How to create a download manager in Java

This example shows how to create a simple download manager in Java. It contains four classes in foru Java source files:

- **Download.java:** Contains Download class which downloads a file from a URL.
- **DownloadManager.java:** Contains the main class for download manager application.
- **DownloadsTableModel.java:** Contains the class which manages the download table's data.
- **ProgressRenderer.java:** Contains the class which is responsible to render a JProgressBar in a table cell.

The contents of the listed files are written below.

Download.java

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

// This class downloads a file from a URL.
class Download extends Observable implements Runnable {

    // Max size of download buffer.
    private static final int MAX_BUFFER_SIZE = 1024;

    // These are the status names.
    public static final String STATUSES[] = {"Downloading",
    "Paused", "Complete", "Cancelled", "Error"};

    // These are the status codes.
    public static final int DOWNLOADING = 0;
    public static final int PAUSED = 1;
    public static final int COMPLETE = 2;
    public static final int CANCELLED = 3;
    public static final int ERROR = 4;

    private URL url; // download URL
    private int size; // size of download in bytes
    private int downloaded; // number of bytes downloaded
    private int status; // current status of download

    // Constructor for Download.
    public Download(URL url) {
        this.url = url;
        size = -1;
        downloaded = 0;
        status = DOWNLOADING;

        // Begin the download.
        download();
    }

    // Get this download's URL.
    public String getUrl() {
        return url.toString();
    }

    // Get this download's size.
    public int getSize() {
        return size;
    }

    // Get this download's progress.
    public float getProgress() {
        return ((float) downloaded / size) * 100;
    }

    // Get this download's status.
    public int getStatus() {
        return status;
    }

    // Pause this download.
    public void pause() {
        status = PAUSED;
        stateChanged();
    }

    // Resume this download.
    public void resume() {
        status = DOWNLOADING;
        stateChanged();
        download();
    }

    // Cancel this download.
    public void cancel() {
        status = CANCELLED;
        stateChanged();
    }

    // Mark this download as having an error.
    private void error() {
        status = ERROR;
        stateChanged();
    }

    // Start or resume downloading.
    private void download() {
        Thread thread = new Thread(this);
        thread.start();
    }

    // Get file name portion of URL.
    private String getFileName(URL url) {
        String fileName = url.getFile();
        return fileName.substring(fileName.lastIndexOf('/') + 1);
    }

    // Download file.
    public void run() {
        RandomAccessFile file = null;
        InputStream stream = null;

        try {
            // Open connection to URL.
            HttpURLConnection connection =
                    (HttpURLConnection) url.openConnection();

            // Specify what portion of file to download.
            connection.setRequestProperty("Range",
                    "bytes=" + downloaded + "-");

            // Connect to server.
            connection.connect();

            // Make sure response code is in the 200 range.
            if (connection.getResponseCode() / 100 != 2) {
                error();
            }

            // Check for valid content length.
            int contentLength = connection.getContentLength();
            if (contentLength < 1) {
                error();
            }

      /* Set the size for this download if it
         hasn't been already set. */
            if (size == -1) {
                size = contentLength;
                stateChanged();
            }

            // Open file and seek to the end of it.
            file = new RandomAccessFile(getFileName(url), "rw");
            file.seek(downloaded);

            stream = connection.getInputStream();
            while (status == DOWNLOADING) {
        /* Size buffer according to how much of the
           file is left to download. */
```

```java
                byte buffer[];
                if (size - downloaded > MAX_BUFFER_SIZE) {
                    buffer = new byte[MAX_BUFFER_SIZE];
                } else {
                    buffer = new byte[size - downloaded];
                }

                // Read from server into buffer.
                int read = stream.read(buffer);
                if (read == -1)
                    break;

                // Write buffer to file.
                file.write(buffer, 0, read);
                downloaded += read;
                stateChanged();
            }

      /* Change status to complete if this point was
         reached because downloading has finished. */
            if (status == DOWNLOADING) {
                status = COMPLETE;
                stateChanged();
            }
        } catch (Exception e) {
            error();
        } finally {
            // Close file.
            if (file != null) {
                try {
                    file.close();
                } catch (Exception e) {}
            }

            // Close connection to server.
            if (stream != null) {
                try {
                    stream.close();
                } catch (Exception e) {}
            }
        }
    }

    // Notify observers that this download's status has changed.
    private void stateChanged() {
        setChanged();
        notifyObservers();
    }
}
```

DownloadManager.java

```java
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

// The Download Manager.
public class DownloadManager extends JFrame
        implements Observer {

    // Add download text field.
    private JTextField addTextField;

    // Download table's data model.
    private DownloadsTableModel tableModel;

    // Table listing downloads.
    private JTable table;

    // These are the buttons for managing the selected download.
    private JButton pauseButton, resumeButton;
    private JButton cancelButton, clearButton;

    // Currently selected download.
    private Download selectedDownload;

    // Flag for whether or not table selection is being cleared.
    private boolean clearing;

    // Constructor for Download Manager.
    public DownloadManager() {
        // Set application title.
        setTitle("Download Manager");

        // Set window size.
        setSize(640, 480);

        // Handle window closing events.
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                actionExit();
            }
        });

        // Set up file menu.
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        fileMenu.setMnemonic(KeyEvent.VK_F);
        JMenuItem fileExitMenuItem = new JMenuItem("Exit",
                KeyEvent.VK_X);
        fileExitMenuItem.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                actionExit();
            }
        });
        fileMenu.add(fileExitMenuItem);
        menuBar.add(fileMenu);
        setJMenuBar(menuBar);

        // Set up add panel.
        JPanel addPanel = new JPanel();
        addTextField = new JTextField(30);
        addPanel.add(addTextField);
        JButton addButton = new JButton("Add Download");
        addButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                actionAdd();
            }
        });
        addPanel.add(addButton);

        // Set up Downloads table.
        tableModel = new DownloadsTableModel();
        table = new JTable(tableModel);
        table.getSelectionModel().addListSelectionListener(new
                ListSelectionListener() {
            public void valueChanged(ListSelectionEvent e) {
                tableSelectionChanged();
            }
        });
        // Allow only one row at a time to be selected.
        table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        // Set up ProgressBar as renderer for progress column.
        ProgressRenderer renderer = new ProgressRenderer(0, 100);
        renderer.setStringPainted(true); // show progress text
        table.setDefaultRenderer(JProgressBar.class, renderer);

        // Set table's row height large enough to fit JProgressBar.
        table.setRowHeight(
                (int) renderer.getPreferredSize().getHeight());

        // Set up downloads panel.
        JPanel downloadsPanel = new JPanel();
        downloadsPanel.setBorder(
                BorderFactory.createTitledBorder("Downloads"));
        downloadsPanel.setLayout(new BorderLayout());
        downloadsPanel.add(new JScrollPane(table),
                BorderLayout.CENTER);

        // Set up buttons panel.
        JPanel buttonsPanel = new JPanel();
        pauseButton = new JButton("Pause");
        pauseButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                actionPause();
            }
        });
        pauseButton.setEnabled(false);
        buttonsPanel.add(pauseButton);
        resumeButton = new JButton("Resume");
        resumeButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                actionResume();
            }
        });
        resumeButton.setEnabled(false);
        buttonsPanel.add(resumeButton);
        cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                actionCancel();
            }
        });
        cancelButton.setEnabled(false);
        buttonsPanel.add(cancelButton);
        clearButton = new JButton("Clear");
        clearButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                actionClear();
            }
        });
        clearButton.setEnabled(false);
        buttonsPanel.add(clearButton);

        // Add panels to display.
        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(addPanel, BorderLayout.NORTH);
        getContentPane().add(downloadsPanel, BorderLayout.CENTER);
        getContentPane().add(buttonsPanel, BorderLayout.SOUTH);
    }

    // Exit this program.
    private void actionExit() {
        System.exit(0);
    }

    // Add a new download.
    private void actionAdd() {
        URL verifiedUrl = verifyUrl(addTextField.getText());
        if (verifiedUrl != null) {
            tableModel.addDownload(new Download(verifiedUrl));
            addTextField.setText(""); // reset add text field
        } else {
            JOptionPane.showMessageDialog(this,
                    "Invalid Download URL", "Error",
                    JOptionPane.ERROR_MESSAGE);
        }
    }

    // Verify download URL.
    private URL verifyUrl(String url) {
        // Only allow HTTP URLs.
        if (!url.toLowerCase().startsWith("http:// (http://)"))
            return null;

        // Verify format of URL.
        URL verifiedUrl = null;
        try {
            verifiedUrl = new URL(url);
        } catch (Exception e) {
            return null;
        }

        // Make sure URL specifies a file.
        if (verifiedUrl.getFile().length() < 2)
            return null;

        return verifiedUrl;
    }

    // Called when table row selection changes.
    private void tableSelectionChanged() {
    /* Unregister from receiving notifications
       from the last selected download. */
        if (selectedDownload != null)
            selectedDownload.deleteObserver(DownloadManager.this);

    /* If not in the middle of clearing a download,
       set the selected download and register to
       receive notifications from it. */
        if (!clearing) {
            selectedDownload =
                    tableModel.getDownload(table.getSelectedRow());
            selectedDownload.addObserver(DownloadManager.this);
            updateButtons();
        }
    }

    // Pause the selected download.
    private void actionPause() {
        selectedDownload.pause();
        updateButtons();
    }

    // Resume the selected download.
    private void actionResume() {
        selectedDownload.resume();
        updateButtons();
    }

    // Cancel the selected download.
    private void actionCancel() {
        selectedDownload.cancel();
        updateButtons();
    }

    // Clear the selected download.
    private void actionClear() {
        clearing = true;
        tableModel.clearDownload(table.getSelectedRow());
        clearing = false;
        selectedDownload = null;
        updateButtons();
    }

  /* Update each button's state based off of the
     currently selected download's status. */
    private void updateButtons() {
        if (selectedDownload != null) {
            int status = selectedDownload.getStatus();
            switch (status) {
                case Download.DOWNLOADING:
                    pauseButton.setEnabled(true);
                    resumeButton.setEnabled(false);
                    cancelButton.setEnabled(true);
                    clearButton.setEnabled(false);
                    break;
                case Download.PAUSED:
                    pauseButton.setEnabled(false);
                    resumeButton.setEnabled(true);
                    cancelButton.setEnabled(true);
                    clearButton.setEnabled(false);
                    break;
                case Download.ERROR:
                    pauseButton.setEnabled(false);
                    resumeButton.setEnabled(true);
                    cancelButton.setEnabled(false);
                    clearButton.setEnabled(true);
                    break;
                default: // COMPLETE or CANCELLED
                    pauseButton.setEnabled(false);
```

```java
          resumeButton.setEnabled(false);
          cancelButton.setEnabled(false);
          clearButton.setEnabled(true);
      }
    } else {
      // No download is selected in table.
      pauseButton.setEnabled(false);
      resumeButton.setEnabled(false);
      cancelButton.setEnabled(false);
      clearButton.setEnabled(false);
    }
  }

 /* Update is called when a Download notifies its
   observers of any changes. */
  public void update(Observable o, Object arg) {
    // Update buttons if the selected download has changed.
    if (selectedDownload != null && selectedDownload.equals(o))
      updateButtons();
  }

  // Run the Download Manager.
  public static void main(String[] args) {
    DownloadManager manager = new DownloadManager();
    manager.show();
  }
}
```

DownloadTableModel.java

```java
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;

// This class manages the download table's data.
class DownloadsTableModel extends AbstractTableModel
        implements Observer {

    // These are the names for the table's columns.
    private static final String[] columnNames = {"URL", "Size",
    "Progress", "Status"};

    // These are the classes for each column's values.
    private static final Class[] columnClasses = {String.class,
    String.class, JProgressBar.class, String.class};

    // The table's list of downloads.
    private ArrayList downloadList = new ArrayList();

    // Add a new download to the table.
    public void addDownload(Download download) {

        // Register to be notified when the download changes.
        download.addObserver(this);

        downloadList.add(download);

        // Fire table row insertion notification to table.
        fireTableRowsInserted(getRowCount() - 1, getRowCount() - 1);
    }

    // Get a download for the specified row.
    public Download getDownload(int row) {
        return (Download) downloadList.get(row);
    }

    // Remove a download from the list.
    public void clearDownload(int row) {
        downloadList.remove(row);

        // Fire table row deletion notification to table.
        fireTableRowsDeleted(row, row);
    }

```

t&p

```java
    // Get table's column count.
    public int getColumnCount() {
        return columnNames.length;
    }

    // Get a column's name.
    public String getColumnName(int col) {
        return columnNames[col];
    }

    // Get a column's class.
    public Class getColumnClass(int col) {
        return columnClasses[col];
    }

    // Get table's row count.
    public int getRowCount() {
        return downloadList.size();
    }

    // Get value for a specific row and column combination.
    public Object getValueAt(int row, int col) {

        Download download = (Download) downloadList.get(row);
        switch (col) {
            case 0: // URL
                return download.getUrl();
            case 1: // Size
                int size = download.getSize();
                return (size == -1) ? "" : Integer.toString(size);
            case 2: // Progress
                return new Float(download.getProgress());
            case 3: // Status
                return Download.STATUSES[download.getStatus()];
        }
        return "";
    }

  /* Update is called when a Download notifies its
     observers of any changes */
    public void update(Observable o, Object arg) {
        int index = downloadList.indexOf(o);

        // Fire table row update notification to table.
        fireTableRowsUpdated(index, index);
    }
}
```

ProgressRenderer.java

```java
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;

// This class renders a JProgressBar in a table cell.
class ProgressRenderer extends JProgressBar
        implements TableCellRenderer {

    // Constructor for ProgressRenderer.
    public ProgressRenderer(int min, int max) {
        super(min, max);
    }

  /* Returns this JProgressBar as the renderer
     for the given table cell. */
    public Component getTableCellRendererComponent(
            JTable table, Object value, boolean isSelected,
            boolean hasFocus, int row, int column) {
        // Set JProgressBar's percent complete value.
        setValue((int) ((Float) value).floatValue());
        return this;
    }
}
```