```java
/*****************************************************************************
*   FILE: SDESBits.java
*   AUTHOR: Connor Beardsmore - 15504319
*   UNIT: FCC200
*   PURPOSE: BitSet alternative using a int
*   LAST MOD: 24/03/17
*   REQUIRES: NONE
*****************************************************************************/

public class SDESBits
{
    //CONSTANTS
    public static final int MIN_SIZE = 4;
    public static final int MAX_SIZE = 10;

    //CLASSFIELDS
    private int bits;
    private int size;
    private int half;
    // private only applies to different classes, so we can
    // import an SDESBits and retreive bits without a getter

//---------------------------------------------------------------------------
    //ALTERNATE CONSTRUCTOR

    public SDESBits( int inBits, int inSize )
    {
        // Check inBits and inSize validity
        if ( inBits < 0 )
            throw new IllegalArgumentException( "INVALID SDESBits VALUE" );
        if ( ( inSize < MIN_SIZE ) || ( inSize > MAX_SIZE ) )
            throw new IllegalArgumentException( "INVALID SDESBits SIZE" );
        if ( inSize % 2 != 0 )
            throw new IllegalArgumentException( "INVALID SDESBits SIZE" );

        bits = inBits;
        size = inSize;
        half = inSize >>> 1;
    }

//---------------------------------------------------------------------------
    //COPY CONSTRUCTOR

    public SDESBits( SDESBits inBits )
    {
        bits = inBits.bits;
        size = inBits.size;
        half = inBits.half;
    }

//---------------------------------------------------------------------------
    //FUNCTION: switchHalves()
    //PURPOSE: Switch the left half of bits with the right half

    public void switchHalves()
    {
        // Get the right half of the bits
        int oRight = bits & ( ( 1 << half ) - 1 );
        // Shift the left half of the bits down
        bits >>>= half;
        // Combine left half with right half shifted up
        bits |= ( oRight << half );
    }

//---------------------------------------------------------------------------
    //FUNCTION: permute()
    //IMPORT: permTable (int[])
    //EXPORT: permuted (SDESBits)
    //PURPOSE: Create a permutation of this objects bits in a new SDESBits

    public SDESBits permute( int[] permTable )
    {
```

```
73          // Create temporary space the size of the permutation
74          SDESBits permuted = new SDESBits( 0, permTable.length );
75          // Iterate across the permutation, getting and setting bits
76          for ( int ii = 0; ii < permTable.length; ii++ )
77              permuted.setBit( getBit( permTable[ii] ), ii );
78          return permuted;
79      }
80
81  //-------------------------------------------------------------------------
82      //FUNCTION: leftShift()
83      //IMPORT: shifts (int)
84      //PURPOSE: Perfrom a circular left shift on the bits of each half
85
86      public void leftShift( int shifts )
87      {
88          //Check shift validity
89          if ( shifts < 1 )
90              throw new IllegalArgumentException( "ILLEGAL SHIFT VALUE" );
91
92          // Temp variable for repeated 1's for a half
93          int ones = ( 1 << half ) - 1;
94          // Avoid shifting more than required
95          if ( half > shifts )
96              shifts %= half;
97
98          // Get the left half and right half
99          int left = bits >>> half;
100         int right = bits & ones;
101
102         // Loop for each shift individually
103         for ( int ii = 0; ii < shifts; ii++ )
104         {
105             // Get the leftmost bit of the left sub-half
106             int leftBit = ( left & ones );
107             leftBit >>>= MIN_SIZE;
108             // Get the rightmost bit of the right sub-half
109             int rightBit = ( right & ones );
110             rightBit >>>= MIN_SIZE;
111
112             // Perform the actual shifting of the bits
113             left = ( left << 1 ) & ones;
114             right = ( right << 1 ) & ones;
115
116             // If the first bits of the halves were one, set final bit
117             if ( leftBit == 1 )      left++;
118             if ( rightBit == 1 )     right++;
119         }
120
121         // Recombine both halves back together
122         bits = ( left << half ) | right;
123     }
124
125 //-------------------------------------------------------------------------
126     //FUNCTION: split()
127     //EXPORT: halves (SDESBits[])
128     //PURPOSE: Split the bits into two sub-halves and return as objects
129
130     public SDESBits[] split()
131     {
132         // New container for the halves
133         SDESBits[] halves = new SDESBits[2];
134         // Get the left half and create object
135         int leftInt = bits >>> half;
136         halves[0] = new SDESBits( leftInt, half );
137         // Get the right half and create object
138         int rightInt = ( bits & ( ( 1 << half ) - 1 ) );
139         halves[1] = new SDESBits( rightInt, half );
140
141         return halves;
142     }
143
144 //-------------------------------------------------------------------------
145     //FUNCTION: xor()
146     //IMPORT: inBits (SDESBits)
```

```
147     //PURPOSE: XOR bits with the bits value of inBits
148
149     public void xor( SDESBits inBits )
150     {
151         // Ensure the same size
152         if ( size != inBits.size )
153             throw new IllegalArgumentException( "CANNOT XOR DIFFERENT SIZES" );
154
155         // Call simple exclusive-or on both bits
156         bits ^= inBits.bits;
157     }
158
159 //-------------------------------------------------------------------------
160     //FUNCTION: setBit()
161     //IMPORT: val (boolean), index (int)
162     //PURPOSE: Set the value at the specifed index with the specified value
163
164     public void setBit( boolean val, int index )
165     {
166         // Validity
167         if ( ( index < 0 ) || ( index >= size) )
168             throw new IllegalArgumentException("SETBIT IMPORTS INVALID");
169
170         // Reset the given bit
171         bits &= ~(1 << ( size - index - 1 ) );
172         // Reset the bits greater than the size we want
173         bits &= (1 << size)-1;
174         // Set the required bit
175         bits |= ((val) ? 1 : 0 ) << ( size - index - 1 );
176     }
177
178 //-------------------------------------------------------------------------
179     //FUNCTION: getBit()
180     //IMPORT: index (int)
181     //EXPORT: value (boolean)
182     //PURPOSE: Get the value of the bit at the specified index
183
184     public boolean getBit( int index )
185     {
186         if ( ( index < 0 ) || ( index >= size) )
187             throw new IllegalArgumentException("SETBIT IMPORTS INVALID");
188
189         // Bits are reverse ordered
190         return (bits & 1 << ( size - index - 1 ) ) != 0;
191     }
192
193 //-------------------------------------------------------------------------
194
195     public int getBits() { return bits; }
196
197 //-------------------------------------------------------------------------
198     //FUNCTION: append()
199     //IMPORT: newBits (SDESBits)
200     //PURPOSE: Append new set of bits to the original set
201
202     public void append( SDESBits newBits )
203     {
204         // Increment size
205         size += newBits.size;
206         // Shift original across and add new bits
207         bits = ( bits << newBits.size ) | newBits.bits;
208         // Update half value
209         half = size >>> 1;
210     }
211
212 //-------------------------------------------------------------------------
213     //FUNCTION: sbox()
214     //EXPORT: result (int)
215     //PURPOSE: Find the sbox values for the bits in this object
216
217     public int sbox()
218     {
219         // Split into halves
220         SDESBits halves[] = this.split();
```

```
221
222         // Get row and column of the first four bits
223         int colS0 = ( halves[0].bits & 6 ) >>> 1;
224         int rowS0 = ( ( halves[0].bits & 8 ) >>> 2 ) | ( halves[0].bits & 1 );
225         // Get row and column of the second four bits
226         int colS1 = ( halves[1].bits & 6 ) >>> 1;
227         int rowS1 = ( ( halves[1].bits & 8 ) >>> 2 ) | ( halves[1].bits & 1 );
228
229         // Get the appropriate sbox value
230         int s0Val = SDESConstants.S0[rowS0][colS0];
231         int s1Val = SDESConstants.S1[rowS1][colS1];
232
233         // Combine the result
234         int result = ( s0Val << 2 ) | s1Val;
235
236         return result;
237     }
238
239 //-------------------------------------------------------------------------
240     //FUNCTION: toString()
241     //EXPORT: state (String)
242     //PURPOSE: Export bits in a readable binary format
243
244     public String toString()
245     {
246         return Integer.toBinaryString( bits );
247     }
248
249 //-------------------------------------------------------------------------
250 }
251
```