CS 102: Data Structures

Project Two - Badly Beautiful Binary

Student Name	Mohammad El-Abid
Student ID	8905652
Professor	Subrina Thompson
Date	Thursday, February 23, 2012

Table of Contents

Abstract

This project can understand 2s compliment numbers as well as add and subtract them (in binary form). It can also convert biased notation to 2s compliment.

Introduction

The program stores the binary digits in a character array with 8 indexes. The binary class also deals with adding and converting other binaries. The driver then loads a file of 2s compliment and biased notation numbers and prints them out and their sum.

Screenshots

Application output to `stdout` (following page, too large for this page):

```
  00010110   2's complement       22
+ 10000011   biased notation   +    3
  --------                      ----
  00011001   2's complement       25

  10000001   2's complement     -127
+ 00000000   biased notation   + -128
  --------                      ----
  Underflow  2's complement     -128

  11111111   2's complement       -1
+ 00000000   biased notation   + -128
  --------                      ----
  Underflow  2's complement     -128

  00000011   2's complement        3
+ 10000111   biased notation   +    7
  --------                      ----
  00011000   2's complement       24

  00001111   2's complement       15
+ 10000111   biased notation   +    7
  --------                      ----
  01110000   2's complement      112

  10000000   2's complement     -128
+ 11111111   biased notation   +  127
  --------                      ----
  11111111   2's complement       -1

  11110000   2's complement      -16
+ 10001000   biased notation   +    8
  --------                      ----
  11111000   2's complement       -8

  10000001   2's complement     -127
+ 00000001   biased notation   + -127
  --------                      ----
  Underflow  2's complement     -128
```

```
  01111111   2's complement      127
+ 00000000   biased notation   + -128
  --------                      ----
  11111111   2's complement       -1

  01110101   2's complement      117
+ 11010001   biased notation   +  81
  --------                      ----
  Overflow   2's complement      127

  00000000   2's complement        0
+ 10000000   biased notation   +   0
  --------                      ----
  00000000   2's complement        0

  00001111   2's complement       15
+ 11110000   biased notation   + 112
  --------                      ----
  01111111   2's complement      127
```

Code

File: Byte.java

```
package edu.bridgeport.mohammad.binary;

public class Byte {
    private char[] bits = { '0', '0', '0', '0', '0', '0', '0', '0' };
    private boolean underflow = false;
    private boolean overflow = false;

    public Byte() {
          // default
    }

    public Byte(String newBits) {
          this(newBits.toCharArray());
    }

    public Byte(char[] newBits) {
          for (int i = 0; i < bits.length; i++) {
               bits[i] = newBits[i]; // will out of range on bits-n !
= newBits-n
          }
    }

    public Byte(Byte b) {
          char[] other = b.getBits();
          for (int i = 0; i < bits.length; i++) {
               bits[i] = other[i];
          }
    }

    public Byte add(Byte other) {
          Byte res = new Byte();
          char[] resultSet = res.getBits();
          char[] firstSet = bits;
          char[] secondSet = other.getBits();

          int carryOver = 0;
          for (int i = firstSet.length - 1; i >= 0; i--) {
               if (firstSet[i] == '1' && secondSet[i] == firstSet[i])
{
                    // both one
                    resultSet[i] = '0';
                    carryOver++;
               } else if (firstSet[i] != secondSet[i]) {
                    // one zero, one one
```

```
                    // can one of zeros borrow from a carry over?
                    if (carryOver > 0) {
                        // if so carry again
                        resultSet[i] = '0';
                    } else {
                        resultSet[i] = '1';
                    }
                } else {
                    // both are zero
                    // do we have any carry overs?
                    if (carryOver > 0) {
                        // if so apply them
                        resultSet[i] = '1';
                        carryOver--;
                    } else {
                        // otherwise, set to zero
                        resultSet[i] = '0';
                    }
                }
            }

            if (carryOver > 0) {
                // Overflow: first result bit is 1 but both a and b's
1st bit is 0
                if (resultSet[0] == '1' && firstSet[0] == '0'
                        && secondSet[0] == '0') {
                    res.overflow = true;
                    try {
                        res.setBits(new char[]
{'0','1','1','1','1','1','1','1'});
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                    // Underflow: first result bit is 0 but both a
and b's 1st bit
                    // is 1
                } else if (resultSet[0] == '0' && firstSet[0] == '1'
                        && secondSet[0] == '1') {
                    res.underflow = true;
                    try{
                        res.setBits(new char[]
{'1','0','0','0','0','0','0','0'});
                    } catch(Exception e) {
                        e.printStackTrace();
                    }
                }
            }

            return res;
```

```
}

private void setBits(char[] cs) throws Exception {
	if(cs.length != 8){
		throw new Exception("Not 8 bits");
	}
	bits = cs;
}

public Byte biasedToTwosCompliment() {
	Byte copy = new Byte(this);
	char[] bits = copy.getBits();
	bits[0] = (bits[0] == '0' ? '1' : '0');
	return copy;
}

public int magnitude() {
	// This is 2 complements

	int value = 0;
	char[] mangle = new char[bits.length];
	for (int i = 0; i < bits.length; i++)
		mangle[i] = bits[i];

	// if negative
	if (bits[0] == '1') {
		// Flip the bits
		for (int i = 0; i < mangle.length; i++) {
			mangle[i] = (mangle[i] == '0' ? '1' : '0');
		}

		// adds one
		for (int i = mangle.length - 1; i > -1; i--) {
			if (mangle[i] == '0') {
				mangle[i] = '1';
				break;
			} else { // the bit is 1
				mangle[i] = '0';
			}
		}
	}

	// add values
	for (int i = 0; mangle.length > i; i++) {
		if (mangle[i] == '1') {
			value += Math.pow(2, mangle.length - i - 1);
		}
	}
```

```
            // return value, prepend negative if negative number
            if (bits[0] == '1') {
                  // -(value+1)
                  return -value;
            } else {
                  return value;
            }
      }

      public char[] getBits() {
            return bits;
      }

      @Override
      public String toString() {
            if (overflow)
                  return "Overflow";
            if (underflow)
                  return "Underflow";

            StringBuilder builder = new StringBuilder();

            for (char element : bits)
                  builder.append(element);
            return builder.toString();
      }
}
```

File: Application.java

```
package edu.bridgeport.mohammad.binary;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class Application {
      private final static String twosComplimentForm = "  %8s   2's
complement     %4d";
      private final static String biasedNotationForm = "+ %8s   biased
notation  + %4d";
      private final static String resultForm         = "  %-9s  2's
complement     %4d";

      public static void main(String[] args) throws
FileNotFoundException{
            Scanner input = new Scanner(new FileInputStream("binary-
input.txt"));
```

```
            while(input.hasNextLine()) {
                  Byte twosCompliment = new Byte(input.next());
                  Byte biasedNotation = new Byte(input.next());
                  Byte biasedToCompliment =
biasedNotation.biasedToTwosCompliment();
                  Byte result = biasedToCompliment.add(twosCompliment);

                  System.out.println(String.format(twosComplimentForm,
twosCompliment.toString(), twosCompliment.magnitude()));
                  System.out.println(String.format(biasedNotationForm,
biasedNotation.toString(), biasedToCompliment.magnitude()));
                  System.out.println("  --------
----");
                  System.out.println(String.format(resultForm,
result.toString(), result.magnitude()));
                  System.out.println();
            }
      }
}
```

Conclusion

Binary numbers are simplistic once understood, but may be hard for people used to thinking in base ten (decimal). I found that most of my bugs were logic errors due to my incomplete understanding of how binary digits worked.