```java
package BigNumbers;

/**
 * This class is an implementation of the BNum interface using a linked-list.
 *
 * @author Matt Laidman
 * @version 1.0 (March 2014)
 */

public class LinkBNum implements BNum {

    private Node lNum;


    public LinkBNum() { // default constructor creates LinkBNum object with value 0
        this(0);
    }


    public LinkBNum (long n) { // Takes long value and converts to integer node list
        toList(String.valueOf(n));
    }


    public LinkBNum (String n) { // Takes String and converts to integer node list
        try {
            toList(n);
        } catch (Exception E) { // throws BadNumberFormatException if string contains invalid chars
            throw new BadNumberFormatException("Invalid number format");
        }
    }


    public void toList (String n) { // adds given string to node list
        for (char c : reverse(checkSign(n)).toCharArray()) { // assign LSD + reverse
            lNum = new Node(Character.getNumericValue(c), lNum); // add to node list
        }
    }


    public String toString ( ) { // returns string representation of 'this'
        String t = "";
        Node p = lNum;
        while (p != null) {
            t = t + p.digit;
            p = p.next;
        }
        return t;
    }


    public BNum clone() { // returns a clone of 'this'
        if (lNum.digit == 0) {
            return new LinkBNum(this.toString());
        } else {
            return new LinkBNum("-" + this.toString().substring(1));
        }
    }


    public boolean equals(BNum n) { // returns true if 'this' == n
        return this.toString().equals(n.toString());
    }


    public boolean lessThan(BNum n) { // true if 'this' < n
        Node p;
        if (lNum.digit ==  0 & n.getSign() == 0) { // if both positive
            if (this.toString().length() != n.toString().length()) { // if different lengths
                return this.toString().length() < n.toString().length(); // true if 'this' shorter than n
            } else { // if same length
                p = lNum.next;
                for (int i = 1 ; i < this.toString().length() ; i++) {
                    if (p.digit != n.getDigit(i)) { // if digits from left to right aren't equal
                        return p.digit < n.getDigit(i); // true if 'this' digit less than n digit
                    }
                    p = p.next;
                }
            }
        } else if (lNum.digit == 1 & n.getSign() == 1) { // if both negative
            if (this.toString().length() != n.toString().length()) { // if different lengths
                return this.toString().length() > n.toString().length(); // true if 'this' longer than n
            } else { // if same length
                p = lNum.next;
                for (int i = 1 ; i < this.toString().length() ; i++) {
                    if (p.digit != n.getDigit(i)) { // if digits from left to right aren't equal
                        return p.digit > n.getDigit(i); // true if 'this' digit greater than n digit
                    }
                    p = p.next;
                }
            }
        } else { // if different signs
            return lNum.digit > n.getSign(); // true if 'this' negative and n positive
        }
        return false;
    }


    public BNum add(BNum n) { // returns 'this' + n
        if (lNum.digit == n.getSign()) { // if same sign
            if (lNum.digit == 0) { // both are positive
                return new LinkBNum(doAdd(this, n));
            } else { // both are negative
                return new LinkBNum('-' + doAdd(this, n));
            }
        } else { // if different signs
            if (new LinkBNum(this.toString().substring(1)).lessThan(new LinkBNum(n.toString().substring(1)))) { // |this| < |n|
                if (n.getDigit(0) == 1) { // if n is negative
                    return new LinkBNum('-' + doSub(n, this));
                } else { // if n is positive
                    return new LinkBNum(doSub(n, this));
                }
            } else { // |n| < |this|
                if (lNum.digit == 1) { // if 'this' is negative
                    return new LinkBNum('-' + doSub(this, n));
                } else { // if 'this' is positive
                    return new LinkBNum(doSub(this, n));
                }
            }
        }
    }


    private String doAdd (BNum t, BNum n) { // adds the two given BNums
        int a = t.toString().length()-1;
        int b = n.toString().length()-1;
        int carry = 0;
        int sum;
        String total = "";
        while (a > 0 | b > 0) { // while both have digits
            if (a <= 0) { // if t has no more digits
                sum = n.getDigit(b) + carry;
            } else if (b <= 0) { // if n has no more digits
                sum = t.getDigit(a) + carry;
            } else { // if both still have digits
                sum = t.getDigit(a) + n.getDigit(b) + carry;
            }
            if (sum > 9) {
                sum = sum % 10;
                carry = 1;
            } else {
                carry = 0;
            }
            total = total + sum;
            a--;
            b--;
        }
        if (carry == 1) {
            return 1 +  reverse(total);
        } else {
            return reverse(total);

        }
    }


    public BNum sub(BNum n) { // returns 'this' - n
        if (lNum.digit == n.getSign()) { // same sign
            if (!new LinkBNum(this.toString().substring(1)).lessThan(new LinkBNum(n.toString().substring(1)))) { // |this| > |n|
                if (lNum.digit == 0) { // both are positive
                    return new LinkBNum(doSub(this, n));
                } else { // both are negative
                    return new LinkBNum('-' + doSub(this, n));
                }
            } else { // |this| < |n|
                if (lNum.digit == 0) { // both are positive
                    return new LinkBNum('-' + doSub(n, this));
                } else { // both are negative
                    return new LinkBNum(doSub(n, this));
                }
            }
        } else { // different signs
            if (this.lessThan(n)) { // negative sub positive
                return new LinkBNum('-' + doAdd(this, n));
            } else { // positive sub negative
                return new LinkBNum(doAdd(this, n));
            }
        }
    }


    private String doSub (BNum t, BNum n) { // subtracts the two given BNums
        int a = t.toString().length()-1;
        int b = n.toString().length()-1;
        int carry = 0;
        int diff;
        String total = "";
        while (a > 0 | b > 0) { // while both have digits
            if (a <= 0) { // if t has no more digits
                diff = n.getDigit(b) - carry;
            } else if ( b <= 0) { // if n has no more digits
                diff = t.getDigit(a) - carry;
            } else { // if both have digits
                diff = t.getDigit(a) - n.getDigit(b) - carry;
            }
            if (diff < 0) {
                diff = 10 + diff;
                carry = 1;
            } else {
                carry = 0;
            }
            total = total + diff;
            a--;
            b--;
        }
        return reverse(total);
    }


    public int getSign() { // returns the LSD of the BNum
        return lNum.digit;
    }


    public int getDigit(int i) { // returns the digit at the given index, 0 is LSD
        Node p;
        try {
            p = lNum;
            for (int j = 0 ; j < i ; j++) {
                p = p.next;
            }
            return p.digit;
        } catch (Exception E) { // throws DigitOutOfRangeException if index
```

```
 out of range
                throw new DigitOutOfRangeException("Index out of range");
            }
        }


    private String reverse (String n) { // returns the reverse of given string
        String rTotal = "";
        for (int i = n.length()-1 ; i >= 0 ; i--) {
            rTotal = rTotal + n.charAt(i);
        }
        return rTotal;
    }


    private String checkSign (String n) { // checks string and applies LSD
        if (n.charAt(0) == '-') {
            return 1 + n.substring(1);
        } else if (n.charAt(0) == '0') {
            return n;
        } else {
            return 0 + n;
        }
    }
}
```