VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
Winter Semester 2021-22

PROJECT REVIEW 3

on

OPTIMIZED MULTITHREADED IDM IN JAVA

Submitted
By

Kulvir Singh - 19BCE2074
Dhruv Vaidh - 19BCE0817
Silvi Taneja - 19BCE0782

Course Code: CSE4001 Programme: B.Tech

Course Name: Parallel and Distributed Computing(EPJ) Slot: L21+L22

Class: VL2021220502647

Under the guidance of

Prof Vimala Devi K.

VIT, Vellore.

Abstract

Download managers were among the top (includes torrent customers as they are actually download managers also) applications showing a flag promotion in the UI. Many download managers accompany the highlights like video and sound retrieving from well-known websites like YouTube etc., They likewise support site snatching. queue handling is another main element of download administrator. They additionally can pause and resume downloads, and force speed confinements too. This highlight come exceptionally helpful in locales where power failures is an issue. Furthermore, a large portion of the business download chiefs can download following client arranged timetables and download in like manner. A couple of download administrators claims to build the download speed by a factor of ordinarily. Download managers additionally have tight integration with engines. For the most part they do this by introducing an expansion to the client's engine(browser). Download speeding up, otherwise called multipart download, is a term for the strategy utilized by programming, for example, download administrators to download a solitary record by part.

In todays' world, downloading a file is a frequently worked domain where we use sources from the internet to download a chunk of data in various file formats. For most of the download purposes we use the download manager or download client provided by the internet browser itself. These default download managers can be suitable for small downloads however can take a lot of time and resources when it comes to download multiple small or large files at once.

By using a multithreaded download manager system, we can create a solution for time consuming downloads. The idea is to apply a distributed network to parallelize the download process. This will yield better results when it comes to heavy downloads and multiple downloads. This project will be using one of the better known faster programming language JAVA, to create an internet download manager which can optimize the time and resources used for downloading data from the internet and provide a comparative study of the time and resources used by the default download managers of the web browsers.

Problem Statement and Objectives

Problem Statement

The project aims to create an internet download manager which can handle multiple downloads at once. The files being downloaded should have a lesser download time than other internet download managers. The download manager should be able to check the progress of the download, error handling while download is on and give an accurate response message when the download is complete.

Objectives

- Our major deliverable through this project would be to parallelize the download process to decrease the download time for multiple downloads.
- Another deliverable of this project would be to divide the downloads into multiple threads and associate each thread to execute it in parallel to the other existing processes.
- The third objective would be to check if the download is successful and if an error occurs during the download process, proper error handling is done.
- Lastly, our project should have a good user interface to be appealing to the user.

Literature Survey

S No	Paper	Problem and Objectives	Proposed Methodology	Limitations
1	DPRS: A dynamic popularity aware replication strategy with parallel download scheme in cloud environments N.Mansouri, M. Kuchaki, Rafsanjani, M.M.Javidi (2017) Elsevier	In this paper, the authors want to create a system that can fetch data from an outsourced website, download it and display the downloaded contents on the host web page. Their objective is to use cloud computing and combine it with parallel processing.	To fetch the contents missing on the host website, the authors have suggested an algorithm called DPRS which will find the most accurate replica and download it. For downloading, a parallel process runs with the DPRS capturing the metadata and boosting the speed of download.	The DPRS algorithm and download work parallelly which on certain occasions take a lot of time to replicate hampering the download. It is restricted to distributed and cloud based applications.
2	A Parallel File Download System (PFDS) for Smartphone Devices Dhuha Basheer Abdullah; Zeena Abdulgaffar Thannoon (2018) IEEE	Power consumption is a critical concern for smartphone batteries; since batteries have limited capacities therefore it is important to reduce power consumption. In this paper, a file download system named (PFDS) was designed with a new file download method to reduce the download time in smartphone devices; which yields to decrease the battery power consumption.	A file server system was designed to be monitored by a smartphone in real time through the use of a dynamic scheduling algorithm. The model is suggested for a network file server used by smartphone devices to download files. The proposed system consists of three programs. The first runs on a central computer, steering the control between the file servers and the smart device. The seconds run on a file server, and the third runs on the smart device.	The suggested algorithms use parallel processing and distributed computing for performing downloads; however the creation of a file system with multiple servers can be optimized to a single connection server.
3	Optimization	Using the open source	The authors have	CEPH optimization

	of Ceph Reads/Writes Based on Multi-threade d Algorithms Ke Zhan; Ai Hua Piao (2017) IEEE	software file storage platform CEPH, to read and write files quicker thereby optimizing the entire system of CEPH is the main objective of the authors.	incorporated multithreading algorithms to reduce the processing time of the read and write operations on CEPH. They have used the PRODUCER-CONSUM ER algorithm to sustain the software.	uses threads for read and write operations however when it comes to download, there is a continuous stream of metadata hence an overhead is speculated which will complicate the optimization proposed in the paper.
4	Peer Assisted Parallel Downloading System Akshat Shetty, Simran Mhatre, Nisheet Sinvhal and Kailas K. Devadkar (2019) IEEE	This paper presents a peer-assisted parallel downloading system which uses the concept of segmented file transfer to decrease the download time and also aims to be truly decentralized. The proposed system has peers that can offer high data transfer rates and are in close proximity to the peers interested in downloading the file, themselves download chunks of the file and stream it to the requestor. This allows the utilization of unused bandwidth while reducing the download time.	This paper presents a peer-assisted parallel downloading system which uses the concept of segmented file transfer to decrease the download time and also aims to be truly decentralised.The paper proposes a scoring based completely decentralized peer-assisted P2P system which penalizes free riding by increasing the probability of those peers being used as assistants. Thus bringing better balance to the network.	The system can be further developed to handle the reallocation of chunks from one region to another to prevent excess duplication within a small region. Realtime adaptiveness of the scoring system needs to be developed to handle and penalize assistants for disconnecting during the assistance process.
5	Research and Application on Optimization of Multi-Thread Download Technology for Enhanced Search	With the growth of huge amounts of data on the Internet, the data collection must achieve real-time, which requires completing download of data in the possible shortest time, and as a data transmission technology,	In this paper, we do the research on optimization technologies which include multithreaded download based on P2SP, task scheduling based on MapReduce and download based on the protocol adaptation, designed to	The main performance index of file download system is quantity of concurrent task processing and capacity of concurrent merging file, when the download task strength gradually

	Engine Ya Juan Sun, Hong Lin, Bao Hui Wang (2013) IEEE	multi-thread download technology to efficiently complete the download tasks will directly affect the quality of the acquisition.	improve enhanced search engine efficiency.	increasing, since in this download system the same file can choose multiple source address to download, the network bandwidth can be used are more extensive, the total download speed are more quickly than other download tools.

Proposed Methodology

Algorithm and Analysis

- A thread is created for IDM which creates a thread for Download manager.
- Another thread is created for ProgressRender which monitors the progress of the download.
- When download is complete, a new action is executed in the main thread which stops the 2 running threads and executes the Download thread.
- Download class is executed with the help of a thread which displays the final download complete screen which displays the downloaded file. We have followed Object-Oriented approach to build this project in JAVA. We have divided our tasks into sub-modules, each module performing some specific functionality. These modules work together and download a file
- By splitting the download process into multiple blocks and assigning a thread to each block which opens up a connection with the server and finally when all threads are finished, all threads are joined to get the final downloaded file.

In this project, a multithreaded download handler would be created which can compete with internet browser based download managers in terms of throughput and time.

To achieve this optimisation we will incorporate the following system:-

1. **Download Manager Thread**

When the application is started, a thread will be created which will act as the starting point for the whole download files process.
This thread will allow a URL or URI based file link to be set up as a stream of data and get connected to the Download Manager. Once this thread is started, it will trigger other lateral threads to run which will be discussed below.
Similarly, when multiple files are downloaded, multiple threads are created which will contain the same instances, member methods and member data as the Download Manager Thread.

File handles following tasks:
a) Creates map(a type of list) of threads.
b) declares current state of the download as one of the following: PAUSED, DOWNLOADING or CANCELED.
c) Taking the url and type conversion to string, fetching the size of the file.

d) Calculating the progress (%) using the formula (Downloaded / FileSize) * 100.
e) Set the state of thedownload.
f) Start or resume the download.
g) Increase the downloaded size.
h) Set the state has changed and notify the observers.
i) Create thread to download a part of file.
j) See whether the thread is finished or not, wait for the thread to finish.

2. Progress Render Thread

As explained in the download manager thread, the start of the download stream triggers various threads. One such thread is the Progress Render Thread.
For each download stream, a separate thread will be created.
This thread is associated with calculating vital and necessary information related to the download such as file size, speed, time left for download and other features related to an internet download manager.

File handles following tasks:
a) Get and set the maximum number of connections possible per download(This isessentially the
serviceprovided by the downloading website).
b) Get the downloader object in a list(the object from previous class).
c) Get the download list.
d) Verify if the URL is valid.

3. Controller

This is a class which would extend the Download Manager Thread and there-by will provide the functionality of status control.
On completion of the download stream, this class would be responsible for terminating the active threads associated with the respective download stream.
On termination of the threads, it would also kick start a graphical user interface with the help of an external process. This would display the necessary completion message.
The controller would also be responsible for checking for errors and blocking the download if there is any exception caught.

File handles following tasks:
a) Create an instance of DownloadTableModel.
b) Set up the table.
c) Allow only one row at a time to be selected.

d) Set up progress bar as progress renderer for progress column.
e) Set table's row height large enough to fit progress bar.
f) Create button, text fields, scrollpane and table.
g)Set the text for title bar, for jbnAdd button set the text to "Add Download", for jbnPausebutton set the text to "Pause", for jbnRemove button set the text to "Remove", for jbnRemove button set the text to "Remove", for jbnResume button set the text to "Resume", for jbnExit button set the text to "Exit".
h) Create a layout and add these components to the layout specifying the dimensions of every component.
i) Assign the function to the button.
j) If buttons are pressed then change the state of the button.

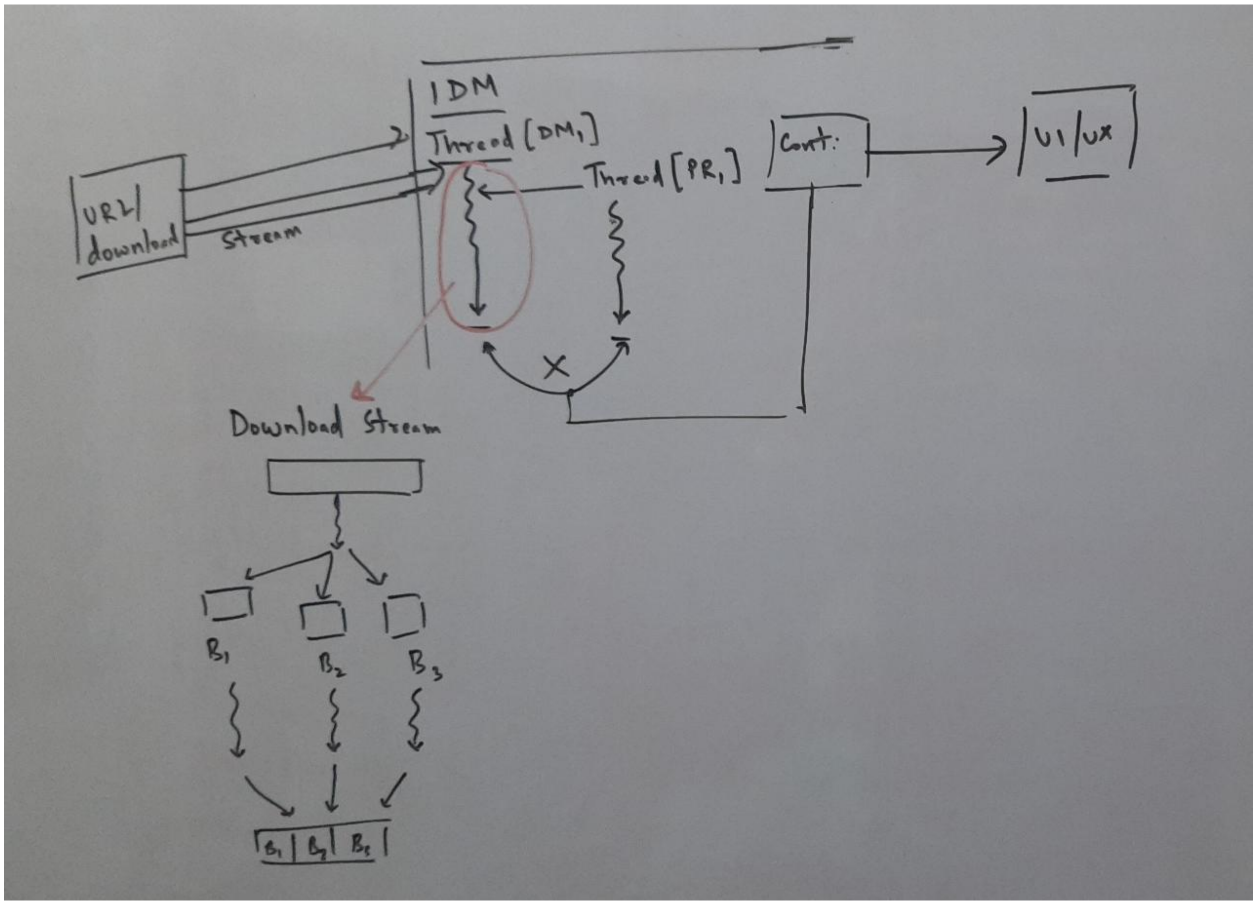
4. Download Stream

For each download link, a download stream is associated. This stream is provided by the download provider and is divided into various blocks. The download stream that will be fetched will try to distribute the entire stream in the maximum number of blocks provided by the download stream to minimize the time complexity of the download manager. By splitting the download process into multiple blocks and assigning a thread to each block which opens up a connection with the server and finally when all threads are finished, all threads are joined to get the final downloaded file.

This class manages the download table data:
a) Initializes various components of the progress check function
b) Reflects the progress of download by showing percentage of file downloaded.
c) Invokes startThread(IDM) to create and stop new thread for download.
d) JSwing is used to develop the GUI and various functions like JPanel,JFrame,JButton are used fordevelopment.

5. UI/UX

A comprehensive User Interface will be created which will display the necessary details and useful information to the user using our IDM. This would provide encapsulation to the project as only the necessary information is visible to the user while the complex backend processes and codes will remain hidden.



Areas of Parallelism

The download manager problem is divided broadly into sections of:

1. Checking the progress of download
2. Executing the download
3. Giving feedback when the download is complete
4. Separate threads have been allocated for these actions and hence they have been executed parallelly.

Implementation Details along with screenshots

As mentioned previously, the various modules have been implemented and a certain set of results, observations and conclusions have been received.

The entire code-base is spread out across 4 files :
Download.java
DownloadManager.java
DownloadTableModel.java
ProgressRenderer.java

Progress Renderer.java

This file contains the code which renders the progress bar for the download. It compiles the download stream that is completed and yet to be completed. It shows the progress left and displays accordingly. The file uses the JProgressBar class to display the same.

```
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;
    }
}
```

DownloadTableModel.java

This file contains the code which renders the entire interface of our application. It compiles to give the entire UI/UX of the project. It uses an abstract model of a download table provided by javax.swing library. The table is further customized to work according to our requirements.
A addDownload method is used to register a new download in the application. A getDownload is used to get download data from the specified row. A clearDownload method is used to remove a download from the manager. Other simplistic functions are created to get the meta data and statistics from the table for the manager.
A switch case is used to manage all the functions present in this file.

```
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 in MB",
"Progress", "Speed in KB/s",
    "Avg Speed in KB/s", "Elapsed Time", "Remaing Time" ,"Status"};

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

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

    // 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 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);
    }

    // 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 = downloadList.get(row);
        switch (col) {
            case 0: // URL
                return download.getUrl();
            case 1: // Size
                long size = download.getSize();
                return (size == -1) ? "" :
Float.toString((float)size/1048576);
            case 2: // Progress
                return new Float(download.getProgress());
            case 3: //Speed
                return download.getSpeed();
            case 4: //Avg Speed
                return download.getAvgSpeed();
            case 5: //Elapsed Time
                return download.getElapsedTime();
            case 6: //Remaining Time
                return download.getRemainingTime();
            case 7: // 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);
    }
}
```

Download.java

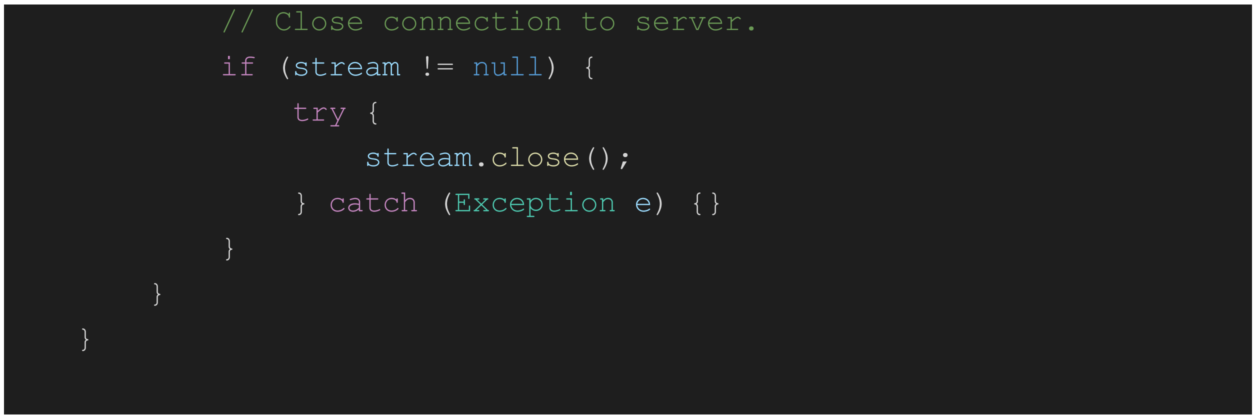
We implement the Runnable method for the download thread in this method. We create a buffer of the size of 1024 bytes, and a status array for holding text to be displayed. We create status code variables for proper handling of the downloads.

We create a Download constructor that takes the url as a parameter and invokes the download method which begins the download. We create a getURL method to get the download URL of the download link. We create a method to get the download size. We create a method to get the download speed. We create a method to fetch the average download speed. We create a method to get the elapsed time as well as the remaining download time. We then need to create a method to provide proper formatting for the time so that it will be in user understandable terms. We create a method to get progress and status of the download. We create a method to pause the download thread, resume the download thread, cancel the download thread. We also provide an error method to catch errors and finally use the download method to invoke the multithreading concepts.

```
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();
          }
          // used to update speed at regular intervals
          int i=0;
          // Open file and seek to the end of it.
          file = new RandomAccessFile(getFileName(url), "rw");
          file.seek(downloaded);

          stream = connection.getInputStream();
          initTime = System.nanoTime();
          while (status == DOWNLOADING) {
      /* Size buffer according to how much of the
         file is left to download. */
              if(i==0)
              {   startTime = System.nanoTime();
                  readSinceStart = 0;
              }
              byte buffer[];
              if (size - downloaded > MAX_BUFFER_SIZE) {
                  buffer = new byte[MAX_BUFFER_SIZE];
              } else {
                  buffer = new byte[(int)(size - downloaded)];
              }
              // Read from server into buffer.
              int read = stream.read(buffer);
              if (read == -1)
```

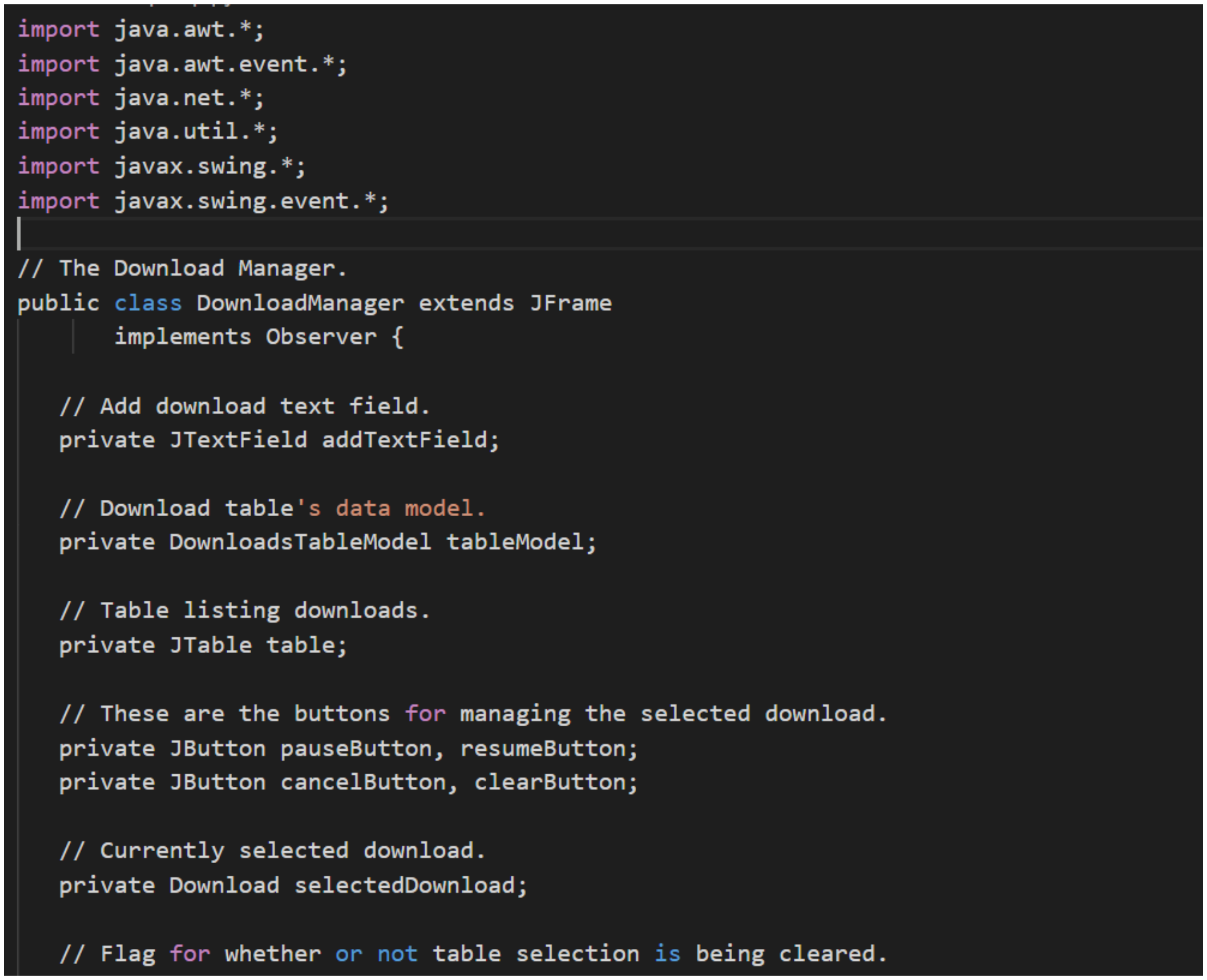
```
                    break;
                // Write buffer to file.
                file.write(buffer, 0, read);
                downloaded += read;
                readSinceStart+=read;
                //update speed
                i++;
                if(i>=50)
                {
speed=(readSinceStart*976562.5f)/(System.nanoTime()-startTime);
                    if(speed>0)
remainingTime=(long)((size-downloaded)/(speed*1024));
                    else remainingTime=-1;

elapsedTime=prevElapsedTime+(System.nanoTime()-initTime);
                    avgSpeed=(downloaded*976562.5f)/elapsedTime;
                    i=0;
                }
                stateChanged();
            }

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



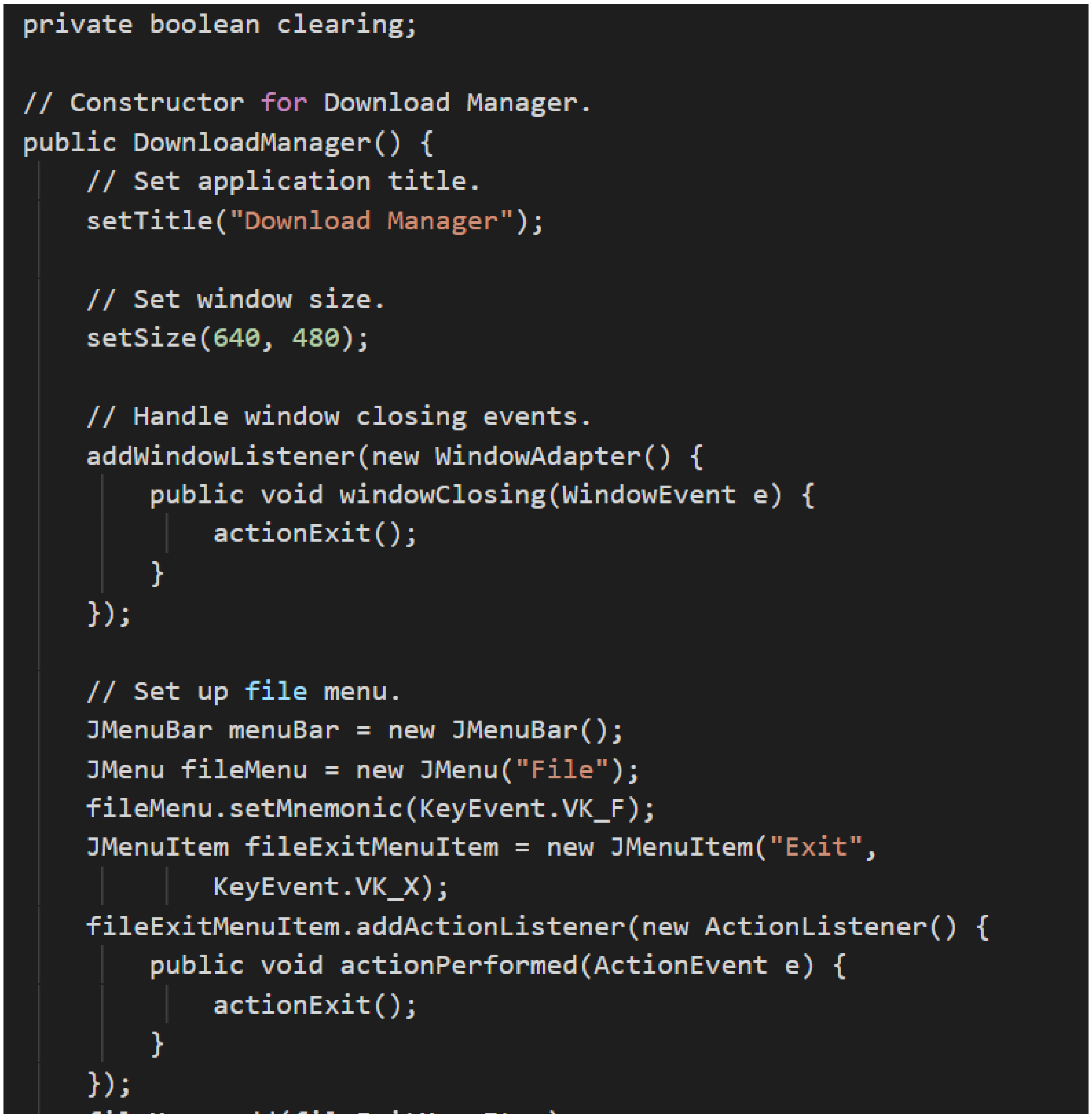
```java
            // Close connection to server.
            if (stream != null) {
                try {
                    stream.close();
                } catch (Exception e) {}
            }
        }
    }
```

DownloadManager.java

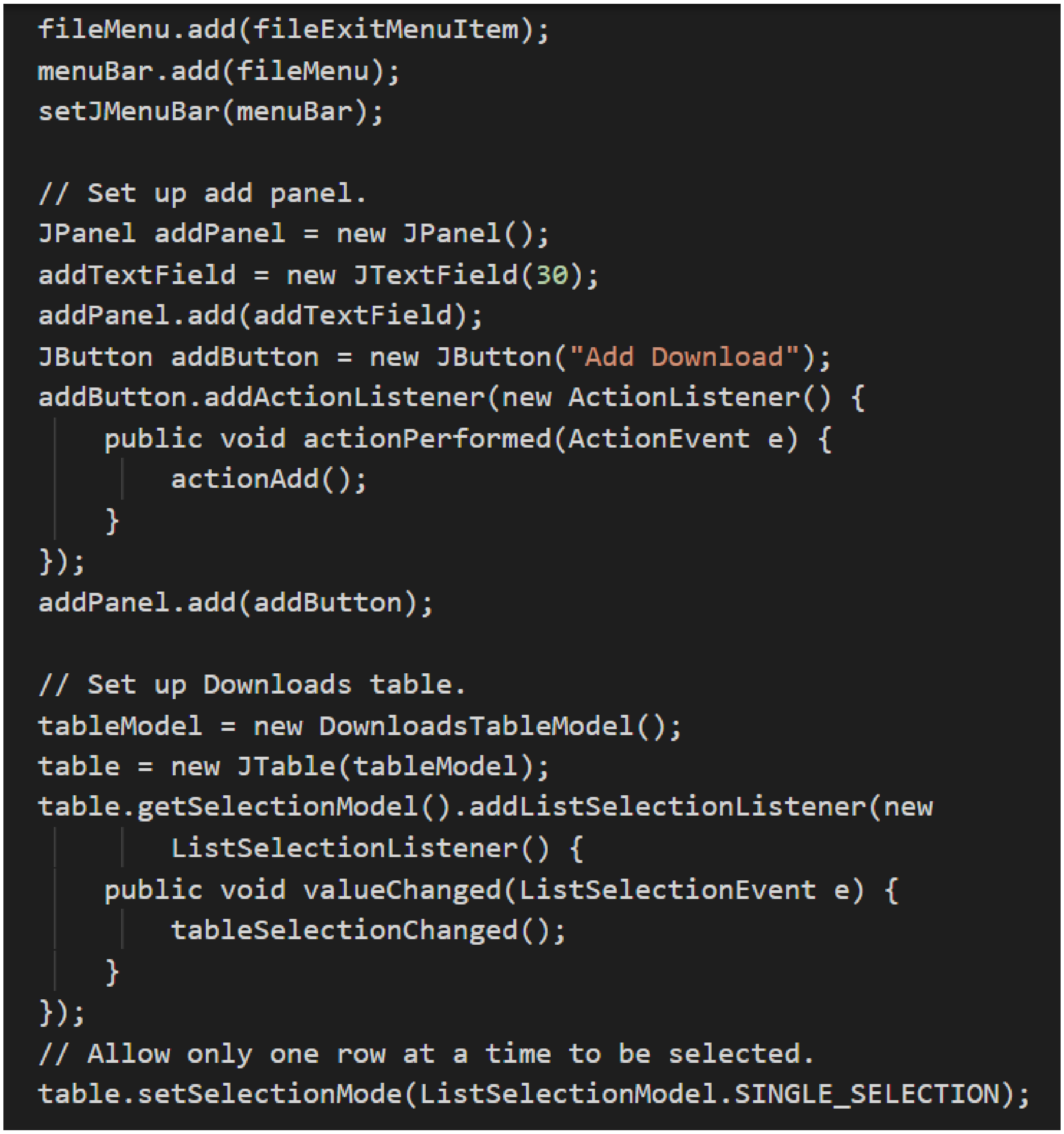
This section consists of the code which will link the download module with the user interface and the code files will be able to communicate with each other using this module.



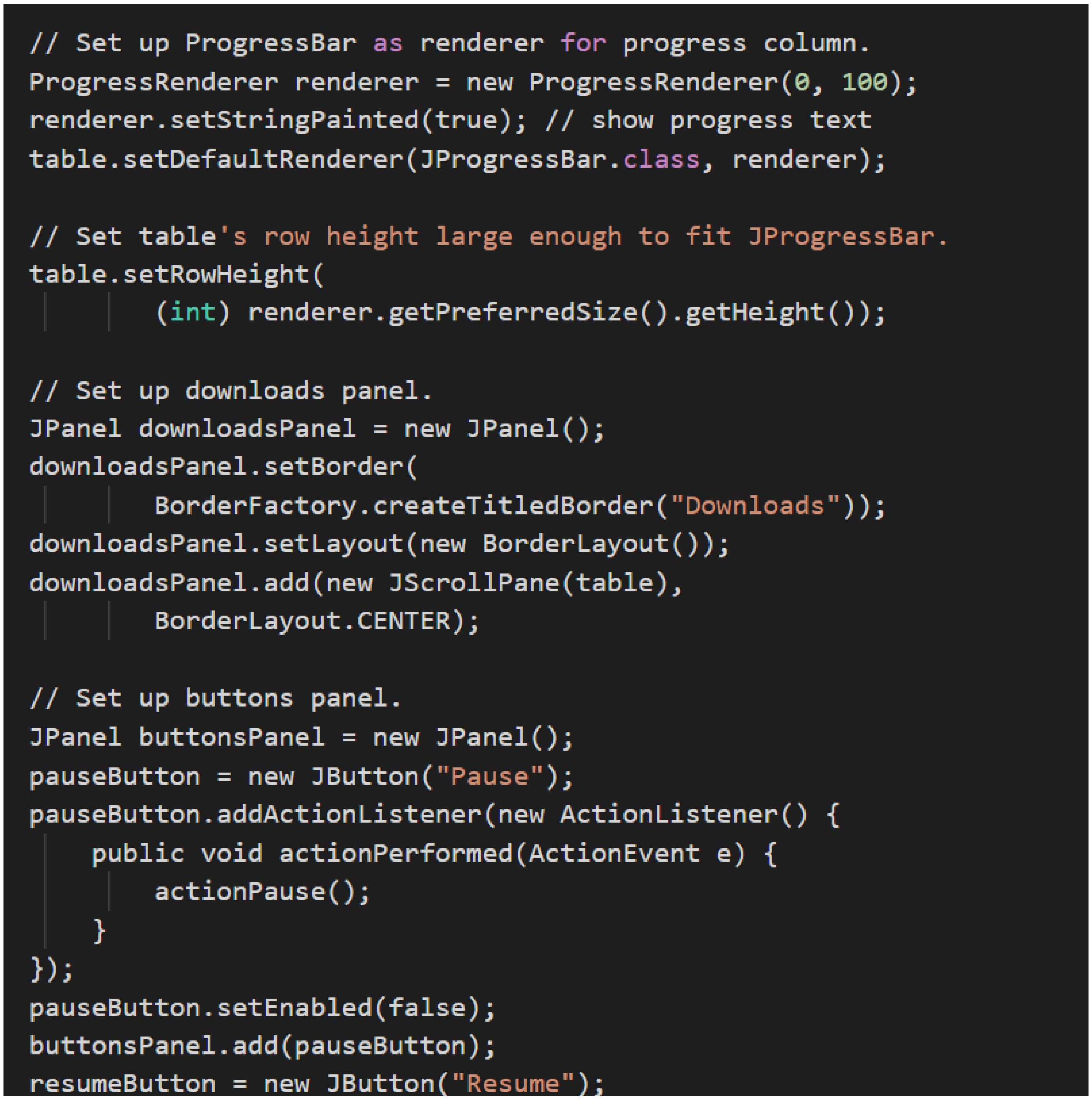
```java
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

// 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.
```



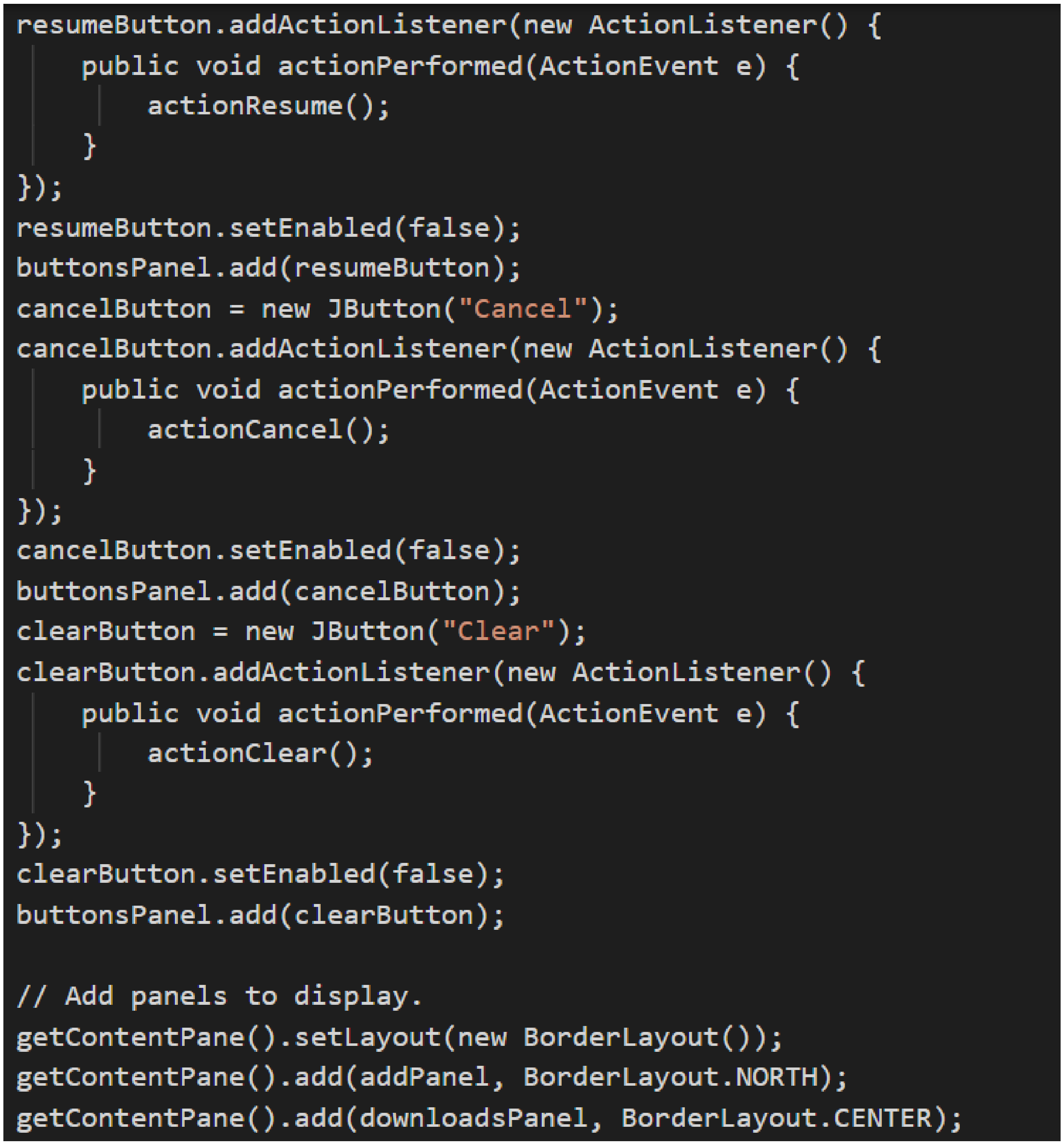
```java
private boolean clearing;

// 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();
        }
    });
```



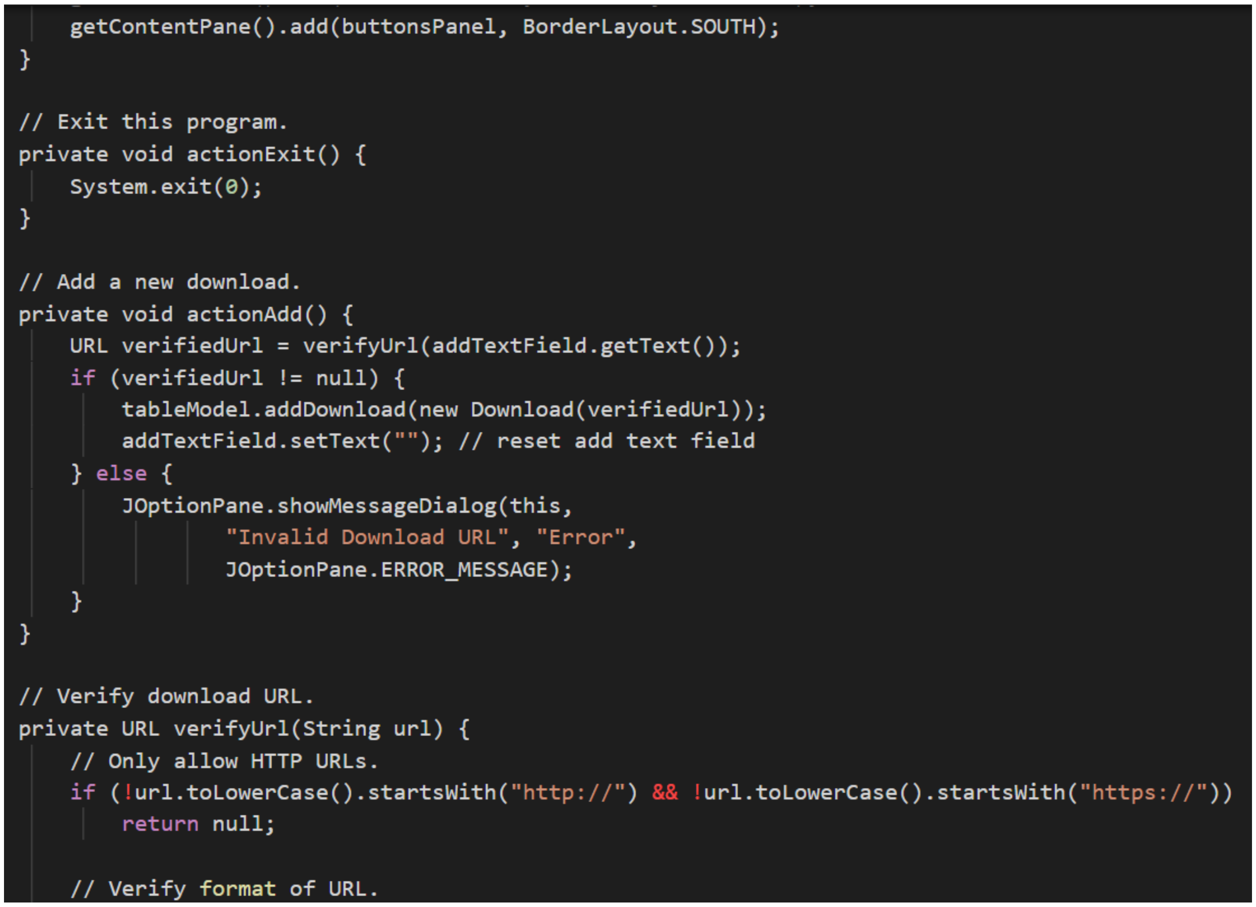
```java
fileMenu.add(fileExitMenuItem);
menuBar.add(fileMenu);
setJMenuBar(menuBar);

// 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);
```



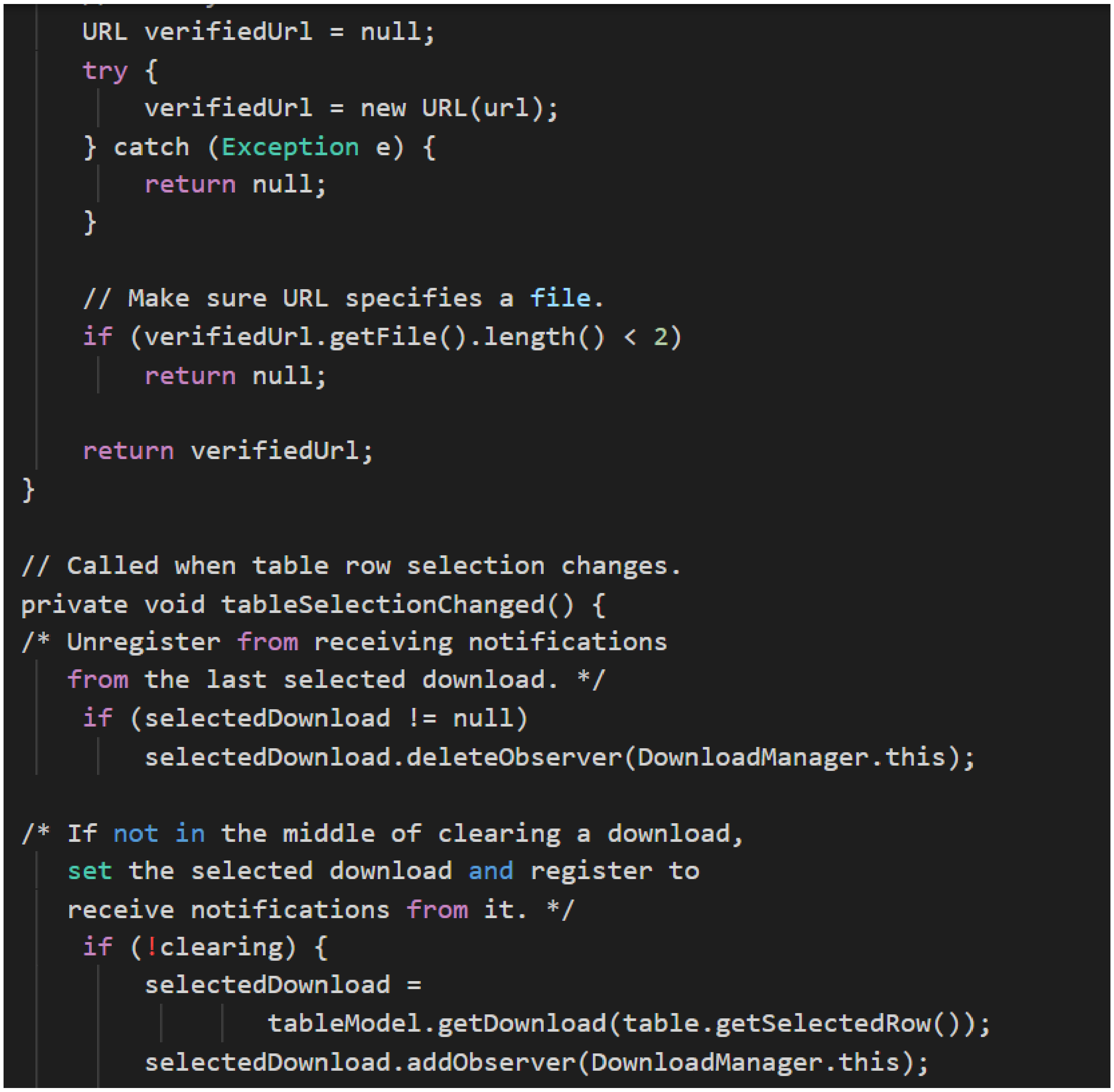
```java
// Set up ProgressBar as renderer for progress column.
ProgressRenderer renderer = new ProgressRenderer(0, 100);
renderer.setStringPainted(true); // show progress text
table.setDefaultRenderer(JProgressBar.class, renderer);

// 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");
```



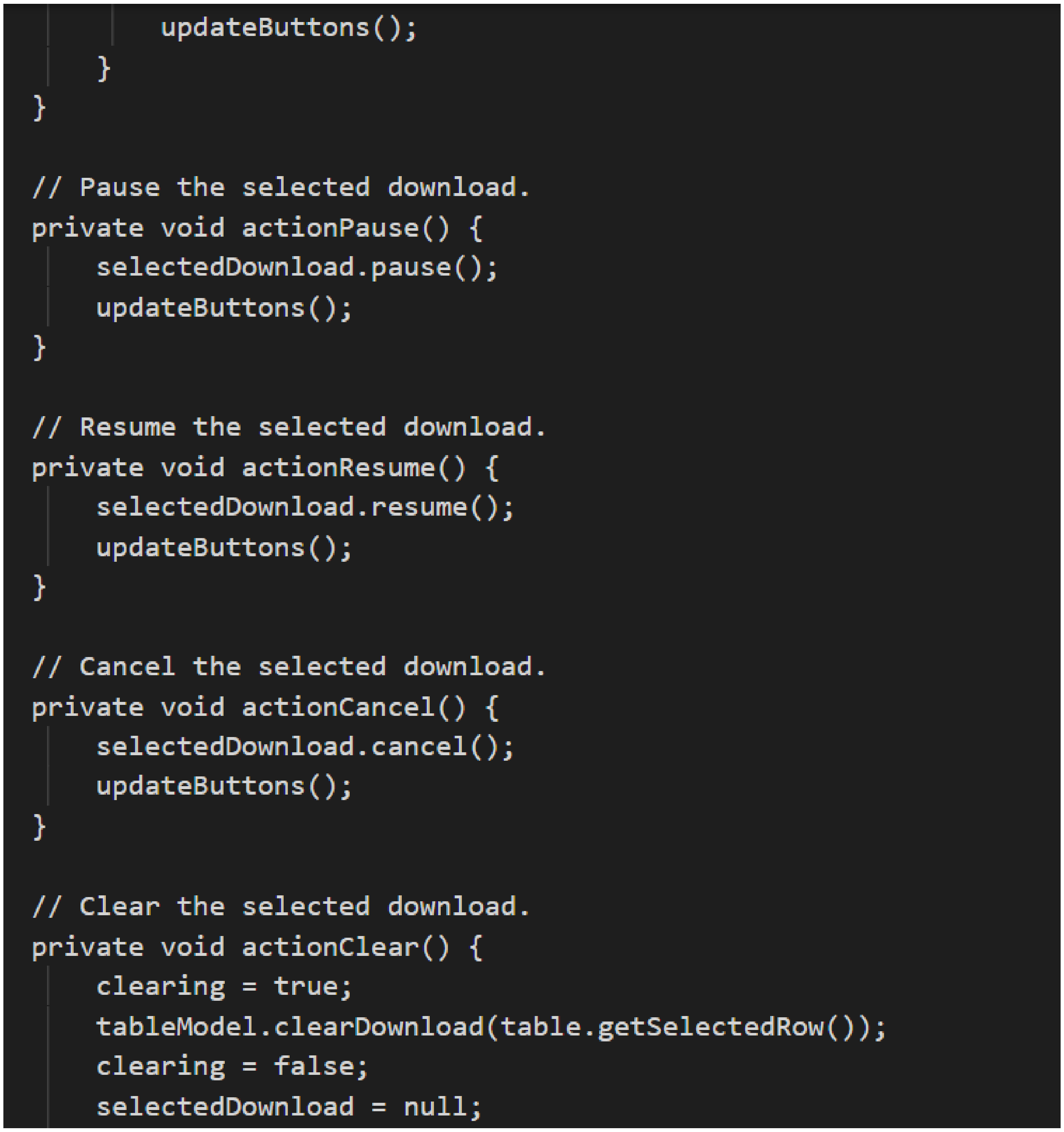
```java
resumeButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        actionResume();
    }
});
resumeButton.setEnabled(false);
buttonsPanel.add(resumeButton);
cancelButton = new JButton("Cancel");
cancelButton.addActionListener(new ActionListener() {
    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);
```



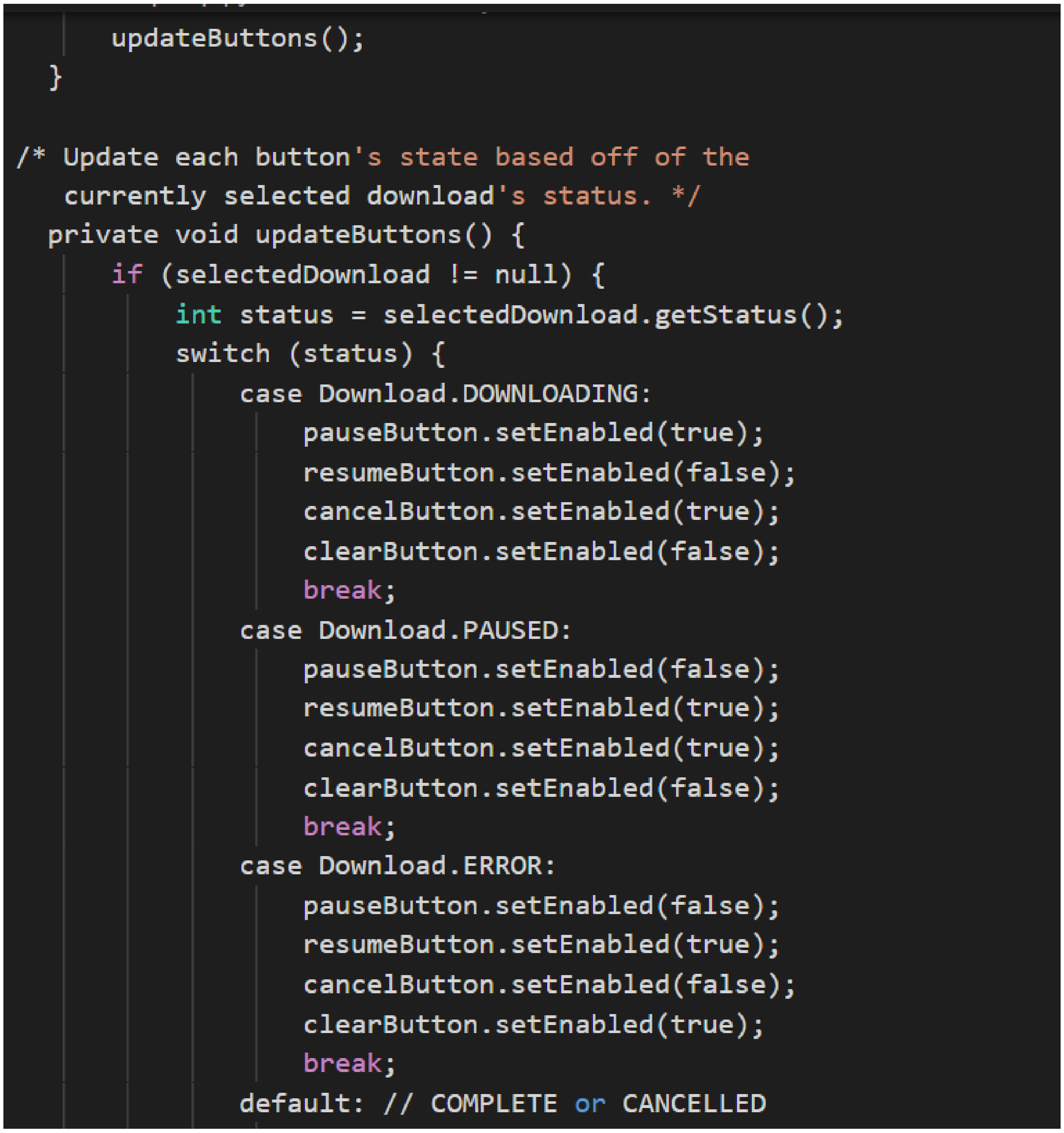
```
    getContentPane().add(buttonsPanel, BorderLayout.SOUTH);
}

// 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://") && !url.toLowerCase().startsWith("https://"))
        return null;

    // Verify format of URL.
```



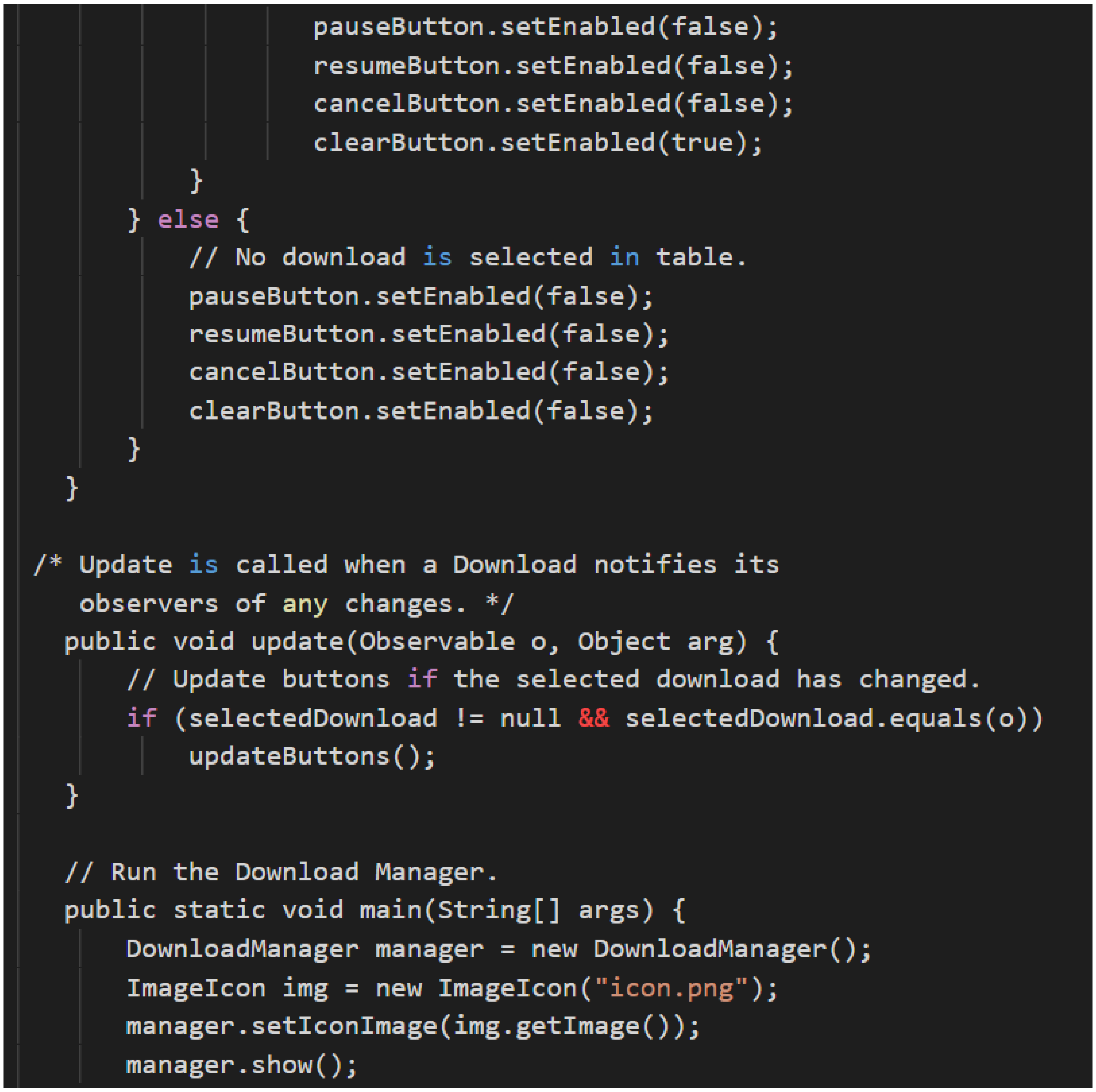
```
    URL verifiedUrl = null;
    try {
        verifiedUrl = new URL(url);
    } catch (Exception e) {
        return null;
    }

    // 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);
```



```java
        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;
```



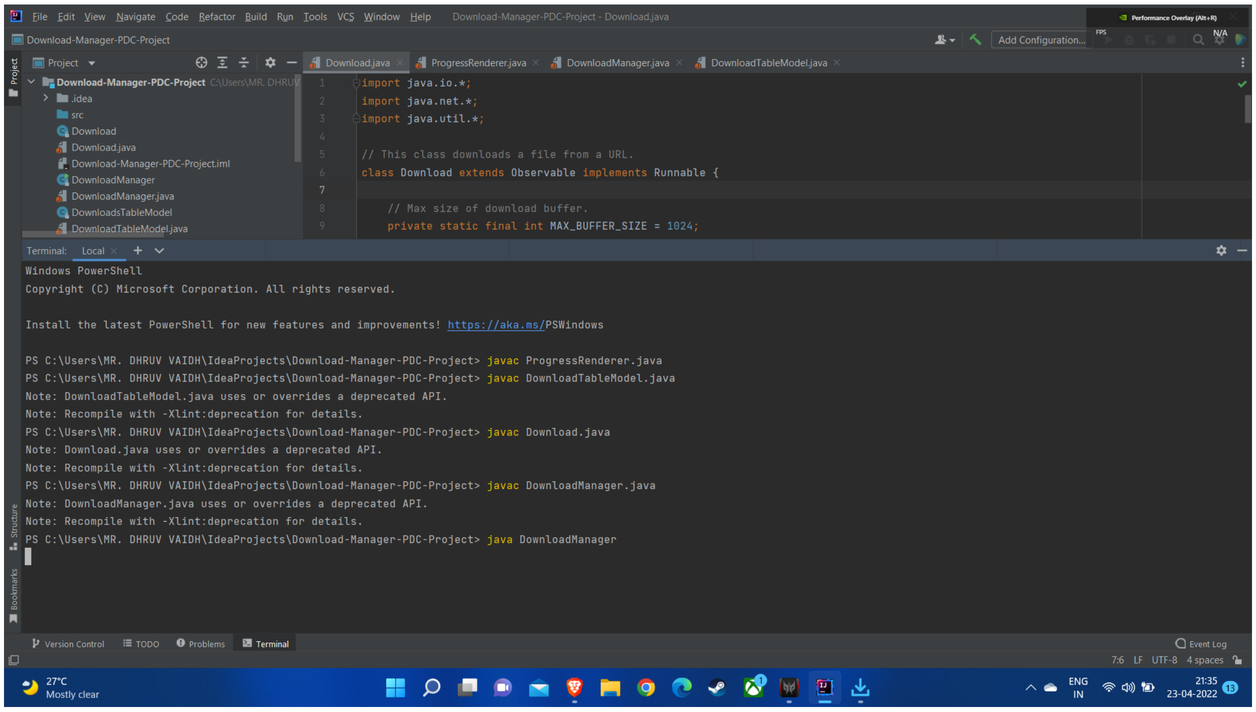
```
        updateButtons();
    }

/* 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);
                  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();
      ImageIcon img = new ImageIcon("icon.png");
      manager.setIconImage(img.getImage());
      manager.show();
```

```
  }
}
```

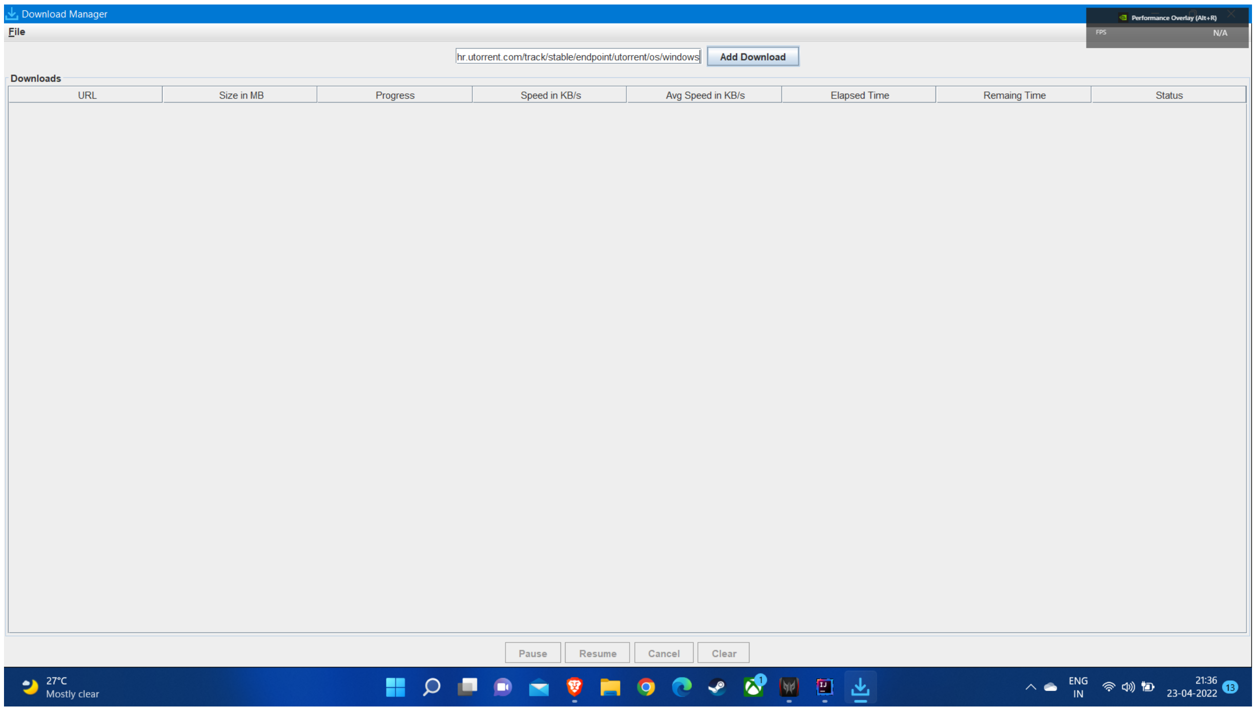
Results



```
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;
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\MR. DHRUV VAIDH\IdeaProjects\Download-Manager-PDC-Project> javac ProgressRenderer.java
PS C:\Users\MR. DHRUV VAIDH\IdeaProjects\Download-Manager-PDC-Project> javac DownloadTableModel.java
Note: DownloadTableModel.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
PS C:\Users\MR. DHRUV VAIDH\IdeaProjects\Download-Manager-PDC-Project> javac Download.java
Note: Download.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
PS C:\Users\MR. DHRUV VAIDH\IdeaProjects\Download-Manager-PDC-Project> javac DownloadManager.java
Note: DownloadManager.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
PS C:\Users\MR. DHRUV VAIDH\IdeaProjects\Download-Manager-PDC-Project> java DownloadManager
```

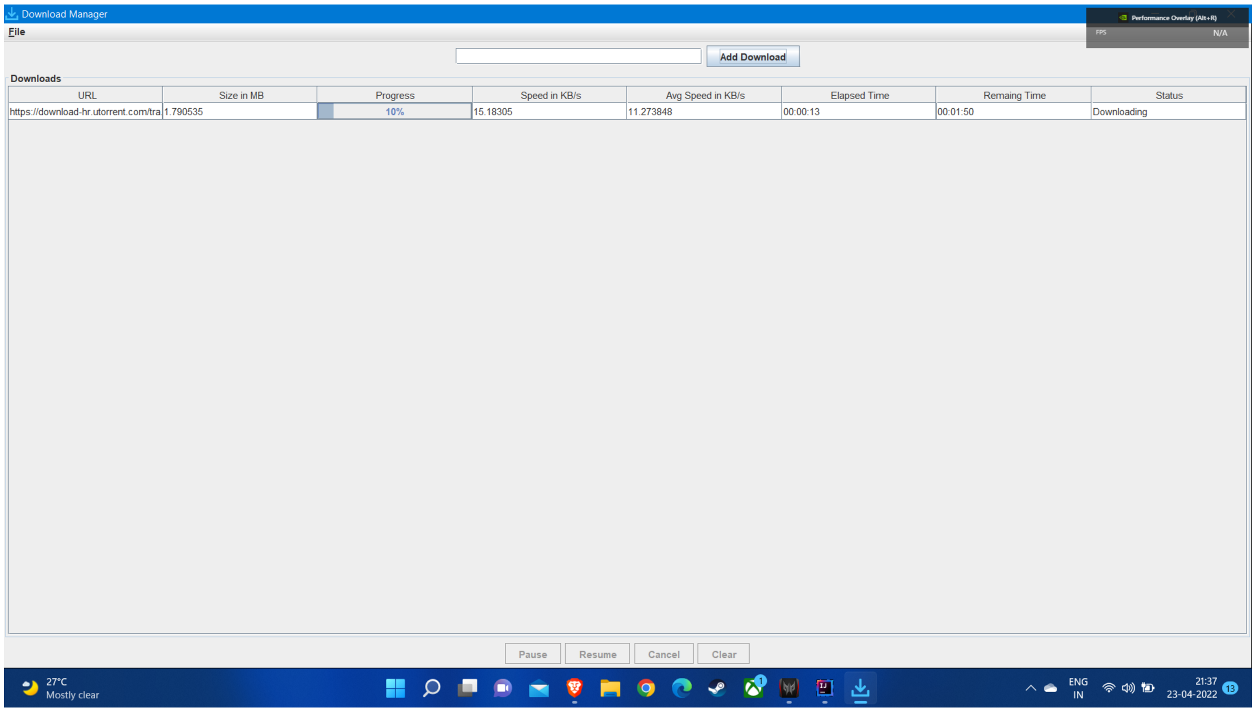
For running the download manager for the first time, we need to compile all the following files in java using the java compile command. After the compilation we can run the manager.



```
hr.utorrent.com/track/stable/endpoint/utorrent/os/windows
```

Downloads

URL	Size in MB	Progress	Speed in KB/s	Avg Speed in KB/s	Elapsed Time	Remaing Time	Status



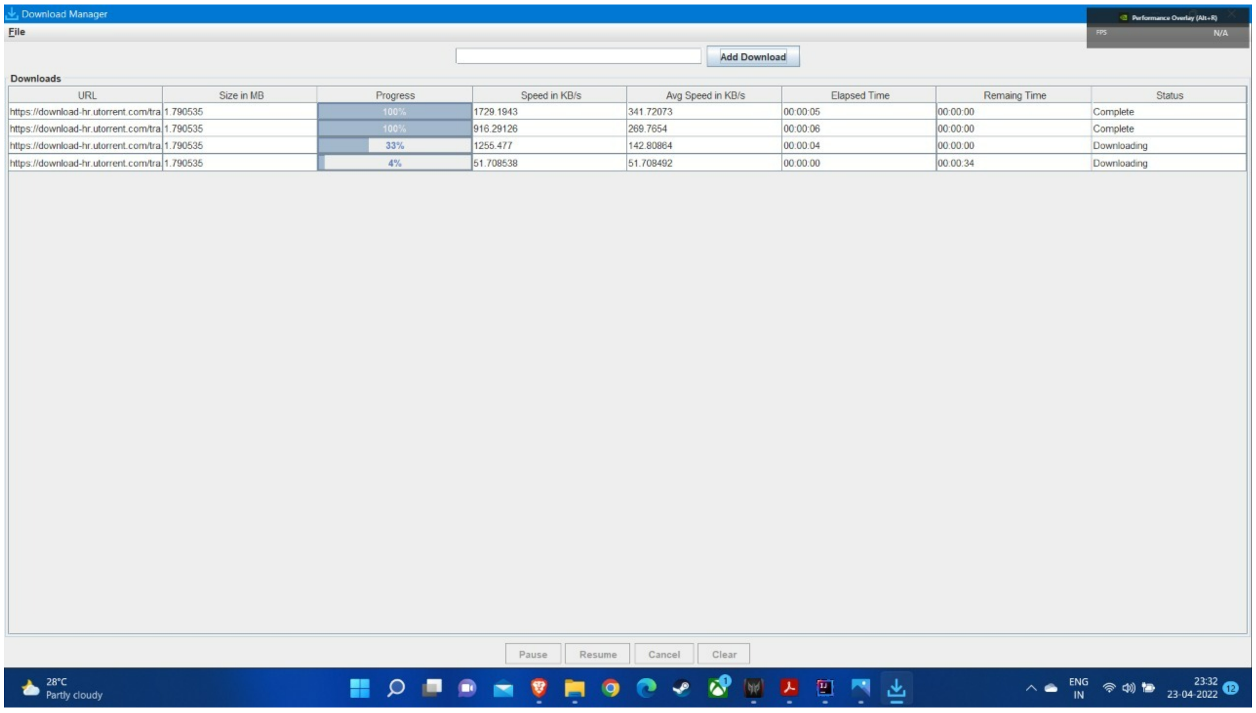
Downloads

URL	Size in MB	Progress	Speed in KB/s	Avg Speed in KB/s	Elapsed Time	Remaing Time	Status
https://download-hr.utorrent.com/tra	1.790535	10%	15.18305	11.273848	00:00:13	00:01:50	Downloading

As seen in the above screenshot, on compilation, the window opens which contains our download manager. The download manager contains the design as specified in the code files and has the url input option which initiates the download.

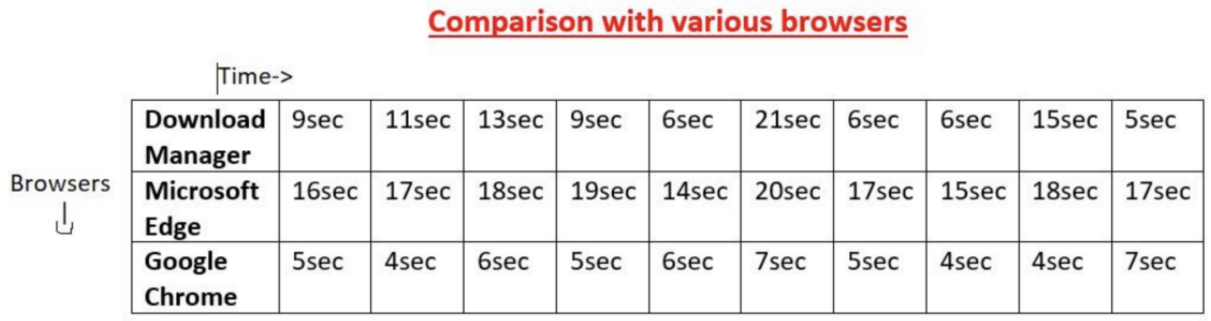
Conclusion and Result Analysis

When using multiple threads, an increase in performance is to be expected, but it is bound to its max potential. As shown in the result of the experiment of thread threshold, the max performance potential of the threads are bound to the number of physical cores available, which in the work is four cores, four threads. Number of threads does not outperform four threads in this case, even if the parallelity is increased. Here, creating threads increases the overall exe- cution time, but in some instances a sequential approach can be better than a parallel. When an input is very small, the thread overhead takes more time than solving the actual problem. However, when the input grows bigger, the overhead is compensated by the performance of the executing thread, such that the overall execution time is faster. When the overhead is compensated, the performance increases. At this point, it is shown that multithreading is faster than its sequen- tial counterpart and its parallel counterpart, if the thread is increased within the bound of physical available threads. Thus in a performance aspect, using threads are only suitable when the data is big enough such that the thread overhead is compensated for and that the number of threads used are within the number of physical available threads.



Downloads

URL	Size in MB	Progress	Speed in KB/s	Avg Speed in KB/s	Elapsed Time	Remaing Time	Status
https://download-hr.utorrent.com/tra	1.790535	100%	1729.1943	341.72073	00:00:05	00:00:00	Complete
https://download-hr.utorrent.com/tra	1.790535	100%	916.29126	269.7654	00:00:06	00:00:00	Complete
https://download-hr.utorrent.com/tra	1.790535	33%	1255.477	142.80864	00:00:04	00:00:00	Downloading
https://download-hr.utorrent.com/tra	1.790535	4%	51.708538	51.708492	00:00:00	00:00:34	Downloading



Comparison with various browsers

Time->

Browsers ↓

Download Manager	9sec	11sec	13sec	9sec	6sec	21sec	6sec	6sec	15sec	5sec
Microsoft Edge	16sec	17sec	18sec	19sec	14sec	20sec	17sec	15sec	18sec	17sec
Google Chrome	5sec	4sec	6sec	5sec	6sec	7sec	5sec	4sec	4sec	7sec

Fig: Shows Comparison with other Browsers

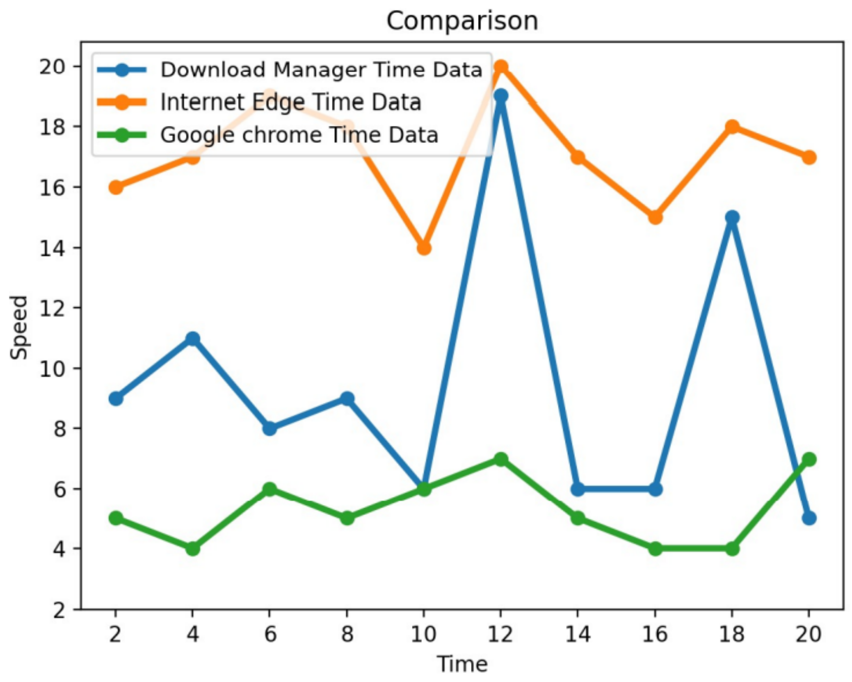


Fig: Graphical Representation of the above Comparison with other Browsers

In these screen shots, we have shown comparison of time taken for downloading a same file on different browsers. We tried downloading multiple times to get better understanding.

Average time for Browsers:

1. Download Manager - 10.1 sec
2. Microsoft Edge - 17.1 sec
3. Google Chrome. - 5.3 sec

Conclusion and Future Work

We have presented a parallel downloading method using multi threading in JAVA. The performance is better than serial downloading as the bandwidth is utilized more efficiently. We have compared the time taken for different download managers and came to a conclusion that multi-threaded downloading is faster.

As a part of the future scope, we would like to add more features to the application in the future like download scheduling, bandwidth allocation etc.

References

1. Sun, Yajuan & Lin, Hong & Wang, Baohui. (2014). Research and Application on Optimization of Multi-Thread Download Technology for Enhanced Search Engine. Advanced Materials Research. 756-759. 10.2991/iccia.2012.365.

2. C. Gkantsidis, M. Ammar and E. Zegura, "On the effect of large-scale deployment of parallel downloading," Proceedings the Third IEEE Workshop on Internet Applications. WIAPP 2003, 2003, pp. 79-89, doi: 10.1109/WIAPP.2003.1210291

3. https://luugiathuy.com/2011/03/download-manager-java/

4. https://www.geeksforgeeks.org/java-thread-priority-multithreading/?ref=rp

5. C. Gkantsidis, M. Ammar and E. Zegura. "Proceedings the Third IEEE Workshop on Internet Applications.WIAPP 2003”.

6. S.G.M. Koo, C. Rosenberg and Dongyan Xu. "The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, 2003. FTDCS 2003. Proceedings.”

7. Zhou Xu, Lu Xianliang, Hou Mengshu and Zhan Chaun. " ACM SIGOPS Operating Systems ReviewHomepage archive Volume 39 Issue 1, January 2005 Pages 63-69 “ 36

8. Jiantao Song, Chaofeng Sha and Hong Zhu. "Distributed Computing Systems, 2004. Proceedings.”

9. Allen Miu and Eugene Shih. "Laboratory of Computer Science Massachusetts Institute of TechnologyCambridge, MA, USA”

10.J. Funasaka, N. Nakawaki and K. Ishida. "Distributed Computing Systems Workshops, 2003. Proceedings.