Java source code

GammaCellApplet.java

```
import java.awt.*;
import java.applet.*;

public class GammaCellApplet extends Applet
{
   public void init()
   {  if (f == null)
      {  f = new GammaCell();
      }
      f.show();
   }

   public void start()
   {  if (f == null)
      {  f = new GammaCell();
      }
      f.setVisible(true);
   }

   public void stop()
   {  f.setVisible(false);
   }

   public void paint(Graphics g)
   {  g.drawString("Gamma Cell Irradiation Dose Applet",10,25);
   }

   private Frame f;
}
```

GammaCell.java

```
/* written by Brian Chow
 * created February 27, 1998
 * last modified May 11, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
public class GammaCell extends Frame implements ActionListener
{  /**
    *
    */
   public GammaCell()
   {  experimentUnits = new Units();
      experimentDate = new Day();

      sources = new BundleList();
      sources.add(new RadBundle("1994",new Day(1994,3,11),10794,12));
      sources.add(new RadBundle("1979",new Day(1979,9,19),9950,12));
      sources.add(new RadBundle("1963",new Day(1963,1,14),10600,20));
```



```
      sources.setDate(experimentDate);
      sources.setExpBundle("1979");

      experimentFixture = new AnnularFixture(new Coordinate(),
        sources.getExpBundle(),1,6);
//      experimentTarget = new TargetLine(new Coordinate(-1,0,0),
//        new Coordinate(1,0,0),11);
      experimentTarget = new TargetRect(new Coordinate(10,0,0),6,3,3);
      setOptionLabels();

      setTitle("Gamma Cell Dosage");
      setLayout(new BorderLayout());
      optionsPanel = new Panel();
      constructOptionsPanel(optionsPanel);
      add(optionsPanel,"Center");

// optionButtonsArray[3].setEnabled(false);

      actionButtonsPanel = new Panel();
      aboutButton = new Button("About");
      aboutButton.addActionListener(this);
      actionButtonsPanel.add(aboutButton);
      runButton = new Button("Calculate");
      runButton.addActionListener(this);
      actionButtonsPanel.add(runButton);
      quitButton = new Button("Quit");
      quitButton.addActionListener(this);
      actionButtonsPanel.add(quitButton);
      add(actionButtonsPanel,"South");

      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  quitProgram();
         }
      });

      pack();
   }

   public void actionPerformed(ActionEvent evt)
   {  String arg = evt.getActionCommand();
      if (arg.equals(optionButtonLabels[0]))
      {  UnitDialog experimentUnitDialog;
         experimentUnitDialog = new UnitDialog(this);
         experimentUnitDialog.showDialog();
      }
      else if (arg.equals(optionButtonLabels[1]))
      {  DateDialog experimentDateDialog;
         Day changedDate = (Day)experimentDate.clone();
         experimentDateDialog = new DateDialog(this,
           "Change Experiment Date",changedDate);
         if (experimentDateDialog.showDialog())
         {  experimentDate = changedDate;
            sources.setDate(experimentDate);
         }
      }
      else if (arg.equals(optionButtonLabels[2]))
      {  BundleDialog configureBundleDialog;
         BundleList changedBL = (BundleList)sources.clone();
         configureBundleDialog = new BundleDialog(this,changedBL,
           experimentDate);
```



```
         if (configureBundleDialog.showDialog())
         {  sources = changedBL;
            experimentFixture.setSource(sources.getExpBundle());
         }
      }
      else if (arg.equals(optionButtonLabels[3]))
      {  FixtureDialog selectFixtureDialog;
         selectFixtureDialog = new FixtureDialog(this,experimentFixture,
         sources.getExpBundle());
         experimentFixture = selectFixtureDialog.getFixture();
      }
      else if (arg.equals(optionButtonLabels[4]))
      {  TargetDialog selectTargetDialog;
         selectTargetDialog = new TargetDialog(this,experimentTarget);
         experimentTarget = selectTargetDialog.getTarget();
      }
      else if (arg.equals("About"))
      {  Dialog d = new AboutDialog(this);
         d.show();
      }
      else if (arg.equals("Calculate"))
      {  experimentTarget.calculate(experimentFixture);
      }
      else if (arg.equals("Quit"))
      {  quitProgram();
      }
      setOptionLabels();
   }

   /**
    *
    */
   public static void main(String[] args)
   {  Frame f = new GammaCell();
      f.show();
   }

   private void setOptionLabels()
   {  optionLabels[0] = experimentUnits.toString();
      optionLabels[1] = experimentDate.toString();
      optionLabels[2] = sources.getExpBundle().toString();
      optionLabels[3] = experimentFixture.toString();
      optionLabels[4] = experimentTarget.toString();
      for (int i = 0;i < numOptions;i++)
      {  if (optionLabelsArray[i] == null)
         {  optionLabelsArray[i] = new Label(optionLabels[i]);
         }
         else
         {  optionLabelsArray[i].setText(optionLabels[i]);
         }
      }
   }

   private void constructOptionsPanel(Panel p)
   {  p.setLayout(new GridBagLayout());
      GridBagConstraints gbc = new GridBagConstraints();
      gbc.weighty = 100;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      for (int i = 0;i < numOptions;i++)
      {  gbc.gridy = i;
```



```
         optionPanelsArray[i] = new Panel(new GridLayout(3,1));
         optionTitlesArray[i] = new Label(optionTitles[i]);
         optionTitlesArray[i].setFont(titleFont);
         optionPanelsArray[i].add(optionTitlesArray[i]);
         optionLabelsArray[i].setFont(labelFont);
         optionPanelsArray[i].add(optionLabelsArray[i]);
         Label tempLabel = new Label(" ");
         tempLabel.setFont(labelFont);
         optionPanelsArray[i].add(tempLabel);
         gbc.weightx = 100;
         gbc.fill = GridBagConstraints.BOTH;
         gbc.gridx = 1;
         p.add(optionPanelsArray[i],gbc);

         optionButtonsArray[i] = new Button(optionButtonLabels[i]);
         optionButtonsArray[i].addActionListener(this);
         gbc.weightx = 0;
         gbc.fill = GridBagConstraints.HORIZONTAL;
         gbc.gridx = 2;
         p.add(optionButtonsArray[i],gbc);
      }
   }

   private void quitProgram()
   {  setVisible(false);
      if (System.getSecurityManager() == null)
      {  System.exit(0);
      }
   }

   private static final int numOptions = 5;
   private static final String[] optionTitles = {"Units",
     "Experiment Date","Source Bundle","Source Fixture","Target"};
   private static final String[] optionButtonLabels = {"Set Units",
     "Set Date","Select Bundle","Configure Fixtures",
     "Configure Target"};

   private Day experimentDate;
   private Units experimentUnits;
   private BundleList sources;
   private Fixture experimentFixture;
   private Target experimentTarget;
   private Font titleFont = new Font("Serif",Font.BOLD,14);
   private Font labelFont = new Font("San Serif",Font.PLAIN,12);
   private String[] optionLabels = new String[numOptions];
   private Panel optionsPanel,actionButtonsPanel;
   private Panel[] optionPanelsArray = new Panel[numOptions];
   private Label[] optionTitlesArray = new Label[numOptions];
   private Label[] optionLabelsArray = new Label[numOptions];
   private Button aboutButton,runButton,quitButton;
   private Button[] optionButtonsArray = new Button[numOptions];
}
```

UnitDialog.java

```
/* written by Brian Chow
 * created March 18, 1998
 * last modified April 5, 1998
 */
/**
```

```
 *
 */
import java.awt.*;
import java.awt.event.*;

class UnitDialog extends Dialog implements TextListener,ActionListener,
  ItemListener
{  /**
    *
    */
   public UnitDialog(Frame parent)
   {  super(parent,"Units Configuration",true);

      setLayout(new BorderLayout());

      Panel pMain = new Panel(new BorderLayout());

      pUnits = new Panel(new GridLayout(2,3));

      Label lDose = new Label("Dose Units:");
      pUnits.add(lDose);
      cbGDose = new CheckboxGroup();
      addCheckbox(pUnits,"Rads",cbGDose,Units.getDoseUnits().
        equals("Rads"));
      addCheckbox(pUnits,"Grays",cbGDose,Units.getDoseUnits().
        equals("Grays"));
      Label lLength = new Label("Length Units:");
      pUnits.add(lLength);
      cbGLength = new CheckboxGroup();
      addCheckbox(pUnits,"Inches",cbGLength,Units.getLengthUnits().
        equals("Inches"));
      addCheckbox(pUnits,"Centimeters",cbGLength,Units.getLengthUnits().
        equals("Centimeters"));
      pMain.add(pUnits,"North");

      pAccuracy = new Panel(new FlowLayout(FlowLayout.LEFT));
      Label lAccuracy = new Label("Length Accuracy:  1/");
      pAccuracy.add(lAccuracy);
      tAccuracy = new TextField("" + Units.getLengthUnits().
        getAccuracy(),5);
      tAccuracy.addTextListener(this);
      pAccuracy.add(tAccuracy);
      lLengthUnits = new Label(Units.getLengthUnits().toString());
      pAccuracy.add(lLengthUnits);
      pMain.add(pAccuracy,"South");

      add(pMain,"Center");

      pButtons = new Panel(new FlowLayout());
      okButton = new Button("OK");
      cancelButton = new Button("Cancel");
      pButtons.add(okButton);
      pButtons.add(cancelButton);
      okButton.addActionListener(this);
      cancelButton.addActionListener(this);
      add(pButtons,"South");

      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  setVisible(false);
            dispose();
```



```
         }
      });

      pack();
   }

   public boolean showDialog()
   {  show();
      return ok;
   }

   public void actionPerformed(ActionEvent evt)
   {  String arg = evt.getActionCommand();
      if (arg.equals("OK"))
      {  ok = true;
         setVisible(false);
         UnitLength lengthUnit = Units.getLengthUnits();
         lengthUnit.setAccuracy(Integer.parseInt(tAccuracy.getText().
           trim()));
         lengthUnit.setUnits(cbGLength.getSelectedCheckbox().getLabel());
         UnitDose doseUnit = Units.getDoseUnits();
         doseUnit.setUnits(cbGDose.getSelectedCheckbox().getLabel());
         dispose();
      }
      else if (arg.equals("Cancel"))
      {  setVisible(false);
         dispose();
      }
   }

   public void textValueChanged(TextEvent evt)
   {  try
      {  int a = Integer.parseInt(tAccuracy.getText().trim());
         if (a > 1)
         {  okButton.setEnabled(true);
         }
         else
         {  okButton.setEnabled(false);
         }
      }
      catch(Exception e)
      {  okButton.setEnabled(false);
      }
   }

   public void itemStateChanged(ItemEvent evt)
   {  if (evt.getStateChange() == ItemEvent.DESELECTED)
      {  return;
      }
      String s = (String)evt.getItem();
      if (s.equals("Centimeters") || s.equals("Inches"))
      {  lLengthUnits.setText(s);
      }
   }

   private void addCheckbox(Panel p,String name,CheckboxGroup cbG,boolean
     state)
   {  Checkbox cb = new Checkbox(name,cbG,state);
      cb.addItemListener(this);
      p.add(cb);
   }
```



```
   private boolean ok = false;
   private Panel pUnits,pAccuracy,pButtons;
   private CheckboxGroup cbGDose,cbGLength;
   private TextField tAccuracy;
   private Label lLengthUnits;
   private Button okButton,cancelButton;
}
```

DateDialog.java

```
/* written by Brian Chow
 * created March 1, 1998
 * last modified April 5, 1998
 */

import java.awt.*;
import java.awt.event.*;

class DateDialog extends Dialog implements TextListener,ActionListener
{  /**
    *
    */
   public DateDialog(Frame parent,String title,Day d)
   {  super(parent,title,true);
      expDate = d;

      setLayout(new BorderLayout());

      pDate = new Panel(new GridLayout(3,2));
      tMonth = addTextField(pDate,"Month (mm)","" + d.getMonth(),5);
      tDay = addTextField(pDate,"Day (dd)","" + d.getDay(),5);
      tYear = addTextField(pDate,"Year (yyyy)","" + d.getYear(),5);
      add(pDate,"Center");

      pButtons = new Panel(new FlowLayout());
      okButton = new Button("OK");
      cancelButton = new Button("Cancel");
      pButtons.add(okButton);
      pButtons.add(cancelButton);
      okButton.addActionListener(this);
      cancelButton.addActionListener(this);
      add(pButtons,"South");

      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  setVisible(false);
            dispose();
         }
      });
      pack();
   }

   public boolean showDialog()
   {  show();
      return ok;
   }

   public void actionPerformed(ActionEvent evt)
   {  String arg = evt.getActionCommand();
```



```
      if (arg.equals("OK"))
      {  ok = true;
         setVisible(false);
         dispose();
      }
      else if (arg.equals("Cancel"))
      {  setVisible(false);
         dispose();
      }
   }

   public void textValueChanged(TextEvent evt)
   {  try
      {  int m = Integer.parseInt(tMonth.getText().trim());
         int d = Integer.parseInt(tDay.getText().trim());
         int y = Integer.parseInt(tYear.getText().trim());
         expDate.setDay(y,m,d);
         okButton.setEnabled(true);
      }
      catch(Exception e)
      {  okButton.setEnabled(false);
      }
   }

   private TextField addTextField(Panel p,String name,String field,
     int col)
   {  TextField t = new TextField(field,col);
      t.addTextListener(this);
      p.add(new Label(name));
      p.add(t);
      return t;
   }

   private Day expDate;
   private boolean ok = false;
   private Panel pDate,pButtons;
   private TextField tMonth,tDay,tYear;
   private Button okButton,cancelButton;
}
```

BundleDialog.java

```
/* written by Brian Chow
 * created February 27, 1998
 * last modified March 27, 1998
 */

import java.awt.*;
import java.awt.event.*;
import java.util.*;

class BundleDialog extends Dialog implements ActionListener
{  /** */

   public BundleDialog(Frame parent,BundleList bL,Day d)
   {  super(parent,"Source Bundle Selection",true);
      bundles = bL;
      expDate = d;

      setLayout(new BorderLayout());
```

```
   Panel configurePanel = new Panel();
   configurePanel.setLayout(new BorderLayout());

   Panel labelPanel = new Panel();
   labelPanel.setLayout(new GridLayout(1,5));
   addLabel(labelPanel,"Source","To Use");
   addLabel(labelPanel,"Pencils","per Bundle");
   addLabel(labelPanel,"Total Measured","Activity (Ci)");
   addLabel(labelPanel,"Measurement","Date");
   addLabel(labelPanel,"One Pencil Activity","(Ci) on " +
     expDate.toString());
   configurePanel.add(labelPanel,"North");

   pBundles = new Panel();
   pBundles.setLayout(new GridLayout(bundles.size(),1));
   bundlesCBG = new CheckboxGroup();
   Enumeration bundlesEnum = bundles.elements();
   while (bundlesEnum.hasMoreElements())
   {  RadBundle currentBundle = (RadBundle)bundlesEnum.nextElement();
      addCBBundle(pBundles,bundlesCBG,currentBundle,bundles.
        getExpBundle() == currentBundle);
   }
   configurePanel.add(pBundles,"Center");

   add(configurePanel,"Center");

   pButtons = new Panel();
   pButtons.setLayout(new FlowLayout());
   okButton = new Button("OK");
   cancelButton = new Button("Cancel");
   pButtons.add(okButton);
   pButtons.add(cancelButton);
   okButton.addActionListener(this);
   cancelButton.addActionListener(this);
   add(pButtons,"South");

   addWindowListener(new WindowAdapter()
   {  public void windowClosing(WindowEvent e)
      {  setVisible(false);
         dispose();
      }
   });

   pack();
}

public boolean showDialog()
{  show();
   return ok;
}

public void actionPerformed(ActionEvent evt)
{  String arg = evt.getActionCommand();
   if (arg.equals("OK"))
   {  bundles.setExpBundle(bundlesCBG.getSelectedCheckbox().
        getLabel());
      ok = true;
      setVisible(false);
      dispose();
   }
```



```
   else if (arg.equals("Cancel"))
   {  setVisible(false);
      dispose();
   }
}

private void addCBBundle(Panel p,CheckboxGroup g,RadBundle b,
  boolean defaultSelection)
{  Panel p2 = new Panel();
   p2.setLayout(new GridLayout(1,5));
   Checkbox cb = new Checkbox(b.getName(),g,defaultSelection);
   p2.add(cb);
   addLabel(p2,"" + b.getNumInBundle());
   addLabel(p2,"" + b.getMeasureActivity());
   addLabel(p2,b.getMeasureDate().toString());
   addLabel(p2,"" + (float)b.getActivity());
   p.add(p2);
}

private void addLabel(Panel p,String labelName)
{  Label newLabel = new Label(labelName);
   p.add(newLabel);
}

private void addLabel(Panel p,String labelName1,String labelName2)
{  Panel newPanel = new Panel();
   newPanel.setLayout(new BorderLayout());
   Label newLabel1 = new Label(labelName1);
   Label newLabel2 = new Label(labelName2);
   newPanel.add(newLabel1,"North");
   newPanel.add(newLabel2,"South");
   p.add(newPanel);
}

private BundleList bundles;
private Day expDate;
private CheckboxGroup bundlesCBG;
private boolean ok = false;
private Panel pBundles,pButtons;
private Button okButton,cancelButton;
}
```

FixtureDialog.java

```
/* written by Brian Chow
 * created April 14, 1998
 * last modified May 11, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
class FixtureDialog extends Dialog implements ActionListener,ItemListener,
  TextListener
{  /**
    *
    */
```



```
public FixtureDialog(Frame parent,Fixture originalFixture,RadBundle
  defaultSource)
{  super(parent,"Fixture Selection",true);
   this.originalFixture = originalFixture;
   currentFixture = (Fixture)originalFixture.clone();
   this.defaultSource = defaultSource;

   setLayout(new BorderLayout());

   pMain = new Panel();
   pMain.setLayout(new GridBagLayout());
   gbc = new GridBagConstraints();
   gbc.weightx = 50;
   gbc.weighty = 50;
   gbc.fill = GridBagConstraints.NONE;
   gbc.anchor = GridBagConstraints.NORTHWEST;
   gbc.gridwidth = 1;
   gbc.gridheight = 1;

   gbc.gridx = 1;
   gbc.gridy = 1;
   Label step1Label = new Label("Step 1:");
   step1Label.setFont(stepLabelFont);
   pMain.add(step1Label,gbc);

   pStep1 = new Panel();
   pStep1.setLayout(new FlowLayout(FlowLayout.LEFT));
   cbGStep1 = new CheckboxGroup();
   addCheckbox(pStep1,"Custom",cbGStep1,
     currentFixture instanceof Fixture);
   addCheckbox(pStep1,"Annular",cbGStep1,
     currentFixture instanceof AnnularFixture);
   gbc.gridx = 2;
   gbc.gridy = 1;
   pMain.add(pStep1,gbc);

   gbc.gridx = 1;
   gbc.gridy = 2;
   Label step2Label = new Label("Step 2:");
   step2Label.setFont(stepLabelFont);
   pMain.add(step2Label,gbc);

   gbc.gridx = 2;
   gbc.gridy = 2;
   gbc.fill = GridBagConstraints.BOTH;

   add(pMain,"Center");

   pButtons = new Panel();
   pButtons.setLayout(new FlowLayout());
   okButton = new Button("OK");
   cancelButton = new Button("Cancel");
   pButtons.add(okButton);
   pButtons.add(cancelButton);
   okButton.addActionListener(this);
   cancelButton.addActionListener(this);
   add(pButtons,"South");

   processStep1(cbGStep1.getSelectedCheckbox());

   pack();
```



```
   addWindowListener(new WindowAdapter()
   {  public void windowClosing(WindowEvent e)
      {  setVisible(false);
         dispose();
      }
   });
}

public Fixture getFixture()
{  show();
   if (ok)
   {  return currentFixture;
   }
   else
   {  return originalFixture;
   }
}

public void actionPerformed(ActionEvent evt)
{  String arg = evt.getActionCommand();
   if (arg.equals("OK"))
   {  processStep2();
      ok = true;
      setVisible(false);
      dispose();
   }
   else if (arg.equals("Cancel"))
   {  setVisible(false);
      dispose();
   }
}

public void itemStateChanged(ItemEvent evt)
{  if (evt.getStateChange() == ItemEvent.DESELECTED)
   {  return;
   }
   processStep1((Checkbox)evt.getItemSelectable());
}

public void textValueChanged(TextEvent evt)
{  okButton.setEnabled(pStep2.textValid());
}

private void addCheckbox(Panel p,String name,CheckboxGroup cbG,boolean
  state)
{  Checkbox cb = new Checkbox(name,cbG,state);
   cb.addItemListener(this);
   p.add(cb);
}

private void processStep1(Checkbox chosenFixture)
{  if (pStep2 != null)
   {  pStep2.removeTextListener(this);
      pMain.remove(pStep2);
   }
   if (chosenFixture.getLabel().equals("Custom"))
   {  if (pStep2Custom == null)
      {  if (currentFixture instanceof Fixture)
         {  pStep2Custom = new FixturePanelCustom(currentFixture,
              defaultSource);
```

```
            }
            else
            {  pStep2Custom = new FixturePanelCustom(defaultSource);
            }
         }
         pStep2 = pStep2Custom;
      }
      else if (chosenFixture.getLabel().equals("Annular"))
      {  if (pStep2Annular == null)
         {  if (currentFixture instanceof AnnularFixture)
            {  pStep2Annular = new FixturePanelAnnular(currentFixture);
            }
            else
            {  pStep2Annular = new FixturePanelAnnular(defaultSource);
            }
         }
         pStep2 = pStep2Annular;
      }
      pStep2.addTextListener(this);
      pMain.add(pStep2,gbc);
      pack();
      currentFixture = pStep2.getFixture();
      okButton.setEnabled(pStep2.textValid());
   }

   private void processStep2()
   {  currentFixture = pStep2.getFixture();
   }

   private Fixture originalFixture,currentFixture;
   private RadBundle defaultSource;
   private boolean ok = false;
   private Font stepLabelFont = new Font("Serif",Font.BOLD,14);
   private GridBagConstraints gbc;
   private Panel pStep1;
   private FixturePanel pStep2 = null;
   private FixturePanel pStep2Custom,pStep2Annular;
   private Panel pMain,pButtons;
   private CheckboxGroup cbGStep1;
   private Button okButton,cancelButton;
}
```

FixturePanel.java

```
/* written by Brian Chow
 * created April 14, 1998
 * last modified April 14, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
abstract class FixturePanel extends Panel implements TextListener
{  /**
    *
    */
   public abstract Fixture getFixture();
```



```
   public abstract boolean textValid();

   public void addTextListener(TextListener l)
   {  textListener = AWTEventMulticaster.add(textListener,l);
   }

   public void removeTextListener(TextListener l)
   {  textListener = AWTEventMulticaster.remove(textListener,l);
   }

   public void textValueChanged(TextEvent evt)
   {  if (textListener != null)
      {  textListener.textValueChanged(new TextEvent(this,evt.getID()));
      }
   }

   protected TextListener textListener;
}
```

FixturePanelAnnular.java

```
/* written by Brian Chow
 * created May 11, 1998
 * last modified May 12, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
class FixturePanelAnnular extends FixturePanel
{  /**
    *
    */
   public FixturePanelAnnular(RadBundle defaultSource)
   {  center = new Coordinate();
      source = defaultSource;
      radius = 1;
      numPencils = 6;
      changed = true;
      constructPanel();
   }

   public FixturePanelAnnular(Fixture currentFixture)
   {  oldFixture = (AnnularFixture)currentFixture;
      center = oldFixture.getCenter();
      source = oldFixture.getSource();
      radius = oldFixture.getRadius();
      numPencils = oldFixture.getNumPencils();
      constructPanel();
   }

   public Fixture getFixture()
   {  if (changed)
      {  return new AnnularFixture(center,source,radius,numPencils);
      }
      else
      {  return oldFixture;
```



```
      }
   }

   public boolean textValid()
   {  return pCenter.textValid() && dimensionsValid;
   }

   public void textValueChanged(TextEvent evt)
   {  if (evt.getSource() == tRadius || evt.getSource() == tNumPencils)
      {  try
         {  double r;
            int n;
            r = Double.valueOf(tRadius.getText().trim()).doubleValue();
            n = Integer.parseInt(tNumPencils.getText().trim());
            if (r > 0 && n > 0)
            {  radius = Units.inLength(r);
               numPencils = n;
               dimensionsValid = true;
               changed = true;
            }
            else
            {  dimensionsValid = false;
            }
         }
         catch(Exception e)
         {  dimensionsValid = false;
         }
      }
      else
      {  center = pCenter.getCoordinate();
         changed = true;
      }
      if (textListener != null)
      {  textListener.textValueChanged(new TextEvent(this,evt.getID()));
      }
   }

   private void constructPanel()
   {  setLayout(new GridBagLayout());
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.anchor = GridBagConstraints.NORTHWEST;
      gbc.insets = new Insets(10,10,10,10);

      String lengthUnits = Units.getLengthUnits().toString();
      Label lCenter = new Label("Center:");
      Label lRadius = new Label("Radius (" + lengthUnits + "):");
      Label lNumPencils = new Label("Number of Pencils:");
      pCenter = new CoordinateTextFieldPanel(center);
      pCenter.addTextListener(this);
      tRadius = new TextField("" + Units.outLength(radius),4);
      tRadius.addTextListener(this);
      tNumPencils = new TextField("" + numPencils,4);
      tNumPencils.addTextListener(this);

      add(lCenter,pCenter,0);
      add(lRadius,tRadius,1);
      add(lNumPencils,tNumPencils,2);
   }
```



```
   private void add(Component c1,Component c2,int row)
   {  gbc.gridy = row;
      gbc.weightx = 20;
      gbc.weighty = 20;
      gbc.gridx = 0;
      add(c1,gbc);
      gbc.weightx = 50;
      gbc.weighty = 50;
      gbc.gridx = 1;
      add(c2,gbc);
   }

   private AnnularFixture oldFixture;
   private Coordinate center;
   private RadBundle source;
   private double radius;
   private int numPencils;

   private boolean dimensionsValid = true;
   private boolean changed = false;
   private GridBagConstraints gbc = new GridBagConstraints();
   private CoordinateTextFieldPanel pCenter;
   private TextField tRadius,tNumPencils;
}
```

FixturePanelCustom.java

```
/* written by Brian Chow
 * created April 14, 1998
 * last modified May 12, 1998
 */

import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 *
 */
class FixturePanelCustom extends FixturePanel implements ItemListener,
  ActionListener
{  /**
    *
    */
   public FixturePanelCustom(RadBundle defaultSource)
   {  this.defaultSource = defaultSource;
      currentFixture = new Fixture();
      constructPanel();
   }

   public FixturePanelCustom(Fixture currentFixture,RadBundle
     defaultSource)
   {  this.currentFixture = (Fixture)currentFixture;
      this.defaultSource = defaultSource;
      Enumeration pencilsEnum = currentFixture.elements();
      while (pencilsEnum.hasMoreElements())
      {  pointsList.add(pencilsEnum.nextElement().toString());
      }
      constructPanel();
   }
```

```
   public Fixture getFixture()
   {  return new Fixture(currentFixture);
   }

   public boolean textValid()
   {  return pointsList.getItemCount() > 0;
   }

   public void actionPerformed(ActionEvent evt)
   {  String arg = evt.getActionCommand();
      if (arg.equals("Add"))
      {  addPoint();
      }
      else if (arg.equals("Change"))
      {  changePoint();
      }
      else if (arg.equals("Delete"))
      {  deletePoint();
      }
   }

   public void textValueChanged(TextEvent evt)
   {  boolean pointOK = pointPanel.textValid();
      addButton.setEnabled(pointOK);
      changeButton.setEnabled(pointOK);
   }

   public void itemStateChanged(ItemEvent evt)
   {  if (evt.getStateChange() == ItemEvent.DESELECTED)
      {  return;
      }
      Coordinate changedCoordinate = getCenter(pointsList.
        getSelectedIndex());
      pointPanel.setCoordinate(changedCoordinate);
   }

   private Coordinate getCenter(int index)
   {  return currentFixture.getPencil(index).getCenter();
   }

   private void constructPanel()
   {  setLayout(new GridBagLayout());
      gbc.weightx = 100;
      gbc.weighty = 100;
      gbc.insets = new Insets(10,10,10,10);
      gbc.fill = GridBagConstraints.NONE;

      pointsList.addItemListener(this);
      pointsList.select(0);
      gbc.anchor = GridBagConstraints.NORTHWEST;
      add(pointsList,1,2,1,1);

      if (currentFixture.size() > 0)
      {  pointPanel.setCoordinate(getCenter(0));
      }
      else
      {  pointPanel.setCoordinate(new Coordinate());
      }
      pointPanel.addTextListener(this);
      add(pointPanel,3,1,2,1);
```



```
      addButton = new Button("Add");
      addButton.addActionListener(this);
      deleteButton = new Button("Delete");
      deleteButton.setEnabled(pointsList.getItemCount() > 1);
      deleteButton.addActionListener(this);
      changeButton = new Button("Change");
      changeButton.addActionListener(this);

      gbc.anchor = GridBagConstraints.CENTER;
      add(addButton,1,1,2,2);
      add(deleteButton,1,1,3,2);
      add(changeButton,1,1,4,2);
   }

   private void add(Component c,int width,int height,int x,int y)
   {  gbc.gridwidth = width;
      gbc.gridheight = height;
      gbc.gridx = x;
      gbc.gridy = y;
      add(c,gbc);
   }

   private void addPoint()
   {  Coordinate changedPoint = pointPanel.getCoordinate();
      int indexToAdd = findPoint(changedPoint);
      if (indexToAdd < 0)
      {  currentFixture.add(new Pencil(changedPoint,defaultSource));
         pointsList.add("" + changedPoint + " " + defaultSource);
         pointsList.select(pointsList.getItemCount() - 1);
      }
      else
      {  pointsList.select(indexToAdd);
      }
      pointPanel.setCoordinate(changedPoint);
      updateButtons();
   }

   private void deletePoint()
   {  int indexToDelete = pointsList.getSelectedIndex();
      currentFixture.remove(indexToDelete);
      pointsList.remove(indexToDelete);
      int numItems = pointsList.getItemCount();
      if (indexToDelete >= numItems)
      {  pointsList.select(numItems - 1);
      }
      else
      {  pointsList.select(indexToDelete);
      }
      pointPanel.setCoordinate(getCenter(pointsList.getSelectedIndex()));
      updateButtons();
   }

   private void changePoint()
   {  int indexToChange = pointsList.getSelectedIndex();
      Coordinate changedPoint = pointPanel.getCoordinate();
      int indexFound = findPoint(changedPoint);
      if (indexToChange == indexFound)
      {  pointsList.select(indexToChange);
      }
      else if (indexFound < 0)
```



```
      {  currentFixture.setPencilCenter(indexToChange,changedPoint);
         pointsList.replaceItem("" + changedPoint + " " + defaultSource,
           indexToChange);
         pointsList.select(indexToChange);
      }
      else
      {  pointsList.select(indexFound);
         currentFixture.remove(indexToChange);
         pointsList.remove(indexToChange);
      }
      pointPanel.setCoordinate(changedPoint);
      updateButtons();
   }

   private int findPoint(Coordinate p)
   {  int i = 0;
      boolean found = false;
      while (i < currentFixture.size() && !found)
      {  found = getCenter(i).equals(p);
         i++;
      }
      if (found)
      {  return i - 1;
      }
      else
      {  return -1;
      }
   }

   private void updateButtons()
   {  boolean pointOK = pointPanel.textValid();
      boolean pointSelected = pointsList.getSelectedItem() != null;
      deleteButton.setEnabled(pointsList.getItemCount() > 1 &&
        pointSelected);
      addButton.setEnabled(pointOK);
      changeButton.setEnabled(pointOK && pointSelected);
   }

   private Fixture currentFixture;
   private RadBundle defaultSource;
   private GridBagConstraints gbc = new GridBagConstraints();
   private List pointsList = new List(15,false);
   private CoordinateTextFieldPanel pointPanel
     = new CoordinateTextFieldPanel();
   private Button addButton,deleteButton,changeButton;
}
```

CoordinateTextFieldPanel.java

```
/* written by Brian Chow
 * created March 23, 1998
 * last modified April 5, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
```



```
public class CoordinateTextFieldPanel extends Panel implements
  TextListener
{  /**
    *
    */
   public CoordinateTextFieldPanel()
   {  x = 0;
      y = 0;
      z = 0;
      constructPanel();
   }

   public CoordinateTextFieldPanel(Coordinate point)
   {  x = Units.outLength(point.x);
      y = Units.outLength(point.y);
      z = Units.outLength(point.z);
      constructPanel();
   }

   public boolean textValid()
   {  return pX.textValid() && pY.textValid() && pZ.textValid();
   }

   public Coordinate getCoordinate()
   {  double x,y,z;
      x = Units.inLength(this.x);
      y = Units.inLength(this.y);
      z = Units.inLength(this.z);
      return new Coordinate(x,y,z);
   }

   public void setCoordinate(Coordinate newPoint)
   {  x = Units.outLength(newPoint.x);
      y = Units.outLength(newPoint.y);
      z = Units.outLength(newPoint.z);
      pX.setValue(x);
      pY.setValue(y);
      pZ.setValue(z);
   }

   public void addTextListener(TextListener l)
   {  textListener = AWTEventMulticaster.add(textListener,l);
   }

   public void removeTextListener(TextListener l)
   {  textListener = AWTEventMulticaster.remove(textListener,l);
   }

   public void textValueChanged(TextEvent evt)
   {  if (textValid())
      {  x = pX.getValue();
         y = pY.getValue();
         z = pZ.getValue();
      }
      if (textListener != null)
      {  textListener.textValueChanged(evt);
      }
   }

   private void constructPanel()
   {  String currentUnits = Units.getLengthUnits().toString();
```

```
      setLayout(new GridLayout(3,1));
      pX = new DoubleTextFieldPanel(this,"x (" + currentUnits + ")","" +
        x,5);
      pY = new DoubleTextFieldPanel(this,"y (" + currentUnits + ")","" +
        y,5);
      pZ = new DoubleTextFieldPanel(this,"z (" + currentUnits + ")","" +
        z,5);
      add(pX);
      add(pY);
      add(pZ);
   }

   private TextListener textListener = null;
   private double x,y,z;
   private DoubleTextFieldPanel pX,pY,pZ;
}
```

DoubleTextFieldPanel.java

```
/* written by Brian Chow
 * created March 23, 1998
 * last modified March 29, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
public class DoubleTextFieldPanel extends Panel
{  /**
    *
    */
   public DoubleTextFieldPanel(TextListener parent,String labelText,
     String textFieldValue,int columnWidth)
   {  setLayout(new FlowLayout(FlowLayout.RIGHT));
      Label lField = new Label(labelText);
      add(lField);
      tValue = new TextField(textFieldValue,columnWidth);
      tValue.addTextListener(parent);
      add(tValue);
   }

   public boolean textValid()
   {  Double tempVal;
      try
      {  tempVal = new Double(tValue.getText());
      }
      catch (NumberFormatException e)
      {  tempVal = null;
      }
      return (tempVal != null);
   }

   public double getValue()
   {  if (textValid())
      {  value = (new Double(tValue.getText())).doubleValue();
      }
      return value;
```



```
   }

   public void setValue(double newValue)
   {  tValue.setText("" + newValue);
   }

   private TextField tValue;
   private double value;
}
```

TargetDialog.java

```
/* written by Brian Chow
 * created March 21, 1998
 * last modified April 7, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
class TargetDialog extends Dialog implements ActionListener,ItemListener,
  TextListener
{  /**
    *
    */
   public TargetDialog(Frame parent,Target originalTarget)
   {  super(parent,"Target Selection",true);
      this.originalTarget = originalTarget;
      currentTarget = (Target)originalTarget.clone();

      setLayout(new BorderLayout());

      pMain = new Panel();
      pMain.setLayout(new GridBagLayout());
      gbc = new GridBagConstraints();
      gbc.weightx = 50;
      gbc.weighty = 50;
      gbc.fill = GridBagConstraints.NONE;
      gbc.anchor = GridBagConstraints.NORTHWEST;
      gbc.gridwidth = 1;
      gbc.gridheight = 1;

      gbc.gridx = 1;
      gbc.gridy = 1;
      Label step1Label = new Label("Step 1:");
      step1Label.setFont(stepLabelFont);
      pMain.add(step1Label,gbc);

      pStep1 = new Panel();
      pStep1.setLayout(new FlowLayout(FlowLayout.LEFT));
      cbGStep1 = new CheckboxGroup();
      addCheckbox(pStep1,"Points",cbGStep1,
        currentTarget instanceof TargetPoints);
      addCheckbox(pStep1,"Line",cbGStep1,
        currentTarget instanceof TargetLine);
      addCheckbox(pStep1,"Rectangular Parallelepiped",cbGStep1,
        currentTarget instanceof TargetRect);
```



```
      addCheckbox(pStep1,"Cylinder",cbGStep1,
        currentTarget instanceof TargetCyl);
      gbc.gridx = 2;
      gbc.gridy = 1;
      pMain.add(pStep1,gbc);

      gbc.gridx = 1;
      gbc.gridy = 2;
      Label step2Label = new Label("Step 2:");
      step2Label.setFont(stepLabelFont);
      pMain.add(step2Label,gbc);

      gbc.gridx = 2;
      gbc.gridy = 2;
      gbc.fill = GridBagConstraints.BOTH;

      add(pMain,"Center");

      pButtons = new Panel();
      pButtons.setLayout(new FlowLayout());
      okButton = new Button("OK");
      cancelButton = new Button("Cancel");
      pButtons.add(okButton);
      pButtons.add(cancelButton);
      okButton.addActionListener(this);
      cancelButton.addActionListener(this);
      add(pButtons,"South");

      processStep1(cbGStep1.getSelectedCheckbox());

      pack();

      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  setVisible(false);
            dispose();
         }
      });
   }

   public Target getTarget()
   {  show();
      if (ok)
      {  return currentTarget;
      }
      else
      {  return originalTarget;
      }
   }

   public void actionPerformed(ActionEvent evt)
   {  String arg = evt.getActionCommand();
      if (arg.equals("OK"))
      {  processStep2();
         ok = true;
         setVisible(false);
         dispose();
      }
      else if (arg.equals("Cancel"))
      {  setVisible(false);
         dispose();
```



```
      }
   }

   public void itemStateChanged(ItemEvent evt)
   {  if (evt.getStateChange() == ItemEvent.DESELECTED)
      {  return;
      }
      processStep1((Checkbox)evt.getItemSelectable());
   }

   public void textValueChanged(TextEvent evt)
   {  okButton.setEnabled(pStep2.textValid());
   }

   private void addCheckbox(Panel p,String name,CheckboxGroup cbG,boolean
     state)
   {  Checkbox cb = new Checkbox(name,cbG,state);
      cb.addItemListener(this);
      p.add(cb);
   }

   private void processStep1(Checkbox chosenTarget)
   {  if (pStep2 != null)
      {  pStep2.removeTextListener(this);
         pMain.remove(pStep2);
      }
      if (chosenTarget.getLabel().equals("Points"))
      {  if (pStep2Points == null)
         {  if (currentTarget instanceof TargetPoints)
            {  pStep2Points = new TargetPanelPoints(currentTarget);
            }
            else
            {  pStep2Points = new TargetPanelPoints();
            }
         }
         pStep2 = pStep2Points;
      }
      else if (chosenTarget.getLabel().equals("Line"))
      {  if (pStep2Line == null)
         {  if (currentTarget instanceof TargetLine)
            {  pStep2Line = new TargetPanelLine(currentTarget);
            }
            else
            {  pStep2Line = new TargetPanelLine();
            }
         }
         pStep2 = pStep2Line;
      }
      else if (chosenTarget.getLabel().
        equals("Rectangular Parallelepiped"))
      {  if (pStep2Rect == null)
         {  if (currentTarget instanceof TargetRect)
            {  pStep2Rect = new TargetPanelRect(currentTarget);
            }
            else
            {  pStep2Rect = new TargetPanelRect();
            }
         }
         pStep2 = pStep2Rect;
      }
      else if (chosenTarget.getLabel().
```

```
        equals("Cylinder"))
      {  if (pStep2Cyl == null)
         {  if (currentTarget instanceof TargetCyl)
            {  pStep2Cyl = new TargetPanelCyl(currentTarget);
            }
            else
            {  pStep2Cyl = new TargetPanelCyl();
            }
         }
         pStep2 = pStep2Cyl;
      }
      pStep2.addTextListener(this);
      pMain.add(pStep2,gbc);
      pack();
      currentTarget = pStep2.getTarget();
      okButton.setEnabled(pStep2.textValid());
   }

   private void processStep2()
   {  currentTarget = pStep2.getTarget();
   }

   private Target originalTarget,currentTarget;
   private boolean ok = false;
   private Font stepLabelFont = new Font("Serif",Font.BOLD,14);
   private GridBagConstraints gbc;
   private Panel pStep1;
   private TargetPanel pStep2 = null;
   private TargetPanel pStep2Points,pStep2Line,pStep2Rect,pStep2Cyl;
   private Panel pMain,pButtons;
   private CheckboxGroup cbGStep1;
   private Button okButton,cancelButton;
}
```

TargetPanel.java

```
/* written by Brian Chow
 * created March 21, 1998
 * last modified March 29, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
abstract class TargetPanel extends Panel implements TextListener
{  /**
    *
    */
   public abstract Target getTarget();
   public abstract boolean textValid();

   public void addTextListener(TextListener l)
   {  textListener = AWTEventMulticaster.add(textListener,l);
   }

   public void removeTextListener(TextListener l)
   {  textListener = AWTEventMulticaster.remove(textListener,l);
```



```
   }

   public void textValueChanged(TextEvent evt)
   {  if (textListener != null)
      {  textListener.textValueChanged(new TextEvent(this,evt.getID()));
      }
   }

   protected TextListener textListener;
}
```

TargetPanelPoints.java

```
/* written by Brian Chow
 * created March 21, 1998
 * last modified April 25, 1998
 */

import java.awt.*;
import java.awt.event.*;
import java.util.*;

/**
 *
 */
class TargetPanelPoints extends TargetPanel implements ItemListener,
  ActionListener
{  /**
    *
    */
   public TargetPanelPoints()
   {  currentPointsTarget = new TargetPoints();
      points = new Vector(5,5);
      points.addElement(currentPointsTarget.getPoints()[0]);
      pointsList.add((Units.outLength(currentPointsTarget.
        getPoints()[0])).toString());
      constructPanel();
   }

   public TargetPanelPoints(Target currentPointsTarget)
   {  this.currentPointsTarget = (TargetPoints)currentPointsTarget;
      Coordinate[] tempPoints = this.currentPointsTarget.getPoints();
      points = new Vector(tempPoints.length,5);
      for (int i = 0;i < tempPoints.length;i++)
      {  points.addElement(tempPoints[i]);
         pointsList.add((Units.outLength(tempPoints[i])).toString());
      }
      constructPanel();
   }

   public Target getTarget()
   {  return new TargetPoints(points);
   }

   public boolean textValid()
   {  return !points.isEmpty();
   }

   public void actionPerformed(ActionEvent evt)
   {  String arg = evt.getActionCommand();
```



```
      if (arg.equals("Add"))
      {  addPoint();
      }
      else if (arg.equals("Change"))
      {  changePoint();
      }
      else if (arg.equals("Delete"))
      {  deletePoint();
      }
   }

   public void textValueChanged(TextEvent evt)
   {  boolean pointOK = pointPanel.textValid();
      addButton.setEnabled(pointOK);
      changeButton.setEnabled(pointOK);
   }

   public void itemStateChanged(ItemEvent evt)
   {  if (evt.getStateChange() == ItemEvent.DESELECTED)
      {  return;
      }
      Coordinate changedCoordinate = (Coordinate)points.
        elementAt(pointsList.getSelectedIndex());
      pointPanel.setCoordinate(changedCoordinate);
   }

   private void constructPanel()
   {  setLayout(new GridBagLayout());
      gbc.weightx = 100;
      gbc.weighty = 100;
      gbc.insets = new Insets(10,10,10,10);
      gbc.fill = GridBagConstraints.NONE;

      pointsList.addItemListener(this);
      pointsList.select(0);
      gbc.anchor = GridBagConstraints.NORTHWEST;
      add(pointsList,1,2,1,1);

      pointPanel.setCoordinate((Coordinate)points.elementAt(0));
      pointPanel.addTextListener(this);
      add(pointPanel,3,1,2,1);

      addButton = new Button("Add");
      addButton.addActionListener(this);
      deleteButton = new Button("Delete");
      deleteButton.setEnabled(pointsList.getItemCount() > 1);
      deleteButton.addActionListener(this);
      changeButton = new Button("Change");
      changeButton.addActionListener(this);

      gbc.anchor = GridBagConstraints.CENTER;
      add(addButton,1,1,2,2);
      add(deleteButton,1,1,3,2);
      add(changeButton,1,1,4,2);
   }

   private void add(Component c,int width,int height,int x,int y)
   {  gbc.gridwidth = width;
      gbc.gridheight = height;
      gbc.gridx = x;
      gbc.gridy = y;
```



```
      add(c,gbc);
   }

   private void addPoint()
   {  Coordinate changedPoint = pointPanel.getCoordinate();
      int indexToAdd = findPoint(changedPoint);
      if (indexToAdd < 0)
      {  points.addElement(changedPoint);
         pointsList.add((Units.outLength(changedPoint)).toString());
         pointsList.select(pointsList.getItemCount() - 1);
      }
      else
      {  pointsList.select(indexToAdd);
      }
      pointPanel.setCoordinate(changedPoint);
      updateButtons();
   }

   private void deletePoint()
   {  // Stop if only one point is left.
      if (points.size() <= 1)
      {  return;
      }
      int indexToDelete = pointsList.getSelectedIndex();
      points.removeElementAt(indexToDelete);
      pointsList.remove(indexToDelete);
      int numItems = pointsList.getItemCount();
      if (indexToDelete >= numItems)
      {  pointsList.select(numItems - 1);
      }
      else
      {  pointsList.select(indexToDelete);
      }
      pointPanel.setCoordinate((Coordinate)points.elementAt(pointsList.
        getSelectedIndex()));
      updateButtons();
   }

   private void changePoint()
   {  int indexToChange = pointsList.getSelectedIndex();
      Coordinate changedPoint = pointPanel.getCoordinate();
      int indexFound = findPoint(changedPoint);
      if (indexToChange == indexFound)
      {  pointsList.select(indexToChange);
      }
      else if (indexFound < 0)
      {  points.setElementAt(changedPoint,indexToChange);
         pointsList.replaceItem((Units.outLength(changedPoint)).
           toString(),indexToChange);
         pointsList.select(indexToChange);
      }
      else
      {  pointsList.select(indexFound);
         points.removeElementAt(indexToChange);
         pointsList.remove(indexToChange);
      }
      pointPanel.setCoordinate(changedPoint);
      updateButtons();
   }

   private int findPoint(Coordinate p)
```

```
   {  int i = 0;
      boolean found = false;
      while (i < points.size() && !found)
      {  found = ((Coordinate)points.elementAt(i)).equals(p);
         i++;
      }
      if (found)
      {  return i - 1;
      }
      else
      {  return -1;
      }
   }

   private void updateButtons()
   {  boolean pointOK = pointPanel.textValid();
      boolean pointSelected = pointsList.getSelectedItem() != null;
      deleteButton.setEnabled(pointsList.getItemCount() > 1 &&
        pointSelected);
      addButton.setEnabled(pointOK);
      changeButton.setEnabled(pointOK && pointSelected);
   }

   private TargetPoints currentPointsTarget;
   private Vector points;
   private GridBagConstraints gbc = new GridBagConstraints();
   private List pointsList = new List(15,false);
   private CoordinateTextFieldPanel pointPanel
     = new CoordinateTextFieldPanel();
   private Button addButton,deleteButton,changeButton;
}
```

TargetPanelLine.java

```
/* written by Brian Chow
 * created March 21, 1998
 * last modified April 24, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
class TargetPanelLine extends TargetPanel
{  /**
    *
    */
   public TargetPanelLine()
   {  currentLineTarget = new TargetLine();
      point1 = currentLineTarget.getPoint1();
      point2 = currentLineTarget.getPoint2();
      numPoints = currentLineTarget.getNumPoints();
      constructPanel();
   }

   public TargetPanelLine(Target currentLineTarget)
   {  this.currentLineTarget = (TargetLine)currentLineTarget;
      point1 = this.currentLineTarget.getPoint1();
```



```
      point2 = this.currentLineTarget.getPoint2();
      numPoints = this.currentLineTarget.getNumPoints();
      constructPanel();
   }

   public Target getTarget()
   {  currentLineTarget.setLine(point1,point2,numPoints);
      return currentLineTarget;
   }

   public boolean textValid()
   {  return pPoint1.textValid() && pPoint2.textValid() &&
        numPointsValid;
   }

   public void textValueChanged(TextEvent evt)
   {  if (evt.getSource() == tNumPoints)
      {  try
         {  int n = Integer.parseInt(tNumPoints.getText().trim());
            // Make sure the number of points is greater than one.
            if (n > 1)
            {  numPoints = n;
               numPointsValid = true;
            }
            else
            {  numPointsValid = false;
            }
         }
         catch(Exception e)
         {  numPointsValid = false;
         }
      }
      else
      {  point1 = pPoint1.getCoordinate();
         point2 = pPoint2.getCoordinate();
      }
      if (textListener != null)
      {  textListener.textValueChanged(new TextEvent(this,evt.getID()));
      }
   }

   private void constructPanel()
   {  setLayout(new GridBagLayout());
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.anchor = GridBagConstraints.NORTHWEST;
      gbc.insets = new Insets(10,10,10,10);

      Label lPoint1 = new Label("Endpoint 1:");
      Label lPoint2 = new Label("Endpoint 2:");
      Label lNumPoints = new Label("Number of points:");
      pPoint1 = new CoordinateTextFieldPanel(point1);
      pPoint1.addTextListener(this);
      pPoint2 = new CoordinateTextFieldPanel(point2);
      pPoint2.addTextListener(this);
      tNumPoints = new TextField("" + numPoints,4);
      tNumPoints.addTextListener(this);
      add(lPoint1,pPoint1,0);
      add(lPoint2,pPoint2,1);
      add(lNumPoints,tNumPoints,2);
```



```
   }

   private void add(Component c1,Component c2,int row)
   {  gbc.gridy = row;
      gbc.weightx = 20;
      gbc.weighty = 20;
      gbc.gridx = 0;
      add(c1,gbc);
      gbc.weightx = 50;
      gbc.weighty = 50;
      gbc.gridx = 1;
      add(c2,gbc);
   }

   private TargetLine currentLineTarget;
   private Coordinate point1,point2;
   private int numPoints;
   private boolean numPointsValid = true;
   private GridBagConstraints gbc = new GridBagConstraints();
   private CoordinateTextFieldPanel pPoint1,pPoint2;
   private TextField tNumPoints;
}
```

TargetPanelCyl.java

```
/* written by Brian Chow
 * created April 7, 1998
 * last modified April 7, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
class TargetPanelCyl extends TargetPanel
{  /**
    *
    */
   public TargetPanelCyl()
   {  currentCylTarget = new TargetCyl();
      center = currentCylTarget.getCenter();
      height = currentCylTarget.getHeight();
      radius = currentCylTarget.getRadius();
      constructPanel();
   }

   public TargetPanelCyl(Target currentCylTarget)
   {  this.currentCylTarget = (TargetCyl)currentCylTarget;
      center = this.currentCylTarget.getCenter();
      height = this.currentCylTarget.getHeight();
      radius = this.currentCylTarget.getRadius();
      constructPanel();
   }

   public Target getTarget()
   {  currentCylTarget.setCyl(center,height,radius);
      return currentCylTarget;
   }
```



```
   public boolean textValid()
   {  return pCenter.textValid() && dimensionsValid;
   }

   public void textValueChanged(TextEvent evt)
   {  if (evt.getSource() == tHeight || evt.getSource() == tRadius)
      {  try
         {  double h,r;
            h = Double.valueOf(tHeight.getText().trim()).doubleValue();
            r = Double.valueOf(tRadius.getText().trim()).doubleValue();
            if (h > 0 && r > 0)
            {  height = Units.inLength(h);
               radius = Units.inLength(r);
               dimensionsValid = true;
            }
            else
            {  dimensionsValid = false;
            }
         }
         catch(Exception e)
         {  dimensionsValid = false;
         }
      }
      else
      {  center = pCenter.getCoordinate();
      }
      if (textListener != null)
      {  textListener.textValueChanged(new TextEvent(this,evt.getID()));
      }
   }

   private void constructPanel()
   {  setLayout(new GridBagLayout());
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.anchor = GridBagConstraints.NORTHWEST;
      gbc.insets = new Insets(10,10,10,10);

      String lengthUnits = Units.getLengthUnits().toString();
      Label lCenter = new Label("Center:");
      Label lHeight = new Label("Height (" + lengthUnits + "):");
      Label lRadius = new Label("Radius (" + lengthUnits + "):");
      pCenter = new CoordinateTextFieldPanel(center);
      pCenter.addTextListener(this);
      tHeight = new TextField("" + Units.outLength(height),4);
      tHeight.addTextListener(this);
      tRadius = new TextField("" + Units.outLength(radius),4);
      tRadius.addTextListener(this);
      add(lCenter,pCenter,0);
      add(lHeight,tHeight,1);
      add(lRadius,tRadius,2);
   }

   private void add(Component c1,Component c2,int row)
   {  gbc.gridy = row;
      gbc.weightx = 20;
      gbc.weighty = 20;
      gbc.gridx = 0;
      add(c1,gbc);
```

```
      gbc.weightx = 50;
      gbc.weighty = 50;
      gbc.gridx = 1;
      add(c2,gbc);
   }

   private TargetCyl currentCylTarget;
   private Coordinate center;
   private double height,radius;
   private boolean dimensionsValid = true;
   private GridBagConstraints gbc = new GridBagConstraints();
   private CoordinateTextFieldPanel pCenter;
   private TextField tHeight,tRadius;
}
```

TargetPanelRect.java

```
/* written by Brian Chow
 * created April 7, 1998
 * last modified April 7, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 *
 */
class TargetPanelRect extends TargetPanel
{  /**
    *
    */
   public TargetPanelRect()
   {  currentRectTarget = new TargetRect();
      center = currentRectTarget.getCenter();
      height = currentRectTarget.getHeight();
      width = currentRectTarget.getWidth();
      length = currentRectTarget.getLength();
      constructPanel();
   }

   public TargetPanelRect(Target currentRectTarget)
   {  this.currentRectTarget = (TargetRect)currentRectTarget;
      center = this.currentRectTarget.getCenter();
      height = this.currentRectTarget.getHeight();
      width = this.currentRectTarget.getWidth();
      length = this.currentRectTarget.getLength();
      constructPanel();
   }

   public Target getTarget()
   {  currentRectTarget.setRect(center,height,width,length);
      return currentRectTarget;
   }

   public boolean textValid()
   {  return pCenter.textValid() && dimensionsValid;
   }

   public void textValueChanged(TextEvent evt)
```



```
   {  if (evt.getSource() == tHeight || evt.getSource() == tWidth ||
         evt.getSource() == tLength)
      {  try
         {  double h,w,l;
            h = Double.valueOf(tHeight.getText().trim()).doubleValue();
            w = Double.valueOf(tWidth.getText().trim()).doubleValue();
            l = Double.valueOf(tLength.getText().trim()).doubleValue();
            if (h > 0 && w > 0 && l > 0)
            {  height = Units.inLength(h);
               width = Units.inLength(w);
               length = Units.inLength(l);
               dimensionsValid = true;
            }
            else
            {  dimensionsValid = false;
            }
         }
         catch(Exception e)
         {  dimensionsValid = false;
         }
      }
      else
      {  center = pCenter.getCoordinate();
      }
      if (textListener != null)
      {  textListener.textValueChanged(new TextEvent(this,evt.getID()));
      }
   }

   private void constructPanel()
   {  setLayout(new GridBagLayout());
      gbc.gridwidth = 1;
      gbc.gridheight = 1;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.anchor = GridBagConstraints.NORTHWEST;
      gbc.insets = new Insets(10,10,10,10);

      String lengthUnits = Units.getLengthUnits().toString();
      Label lCenter = new Label("Center:");
      Label lHeight = new Label("Height (" + lengthUnits + "):");
      Label lWidth = new Label("Width (" + lengthUnits + "):");
      Label lLength = new Label("Length (" + lengthUnits + "):");
      pCenter = new CoordinateTextFieldPanel(center);
      pCenter.addTextListener(this);
      tHeight = new TextField("" + Units.outLength(height),4);
      tHeight.addTextListener(this);
      tWidth = new TextField("" + Units.outLength(width),4);
      tWidth.addTextListener(this);
      tLength = new TextField("" + Units.outLength(length),4);
      tLength.addTextListener(this);
      add(lCenter,pCenter,0);
      add(lHeight,tHeight,1);
      add(lWidth,tWidth,2);
      add(lLength,tLength,3);
   }

   private void add(Component c1,Component c2,int row)
   {  gbc.gridy = row;
      gbc.weightx = 20;
      gbc.weighty = 20;
      gbc.gridx = 0;
```



```
      add(c1,gbc);
      gbc.weightx = 50;
      gbc.weighty = 50;
      gbc.gridx = 1;
      add(c2,gbc);
   }

   private TargetRect currentRectTarget;
   private Coordinate center;
   private double height,width,length;
   private boolean dimensionsValid = true;
   private GridBagConstraints gbc = new GridBagConstraints();
   private CoordinateTextFieldPanel pCenter;
   private TextField tHeight,tWidth,tLength;
}
```

AboutDialog.java

```
/* written by Brian Chow
 * created February 28, 1998
 * last modified March 27, 1998
 */

import java.awt.*;
import java.awt.event.*;

class AboutDialog extends Dialog
{
   public AboutDialog(Frame parent)
   {  super(parent,"About Gamma Cell",true);
      setLayout(new BorderLayout());

      Panel p = new Panel();
      String[] text = {"Gamma Cell Irradiation Dose Applet"," ",
        "  by Brian Y. Chow","  advisor Jean B. Hunter",
        "    Department of Agricultural and Biological Engineering",
        "  special thanks Samuel J. DiPasquale",
        "    Ward Center for Nuclear Sciences",
        " ","  Cornell University, Ithaca NY"};
      p.setLayout(new GridLayout(text.length,1));
      Label l;
      for (int i = 0;i < text.length;i++)
      {  l = new Label(text[i]);
         p.add(l);
      }
      add(p,"Center");

      Panel p2 = new Panel();
      p2.setLayout(new FlowLayout());
      Button okButton = new Button("OK");
      okButton.addActionListener(new ActionListener()
      {  public void actionPerformed(ActionEvent e)
         {  setVisible(false);
         }
      });
      p2.add(okButton);
      add(p2,"South");

      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
```



```
         {  setVisible(false);
         }
      });
      pack();
   }

}
```

CalculationProgress.java

```
/* written by Brian Chow
 * created April 24, 1998
 * last modified May 11, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 * Calculation progress window.
 */
public class CalculationProgress extends Frame
{  /**
    * Constructs new window with a text for the output.
    */
   public CalculationProgress()
   {  // Set up the window with title and output area.
      setTitle("Calculating...");
      setLayout(new GridLayout(2,1));
      Label waitLabel = new Label("Please Wait");
      add(waitLabel);
      outputArea = new Label(spaces);
      outputArea.setFont(outputFont);
      add(outputArea);
      setSize(350,100);

      // Center the window in the screen.
      Dimension screenSize = getToolkit().getScreenSize();
      Dimension windowSize = getSize();
      setLocation((screenSize.width - windowSize.width) / 2,
        (screenSize.height - windowSize.height) / 2);

      // Allow the window to be closed.
      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  setVisible(false);
         }
      });
   }

   /**
    * Change the output string value.
    */
   public void print(String value)
   {  outputArea.setText(value);
   }

   /**
    * Clear the text window.
    */
```

```
   public void clear()
   {  outputArea.setText(spaces);
   }

   /**
    * String of spaces for initialization of the output area.
    */
   private static final String spaces =
     "                                        ";
   /**
    * Text for the output.
    */
   private Label outputArea;
   /**
    * Font used for the output area.
    */
   private Font outputFont = new Font("Monospaced",Font.PLAIN,12);
}
```

CalculationOutput.java

```
/* written by Brian Chow
 * created March 19, 1998
 * last modified May 8, 1998
 */

import java.awt.*;
import java.awt.event.*;

/**
 * General purpose output window with a text area.
 */
public class CalculationOutput extends Frame
{  /**
    * Constructs new window with a text area for the output.
    */
   public CalculationOutput(int outputRows,int outputColumns)
   {  setLayout(new FlowLayout());
      setResizable(false);
      outputArea = new TextArea("",outputRows,outputColumns,
        TextArea.SCROLLBARS_VERTICAL_ONLY);
      outputArea.setEditable(false);
      outputArea.setFont(outputFont);
      add(outputArea);

      pack();
      Dimension screenSize = getToolkit().getScreenSize();
      Dimension windowSize = getSize();
      setLocation((screenSize.width - windowSize.width) / 2,
        (screenSize.height - windowSize.height) / 2);

      addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
         {  setVisible(false);
         }
      });
   }

   /**
    * Appends the string value to the end of the output text area.
```



```
    */
   public void print(String value)
   {  outputString += value;
      if (visible)
      {  outputArea.append(value);
      }
   }

   /**
    * Adds pad spaces to begining of string until string is the specified
    * width unless the string length exceeds or is equal to the specified
    * width.
    */
   public void print(String value,int width)
   {  while (width > value.length())
      {  value = " " + value;
      }
      print(value);
   }

   /**
    * Append a blank line to the end of the output text area.
    */
   public void println()
   {  println("");
   }

   /**
    * Appends the string value to the end of the output text area
    * terminated with a new line character.
    */
   public void println(String value)
   {  print(value + separator);
   }

   /**
    * Appends the string value to the end of the output text area with
    * padded spaces added to the begining of the string to satisfy the
    * specified width and terminated with a new line character.
    */
   public void println(String value,int width)
   {  print(value,width);
      println();
   }

   /**
    * Clear the text window.
    */
   public void clear()
   {  outputString = "";
      outputArea.setText("");
   }

   /**
    * Show the window with first line visible.
    */
   public void setVisible(boolean visible)
   {  outputArea.setText(outputString);
      outputArea.setCaretPosition(0);
      super.setVisible(visible);
   }
```



```
   /**
    * Platform specific line separator.
    */
   private static final String separator =
     System.getProperty("line.separator");

   /**
    * Text area for the output starting with no text and no scroll bars.
    */
   private TextArea outputArea;
   /**
    * Keep track whether or not the window is visible;
    */
   private boolean visible;
   /**
    * String to be outputed to the output area.
    */
   private String outputString = "";
   /**
    * Font used for the text area.
    */
   private Font outputFont = new Font("Monospaced",Font.PLAIN,12);
}
```

Units.java

```
/* written by Brian Chow
 * created March 20, 1998
 * last modified April 5, 1998
 */

/**
 *
 */
class Units
{  /**
    *
    */
   public Units()
   {  lengthUnits = new UnitLength("Inches",100);
      doseUnits = new UnitDose("Rads");
   }

   public Units(UnitLength lengthUnits,UnitDose doseUnits)
   {  this.lengthUnits = lengthUnits;
      this.doseUnits = doseUnits;
   }

   public static UnitLength getLengthUnits()
   {  return lengthUnits;
   }

   public static UnitDose getDoseUnits()
   {  return doseUnits;
   }

   public static void setLengthUnits(UnitLength u)
   {  lengthUnits = u;
   }
```



```
   public static void setDoseUnits(UnitDose u)
   {  doseUnits = u;
   }

   public static double outLength(double value)
   {  return lengthUnits.outUnits(value);
   }

   public static Coordinate outLength(Coordinate point)
   {  double x,y,z;
      x = lengthUnits.outUnits(point.x);
      y = lengthUnits.outUnits(point.y);
      z = lengthUnits.outUnits(point.z);
      return new Coordinate(x,y,z);
   }

   public static double outDose(double value)
   {  return doseUnits.outUnits(value);
   }

   public static double inLength(double value)
   {  return lengthUnits.inUnits(value);
   }

/*
   public static Coordinate inLength(Coordinate point)
   {  double x,y,z;
      x = lengthUnits.inUnits(point.x);
      y = lengthUnits.inUnits(point.y);
      z = lengthUnits.inUnits(point.z);
      return new Coordinate(x,y,z);
   }
*/

   public String toString()
   {  return "Dose: " + doseUnits + "  Accuracy: 1/" + lengthUnits.
        getAccuracy() + " " + lengthUnits;
   }

   private static UnitLength lengthUnits;
   private static UnitDose doseUnits;
}
```

UnitLength.java

```
/* written by Brian Chow
 * created March 17, 1998
 * last modified April 7, 1998
 */
/**
 *
 */
public class UnitLength implements Cloneable
{  /**
    *
    */
   public UnitLength(String units,int accuracy)
   {  setUnits(units);
      this.accuracy = accuracy;
```

```
   }

   public double inUnits(double value)
   {  if (cm)
      {  value /= 2.54;
      }
      return value;
   }

   public double outUnits(double value)
   {  if (cm)
      {  value *= 2.54;
      }
      // Rounds value to the nearest accuracy unit.  Note:  the floor
      // function was necessary to avoid an implementation error in the
      // Metrowerks VM for Macintosh dealing with negative values
      // otherwise, the round function could have been used.
      value = (Math.floor(value * accuracy + 0.5)) / (double)accuracy;
      return value;
   }

   public int getAccuracy()
   {  return accuracy;
   }

   public void setUnits(String units)
   {  if (units.equals("Centimeters"))
      {  cm = true;
      }
      else if (units.equals("Inches"))
      {  cm = false;
      }
      else
      {  System.err.println("Length unit error");
         cm = false;
      }
   }

   public void setAccuracy(int accuracy)
   {  this.accuracy = accuracy;
   }

   public boolean equals(String units)
   {  return this.toString().equals(units);
   }

   public String toString()
   {  if (cm)
      {  return "Centimeters";
      }
      else
      {  return "Inches";
      }
   }

   public Object clone()
   {  try
      {  return super.clone();
      } catch (CloneNotSupportedException e)
      {  // this shouldn't happen, since we are Cloneable
         return null;
```



```
      }
   }

   private boolean cm;
   private int accuracy;
}
```

UnitDose.java

```
/* written by Brian Chow
 * created March 17, 1998
 * last modified May 9, 1998
 */
/**
 *
 */
public class UnitDose implements Cloneable
{  /**
      *
       */
   public UnitDose(String units)
   {  setUnits(units);
   }

   public double inUnits(double value)
   {  if (grays)
      {  value *= 100;
      }
      return value;
   }

   public double outUnits(double value)
   {  int accuracy = 10;
      if (grays)
      {  value /= 100;
         accuracy *= 100;
      }
      // Rounds value to the nearest Rad unit.  Note:  the floor function
      // was necessary to avoid an implementation error in the Metrowerks
      // VM for Macintosh dealing with negative values otherwise, the
      // round function could have been used.
      value = Math.floor(value * accuracy + 0.5) / (double)accuracy;
      return value;
   }

   public void setUnits(String units)
   {  if (units.equals("Grays"))
      {  grays = true;
      }
      else if (units.equals("Rads"))
      {  grays = false;
      }
      else
      {  System.err.println("Radiation dose unit error");
         grays = false;
      }
   }

   public boolean equals(String units)
   {  return this.toString().equals(units);
```



```
   }

   public String toString()
   {  if (grays)
      {  return "Grays";
      }
      else
      {  return "Rads";
      }
   }

   public Object clone()
   {  try
      {  return super.clone();
      } catch (CloneNotSupportedException e)
      {  // this shouldn't happen, since we are Cloneable
         return null;
      }
   }

   private boolean grays;
}
```

Day.java

```
/* Modified by Brian Chow */

/*
 * Cay S. Horstmann & Gary Cornell, Core Java
 * Published By Sun Microsystems Press/Prentice-Hall
 * Copyright (C) 1997 Sun Microsystems Inc.
 * All Rights Reserved.
 *
 * Permission to use, copy, modify, and distribute this
 * software and its documentation for NON-COMMERCIAL purposes
 * and without fee is hereby granted provided that this
 * copyright notice appears in all copies.
 *
 * THE AUTHORS AND PUBLISHER MAKE NO REPRESENTATIONS OR
 * WARRANTIES ABOUT THE SUITABILITY OF THE SOFTWARE, EITHER
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHORS
 * AND PUBLISHER SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED
 * BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 */

/**
 * Store dates and perform date arithmetic
 * (another Date class, but more convenient that
 * java.util.Date or java.util.Calendar)
 * @version 1.02 13 Jun 1996
 * @author Cay Horstmann
 */

import java.util.*;

public class Day implements Cloneable
{  /**
```



```
    *  Constructs today's date
    */

   public Day()
   {  GregorianCalendar todaysDate
         = new GregorianCalendar();
      year = todaysDate.get(Calendar.YEAR);
      month = todaysDate.get(Calendar.MONTH) + 1;
      day = todaysDate.get(Calendar.DAY_OF_MONTH);
   }

   /**
    * Constructs a specific date
    * @param yyyy year (full year, e.g., 1996,
    * <i>not</i> starting from 1900)
    * @param m month
    * @param d day
    * @exception IllegalArgumentException if yyyy m d not a
    * valid date
    */

   public Day(int yyyy, int m, int d)
   {  year = yyyy;
      month = m;
      day = d;
      if (!isValid())
         throw new IllegalArgumentException();
   }

   /**
    * Advances this day by n days. For example.
    * d.advance(30) adds thirdy days to d
    * @param n the number of days by which to change this
    * day (can be < 0)
    */

   public void advance(int n)
   {  fromJulian(toJulian() + n);
   }

   public int getDay()
   /**
    * Gets the day of the month
    * @return the day of the month (1...31)
    */

   {  return day;
   }

   public int getMonth()
   /**
    * Gets the month
    * @return the month (1...12)
    */

   { return month;
   }

   public int getYear()
   /**
    * Gets the year
```

```
    * @return the year (counting from 0, <i>not</i> from 1900)
    */

   { return year;
   }

   /**
    * Gets the weekday
    * @return the weekday (0 = Sunday, 1 = Monday, ...,
    * 6 = Saturday)
    */

   public int weekday() { return (toJulian() + 1)% 7; }

   /**
    * The number of days between this and day parameter
    * @param b any date
    * @return the number of days between this and day parameter
    * and b (> 0 if this day comes after b)
    */

   public int daysBetween(Day b)
   {  return toJulian() - b.toJulian();
   }

   /**
    * Changes day
    */

   public void setDay(int yyyy, int m, int d)
   {  year = yyyy;
      month = m;
      day = d;
      if (!isValid())
         throw new IllegalArgumentException();
   }

   /**
    * A string representation of the day
    * @return a string representation of the day
    */

   public String toString()
   {  String m = "";
      String d = "";
      if (month < 10)
      {  m = "0";
      }
      if (day <10)
      {  d = "0";
      }
      return m + month + "/" + d + day + "/" + year;
   }

   /**
    * Makes a bitwise copy of a Day object
    * @return a bitwise copy of a Day object
    */

   public Object clone()
   {  try
```



```
         {  return super.clone();
         } catch (CloneNotSupportedException e)
         {  // this shouldn't happen, since we are Cloneable
            return null;
         }
      }

      /**
       * Computes the number of days between two dates
       * @return true iff this is a valid date
       */

      private boolean isValid()
      {  Day t = new Day();
         t.fromJulian(this.toJulian());
         return t.day == day && t.month == month
            && t.year == year;
      }

      private int toJulian()
      /**
       * @return The Julian day number that begins at noon of
       * this day
       * Positive year signifies A.D., negative year B.C.
       * Remember that the year after 1 B.C. was 1 A.D.
       *
       * A convenient reference point is that May 23, 1968 noon
       * is Julian day 2440000.
       *
       * Julian day 0 is a Monday.
       *
       * This algorithm is from Press et al., Numerical Recipes
       * in C, 2nd ed., Cambridge University Press 1992
       */
      {  int jy = year;
         if (year < 0) jy++;
         int jm = month;
         if (month > 2) jm++;
         else
         {  jy--;
            jm += 13;
         }
         int jul = (int) (java.lang.Math.floor(365.25 * jy)
         + java.lang.Math.floor(30.6001*jm) + day + 1720995.0);

         int IGREG = 15 + 31*(10+12*1582);
            // Gregorian Calendar adopted Oct. 15, 1582

         if (day + 31 * (month + 12 * year) >= IGREG)
            // change over to Gregorian calendar
         {  int ja = (int)(0.01 * jy);
            jul += 2 - ja + (int)(0.25 * ja);
         }
         return jul;
      }

      private void fromJulian(int j)
      /**
       * Converts a Julian day to a calendar date
       * @param j  the Julian date
       * This algorithm is from Press et al., Numerical Recipes
```



```
       * in C, 2nd ed., Cambridge University Press 1992
       */
      {  int ja = j;

         int JGREG = 2299161;
            /* the Julian date of the adoption of the Gregorian
               calendar
            */

         if (j >= JGREG)
         /* cross-over to Gregorian Calendar produces this
            correction
         */
         {  int jalpha = (int)(((float)(j - 1867216) - 0.25)
                / 36524.25);
            ja += 1 + jalpha - (int)(0.25 * jalpha);
         }
         int jb = ja + 1524;
         int jc = (int)(6680.0 + ((float)(jb-2439870) - 122.1)
             /365.25);
         int jd = (int)(365 * jc + (0.25 * jc));
         int je = (int)((jb - jd)/30.6001);
         day = jb - jd - (int)(30.6001 * je);
         month = je - 1;
         if (month > 12) month -= 12;
         year = jc - 4715;
         if (month > 2) --year;
         if (year <= 0) --year;
      }

      private int day;
      private int month;
      private int year;
   }
```

BundleList.java

```
   import java.util.*;

   public class BundleList implements Cloneable
   {

      public void add(RadBundle b)
      {  if (size() == 0)
         {  expBundle = b;
         }
         bundles.addElement(b);
      }

      public Enumeration elements()
      {  return bundles.elements();
      }

      public int size()
      {  return bundles.size();
      }

      public RadBundle find(String name)
      {  RadBundle b;
         for (int i = 0;i < bundles.size();i++)
```



```
         {  b = (RadBundle)bundles.elementAt(i);
            if (b.getName().equals(name))
            {  return b;
            }
         }
         return null;
      }

      public void setDate(Day d)
      {  RadBundle b;
         RadBundle.setDate(d);
         for (int i = 0;i < bundles.size();i++)
         {  b = (RadBundle)bundles.elementAt(i);
            b.recalculate();
         }
      }

      public void setExpBundle(String name)
      {  expBundle = find(name);
      }

      public RadBundle getExpBundle()
      {  return expBundle;
      }

      public Object clone()
      {  try
         {  return super.clone();
         } catch (CloneNotSupportedException e)
         {  // this shouldn't happen, since we are Cloneable
            return null;
         }
      }

      private Vector bundles = new Vector(3,3);
      private RadBundle expBundle = null;
   }
```

RadBundle.java

```
   /* written by Brian Chow
    * created February 21, 1998
    * last modified March 29, 1998
    */

   /**
    * Sets up radioactive source bundle with a measured activity on the
    * measurement day and calculates the current activity and radiation of
    * one source based on current (experiment) day.
    */
   public class RadBundle implements Cloneable
   {  /**
       * Cobalt-60 half life in days.  From:
       * <a href="http://physics.nist.gov/PhysRefData/Halflife/halflife.html">
       * Radionuclide Half-Life Measurements Made at NIST</a>,
       * http://physics.nist.gov/PhysRefData/Halflife/halflife.html,
       * accessed on March 2, 1998.
       */
      public static final double CoHalfLife = 1925.12;
      /**
```

```
 * Curies to rad/hr at one meter for Cobalt-60 is 1.32.  From:
 * Radiological Health Handbook, Jan 1970, p.131.
 */
public static final double CiToRadPerHour = 1.32 / Math.pow(0.0254,2);

/**
 * Creates new bundle with a unique name, one pencil per bundle and no
 * activity.
 */
public RadBundle()
{  name = "New" + numBundles;
   measureDate = new Day();
   measureActivity = 0;
   numInBundle = 1;
   numBundles++;
}

/**
 * Creates new bundle with the given name, measurement date, measured
 * activity, and number of pencils in the bundle.
 */
public RadBundle(String name,Day measureDate,double measureActivity,
  int numInBundle)
{  this.name = name;
   this.measureDate = measureDate;
   this.measureActivity = measureActivity;
   this.numInBundle = numInBundle;
   numBundles++;
}

/**
 * Returns name of source.
 */
public String getName()
{  return name;
}

/**
 * Returns the number of pencils in the bundle.
 */
public int getNumInBundle()
{  return numInBundle;
}

/**
 * Returns the measured activity.
 */
public double getMeasureActivity()
{  return measureActivity;
}

/**
 * Returns the measurement date.
 */
public Day getMeasureDate()
{  return measureDate;
}

/**
 * Returns current activity of one source.
 */
```



```
public double getActivity()
{  return curActivity;
}

/**
 * Returns current radiation of one source.
 */
public double getRad()
{  return curRad;
}

/**
 * Sets the bundle name to the argument.
 */
public void setName(String name)
{  this.name = name;
}

/**
 * Sets the number of pencils in the bundle to the argument.
 */
public void setNumInBundle(int numInBundle)
{  this.numInBundle = numInBundle;
   recalculate();
}

/**
 * Changes experiment date to argument.
 */
public static void setDate(Day d)
{  curDate = d;
}

/**
 * Force recalculation of activity and radiation.
 */
public void recalculate()
{  calculateActRad();
}

/**
 * Sets the measured activity to the argument.
 */
public void setMeasureActivity(double measureActivity)
{  this.measureActivity = measureActivity;
   recalculate();
}

/**
 * Sets the measurement date to the argument.
 */
public void setMeasureDate(Day measureDate)
{  this.measureDate = measureDate;
   recalculate();
}

/**
 * Returns the string representation of the bundle.
 */
public String toString()
{  return name + " Source";
```



```
}

/**
 * Makes a copy of the bundle.
 */
public Object clone()
{  RadBundle b = new RadBundle();
   b.name = name;
   b.measureDate = (Day)measureDate.clone();
   b.measureActivity = measureActivity;
   b.numInBundle = numInBundle;
   b.curActivity = curActivity;
   b.curRad = curRad;
   numBundles--;
   return b;
}

/**
 * Calculates current activity and radiation at one length unit.
 */
private void calculateActRad()
{  double elapsedTime = curDate.daysBetween(measureDate);
   curActivity = measureActivity *
     Math.pow(0.5,elapsedTime / CoHalfLife) / numInBundle;
   curRad = curActivity * CiToRadPerHour;
}

/**
 * User friendly name for bundle.
 */
private String name;

/**
 * Date activity of bundle was measured.
 */
private Day measureDate;

/**
 * Total measured activity of the bundle.
 */
private double measureActivity;

/**
 * Number of pencils in the bundle.
 */
private int numInBundle;

/**
 * Current calculated activity of one pencil.
 */
private double curActivity;

/**
 * Current water equivalent dose of one pencil at one distance unit.
 */
private double curRad;

/**
 * Date of experiment.
 */
private static Day curDate;
```



```
   /**
    * Counter for number of bundles created used for assigning a unique
    * name to new bundles.
    */
   private static int numBundles = 0;
}
```

Pencil.java

```
/* written by Brian Chow
 * created March 10, 1998
 * last modified May 12, 1998
 */
class Pencil
{  //
   public Pencil(Coordinate center,RadBundle source)
   {  this.center = center;
      radius = 0.375 / 2;
      length = 7.375;
      halfLength = length / 2;
      setMaxMin();
      this.source = source;
   }

   //
   public Pencil(Coordinate center,RadBundle source,double diameter,
     double length)
   {  this.center = center;
      radius = diameter / 2;
      this.length = length;
      halfLength = length / 2;
      setMaxMin();
      this.source = source;
   }

   public Coordinate getCenter()
   {  return center;
   }

   public void setCenter(Coordinate center)
   {  this.center = center;
      setMaxMin();
   }

   public RadBundle getSource()
   {  return source;
   }

   public void setSource(RadBundle source)
   {  this.source = source;
   }

   public double getAttCoefficient()
   {  return attCoefficient;
   }

   public void translate(double dX,double dY,double dZ)
   {  center.translate(dX,dY,dZ);
      setMaxMin();
```

```
   }

   /**
    * Given two endpoints of a line segment, return the intersection
    * distance with this pencil.
    * Reference: http://www.mhri.edu.au/~pdb/geometry/sphereline/
    */
   public double intersectDistance(Coordinate p1,Coordinate p2)
   {  if (p1.x <= min.x && p2.x <= min.x ||
        p1.y <= min.y && p2.z <= min.y ||
        p1.z <= min.z && p2.z <= min.z ||
        p1.x >= max.x && p2.x >= max.x ||
        p1.y >= max.y && p2.z >= max.y ||
        p1.z >= max.z && p2.z >= max.z)
      {  return 0.0;
      }
      final double dX,dY,dZ;
      dX = p2.x - p1.x;
      dY = p2.y - p1.y;
      dZ = p2.z - p1.z;
      final double uA;
      uA = ((center.x - p1.x) * dX + (center.y - p1.x) * dY) /
        (Math.pow(dX,2) + Math.pow(dY,2));
      if (uA < 0 || uA > 1)
      {  return 0.0;
      }
      Coordinate perpIntersect = new Coordinate();
      perpIntersect.x = p1.x + uA * dX;
      perpIntersect.y = p1.y + uA * dY;
      if (center.distanceZ(perpIntersect) >= radius)
      {  return 0.0;
      }
      else
      {  final boolean p1Inside,p2Inside;
         p1Inside = (center.distanceZ(p1) <= radius &&
           Math.abs(center.z - p1.z) <= halfLength);
         p2Inside = (center.distanceZ(p2) <= radius &&
           Math.abs(center.z - p2.z) <= halfLength);
         if (p1Inside && p2Inside)
         {  return p1.distance(p2);
         }
         else
         {  final double a,b,c,determinant;
            a = Math.pow(dX,2) + Math.pow(dY,2);
            b = 2 * (dX * (p1.x - center.x) + dY * (p1.y - center.y));
            c = Math.pow(center.x,2) + Math.pow(center.y,2) +
              Math.pow(p1.x,2) + Math.pow(p1.y,2) -
              2 * (center.x * p1.x + center.y * p1.y) -
              Math.pow(radius,2);
            determinant = Math.pow(b,2) - 4 * a * c;
            Coordinate end1 = null;
            Coordinate end2 = null;
            if (p1Inside)
            {  end1 = p1;
            }
            else if (determinant <= 0)
            {  if (dX == 0 && dY == 0)
               {  end1 = new Coordinate();
                  end1.x = p1.x;
                  end1.y = p1.y;
                  if (p1.z > p2.z)
```



```
                  {  end1.z = max.z;
                  }
                  else
                  {  end1.z = min.z;
                  }
               }
               else
               {  //System.err.println("Intersection Distance error");
                  return 0.0;
               }
            }
            else
            {  final double u1 = (-b - Math.sqrt(determinant)) /
                 (2 * a);
               end1 = new Coordinate();
               end1.x = p1.x + u1 * dX;
               end1.y = p1.y + u1 * dY;
//             double t = (end1.x - p1.x) / dX;
               end1.z = p1.z + u1 * dZ;
               if (end1.z > max.z)
               {  double u = (max.z - p1.z) / dZ;
                  end1.x = p1.x + u * dX;
                  end1.y = p1.y + u * dY;
                  end1.z = max.z;
               }
               else if (end1.z < min.z)
               {  double u = (min.z - p1.z) / dZ;
                  end1.x = p1.x + u * dX;
                  end1.y = p1.y + u * dY;
                  end1.z = min.z;
               }
            }
            if (p2Inside)
            {  end2 = p2;
            }
            else if (determinant <= 0)
            {  if (dX == 0 && dY == 0)
               {  end2 = new Coordinate();
                  end2.x = p2.x;
                  end2.y = p2.y;
                  if (p2.z > p1.z)
                  {  end2.z = max.z;
                  }
                  else
                  {  end2.z = min.z;
                  }
               }
               else
               {  //System.err.println("Intersection Distance error");
                  return 0.0;
               }
            }
            else
            {  final double u2 = (-b + Math.sqrt(determinant)) /
                 (2 * a);
               end2 = new Coordinate();
               end2.x = p1.x + u2 * dX;
               end2.y = p1.y + u2 * dY;
//             double t = (end2.x - p1.x) / dX;
               end2.z = p1.z + u2 * dZ;
               if (end2.z > max.z)
```



```
               {  double u = (max.z - p1.z) / dZ;
                  end2.x = p1.x + u * dX;
                  end2.y = p1.y + u * dY;
                  end2.z = max.z;
               }
               else if (end2.z < min.z)
               {  double u = (min.z - p1.z) / dZ;
                  end2.x = p1.x + u * dX;
                  end2.y = p1.y + u * dY;
                  end2.z = min.z;
               }
            }
//          System.out.println("*" + end1 + " * " + end2);
            return end1.distance(end2);
         }
      }
   }

   public String toString()
   {  return "" + Units.outLength(center) + " " + source;
   }

/*
   public static void main(String[] args)
   {  Pencil p = new Pencil(new Coordinate(1,1,1),new RadBundle());
      System.out.println(p);
      System.out.println(p.intersectDistance(new Coordinate(0,0,1),
        new Coordinate(2,2,1)));
      System.out.println(p.intersectDistance(new Coordinate(1.1,1,1),
        new Coordinate(-0.9,1,1)));
      System.out.println(p.intersectDistance(new Coordinate(0,-1,0),
        new Coordinate(0,1,0)));
      System.out.println(p.intersectDistance(new Coordinate(0,-1,3),
        new Coordinate(0,1,3)));
      System.out.println(p.intersectDistance(new Coordinate(1,0,10),
        new Coordinate(1,0,10)));
      System.out.println(p.intersectDistance(new Coordinate(0.15,0,10),
        new Coordinate(0.15,0,0)));
      System.out.println(p.intersectDistance(new
Coordinate(p.min.x,0,p.max.z),
        new Coordinate(p.max.x,0,p.min.z)));
      System.out.println(p.intersectDistance(new Coordinate(-
0.2,0,p.max.z),
        new Coordinate(0.2,0,p.min.z)));
      System.out.println(p.intersectDistance(new Coordinate(1,0,10),
        new Coordinate(1,0,10)));

   }
*/

   private void setMaxMin()
   {  min.x = center.x - radius;
      max.x = center.x + radius;
      min.y = center.y - radius;
      max.y = center.y + radius;
      min.z = center.z - halfLength;
      max.z = center.z + halfLength;
   }

   public final double radius;
   public final double length,halfLength;
```



```
   private static final double attCoefficient = -0.128 * 2.54;
   private Coordinate center;
   private Coordinate min = new Coordinate();
   private Coordinate max = new Coordinate();
   private RadBundle source;
}
```

Fixture.java

```
/* written by Brian Chow
 * created March 10, 1998
 * last modified May 12, 1998
 */
import java.util.*;

public class Fixture implements Cloneable
{  public Fixture()
   {  pencils = new Vector(12,8);
   }

   public Fixture(Fixture oldFixture)
   {  pencils = oldFixture.pencils;
   }

   public Enumeration elements()
   {  return pencils.elements();
   }

   public void add(Pencil p)
   {  pencils.addElement(p);
   }

   public void remove(int index)
   {  pencils.removeElementAt(index);
   }

   public Pencil getPencil(int index)
   {  return (Pencil)pencils.elementAt(index);
   }

   public void setPencil(Pencil p,int index)
   {  pencils.setElementAt(p,index);
   }

   public void setPencilCenter(int index,Coordinate center)
   {  ((Pencil)pencils.elementAt(index)).setCenter(center);
   }

   public void setPencilSource(int index,RadBundle source)
   {  ((Pencil)pencils.elementAt(index)).setSource(source);
   }

   public RadBundle getSource(int index)
   {  return ((Pencil)pencils.elementAt(index)).getSource();
   }

   public void setSource(RadBundle source)
   {  for (int i = 0;i < pencils.size();i++)
      {  ((Pencil)pencils.elementAt(i)).setSource(source);
      }
```

```
   }

   public void setSource(RadBundle source,int index)
   {  ((Pencil)pencils.elementAt(index)).setSource(source);
   }

   public int size()
   {  return pencils.size();
   }

   /**
    * Clone this fixture.
    */
   public Object clone()
   {  try
      {  return super.clone();
      } catch (CloneNotSupportedException e)
      {  // this shouldn't happen, since we are Cloneable
         return null;
      }
   }

   public String toString()
   {  return "Custom Fixture";
   }

   protected void translate(double dX,double dY,double dZ)
   {  for (int i = 0;i < pencils.size();i++)
      {  ((Pencil)pencils.elementAt(i)).translate(dX,dY,dZ);
      }
   }

   private Vector pencils;

}
```

AnnularFixture.java

```
/* written by Brian Chow
 * created March 11, 1998
 * last modified May 11, 1998
 */

import java.util.*;

public class AnnularFixture extends Fixture
{  /**
    * Constructs new annular fixture with specified center, sources,
    * radius, and number of pencils desired.
    */
   public AnnularFixture(Coordinate center,RadBundle source,
     double radius,int numPencils)
   {  this.center = center;
      this.radius = radius;
      this.numPencils = numPencils;
      // Distributes pencils an equal radial distance apart about the
      // origin.
      double deltaTheta = 360 / numPencils;
      Coordinate pencilCenter;
      for (int i = 0;i < numPencils;i++)
```



```
      {  pencilCenter = new CylinderCoordinate(radius,
           deltaTheta * i,0.0);
         add(new Pencil(pencilCenter,source));
      }
      // Shifts fixture from the origin to the desired center position.
      translate(center.x,center.y,center.z);
   }

   public Coordinate getCenter()
   {  return center;
   }

   public RadBundle getSource()
   {  return getSource(0);
   }

   public double getRadius()
   {  return radius;
   }

   public int getNumPencils()
   {  return numPencils;
   }

   public String toString()
   {  return "Annular: Center " + Units.outLength(center) + " Radius " +
        Units.outLength(radius) + " Pencils " + numPencils;
   }

/*
   public static void main(String[] args)
   {  Fixture f = new AnnularFixture(new Coordinate(),new RadBundle(),2,
        12);
      Enumeration fEnum = f.elements();
      while (fEnum.hasMoreElements())
      {  System.out.println("" + (Pencil)fEnum.nextElement());
      }
   }
*/

   private Coordinate center;
   private double radius;
   private int numPencils;
}
```

Coordinate.java

```
/* written by Brian Chow
 * created March 7, 1998
 * last modified May 2, 1998
 */

/**
 * Data structure that holds three double precision numbers used to
 * represent a point in rectangular coordinates.
 */

class Coordinate implements Cloneable
{  /**
    * Initializes coordinate to origin for no arguments.
```



```
    */
   public Coordinate()
   {  x = 0.0;
      y = 0.0;
      z = 0.0;
   }

   /**
    * Initializes coordinate to specified Cartesian coordinates.
    */
   public Coordinate(double x,double y,double z)
   {  this.x = x;
      this.y = y;
      this.z = z;
   }

   /**
    * Shifts point according to specified X, Y, and Z distances.
    */
   public void translate(double deltaX,double deltaY,double deltaZ)
   {  x += deltaX;
      y += deltaY;
      z += deltaZ;
   }

   /**
    * Rotates point in the XY plane by degree angle about point (x,y).
    */
   public void rotateXY(double theta,double aboutX,double aboutY)
   {  double tempX = x - aboutX;
      double tempY = y - aboutY;
      tempX = tempX * Math.cos(theta * Math.PI / 180) -
        tempY * Math.sin(theta * Math.PI / 180);
      tempY = tempY * Math.sin(theta * Math.PI / 180) +
        tempY * Math.cos(theta * Math.PI / 180);
      x = tempX + aboutX;
      y = tempY + aboutY;
   }

   /**
    * Calculates the distance between points.
    */
   public double distance(Coordinate c2)
   {  return Math.sqrt(Math.pow(x-c2.x,2) + Math.pow(y-c2.y,2) +
        Math.pow(z-c2.z,2));
   }

   /**
    * Calculates the distance between points projected onto the z plane.
    */
   public double distanceZ(Coordinate c2)
   {  return Math.sqrt(Math.pow(x-c2.x,2) + Math.pow(y-c2.y,2));
   }

   /**
    * Converts coordinate to (x,y,z) string.
    */
   public String toString()
   {  return ("(" + x + ", " + y + ", " + z + ")");
   }
```



```
   /**
    * Check equality of two points by checking each of the fields.
    */
   public boolean equals(Coordinate p)
   {  return (this.x == p.x) && (this.y == p.y) && (this.z == p.z);
   }

   /**
    * Makes a bitwise copy of this point.
    */
   public Object clone()
   {  try
      {  return super.clone();
      } catch (CloneNotSupportedException e)
      {  return null;
      }
   }

   /**
    * Double precision values for rectangular coordinates.
    */
   double x,y,z;
}
```

CylinderCoordinate.java

```
/* written by Brian Chow
 * created March 7, 1998
 * last modified March 16, 1998
 */

/**
 * Extension of coordinate that takes a cylindrical coordinate and stores
 * it as its rectangular equivalent.
 */
class CylinderCoordinate extends Coordinate
{  /**
    * Sets coordinate using cylindrical coordinate arguments (r,theta,z).
    */
   public CylinderCoordinate(double r,double theta,double z)
   {  x = r * Math.cos(theta * Math.PI / 180);
      y = r * Math.sin(theta * Math.PI / 180);
      this.z = z;
   }
}
```

Target.java

```
/* written by Brian Chow
 * created March 19, 1998
 * last modified April 25, 1998
 */

/**
 * Representation of a target.
 */
abstract class Target implements Cloneable
{  /**
    * Calculates doses at target points and keeps track of the
    * calculation time.
```

```
     */
    public void calculate(Fixture currentFixture)
    {  long timeBegin = System.currentTimeMillis();
       dose = Dose.pointSource(currentFixture,points);
       long timeEnd = System.currentTimeMillis();
       calculationTime = (int)(timeEnd - timeBegin);
    }

    /**
     * Clone this target.
     */
    public Object clone()
    {  try
       {  return super.clone();
       } catch (CloneNotSupportedException e)
       {  // this shouldn't happen, since we are Cloneable
          return null;
       }
    }

    /**
     * Array of points to represent target.
     */
    protected Coordinate[] points;
    /**
     * Array containing the dose at the corresponding target point.
     */
    protected double[] dose;
    /**
     * Calculation time in milliseconds.
     */
    protected int calculationTime;
}
```

TargetPoints.java

```
/* written by Brian Chow
 * created March 20, 1998
 * last modified May 9, 1998
 */

import java.util.*;

/**
 * Target consisting of a set of points.
 */
class TargetPoints extends Target
{  /**
    * Sets default point target with one point at the origin.
    */
   public TargetPoints()
   {  points = new Coordinate[1];
      points[0] = new Coordinate();
   }

   /**
    * Sets point target to the Vector of points specified.
    */
   public TargetPoints(Vector points)
   {  this.points = new Coordinate[points.size()];
```



```
      points.copyInto(this.points);
   }

   /**
    * Sets point target to the array of points specified.
    */
   public TargetPoints(Coordinate[] points)
   {  this.points = points;
   }

   /**
    * Calculates the dosage at each point in this target and outputs the
    * data to a calculation output window.
    */
   public void calculate(Fixture currentFixture)
   {  // Do calculations.
      super.calculate(currentFixture);

      // Output to user.
      CalculationOutput out = new CalculationOutput(25,75);
      out.setTitle("Calculation Results");
      out.println("Calculation date: " + new Day());
      out.println("Fixture " + currentFixture.toString());
      out.println("Target " + this.toString());
      out.println();
      out.print("Coordinate (" + Units.getLengthUnits() + ")",25);
      out.println("Dose (" + Units.getDoseUnits() + "/hour)",25);
      for (int i = 0;i < dose.length;i++)
      {  out.print(Units.outLength(points[i]).toString(),25);
         out.println("" + Units.outDose(dose[i]),25);
      }
      out.println();
      out.println("Calculation time: " + (calculationTime / 1000.0) +
        " s");
      out.setVisible(true);
   }

   /**
    * Returns array of points in this point target.
    */
   public Coordinate[] getPoints()
   {  return points;
   }

   /**
    * Sets points to the specified Vector of points.
    */
   public void setPoints(Vector points)
   {  this.points = new Coordinate[points.size()];
      points.copyInto(this.points);
   }

   /**
    * Sets points to the specified array of points.
    */
   public void setPoints(Coordinate[] points)
   {  this.points = points;
   }

   /**
    * String representation of this point target.
```



```
     */
    public String toString()
    {  return "User specified points";
    }

    /**
     * Clones this point target.
     */
    public Object clone()
    {  return super.clone();
    }
}
```

TargetLine.java

```
/* written by Brian Chow
 * created March 20, 1998
 * last modified April 24, 1998
 */

/**
 * Line target represented by a specified number of points evenly spaced
 * between two endpoints.
 */
class TargetLine extends TargetPoints
{  /**
    * Sets the default line target with two points.  One at the origin
    * and the other at (2,0,0).
    */
   public TargetLine()
   {  numPoints = 2;
      p1 = new Coordinate();
      p2 = new Coordinate(2,0,0);
      points = new Coordinate[numPoints];
      points[0] = p1;
      points[1] = p2;
   }

   /**
    * Sets the line target with the specified endpoints and number of
    * points.
    */
   public TargetLine(Coordinate p1,Coordinate p2,int numPoints)
   {  this.numPoints = numPoints;
      this.p1 = p1;
      this.p2 = p2;
      constructLine();
   }

   /**
    * Returns the first endpoint.
    */
   public Coordinate getPoint1()
   {  return p1;
   }

   /**
    * Returns the second endpoint.
    */
   public Coordinate getPoint2()
```



```
   {  return p2;
   }

   /**
    * Returns the number of points.
    */
   public int getNumPoints()
   {  return numPoints;
   }

   /**
    * Sets the line target to the specified endpoints and number of
    * points.
    */
   public void setLine(Coordinate point1,Coordinate point2,int numP)
   {  p1 = point1;
      p2 = point2;
      numPoints = numP;
      constructLine();
   }

   /**
    * Returns the string representation of the line target.
    */
   public String toString()
   {  if (numPoints == 1)
      {  return "Line: One point from " + Units.outLength(p1).toString()
           + " to " + Units.outLength(p2).toString();
      }
      else
      {  return "Line: " + numPoints + " points from " +
           Units.outLength(p1).toString() + " to " + Units.outLength(p2).
           toString();
      }
   }

   /**
    * Clones this line target.
    */
   public Object clone()
   {  return new TargetLine((Coordinate)p1.clone(),
        (Coordinate)p2.clone(),numPoints);
   }

/*
   public static void main(String[] args)
   {  Target t = new TargetLine(new Coordinate(1.5,1.5,1.5),
        new Coordinate(2.5,2.5,2.5),11);
      Units u = new Units();
      RadBundle b = new RadBundle("1979",new Day(1979,9,19),9950,12);
      b.setDate(new Day(1990,10,19));
      b.recalculate();
      Fixture f = new AnnularFixture(new Coordinate(),b,1,12);
      t.calculate(u,f);
   }
*/

   /**
    * Evenly distributes target points between the endpoints.
    */
   private void constructLine()
```

```
   {  double dx,dy,dz;
      dx = (p2.x - p1.x) / (double)(numPoints - 1);
      dy = (p2.y - p1.y) / (double)(numPoints - 1);
      dz = (p2.z - p1.z) / (double)(numPoints - 1);
      points = new Coordinate[numPoints];
      for (int i = 0;i < numPoints;i++)
      {  points[i] = new Coordinate(p1.x + dx * i,p1.y + dy * i,
           p1.z + dz * i);
      }
   }

   /**
    * Endpoints of the line target.
    */
   private Coordinate p1,p2;
   /**
    * Number of points for the line target.
    */
   private int numPoints;
}
```

TargetObject.java

```
/* written by Brian Chow
 * created March 23, 1998
 * last modified April 7, 1998
 */

/**
 *
 */
abstract class TargetObject extends TargetPoints
{  /**
    *
    */
   public void setHeight(double h)
   {  height = h;
      constructObject();
   }

   public void setCenter(Coordinate c)
   {  center = c;
      constructObject();
   }

   public double getHeight()
   {  return height;
   }

   public Coordinate getCenter()
   {  return center;
   }

   public Object clone()
   {  return super.clone();
   }

   abstract protected void constructObject();

   protected final void translate(Coordinate c)
```



```
   {  for (int i = 0; i < points.length;i++)
      {  points[i].translate(c.x,c.y,c.z);
      }
   }

   protected Coordinate center;
   protected double height;
}
```

TargetCyl.java

```
/* written by Brian Chow
 * created April 7, 1997
 * last modified April 7, 1997
 */

/**
 *
 */
class TargetCyl extends TargetObject
{  /**
    *
    */
   public TargetCyl()
   {  center = new Coordinate();
      height = 1;
      radius = 1;
      constructObject();
   }

   public TargetCyl(Coordinate c,double h,double r)
   {  center = c;
      height = h;
      radius = r;
      constructObject();
   }

   public double getRadius()
   {  return radius;
   }

   public void setCyl(Coordinate c,double h, double r)
   {  center = c;
      height = h;
      radius = r;
      constructObject();
   }

   public Object clone()
   {  return super.clone();
   }

   public String toString()
   {  return "Cylinder: Center " + Units.outLength(center) + " Height " +
        Units.outLength(height) + " Radius " + Units.outLength(radius);
   }

   protected void constructObject()
   {  points = new Coordinate[15];
      for (int i = -1;i <= 1;i++)
```



```
      {  final double z = height / 2 * i;
         points[5 * (i + 1)] = new Coordinate(0,-radius,z);
         points[5 * (i + 1) + 1] = new Coordinate(-radius,0,z);
         points[5 * (i + 1) + 2] = new Coordinate(0,0,z);
         points[5 * (i + 1) + 3] = new Coordinate(radius,0,z);
         points[5 * (i + 1) + 4] = new Coordinate(0,radius,z);
      }
      translate(center);
   }

   private double radius;
}
```

TargetRect.java

```
/* written by Brian Chow
 * created April 7, 1997
 * last modified April 7, 1997
 */

/**
 *
 */
class TargetRect extends TargetObject
{  /**
    *
    */
   public TargetRect()
   {  center = new Coordinate();
      height = 1;
      width = 1;
      length = 1;
      constructObject();
   }

   public TargetRect(Coordinate c,double h,double w,double l)
   {  center = c;
      height = h;
      width = w;
      length = l;
      constructObject();
   }

   public double getWidth()
   {  return width;
   }

   public double getLength()
   {  return length;
   }

   public void setRect(Coordinate c,double h, double w, double l)
   {  center = c;
      height = h;
      width = w;
      length = l;
      constructObject();
   }

   public String toString()
```



```
   {  return "Rectangular Parallelepiped: Center " +
        Units.outLength(center) + " Height " + Units.outLength(height) +
        " Width " + Units.outLength(width) + " Length " +
        Units.outLength(length);
   }

   public Object clone()
   {  return super.clone();
   }

   protected void constructObject()
   {  points = new Coordinate[27];
      int i = 0;
      for (int j = -1;j <= 1;j++)
      {  for (int k = -1;k <= 1;k++)
         {  for (int m = -1;m <= 1;m++)
            {  points[i] = new Coordinate(width / 2 * m,length / 2 * k,
                 height / 2 * j);
               i++;
            }
         }
      }
      translate(center);
   }

   private double width,length;
}
```

Dose.java

```
/* written by Brian Chow
 * created March 18, 1998
 * last modified May 4, 1998
 */

import java.util.*;

/**
 * Dose calculation method(s).
 */
class Dose
{  /**
    * Dose calculation of an array of target points using point sources
    * to approximate a fixture with vertically arranged cylindrical
    * pencils.
    */
   public static double[] pointSource(Fixture f,Coordinate[] c)
   {  // Setup window for user to see calculation progress.
      output.clear();
      output.show();

      double[] returnDoseArray = new double[c.length];
      double returnDose;
      Enumeration fEnum;
      Pencil p;
      // Calculate dose for each target point.
      for (int i = 0;i < c.length;i++)
      {  returnDose = 0;
         fEnum = f.elements();
         // Sum up dose for a single target point due to one pencil.
```

```
            while (fEnum.hasMoreElements())
            {  p = (Pencil)fEnum.nextElement();
               returnDose += pointSourceApprox(p,f,c[i]);
            }
            returnDoseArray[i] = returnDose * waterDoseFactor;
            output.print("" + Units.outLength(c[i]) + " " +
              Units.outDose(returnDose));
         }

         // Close the progress window.
         output.dispose();

         return returnDoseArray;
      }

      /**
       * Calculates dose at a single target point due to one pencil using a
       * point source approximation.
       */
      private static double pointSourceApprox(Pencil p,Fixture f,
        Coordinate c)
      {  double dose = 0;
         // Radiation of one point source.
         final double chunkRad = p.getSource().getRad() / numPoints;
         // Distance between each point source.
         final double chunkSize = p.length / numPoints;
         final Coordinate pencilCenter = p.getCenter();
         final double z = pencilCenter.z;
         // Current point source location.
         Coordinate point = new Coordinate(pencilCenter.x,pencilCenter.y,z);
         for (int i = -numPoints + 1;i <= numPoints - 1;i += 2)
         {  // Move point source along z axis.
            point.z = z + (chunkSize * i / 2.0);
            double attFactor = 0;
            Enumeration fEnum = f.elements();
            // Calculate attenuation factor from the source to the target.
            while (fEnum.hasMoreElements())
            {  Pencil tempP = (Pencil)fEnum.nextElement();
               attFactor += tempP.intersectDistance(point,c) *
                 tempP.getAttCoefficient();
            }
            dose += chunkRad / Math.pow(point.distance(c),2) *
              Math.exp(attFactor);
         }
         return dose;
      }

      /**
       * Number of points used in approximating a single pencil.
       */
      private static int numPoints = 12;
      private static double waterDoseFactor = 0.96;
      /**
       * Output window for displaying calculation progress to user.
       */
      private static CalculationProgress output = new CalculationProgress();
   }
```