```java
package Project1_ASCII;

import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
public class ASCII extends JFrame {

    private JButton openFileButton;
    private JComboBox<String> Jcbdelimiter;
    private JComboBox<String> Jchecksum;
    private JComboBox<String> Jxorsum;
    private JButton Reset;
    private JButton Resetcheck;
    private JTextField JFxorsum;
    private JTextField JFdecimiter;
    private JTextArea txtArea;
    private JTextArea hexArea;
    private JTextArea binArea;
    private JTextArea decArea;
    private String asciiText = ""; // Store the ASCII text

    public ASCII() {
        setTitle("ASCII, Hex, Binary, Decimal, Base64 converter");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        getContentPane().setBackground(new Color(0xFAFAD2));

        // Reset button
        Reset = new JButton("x Reset");
        Reset.setBounds(130, 15, 80, 25);
        Reset.setBackground(new Color(0x808080));
        add(Reset);

        // Number Delimiter
        Jcbdelimiter = new JComboBox<>(new String[]{"None", "Space"
    , "Comma", "User defined"});
        Jcbdelimiter.setBounds(20, 70, 230, 35);
        JLabel Jdelimiter = new JLabel("Number delimiter");
        Jdelimiter.setBounds(20, 50, 100, 15);
        add(Jcbdelimiter);
        add(Jdelimiter);

        //JFdecimiter Jtexfield
        JFdecimiter = new JTextField();
        JFdecimiter.setEditable(false);
        JFdecimiter.setBounds(260, 70, 225, 35);
        add(JFdecimiter);

```

```
        // prefixCheckbox;
        JCheckBox prefixCheckbox = new JCheckBox();
        prefixCheckbox.setBounds(20, 120, 20, 25);
        JLabel prefixbox = new JLabel("0x/0b prefix");
        prefixbox.setBounds(48, 120, 100, 25); // Adjusted width to show the text properly
        add(prefixCheckbox);
        add(prefixbox);

        // ASCII text
        JLabel asciiLabel = new JLabel("ASCII text:");
        asciiLabel.setBounds(20, 155, 100, 15);
        // Create a JTextArea
        txtArea = new JTextArea(3, 30);
        txtArea.setLineWrap(true);  // Enable line wrap

        // Create a JScrollPane for the JTextArea
        JScrollPane asciiScrollPane = new JScrollPane(txtArea);
        asciiScrollPane.setBounds(20, 175, 470, 60);
        asciiScrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
        asciiScrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);

        add(asciiLabel);
        add(asciiScrollPane); // Add the JScrollPane instead of the JTextArea

        // Hex (bytes)
        JLabel hexLabel = new JLabel("Hex (bytes):");
        hexLabel.setBounds(20, 250, 100, 15);

        // Create a JTextArea
        hexArea = new JTextArea(3, 30);
        hexArea.setLineWrap(true);  // Enable line wrap

        // Create a JScrollPane for the JTextArea
        JScrollPane hexScrollPane = new JScrollPane(hexArea);
        hexScrollPane.setBounds(20, 270, 470, 60);
        hexScrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
        hexScrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);

        add(hexLabel);
        add(hexScrollPane); // Add the JScrollPane instead of the JTextArea

        // Binary (bytes)
        JLabel binLabel = new JLabel("Binary (bytes):");
        binLabel.setBounds(20, 350, 100, 15);

        // Create a JTextArea
        binArea = new JTextArea(3, 30);
        // Enable line wrap
        binArea.setLineWrap(true);

```

```
107         // Create a JScrollPane for the JTextArea
108         JScrollPane binScrollPane = new JScrollPane(binArea);
109         binScrollPane.setBounds(20, 370, 470, 60);
110         binScrollPane.setVerticalScrollBarPolicy(
    ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
111         binScrollPane.setHorizontalScrollBarPolicy(
    ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
112 
113         add(binLabel);
114         add(binScrollPane); // Add the JScrollPane instead of the
    JTextArea
115 
116         // Decimal (bytes)
117         JLabel decLabel = new JLabel("Decimal (bytes):");
118         decLabel.setBounds(20, 440, 100, 15);
119 
120         // Create a JTextArea
121         decArea = new JTextArea(3, 30);
122         decArea.setLineWrap(true);  // Enable line wrap
123 
124         // Create a JScrollPane for the JTextArea
125         JScrollPane decScrollPane = new JScrollPane(decArea);
126         decScrollPane.setBounds(20, 460, 470, 60);
127         decScrollPane.setVerticalScrollBarPolicy(
    ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED);
128         decScrollPane.setHorizontalScrollBarPolicy(
    ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED);
129 
130         add(decLabel);
131         add(decScrollPane); // Add the JScrollPane instead of the
    JTextArea
132 
133         // Checksum
134         Jchecksum = new JComboBox<>(new String[]{"8-bit", "16-bit",
     "32-bit"});
135         Jchecksum.setBounds(20, 550, 150, 35);
136         JLabel Lchecksum = new JLabel("Checksum");
137         Lchecksum.setBounds(20, 520, 470, 35);
138         add(Lchecksum);
139         add(Jchecksum);
140 
141         // Xor, Sum, 2's complement
142         Jxorsum = new JComboBox<>(new String[]{"Sum", "2's
    Complement", "Xor"});
143         Jxorsum.setBounds(180, 550, 150, 35);
144         add(Jxorsum);
145 
146         // JTextField after xor, sum, 2's complement JFxorsum
147         JFxorsum = new JTextField();
148         JFxorsum.setEditable(false);
149         JFxorsum.setBounds(340, 550, 151, 35);
150         add(JFxorsum);
151 
152         // Reset button under Checksum
153         Resetcheck = new JButton("x Reset");
154         Resetcheck.setBounds(20, 595, 85, 25);
155         Resetcheck.setBackground(new Color(0x808080));
```

```
        add(Resetcheck);

        // Inside the ActionListener for Number delimiter combo box
        Jcbdelimiter.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                updateFields();
                String selectedDelimiter = Jcbdelimiter.
    getSelectedItem().toString();
                if (selectedDelimiter.equals("User defined")) {
                    JFdecimiter.setEditable(true);
                    JFdecimiter.setText(",0x"); // Clear the
    JTextField
                } else {
                    JFdecimiter.setEditable(false);
                    JFdecimiter.setText("");
                    if (selectedDelimiter.equals("Comma")) {
                        JFdecimiter.setText(",");
                    }
                }
            }
        });

        // Inside the ASCII constructor, after setting up
    JFdecimiter
        JFdecimiter.getDocument().addDocumentListener(new
    DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                updateFields();
            }
            @Override
            public void removeUpdate(DocumentEvent e) {
                updateFields();
            }
            @Override
            public void changedUpdate(DocumentEvent e) {
                updateFields();
            }
        });

        // ActionListener for prefixCheckbox
        prefixCheckbox.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                isPrefixEnabled = prefixCheckbox.isSelected();
                updateFields();
            }
        });

        // ActionListener for Checksum combo box
        Jchecksum.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                updateFields();
            }
        });
```

```
209
210         // ActionListener for Xor, Sum, 2's complement combo box
211         Jxorsum.addActionListener(new ActionListener() {
212             @Override
213             public void actionPerformed(ActionEvent e) {
214                 updateFields();
215             }
216         });
217         Jxorsum.addActionListener(new ActionListener() {
218             @Override
219             public void actionPerformed(ActionEvent e) {
220                 updateFields();
221             }
222         });
223
224         // ActionListener for Reset button
225         Reset.addActionListener(new ActionListener() {
226             @Override
227             public void actionPerformed(ActionEvent e) {
228                 resetFields();
229             }
230         });
231
232         Resetcheck.addActionListener(new ActionListener() {
233             @Override
234             public void actionPerformed(ActionEvent e) {
235                 resetFields();
236             }
237         });
238
239         // ActionListener for ASCII text input
240         txtArea.addKeyListener(new KeyAdapter() {
241             @Override
242             public void keyReleased(KeyEvent e){
243                 updateFields();
244             }
245             public void removeUpdate(DocumentEvent e){
246                 updateFields();
247             }
248             public void changeUpdate(DocumentEvent e){
249                 updateFields();
250             }
251         });
252         setSize(520, 670);
253         setLocationRelativeTo(null); // Center the frame on the
    screen
254
255         // Open File button
256         openFileButton = new JButton("Open File");
257         openFileButton.setBounds(20, 15, 100, 25);
258         openFileButton.setBackground(new Color(0x00ff00));
259         openFileButton.addActionListener(new ActionListener() {
260             @Override
261             public void actionPerformed(ActionEvent e) {
262                 openFile();
263             }
264         });
```

```
265         add(openFileButton);
266     }
267     private void resetFields() {
268         txtArea.setText("");   // Clear ASCII text area
269         hexArea.setText("");   // Clear Hex area
270         binArea.setText("");   // Clear Binary area
271         decArea.setText("");   // Clear Decimal area
272         JFdecimiter.setText(""); // Clear Number delimiter text
    field
273         JFxorsum.setText("");    // Clear Xor, Sum, 2's complement
    result text field
274         Jchecksum.setSelectedIndex(0); // Reset the checksum combo
    box
275         Jxorsum.setSelectedIndex(0);
276         Jcbdelimiter.setSelectedIndex(0);
277     }
278     private void openFile() {
279         JFileChooser fileChooser = new JFileChooser();
280         int returnValue = fileChooser.showOpenDialog(this);
281         if (returnValue == JFileChooser.APPROVE_OPTION) {
282             File selectedFile = fileChooser.getSelectedFile();
283             new FileLoader(selectedFile).execute();
284         }
285     }
286     private class FileLoader extends SwingWorker<StringBuilder,
    Void> {
287         private final File file;
288 
289         public FileLoader(File file) {
290             this.file = file;
291         }
292         @Override
293         protected StringBuilder doInBackground() throws Exception {
294             StringBuilder fileContent = new StringBuilder();
295             try (BufferedReader reader = new BufferedReader(new
    FileReader(file))) {
296                 String line;
297                 while ((line = reader.readLine()) != null) {
298                     fileContent.append(line).append("\n");
299                 }
300             } catch (IOException ex) {
301                 JOptionPane.showMessageDialog(ASCII.this, "Error
    opening file: " + ex.getMessage(), "Error", JOptionPane.
    ERROR_MESSAGE);
302             }
303             return fileContent;
304         }
305         @Override
306         protected void done() {
307             try {
308                 StringBuilder fileContent = get();
309                 txtArea.setText(fileContent.toString());
310                 updateFields(); // Update fields when the file is
    loaded
311             } catch (Exception ex) {
312             }
313         }
```

```
314     }
315     private long calculatedChecksum;
316     private boolean isPrefixEnabled = false;
317 
318     // Inside the getNumberDelimiter() method
319     private String getNumberDelimiter() {
320         String delimiter = "";
321         String selectedDelimiter = Jcbdelimiter.getSelectedItem().
    toString();
322         switch (selectedDelimiter) {
323             case "None":
324                 delimiter = "";
325                 break;
326             case "Space":
327                 delimiter = " ";
328                 break;
329             case "Comma":
330                 delimiter = ",";
331                 break;
332             case "User defined":
333                 delimiter = " " + JFdecimiter.getText();
334                 break;
335         }
336         return delimiter;
337     }
338     private void updateFields() {
339         asciiText = txtArea.getText(); // Store the ASCII text
340         // Update the fields
341         String hexResult = convertTextToHex(asciiText);
342         hexArea.setText(hexResult);
343         String binaryResult = convertTextToBinary(asciiText);
344         binArea.setText(binaryResult);
345         String decimalResult = convertTextToDecimal(asciiText);
346         decArea.setText(decimalResult);
347         calculateChecksum(); // Calculate and display checksum
    values
348     }
349 
350     private String convertTextToDecimal(String asciiText) {
351         StringBuilder decimalBuilder = new StringBuilder();
352         // Loop through each character in the ASCII text
353         for (int i = 0; i < asciiText.length(); i++) {
354             char c = asciiText.charAt(i);
355             // Convert the character to its decimal representation
356             decimalBuilder.append(String.valueOf((int) c));
357             // Add delimiter if it's not the last character
358             if (i < asciiText.length() - 1) {
359                 decimalBuilder.append(getNumberDelimiter());
360             }
361         }
362         return decimalBuilder.toString();
363     }
364 
365     private String convertTextToBinary(String asciiText) {
366         StringBuilder binaryBuilder = new StringBuilder();
367         // Loop through each character in the ASCII text
368         for (int i = 0; i < asciiText.length(); i++) {
```

```
            char c = asciiText.charAt(i);
            // Add "0b" prefix if needed
            if (isPrefixEnabled) {
                binaryBuilder.append("0b");
            }
            // Convert the character to its binary representation
    with 8 bits
            binaryBuilder.append(String.format("%8s", Integer.
    toBinaryString(c)).replace(' ', '0'));
            // Add delimiter if it's not the last character
            if (i < asciiText.length() - 1) {
                binaryBuilder.append(getNumberDelimiter());
            }
        }
        return binaryBuilder.toString();
    }

    private String convertTextToHex(String asciiText) {
        StringBuilder hexBuilder = new StringBuilder();
        // Loop through each character in the ASCII text
        for (int i = 0; i < asciiText.length(); i++) {
            char c = asciiText.charAt(i);
            // Add "0x" prefix if needed
            if (isPrefixEnabled) {
                hexBuilder.append("0x");
            }
            // Convert the character to its hexadecimal
    representation
            hexBuilder.append(String.format("%02X", (int) c));

            // Add delimiter if it's not the last character
            if (i < asciiText.length() - 1) {
                hexBuilder.append(getNumberDelimiter());
            }
        }
        return hexBuilder.toString();
    }

    // Calculate Checksum
    private void calculateChecksum() {
        String asciiText = txtArea.getText().trim(); // Get the
    ASCII text from the text area and remove leading/trailing
    spaces
        int selectedChecksumSize = getSelectedChecksumSize();
        int selectedXorMode = Jxorsum.getSelectedIndex();
        long calculatedChecksumValue = 0;
        byte[] bytes = asciiText.getBytes(); // Convert the ASCII
    text to bytes

        // Calculate the sum of ASCII values of characters in the
    text
        for (int i = 0; i < asciiText.length(); i++) {
            char c = asciiText.charAt(i);
            int asciiValue = (int) c;
            calculatedChecksumValue += asciiValue;
        }
        // Take the modulus to ensure the checksum fits within the
```

```
    selected size
419         calculatedChecksumValue %= (1L << selectedChecksumSize);
420 
421         switch (selectedXorMode) {
422             case 1: // 2's Complement
423                 int twosComplement32 = 0;
424                 for (byte b : bytes) {
425                     int value = b & 0xFF;
426                     twosComplement32 += value;
427                 }
428                 twosComplement32 = (~twosComplement32 + 1) & 0
    xFFFFFFFF;
429                 calculatedChecksumValue = twosComplement32 & ((1L
    << selectedChecksumSize) - 1);
430                 break;
431             case 2: // Xor
432                 byte[] xorBytes = new byte[8]; // Assuming a long
    (64-bit) checksum
433                 for (byte b : bytes) {
434                     int value = b & 0xFF;
435                     for (int i = 0; i < 8; i++) {
436                         xorBytes[i] ^= (value >> i) & 0x01;
437                     }
438                 }
439                 // Calculate the final XOR checksum value from the
    bytes
440                 calculatedChecksumValue = 0;
441                 for (int i = 0; i < 8; i++) {
442                     calculatedChecksumValue |= (xorBytes[i] << i);
443                 }
444                 break;
445         }
446         String checksumValueHex = String.format("%0" + (
    selectedChecksumSize / 4) + "X", calculatedChecksumValue);
447         JFxorsum.setText(checksumValueHex);
448         JFxorsum.updateUI();
449     }
450     private int getSelectedChecksumSize() {
451         String selectedChecksum = Jchecksum.getSelectedItem().
    toString();
452         switch (selectedChecksum) {
453             case "8-bit":
454                 return 8;
455             case "16-bit":
456                 return 16;
457             case "32-bit":
458                 return 32;
459             default:
460                 return 0; // Handle the default case or error
    condition appropriately
461         }
462     }
463 }
```

Listing 1: ASCII Converter Java Code