The `Number` class in Java, version 3.0

Factorize numbers and print out primes

Bjarni Jens Kristinsson

https://notendur.hi.is/~bjk17/

April 1st, 2013

API – Application programming interface

public class	Number	
	Number(long N)	*initialize object with the number N*
void	factorize(long N)	*prints the factors of N on standard output*
void	sieve(int N)	*prints all primes <N on standard output*

Changelog and agenda

```
/** changelog
 *
 *      v3.0 [2013-04-01]:
 *          - changed name of Java class to "Number"
 *          - now includes static methods to factorize the input number N
 *             and print out all primes <N
 *          - modified printFactors() so that now ALL powers are printed
 *             in super characters
 *
 *      v2.1 [2013-02-01]:
 *          - modified printFactors() so that it prints the powers 1-9
 *             with super-characters
 *
 *      v2.0 [2012-01-07]:
 *          - adding function 'sieve' to calculate all primes up to a given
 *             number with a modified version of the Sieve of Eratosthenes
 *          - adding option to read primes (generated by 'sieve' function)
 *             from text file to save computing time
 *
 *      v1.0 [2012-12-23]:
 *          - factorizes all 'long' numbers (up to 2^64 - 1)
 *          - runs in just a few seconds for up to 12 digit primes
 *
 *    agenda
 *      - modify printFactors() so that it uses super-characters [Ctrl + ^]->[Space]->[number]
 *         for all powers (not only one-digit powers)
 *      - raise maximum capable number to unlimited, using BigInteger
 *      - implent a function using Sieve of Eratosthenes to list all primes up to 'N'
 *      - read primes from an external .txt document to save computing time
 *      - implement advanced algorithms to speed up running time and save memory usage
 *     - beat the Java applet's marvellous running time on the site
 *          http://web.math.princeton.edu/math_alive/Crypto/Lab2/Factorization.html
 */
```

Source code

```
import java.util.ArrayList;
import java.util.BitSet;

public class Number {
    static long N;
    static int count;
    static int primeCount;
    static long[] factor;
    static int[] powerFactor;
    static ArrayList<Long> primes;
    // 'N' is the input number which the program factorizes
    // 'count' is the number of different prime factors in 'N'
    // 'primeCount' is the number of different primes having been
    //    tried to divide N
    // 'factor' reserves the prime factors of 'N'
    // 'powerFactor' reserves the power of the factors,
    //    i.e. powerFactor[i] is the power of factor[i] in 'N'
    // 'primes' stores the primes that have been found

    // use: Number n = new Number(N);
    // pre: N>0
    // post: n is an object represening the number n
    private Number (long N) {
        this.N = N;
        count = 0;
        int f = (int) (Math.log(N)/Math.log(2));
        factor = new long[f];
        powerFactor = new int[f];
        primes = new ArrayList<Long>();
         primes.add( (long)  2 );
         primes.add( (long)  3 );
    }


      //~ ----------------------- ~//
     //~ Private helping methods ~//
    //~ ----------------------- ~//

    // use: l = this.nextPrime(p);
    // pre: 'p' is a prime
    // post: 'l' is the next prime after 'p'
    private long nextPrime (long p) {
        if (primes.indexOf(p)==0) return primes.get(1);
        long testIfPrime = p;
        while (true) {
            testIfPrime += 2;
            boolean b = true;
            for (int i=0; primes.get(i)*primes.get(i)<=testIfPrime; i++) {
                if (testIfPrime%primes.get(i)==0) {
                    b=false;
                    break;
                }
            }
            if (b) {
                primes.add(testIfPrime);
                return testIfPrime;
            }
        }
    }

    // use: this.addFactor(i);
    // pre: count >= 0;
    // post: If 'i' was in 'factor' then count>0 and powerFactor[count-1] has
    //       been raised by one. Else factor[count]=i and powerFactor[count]=1.
    private void addFactor (long i) {
        if (count>0 && factor[count-1] == i) {
            powerFactor[count-1]++;
        } else {
            factor[count] = i;
            powerFactor[count] = 1;
            count++;
        }
    }

    // use: this.printFactors();
    // pre: (nothing)
    // post: The factors of 'N' has been printed on standard output.
    private void printFactors() {
        long n = N;
        for (long i=2; i*i<=n; i=nextPrime(i)) {
            while (n%i==0) {
                n = n/i;
                addFactor(i);
            }
        } if (n>1) addFactor(n);

        System.out.print(N+" = ");
        for (int i=0; i<count; i++) {
            if (powerFactor[i]>0)
                System.out.print(factor[i]+factorString(powerFactor[i]));
            if (i<count-1) System.out.print(" · ");
        }
        System.out.println();
    }

    // use: s = factorString(factor);
    // pre: factor>0
    // post: s is a string representation of factor, in super characters
    private String factorString(int factor) {
        if (factor==1) return "";
    String s = Integer.toString(factor);
    String out = "";
    for (int i=0; i<s.length(); i++) {
        switch (s.charAt(i)) {
            case '0':  out += " "; break;
            case '1':  out += "¹"; break;
            case '2':  out += "²"; break;
            case '3':  out += "³"; break;
            case '4':  out += " "; break;
            case '5':  out += " "; break;
            case '6':  out += " "; break;
            case '7':  out += " "; break;
            case '8':  out += " "; break;
            case '9':  out += " "; break;
            default:                break;
        }
    } return out;
}

// use: this.printSieve();
// pre: (nothing)
// post: All primes <min('N',2^(31)-1) has been printed on standard output.
private void printSieve () {
    int n;
     if (N > Integer.MAX_VALUE) n = Integer.MAX_VALUE;
     else                       n = (int)N;

    // Integer.MAX_VALUE  =  2 147 483 647  =  2^31 - 1
    BitSet b = new BitSet(n); // max: ~270MB

    for (int i=2; i<n; i++) {
        if (!b.get(i)) {
            System.out.print(i+" ");
            long temp = i;
            for (long j=temp*temp; j<n; j+=temp) {
                b.set((int)j, true);
            }
        }
    }
    System.out.println();
}


  //~ --------------------- ~//
 //~ Public static methods ~//
//~ --------------------- ~//

// use: Number.factorize(N);
// pre: N>1
// post: The factors of 'N' has been printed on standard output.
public static void factorize (long N) {
    Number p = new Number(N);
    p.printFactors();
}
```

```
    // use: Number.sieve(N);
    // pre: N<1
    // post: All primes <min('N',2^(31)-1) has been printed on standard output.
    public static void sieve (int N) {
        Number p = new Number((long)N);
        p.printSieve();
    }

    // Demo application factorizing input number.
    public static void main (String[] args) {
        N = Long.parseLong(args[0]);
        factorize(N);
    }
}
```