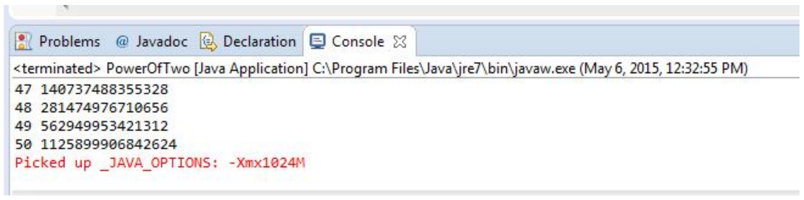
1. Use and adapt the code PowersOfTwo.java, to print the first 50 powers of 2^N. Include your code as well as the output result.

```
/*************************************************************************
 *  Compilation:  javac PowersOfTwo.java
 *  Execution:    java PowersOfTwo N
 *
 *  This program takes a command-line argument N and prints a table of
 *  the powers of 2 that are less than or equal to 2^N.
 *
*  Remarks
 *  ------------
 *  Only works if 0 <= N < 63 since 2^63-1 overflows an Long.
 *
 *************************************************************************/

public class PowersOfTwo {
   public static void main(String[] args) {

      // read in one command-line argument
      int N = Integer.parseInt(args[0]);

      int i = 0;                // count from 0 to N
      long powerOfTwo = 1;      // change to long in order to handle the first 62 powers
      // repeat until i equals N
      while (i <= N) {
         System.out.println(i + " " + powerOfTwo);   // print out the power of two
         powerOfTwo = 2 * powerOfTwo;                // double to get the next one
         i = i + 1;
      }

   }
}
```

Datatype changed on variable powerOfTwo fro integer to long in order to handle the first 62 powers.

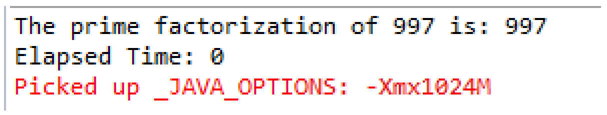


```
<terminated> PowerOfTwo [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 6, 2015, 12:32:55 PM)
47 140737488355328
48 281474976710656
49 562949953421312
50 1125899906842624
Picked up _JAVA_OPTIONS: -Xmx1024M
```

3. 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

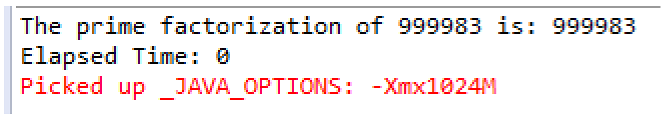
Results in milliseconds

java Factors 997



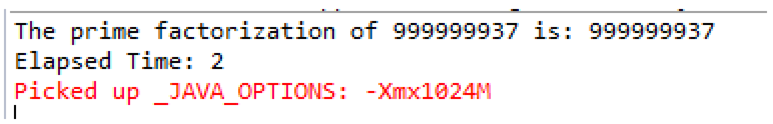
```
The prime factorization of 997 is: 997
Elapsed Time: 0
Picked up _JAVA_OPTIONS: -Xmx1024M
```

java Factors 999983



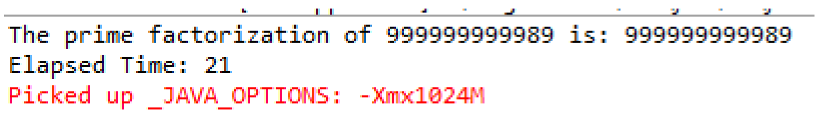
```
The prime factorization of 999983 is: 999983
Elapsed Time: 0
Picked up _JAVA_OPTIONS: -Xmx1024M
```

java Factors 999999937



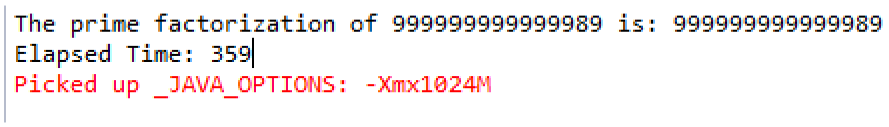
```
The prime factorization of 999999937 is: 999999937
Elapsed Time: 2
Picked up _JAVA_OPTIONS: -Xmx1024M
```

java Factors 999999999989



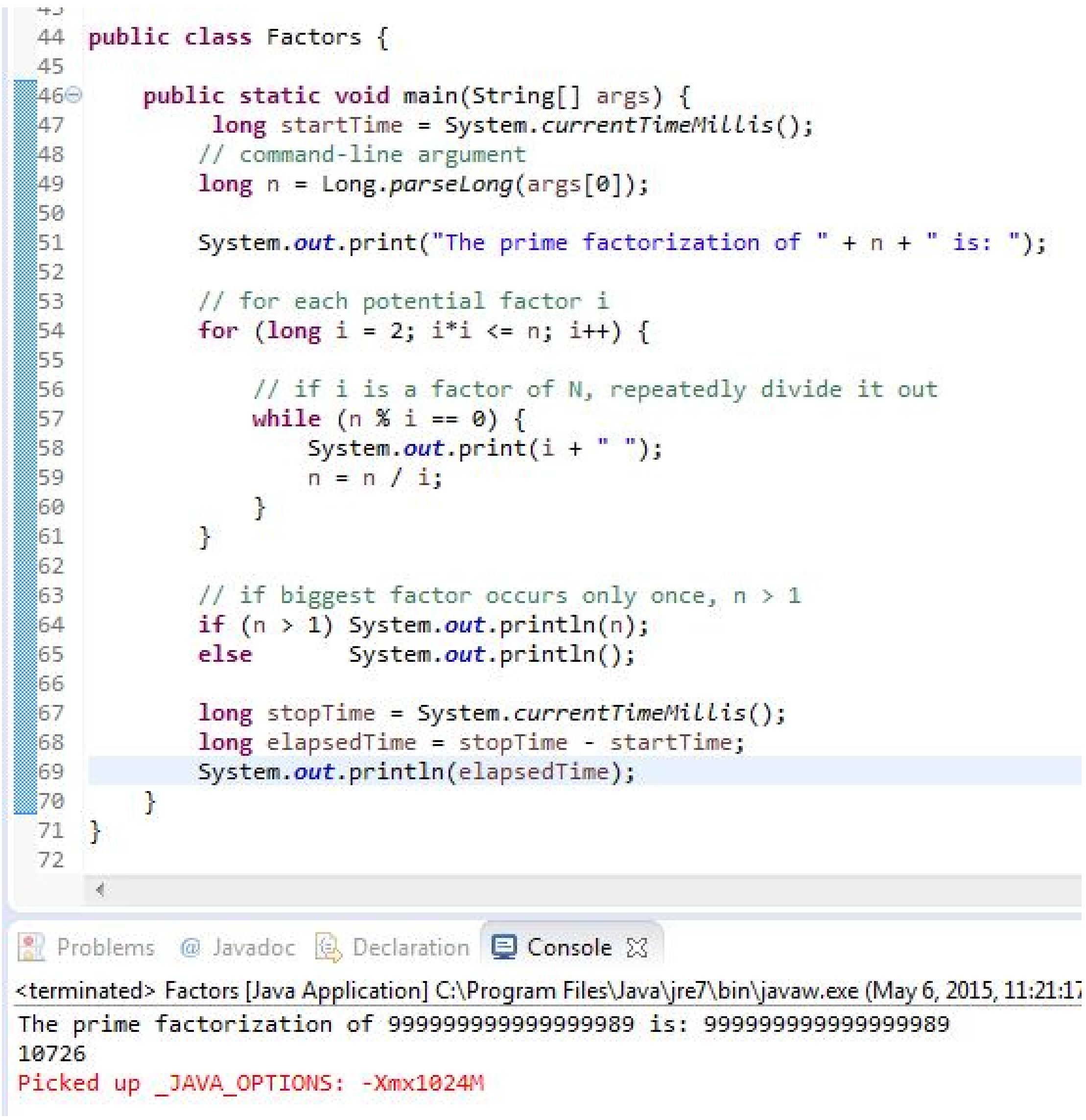
```
The prime factorization of 999999999989 is: 999999999989
Elapsed Time: 21
Picked up _JAVA_OPTIONS: -Xmx1024M
```

java Factors 999999999999989



```
The prime factorization of 999999999999989 is: 999999999999989
Elapsed Time: 359
Picked up _JAVA_OPTIONS: -Xmx1024M
```

java Factors 9999999999999999989

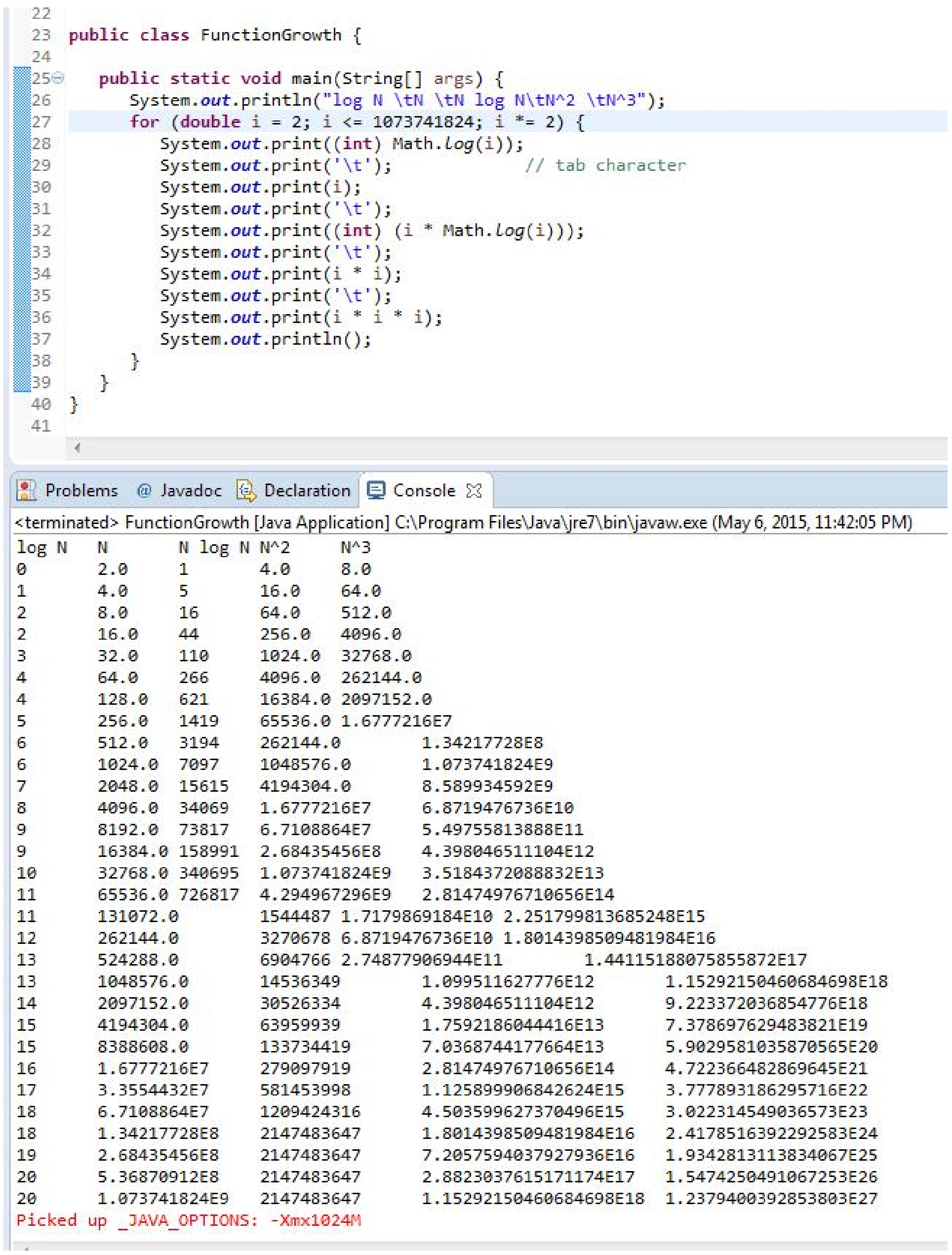


```
public class Factors {

    public static void main(String[] args) {
         long startTime = System.currentTimeMillis();
        // command-line argument
        long n = Long.parseLong(args[0]);

        System.out.print("The prime factorization of " + n + " is: ");

        // for each potential factor i
        for (long i = 2; i*i <= n; i++) {

            // if i is a factor of N, repeatedly divide it out
            while (n % i == 0) {
                System.out.print(i + " ");
                n = n / i;
            }
        }

        // if biggest factor occurs only once, n > 1
        if (n > 1) System.out.println(n);
        else       System.out.println();

        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;
        System.out.println(elapsedTime);
    }
}
```

```
<terminated> Factors [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 6, 2015, 11:21:17
The prime factorization of 999999999999999989 is: 999999999999999989
10726
Picked up _JAVA_OPTIONS: -Xmx1024M
```

4. Use the program FunctionGrowth.java that prints a table of the values of log N, N, N log N, N2, N3, and 2N 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.

Limits, variable printed out is a long it will overflow, to handle a full table for 1073741824 you need to change variable to double.

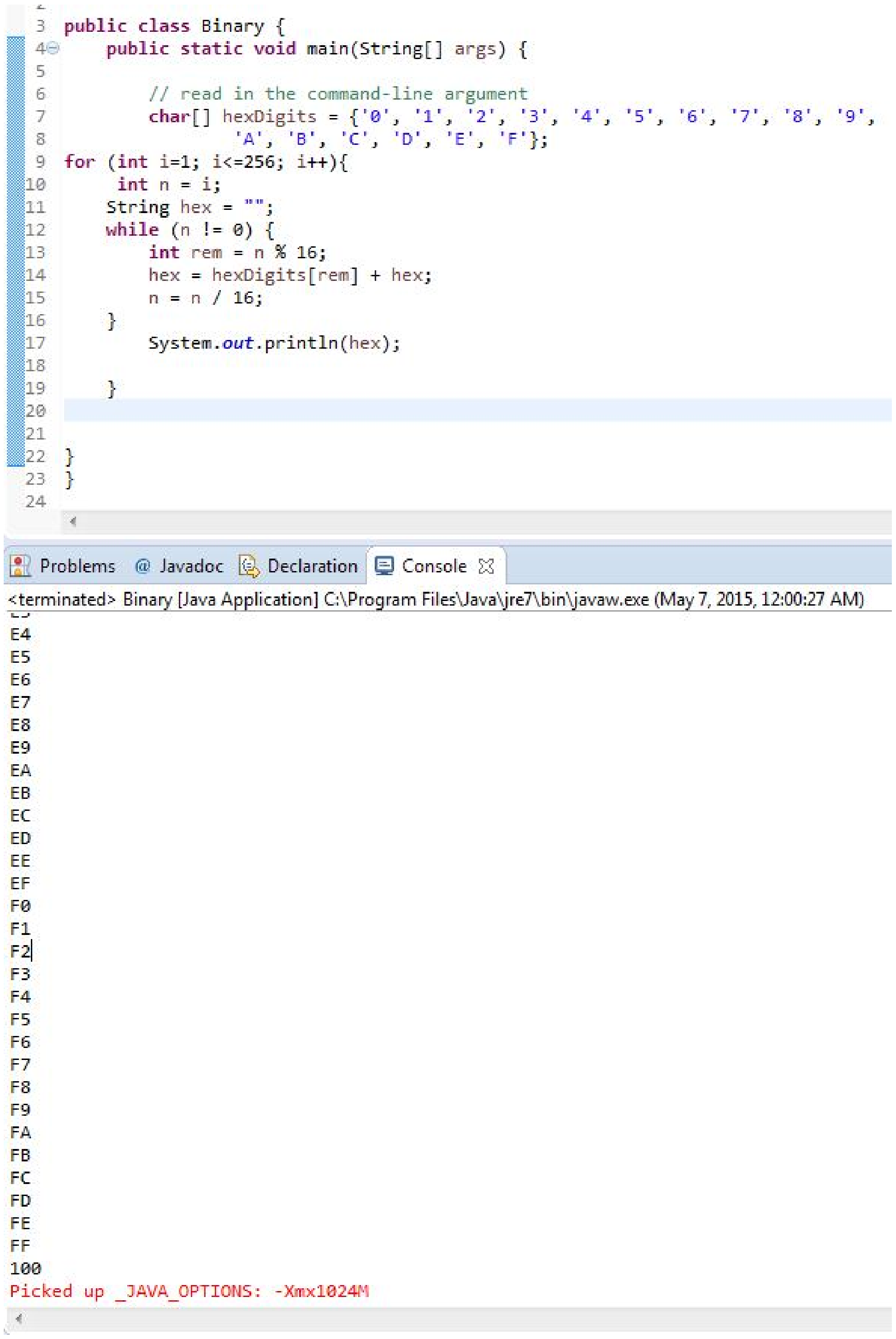


```java

public class FunctionGrowth {

    public static void main(String[] args) {
        System.out.println("log N \tN \tN log N\tN^2 \tN^3");
        for (double i = 2; i <= 1073741824; i *= 2) {
            System.out.print((int) Math.log(i));
            System.out.print('\t');             // tab character
            System.out.print(i);
            System.out.print('\t');
            System.out.print((int) (i * Math.log(i)));
            System.out.print('\t');
            System.out.print(i * i);
            System.out.print('\t');
            System.out.print(i * i * i);
            System.out.println();
        }
    }
}

```

```
<terminated> FunctionGrowth [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 6, 2015, 11:42:05 PM)
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  1.073741824E9   3.5184372088832E13
11      65536.0 726817  4.294967296E9   2.81474976710656E14
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      5.902958103587057E20
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   2.4178516392292583E24
19      2.68435456E8    2147483647      7.2057594037927936E16   1.9342813113834067E25
20      5.36870912E8    2147483647      2.8823037615171174E17   1.5474250491067253E26
20      1.073741824E9   2147483647      1.15292150460684698E18  1.2379400392853803E27
Picked up _JAVA_OPTIONS: -Xmx1024M
```

I found that for handling big numbers, more than double, you can use special Java class: BigInteger, I didn’t know how to implement on this example.

5. 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.

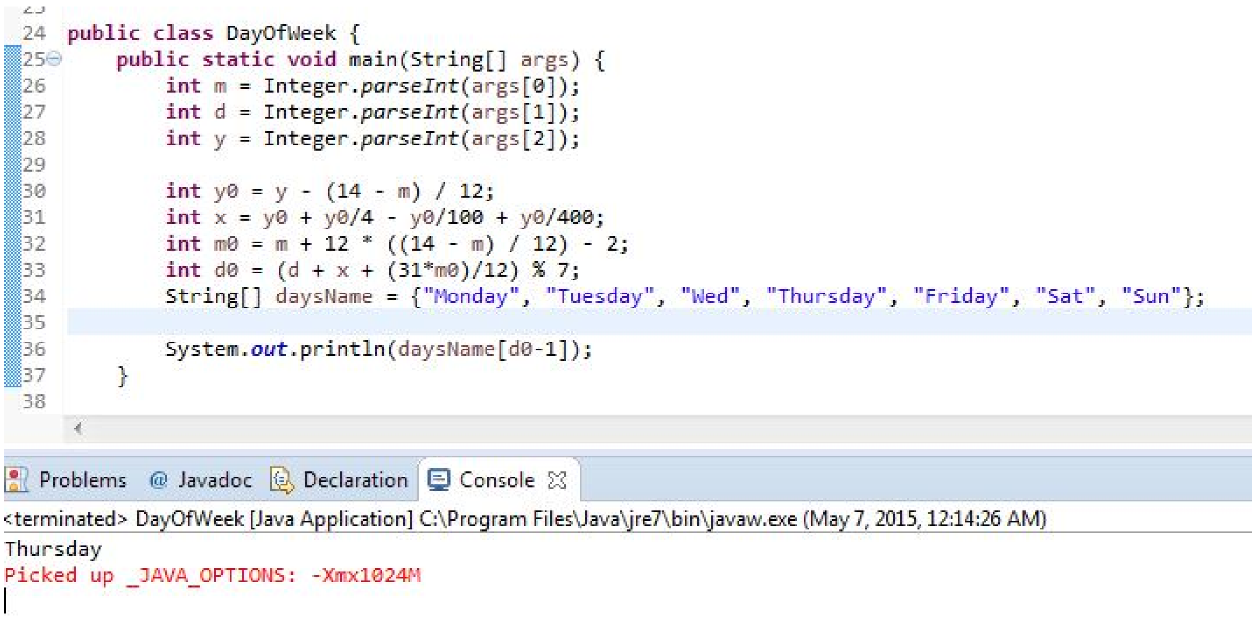


```
public class Binary {
    public static void main(String[] args) {

        // read in the command-line argument
        char[] hexDigits = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                'A', 'B', 'C', 'D', 'E', 'F'};
for (int i=1; i<=256; i++){
     int n = i;
    String hex = "";
    while (n != 0) {
        int rem = n % 16;
        hex = hexDigits[rem] + hex;
        n = n / 16;
    }
        System.out.println(hex);

    }


}
}
```

```
<terminated> Binary [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 7, 2015, 12:00:27 AM)
E4
E5
E6
E7
E8
E9
EA
EB
EC
ED
EE
EF
F0
F1
F2
F3
F4
F5
F6
F7
F8
F9
FA
FB
FC
FD
FE
FF
100
Picked up _JAVA_OPTIONS: -Xmx1024M
```

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

String array added with names of days, the printed out will be the d0 variable minus 1 as index of the String array.



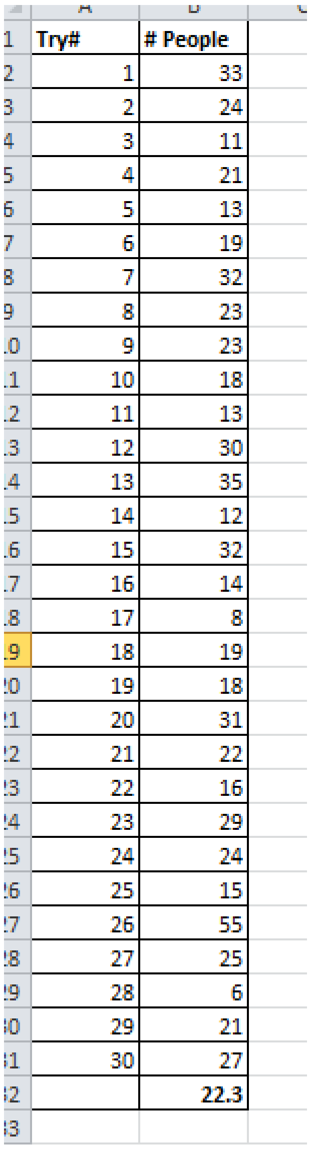
```java
public class DayOfWeek {
    public static void main(String[] args) {
        int m = Integer.parseInt(args[0]);
        int d = Integer.parseInt(args[1]);
        int y = Integer.parseInt(args[2]);

        int y0 = y - (14 - m) / 12;
        int x = y0 + y0/4 - y0/100 + y0/400;
        int m0 = m + 12 * ((14 - m) / 12) - 2;
        int d0 = (d + x + (31*m0)/12) % 7;
        String[] daysName = {"Monday", "Tuesday", "Wed", "Thursday", "Friday", "Sat", "Sun"};

        System.out.println(daysName[d0-1]);
    }
```

```
<terminated> DayOfWeek [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (May 7, 2015, 12:14:26 AM)
Thursday
Picked up _JAVA_OPTIONS: -Xmx1024M
```

The above result is the day for 5 7 2015

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.

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.



Try#	# People
1	33
2	24
3	11
4	21
5	13
6	19
7	32
8	23
9	23
10	18
11	13
12	30
13	35
14	12
15	32
16	14
17	8
18	19
19	18
20	31
21	22
22	16
23	29
24	24
25	15
26	55
27	25
28	6
29	21
30	27
	22.3

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

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.