**CST 8284 Assignment 3 – Final**

(due April 22, total of 32 marks)

The goal of this assignment is to add more functionality to the web browser from Assignment 2. Specifically, you must read and write files in order to save settings, and also download files from the internet to your computer. I will give you example code on how to download a file, but you must incorporate it into the browser, along with a progress bar and cancel option.

In this assignment, there will be extra features worth bonus marks. You can use my solution for Assignment 2 as a starting point for this assignment.

There are several parts to this assignment: Browser Interface improvements, Saving Settings, and a Download manager.

***Saving Settings***

Write a function called saveSettings, which will save several pieces of information to a file. Use an ObjectOutput stream to save an ArrayList<String> representing all of your saved URL bookmarks in a file. It should be written as a java object which can then be read back in using an ObjectInput stream.

In a second file, save the height, width, screenX, and screenY coordinates of your browser as a text file. The file format should be name/value pairs like lab 4, only on different lines:

screenX=…… topLeftX

screenY=…… topLeftY

height=…… Stage Height

width=…… Stage Width

downloadDirectory=… where to save downloaded files

homepage=…default home page

You must also create a function called readSettings, which should check if the files you wrote in saveSettings exist, and if they do, open the files and read in the values that you saved.

If the settings text file exists, then read in one line of text at a time, and scan the name in front of the “=” sign. Set the appropriate values for width/height, and screenX/screenY position, download directory, and home page.

Add an onCloseRequest event handler to primaryStage that calls saveSettings whenever you close your window, and also change the Quit MenuItem event handler to also call saveSettings when the user clicks it.

***Download Manager***

The WebEngine has a location property which stores the URL of the page you are viewing. Whenever you click on a link, the engine tries to load the new address, but the browser doesn’t know how to display many different file types: “.exe, .PDF, .ZIP, .DOC, .DOCX, .XLS, .XLSX, .ISO, .IMG, .DMG, .TAR, .TGZ, .JAR”. I have given you code that adds a ChangeListener to the engine.locationProperty() which gets called every time the engine tries to load a new document. Look at the changeListener, and code that tests if the newLocation ends with any of the above file endings. If it does, then you should download that file to your browser’s default Download directory, which you have to add as part of this assignment.

1. I have given you the starting code for a DownloadBar class, which extends HBox. Since DownloadBar is an HBox, you can add GUI Objects which will represent the information needed for the downloading of a file. Create a constructor for the DownloadBar which takes a String, which should be the URL of the new location for the engine. You must check if a file with that same name exists in your download directory and if it does then add “(1)”, or “(2)” at the end until you find a filename that doesn’t exist, like a normal web browser does. You can not overwrite an existing file with your download.
2. Add a Text object as a private instance variable to DownloadBar, which should display the file name that is being downloaded. This file name should be all the characters after the last “/” of the URL. For instance, in: “http://www.adobe.com/files/report.pdf” the file name is “report.pdf”.
3. Add a ProgressBar as a private instance variable to DownloadBar, which should display the progress of the download. The ProgressBar class has a setProgress() function which takes a float from 0 – 1.0, which represents the progress from 0% – 100%.
4. Add a Button as private instance variable to DownloadBar, which should be a “Cancel” button for canceling the download. When the user clicks on the Cancel button, there should be a dialog box asking the user if they really want to quit. If they confirm the dialog, then it should stop downloading the file, and delete the file that was created. The DownloadBar should then be removed from the VBox where it was added.



Figure A Download window showing 3 downloads in progress, and a cancel dialog from trying to cancel one of them



Figure An example of a DownloadBar with Filename, ProgressBar, and Cancel button.

1. In the DownloadBar class, I have created a new inner class called DownloadTask, which extends javafx.concurrent.Task<Integer>. Task is an object which is meant to run on a separate thread for something that will take a long time, like downloading. The task Task class has 4 important functions that you must implement/Override:
   1. ***protected String call( ).*** This function does the downloading of the file. I have given the URL of a tutorial for downloading a file from the internet. Each iteration of the while loop should read a block of data, and then write it to the hard drive. After each time you read, you must update the task’s progress property by calling updateProgress(amount of data read, file size). Each iteration of the while loop should check if the Task has been cancelled, in which case it should exit the while loop, and delete the file that was created.
2. ***protected void succeeded().*** This function is called only after the call() function has finished, and the task has not failed, or been cancelled. It should remove the DownloadBar from the downloadWindow’s VBox of downloads in progress, and print a message filename + “ was successfully downloaded!” to the text area
3. ***protected void failed()***. This function is called if the task failed. It should remove the DownloadBar from the downloadWindow’s VBox of downloads, and print a message filename + “ download failed” to the text area. It should also remove the file that was written to disk
4. ***protected void cancelled()***. This function is called if the task was cancelled by the user. It should remove the DownloadBar from the downloadWindow’s VBox of downloads, and print a message filename + “ download was cancelled” to the text area. It should also remove the file that was written to disk

Only 1 of the functions succeeded, failed, or cancelled will be called after the call( ) function is finished. Similar to the catch() blocks of exceptions, you should write code that handles each possible outcome of the task:



Figure The result of canceling two download tasks

The Task class is similar to a Transition animation, in that you first set all of the data for the class but it doesn’t start to run until you tell it to. You can automatically bind the value of the task progress, to the progress value of the progress bar. That way whenever you call updateProgress() for the task, it will also update the value of the progress bar. This is done by creating a Thread object for your Task object, and starting the thread:

DownloadTask aFileDownload = new DownloadTask ( );

progressBar.progressProperty().bind( aFileDownload.progressProperty() );

new Thread( downloadTask ) .start();

1. The DownloadBar class should have a static Stage variable, called downloadWindow. This downloadWindow will be a second Stage object that will display the progress of all of the current DownloadTasks. It should have a BorderPane as the root, and a VBox in the center to display DownloadBars (which are a subclass HBox). The bottom of the BorderPane should be a TextArea which shows messages about the DownloadTasks. I have given the constructor for the DownloadBar which checks if the variable downloadWindow is null, and if it is, you should write code that creates the window and shows it. My code adds an onCloseRequest listener to the downloadWindow so that if it closes, the downloadWindow variable is set to null. That means that whenever the window is closed, creating a new DownloadTask object will recreate the window and show it again.
2. You should be able to download several files at the same time. Each DownloadBar is responsible for downloading its own file so they should run independently. The code is given to start a new thread so when a Task is started, you can think of it as an entirely separate sub-program that is running at the same time. For testing, the University of Waterloo’s computer science club has a mirror website for hosting the installation files (.ISO) for many different versions of linux:

<https://mirror.csclub.uwaterloo.ca>



Figure Downloading Multiple Linux ISOs. This shows 3 DownloadTasks running at the same time

This is a good test site to try various downloads running at the same time, and trying to download files when it already exists on your hard drive.

***Browser Interface Improvements***

1. Add a Menu to the GUI for “Settings”. There should be two new menuItems: Homepage and Downloads.
2. Clicking on Homepage should show a new dialog box asking the user to enter a new URL as the default Homepage to display when the browser starts. This URL is what should be saved in the settings text file.
3. Clicking on the Downloads item should show a new dialog box asking the user to enter the name of the subdirectory where to save downloaded files. Your program should create this new directory if it does not exist, or not create the directory if it already exists. You must check if the user has write permissions to this directory and if not, warn the user that their selection is not valid.
4. Add keyboard shortcuts for every menuItem. This is done using the setAccelerator function of menuItems. For example, this sets the combination (CTRL + A) as the shortcut for about:

about.setAccelerator(new KeyCodeCombination(KeyCode.A, KeyCombination.CONTROL\_DOWN));

Quit should have (CTRL+Q), About should have (CTRL+A), Help for java class should have (CTRL+H), History should be (CTRL+Y)

1. Add a key pressed listener to the WebView object so that typing (Ctrl + left arrow key) makes the history go back 1 page, and typing (Ctrl + right arrow key) makes the history go forward 1 page. The KeyEvent has an isControlDown() function to help you.
2. Add Tooltip descriptions to each of the buttons: Back, Forward, Add Bookmark. The JavaFX documentation for the Tooltip class for examples on how to do this.

***Bonus marks***

These next items are not required for the assignment, however you can get bonus marks towards your final grade for implementing any of the following features:

1. Inject javascript code: The WebEngine has an executeScript(String javascript) function that lets you execute javascript code on your page. Set the engine’s setOnAlert( ) function as a callback to show a JavaFX dialog box that displays the Alert text as the main text of the dialog, and then show it. Add a Javascript Menu to the interface that has an “Execute code” menu item. Clicking on the MenuItem prompts the user for a line of javascript code, which the browser will then execute. For example, typing the text: “alert(window.location)” should call the engine’s setOnAlert callback, which should show an Alert window with the engine’s current URL. Typing the text: “history.back()” should make the engine go back a page, and “history.forward()” should make the engine go forward. “history.go(-3)” should go back 3, etc (4 marks)
2. Launch program upon file download. When a file has finished downloading, it should ask the user if they want to launch the file. If the user accepts, open the file with the appropriate program. PDF files should be launched with Acrobat Reader, DOC files should be launched with Microsoft Word, etc. You will have to search google for how to do this. (4 marks)
3. Tabbed browsing. For your web browser, create a TabPane where the original WebView was. A tab pane contains several Tabs, which you create and set the layout, and then add a WebView to each tab. Switching tabs should update the address bar for that tab’s current URL, and update the History ListView for that tab’s WebHistory. (8 marks)

You can detect when a new tab has been selected with: tabs.getSelectionModel().selectedItemProperty().addListener((ov, oldTab, newTab) -> {

// get the engine of the currently selected WebView.

//Set the historyView for the current engine.

});

Add a “New Tab” menu item under the File menu, and set the accelerator (CTRL+T) to create a new tab.

1. Finished downloading animations. When a file has successfully downloaded, create an animation where the DownloadTask (subclass of HBox) appears to “fall” off the downloadManager window and also fade to 0%. You should also play some audio file as a notification. Look at the javafx.scene.media.AudioClip class to see how to do this: <http://what-when-how.com/javafx-2/working-with-audio-clips-using-the-media-classes-javafx-2/> (4 marks)

***Getting started***

I have posted the start of the project as an Eclipse project. It is the file on Blackboard: CST8284\_Assignment3\_YourName.zip . I have provided a function: **getWindow()**. This checks if the downloadWindow variable is null. If it is, it recreates the window and shows it. This means that after calling getDownloadWindow(), you are guaranteed that the downloadWindow is not null, and is visible. You can then add DownloadBar objects to the VBox.

Here is a tutorial on how to save URL to a download directory:

<http://www.codejava.net/java-se/networking/use-httpurlconnection-to-download-file-from-an-http-url>

This should be the contents of your call() function, but you should update the code so that it updates the progress bar as it reads from the stream. Also, the function in the tutorial throws an IOException if anything goes wrong. The call() function must not throw any exceptions, therefore you will have to add all of the catch() blocks that are needed and react accordingly. If the internet connection is disconnected during a download, then the task should end as a failed, instead of instead of succeeded, or canceled.

Here is a website that shows how to implement a simple Task object that counts from 1 to 50 and updates the ProgressBar:

<http://java-buddy.blogspot.ca/2013/08/javafx-example-task.html>

Also, look at the documentation for ProgressBar, and Task in JavaFX:

<https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/ProgressBar.html>

<https://docs.oracle.com/javafx/2/api/javafx/concurrent/Task.html>

Marks:

Saving Settings: (7 marks)

* The bookmarks are saved to a file using ObjectOutputStream. +1
* The window size and position is saved to file +4
* The Homepage and Download directory are saved +2

File Downloads: (11 marks)

* Clicking on one of the file links makes the Download Window appear. +1
* The user is able to cancel a download in progress +1
* Canceling a download deletes the file. +1
* Downloading a file with the same name as a file on the disk properly adds a number to the filename +1
* The Progress Bar shows proper progress +2
* Canceling a download removes the DownloadBar from the Download Window and prints a message to the message area +1
* Finishing a download removes the DownloadBar and prints a message to the message area +2
* Closing the Download Window and starting a new download makes the Download Window reappear. +1
* Disconnecting the internet access during a download is handled properly +1

Browser Interface improvements: (8 marks)

* There are new menu items for Download directory and Homepage that let the user set the values with an input dialog +2
* The Back, Forward and Add Bookmark buttons have tooltip windows. +2
* Menu Items have accelerator shortcuts +2
* The browser history goes forward and back by pressing Ctrl + front /back arrow +2

For JavaDocs: (6 marks)

* Each class has a JavaDoc page. +1
* Each method has a proper brief summary. +1
* Clicking on the method name leads to the methods detailed description. +1
* The JavaDocs are properly generated and included in the zip file. +1
* The Eclipse project of your assignment is properly exported as a zip file and can be imported again without problems (+2)

Submission:

* Modify the CoverPage.html file in the Eclipse project so that it has your name, student number.
* If you have done any work for Bonus Marks, clearly list which items you have implemented. Like Interfaces in java, the work can be there but you can only get marks for the work if you state clearly what parts you have and ask the person marking to verify the work!
* Export your entire java project as a ZIP file through Eclipse’s File->Export menu. Do not just zip the source directory, or use any other compression format (.7z, .rar, etc.)! Call your zip file: CST\_8284\_Assignment3\_***YourName***.zip, but replace ***YourName*** with your name… The assignment is due at 11:59 on April 22, which is a Friday night. **Late submissions will not be accepted!**