```java
package CyclicCode;

import java.applet.Applet;
import java.awt.*;

/**
 * <p>Title: The Cyclic Code Simulator</p>
 * <p>Description: to show the intermediate steps in encoding and decoding, allow the use
of different encoding polynomials and find out whether the selected one is a generator
polynomial </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Dongha Lee
 * @version 1.0
 */

public class ccApplet extends Applet {
  public ccApplet(){
    inAnApplet = true;
    complete = false;
    input_not_refreshed = false;
    poly_not_refreshed = false;
    welcome_string = "Welcome to the Cyclic Code Encoder/Decoder simulator.";
    step = 0;
  }

  public static void main(String[] args){
    ccApplet ccApp = new ccApplet();
    ccApp.inAnApplet = false;
    ccApp.init();
    ccApp.start();
  }

  public void init() {
    status = new ccStatus();
    status.init(9, 7, "111", "1100101", 0);
    step = 0;
    but_encode = new Button("Encode");
    but_reset = new Button("Reset");
    but_step = new Button("Step Run");
    but_run = new Button("Run");
    but_check = new Button("Check Polynomial");
    label_poly = new Label("Polynomial (n-k+1):");
    label_k = new Label("Input bits (k):");
    label_n = new Label("Code bits (n):");
    tf_poly = new TextField("1+x^1+x^2", 10);
    tf_k = new TextField("" + status.k, 5);
    tf_n = new Label("" + status.n);
    label_input = new Label("    Input:");
    label_inputout = new Label("    Input:");
    label_output = new Label("    Output:");
    tf_input = new TextField(status.input, 20);
    can_inputout = new TextField("", 20);
    can_output = new TextField("", 20);
    can_output = new TextField("", 20);
    can_poly = new TextField("", 10);
    can_reminder = new TextField("",10);
    can_inputout.setEditable(false);
    can_output.setEditable(false);
```

```
can_reminder.setEditable(false);
can_poly.setEditable(false);
message = new Label(welcome_string);
message.setAlignment(message.CENTER);
message.setForeground(Color.blue);
copy = new Label("Programmed by Dongha Lee 2003");
copy.setAlignment(copy.RIGHT);
quit = new Button("Quit");
GridBagLayout gridbaglayout = new GridBagLayout();
GridBagConstraints gridbagconstraints = new GridBagConstraints();
Panel panel = new Panel();
panel.setLayout(gridbaglayout);
gridbagconstraints.fill = 1;
gridbagconstraints.gridwidth = 2;
gridbagconstraints.weighty = 0.0D;
gridbagconstraints.weightx = 0.5D;
gridbaglayout.setConstraints(but_encode, gridbagconstraints);
panel.add(but_encode);
gridbaglayout.setConstraints(but_check, gridbagconstraints);
panel.add(but_check);
gridbagconstraints.gridwidth = 1;
gridbaglayout.setConstraints(but_reset, gridbagconstraints);
panel.add(but_reset);
gridbagconstraints.gridwidth = 1;
gridbaglayout.setConstraints(but_run, gridbagconstraints);
panel.add(but_run);
gridbagconstraints.gridwidth = 0;
gridbaglayout.setConstraints(but_step, gridbagconstraints);
panel.add(but_step);
gridbagconstraints.gridwidth = 2;
gridbaglayout.setConstraints(label_poly, gridbagconstraints);
panel.add(label_poly);
gridbagconstraints.gridwidth = 1;
gridbaglayout.setConstraints(tf_poly, gridbagconstraints);
panel.add(tf_poly);
gridbaglayout.setConstraints(can_poly, gridbagconstraints);
panel.add(can_poly);
gridbagconstraints.gridwidth = 2;
gridbaglayout.setConstraints(label_input, gridbagconstraints);
panel.add(label_input);
gridbagconstraints.gridwidth = 0;
gridbaglayout.setConstraints(tf_input, gridbagconstraints);
panel.add(tf_input);
gridbagconstraints.gridwidth = 2;
gridbaglayout.setConstraints(label_k, gridbagconstraints);
panel.add(label_k);
gridbaglayout.setConstraints(tf_k, gridbagconstraints);
panel.add(tf_k);
gridbagconstraints.gridwidth = 2;
gridbaglayout.setConstraints(label_inputout, gridbagconstraints);
panel.add(label_inputout);
gridbagconstraints.gridwidth = 0;
gridbaglayout.setConstraints(can_inputout, gridbagconstraints);
panel.add(can_inputout);
gridbagconstraints.gridwidth = 2;
gridbaglayout.setConstraints(label_n, gridbagconstraints);
panel.add(label_n);
gridbaglayout.setConstraints(tf_n, gridbagconstraints);
panel.add(tf_n);
```

```
    gridbagconstraints.gridwidth = 2;
    gridbaglayout.setConstraints(label_output, gridbagconstraints);
    panel.add(label_output);
    gridbaglayout.setConstraints(can_output, gridbagconstraints);
    panel.add(can_output);
    gridbagconstraints.gridwidth = 0;
    gridbaglayout.setConstraints(can_reminder, gridbagconstraints);
    panel.add(can_reminder);
    Label label3 = new Label();
    gridbaglayout.setConstraints(label3, gridbagconstraints);
    panel.add(label3);
    gridbaglayout.setConstraints(message, gridbagconstraints);
    panel.add(message);
    if (inAnApplet) {
      gridbaglayout.setConstraints(copy, gridbagconstraints);
      panel.add(copy);
    }
    else {
      gridbagconstraints.gridwidth = -1;
      gridbaglayout.setConstraints(copy, gridbagconstraints);
      panel.add(copy);
      gridbagconstraints.gridwidth = 0;
      gridbaglayout.setConstraints(quit, gridbagconstraints);
      panel.add(quit);
    }
    canvas = new ccCanvas();
    setLayout(new BorderLayout());
    add("Center", canvas);
    add("South", panel);
    canvas.setstatus(status);
    display();
  }

  public boolean keyDown(Event event, int i) {
    if (event.target == tf_input){
      switch (i) {
        case 48: // '0'
        case 49: // '1'
          input_not_refreshed = true;
          return super.keyDown(event, i);

        case 10: // '\n'
          input_not_refreshed = false;
          return super.keyDown(event, i);
      }
      input_not_refreshed = true;
      if (i < 48 || i > 122) {
        return super.keyDown(event, i);
      }
      else {
        output("The input string may only consist of bit values (0 or 1).");
        return true;
      }
    }
    else if(event.target == tf_poly)
    {
      switch (i) {
        case 10: // '\n'
          poly_not_refreshed = false;
          return super.keyDown(event, i);
      }
      poly_not_refreshed = true;
      return super.keyDown(event, i);
    }
    else return super.keyDown(event, i);
  }

  public boolean action(Event event, Object obj) {
    if (event.target == tf_input || input_not_refreshed) {
      String s = tf_input.getText();
      for (int i1 = 0; i1 < s.length(); i1++)
        if (!s.substring(i1, i1 + 1).equals("0") &&
            !s.substring(i1, i1 + 1).equals("1")) {
          output("Illegal characters in input string.");
          tf_input.setText(status.input);
          input_not_refreshed = false;
          return true;
        }

      int k1 = status.encode != 0 ? status.n : status.k;
      if (s.length() % k1 == 0) {
        status.input = s;
        restart();
        output("Input string changed successfully.");
      }
      else {
        status.input = correct_input_string_length(s);
        tf_input.setText(status.input);
        restart();
        if (status.encode == 0)
          output("Input string modified to match number of input bits 'k'.");
        else
          output("Input string modified to match number of code bits 'n'.");
      }
      input_not_refreshed = false;
      if (event.target == tf_input)
        return true;
    }
    if (event.target == tf_poly || poly_not_refreshed) {
      poly_not_refreshed = false;
      String s1 = parse_poly(tf_poly.getText());
      if (s1 != null) {
        int j1 = status.k;
        String s3 = status.input;
        if (status.encode == 1)
          s3 = correct_input_string_length(s3, j1, s1.length());
        status.init( (s1.length() + j1) - 1, j1, s1, s3, status.encode);
        tf_input.setText(status.input);
        tf_n.setText(String.valueOf(status.n));
        restart();
        but_encode.setEnabled(true);
        but_reset.setEnabled(true);
        but_step.setEnabled(true);
        tf_k.setEnabled(true);
        tf_input.setEnabled(true);
      }
      else {
        but_encode.setEnabled(false);
        but_reset.setEnabled(false);
        but_step.setEnabled(false);
        tf_k.setEnabled(false);
        tf_input.setEnabled(false);
      }
      if (event.target == tf_poly)
        return true;
    }
    if (event.target == quit) {
      System.exit(0);
      return true;
    }
    if (event.target == tf_k) {
      int k = status.k;
      try {
        k = Integer.valueOf(tf_k.getText()).intValue();
        if (k > 64 || k < 1)
          throw new NumberFormatException();
      }
      catch (NumberFormatException numberformatexception) {
        tf_k.setText(String.valueOf(status.k));
        output("Error, value for K out of range or non-numeric.");
        return true;
      }
      String s2 = correct_input_string_length(status.input, k);
      status.init( (status.gates.length() + k) - 1, k, status.gates, s2,
                  status.encode);
      tf_n.setText("" + status.n);
      tf_k.setText("" + status.k);
      tf_input.setText(status.input);
      restart();
      output("Number of input bits set. Input string be modified accordingly.");
      return true;
    }
    if (event.target == but_encode) {
      if (status.valid()) {
        if (status.encode == 0) {
          but_encode.setLabel("Decode");
          label_input.setText("    Encoded:");
          status.encode = 1;
          status.input = correct_input_string_length(status.input);
          int l = can_output.getText().length();
          if (complete && l > 0 && l % status.n == 0)
            status.input = correct_input_string_length(can_output.getText());
          else
            status.input = correct_input_string_length(status.input);
        }
        else {
          but_encode.setLabel("Encode");
          label_input.setText("    Input:");
          status.encode = 0;
          status.input = correct_input_string_length(status.input);
        }
      }
      else {
        output("Status error.");
      }
      tf_input.setText(status.input);
      restart();
      if (status.valid())
        output("Mode changed to " + but_encode.getLabel() +
               ". Simulation restarting.");
      return true;
    }
    if (event.target == but_check) {
      String op;
      int e = status.encode, j;
      int k = status.k;
      int n = status.n;
      status.encode = 1;
      String s2 = correct_input_string_length("", k+1);
      s2 = "1" + s2.substring(0,k-1) + "1";
      status.init( (status.gates.length() + k+1) - 1, k+1, status.gates, s2,
                  status.encode);
      restart();
      do{
        j = step;
        step = status.calculate(step + 1, true);
        if(j==step){ complete = true; break; }
      }while(true);
      if(status.output.substring(0,status.n-status.k).indexOf("1")>=0){
        op = "This is NOT a Generator Polynomial!";
      }else{
        op = "This is a Generator Polynomial!";
      }
      status.k = k;
      status.n = n;
      status.encode = e;
      status.init( (status.gates.length() + k) - 1, k, status.gates, tf_input.getText(),
                  status.encode);
      restart();
      output(op);
      return true;
    }
    if (event.target == but_reset) {
      restart();
      return true;
    }
    if (event.target == but_run) {
      int j;
      restart();
      do{
        j = step;
        step = status.calculate(step + 1, true);
        if(j==step){ complete = true; break; }
      }while(true);
      display();
      output("Simulation complete.");
      complete = true;
      if(status.encode==1){
        if(status.output.substring(0,status.n-status.k).indexOf("1")>=0){
          label_output.setText("    Output:    Falut!");
          label_output.setForeground(Color.red);
        }else{
          label_output.setText("    Output:    No Fault!");
          label_output.setForeground(Color.blue);
        }
      }
```

```
    return true;
  }
  if (event.target == but_step) {
    int j = step;
    if(!complete){
      step = status.calculate(step + 1, true);
    }
    if (j == step) {
      output("Simulation complete.");
      complete = true;
      if(status.encode==1){
        if(status.output.substring(0,status.n-status.k).indexOf("1")>=0){
          label_output.setText("    Output:    Falut detected!");
          label_output.setForeground(Color.red);
        }else{
          label_output.setText("    Output:    No Fault!");
          label_output.setForeground(Color.blue);
        }
      }
    }
    else {
      complete = false;
    }
    display();
    return true;
  }
  else {
    return false;
  }
}

String correct_input_string_length(String s) {
  return correct_input_string_length(s, status.k, status.gates.length());
}

String correct_input_string_length(String s, int i) {
  return correct_input_string_length(s, i, status.gates.length());
}

String correct_input_string_length(String s, int i, int j) {
  int k = status.encode != 0 ? (j + i) - 1 : i;
  if (s.length() > k) {
    s = s.substring(0, k * (s.length() / k));
  }
  else {
    for (int l = s.length(); l < k; l++)
      s = s + "0";

  }
  return s;
}

void display() {
  if(status.encode==0 || !complete){
    can_reminder.setText("");
    can_output.setText(status.output);
  }else{
    can_output.setText(status.output.substring(status.n-status.k));
    can_reminder.setText(status.output.substring(0,status.n-status.k));
  }
  can_poly.setText(status.gates);
  can_inputout.setText(status.inputout);
  canvas.repaint();
  if (status.message.length() > 0)
    output(status.message);
}

void restart() {
  if (status.valid()) {
    step = 0;
    complete = false;
    status.bits = "";
    status.inputout = "";
    status.output = "";
    label_output.setText("    Output:");
    label_output.setForeground(Color.black);
    display();
    output(welcome_string);
  }
  else {
    output("Status invalid.");
  }
}

void output(String s) {
  if (message == null) {
    return;
  }
  else {
    message.setText(s);
    return;
  }
}

void output(String s, Color c) {
  if (message == null) {
    return;
  }
  else {
    Color sc = message.getForeground();
    message.setForeground(Color.red );
    message.setText(s);
    message.setForeground(sc);
    return;
  }
}


String parse_poly(String s) {
  String s1 = "";
  int j = 0;
  int k = 0;
  int l = 0;
  try {
    for (int i = 0; i < s.length(); i++) {
      String s3 = s.substring(i, i + 1);
      if (l == 0) {
        if (!s3.equals(" "))
    if (s3.equals("1")) {
      s1 = "1" + s1;
      l++;
    }
    else {
      throw new StringIndexOutOfBoundsException();
    }
}
else {
  switch (k) {
    default:
      break;

    case 0: // '\0'
    case 4: // '\004'
      if (s3.equals(" ") || s3.equals("+"))
        break;
      if (s3.equals("0") || s3.equals("1")) {
        s1 = s3 + s1;
        l++;
        k = 0;
        break;
      }
      if (s3.equals("x") || s3.equals("X"))
        k = 1;
      else
        throw new StringIndexOutOfBoundsException();
      break;

    case 2: // '\002'
      if (s3.equals("*")) {
        k = 3;
        j = 0;
      }
      else {
        throw new StringIndexOutOfBoundsException();
      }
      break;

    case 1: // '\001'
      j = 0;
      if (s3.equals(" "))
        break;
      if (s3.equals("^")) {
        k = 3;
        break;
      }
      if (s3.equals("*")) {
        k = 2;
        break;
      }
      if (s3.equals("+") && l == 1) {
        s1 = "1" + s1;
        k = 0;
        l = 2;
        break;
      }// fall through

    case 3: // '\003'
```

```
          if (s3.equals(" "))
            break;
          int i1;
          try {
            i1 = Integer.valueOf(s3).intValue();
          }
          catch (NumberFormatException numberformatexception) {
            i1 = -1;
          }
          if (i1 >= 0) {
            j = 10 * j + i1;
            k = 3;
            break;
          }
          if (s3.equals(" ") || s3.equals("+")) {
            if (s3.equals(" "))
              k = 0;
            if (s3.equals("+"))
              k = 4;
            if (j < l)
              throw new StringIndexOutOfBoundsException();
            for (int k1 = l; k1 < j; k1++){
              s1 = "0" + s1;
            }
            s1 = "1" + s1;
            l = j + 1;
            j = 0;
          }
          else {
            throw new StringIndexOutOfBoundsException();
          }
          break;
      }
    }
  }

  if (k == 1 && l == 1) {
    s1 = "1" + s1;
    k = 0;
    l = 2;
  }
  if (k == 3) {
    if (j < l)
      throw new StringIndexOutOfBoundsException();
    for (int j1 = l; j1 < j; j1++){
      s1 = "0" + s1;
    }
    s1 = "1" + s1;
    l = j + 1;
    j = 0;
  }
  else
  if (k != 0)
    throw new StringIndexOutOfBoundsException();
  if (l > 33 || l < 2)
    throw new StringIndexOutOfBoundsException();
}
catch (StringIndexOutOfBoundsException stringindexoutofboundsexception) {
  output("General polynomial error. Example form: \"1+x+x^3\".");
```

```
      return null;
    }
    return s1;
  }

  public void start() {
    canvas.setsize();
    output(welcome_string);
  }

  public void stop() {
    output("Goodbye for today.");
  }

  ccCanvas canvas;
  ccStatus status;
  boolean inAnApplet;
  boolean complete;
  boolean input_not_refreshed;
  boolean poly_not_refreshed;
  Label label_poly;
  Label label_k;
  Label label_n;
  TextField tf_poly;
  TextField tf_k;
  Label tf_n;
  TextField tf_input;
  TextField can_inputout;
  TextField can_output;
  TextField can_poly;
  Label label_input;
  Label label_inputout;
  Label label_output;
  TextField can_reminder;
  String welcome_string;
  Label message;
  Label copy;
  Button quit;
  Button but_encode;
  Button but_reset;
  Button but_step;
  Button but_run;
  Button but_check;
  int step;
  private static final int MAX_K = 64;
  private static final int MAX_ORDER = 33;
}
```

```
package CyclicCode;

import java.awt.*;
import java.awt.Canvas;

/**
 * <p>Title: The Cyclic Code Simulator</p>
 * <p>Description: to show the intermediate steps in encoding and decoding, allow the use
of different encoding polynomials and find out whether the selected one is a generator
polynomial </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Dongha Lee
 * @version 1.0
 */

public class ccCanvas extends Canvas {
  ccStatus status;
  Dimension d;

  public ccCanvas() {
    status = null;
  }

  void init() {
  }

  boolean setstatus(ccStatus ccs) {
    if (ccs == null || !ccs.valid()) {
      return false;
    }
    else {
      status = ccs;
      return true;
    }
  }

  void setsize() {
    d = getSize();
  }

  void setstructure(String s) {
  }

  void setstatus(String s, String s1) {
  }

  boolean toosmall() {
    return d.width <= 100 || d.height <= 50;
  }

  void drawgrid() {
    drawgrid(getGraphics());
  }

  void drawgrid(Graphics g) {
    byte byte3 = 5;
    Dimension dimension = getSize();
    if (status == null || !status.valid() || toosmall())
  return;
int i2 = (status.n - status.k - 1) + 3;
int l = (4 * dimension.height) / 6;
int k = (2 * dimension.height) / 6;
int j1 = dimension.height / 6;
int i1 = dimension.width / i2;
int j = i1;
byte byte0 = 10;
byte byte1 = 11;
byte byte2 = 20;
byte byte4 = 50;
int l1;
int k1 = l1 = 10;
if (i1 < 80)
  k1 = i1 / 8;
if (j1 < 50)
  l1 = j1 / 5;
drawharrow(g, k, j - i1 / 2, j, 0, byte1, k1);
drawbox(g, j, k, byte1, byte1);
if (status.bits != null && status.bits.length() > 0)
  drawBit(g, Color.black, status.bits.substring(0, 1), j, k, true);
int i;
for (i = 0; i < status.n - status.k - 1; i++) {
  if (status.gates.substring(i + 1, i + 2).equals("1")) {
    drawharrow(g, k, j + i * i1, j + i * i1 + i1 / 2, byte1, byte0, k1);
    drawplus(g, j + i * i1 + i1 / 2, k, byte0);
    drawvarrow(g, j + i * i1 + i1 / 2, l, k, 0, byte0, l1);
    drawharrow(g, k, j + i * i1 + i1 / 2, j + (i + 1) * i1, byte0, byte1,
               k1);
  }
  else {
    drawharrow(g, k, j + i * i1, j + (i + 1) * i1, byte1, byte1, k1);
  }
  drawbox(g, j + (i + 1) * i1, k, byte1, byte1);
  if (status.bits != null && status.bits.length() > i + 1)
    drawBit(g, Color.black, status.bits.substring(i + 1, i + 2),
            j + (i + 1) * i1, k, true);
}

drawvarrow(g, j - i1 / 2, l, k, 0, 0, 0);
drawplus(g, j + (i + 1) * i1, k, byte0);
if(status.isEncoding()){
  drawharrow(g, l, j + (i + 1) * i1 + k1, j - i1 / 2, byte0, 0, 0);
  drawvarrow(g, j + (i + 1) * i1, l, k, 0, byte0, l1);
  drawharrow(g, l,   j + (i + 1) * i1 + i1 / 4,j + (i + 1) * i1 , byte1, 0, k1);
  g.drawString("Data in", j + (i + 1) * i1 + i1 / 4+5, l );
  g.drawString("Encoded out", j + (i + 1) * i1 + i1 / 4+5, k);
}
else{
  drawharrow(g, l, j + (i + 1) * i1 + k1+i1/4, j - i1 / 2, byte0, 0, 0);
  drawvarrow(g, j + (i + 1) * i1+i1/4, k, l, 0, 0, k1);
  drawvarrow(g, j + (i + 1) * i1, (l+k)/2 , k+l1, 0, 0, k1);
  drawharrow(g, (k+l)/2,   j + (i + 1) * i1 + i1 / 2,j + (i + 1) * i1 , byte1, 0, k1);
  g.drawString("Encoded in", j + (i + 1) * i1 + i1 / 4+5, (k+l)/2);
  g.drawString("Decoded out", j + (i + 1) * i1 + i1 / 4+5, k);
}
if (status.controlbits != null && status.controlbits.length() >= 4)
  drawBit(g, Color.red, status.controlbits.substring(0, 1),
          j + (i + 1) * i1 + 10, (l - j1 / 2) + 10);
  drawharrow(g, k, j + i * i1, j + (i + 1) * i1 , byte1, byte0, k1);
  drawharrow(g, k, j + (i + 1) * i1 , j + (i + 1) * i1 + i1 / 4, byte1, 0, k1);

  if (status.controlbits != null && status.controlbits.length() >= 4)
    drawBit(g, Color.blue, status.controlbits.substring(3, 4),
            j + (i + 1) * i1 + i1 / 2 + 10, k + 10);
}

private void drawBit(Graphics g, Color color, String s, int i, int j) {
  drawBit(g, color, s, i, j, false);
}

private void drawBit(Graphics g, Color color, String s, int i, int j,
                     boolean flag) {
  if (g == null || s == null || color == null)
    return;
  Color color1 = g.getColor();
  g.setColor(color);
  if (flag)
    g.drawString(s, i - 3, j + 5);
  else
    g.drawString(s, i, j);
  g.setColor(color1);
}

private void drawharrow(Graphics g, int i, int j, int k, int l, int i1,
                        int j1) {
  if (g == null)
    return;
  if (j < k) {
    j += l;
    k -= i1;
  }
  else {
    j -= l;
    k += i1;
  }
  g.drawLine(j, i, k, i);
  if (j1 > 0) {
    int k1 = j1;
    if (j < k)
      k1 *= -1;
    Polygon polygon = new Polygon();
    polygon.addPoint(k, i);
    polygon.addPoint(k + k1, i + j1 / 2);
    polygon.addPoint(k + k1, i - j1 / 2);
    g.fillPolygon(polygon);
  }
}

private void drawvarrow(Graphics g, int i, int j, int k, int l, int i1,
                        int j1) {
  if (g == null)
    return;
  if (j < k) {
    j += l;
    k -= i1;
  }
  else {
    j -= l;
    k += i1;
  }
  g.drawLine(i, j, i, k);
  if (j1 > 0) {
    int k1 = j1;
    if (j < k)
      k1 *= -1;
    Polygon polygon = new Polygon();
    polygon.addPoint(i, k);
    polygon.addPoint(i + j1 / 2, k + k1);
    polygon.addPoint(i - j1 / 2, k + k1);
    g.fillPolygon(polygon);
  }
}
private void drawbox(Graphics g, int i, int j, int k, int l)
{
  drawbox(g, i, j, k, l, false);
}

private void drawbox(Graphics g, int i, int j, int k, int l, boolean flag) {
  if (g == null) {
    return;
  }
  else {
    if(!flag){
      g.drawRect(i - k, j - l, k * 2, l * 2);
    }else{
      Color color1 = g.getColor();
      g.setColor(Color.cyan);
      g.fillRect(i - k, j - l, k * 2, l * 2);
      g.setColor(color1);
    }
    return;
  }
}

private void drawplus(Graphics g, int i, int j, int k) {
  int l = k / 2 + 1;
  if (g == null) {
    return;
  }
  else {
    g.drawOval(i - k, j - k, 2 * k, 2 * k);
    g.drawLine(i - l, j, i + l, j);
    g.drawLine(i, j - l, i, j + l);
    return;
  }
}

private void drawandgate(Graphics g, int i, int j, int k) {
  if (g == null) {
    return;
  }
  else {
    g.drawArc(i - k, j - k / 2, k, k, 90, 180);
    g.drawRect( (i - k) + k / 2, j - k / 2, k * 2 - k / 2, k);
    return;
    }
  }

  public void paint(Graphics g) {
    drawgrid(g);
  }
}
```

```
package CyclicCode;

/**
 * <p>Title: The Cyclic Code Simulator</p>
 * <p>Description: to show the intermediate steps in encoding and decoding, allow the use
of different encoding polynomials and find out whether the selected one is a generator
polynomial </p>
 * <p>Copyright: Copyright (c) 2003</p>
 * <p>Company: </p>
 * @author Dongha Lee
 * @version 1.0
 */

public class ccStatus {
  public static final int ENCODE = 0;
  public static final int DECODE = 1;
  int n;
  int k;
  int encode;
  String input;
  String gates;
  String controlbits;
  String bits;
  String inputout;
  String output;
  String message;

  ccStatus() {
    input = null;
    gates = null;
    controlbits = null;
    bits = null;
    inputout = "";
    output = "";
    message = "";
    n = 0;
  }

  boolean init(int i, int j, String s, String s1, int l) {
    if (!valid(i, j, s, s1, l))
      return false;
    n = i;
    k = j;
    input = s1;
    gates = s;
    encode = l;
    bits = "";
    controlbits = "      ";
    for (int i1 = 0; i1 < i - j ; i1++) {
      bits += "0";
    }

    //calculate(0);
    return true;
  }

  boolean invalid_input(String s, int i, int j, int l) {
    int i1 = i != 0 ? j : l;
    return s == null || s.length() < 0 || s.length() % i1 != 0;
```

```
}

boolean invalid_gates(String s, int i, int j) {
  return s == null || s.length() < 0 || s.length() != (i - j) + 1;
}

boolean valid() {
  return valid(n, k, gates, input, encode);
}

boolean valid(int i, int j, String s, String s1, int l) {
  return i != 0 && j != 0 && !invalid_gates(s, i, j) &&
      !invalid_input(s1, l, i, j);
}

boolean isEncoding(){
  return (encode == ENCODE);
}

public String toString() {
  String s;
  if (!valid())
    s = "Invalid";
  else
    s = "N=" + n + "  K=" + k + "  gates=" + gates + "  input=" + input;
  return s;
}

int calculate(int i, boolean flag){
  int p[] = new int[n - k];
  if(bits==""){
  for (int j = 0; j < n - k ; j++)
    bits += "0";
  }
  for (int j = 0; j < n - k; j++)
    p[j] = Integer.valueOf(bits.substring(j, j+1)).intValue();;

  int nk = n - k - 1;
  int in = 0;
  try{
    if(encode == 0){
      in = Integer.valueOf(input.substring(k - i, k-i+1)).intValue();
    }else{
      in = Integer.valueOf(input.substring(n - i, n-i+1)).intValue();
    }
    inputout = String.valueOf(in) + inputout;
  }catch(StringIndexOutOfBoundsException stringindexoutofboundsexception){
      in = 0;
  }

  int out = ( p[nk] + in ) % 2;
  if(encode == 1) in = out;
  for(int j = nk; j>0; j--)
  {
    if(!gates.substring(j, j+1).equalsIgnoreCase("1")){
      p[j] = p[j - 1];
    }else{
      p[j] = (p[j - 1] + in) % 2;
    }
    }
    p[0] = in; bits = "";
    for (int j = 0; j < n - k; j++)
      bits += p[j];
    output = String.valueOf(out) + output;
    if(i>=n) return i-1;
    return i;
  }
}
```