# Solution of problems on Bitwise operations

1. Write a Java program to count the number of bits that are set to 1 in an integer using operators.

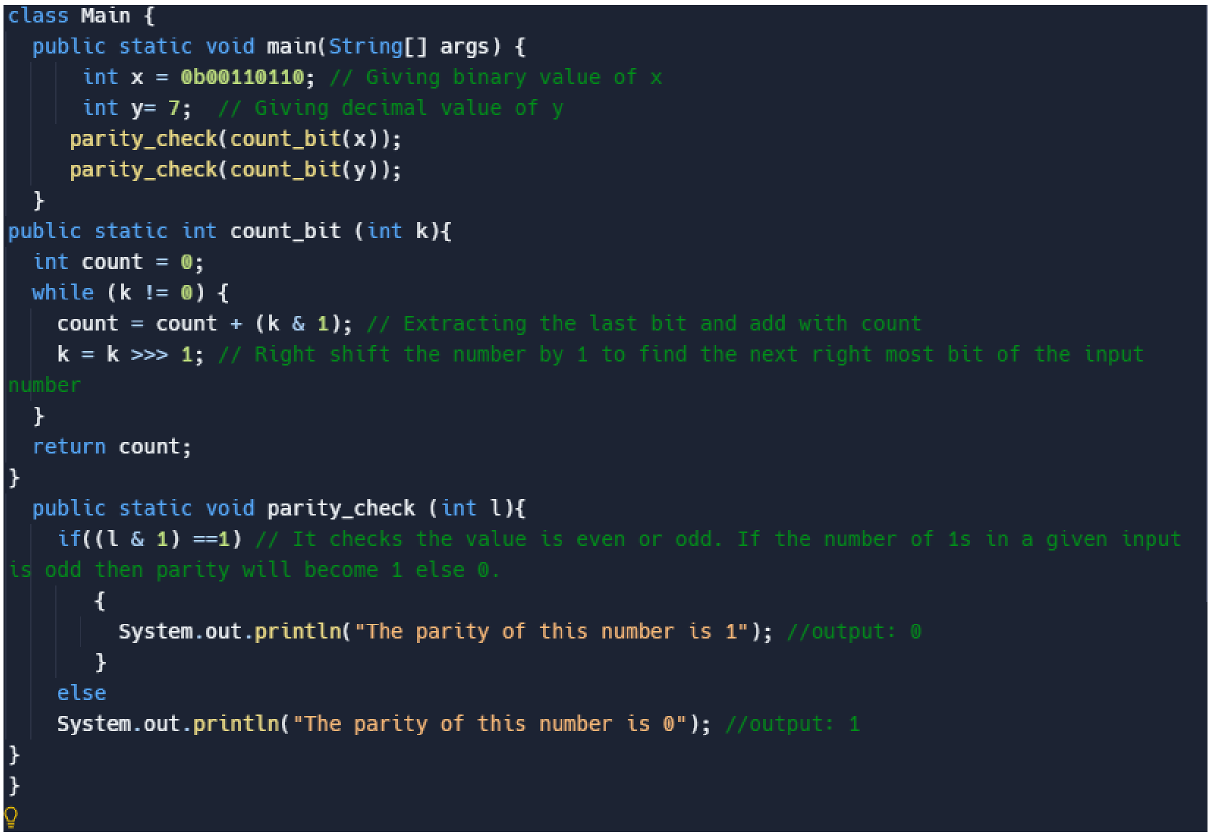
```
class Main {
  public static void main(String[] args) {
      int x = 0b00110110; // Giving binary value of x
      int y= 6;  // Giving decimal value of y
      System.out.println("The number of bits is:" + count_bit(x)); //output: 4
      System.out.println("The number of bits is:" + count_bit(y)); //output: 2

  }
public static int count_bit (int k){
  int count = 0;
  while (k != 0) {
    count = count + (k & 1); // Extracting the last bit and add with count
    k = k >>> 1; // Right shift the number by 1 to find the next right most bit of the input number
  }
  return count;
}
}
```

2. The parity of a binary word is 1 if the number of 1s in the word is odd; otherwise, it is 0. Write a Java program to count the parity of an integer number using bitwise operators.

   Method 1 of parity check:

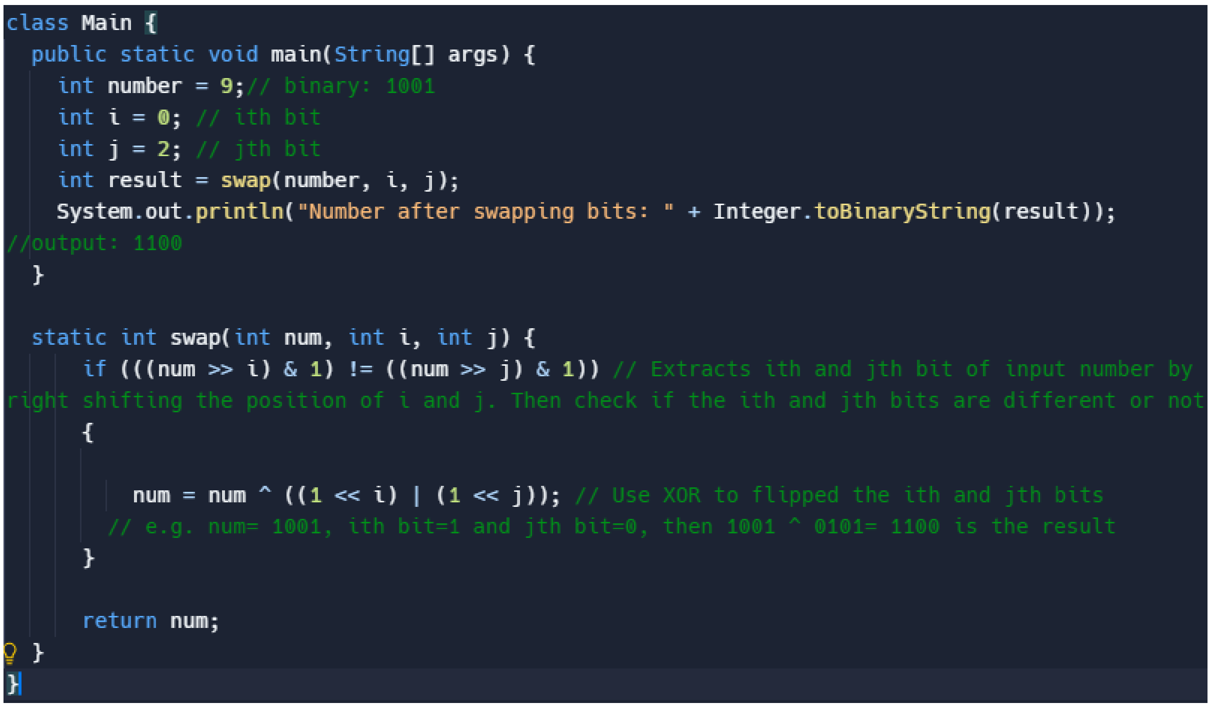
```
class Main {
  public static void main(String[] args) {
    int number1 = 0b110101; // Binary representation of the number
    int number2= 7;
    System.out.println("Parity of the number1 is: " +  countParity(number1)); //output: 0
    System.out.println("Parity of the number2 is: " +  countParity(number2)); //oputput: 1
  }
  static int countParity(int num) {
    int count = 0;

    while (num != 0) {
        count = count ^ (num & 1); //Each bit is checked if it is first 1 then count will become 1. If it is secind 1 then count will become 0 and so on. Finally, when we reach at the left most bit, if the input number contains odd numbers of 1 then count will become 1, otherwise count will become 0.  Here XOR operation is used to check and toggle parity.
        num >>>= 1; // Right shift to move to the next bit
    }

    return count;
}
}
```

   Method 2 of parity check:



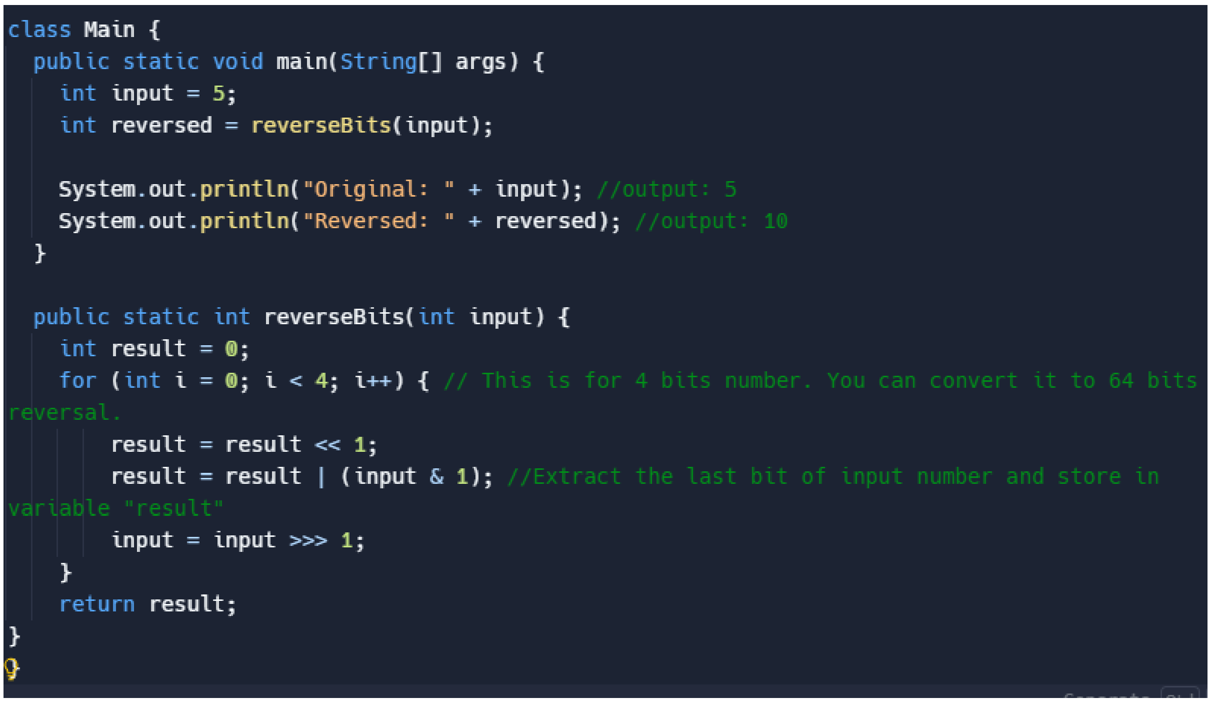
```java
class Main {
  public static void main(String[] args) {
      int x = 0b00110110; // Giving binary value of x
      int y= 7;  // Giving decimal value of y
     parity_check(count_bit(x));
     parity_check(count_bit(y));
  }
public static int count_bit (int k){
  int count = 0;
  while (k != 0) {
    count = count + (k & 1); // Extracting the last bit and add with count
    k = k >>> 1; // Right shift the number by 1 to find the next right most bit of the input
number
  }
  return count;
}
  public static void parity_check (int l){
    if((l & 1) ==1) // It checks the value is even or odd. If the number of 1s in a given input
is odd then parity will become 1 else 0.
       {
         System.out.println("The parity of this number is 1"); //output: 0
       }
    else
    System.out.println("The parity of this number is 0"); //output: 1
}
}
```

3. Write a program to swap the ith bit with jth bit of a number using bit wise operations.



```java
class Main {
  public static void main(String[] args) {
    int number = 9;// binary: 1001
    int i = 0; // ith bit
    int j = 2; // jth bit
    int result = swap(number, i, j);
    System.out.println("Number after swapping bits: " + Integer.toBinaryString(result));
//output: 1100
  }

  static int swap(int num, int i, int j) {
      if (((num >> i) & 1) != ((num >> j) & 1)) // Extracts ith and jth bit of input number by
right shifting the position of i and j. Then check if the ith and jth bits are different or not
      {

          num = num ^ ((1 << i) | (1 << j)); // Use XOR to flipped the ith and jth bits
        // e.g. num= 1001, ith bit=1 and jth bit=0, then 1001 ^ 0101= 1100 is the result
      }

      return num;
  }
}
```

4. Write a program that takes a 64-bit word and returns the 64-bit word consisting of the bits of the input word in reverse order. For example, if the input is alternating 1s and 0s, i.e., (1010...10), the output should be alternating 0s and 1s, i.e., (0101...01).

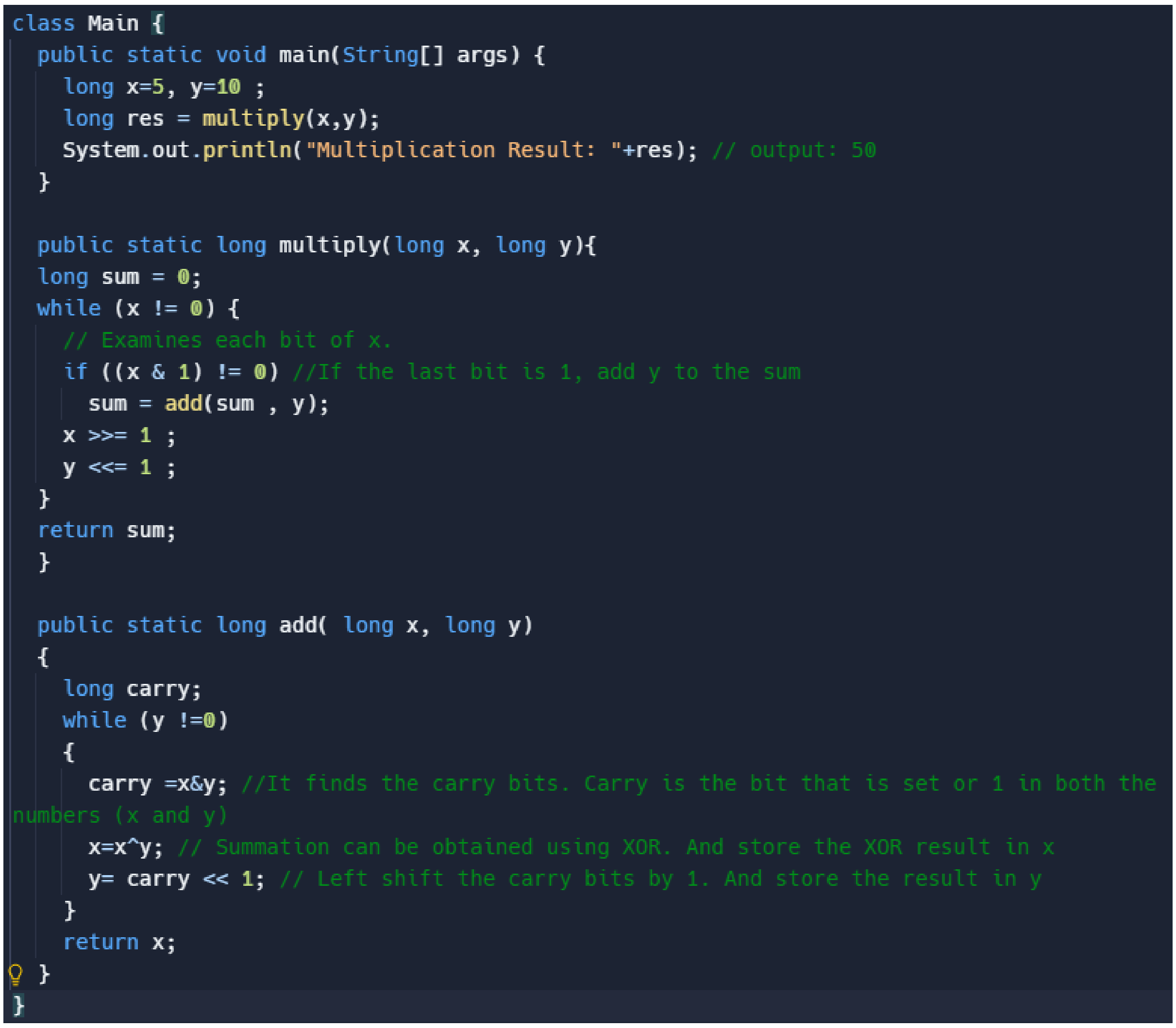


```java
class Main {
  public static void main(String[] args) {
    int input = 5;
    int reversed = reverseBits(input);

    System.out.println("Original: " + input); //output: 5
    System.out.println("Reversed: " + reversed); //output: 10
  }

  public static int reverseBits(int input) {
    int result = 0;
    for (int i = 0; i < 4; i++) { // This is for 4 bits number. You can convert it to 64 bits
reversal.
        result = result << 1;
        result = result | (input & 1); //Extract the last bit of input number and store in
variable "result"
        input = input >>> 1;
    }
    return result;
}
}
```

5. Write a java program to compute x × y without arithmetic operators.

Solution:

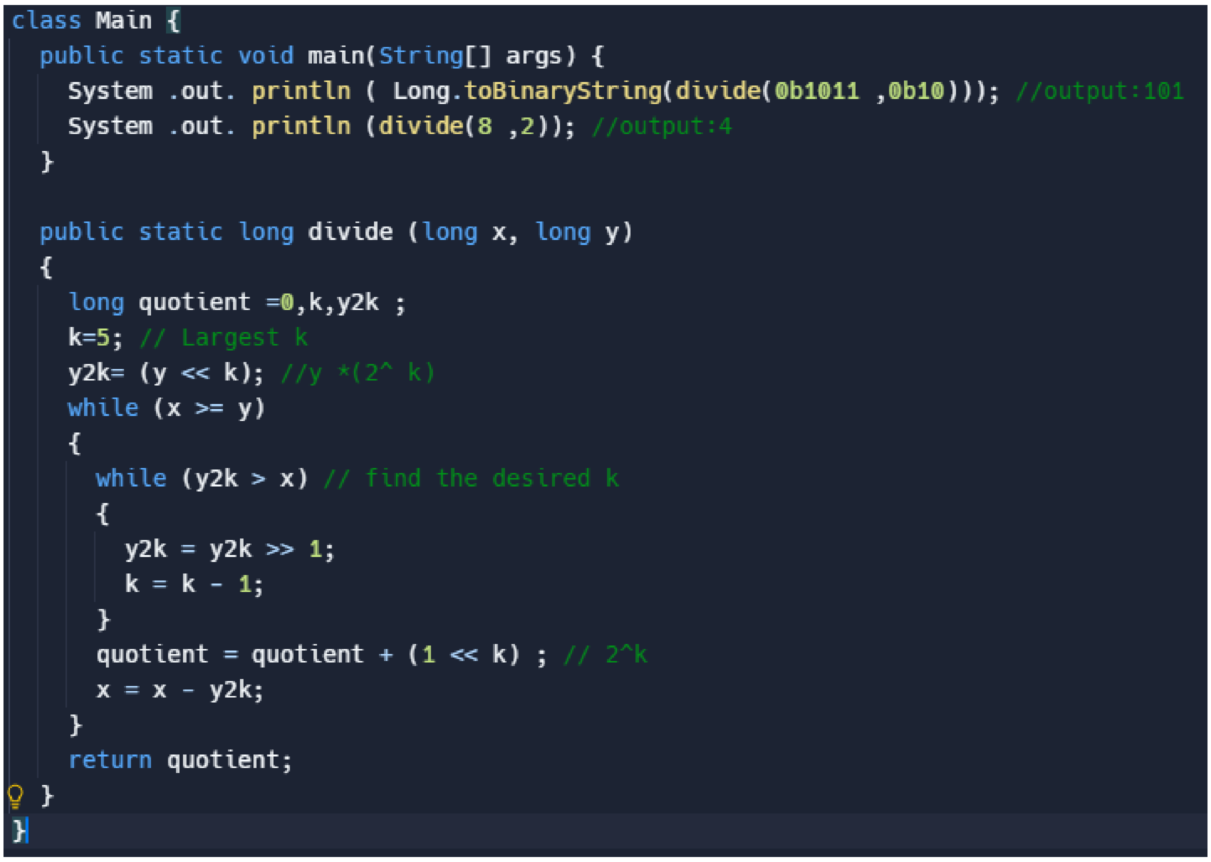
$$\begin{array}{r} Y=1101=13 \\ \times\ X=1001=9 \\ \hline 1101 \\ +\quad 0000\phantom{0} \\ 0000\phantom{00} \\ 1101\phantom{000} \\ \hline 1110101=117 \end{array}$$



```
class Main {
  public static void main(String[] args) {
    long x=5, y=10 ;
    long res = multiply(x,y);
    System.out.println("Multiplication Result: "+res); // output: 50
  }

  public static long multiply(long x, long y){
  long sum = 0;
  while (x != 0) {
    // Examines each bit of x.
    if ((x & 1) != 0) //If the last bit is 1, add y to the sum
      sum = add(sum , y);
    x >>= 1 ;
    y <<= 1 ;
  }
  return sum;
  }

  public static long add( long x, long y)
  {
    long carry;
    while (y !=0)
    {
      carry =x&y; //It finds the carry bits. Carry is the bit that is set or 1 in both the
numbers (x and y)
      x=x^y; // Summation can be obtained using XOR. And store the XOR result in x
      y= carry << 1; // Left shift the carry bits by 1. And store the result in y
    }
    return x;
  }
}
```

6. Write a java program to compute x/y using addition, subtraction, and shifting operators.

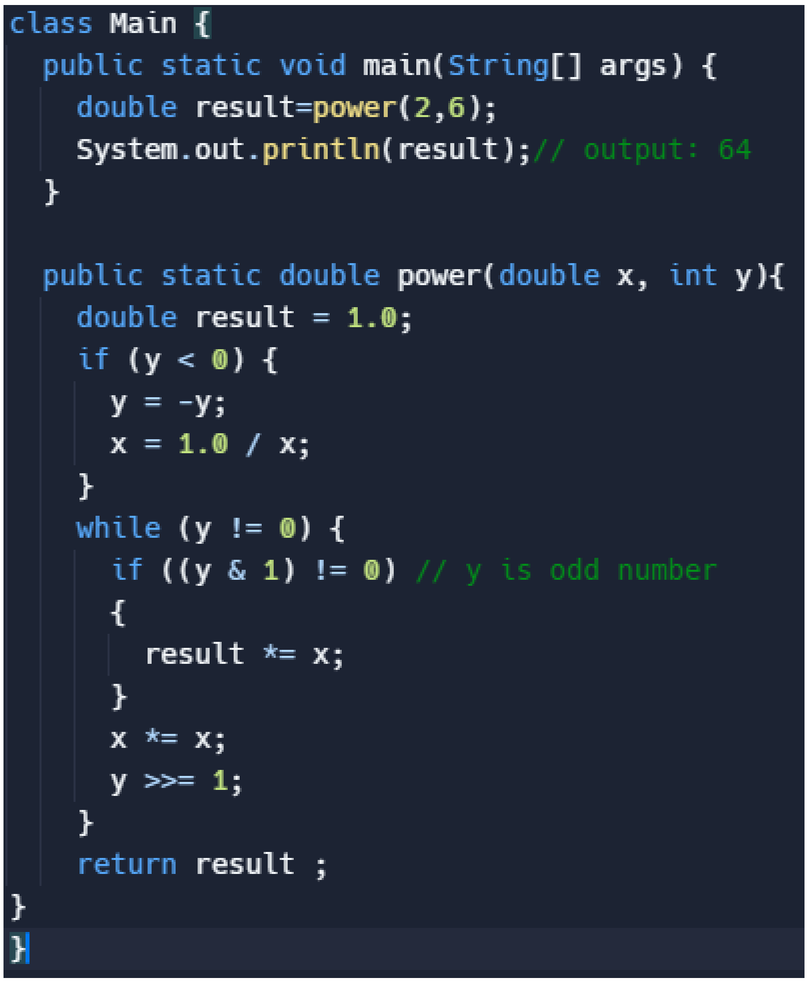
   Solution:
   Step 1: Compute the largest ***k*** such that $2^k y < x$
   Step 2: Subtract $2^k y$ from ***x***
   Step 3: Add $2^k y$ to the quotient



```java
class Main {
  public static void main(String[] args) {
    System .out. println ( Long.toBinaryString(divide(0b1011 ,0b10))); //output:101
    System .out. println (divide(8 ,2)); //output:4
  }

  public static long divide (long x, long y)
  {
    long quotient =0,k,y2k ;
    k=5; // Largest k
    y2k= (y << k); //y *(2^ k)
    while (x >= y)
    {
      while (y2k > x) // find the desired k
      {
        y2k = y2k >> 1;
        k = k - 1;
      }
      quotient = quotient + (1 << k) ; // 2^k
      x = x - y2k;
    }
    return quotient;
  }
}
```

7. Write a program to find $x^y$ without using math libraries and exponent operators. Shifting operator should be used. Ensure that the program handles both positive and negative exponents.

Solution:
Generalizing, if the least significant bit of y is 0, the result is $(X^{y/2})^2$; otherwise, it is $X \times (X^{y/2})^2$. This gives us a recursive algorithm for computing $X^y$ when $y$ is non-negative.
**Example:** Find $x^6$ *and* $x^7$. $(X^y)$
$X^6 = (X^{6/2})^2 = (X^3)^2$, where y=110, LSB is 0 (even)
$X^7 = X \times (X^{6/2})^2 = X \times (X^3)^2$ , where y=111, LSB is 1 (odd)
• The only change when $y$ is negative is replacing $x$ by $1/x$ and $y$ by $-y$.



```java
class Main {
  public static void main(String[] args) {
    double result=power(2,6);
    System.out.println(result);// output: 64
  }

  public static double power(double x, int y){
    double result = 1.0;
    if (y < 0) {
      y = -y;
      x = 1.0 / x;
    }
    while (y != 0) {
      if ((y & 1) != 0) // y is odd number
      {
        result *= x;
      }
      x *= x;
      y >>= 1;
    }
    return result ;
}
}
```