Input and Output
Sample Source Programs

```
// CopyFileUsingByteStream.java: Copy files

import java.io.*;
public class CopyFileUsingByteStream
{
  // Main method: args[0] for sourcefile and args[1] for target file

  public static void main( String[] args )
  {

    // Declare input and output file streams
    FileInputStream   fis = null;
    FileOutputStream fos = null;

    // Check usage
    if ( args.length !=2 )
    {
      System.out.println( "Usage: java CopyFileUsingByteStream fromfile tofile" );
      System.exit(0);
    }

    try
    {
      // Create file input stream
      fis = new FileInputStream( new File(args[0]) );

      // Create file output stream if the file does not exist
      File outFile = new File(args[1]);

      if ( outFile.exists() )
      {
        System.out.println( "file " + args[1] + " already exists" );
        return;
      }
      else
        fos = new FileOutputStream(args[1]);
```

```
      // Display the file size
      System.out.println( "The file " + args[0] + " has "+ fis.available() + " bytes" );

      // Continuously read a byte from fis and write it to fos
      int r;
      while ((r = fis.read()) != -1)
                fos.write( (byte)r );
      }

      catch ( FileNotFoundException ex )
      {
        System.out.println( "File not found: " + args[0] );
      }
      catch ( IOException ex )
      {
        System.out.println( ex.getMessage() );
      }
      finally
      {
        try
        {
          // Close files
          if (fis != null)
                    fis.close();
          if (fos != null)
                    fos.close();
        }
        catch (IOException ex)
        {
          System.out.println(ex);
        }
      }
  }
}
```

```
// TestDataStreams.java: Create a file, store it in binary form, and
// display it on the console
import java.io.*;
public class TestDataStreams
{
  // Main method
  public static void main( String[] args )
  {
    // Declare data input and output streams
    DataInputStream          dis = null;
    DataOutputStream         dos = null;

    // Construct a temp file
    File tempFile = new File("mytemp.dat");

    // Check if the temp file exists
    if ( tempFile.exists() )
    {
      System.out.println( "The file mytemp.dat already exists,"
                                + " delete it, rerun the program" );
      System.exit(0);
    }

    // Write data
    try
    {
      // Create data output stream for tempFile
      dos = new DataOutputStream( new FileOutputStream(tempFile) );

      for (int i=0; i<10; i++)
        dos.writeInt( (int)(Math.random()*1000) );
    }

    catch ( IOException ex )
    {
      System.out.println( ex.getMessage() );
    }
    finally  {
      try
      {
        // Close files
        if (dos != null)
                  dos.close();
      }
      catch (IOException ex)  { }
    }
```

```
    // Read data
    try
    {
      // Create data input stream
      dis = new DataInputStream( new FileInputStream(tempFile) );
      for (int i=0; i<10; i++)
                  System.out.print( "  " + dis.readInt() );
    }

    catch (FileNotFoundException ex)
    {
      System.out.println( "File not found" );
    }
    catch (IOException ex)
    {
      System.out.println(ex.getMessage());
    }
    finally
    {
      try
      {
        // Close files
        if (dis != null) dis.close();
      }
      catch (IOException ex)
      {
        System.out.println(ex);
      }
    }
  }
}
```

```java
// TestPrintWriters.java: Create a text file using PrintWriter
import java.io.*;

public class TestPrintWriters
{
  // Main method: args[0] is the output file
  public static void main( String[] args )
  {
    // Declare print stream
    PrintWriter pw = null;

    // Check usage
    if (args.length != 1)
    {
      System.out.println( "Usage: java TestPrintWriters file" );
      System.exit(0);
    }

    File tempFile = new File(args[0]);
    if ( tempFile.exists() )
    {
      System.out.println( "The file " + args[0] +
                  " already exists, delete it, rerun the program" );
      System.exit(0);
    }

    // Write data
    try
    {
      // Create print writer stream for tempFile
      pw = new PrintWriter(new FileOutputStream(tempFile), true);

      for (int i=0; i<10; i++)
        pw.print( " " + (int)(Math.random()*1000) );
    }
    catch (IOException ex)
    {
      System.out.println(ex.getMessage());
    }
    finally
    {
      // Close files
      if (pw != null) pw.close();
    }
  }
}
```

```
// ViewFile.java: Read a text file and store it in a text area
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class ViewFile extends MyFrameWithExitHandling implements ActionListener
{
  // Button to view
  private JButton jbtView = new JButton("View");

  // Text field to receive file name
  private JTextField jtf = new JTextField(12);

  // Text area to display file
  private JTextArea jta = new JTextArea();

  // Main method
  public static void main(String[] args)
  {
    ViewFile frame = new ViewFile();
    frame.setTitle("View File");
    frame.setSize(400, 300);
    frame.setVisible(true);
  }

  // Constructor
  public ViewFile()  {
    // Panel p to hold a label, a text field, and a button
    Panel p = new Panel();
    p.setLayout(new BorderLayout());
    p.add(new Label("Filename"), BorderLayout.WEST);
    p.add(jtf, BorderLayout.CENTER);
    jtf.setBackground(Color.yellow);
    jtf.setForeground(Color.red);
    p.add(jbtView, BorderLayout.EAST);

    // Add jta to a scroll pane
    JScrollPane jsp = new JScrollPane(jta);
    // Add jsp and p to the frame
    getContentPane().add(jsp, BorderLayout.CENTER);
    getContentPane().add(p, BorderLayout.SOUTH);

    // Register listener
    jbtView.addActionListener(this);
  }
```

```
// Handle the "View" button
public void actionPerformed(ActionEvent e)
{
  if (e.getSource() == jbtView)
    showFile();
}

// Display the file in the text area
private void showFile()
{
  // Use a BufferedStream to read text from the file
  BufferedReader infile = null;

  // Get file name from the text field
  String filename = jtf.getText().trim();

  String inLine;

  try
  {
    // Create a buffered stream
    infile = new BufferedReader(new FileReader(filename));

    // Read a line
    inLine = infile.readLine();

    boolean firstLine = true;

    // Append the line to the text area
    while (inLine != null)
    {
      if (firstLine)
      {
        firstLine = false;
        jta.append(inLine);
      }
      else
      {
        jta.append( "\n" + inLine );
      }

      inLine = infile.readLine();
    }
  }
```

```
    catch (FileNotFoundException ex)
    {
      System.out.println( "File not found: " + filename );
    }
    catch (IOException ex)
    {
      System.out.println(ex.getMessage());
    }
    finally
    {
      try
      {
        if (infile != null)
                    infile.close();
      }
      catch (IOException ex)
      {
        System.out.println(ex.getMessage());
      }
    }
  }
}
```

```java
// ParsingTextFile.java: Process text file using StreamTokenizer
import java.io.*;

public class ParsingTextFile
{

  // Main method
  public static void main(String[] args)
  {

    // Declare file reader and writer streams
    FileReader frs = null;
    FileWriter fws = null;

    // Declare streamTokenizer
    StreamTokenizer in = null;

    // Declare a print stream
    PrintWriter out = null;

    // For input file fields: student name, midterm1,
    // midterm2, and final exam score
    String sname = null;
    double midterm1 = 0;
    double midterm2 = 0;
    double finalScore = 0;

    // Computed total score
    double total = 0;

    try
    {
      // Create file input and output streams
      frs = new FileReader("in.dat");
      fws = new FileWriter("out.dat");

      // Create a stream tokenizer wrapping file input stream
      in = new StreamTokenizer(frs);
      out = new PrintWriter(fws);

      // Read first token
      in.nextToken();
```

```
  // Process a record
  while (in.ttype != in.TT_EOF)
  {
    // Get student name
    if (in.ttype == in.TT_WORD)
      sname = in.sval;
    else
      System.out.println( "Bad file format" );

    // Get midterm1
    if (in.nextToken() == in.TT_NUMBER)
      midterm1 = in.nval;
    else
      System.out.println( "Bad file format" );

    // Get midterm2
    if (in.nextToken() == in.TT_NUMBER)
      midterm2 = in.nval;
    else
      System.out.println( "Bad file format" );

    // Get final score
    if (in.nextToken() == in.TT_NUMBER)
      finalScore = in.nval;

    total = midterm1*0.3 + midterm2*0.3 + finalScore*0.4;
    out.println(sname + " " +total);
    in.nextToken();
  }
}

catch (FileNotFoundException ex)
{
  System.out.println( "File not found: in.dat" );
}
catch (IOException ex)
{
  System.out.println( ex.getMessage() );
}
```

```
    finally
    {
      try
      {
        if (frs != null)
                    frs.close();
        if (fws != null)
                    fws.close();
      }
      catch (IOException ex)
      {
        System.out.println(ex);
      }
    }
  }
}
```

```java
// TestRandomAccessFile.java: Store and read data using RandomAccessFile
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;

public class TestRandomAccessFile extends MyFrameWithExitHandling
{
  // Create a tabbed pane to hold two panels
  private JTabbedPane jtpStudent = new JTabbedPane();

  // Random access file for access the student.dat file
  private RandomAccessFile raf;

  // Main method
  public static void main(String[] args)
  {
    TestRandomAccessFile frame = new TestRandomAccessFile();
    frame.pack();
    frame.setTitle( "Test RandomAccessFile" );
    frame.setVisible(true);
  }

  // Default constructor
  public TestRandomAccessFile()
  {
    // Open or create a random access file
    try
    {
      raf = new RandomAccessFile( "student.dat", "rw" );
    }
    catch(IOException ex)
    {
      System.out.print("Error: " + ex);
      System.exit(0);
    }

    // Place buttons in the tabbed pane
    jtpStudent.add(new RegisterStudent(raf), "Register Student");
    jtpStudent.add(new ViewStudent(raf), "View Student");

    // Add the tabbed pane to the frame
    getContentPane().add(jtpStudent);
  }
}
```

```
// Register student panel
class RegisterStudent extends JPanel implements ActionListener
{
  // Button for registering a student
  private JButton jbtRegister;

  // Student inforamtion panel
  private StudentPanel studentPanel;

  // Random access file
  private RandomAccessFile raf;

  // Constructor
  public RegisterStudent(RandomAccessFile raf)
  {
    // Pass raf to RegisterStudent Panel
    this.raf = raf;

    // Add studentPanel and jbtRegister in the panel
    setLayout(new BorderLayout());
    add(studentPanel = new StudentPanel(), BorderLayout.CENTER);
    add(jbtRegister = new JButton("Register"), BorderLayout.SOUTH);

    // Register listener
    jbtRegister.addActionListener(this);
  }

  // Handle button actions
  public void actionPerformed(ActionEvent e)
  {
    if (e.getSource() == jbtRegister)
    {
      Student student = studentPanel.getStudent();

      try
      {
        raf.seek( raf.length() );
        student.writeStudent(raf);
      }
      catch(IOException ex)
      {
        System.out.print( "Error: " + ex );
      }
    }
  }
}
```

```
// View student panel
class ViewStudent extends JPanel implements ActionListener
{
  // Buttons for viewing student information
  private JButton jbtFirst, jbtNext, jbtPrevious, jbtLast;

  // Random access file
  private RandomAccessFile raf = null;

  // Current student record
  private Student student = new Student();

  // Create a student panel
  private StudentPanel studentPanel = new StudentPanel();

  // File pointer in the random access file
  private long lastPos;
  private long currentPos;

  // Constructor
  public ViewStudent(RandomAccessFile raf)
  {
    // Pass raf to ViewStudent
    this.raf = raf;

    // Panel p to hold four navigator buttons
    JPanel p = new JPanel();
    p.setLayout(new FlowLayout(FlowLayout.CENTER));
    p.add(jbtFirst = new JButton("First"));
    p.add(jbtNext = new JButton("Next"));
    p.add(jbtPrevious = new JButton("Previous"));
    p.add(jbtLast = new JButton("Last"));

    // Add panel p and studentPanel to ViewPanel
    setLayout(new BorderLayout());
    add(studentPanel, BorderLayout.CENTER);
    add(p, BorderLayout.SOUTH);

    // Register listeners
    jbtFirst.addActionListener(this);
    jbtNext.addActionListener(this);
    jbtPrevious.addActionListener(this);
    jbtLast.addActionListener(this);
  }
```

```
// Handle navigation button actions
public void actionPerformed(ActionEvent e)
{
  String actionCommand = e.getActionCommand();

  if (e.getSource() instanceof JButton)
  {
    try
    {
      if ( "First".equals(actionCommand) )
      {
        if ( raf.length() > 0 )
          retrieve(0);
      }

      else if ( "Next".equals(actionCommand) )
      {
        currentPos = raf.getFilePointer();

        if ( currentPos < raf.length() )
          retrieve(currentPos);
      }
      else if ( "Previous".equals(actionCommand) )
      {
        currentPos = raf.getFilePointer();

        if ( currentPos > 0 )
          retrieve( currentPos - 2*2*Student.RECORD_SIZE );
      }
      else if ( "Last".equals(actionCommand) )
      {
        lastPos = raf.length();

        if ( lastPos > 0 )
          retrieve(lastPos - 2*Student.RECORD_SIZE);
      }
    }
    catch(IOException ex)
    {
      System.out.print("Error: " + ex);
    }
  }
}
```

```
  // Retrieve a record at specified position
  public void retrieve(long pos)
  {
    try
    {
      raf.seek( pos );
      student.readStudent( raf );
      studentPanel.setStudent( student );
    }
    catch(IOException ex)
    {
      System.out.print("Error: " + ex);
    }
  }
}

// This class contains static methods for reading and writing fixed length records
class FixedLengthStringIO
{
  // Read fixed number of characters from a DataInput stream
  public static String readFixedLengthString( int size, DataInput in )
                              throws IOException
  {
    char c[] = new char[size];

    for (int i=0; i<size; i++)
      c[i] = in.readChar();

    return new String(c);
  }

  // Write fixed number of characters (string s with padded spaces) to a DataOutput stream
  public static void writeFixedLengthString(String s, int size, DataOutput out)
                              throws IOException
  {
    char cBuffer[] = new char[size];

    s.getChars( 0, s.length(), cBuffer, 0 );

    for ( int i=s.length(); i<cBuffer.length; i++ )
                  cBuffer[i] = ' ';

    String newS = new String(cBuffer);
    out.writeChars(newS);
  }
}
```

```
// FileDialogDemo.java: Demonstrate using JFileDialog to display
// file dialog boxes for opening and saving files
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;

public class FileDialogDemo extends MyFrameWithExitHandling
                implements ActionListener
{
  // Menu items Open, Save, exit, and About
  private JMenuItem jmiOpen, jmiSave,jmiExit, jmiAbout;

  // Text area for displaying and editing text files
  private JTextArea jta = new JTextArea();

  // Status label for displaying operation status
  private JLabel jlblStatus = new JLabel();

  // File dialog box
  private JFileChooser jFileChooser = new JFileChooser();

  // Main method
  public static void main(String[] args)
  {
    FileDialogDemo frame = new FileDialogDemo();
    frame.setSize(300, 150);
    frame.setVisible(true);
  }

  public FileDialogDemo()
  {
    setTitle( "Test JFileChooser" );

    // Create a menu bar mb and attach to the frame
    JMenuBar mb = new JMenuBar();
    setJMenuBar(mb);

    // Add a "File" menu in mb
    JMenu fileMenu = new JMenu( "File" );
    mb.add(fileMenu);

    // Add a "Help" menu in mb
    JMenu helpMenu = new JMenu( "Help" );
    mb.add(helpMenu);
```

```
  // Create and add menu items to the menu
  fileMenu.add( jmiOpen = new JMenuItem( "Open") );
  fileMenu.add( jmiSave = new JMenuItem("Save") );
  fileMenu.addSeparator();
  fileMenu.add( jmiExit = new JMenuItem("Exit") );
  helpMenu.add( jmiAbout = new JMenuItem("About") );

  // Set default directory to the current directory
  jFileChooser.setCurrentDirectory( new File(".") );

  // Set BorderLayout for the frame
  getContentPane().add(new JScrollPane(jta), BorderLayout.CENTER);
  getContentPane().add(jlblStatus, BorderLayout.SOUTH);

  // Register listeners
  jmiOpen.addActionListener(this);
  jmiSave.addActionListener(this);
  jmiAbout.addActionListener(this);
  jmiExit.addActionListener(this);
}

// Handle ActionEvent for menu items
public void actionPerformed(ActionEvent e)
{
  String actionCommand = e.getActionCommand();

  if (e.getSource() instanceof JMenuItem)
  {
    if ( "Open".equals(actionCommand) )
                open();
    else if ( "Save".equals(actionCommand) )
                save();
    else if ( "About".equals(actionCommand) )
                JOptionPane.showMessageDialog(
                        this,
                        "Demonstrate Using File Dialogs",
                        "About This Demo",
                        JOptionPane.INFORMATION_MESSAGE);
    else if ( "Exit".equals(actionCommand) )
                System.exit(0);
  }
}
```

```
// Open file
private void open()
{
  if ( jFileChooser.showOpenDialog(this) == JFileChooser.APPROVE_OPTION )
  {
    open( jFileChooser.getSelectedFile() );
  }
}

// Open file with the specified File instance
private void open(File file)
{
  try
  {
    // Read from the specified file and store it in jta
    BufferedInputStream in = new BufferedInputStream(new FileInputStream(file));

    byte[] b = new byte[ in.available() ];
    in.read(b, 0, b.length);
    jta.append(new String(b, 0, b.length));
    in.close();

    // Display the status of the Open file operation in jlblStatus
    jlblStatus.setText(file.getName() + " Opened");
  }
  catch (IOException ex)
  {
    jlblStatus.setText( "Error opening " + file.getName() );
  }
}

// Save file
private void save()
{
  if (jFileChooser.showSaveDialog(this) == JFileChooser.APPROVE_OPTION)
  {
    save( jFileChooser.getSelectedFile() );
  }
}
```

```
  // Save file with specified File instance
  private void save(File file)
  {
    try
    {
      // Write the text in jta to the specified file
      BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream(file));

      byte[] b = (jta.getText()).getBytes();
      out.write(b, 0, b.length);
      out.close();

      // Display the status of the save file operation in jlblStatus
      jlblStatus.setText( file.getName()  + " Saved " );
    }
    catch (IOException ex)
    {
      jlblStatus.setText( "Error saving " + file.getName() );
    }
  }
}
```

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

public class MyInput
{
  static private StringTokenizer stok;
  static private BufferedReader br =
                    new BufferedReader( new InputStreamReader(System.in), 1 );

  public static int readInt()
  {
    int i = 0;
    try
    {
      String str = br.readLine();
      StringTokenizer stok = new StringTokenizer(str);
      i = new Integer(stok.nextToken()).intValue();
    }
    catch (IOException ex)
    {
      System.out.println(ex);
    }
    return i;
  }

  public static double readDouble()
  {
    double d = 0;
    try
    {
      String str = br.readLine();
      stok = new StringTokenizer(str);
      d = new Double(stok.nextToken()).doubleValue();
    }
    catch (IOException ex)
    {
      System.out.println(ex);
    }
    return d;
  }
}
```

```
// MyFrameWithExitHandling.java: Define a new frame with exit
// capability and the center() method

import java.awt.*;
import java.awt.event.*;
import javax.swing.JFrame;

public class MyFrameWithExitHandling extends JFrame
        implements WindowListener
{
  // Main method
  public static void main(String[] args)
  {
    MyFrameWithExitHandling frame =
      new MyFrameWithExitHandling("Test Frame");
    frame.setSize(200, 200);
    frame.center();
    frame.setVisible(true);
  }

  // Default constructor
  public MyFrameWithExitHandling()
  {
    super();
    addWindowListener(this);                    // Register listener
  }

  // Constructor a frame with a title
  public MyFrameWithExitHandling(String title)
  {
    super(title);
    addWindowListener(this);                    // Register listener
  }

  // Center the frame
  public void center()
  {
    // Get the screen dimension
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int screenWidth = screenSize.width;
    int screenHeight = screenSize.height;

    // Get the frame dimension
    Dimension frameSize = this.getSize();
    int x = (screenWidth - frameSize.width)/2;
    int y = (screenHeight - frameSize.height)/2;
```

```
  // Determine the location of the left corner of the frame
  if (x < 0)
  {
    x = 0;
    frameSize.width = screenWidth;
  }

  if (y < 0)
  {
    y = 0;
    frameSize.height = screenHeight;
  }

  // Set the frame to the specified location
  this.setLocation(x, y);
}

// Handler for window closed event
public void windowClosed(WindowEvent event)
{
}

// Handler for window deiconified event
public void windowDeiconified(WindowEvent event)
{
}

// Handler for window iconified event
public void windowIconified(WindowEvent event)
{
}

// Handler for window activated event
public void windowActivated(WindowEvent event)
{
}

// Handler for window deactivated event
public void windowDeactivated(WindowEvent event)
{
}
```

```
  // Handler for window opened event
  public void windowOpened(WindowEvent event)
  {
  }

  // Handler for window closing event
  public void windowClosing(WindowEvent event)
  {
    dispose();
    System.exit(0);
  }
}
```