
14 16384
15 32768
16 65536
17 131072
18 262144
19 524288
20 1048576
21 2097152
22 4194304
23 8388608
24 16777216
25 33554432
26 67108864
27 134217728
28 268435456
29 536870912
30 1073741824
31 2147483648
32 4294967296
33 8589934592
34 17179869184
35 34359738368
36 68719476736
37 137438953472
38 274877906944
39 549755813888
40 1099511627776
41 2199023255552
42 4398046511104
43 8796093022208
44 17592186044416
45 35184372088832
46 70368744177664
47 140737488355328
48 281474976710656
49 562949953421312
```

```
50 1125899906842624
```

R: Change type int to long: long powerOfTwo = 1;

Ex2: Use the code for RandomWalk.java to create 3 pictures that you like, using the number 100 as argument. To compile, you are required to previously compile StdDraw.java. You will produce 3 plots to be copied into your Activity log document.



Ex3: Use the code Factors.java that prints the prime factors of a number. Follow the examples in the code headings comments and you are required to measure the computation time for the next 6 cases: 3, 6, 9, 12, 15, and 18 digit primes

- java Factors 997
- java Factors 999983
- java Factors 999999937
- java Factors 999999999989
- java Factors 999999999999989
- java Factors 999999999999999989

```
The prime factorization of 997 is: 997
Tiempo de ejecución en nanoseg: 319754

The prime factorization of 999983 is: 999983
Tiempo de ejecución en nanoseg: 113690

The prime factorization of 999999937 is: 999999937
Tiempo de ejecución en nanoseg: 1587323

The prime factorization of 999999999989 is: 999999999989
Tiempo de ejecución en nanoseg: 18310868

The prime factorization of 999999999999989 is: 999999999999989
Tiempo de ejecución en nanoseg: 355648104
```

```
The prime factorization of 999999999999999989 is: 999999999999999989
Tiempo de ejecución en nanoseg: 10025784829
```

Ex 4: Use the program FunctionGrowth.java that prints a table of the values of *log N*, *N*, *N log N*, N^2, N^3, and 2^N for *N* = 16, 32, 64, ..., 2048. What are the limits of this code? Suppose we want to stop not at N=2048. but at N=1073741824. Modify your code to do this. Add the modified code to your document and include generated output.

```
log N  N      N log N      N^2    N^3
0      2.0    1      4.0    8.0
1      4.0    5      16.0   64.0
2      8.0    16     64.0   512.0
2      16.0   44     256.0  4096.0
3      32.0   110    1024.0 32768.0
4      64.0   266    4096.0 262144.0
4      128.0  621    16384.0       2097152.0
5      256.0  1419   65536.0       1.6777216E7
6      512.0  3194   262144.0      1.34217728E8
6      1024.0 7097   1048576.0     1.073741824E9
7      2048.0 15615  4194304.0     8.589934592E9
8      4096.0 34069  1.6777216E7   6.8719476736E10
9      8192.0 73817  6.7108864E7   5.49755813888E11
9      16384.0       158991 2.68435456E8 4.398046511104E12
10     32768.0       340695 1.073741824E93.5184372088832E13
11     65536.0       726817 4.294967296E92.81474976710656E14
11     131072.0      1544487       1.7179869184E10      2.251799813685248E15
12     262144.0      3270678       6.8719476736E10      1.8014398509481984E16
13     524288.0      6904766       2.74877906944E11     1.44115188075855872E17
13     1048576.0     14536349      1.099511627776E12    1.15292150460684698E18
14     2097152.0     30526334      4.398046511104E12    9.223372036854776E18
15     4194304.0     63959939      1.7592186044416E13   7.378697629483821E19
15     8388608.0     133734419     7.0368744177664E13   5.9029581035870565E20
16     1.6777216E7   279097919     2.81474976710656E14 4.722366482869645E21
17     3.3554432E7   581453998     1.125899906842624E15       3.777893186295716E22
18     6.7108864E7   1209424316    4.503599627370496E15       3.022314549036573E23
18     1.34217728E8  2147483647    1.8014398509481984E16      2.417851639229258 3E24
19     2.68435456E8  2147483647    7.2057594037927936E16      1.9342813113834067E25
20     5.36870912E8  2147483647    2.8823037615171174E17      1.5474250491067253E26
20     1.073741824E92147483647     1.15292150460684698E18     1.2379400392853803E27
```

Ex5: Modify the code Binary.java that converts any number to binary form, to convert any number to its hexadecimal form. Print the first 256 numbers in hex. Include code and output in your working document.

```
0 - 0
1 - 1
2 - 10
3 - 11
4 - 100
5 - 101
6 - 110
7 - 111
8 - 1000
```

9 - 1001
10 - 1010
11 - 1011
12 - 1100
13 - 1101
14 - 1110
15 - 1111
16 - 10000
17 - 10001
18 - 10010
19 - 10011
20 - 10100
21 - 10101
22 - 10110
23 - 10111
24 - 11000
25 - 11001
26 - 11010
27 - 11011
28 - 11100
29 - 11101
30 - 11110
31 - 11111
32 - 100000
33 - 100001
34 - 100010
35 - 100011
36 - 100100
37 - 100101
38 - 100110
39 - 100111
40 - 101000
41 - 101001
42 - 101010
43 - 101011
44 - 101100
45 - 101101
46 - 101110
47 - 101111
48 - 110000
49 - 110001
50 - 110010
51 - 110011
52 - 110100
53 - 110101
54 - 110110
55 - 110111
56 - 111000
57 - 111001
58 - 111010
59 - 111011
60 - 111100
61 - 111101
62 - 111110
63 - 111111

64 - 1000000
65 - 1000001
66 - 1000010
67 - 1000011
68 - 1000100
69 - 1000101
70 - 1000110
71 - 1000111
72 - 1001000
73 - 1001001
74 - 1001010
75 - 1001011
76 - 1001100
77 - 1001101
78 - 1001110
79 - 1001111
80 - 1010000
81 - 1010001
82 - 1010010
83 - 1010011
84 - 1010100
85 - 1010101
86 - 1010110
87 - 1010111
88 - 1011000
89 - 1011001
90 - 1011010
91 - 1011011
92 - 1011100
93 - 1011101
94 - 1011110
95 - 1011111
96 - 1100000
97 - 1100001
98 - 1100010
99 - 1100011
100 - 1100100
101 - 1100101
102 - 1100110
103 - 1100111
104 - 1101000
105 - 1101001
106 - 1101010
107 - 1101011
108 - 1101100
109 - 1101101
110 - 1101110
111 - 1101111
112 - 1110000
113 - 1110001
114 - 1110010
115 - 1110011
116 - 1110100
117 - 1110101
118 - 1110110

119 - 1110111
120 - 1111000
121 - 1111001
122 - 1111010
123 - 1111011
124 - 1111100
125 - 1111101
126 - 1111110
127 - 1111111
128 - 10000000
129 - 10000001
130 - 10000010
131 - 10000011
132 - 10000100
133 - 10000101
134 - 10000110
135 - 10000111
136 - 10001000
137 - 10001001
138 - 10001010
139 - 10001011
140 - 10001100
141 - 10001101
142 - 10001110
143 - 10001111
144 - 10010000
145 - 10010001
146 - 10010010
147 - 10010011
148 - 10010100
149 - 10010101
150 - 10010110
151 - 10010111
152 - 10011000
153 - 10011001
154 - 10011010
155 - 10011011
156 - 10011100
157 - 10011101
158 - 10011110
159 - 10011111
160 - 10100000
161 - 10100001
162 - 10100010
163 - 10100011
164 - 10100100
165 - 10100101
166 - 10100110
167 - 10100111
168 - 10101000
169 - 10101001
170 - 10101010
171 - 10101011
172 - 10101100
173 - 10101101

174 - 10101110
175 - 10101111
176 - 10110000
177 - 10110001
178 - 10110010
179 - 10110011
180 - 10110100
181 - 10110101
182 - 10110110
183 - 10110111
184 - 10111000
185 - 10111001
186 - 10111010
187 - 10111011
188 - 10111100
189 - 10111101
190 - 10111110
191 - 10111111
192 - 11000000
193 - 11000001
194 - 11000010
195 - 11000011
196 - 11000100
197 - 11000101
198 - 11000110
199 - 11000111
200 - 11001000
201 - 11001001
202 - 11001010
203 - 11001011
204 - 11001100
205 - 11001101
206 - 11001110
207 - 11001111
208 - 11010000
209 - 11010001
210 - 11010010
211 - 11010011
212 - 11010100
213 - 11010101
214 - 11010110
215 - 11010111
216 - 11011000
217 - 11011001
218 - 11011010
219 - 11011011
220 - 11011100
221 - 11011101
222 - 11011110
223 - 11011111
224 - 11100000
225 - 11100001
226 - 11100010
227 - 11100011
228 - 11100100

```
229 - 11100101
230 - 11100110
231 - 11100111
232 - 11101000
233 - 11101001
234 - 11101010
235 - 11101011
236 - 11101100
237 - 11101101
238 - 11101110
239 - 11101111
240 - 11110000
241 - 11110001
242 - 11110010
243 - 11110011
244 - 11110100
245 - 11110101
246 - 11110110
247 - 11110111
248 - 11111000
249 - 11111001
250 - 11111010
251 - 11111011
252 - 11111100
253 - 11111101
254 - 11111110
255 - 11111111
256 – 100000000
```

Ex 6: Modify the code DayOfWeek.java to print the Day of the Week (Sunday, Monday, ...).

```
Thursday
```

Ex 7: Let's play cards. Use the code Deal.java to play 21 or BlackJack for 2 users. You are always the first deal of cards, the house the second. Modify the code to ask for an additional card (Hit=1) or none (Stay=0) for the user. In 20 trials, how many times did you beat the house?. Add the modified code to your working document and describe your experience.

```
public class Deal {
    public static void main(String[] args) {
        int CARDS_PER_PLAYER = 5;
        int more = 1;

        Scanner console = new Scanner(System.in);
        System.out.print("Number of players: ");
        int PLAYERS = console.nextInt();

        String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };
        String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q",
"K", "A" };
        // avoid hardwired constants
        int SUITS = suit.length;
        int RANKS = rank.length;
        int CARDS = SUITS * RANKS;
```

```
        if (CARDS_PER_PLAYER * PLAYERS > CARDS) throw new RuntimeException("Too many
players");

        // initialize deck
        String[] deck = new String[CARDS];
        for (int i = 0; i < RANKS; i++) {
            for (int j = 0; j < SUITS; j++) {
                deck[SUITS*i + j] = rank[i] + " of " + suit[j];
            }
        }

        // shuffle
        for (int i = 0; i < CARDS; i++) {
            int r = i + (int) (Math.random() * (CARDS-i));
            String t = deck[r];
            deck[r] = deck[i];
            deck[i] = t;
        }

        // print player deck
        int j = 0;
        while (more == 1)
        {
             System.out.println(deck[j] + " Player ");
                Scanner answer = new Scanner(System.in);
                System.out.print("Other Card  (1 to Yes) (0 to No)");
                more = answer.nextInt();
                j++;
        }

        System.out.println();
        Scanner dealer = new Scanner(System.in);
        System.out.print("Number of cards for dealer");
        int moredealer = dealer.nextInt();

        // print shuffled deck
        for (int i = 0; i <  moredealer; i++) {
            System.out.println(deck[i]);
        }
    }

}
I win 8 times
```

Ex 8: Use the code Birthday.java, to run at least 20 experiments and compute the average number of people needed to show up in a room in order that 2 people share the same birthday.

```
53
12
23
4
20
52
35
29
50
```

```
31
14
9
16
5
38
4
24
19
43
6
Average: 24.0
```

Ex 9: Use the code to build the Pascal triangle, Pascal.java. Produce a Pascal Triangle to level 10

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

Enter 11 as arg.
No changes in code.

Ex 10: You are required to run the code that generates a Sierpinski triangle: Sierpinski.java. This code requires compiling beforehand DrawingPanel.java. Can you guess an algorithm that counts how many solid black inverted triangles and how many upright white triangles per level N. Justify your answer.

```
black = Math.pow(3, n);
white =(black - 1)/2;
```