

# [CS209A 2022SP] Assignment 2 Text Reader

## Introduction

In this assignment, you will implement a simple text reader, which has the functions of viewing text, finding keywords, replace keywords and so on. During the process of finishing this assignment, you will have a chance to practice your JavaFX programming skill.

We will give you an incomplete project, and what you need to do is completing the code and make it work well.

Notice: In this assignment, you need to use some functions that you have implemented in **assignment 1 Text Processor**.

## Preparation: Load the incomplete project

This is the first step you need to do in this assignment.

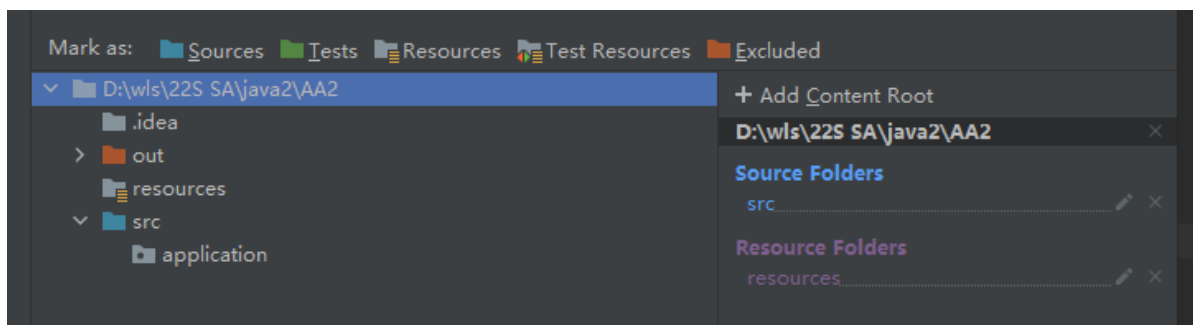
Firstly, download the **TextReader.zip** from Sakai. Then unzip it to the location you want.

Secondly, open IntelliJ IDEA. Click **File -> New -> Project from Existing Resources...** Then choose the folder you have just unzipped.

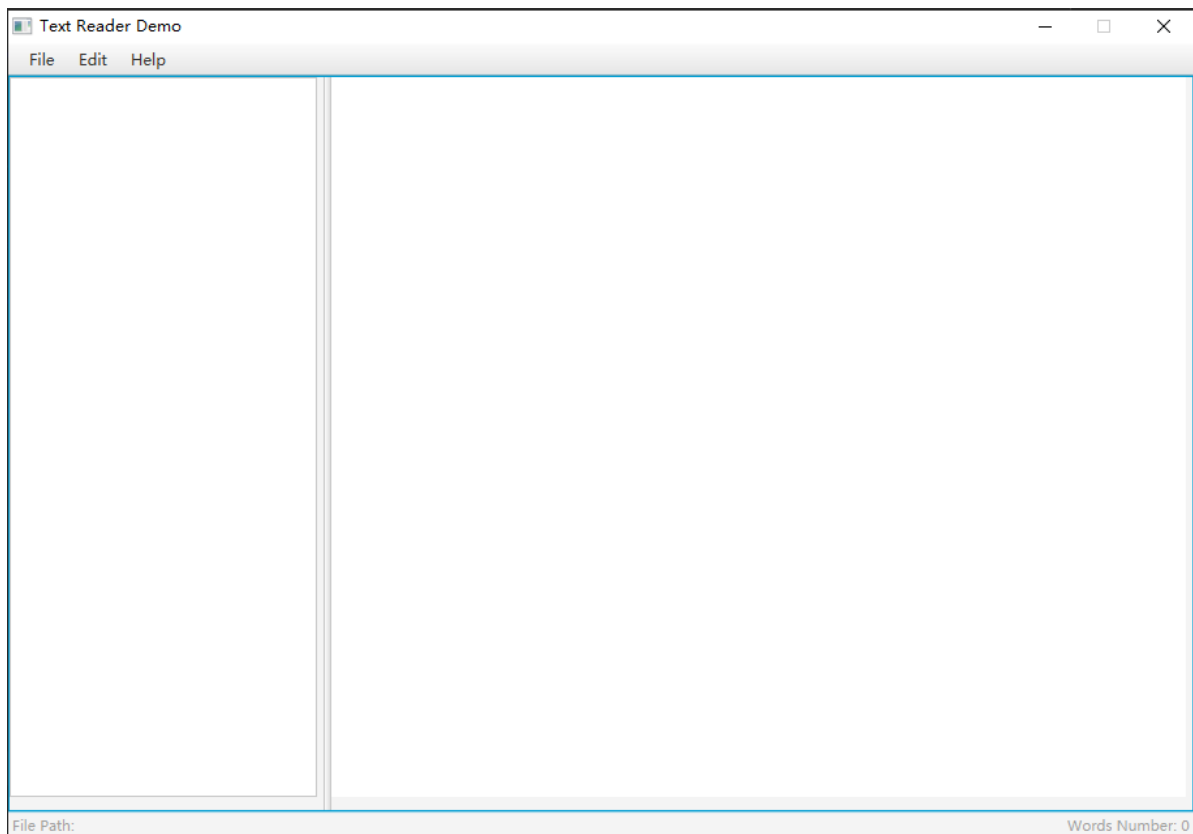
Thirdly, constantly click the **next** button until successfully creating the project.

**Notice:** In this step, we recommend you use **JDK8**, since that will help you skip the step of configuring the JavaFX environment. If you really want to use version higher than JDK11, you can configure the environment by yourself. :)

Finally, you need to mark the resources folder as **Resources** in the **Project Structure** page, just like the following figure.



Run `TextReader.main()`. If you can see a JavaFX program like the following figure, congratulations, you have already finished the preparation part!



## Part ONE: Complete the FXML file(10 points)

After loading the unfinished project. You can see the kind teachers and SAs have already design a concise application style for you in file **TextReader.fxml**. However, this file is also incomplete, since all items in FXML file haven't bond to the corresponding location in the controller.

What you need to do in this part is to complete the FXML file. You need to complete three kinds of elements: `fx:controller`, `fx:id` and `onAction`.

The controller is `application.TextReaderController`, and we have already create the needed fields and methods in controller. The location you are required to complete text is marked as `<!-- fx:controller-->`, `<!-- fx:id-->` and `<!--onAction-->` annotations.

If you are still confused to what to do, here are the examples:

```
<MenuItem mnemonicParsing="false" text="Open..." /> <!--onAction-->
```

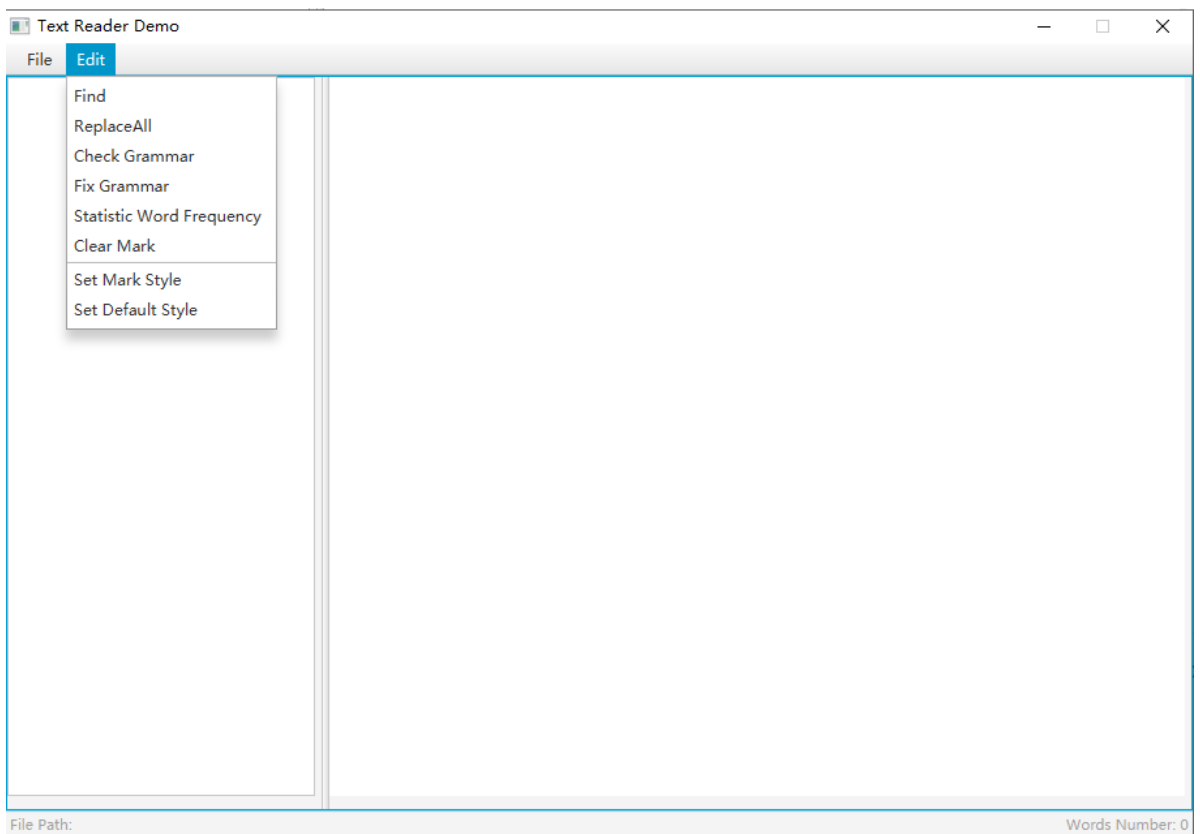
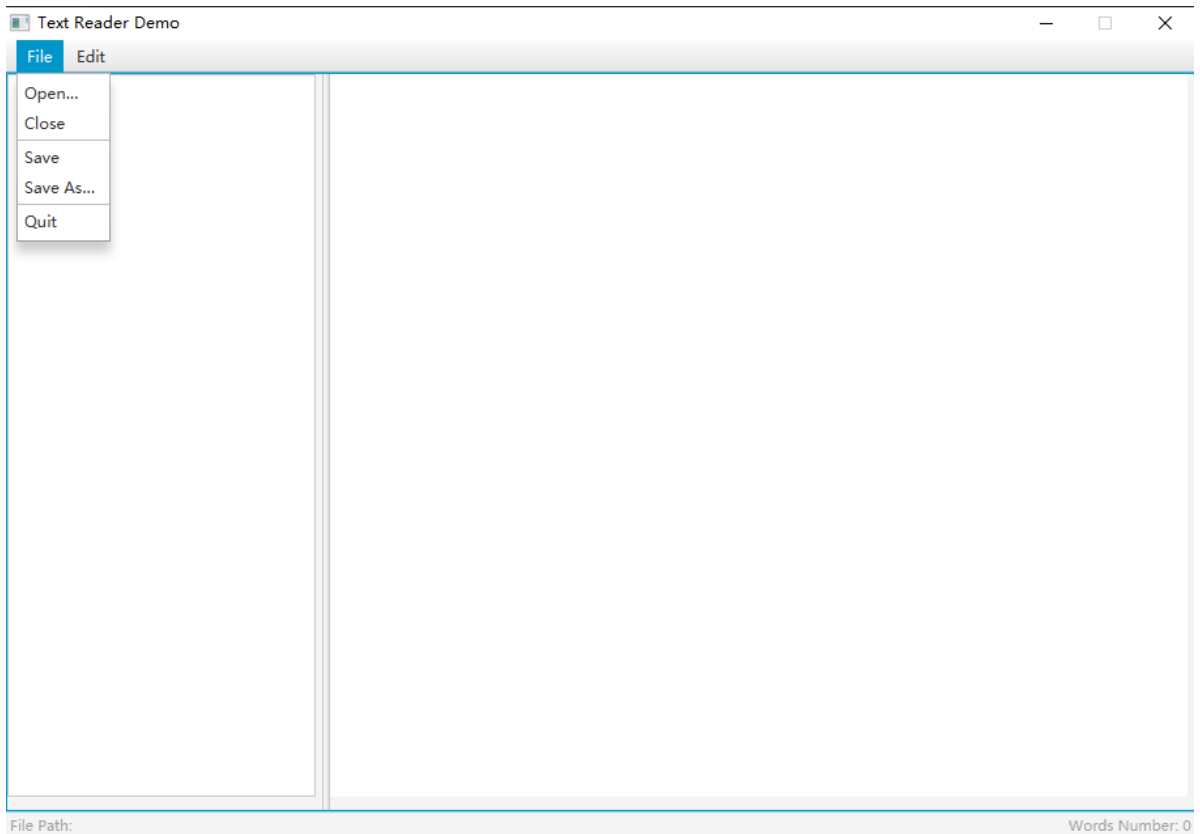
In this line, you need to add a `onAction` element. After read the text "Open..." on this item, you can infer that item is used to open the file. Then you check the methods in `TextReaderController`, and find that the method `openFile()` is very accord with its function. So, you can fill up the blank and make the line be:

```
<MenuItem mnemonicParsing="false" text="Open..." onAction="#openFile"/> <!--onAction-->
```

And that is right the thing you need to do.

You can click some buttons in the menu to check whether you have done well in this part. If the console print some information or give some reaction each time after you click the button, that means you might be right. If not, you must be wrong.

Here are the figures of the menu in the Text Reader, which may help you complete this part.



## Part TWO: Complete the Basic Methods(20 points)

Guess what is the most basic function for a text reader? You're right!. That is opening file and closing file. In this part, you are required to complete four methods in controller: `openFile`, `saveFile`, `saveAsFile` and `closeFile`.

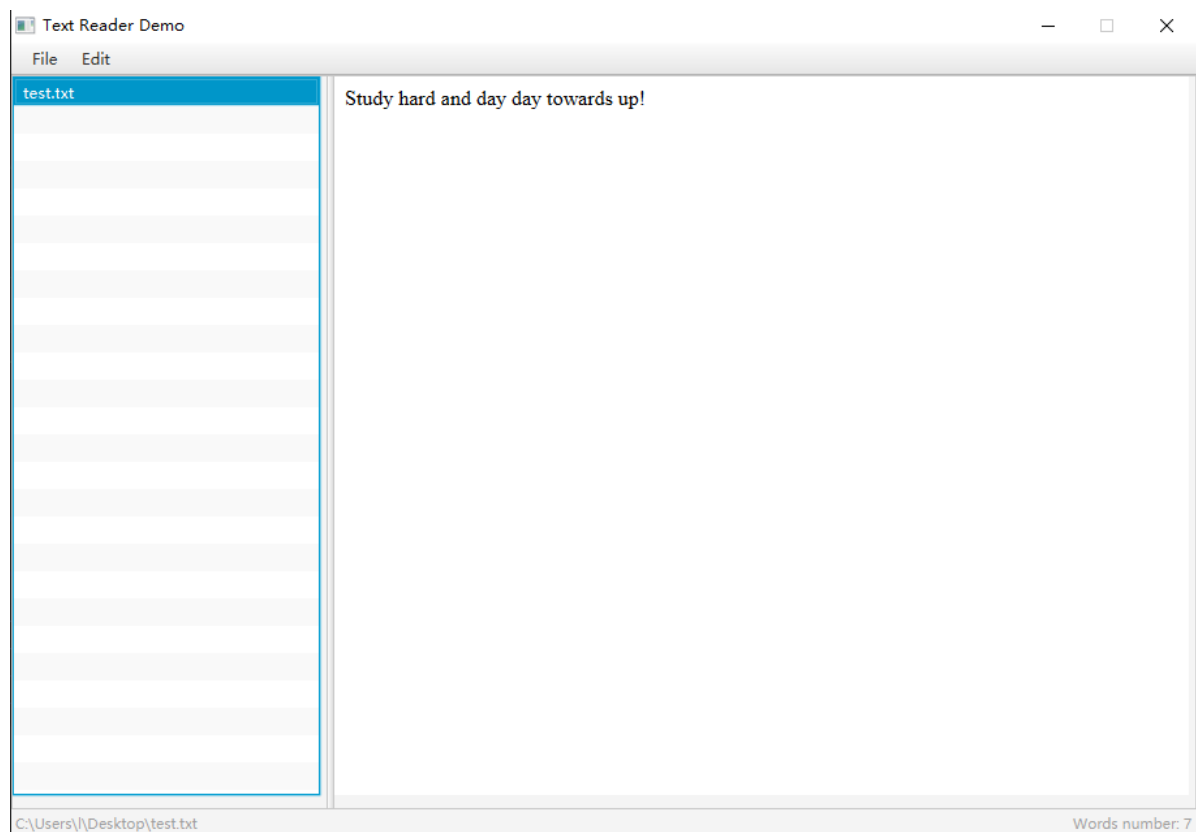
## Open File

The method `openFile` is to open a file using `FileChooser`, then add the file object to the `fileListView` Component.

Here is the detail you should do:

1. Provide a `FileChooser` to select a `.txt` file;
2. Use the file just selected to create a `FileCell` Object; (`FileCell` is a class already given in `TextReaderController.java`, you need to read its source code by yourself)
3. Add the `FileCell` into `fileListView`;
4. Refresh the `fileListView`;

If you correctly complete this function, the open file button in the menu will work well. You can see the successful case in the following figure.



**notice:**

- You should restrict the `FileChooser` to only select `*.txt` files.
- If your are confused to `FileChooser`, see the [Useful FileChooser API Refer](#)

## Save File

The method `saveFile` is to save the selected file to the disk. The save path is the origin path of the text file.

Here is the detail you should do:

1. Obtain the `FileCell` which is currently selected in `fileListView`.
2. Use `getReaderContent` method to obtain the current content in the right text-view window.
3. Save the content into the `FileCell`.
4. Write the content to the the File in the `FileCell`.

## Save As File

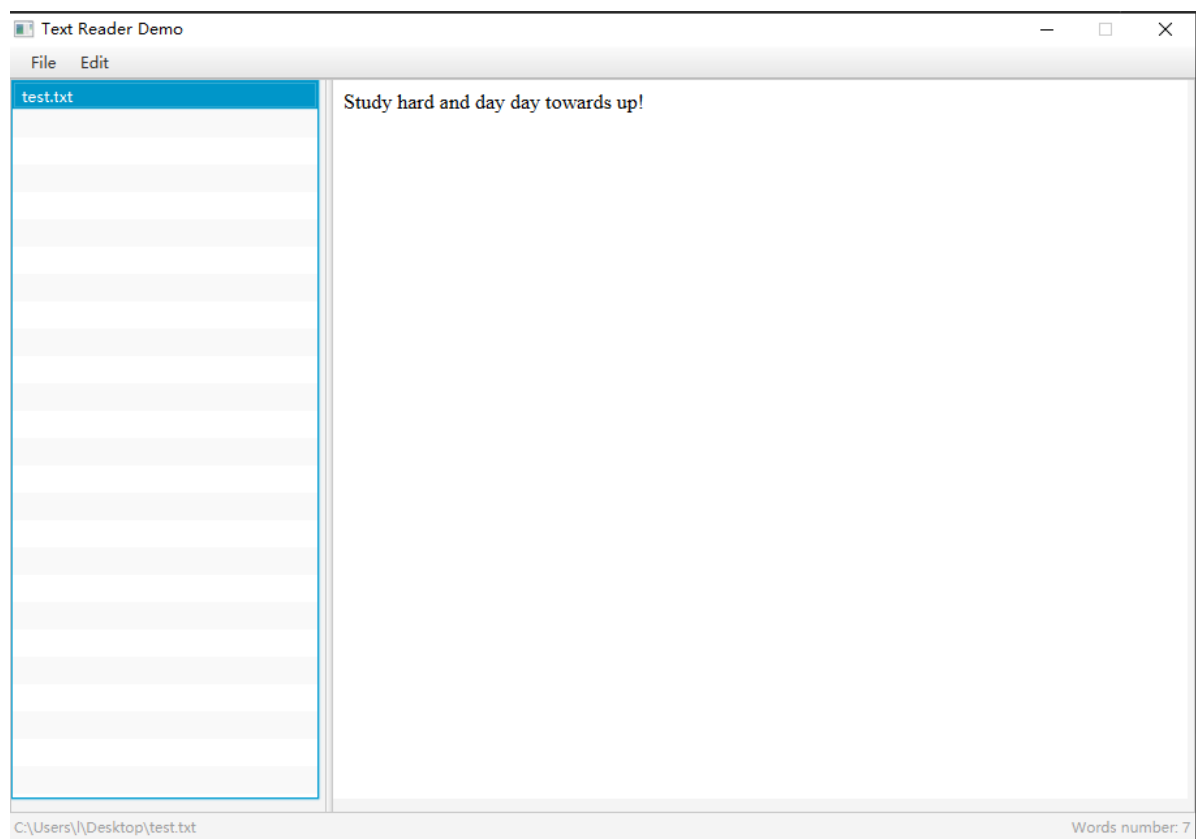
The method `saveAsFile` is also to save the selected file to the disk. However, you can save this file in another place. (另存为)

Here is the detail you should do:

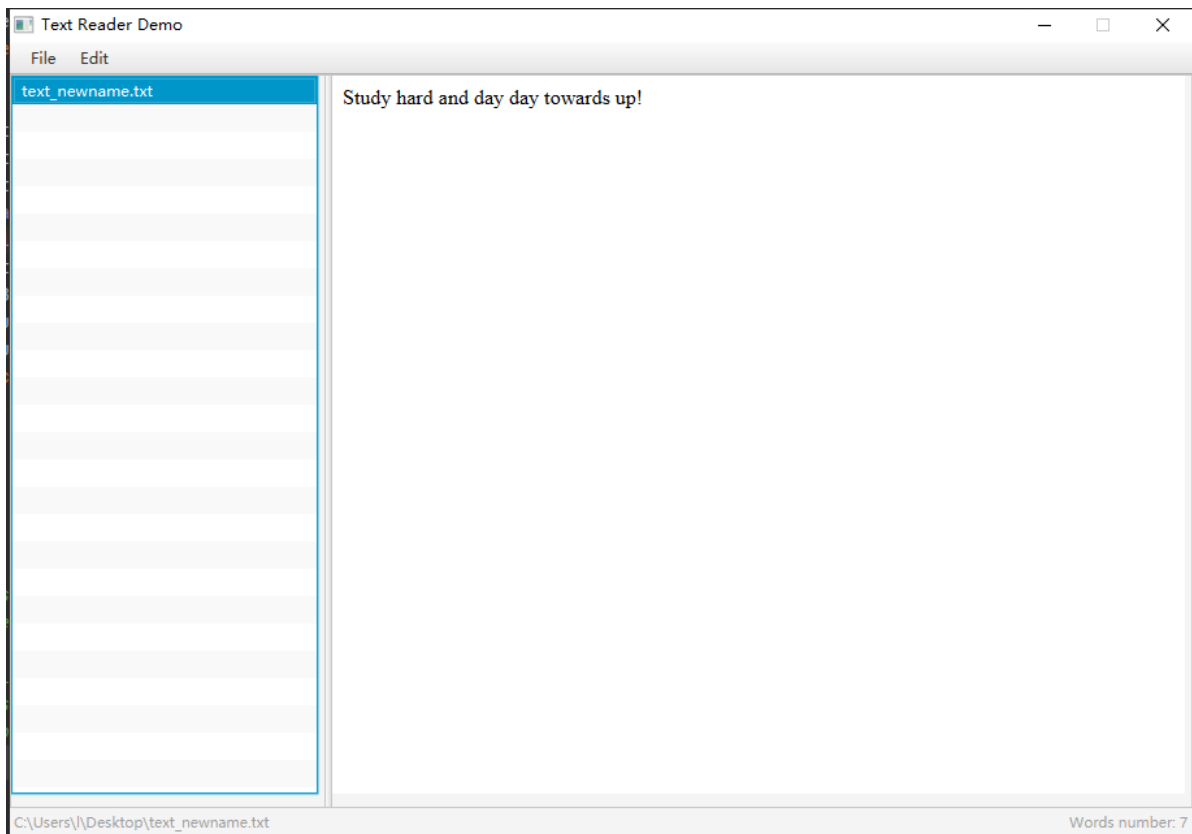
1. Use `FileChooser` to select the place to save.
2. Do what you do in `saveFile` method. The difference is this time the path to write is the path just selected.
3. Change the values of the FileCell, includes `File`, `absoluteFilePath`, `fileName`. Change
4. Finally, refresh the `fileListView`.

**Notice:** here is an example of `saveAsFile` method:

Before saving as file, the file name in the fileList is text.txt and the file path at the left bottom is "C:\Users\...\Desktop\test.txt"



After saving as file, the file name turned to be "text\_newname.txt" and the file path be "C:\Users\...\Desktop\text\_newname.txt"



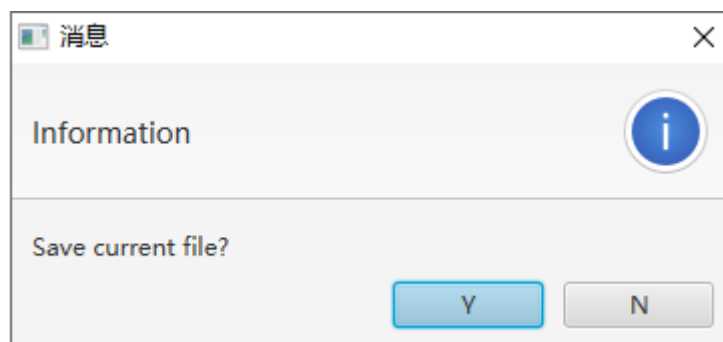
## Close File

The method `closeFile` is to close the selected file.

Here is the detail you should do:

1. Obtain the `FileCell` which is currently selected in `fileListView`.
2. Use a dialog to alert user whether to save the file. If the user choose yes, save the file.
3. Delete the `FileCell` in `fileListView`

Here is an example of the dialog:



## Part THREE: Find and Replace (20 points)

After finishing part two, your text reader should be able to do some basic operations. That isn't enough.

In this part you need to complete the `find` and `replaceAll` method.

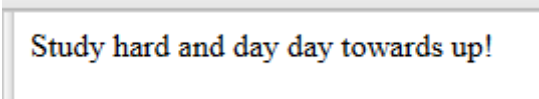
## Find

The method `find` is to find the user-input word in the reader. You need to highlight the word found in the reader.

### Important:

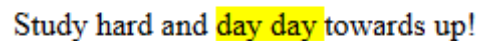
Before tell you the detail you need to do in this method, let me show you how to highlight the word in the reader:

Look, if the current text is `Study hard and day day towards up!`, the text in the File Reader will be:



Study hard and day day towards up!

Then, if the current text is `Study hard and <span class = "mark"> day day </span> towards up! '`



Study hard and day day towards up!

That is magic! Isn't it? We can see that the word "day day" has been highlighted!

...Well, maybe not so magic if you know something about html and css. However, what I want to tell you is that if you want to highlight some word, you can **use** `<span class = "mark">` **and** `</span>` **to surround** such word. That is easy, right?

Let's continue, here is the detail you should do:

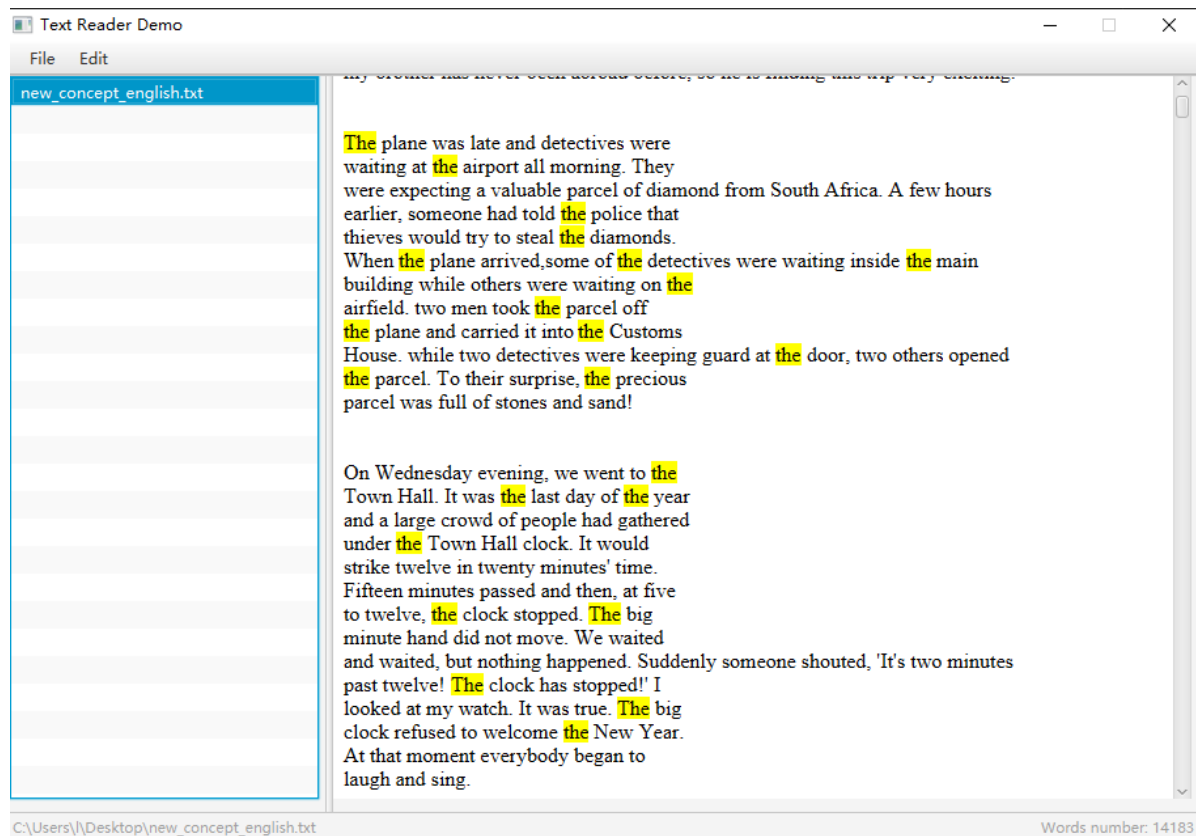
1. Provide a dialog to receive the word to find.
2. Use `getReaderContent` method to obtain the current content in the right text-view window.
3. Fix the content to highlight the keywords.
4. Use `setReaderContent` to put the fixed content into right window.

### Notice:

1. Matching is **case insensitive(忽略大小写)**.
2. The `find` function is to find the input **word**, but **not the input substring**.

For example, if the input word is "The", you need to highlight all "The" and "the" in the Reader. You can't highlight the words which contain "The" but do not equal to "The", like "They", "There" or "Theater".

Here is an example of this function, we use the keyword "The" to find in the given file "new\_concept\_english.txt", and here is the result:



## ReplaceAll

The method `replaceAll` is to find all the user-input keyword in the reader and replace it to substitution word. Then highlight all the substitution word.

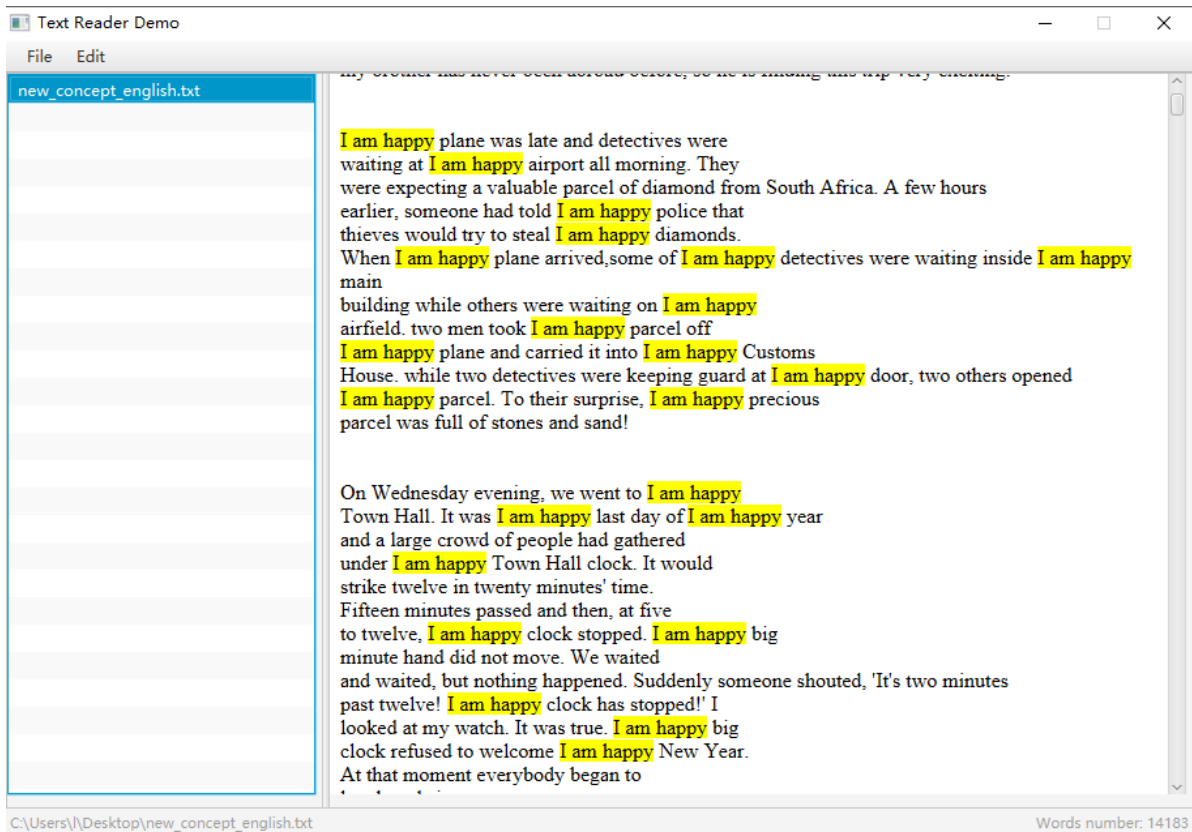
Here is the detail you should do:

1. provide a dialog to receive the keyword and the substitution word.
2. Use `getReaderContent` method to obtain the current content in the right text-view window.
3. Fix the content to replace the keyword and highlight the substitution word.
4. Use `setReaderContent` to put the fixed content into right window.

**Notice:** The same as the function `find`, the match is case insensitive and you need to replace the words equal to the input keyword, but not the words contain keyword.

Here is an example of this function, we use the keyword "The" and the substitution word "I am happy" to do replace all keyword in the given file "new\_concept\_english.txt", and here is the result:





## Part FOUR: Update Words Number (10 points)

You may find that after you used `replaceAll` operation, the words number at the right bottom **didn't** change. That's because you haven't complete the `updateWordsNumber` method. In this part you need to complete it.

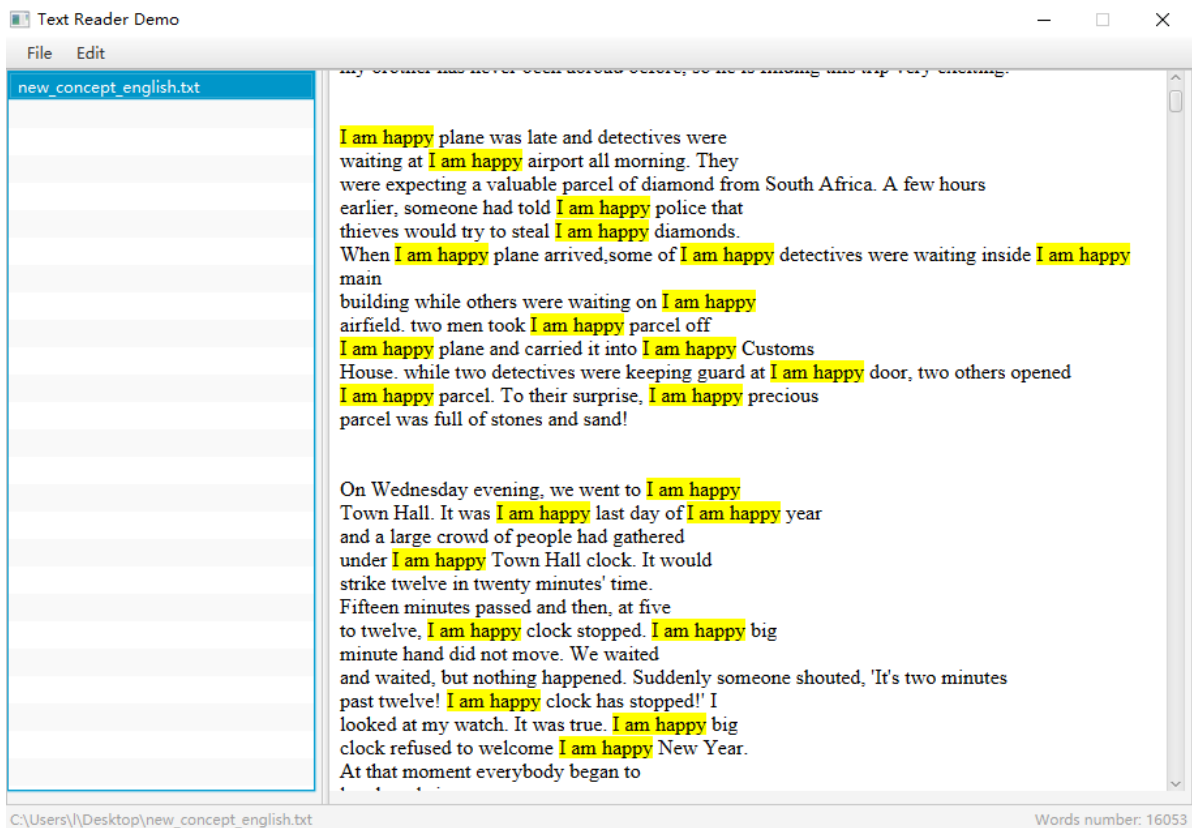
### Update Words Number

The method `updateWordsNumber` is to update the words number in bottom-right label.

Here is the detail you should do:

1. Use `getReaderContent` method to obtain the current content in the right text-view window.
2. Find out the number of words in the content.
3. change the text in `wordsNumberLabel1`.

After you complete this function, you can see the change of the "Words number" label after you make replacing all function. Just like the following figure. The correct value of "Words number" should be 16053.



## Part FIVE: Check Grammar and Fix Grammar (20 points)

In this part you need to implement the Checking Grammar and Fixing Grammar functions. You may need to use some functions that you have implemented in **assignment 1 Text Processor**.

**Notice:** In the following two functions, you don't need to provide any dialogs or buttons. User should operate these functions by clicking the item in the menu.

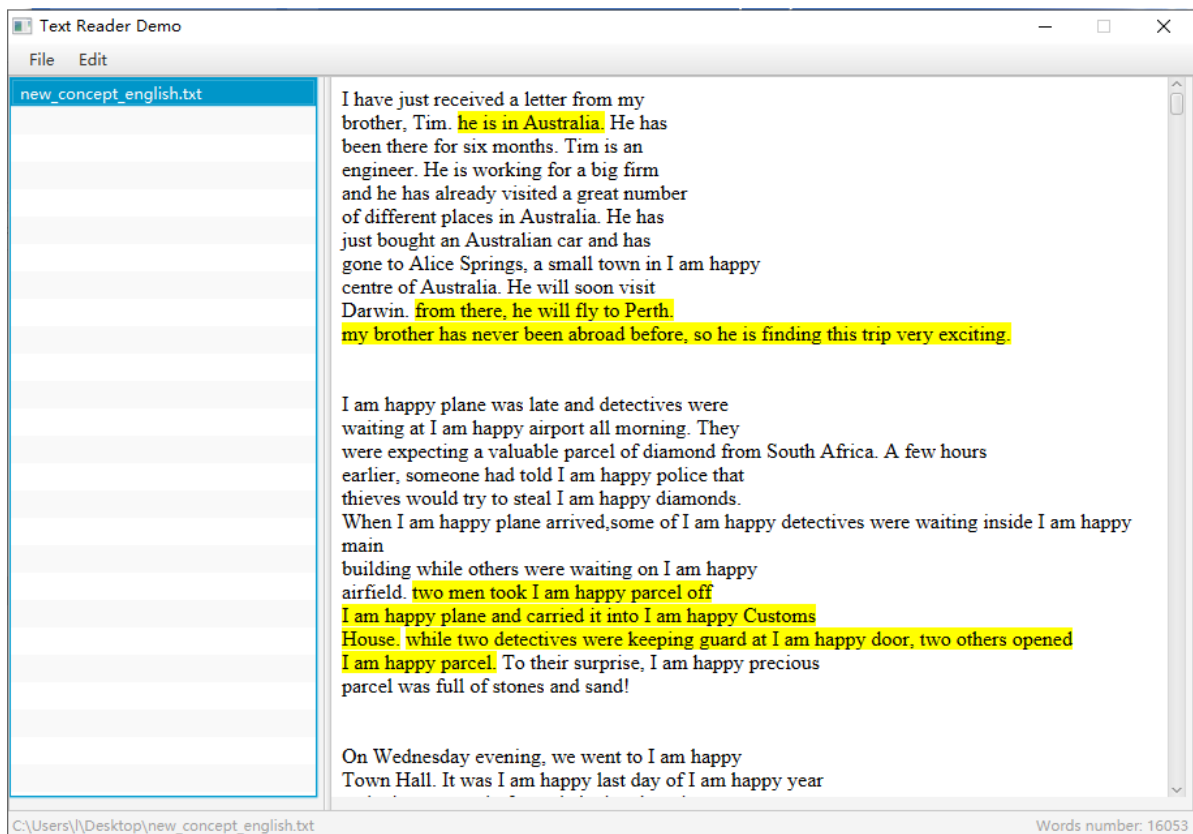
### Check Grammar

The method `checkGrammar` is to check the correctness of the first letter of the English sentence. (Just like what in assignment 1)

Here is the detail you should do:

1. Use `getReaderContent` method to obtain the current content in the right text-view window.
2. Find all the sentences with lower case first letter in the content.
3. Highlight the sentences with lower case first letter.

Here is an example of using check grammar operation:



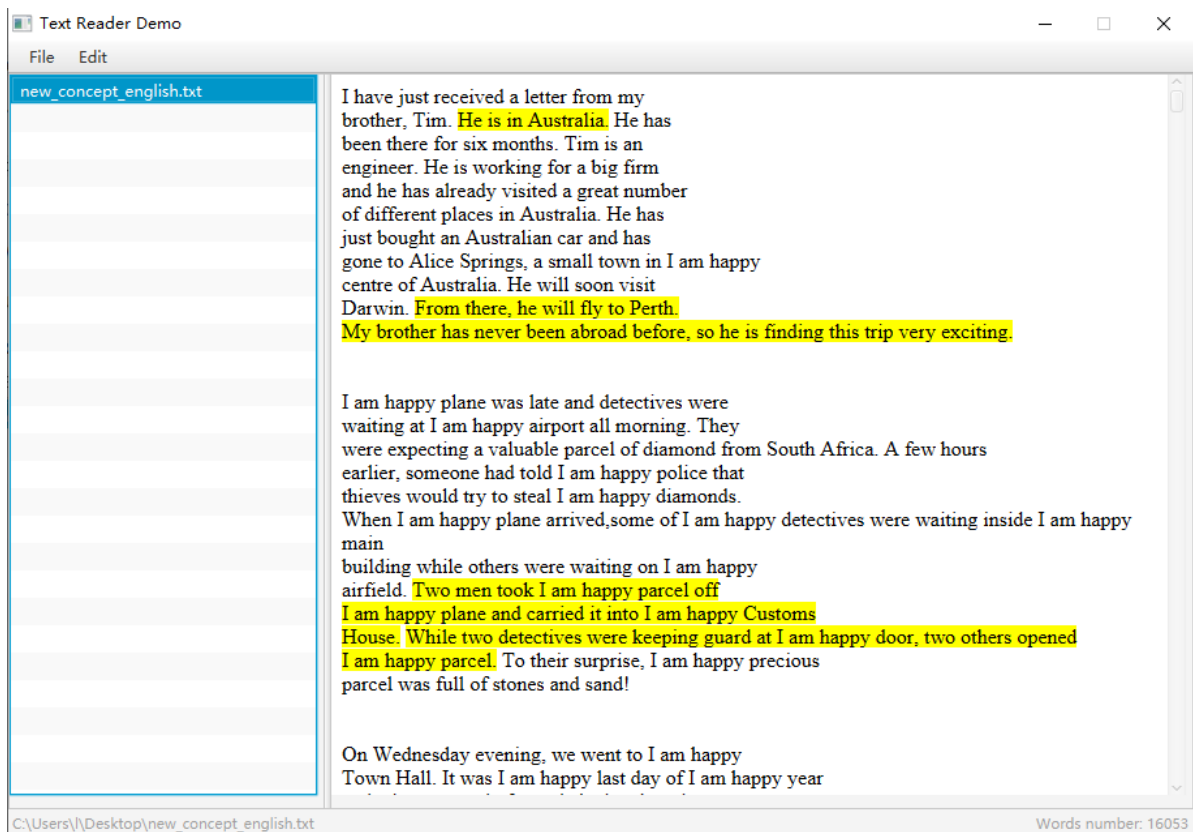
## Fix Grammar

The method `fixGrammar` is to fix the correctness of the first letter of the English sentence.

Here is the detail you should do:

1. Use `getReaderContent` method to obtain the current content in the right text-view window.
2. Find all the sentences with lower case first letter in the content.
3. Fix the sentences with lower case first letter(make it be higher case), then highlight the sentences.

Here is an example of using fix grammar operation:



## Part SIX: Statistic Word Frequency (20 points)

In this part, you need to statistic the frequency of the words in the reader. Then, you need to draw a diagram to show the result. What you need to do is complete the `statisticwordFrequency` method.

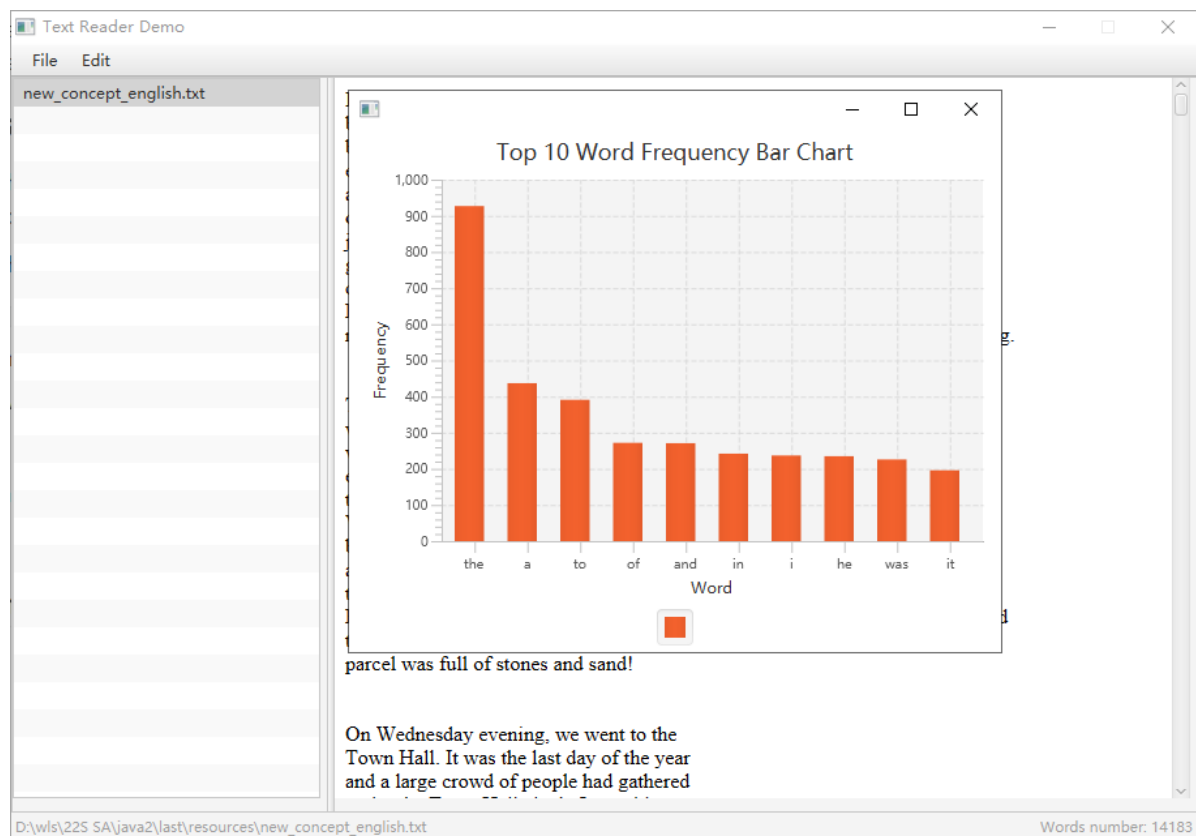
### Statistic Word Frequency

The method `statisticwordFrequency` is to statistic the **top-10** words frequency in the reader. (Just like what in assignment 1, however, you do **not** need to consider the stop words this time)

Here is the detail you should do:

1. Use `getReaderContent` method to obtain the current content in the right text-view window.
2. count the appear times of each word in the content.
3. draw a diagram to show the result.

Here is an example of using statistic word frequency operation:



## What to Submit

Submit a "zip" format file to Sakai, which should contain your entire Java project.

### Notice:

1. The file name should be: "Student Id+Name.zip"(e.g., 11911012王力爽.zip)
2. Please ensure your code compiles.
3. **DO NOT** use Chinese in your code comments.