King Saud University

College of Computer and Information Sciences

Department of Computer Science

CSC 212 Data Structures Project Report

1st Semester 2017-2018

# Developing an Application for Manipulating Subtitles

## Authors

| Name | ID | Section |
|---|---|---|
| Shahad Abdullah Almuhaizi | 436201525 | 44126 |
| Maram Khalid Alodan | 436200913 | 44126 |
| Reema Essam Almuhanna | 436200927 | 44126 |
| Noura Abdullah Alsa'adan | 436202586 | 44126 |

## 1. Introduction

This Application Subtitles display the cinema or television screen translate, this translate take place in audio or video and in this application we are working on translator “srt format “ that have the basic things like Start time and End time of the sentence and the main text in this specific time. This project manipulate subtitle files.

## 2. Specification

1. We have interface called "List".
2. We have interface called "Subtitle".
3. We have interface called "SubtitleSeq".
4. We have interface called "Time".
5. We have class called “ SubtitleSeqFactory”.
6. We have class called “ SubtitleSeqImp”.
7. We have class called “ TimeInterval”.
8. We have class called "LinkedList".
9. We have class "SortedBST".
10. We have the main class "perf".
11. We have class "pair”.
12. We have class "SubtitleSeq".
13. We have class called “ SubtitleImp”.

14.We have class called "TimeImp".

15.We have class called "LinkedList".

## Phase1:

We have a class called"DoubleLinkedList".
Elements: The elements are of generic type <Type> (The elements are placed in nodes

for linked list implementation).

Structure: the elements are linearly arranged. The first element is called head, there is a element called current.

Domain: the number of elements in the list is bounded therefore the domain is finite. Type name of elements in the domain: List

Linkedlist

1.Method FindFirst ( )
    requires: list L is not empty. input: none
    results: first element set as the current element. output: none.

2.Method FindNext ( )
    requires: list L is not empty. Cur is not last. input: none
    results: element following the current element is made the current element. output: none.

3.Method Retrieve (Type e)
    requires: list L is not empty. input: none
    results: current element is copied into e. output: element e.

4.Method Update (Type e).

requires: list L is not empty. input: e.
results: the element e is copied into the current node. output: none.

5.Method Insert (Type e).

requires: list L is not full. input: e.
results: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. output: none.

6.Method Remove ( )

requires: list L is not empty. input: none
results: the current element is removed from the list. If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element otherwise first element is made the new current element. output: none.

7.Method Full (boolean flag)

input: none. returns: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. output: flag.

8.Method Empty (boolean flag).

input: none. results: if the number of elements in L is zero, then flag is set to
true otherwise false. Output: flag.

9.Method Last (boolean flag).

input: none. requires: L is not empty. Results: if the last element is the current element then flag is set to true otherwise false. Output: flag

Double linked list

Operations: We assume all operations operate on a list L.

1.Method FindFirst ( )
requires: list L is not empty. input: none
results: first element set as the current element. output: none.

2.Method FindNext ( )
requires: list L is not empty. Cur is not last. input: none
results: element following the current element is made the current element. output: none.

2.Method FindPrevious ( )
requires: list L is not empty. Cur is not Head. input: none
results: element Previous to the current element is made the current element. output: none.

3.Method Retrieve (Type e)
requires: list L is not empty. input: none
results: current element is copied into e. output: element e.

4.Method Update (Type e).
requires: list L is not empty. input: e.
results: the element e is copied into the current node. output: none.

5.Method Insert (Type e).

requires: list L is not full. input: e.
results: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. output: none.

6.Method Remove ( )

requires: list L is not empty. input: none
results: the current element is removed from the list. If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element otherwise first element is made the new current element. output: none.

7.Method Full (boolean flag)

input: none. returns: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. output: flag.

Elements: The elements are of generic type <Subtitle> (The elements are placed in

nodes for linked list implementation).

Structure: the elements are linearly arranged. The first element is called head, there is a element called current.

Domain: the number of elements in the list is bounded therefore the domain is finite.

Type name of elements in the domain: List

Method: getStartTime()
Requires: Input:none
Results: Return the start time of the Subtitle. Output: startTime.

Method: getEndTime()
Requires: Input: none.
Results: Return the end time of the Subtitle. Output: endTime.

Method: getText()
Requires: Input: None.
Results: Return the subtitle text. Output: text.

Method: setStartTime(Time startTime) Requires: Input: startTime.
Results: Set the start time of the Subtitle. Output: None.

Method: setEndTime(Time endTime) Requires: Input: endTime.
Results: Set the end time of the subtitle.

Output: None.

Method: setText(String text) Requires: Input: text. Results: Set the subtitle text.

Output: None.

Method addSubtitle(Subtitle st).
requires: list L is not full input: st

results:subtitle is added to the list. output: none. Method getSubtitles().

requires: list L is not empty. input:none.

results: time is sorted and a list containing the sorted time is returned. output: list.

Method getSubtitle (Time time).
requires: list L is not last. input: time.

results: the time received is compared with start and end time. if the currents’ startTime is greater than time and the currents’ endTime is greater than time, then the the current is retuned. output: Subtitle.

Method getSubtitles (Time startTime, Time endTime).

requires: list L is not last. input: startTime,endTime.

results: startTme received is compared with endTime. if current time in the list is greater than startTime, and if endTime is greater than current time in the list , then the current time is inserted in a new list and this list it returned output:list.

Method getSubtitles(String str).
requires: list L is not last input: str.

results: the method checks if the list contains str, if it does then the current element is inserted in a new list and the new list is returned output: list.

Method remove(String str).
requires: list L is not empty and str is not null. input:str.

results: the method checks if the list contains str, if it does then the current element is removed from the list output: none.

Method replace(String str1,String str2).
requires: list L is not empty and str1 is not null. input: str1,str2.

results: the method checks if the list contains str1, if it does then the current element is replaced with str2 output: none.

Method shift (int offset).
requires: list L is not last. input: offset. results: the time is shifted output: none.

Method cut (Time startTime, Time endTime).
requires: list L is not empty and not last input: startTime, endTime.

results: the method check is the current time is greater than strartTime and the current time is greater than endTime, then the current is removed. output: none.

Method SubtitleSeqFactory ()
Requires : none , Input : none
Results : return an empty subtitles sequence , Output : SubtitleSeq .

Method loadSubtitleSeq(String fileName)
Requires : The file not empty . Input : File Name
Results : return subtitle sequence from SRT file , or return null if the file incorrect format or doesn't exist . Output : SubtitleSeq .

*Class Perf:*

Method calcToMS (TimeSt st):

Requires: Map not empty. input: TimeSt st.

Results: the method calculates and returns time in milliseconds.

Output: Time in milliseconds.

## Phase2:

Method calc (int offset ):

Requires: Map not empty. Input: int offset.

Results: The method shifts the subtitle by comparing time in milliseconds with offset. If offset is greater that or equal milliseconds then it's shifted by addition. If offset is less than milliseconds then it will be shifted by subtraction
Output:none.

Method compareTo (TimeSt t ).

Requires: Map not empty. Input: TimeSt t.

Results:the method returns 1 if the milliseconds greater than the millisecond of the received object , returns -1 if its less than the millisecond of the received object and returns zero otherwise.

Output:1, 0 or -1.

*Class sortedBST:*

Method updateKey (K newKey ).

Requires: Map not empty. Input:K newKey.

Results: If newKey does not respect the order, no change is made and false is returned.

Output: Flag.

Method inRange (K k1,K k2 ):

Requires: Map not empty. Input:K k1,k2 .

Results: Return in a list all elements with key k satisfying: k1 <= k <= k2.
Output: list.

Method insert (K key,T data ):

Requires: Map not full. Input: K key, T data .

Results: the method Inserts a new element if does not exist and return true. The current points to the new element. If the element already exists, current does not change and false is returned.

Output: Flag.

Method find (K key ):

Requires: Map not empty. Input:K key .

Results: the method Search for element with key k and make it the current element if it exists.If the element does not exist the current is unchanged and false is returned.

Output: Flag.

Method remove (K key ):

Requires: Map not empty. Input: K key .

Results: the method removes the element with key k if it exists and return true. If the element does not exist false is returned (the position of current is unspecified after calling this method)

Output: Flag.

Method nbNodesInSearchPath (K key):

Requires: Map not empty. Input: K key .

Results: the method returns the number of nodes in the search path for key
Output: number of nodes .

Method remove ():

Requires: Map not empty. Input: none.

Results: the method removes the current element. The next element if it exists is made current otherwise current moves to the first element

Output: none.

Method FindFirst ( )

Requires: Map is not empty. Input: none.

Results: Move current to the first element .

Output: none.

Method FindNext ( )

Requires: map is not empty & current not last. Input: none.

Results: Move current to the next element.

Output: none.

Method findPrevious ()

Requires: map is not empty& current must not be first. Input: none.
Results: Move current to the previous element.
Output: none.

Method first () :

Requires: Map not empty Input: none.
Result: returns Returns true if current is the first element.
Output: Flag.

Method findLast ().

Requires:Map not empty.Input: none.

Results: Move current to the last element .

Output: none.

Method Last ().
requires:map not empty .Input: none.

Requires: L is not empty. Results: Returns true if current is the last element .

Output:flag.

Method size ( ):

Input: none
results: the method returns the size of the map

Output: size.

Method empty ( )
:

Input: none.

Results: Return true if the map is empty. Output: flag.

Method full ( ):

Input: none.Results: Return true if the map is full . Output: flag.

Method update (T e):

Requires: Map is not empty. Input:T e.

Results: the method updates the data of current element.
Output: none.

Method retrieve ( ):
Requires: Map is not empty. Input:none.
Results: the key and data of the current element. Output: none.

*Class TimeInterval:*

Method compareTo(TimeInterval that):

Input:that. Requires:Map not empty.

Results:the method returns 1 if startTime compared to endTime is greater than zero,and returns -1 if startTime compared to endTime is less than zero , and returns zero otherwise.

Output:1,-1 or 0.

*Class SubtitleSeqImp:*

Method addSubtitle (Subtitle st):

Input:st. Requires:Map not full.

Results:the method adds a subtitle.

Output:none.

Method getSubtitle ():

Input:none. Requires:Map is not empty.

Results:the method Return all subtitles in their chronological order .

Output:list of Subtitles.

Method getSubtitle (Time time):

Input:Time time. Requires:Node not last.

Results:the method returns the subtitle displayed at the specified time, null if no subtitle is displayed.

Output:subtitle.

Method getSubtitle (Time startTime,Time endTime):

Input:Time startTime,Time endTime. Requires:Node is not last.

Results:the method Returns in chronological order, all subtitles displayed between the specified start and end times. The first element of this list is the subtitle of which the display interval contains or otherwise comes Immediately after startTime. The last element of this list is the subtitle of which the display interval contains or otherwise comes immediately before endTime.

Output:list.

Method nbNodesInSearchPath (Time time):

Input:Time time. Requires:Map not empty.

Results:the method returns the number of nodes in the search path for finding the subtitle.

Output:number of nodes.

Method shift (int offset):

Input:offset. Requires:Node not last.

Results:the method Shifts the subtitles by offseting their start/end times with the specified offset (in milliseconds). The value offset can be positive or negative. Negative time is not allowed and must be replaced with 0. If the end time becomes 0, the subtitle must be removed.
Output:none.

*<u>Calss SubtitleSeqFactory:</u>*

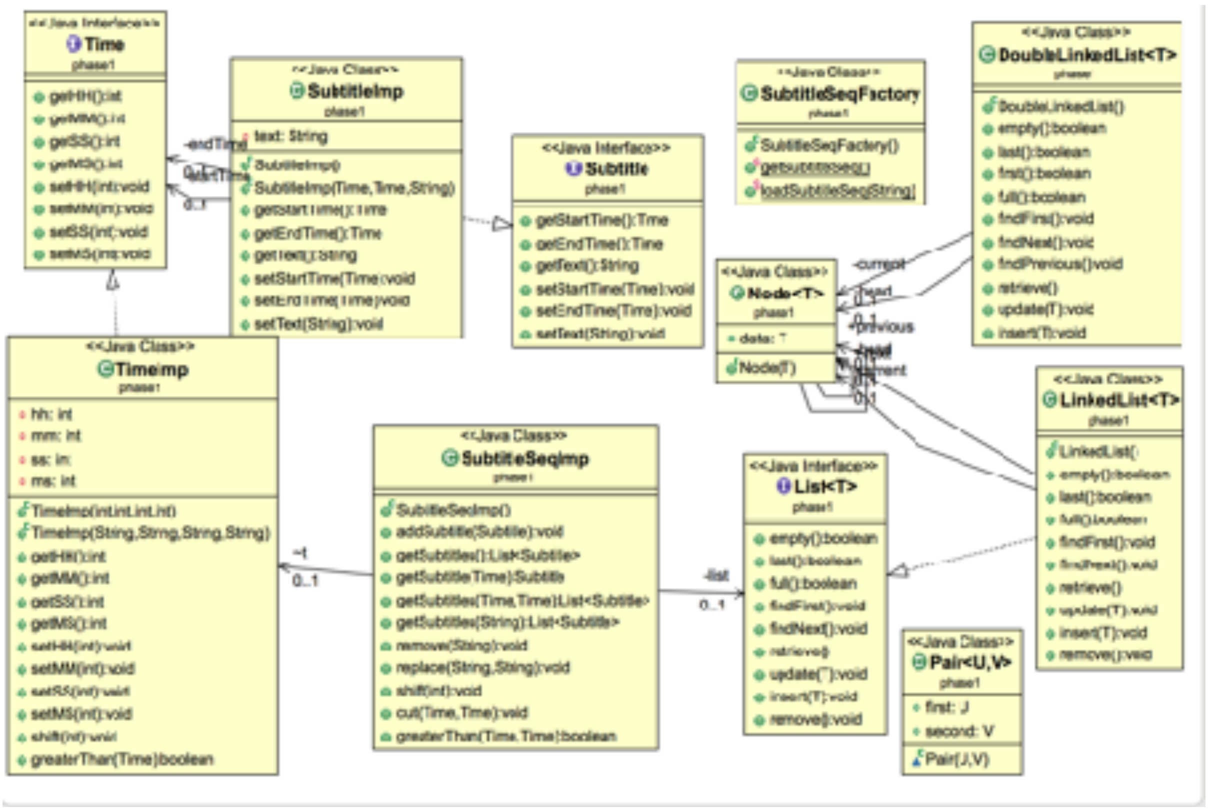
Method loadSubtitleSeq (String fileName):

Input:String fileName. Requires:Map not empty.

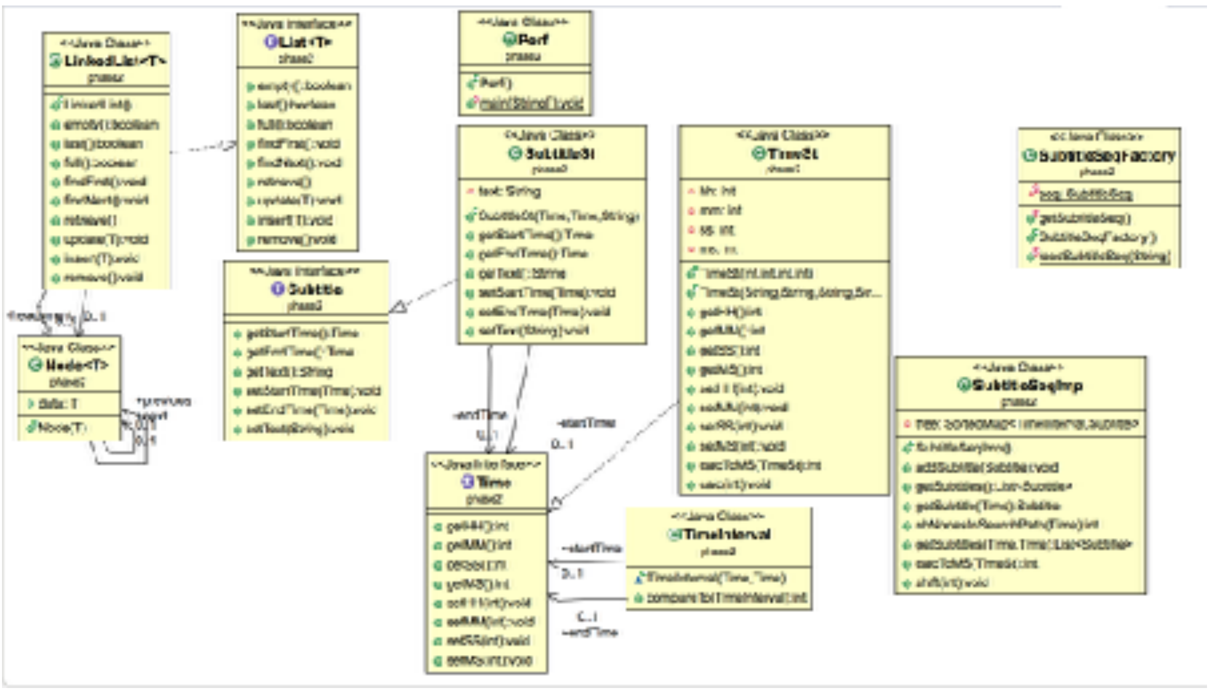
Results:the method Loads a subtitle sequence from an SRT file. If the file does not exist or is corrupted (incorrect format), null is returned.
Output: Subtitles or null.

## 3. Design

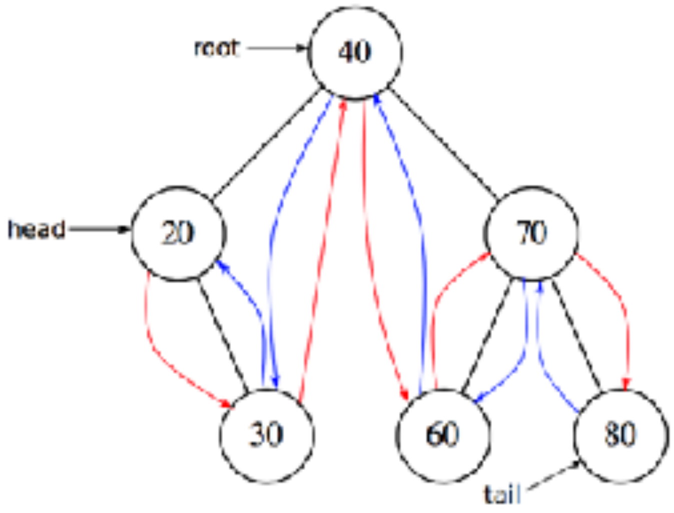
### Linked List UML:



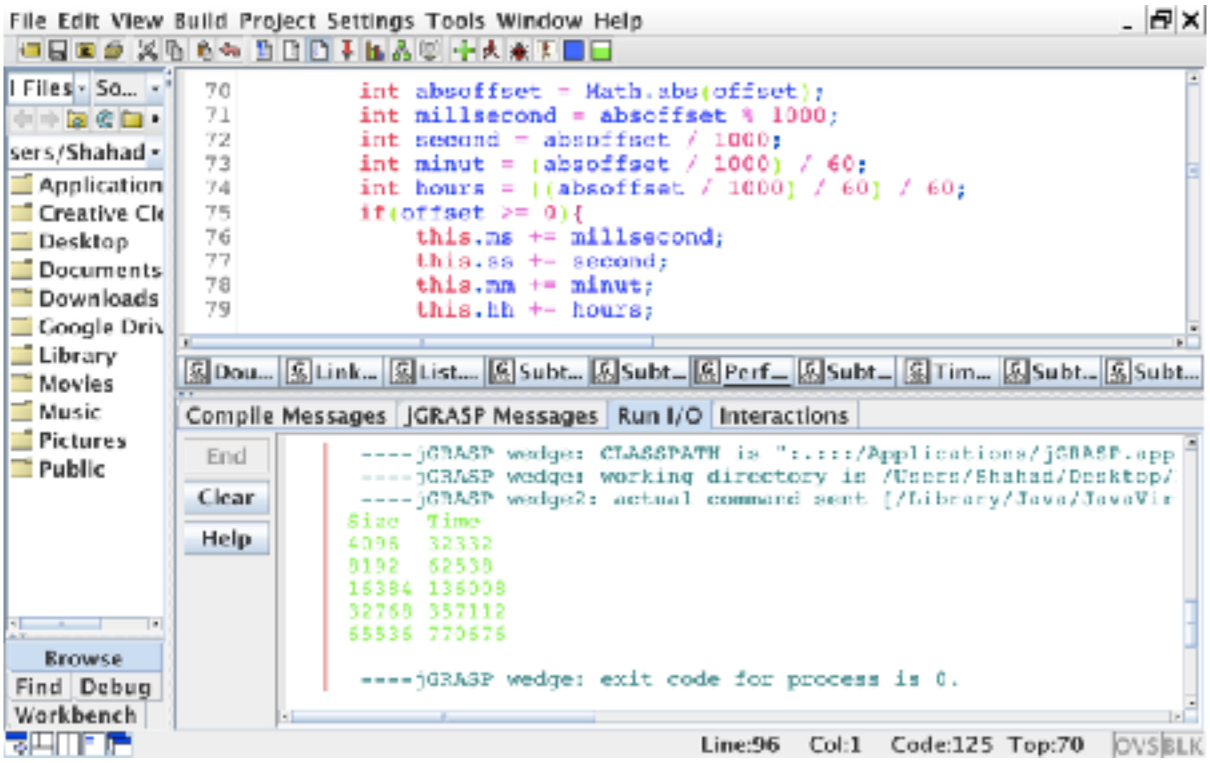
### SortedBST UML



## SortedBST Structure



## Linked list Perf Run:



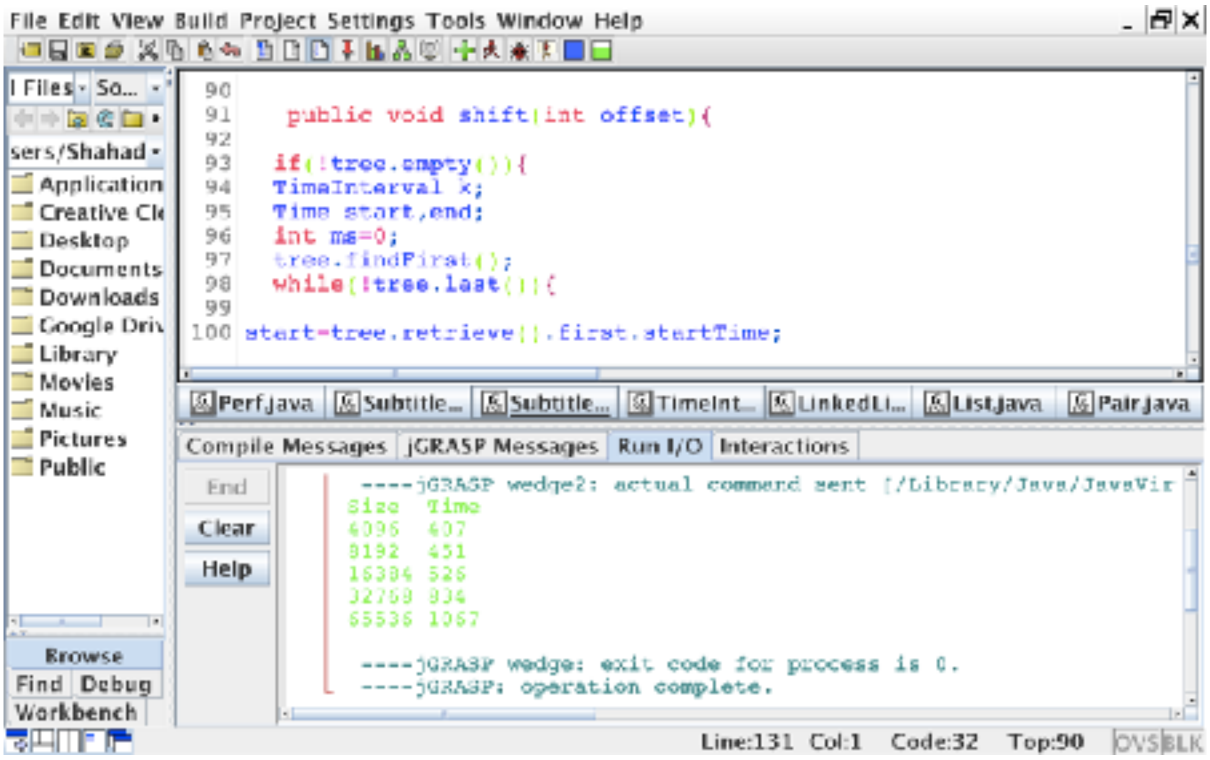
```
        int absoffset = Math.abs(offset);
        int millsecond = absoffset % 1000;
        int second = absoffset / 1000;
        int minut = (absoffset / 1000) / 60;
        int hours = ((absoffset / 1000) / 60) / 60;
        if(offset >= 0){
            this.ms += millsecond;
            this.ss += second;
            this.mm += minut;
            this.hh += hours;
```

```
 ----jGRASP wedge: CLASSPATH is ":.:::/Applications/jGRASP.app
 ----jGRASP wedge: working directory is /Users/Shahad/Desktop/
 ----jGRASP wedge2: actual command sent [/Library/Java/JavaVir
Size  Time
4096  32332
8192  62538
16384 135008
32768 357112
65536 770676

 ----jGRASP wedge: exit code for process is 0.
```

Line:96 Col:1 Code:125 Top:70

## SortedBST Perf Run:



```

  public void shift(int offset){

 if(!tree.empty()){
 TimeInterval x;
 Time start,end;
 int ms=0;
 tree.findFirst();
 while(!tree.last()){

start=tree.retrieve().first.startTime;
```

```
 ----jGRASP wedge2: actual command sent [/Library/Java/JavaVir
Size  Time
4096  407
8192  451
16384 526
32768 834
65536 1067

 ----jGRASP wedge: exit code for process is 0.
 ----jGRASP: operation complete.
```

Line:131 Col:1 Code:32 Top:90

## Performance Analysis

| Linked List | | SortedBST | |
|---|---|---|---|
| size | time | size | time |
| 4096 | 32332 | 4096 | 407 |
| 8192 | 625328 | 8192 | 451 |
| 16384 | 136008 | 16384 | 526 |
| 32768 | 357112 | 32768 | 834 |
| 65536 | 770676 | 65536 | 1067 |

## Big o

| Method | big O | |
|---|---|---|
| addSubtitle(Subtitle st) | O(1) | |
| getSubtitles() | O(1) | |
| getSubtitle(Time time) | O(log n) | |
| nbNodesInSearchPath(Time time) | O(log n) | |
| getSubtitles(Time startTime,Time endTime) | O(1) | |
| shift(int offset) | O(1) | |

## 4. Implementation

Method calcToMS (TimeSt st):

Requires: Tree not empty. input: TimeSt st.

Results: the method calculates hours by getting the hour of start time(st) times 3600000 , and minuets by getting the minuets of start time(st) times 60000 and seconds by getting the seconds of start time(st) times 1000 and milliseconds by getting milliseconds of start time(st).then returns time in milliseconds.

Output: time in milliseconds.

Method calc (int offset ):

Requires: Tree not empty. input: int offset.

Results: The method shifts the subtitle by comparing time in milliseconds with offset. If offset is greater that or equal milliseconds then it's shifted by addition. If offset is less than milliseconds then it will be shifted by subtraction
.

Output: none.

Method compareTo (TimeSt t ).

Requires: Tree not empty. input: TimeSt t.

Results: the method returns 1 if the milliseconds in this class greater than the millisecond of the received object , returns -1 if its less than the millisecond of the received object and returns zero otherwise.

Output:1, 0 or -1.

## 5. Conclusion

In conclusion after finishing this project we can manipulate subtitles for example: adding, removing, getting…etc, subtitles that synchronizes with a specific time. In the future we are able to create other applications to work with other formats.