IDM-JAVA

Download.java

```java
import java.io.InputStream;
import java.io.RandomAccessFile;
import java.net.HttpURLConnection;
import java.net.URL;

// This class downloads a file from a URL.
class Download 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 long size; // size of download in bytes
    private long downloaded; // number of bytes downloaded
    private int status; // current status of download
    private long initTime; //inital time when download started or resumed
    private long startTime; // start time for current bytes
    private long readSinceStart; // number of bytes downloaded since startTime
    private long elapsedTime=0; // elapsed time till now
    private long prevElapsedTime=0; // time elapsed before resuming download
    private long remainingTime=-1; //time remaining to finish download
    private float avgSpeed=0; //average download speed in KB/s
    private float speed=0; //download speed in KB/s
    // 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 long getSize() {
        return size;
    }
    // Get download speed.
    public float getSpeed() {
        return speed;
    }
    // Get average speed
    public float getAvgSpeed() {
        return avgSpeed;
    }
    // Get elapsed time
    public String getElapsedTime() {
        return formatTime(elapsedTime/1000000000);
    }
    // Get remaining time
    public String getRemainingTime() {
        if(remainingTime<0)    return "Unknown";
        else    return formatTime(remainingTime);
    }
    // Format time
    public String formatTime(long time) { //time in seconds
        String s="";
        s+=(String.format("%02d", time/3600))+":";
        time%=3600;
        s+=(String.format("%02d", time/60))+":";
        time%=60;
        s+=String.format("%02d", time);
        return s;
    }
    // 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() {
        prevElapsedTime=elapsedTime;
        status = PAUSED;
    }

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

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

    // Mark this download as having an error.
    private void error() {
        prevElapsedTime=elapsedTime;
        status = ERROR;
        System.out.println("Sorry your file type is not valid:");
    }

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

    // Get file name portion of URL.
    public 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();
            System.out.println("Connected to the server, Response: "
+connection.getResponseMessage());

            // 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;
            }
            // 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;
                }
            }

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

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

}
```

Main.java

```java
import java.net.*;
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
    private static Scanner s;
    public static void main(String[] args) throws Exception {
        System.out.println("               Welcome to IDM - Internet Download 
Manager");
        System.out.println("--------------------------------MENU:------------
------------------------");
        System.out.println("1 : Download a new file");
        System.out.println("2 : Show a list of currently ongoing/paused 
downloads");
        System.out.println("3 : Pause a download");
        System.out.println("4 : Resume a download");
        System.out.println("5 : Cancel a download");
        System.out.println("6 : Get detailed info of a download");
        System.out.println("7 : Exit");
        System.out.println("---------------------------------------------------
-------------------------");

        ArrayList<Download> downloads = new ArrayList<Download>();

        while(true) {
            s = new Scanner(System.in);
            System.out.print("Enter your choice: ");
            int choice = s.nextInt();
            switch(choice) {
            case 1:
                System.out.print("Enter valid download link 
[Ex:jpeg/png/zip/pdf] :");
                s.nextLine();
                String link = s.nextLine();
                Download d = new Download(new URL(link));
                while(d.getProgress()<=0) {}
                System.out.println("Download has started");
                downloads.add(d);
                break;
            case 2:
                if(downloads.size()==0) {
                    System.out.println("There are currently no ongoing 
downloads!");
                    break;
                }
                for(int i = 0;i<downloads.size();i++) {
                    Download temp = downloads.get(i);
                    String status;
                    if(temp.getStatus()==1) status = "PAUSED";
                    else if(temp.getStatus()==0) status = "Downloading";
                    else status = "N/A";
                    if(temp.getStatus()!=2 && temp.getStatus()!=4) {
                        System.out.println(i+1 +" : " +temp.getFileName(new 
URL(temp.getUrl())) +" " +status);
                    }if(temp.getStatus()==2||temp.getStatus()==4) 
downloads.remove(i);
                }
                break;
            case 3:
                System.out.print("Enter the download number to pause: ");
                int num = s.nextInt();
                if(downloads.get(num-1).getStatus()==1) 
System.out.println("The download is already paused!");
                else {
                    downloads.get(num-1).pause();
                    System.out.println("Download " +downloads.get(num-
1).getFileName(new URL(downloads.get(num-1).getUrl())) +" paused");
                }
                break;
            case 4:
                System.out.print("Enter the download number to resume: ");
                num = s.nextInt();
                if(downloads.get(num-1).getStatus()==0) 
System.out.println("The file is already downloading!");
                else {
                    downloads.get(num-1).resume();
                    System.out.println("Download  for " +downloads.get(num-
1).getFileName(new URL(downloads.get(num-1).getUrl())) +" resumed");
                }
                break;
            case 5:
                System.out.print("Enter download number to cancel: ");
                num = s.nextInt();
                downloads.get(num-1).cancel();
                System.out.println("Download  for " +downloads.get(num-
1).getFileName(new URL(downloads.get(num-1).getUrl())) +" cancelled");
                downloads.remove(num-1);
                break;
            case 6:
                System.out.println("Enter download number to get detailed 
info: ");
                num = s.nextInt();
                Download temp = downloads.get(num-1);
```

```java
                String name = temp.getFileName(new URL(temp.getUrl()));
                String elapsedTime = temp.getElapsedTime();
                float progress = temp.getProgress();
                String remainingTime = temp.getRemainingTime();
                float speed = temp.getSpeed();
                long size = temp.getSize();
                String status;
                if(temp.getStatus()==1) status = "PAUSED";
                else if(temp.getStatus()==0) status = "Downloading";
                else if(temp.getStatus()==2) status = "Complete";
                else status = "N/A";

                System.out.println("------------------------------------------
---------------");
                System.out.println("File Name       : " +name);
                System.out.println("File Size       : " +size +" bytes");
                System.out.println("Status          : " +status);
                System.out.println("Progress        : " +progress +" %");
                System.out.println("Elapsed Time    : " +elapsedTime);
                System.out.println("Remaining Time  : " +remainingTime);
                System.out.println("Speed           : " +speed +" KB/S");
                System.out.println("------------------------------------------
---------------");

                break;
            case 7:
                return;
            default:
                System.out.println("Please enter a valid option!");
            }
        }
    }
}
```

Main.java:

Explanation of the `Main` class:

1. **Import Statements:**
   - **import java.net.*; import java.util.ArrayList; import java.util.Scanner;**

- This code imports necessary Java classes for networking (`URL`) and user input (`Scanner`), as well as the `ArrayList` class for managing a list of downloads.

2. **Class Declaration:**

```
public class Main {
```

- The class declaration for the `Main` class.

3. **Main Method:**

```
public static void main(String[] args) throws Exception {
```

- The `main` method is the entry point of the program. It handles user interactions, menu display, and manages the list of downloads.

4. **Menu Display:**

```
System.out.println(" Welcome to IDM - Internet Download Manager"); System.out.println("------
--------------------------MENU:-----------------------------------"); // ... (Displaying menu options)
System.out.println("-----------------------------------------------------------------------");
```

- Displays a welcome message and a menu with options for the user to choose from.

5. **ArrayList Initialization:**

```
ArrayList<Download> downloads = new ArrayList<Download>();
```

- Initializes an `ArrayList` to store instances of the `Download` class.

6. **User Input and Switch Statement:**

```
while (true) { s = new Scanner(System.in); System.out.print("Enter your choice: "); int choice
= s.nextInt(); switch (choice) { // ... (Handling user choices) } }
```

- Enters a loop for continuous user interaction. Reads user input, and uses a `switch` statement to handle different choices.

7. **Option 1 - Download a New File:**

```
case 1: // ... (Prompt user for download link and create a new Download object) break;
```

- Prompts the user for a download link and creates a new `Download` object to handle the download process.

8. **Option 2 - Show Ongoing/Paused Downloads:**

```
case 2: // ... (Displays the list of ongoing and paused downloads) break;
```

- Displays a list of ongoing and paused downloads.

9. **Option 3 - Pause a Download:**

```
case 3: // ... (Pauses a selected download) break;
```

- Pauses a selected download.

10. **Option 4 - Resume a Download:**

```
case 4: // ... (Resumes a selected download) break;
```

- Resumes a selected download.

11. **Option 5 - Cancel a Download:**

```
case 5: // ... (Cancels a selected download) break;
```

- Cancels a selected download.

12. **Option 6 - Get Detailed Info of a Download:**

```
case 6: // ... (Displays detailed information about a selected download) break;
```

- Displays detailed information about a selected download.

13. **Option 7 - Exit:**

```
case 7: return;
```

- Exits the program.

14. **Default Case:**

javaCopy code

```
default: System.out.println("Please enter a valid option!");
```

- Handles cases where the user enters an invalid option.

Download.java:

Explanation of the `Download` class:

1. **Import Statements:**

```
import java.io.InputStream; import java.io.RandomAccessFile; import
java.net.HttpURLConnection; import java.net.URL;
```

- Import statements for handling file I/O, networking, and URL operations.

2. **Class Declaration:**

```
class Download implements Runnable {
```

- The `Download` class implements the `Runnable` interface, indicating that instances of this class can be executed by a thread.

3. **Constants:**

```
private static final int MAX_BUFFER_SIZE = 1024; public static final String STATUSES[] =
{"Downloading", "Paused", "Complete", "Cancelled", "Error"}; 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;
```

- Constants for buffer size and different download statuses.

4. **Instance Variables:**

```
private URL url; private long size; private long downloaded; private int status; private long
initTime; private long startTime; private long readSinceStart; private long elapsedTime =
0; private long prevElapsedTime = 0; private long remainingTime = -1; private float
avgSpeed = 0; private float speed = 0;
```

- Variables to store information about the download, such as URL, size, status, time-related metrics, and download speed.

5. **Constructor:**

```
public Download(URL url) { // ... (Initialization and start the download) }
```

- The constructor initializes the `Download` object with a given URL and starts the download.

6. **Getter Methods:**

```
public String getUrl() { /* ... */ } public long getSize() { /* ... */ } public float getSpeed() { /*
... */ } public float getAvgSpeed() { /* ... */ } public String getElapsedTime() { /* ... */ }
public String getRemainingTime() { /* ... */ } public float getProgress() { /* ... */ } public int
getStatus() { /* ... */ }
```

- Getter methods to retrieve information about the download.

7. **Control Methods:**

javaCopy code

```
public void pause() { /* ... */ } public void resume() { /* ... */ } public void cancel() { /* ... */ } private void error() { /* ... */ } private void download() { /* ... */ }
```

   - Methods to control the state of the download (pause, resume, cancel), handle errors, and initiate the download.

8. **Run Method (Thread Execution):**

```
public void run() { // ... (Download logic) }
```

   - The `run` method contains the main logic for downloading the file. It uses multi-threading to download the file in the background.

9. **File Name Extraction:**

```
public String getFileName(URL url) { /* ... */ }
```

   - Method to extract the filename from the given URL.

10. **Download Logic:**
    - The `run` method contains the logic for opening a connection to the URL, specifying the range of bytes to download, and reading from the server into a buffer. It writes the buffer to a file, updates download metrics, and handles errors. The download progresses until completion, and the status is updated accordingly.