```java
/*******************************************************************************
*   FILE: SDES.java
*   AUTHOR: Connor Beardsmore - 15504319
*   UNIT: FCC200
*   PURPOSE: Performs SDES encryption or decryption on a given file
*   LAST MOD: 21/03/17
*   REQUIRES: NONE
*******************************************************************************/

import java.util.*;
import java.io.*;

public class SDES
{

    public static final int NUM_ARGS = 4;
    public static final int MAX_KEY = 1023;
    public static final int KEY_SIZE = 10;
    public static final int MESSAGE_SIZE = 8;

//------------------------------------------------------------------------------

    public static void main( String[] args )
    {
        // Check argument length and output usage
        if ( args.length != NUM_ARGS )
        {
            System.out.println("USAGE: SDES <mode> <key> <input file> <output file>");
            System.out.println("modes = -e encryption, -d decryption");
            System.out.println("keys = int between 0 and 255");
            System.exit(1);
        }

        // Rename variables for simplicity
        String mode = args[0];
        String key = args[1];
        String inFile = args[2];
        String outFile = args[3];
        SDESBits message, output;
        int intKey = Integer.parseInt( key );

        try
        {
            // Generate subkeys
            SDESBits subkeys[] = keyGeneration( intKey );

            // Open file streams
            FileInputStream fis = new FileInputStream( new File( inFile ) );
            FileOutputStream fos = new FileOutputStream( new File( outFile ) );

            // Read bytes until end of file
            int next = fis.read();
            while ( next != -1 )
            {
                message = new SDESBits( next, MESSAGE_SIZE );

                // Select function based on mode
                if ( mode.equals( "-e" ) )
                    output = encrypt( message, subkeys  );
                else if ( mode.equals( "-d") )
                    output = decrypt( message, subkeys  );
                else
                    throw new IllegalArgumentException("INVALID MODE");

                // Write converted output to file
                int outputInt = output.getBits();
                fos.write( outputInt );
                next = fis.read();
            }
        }
        catch (Exception e)
        {
```

```
73              System.out.println( e.getMessage() );
74          }
75
76      }
77
78  //--------------------------------------------------------------------------
79      //FUNCTION: encrypt()
80      //IMPORT: message (SDESBits), subkeys (SDESBits[])
81      //EXPORT: message (SDESBits)
82      //PURPOSE: Encrypt given message with given subkeys
83
84      public static SDESBits encrypt( SDESBits message, SDESBits[] subkeys )
85      {
86          // Initial Permutation
87          message = message.permute( SDESConstants.IP );
88          // First feistal key round with subkey 1
89          message = feistalRound( message, subkeys[0] );
90          // Switch left and right subhalves
91          switchFunction( message );
92          // Second feistal key round with subkey 2
93          message = feistalRound( message, subkeys[1] );
94          // Inverse of Initial Permutation
95          message = message.permute( SDESConstants.IPI );
96          return message;
97      }
98
99  //--------------------------------------------------------------------------
100     //FUNCTION: decrypt()
101     //IMPORT: message (SDESBits), subkeys (SDESBits[])
102     //EXPORT: message (SDESBits)
103     //PURPOSE: Decrypt given message with given subkeys
104
105     public static SDESBits decrypt( SDESBits message, SDESBits[] subkeys )
106     {
107         // Initial Permutation
108         message = message.permute( SDESConstants.IP );
109         // First feistal key round with subkey 2
110         message = feistalRound( message, subkeys[1] );
111         // Switch left and right subhalves
112         switchFunction( message );
113         // First feistal key round with subkey 2
114         message = feistalRound( message, subkeys[0] );
115         // Inverse of Initial Permutation
116         message = message.permute( SDESConstants.IPI );
117         return message;
118     }
119
120 //--------------------------------------------------------------------------
121     //FUNCTION: switchFunction()
122     //IMPORT: input (SDESBitSet)
123     //PURPOSE: Import 8-bit binary and swap the first and last 4 bits
124
125     public static void switchFunction( SDESBits input )
126     {
127         input.switchHalves();
128     }
129
130 //--------------------------------------------------------------------------
131     //FUNCTION: keyGeneration()
132     //IMPORT: keyDec (int)
133     //EXPORT: subkeys (SDESBits[])
134     //PURPOSE: Generate subkeys given the full key
135
136     public static SDESBits[] keyGeneration( int keyDec )
137     {
138         // Check key validity
139         if ( ( keyDec < 0 ) || ( keyDec > MAX_KEY ) )
140             throw new IllegalArgumentException("INVALID KEY");
141
142         // Convert int key into an SDESBits object and create subkey array
143         SDESBits key = new SDESBits( keyDec, KEY_SIZE );
144         SDESBits[] subkeys = new SDESBits[2];
145
146         // P10 permutation, left shift and P8 permutation to form subkey 1
```

```
147         key = key.permute( SDESConstants.P10 );
148         key.leftShift(1);
149         subkeys[0] = key.permute( SDESConstants.P8 );
150         // P8 permutation and double left shift to form subkey 2
151         key.leftShift(2);
152         subkeys[1] = key.permute( SDESConstants.P8 );
153 
154         return subkeys;
155     }
156 
157 //---------------------------------------------------------------------------
158     //FUNCTION: feistalRound()
159     //IMPORT: message (SDESBits), subkey (SDESBits)
160     //EXPORT: halves (SDESBits)
161     //PURPOSE: Perform feistal key round on message given a subkey
162 
163     public static SDESBits feistalRound( SDESBits message, SDESBits subkey )
164     {
165         // Split message in half
166         SDESBits halves[] = message.split();
167         // Perform fMapping function
168         SDESBits fMap = fMapping( halves[1], subkey );
169         // XOR the halves and append
170         halves[0].xor( fMap );
171         halves[0].append(halves[1]);
172         return halves[0];
173     }
174 
175 //---------------------------------------------------------------------------
176     //FUNCTION: fMapping()
177     //IMPORT: message (SDESBits), subkey (SDESBits)
178     //EXPORT: message (SDESBits)
179     //PURPOSE: Perform fMapping function on given message with subkey
180 
181     public static SDESBits fMapping( SDESBits message, SDESBits subkey )
182     {
183         // Expansion permutation and XOR with subkey
184         message = message.permute( SDESConstants.EP );
185         message.xor( subkey );
186         // Calculate SBOX values and P4 permutation
187         message = new SDESBits( message.sbox(), MESSAGE_SIZE/2 );
188         message = message.permute( SDESConstants.P4 );
189         return message;
190     }
191 
192 //---------------------------------------------------------------------------
193 }
194 
```